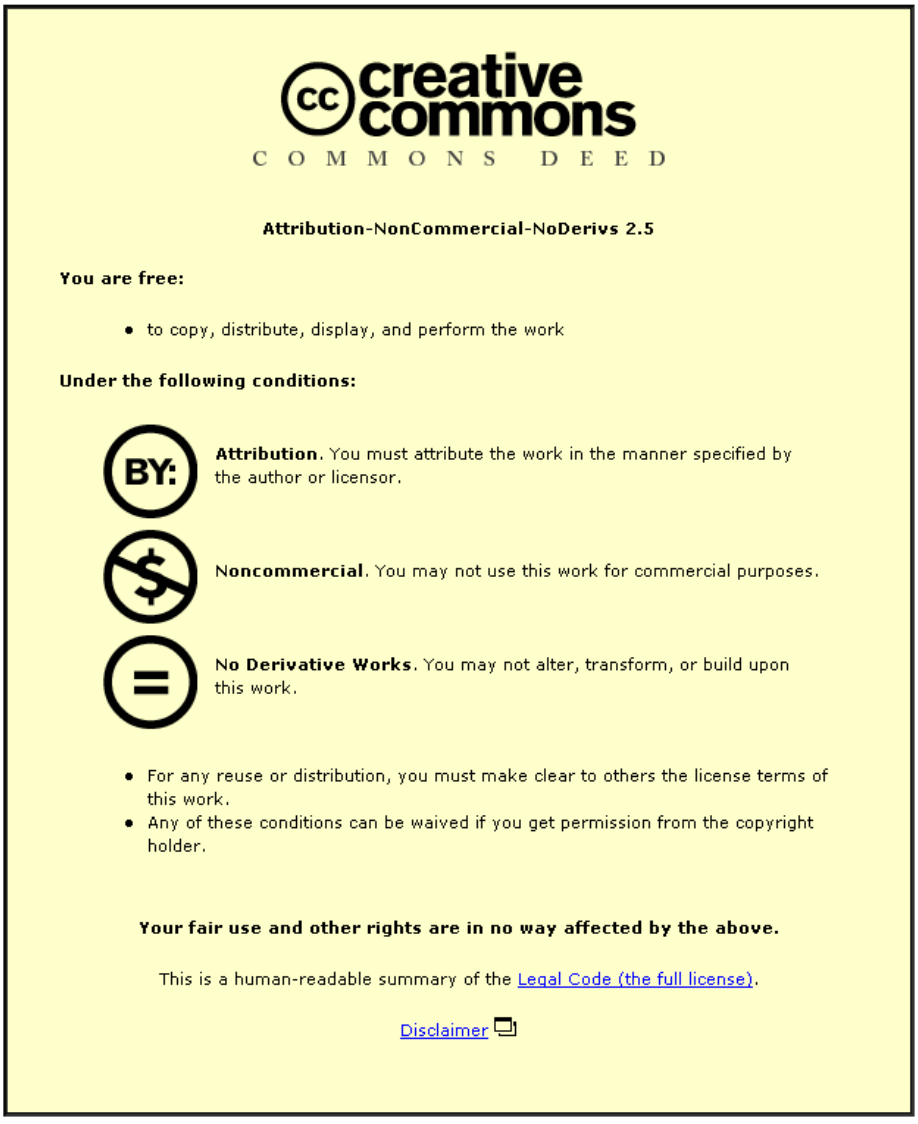


creative commons

COMMONS DEED

Attribution-NonCommercial-NoDerivs 2.5

You are free:

- to copy, distribute, display, and perform the work

Under the following conditions:

Attribution. You must attribute the work in the manner specified by the author or licensor.

Noncommercial. You may not use this work for commercial purposes.

No Derivative Works. You may not alter, transform, or build upon this work.

- For any reuse or distribution, you must make clear to others the license terms of this work.
- Any of these conditions can be waived if you get permission from the copyright holder.

Your fair use and other rights are in no way affected by the above.

This is a human-readable summary of the Legal Code (the full license).

Disclaimer

SPEA2-BASED SAFETY SYSTEM MULTI-OBJECTIVE OPTIMIZATION

By

Jelena Riauke

A Doctoral Thesis

submitted in partial fulfillment of the requirements for the award of

Doctor of Philosophy of Loughborough University

November, 2009

ABSTRACT

Safety systems are designed to prevent the occurrence of certain conditions and their future development into a hazardous situation. The consequence of the failure of a safety system of a potentially hazardous industrial system or process varies from minor inconvenience and cost to personal injury, significant economic loss and death. To minimise the likelihood of a hazardous situation, safety systems must be designed to maximise their availability. Therefore, the purpose of this thesis is to propose an effective safety system design optimization scheme. A multi-objective genetic algorithm has been adopted, where the criteria catered for includes unavailability, cost, spurious trip and maintenance down time.

Analyses of individual system designs are carried out using the latest advantages of the fault tree analysis technique and the binary decision diagram approach (BDD). The improved strength Pareto evolutionary approach (SPEA2) is chosen to perform the system optimization resulting in the final design specifications.

The practicality of the developed approach is demonstrated initially through application to a High Integrity Protection System (HIPS) and subsequently to test scalability using the more complex Firewater Deluge System (FDS). Computer code has been developed to carry out the analysis. The results for both systems are compared to those using a single objective optimization approach (GASSOP) and exhaustive search. The overall conclusions show a number of benefits of the SPEA2 based technique application to the safety system design optimization.

It is common for safety systems to feature dependency relationships between its components. To enable the use of the fault tree analysis technique and the BDD approach for such systems, the Markov method is incorporated into the optimization process. The main types of dependency which can exist between the safety system component failures are identified. The Markov model generation algorithms are suggested for each type of dependency. The modified optimization tool is tested on the HIPS and FDS. Results comparison shows the benefit of using the modified technique for safety system optimization. Finally the effectiveness and application to general safety systems is discussed.

Key Words: Optimization, Fault Tree Analysis, Binary Decision Diagrams, Genetic Algorithms, SPEA2, Markov Analysis.

DEDICATION

I dedicate this thesis in loving memory of my mother, Zinaida Borisevic, without whom none of this would have been ever possible.

ACKNOWLEDGEMENTS

I would like to thank, first and foremost, my supervisor, Dr Lisa Bartlett, for her generosity, support and critical guidance through my research and during the completion of this thesis.

My sincere gratitude is expressed to Professor John Andrews for his understanding and support through the most difficult time in my life.

I would like to acknowledge other members of Risk and Reliability research group and the staff members of the Department of Aeronautical and Automotive Engineering for their assistance, continued interest and encouragement.

Finally, I offer my sincere appreciation to my husband, Nerijus, and my friend, Akvilina, who constantly offered love, companionship and support during the research.

CONTENTS

NOMENCLATURE

LIST OF FIGURES

Chapter 2

Chapter 4

Chapter 5

Chapter 6

Chapter 7

Chapter 8

Chapter 9

LIST OF TABLES

Chapter 9

ABBREVIATIONS

AFFF	Aqueous Film-Forming Foam
BDD	Binary Decision Diagram
CSD	Concurrent Subspace Design
DMOEA	Dynamic Multi-Objective Evolutionary Algorithm
EP	Evolutionary Programming
ESD	Emergency Shut Down
FDS	Firewater Deluge System
FT	Fault Tree
GA	Genetic Algorithm
GASSOP	Genetic Algorithm Safety System Optimization Procedure
GDA	Great Deluge Algorithm
GP	Geometric Programming
HIPS	High Integrity Protection System
ISPEASSOP	Improved Strength Pareto Evolutionary Algorithm Safety System Optimization Procedure
LCC	Life Cycle Cost
LCD	Liquid Crystal Display
LP	Linear Programming
MDT	Maintenance Down Time
MFGP	Main Fire and Gas Panel
MOEA	Multi-Objective Evolutionary Algorithm
MOGA	Multi-Objective Genetic Algorithm
MTI	Maintenance Test Interval
NF	Neutrino Factory
NPGA	Niched Pareto Genetic Algorithm
NPP	Nuclear Power Plant
NSGA	Non-dominated Sorted Genetic Algorithm
NSGA-II	Elitist Non-dominated Sorting Genetic Algorithm
PAES	Pareto Archive Evolutionary Algorithm
PDE	Pareto-frontier Differential Evolution
PESA	Pareto Envelope-based Selection Algorithm
RPIS	Reactor Protection Instrumentation System

PSO	Particle Swarm Optimization
PT	Pressure Transmitter
RS	Random Search
SA	Simulated Annealing
SCMC	System Corrective Maintenance Cost
SGA	Simple Genetic Algorithm
SIC	System Initial Cost
SM	Safety Manager
SPEA	Strength Pareto Evolutionary Approach
SPEA2	Improved Strength Pareto Evolutionary Algorithm
SPMC	System Preventative Maintenance Cost
STC	System Testing Cost
STCA	Short-Term Conflict Alert system
TA	Threshold Accepting
TFT	Thin Film Transistor
TMEC	Total Maintenance Effort Cost
TS	Tabu Search

SYMBOLS

β	Parameter for Weibull distribution
C_i	Minimal cut set *i*
C_p	Penalty function for excess cost
CM_i	Corrective maintenance cost of component *i*
$d(r,s)$	Average distance between all pairs of objects in cluster *r* and cluster *s*
$D(i)$	Density function
$D(x,y)$	Euclidean distance
η	Parameter for Weibull distribution
f_i	Fitness of string *i*
$f(a^{'})$	Pareto front
$f(X)$	Objective function

$F(i)$	Fitness function
F_{sys}	System spurious trip frequency
$G_i(q)$	Birnbaum's measure of importance
$g_j(X)$	Inequality constraints
$l_j(X)$	Equality constraints
LCC_P	Penalty due to excess life cycle cost
λ	Failure rate
M_p	MDT penalty function
p_i	Selection function
$pr_{x_i}(q)$	Probability of the path section from the root node to the node x_i
$po^1_{x_i}(q)$	Probability of the path section from the 1 branch of the node x_i
$po^0_{x_i}(q)$	Probability of the path section from the 0 branch of the node x_i
P	Initial population
P'	External set of nondominated individuals
$\overline{P}_{t+1}$	Archive for $t+1$ generation
PMC_i	Preventative maintenance cost of component i
Q_{sys}	System unavailability
Q'_{sys}	Penalised system unavailability
$Q_i(t)$	Probability of system residing in state i at time t
$Q_M(t)$	Module failure probability
$Q(1_i, q)$	Probability of system failure with $q_i(t) = 1$
$Q(0_i, q)$	Probability of system failure with $q_i(t) = 0$
$R(i)$	Raw fitness
s_i	Strength of a Pareto solution i
$S(i)$	Individual strength value
S_p	Spurious trip penalty function
$SPMC_P$	Penalty due to excess system preventative maintenance cost
STC_P	Penalty due to excess system testing cost
τ	Mean time to repair

θ	Inspection interval
T	Top event
$TMEC_P$	Penalty due to excess system total maintenance effort cost
$w_i(t)$	Unconditional failure intensity of event *i*
$w_M(t)$	Module unconditional failure intensity
$w_{sys}(t)$	System unconditional failure intensity
x	Decision vector
X	Parameter space
y	Objective vector
Y	Objective space
$Z(q)$	Probability of the paths from the root node to the terminal 1 node

CHAPTER 1

INTRODUCTION

1.1 Introduction to Reliability and Risk Assessment

A safety system is an essential part of any industrial system as it operates to prevent the occurrence of certain conditions and their future development into a hazardous situation. Failure of such systems may have catastrophic consequences from small injuries to even death of members of the workforce and public. History has witnessed a number of disastrous accidents due to the failure of such systems, for example, the Chernobyl nuclear power plant accident in 1986, the explosion on the Piper Alpha oil platform in 1988 and the explosion in the Texas City BP Refinery in 2005. Therefore, the probability of the safety system failure should be quantified in order to manage its reliability.

During the last decades a number of techniques have been developed for the system failure assessment. These methods enable the evaluation of a specific hazardous event occurrence probability or frequency.

Reliability is an important part of the engineering design process. It is also a necessary function in the system life-cycle costing and repair and facility resourcing. Reliability determines the inventories and spare part requirements, establishes the preventative maintenance programs and influences the cost-benefit analysis.

1.2 Terminology

The key definitions used in reliability assessment terminology are [Andrews & Moss, 2002]:

- ***Failure*** *is the termination of the ability of a system or component to perform a required function.* It is assumed, that the failure of the system or component occurs when its ability to perform the required function is altered.

- ***Fault*** is *the inability of a system or component to perform its required function*. It should be noted, that the fault is always the result of a failure. However, it may not be a direct failure of the system or component itself.

- ***Availability*** *is the fraction of the total time that a system or component is able to perform its required function.* Regarding a specified time point, *t*, the availability can also be specified as the probability that a system or component is working at time *t*. On the other hand, the ***Unavailability*** *is the probability that that system or component is failed at time t*:

$$\text{Unavailability} = 1 - \text{Availability}. \tag{1.1}$$

- ***Reliability*** *is the probability that a system or component will operate without failure for a stated period of time under specified conditions.* Therefore, this characteristic requires identification of the time period needed for the system or component to function.

1.3 Safety System Design Problem

It is imperative that the safety system design has a high likelihood of functioning on demand in order to prevent possible hazards associated with industrial systems and minimize the consequences from the hazardous events.

The system design can be chosen by traditional approaches, which combine the preliminary design, analysis, appraisal and redesign stages until what is regarded as an acceptable design is achieved. For the system whose failure could result in fatality an adequate level for the system unavailability is not sufficient. Therefore, an optimal design within the constraints and available resources should be produced.

It is highly unlikely that the design parameters can be manually selected due to a large number of options. This task becomes even more difficult to complete when the system

design is constrained by available resources. Therefore, an optimization algorithm integrated within the design process is required.

1.4 Safety System Design Optimization Methodology

The safety system design optimization methodology needs to include a number of different techniques to achieve the tasks of the system evaluation and optimization. This section briefly introduces the main techniques suitable for such system design optimization.

1.4.1 Fault Tree Analysis

Fault trees are commonly used for the risk and reliability assessment of different industrial systems. It is a top-down deductive technique that graphically represents the relationship between certain specific events and the undesired event, which is often called a top event [Andrews & Moss, 2002]. The casual events in the tree are combined by Boolean logic. Various combinations of the events cause the top event to occur. A task to identify these combinations is performed by a qualitative analysis. A quantitative evaluation can then be performed by assigning failure probabilities to the basic events and computing the probability of the safety system top event.

1.4.2 Binary Decision Diagrams

The Binary Decision Diagram (BDD) is potentially the most successful approach for the safety system fault tree top event probability evaluation. A BDD is a directed acyclic graph [Andrews & Moss, 2002]. Each node of the graph corresponds to a basic event in the fault tree. A basic event ordering is essential for each BDD construction. A BDD performs both the qualitative and the quantitative analysis, and the exact solution can be obtained from a BDD without use of any approximations, which when applying the additional fault tree approach may be necessary.

1.4.3 Markov Analysis

The strong dependency relationships between components may exist in the majority of safety systems. In this case, the use of the fault tree analysis and the BDD method is no longer appropriate since both techniques are based on the assumption that component failures in the system are independent. The Markov method can be used to solve this problem.

The Markov analysis is based on a state-space approach [Andrews & Moss, 2002]. It looks at the system as being in one of several states. The state transition diagram, i.e. Markov model, identifies all the discrete states of the system and the possible transitions between those states. In a Markov process the transition frequencies between states depends only on the current state probability values and the constant transition rates between states. In this way the Markov model does not need to know about the history of how the state probabilities have evolved in time in order to calculate future state probabilities.

The major disadvantage of this method is that Markov models for large safety systems are generally exceedingly large, complicated and difficult to construct. However, systems that exhibit strong component dependencies for their isolated dependent sections may be analysed using a combination of Markov analysis and simpler quantitative models, for example, the BDD method for the remaining non-dependent parts.

1.4.4 Multi-Objective Genetic Algorithms

A number of mathematical optimization methods are available. However, the features offered by most methods make them inappropriate for safety system optimization; for example, classical optimization techniques require continuous and differentiable objective functions. During the last decade several heuristic techniques have evolved and facilitated the solution of optimization problems that were previously difficult or impossible to solve. Among all modern optimization methods the multi-objective genetic (evolutionary) algorithms (MOGAs) are the most popular for the solution of the

multi-objective problem [Everson and Fieldsend, 2006]. MOGAs popularity in recent years is explained by the following advantages: they are multi point search methods and are particularly suitable to derive the optimal set of solutions. Furthermore, they require little knowledge about the problem being solved. Another advantage is that MOGAs are easy to implement, robust, and inherently parallel. Due to their universality, these methods often take less time to find the optimal solution than other multi-objective approaches. Hence, this group of methods is suitable for safety system optimization problems.

1.5 Deficiencies in Optimization Techniques

The following possible deficiencies may occur in safety system optimization techniques:

- *The time to perform a system optimization*: most existing classical optimization techniques are time-consuming. Though the optimization can be carried out, their application is not ideal and can be improved.

- *Lack of flexibility*: the majority of real safety systems involve objective functions and constraints that are too complicated to manipulate with standard approaches (for example, classical and linear programming optimization techniques). Therefore, most existing optimization techniques can not be applied to such systems since they usually require constraints and an objective function to satisfy certain conditions and be in a particular form.

- *Optimization process complexity*: most optimization methods are not equipped to deal with the increasing complexity of safety systems, i.e. these techniques can take into account only a limited number of objectives and constraints. A technique is required to deal with multiple objectives (constraints) and to be applicable to a variety of complex systems.

- *The accuracy of the results produced*: The accuracy of the results is limited by some techniques potential to locate local minimum (maximum). A technique is required with a high likelihood of finding global minimum (maximum).

1.6 Research Objectives

The aim of this research is to improve the application of optimization techniques in the field of safety system design. The main task is to develop a multi-objective optimization tool in order to enable the optimal use of resources in safety system designs and, hence, ensure the best performance possible not just an adequate one. The specific research objectives are:

1) Develop the optimization scheme based on the chosen reliability and risk assessment method and multi-objective optimization technique.

2) Apply the developed optimization scheme to an example safety system:
   - Compare the results to those obtained by the simple genetic algorithm (GA) based optimization tool;
   - Improve the developed techniques efficiency.

3) Test the improved technique further by application to a more complex safety system and compare the results with those obtained by the simple GA.

4) Incorporate the Markov method into the developed optimization technique in order to allow the safety system component dependency modelling:
   - Test the modified optimization tool performance on the example safety systems;
   - Discuss its potential for any safety system application.

1.7 Thesis Structure

The thesis has the following structure:

- *Chapter two* discusses the main aspects of the fault tree analysis (FTA) and binary decision diagram (BDD) method and suggests the efficient way of their application to a safety system multi-objective optimization.

- *Chapter three* provides an introduction to engineering optimization and optimum design. It opens with a statement of an optimization problem and follows with an overview of the main classical and some modern groups of optimization methods. The chapter finishes with a discussion of the possible application of the mentioned methods to the safety system optimization problem.

- *Chapter four* describes the main features of the genetic algorithms (GAs), compares them to more traditional techniques and introduces the algorithm of the chosen method for safety system multi-objective optimization.

- *Chapter five* analyzes the structure of the example high-integrity protection system (HIPS), discusses its design options and reviews this systems previous optimization by single GA-based technique.

- *Chapter six* discusses the application of the developed optimization scheme to the HIPS. Three main parts of the program are identified and described in detail. The developed techniques performance is compared to the simple GA based approach. The chapter finishes with a discussion of possible modifications to the suggested optimization tool, which may improve its performance further.

- *Chapter seven* introduces and describes the main features of the more complex example firewater deluge system (FDS), provides this system analysis and reviews its optimization by the improved version of the developed technique. The comparative analysis of the results is represented at the end of the chapter, followed by the summary.

- *Chapter eight* introduces the modified optimization technique, which enables optimization of safety systems with dependencies by effective use of the Markov modelling tool. Different component dependency types are overviewed and the

modified technique application to the HIPS is illustrated in this chapter, followed by the discussion of the results and their potential improvement.

- *Chapter nine* describes the FDS optimization by the modified version of the developed optimization tool. The FDS dependency groups are discussed and the Markov model generation algorithms for all dependency types are suggested in this chapter. This is followed by the discussion of the results and the program potential application to any safety system.

- *Chapter ten* summarizes the research, provides the main conclusions and suggest potential areas for future work.

CHAPTER 2

FAULT TREE ANALYSIS

2.1 Fault Tree Analysis Overview

One of the main goals of reliability and safety analysis can be formulated as identifying the causal relationships between events, which results in system failure, and, furthermore, finding the ways to improve their impact by system redesigns and upgrades. Fault Tree Analysis was first conceived in 1961 by H. A. Watson of Bell Telephone Laboratories in connection with a US Air Force contract to study the Minuteman Missile Launch Control System [Watson, 1961]. It is one of the most powerful and widely used analytical techniques in the field of reliability engineering. It provides a well accepted means of predicting the reliability of complex systems. During the late 60's and 70's fault trees were successfully applied to several studies to obtain qualitative reliability information about relatively complex systems [Dhillon, 1978].

The deductive analysis starts with an enumeration of the potential hazards and works down through the system to identify the system hardware failures or human errors, which could have caused these. Fault tree analysis is a top-down technique, structured in terms of events rather than components. Moreover, the perspective is on faults rather than reliability. The fault tree has an established form: each fault tree has a *top event*, a root which represents a system failure mode. The top event is the first level of the tree. Each level is systematically deduced [Andrews and Moss, 2002]. Once the top event is defined *branches* are extended to intermediate events directly responsible for its occurrence. The lowest resolution of the tree are *basic events*. Hence, branches are terminated when basic events are encountered.

The fault tree is not a model representing all possible causes of system failure. Therefore, more than one fault tree may be constructed during the analysis of any potential system. Moreover, the fault tree acts as a visual tool, a graphical representation of the various parallel and sequential combinations of faults that lead to the occurrence of the top event. Fault tree construction is usually followed by qualitative and

quantitative analysis. Qualitative methods help to understand the logical structure of the various failure modes of a system and their relationship. On the other hand, quantitative methods assign failure probabilities or unavailabilities to the basic events and predict the probability of the top event. All these techniques are briefly discussed in this chapter.

2.2 Fault Tree Construction

The fault tree logic diagram is constructed from two basic elements: *gates* and *events*. Gates allow or inhibit the passage of fault logic up the tree and show the relationship between the events needed for the occurrence of a higher level event [Andrews and Moss, 2002]. Gates may have one or more input events, depending on the gate type, but only one output event. The most common gates are OR, AND and NOT. The main events are: basic, intermediate, house and transfer. An example of a simple fault tree is shown in figure 2.1.

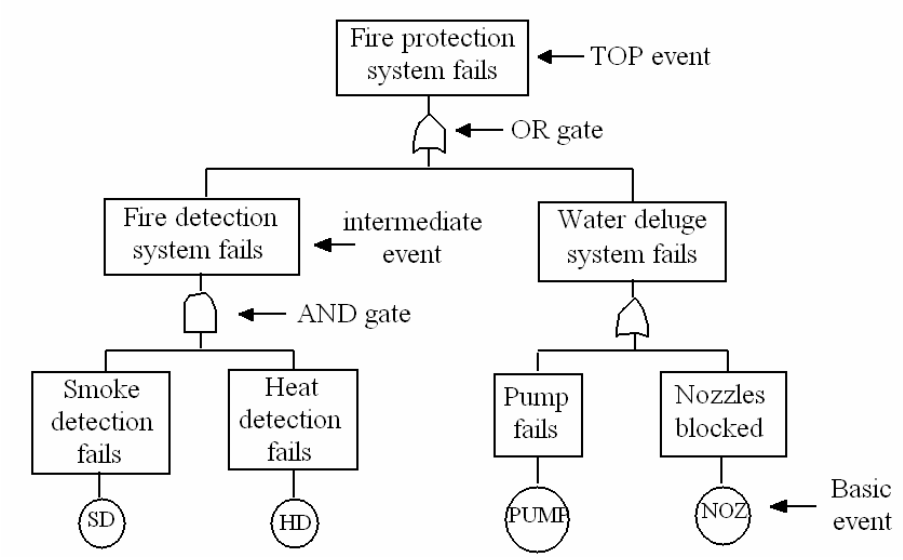


Figure 2.1 Fault Tree Example

In figure 2.1 the logic gate just below the top event 'Fire protection system fails' is an OR gate with inputs 'Fire detection system fails' and 'Water deluge system fails'. The top event will occur if at least one of the lower level (input) events occurs. It means that the fire protection system will fail if one or more of these systems were to fail. The

same logic is incorporated in the intermediate event 'Water deluge system fails'. In this case the intermediate event is terminated with basic events, 'Pump fails' and 'Nozzles blocked'. Another type of gate is just below the intermediate event 'Fire detection system fails'. This is the AND gate, which represents some redundancy in the system in that both of the basic events 'Smoke detection fails' and 'Heat detection fails' would need to have occurred to result in a total failure of the mentioned detection system to operate.

House events are used to model two state events which either occur or do not occur, and, therefore, have probabilities 1 or 0 [Andrews and Moss, 2002]. They provide a very effective means of turning sections of the fault tree on and off. One of the advantages of this is that the same fault tree can be used to model several scenarios. Figure 2.2 illustrates how house events can be used to define safety system design. In the example the safety system may consist of one or two pump sub-systems (i.e. A and B). Two house events are used to represent the design options. The presence of either sub-system can be determined by setting the appropriate house event value to TRUE (or ON). Otherwise, the house event value is set to FALSE (or OFF).

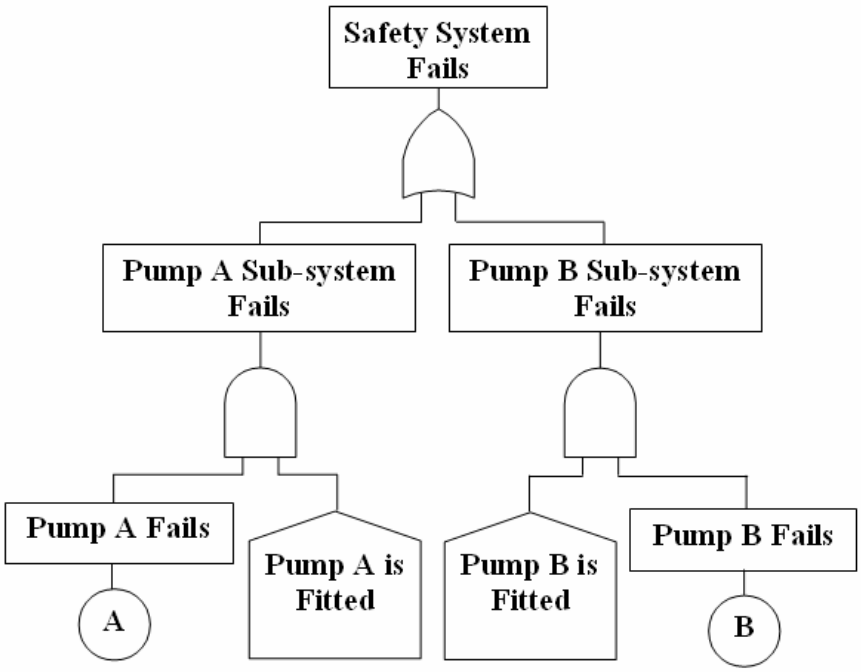


Figure 2.2 Example with House Events

2.3 Qualitative Fault Tree Analysis

2.3.1 Main Concepts

A fault tree represents the casual relationships resulting in an undesired system state. Many different combinations of events can cause the occurrence of the undesired state. Moreover, each unique combination causes a system failure mode. As a result, it may involve single or multiple component failures. The main function of qualitative analysis is to identify these causes of the system failure mode. These combinations can be clearly defined in the concept of a *cut set* [Andrews and Moss, 2002]. For most engineering systems there are generally a very large number of cut sets. However, the main interest for each analyst is to identify combinations of component failure modes which are both necessary and sufficient to produce system failure, i. e. to find *minimal cut sets.*

Definition 2.1. *Minimal cut set* is a smallest combination of component failures, which if they all occur will cause the top event to occur.

The minimal cut set expression for the top event *T* can be written in the form

$$T = C_1 + C_2 + ... + C_n, \tag{2.1}$$

where $C_i,\ i = 1,...,n$ are the minimal cut sets, and the '+' symbol represents OR logic.

Each minimal cut set consists of a combination of component failures and can be expressed as:

$$C_i = X_1.X_2...X_m, \tag{2.2}$$

where $X_i,\ i = 1,...,m$ are basic component failures on the tree and the '.' symbol represents AND logic.

2.3.2 Approaches to Obtain Minimal Cut Sets

Monte Carlo simulation techniques were the base for the first minimal cut set determination algorithms [Crosetti, 1970]. Later methods are deterministic, i.e. based on direct expansion of the top event in terms of the constituent basic events using Boolean algebra. For example, an OR gate with two input events A and B and the output event T can be represented by the equivalent Boolean expression

$$T = A \cup B \text{ or } T = A + B .$$

The AND logic for the similar gate structure in Boolean equivalent is

$$T = A \cap B \text{ or } T = A.B .$$

The NOT gate is equivalent to the Boolean operation of complementation.

The main laws of Boolean algebra used to manipulate the top event structure logic expression are:

1) **Commutative law**:

$$A + B = B + A ,\ A.B = B.A \tag{2.3}$$

2) **Associative law**:

$$(A + B) + C = A + (B + C) ,\ (A.B).C = A.(B.C) \tag{2.4}$$

3) **Distributive law**:

$$A + (B.C) = (A + B).(A + C) ,\ A.(B + C) = A.B + A.C \tag{2.5}$$

4) **Identities**:

$$A + 0 = A ,\ A + 1 = 1 ,\ A.0 = 0 ,\ A.1 = A \tag{2.6}$$

5) **Idempotent law**:

$$A + A = A\ ,\ A.A = A \tag{2.7}$$

6) **Absorption law**:

$$A + A.B = A\ ,\ A.(A + B) = A \tag{2.8}$$

One of the commonly used approaches to determine cut sets using the above expressions is a top down method.

Top Down Approach: The top event is the start point of this method. The top event is expanded by substituting in the Boolean events appearing lower down in the tree and simplifying until the expression remaining has only basic component failures.

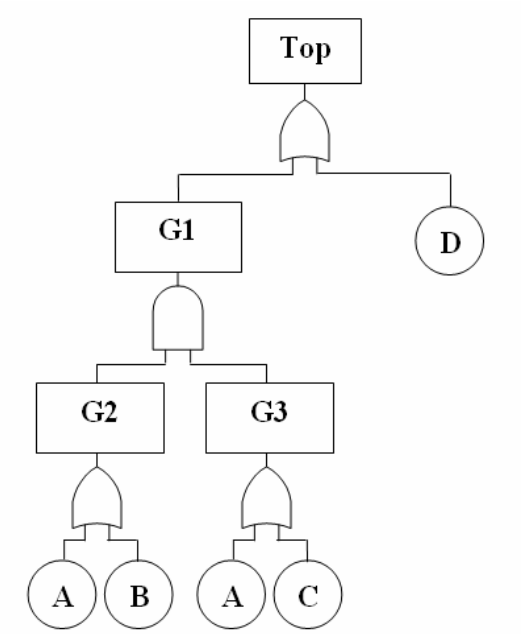


Figure 2.3 Example Fault Tree for Minimal Cut Sets Calculation

Considering the fault tree example from figure 2.3, this method works as follows:

- Start at Top event:

  Top = *G1* + *D*

- Substitute in for *G1*: *G1* = *G2.G3*

  Top = *G2.G3* + *D*

- Substitute in for *G2* and *G3*:

  $G2 = A + B$

  $G3 = A + C$

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

- By Distributive Law (equation 2.5):

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

Hence, there are 3 minimal cut sets *A*, *BC* and *D*.

2.4 Top Event Probability

The top event probability is an important quantification parameter of the fault tree. It can be calculated when the failure probabilities are assigned to each basic event and the minimal cut sets have been obtained. The top event probability (*unavailability*) usually is denoted as Q_{sys}. Depending on the fault tree structure (i.e. the inclusion of multiple basic events and, hence, dependence) the inclusion/exclusion formula can be implemented for the top event calculation.

2.4.1 Inclusion/Exclusion Formula

If a fault tree has *n* minimal cut sets C_i, $i = 1,...,n$ then the top event *Top* exists if at least one minimal cut set exists:

$$Top = C_1 + C_2 + ... + C_n = \bigcup_{i=1}^{n} C_i$$

and

$$P(Top) = P\left(\bigcup_{i=1}^{n} C_i\right).$$

Then the *Inclusion-exclusion expansion* is defined as

$$P(Top) = \sum_{i=1}^{n} P(C_i) - \sum_{i=2}^{n}\sum_{j=1}^{i-1} P(C_i \cap C_j) + ... + (-1)^{n-1} P(C_1 \cap C_2 \cap ... \cap C_n). \quad (2.9)$$

Consider the fault tree (Figure 2.4) with three basic events A, B and C. The basic event A appears twice. Each basic event probability of occurrence is $q = 0.1$.

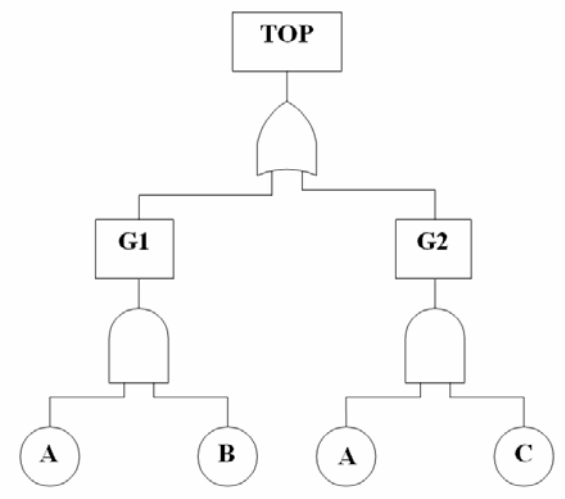


Figure 2.4 Example Fault Tree

From figure 2.4 the top event can be written as:

$$Top = AB + AC\,.$$

The inclusion-exclusion expansion (equation 2.9) gives:

$$\begin{aligned} P(Top) &= P(AB + AC) \\ &= P(AB) + P(AC) - P(AB \text{ AND } AC) \\ &= q_A q_B + q_A q_C - q_A q_B q_c \\ &= 0.01 + 0.01 - 0.001 \\ &= 0.019\,. \end{aligned}$$

When the number of minimal cut sets is small, the inclusion-exclusion expansion can be applied easily. However, calculating each higher term is a tedious and time consuming task. Most engineering systems have a large number of minimal cut sets, therefore, such calculations can be impractical even with modern computers. Hence, the use of approximations is required.

2.4.2 Approximations

Rare event approximation (*Upper bound*). This approximation is provided by truncating the series at the first odd numbered term in the inclusion-exclusion expansion:

$$Q(t) = \sum_{i=1}^{n} P(C_i). \tag{2.10}$$

Lower bound. This approximation is provided by truncating the series after the first even-numbered term in the inclusion-exclusion expansion:

$$Q(t) = \sum_{i=1}^{n} P(C_i) - \sum_{i=2}^{n} \sum_{j=1}^{i-1} P(C_i \cap C_j). \tag{2.11}$$

Minimal Cut Set Upper Bound. This approximation is a more accurate upper bound. The main principle can be formulated as: the system fails if at least one minimal cut set occurs, which is equal to the difference between 1 (whole probability) and the probability that no minimal cut set occurs (equation 2.12):

$$Q(t) \le 1 - \prod_{i=1}^{n} (1 - P(C_i)). \tag{2.12}$$

The relation between all mentioned approximations is given below:

$$\sum_{i=1}^{n} P(C_i) - \sum_{i=2}^{n} \sum_{j=1}^{i-1} P(C_i \cap C_j) \le Q_{sys} \le 1 - \prod_{i=1}^{n} (1 - P(C_i)) \le \sum_{i=1}^{n} P(C_i). \tag{2.13}$$

lower bound — **exact** — **minimal cut set upper bound** — **rare event approximation**

2.5 Unconditional System Failure Intensity

Another important quantification measure is the top event unconditional failure intensity ($w_{sys}(t)$), that is the probability per unit time that the system fails at time *t*. It can be calculated by the equation 2.14:

$$w_{sys}(t) = \sum_{i=1}^{n} G_i(q) \cdot w_i(t), \tag{2.14}$$

where $w_i(t)$ is the unconditional failure intensity of event *i* and $G_i(q)$ is the criticality function also known as Birnbaum's measure of importance. The criticality function is defined as the probability that the system is in the critical state with respect to component *i*. Failure of component *i*, therefore, transforms the system from the working to the failed state. The criticality function calculation requires determination of the probability that the system fails only if component *i* fails. This is evaluated by the difference between the probability that the system fails with component *i* failed and the probability that the system fails with component *i* working. Hence,

$$G_i(q) = Q(1_i, q) - Q(0_i, q), \tag{2.15}$$

where

$Q(1_i, q)$ is the probability of system failure with $q_i(t) = 1$,

$Q(0_i, q)$ is the probability of system failure with $q_i(t) = 0$.

2.6 Binary Decision Diagrams

The fault tree is a powerful visual tool which translates a physical system into a structured logic diagram. Therefore, this technique is widely used by engineers for safety system optimization (Los Angeles water system [Haimes, 1999], height control system for a new tunnel tube crossing river Elbe in Hamburg [Reif, 2003], railroad crossing system [Ortmeier et al, 2005]). However, this method has its limitations especially when the structure of the fault tree is very large. In such cases analysis of the top event probability usually requires the use of alternative methods, since the exact technique makes significant use of computer recourses. Among these the Binary Decision Diagram (BDD) approach is potentially the most successful [Andersen, 1998]. The conversion of the fault tree to the BDD format improves both the efficiency of determining the minimal cut sets of the fault tree and also the accuracy of the calculation procedure used to determine the top event parameters. The detailed BDD structure and construction methods are discussed in section 2.6.1.

2.6.1 BDD Structure and Construction

A BDD can be described as a rooted, directed acyclic graph (Figure 2.5). All paths through the BDD start at the root vertex (*A*) and terminate in one of the two states, either 1 or 0. State 1 corresponds to the system failure, state 0, conversely, corresponds to system success.

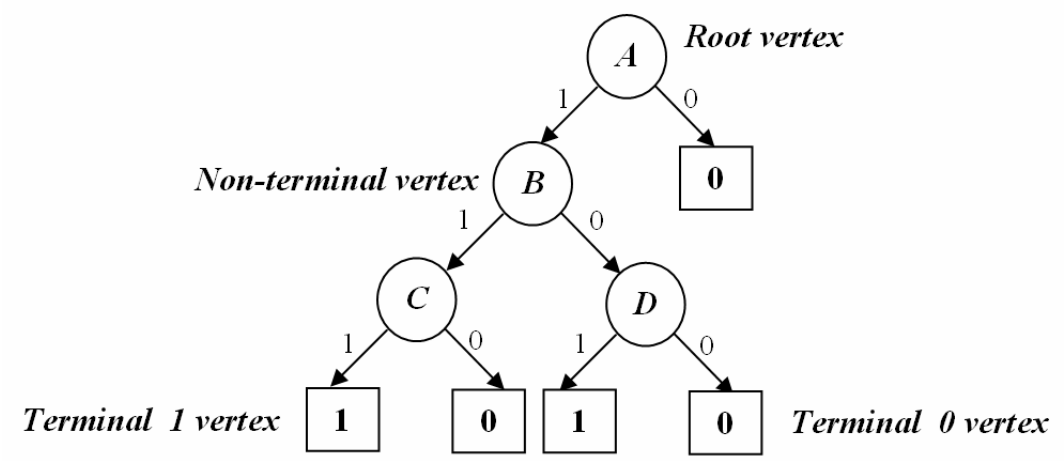


Figure 2.5 Example of the Binary Decision Diagram

Each BDD is composed of vertices, connected by branches, which are divided into terminal and non-terminal. Non-terminal vertices correspond to the basic events of the fault tree (vertices *B, C* and *D*). All the paths terminating in a 1 state give the cut sets of the fault tree. Therefore, the cut sets are {*A, B, C*} and {*A, D*}.

Usually the fault tree structure is converted into a BDD either by the top event logic function or by using an If-Then-Else method.

The If-Then-Else Method. This technique was developed by Rauzy [Rauzy, 1993]. The approach constructs the BDD from its equivalent fault tree by using a bottom up procedure. The variables can be assigned an ordering by use of the top-down, left-right approach where the basic events which are placed higher up the tree are listed first. Moreover, each basic event X is assigned an **ite** structure, $\mathbf{ite}(X, F_1, F_2)$. This can be

interpreted as if X fails then consider F_1, else consider F_2. The following procedures are then implemented to compute the BDD.

Consider $X = \mathbf{ite}(x, F_1, F_2)$ and $Y = \mathbf{ite}(y, G_1, G_2)$, and <op> is a Boolean operation of the logic gates in the fault tree. If the gate type is *AND* <op> is replaced by the '.' symbol, in the case of the *OR* gate <op> is replaced by the '+' symbol. Then the main rules are influenced by the ordering of the variables, i.e. x before y ($x < y$) or x equal to y ($x = y$):

1) If $x < y$: X<op>Y = **ite**$(x, F_1 < op > Y, F_2 < op > Y)$,

2) If $x = y$: X <op>Y = **ite**$(x, F_1 < op > G_1, F_2 < op > G_2)$,

The following identities are used in addition to simplify the **ite** structure at each stage:

1) If <op> is an *OR* gate:

$1 + X = 1$,

$0 + X = X$.

2) If <op> is an *AND* gate:

1. $X = X$,

0. $X = 0$.

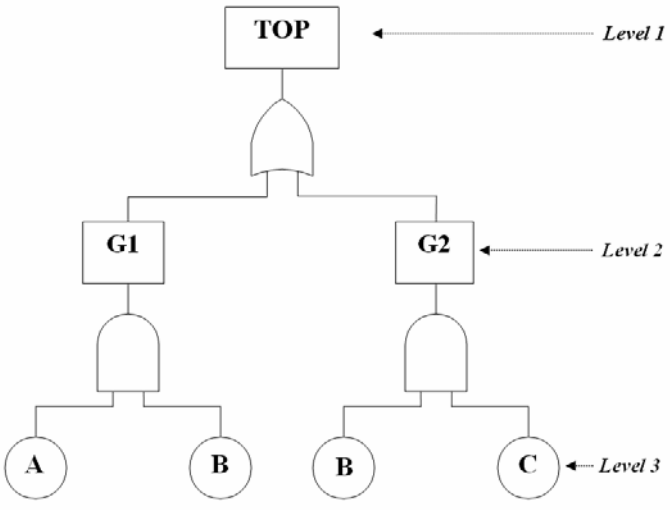


Figure 2.6 Example Fault Tree

Consider the example fault tree (Figure 2.6). According to the top-down, left-right approach, the variables are ordered as $A < B < C$., i. e. A is considered first, then B and finally C. Then all three basic events are given relevant **ite** structures, i.e. A = **ite**(A, 1, 0), B = **ite**(B, 1, 0), C = **ite**(C, 1, 0). Working from the bottom of the top and applying the simplification identities as explained the results for the intermediate gates are:

$$
\begin{aligned}
G1 &= A.B \\
&= \textbf{ite}(A, 1, 0).\ \textbf{ite}(B, 1, 0) \\
&= \textbf{ite}(A, 1.\ \textbf{ite}(B, 1, 0), 0.\ \textbf{ite}(B, 1, 0)) \\
&= \textbf{ite}(A, \textbf{ite}(B, 1, 0), 0);
\end{aligned}
$$

$$
\begin{aligned}
G2 &= B.C \\
&= \textbf{ite}(B, 1, 0).\ \textbf{ite}(C, 1, 0) \\
&= \textbf{ite}(B, 1.\ \textbf{ite}(C, 1, 0), 0.\ \textbf{ite}(C, 1, 0)) \\
&= \textbf{ite}(B, \textbf{ite}(C, 1, 0), 0).
\end{aligned}
$$

The top gate is evaluated as:

$$
\begin{aligned}
Top &= G1 + G2 \\
&= \textbf{ite}(A, \textbf{ite}(B, 1, 0), 0) + \textbf{ite}(B, \textbf{ite}(C, 1, 0), 0) \\
&= \textbf{ite}(A, \textbf{ite}(B, 1, 0) + \textbf{ite}(B, \textbf{ite}(C, 1, 0), 0), 0 + \textbf{ite}(B, \textbf{ite}(C, 1, 0), 0)) \\
&= \textbf{ite}(A, \textbf{ite}(B, 1 + \textbf{ite}(C, 1, 0)\ , 0 + 0), \textbf{ite}(B, \textbf{ite}(C, 1, 0), 0)) \\
&= \textbf{ite}(A, \textbf{ite}(B, 1, 0), \textbf{ite}(B, \textbf{ite}(C, 1, 0), 0)).
\end{aligned}
$$

Working from the left to right, each variable is successively broken down into its left and right branches. Hence, A is the root variable. The failure branch (left branch) outcome is B, which, breaks into 0 on its right branch (meaning system success) and 1 on its left branch, meaning system failure or the top event occurrence. Considering the right branch, B breaks down into 0 on its right branch and C on the left, where C represents the terminal **ite** structure, i.e. C = **ite**(C, 1, 0). Figure 2.7 shows the resulting BDD structure for the example fault tree in general and simplified forms. The highlighted branches of the simplified BDD give the cut sets $\{A, B\}$ and $\{B, C\}$, which are minimal.

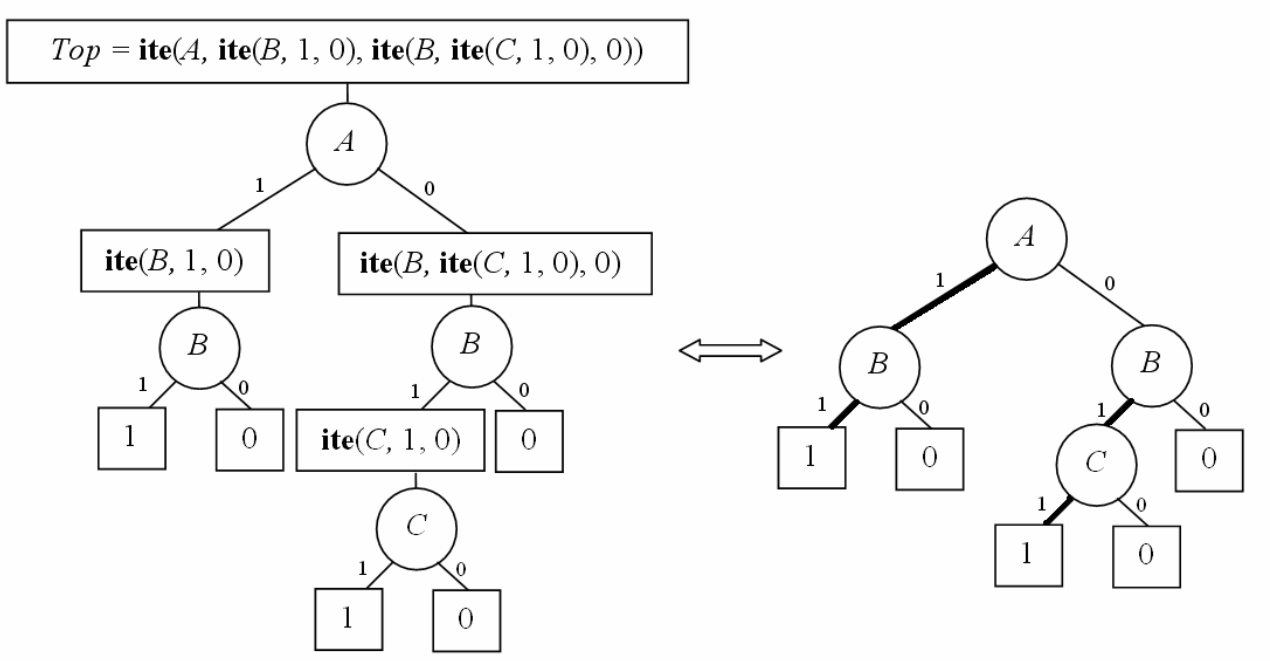


Figure 2.7 BDD Structure for the Example Fault Tree

2.6.2 BDD Minimization

The BDD does not always give minimal cut sets. Rauzy [Rauzy, 1993] developed a minimising algorithm that generates a BDD defining exactly the minimal cut sets of the fault tree. The algorithm states that:

a) If the output of the node is represented by the function F :

$$F = \textbf{ite}(x,\ G,\ H);$$

b) and δ is a minimal solution of G and together is not a minimal solution of H; then a minimal solution of F is given by the intersection of δ and x, i.e.

$$F_{\min} = \{\delta\} \cap x\ .$$

The algorithm completes with the conclusion, that the set of all minimal solutions of F ($sol_{\min}(F)$) includes both these minimal solutions of G which are not contained within H, (δ), and the minimal solutions of H. Hence,

$$sol_{\min}F = [\{\delta\} \cap x] \cup [sol_{\min}(H)]\ .$$

2.6.3 Top Event Quantification using BDD

System Failure Probability. Top event quantification using a BDD avoids the need to use approximations and obtains an exact probability of the top event directly from the diagram. If the top event **ite** structure is $f(x)$ = **ite**(x_i, f_1, f_2), then the corresponding Boolean function is $f(x) = x_i f_1 + \overline{x}_i f_2$, where $P(\overline{x}_i) = 1 - P(x_i) = 1 - q_i$. In this case the probability of the top event is obtained by taking the expectation of each term (equation 2.16):

$$Q_{sys} = E[f(x)] = q_i E[f_1] + (1 - q_i) E[f_2], \tag{2.16}$$

where $q_i = E[x_i]$ the probability that event i has occurred. Each path to a terminal 1 vertex is, therefore, mutually exclusive or disjoint. The probability of occurrence of the top event is calculated by the sum of the probabilities of the disjoint paths through the BDD [Shi and Lee, 1997]. The probability of each disjoint path is a product of the probabilities of included basic events encountered on route from the root node to the terminal vertex. Here the 0 branch infers the use of the component success probabilities and the 1 branch component failure.

To illustrate, consider the example BDD from figure 2.8.

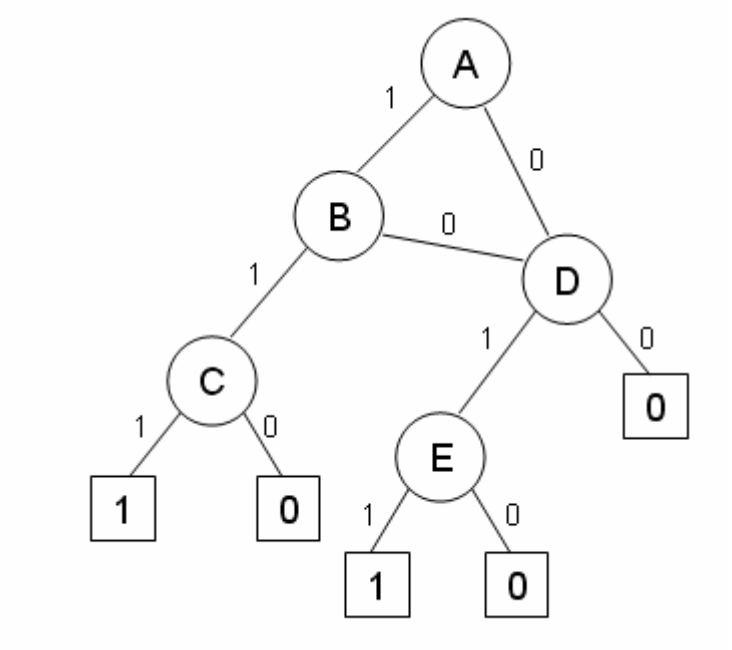


Figure 2.8 Example BDD

The disjoint paths of the BDD shown in figure 2.8 are:

1) $A.B.C,$
2) $\overline{A}.D.E,$
3) $A.\overline{B}.D.E.$

Therefore, the system unavailability (Q_{sys}) for the example fault tree is the sum of the product of each disjoint path, i.e.:

$$Q_{sys} = P\left(A.B.C + \overline{A}.D.E + A.\overline{B}.D.E\right)$$
$$= q_A q_B q_C + (1-q_A)q_D q_E + q_A(1-q_B)q_D q_E .$$

System Failure Intensity. The system failure intensity is calculated by equation 2.14. The main part of this equation is the evaluation of the criticality function $G_i(q)$ (Equations 2.15) depending on two terms, $Q(1_i, q)$ and $Q(0_i, q)$. These two terms can be calculated using the BDD approach, which only requires one pass of the BDD structure for each component. The fault tree approach, in contrast, requires two passes. The formulae used when applying the BDD approach are:

$$Q(1_i, q) = \sum_{i=1}^{n}\left(pr_{x_i}(q)\cdot po^{1}{}_{x_i}(q)\right) + Z(q), \tag{2.17}$$

$$Q(0_i, q) = \sum_{i=1}^{n}\left(pr_{x_i}(q)\cdot po^{0}{}_{x_i}(q)\right) + Z(q), \tag{2.18}$$

where

$pr_{x_i}(q)$ is the probability of the path section from the root node to the node x_i,

$po^{1}_{x_i}(q)$ is the probability of the path section from the 1 branch of the node x_i to a terminal 1 node (excluding probability of x_i),

$po^{0}_{x_i}(q)$ is the probability of the path section from the 0 branch of the node x_i to a terminal 1 node (excluding probability of x_i),

$Z(q)$ is the probability of the paths from the root node to the terminal 1 node not passing through the node for variable x_i.

By substituting equations 2.17 and 2.18 in 2.15, the critically function can be rewritten as:

$$G_i(q) = \sum_{i=1}^{n} pr_{x_i}(q) \cdot \left[po^1_{x_i}(q) - po^0_{x_i}(q) \right] . \tag{2.19}$$

For better understanding of the system failure intensity evaluation consider an example BDD (Figure 2.10). The mathematical solution for this BDD is represented by tables 2.3-2.7.

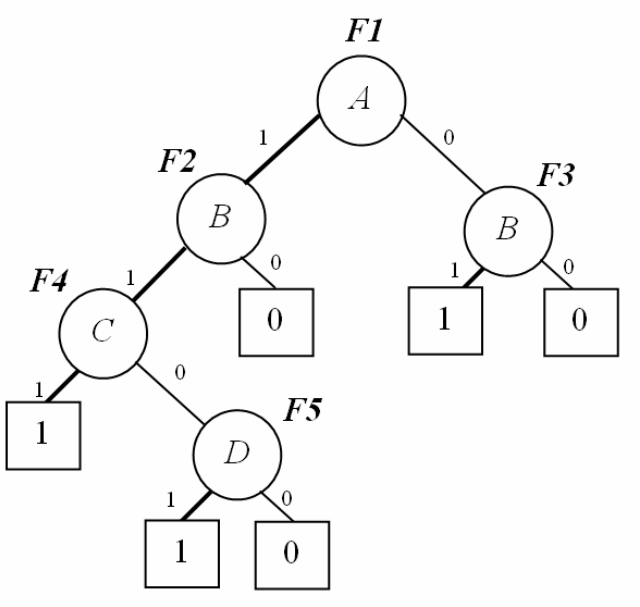


Figure 2.9 Example BDD for System Failure Intensity Evaluation

Table 2.1 shows the first step of the evaluation approach, i.e. the identification of the connections between the nodes (**ite** table) in the example BDD.

Table 2.1 Connections between the Nodes in Example BDD

Node	Variable	1 branch pointer	0 branch pointer
F1	*A*	*B*	*B*
F2	*B*	*C*	0
F3	*B*	1	0
F4	*C*	1	*D*
F5	*D*	1	0

Next steps of the evaluation process involve the calculation of all terms from equation 2.19. Table 2.2 represents the calculation process of the probability of the root vertex to node x_i, $pr_{x_i}(q)$. This probability is obtained by summing the probabilities along the relevant path, which is summarized in the "Comments" column.

Table 2.2 Calculation of $pr_{x_i}(q)$

Node	$pr_{x_i}(q)$	Comments
F1	1	Root vertex itself
F2	q_A	Probability of going along 1 branch of node *F1*
F3	$1-q_A$	Probability of going along 0 branch of node *F1*
F4	$q_A q_B$	Probability of path: 1 branch of *F1* and 1 branch of *F2*
F5	$q_A q_B (1-q_C)$	Probability of path: 1 branch of *F1*, 1 branch of *F2* and 0 branch of *F4*

Table 2.3 summarizes the calculation steps for the probability $po^1_{x_i}(q)$, the path from the selected node along the 1 branch to any terminal 1 vertex, excluding the probability of the selected node. The "Calculation" column shows the intermediate calculation of the full probability (where the selected node is included). Every selected route is explained in the column "Comments".

Table 2.3 Calculation of $po^1_{x_i}(q)$

Node	Calculation	$po^1_{x_i}(q)$	Comments
F1	$q_A q_B q_C + q_A q_B (1-q_C) q_D$	$q_B q_C + q_B (1-q_C) q_D$	**2 routes**: branch 1 of *F1*, 1 of *F2*, 1 of *F4* **or** branch 1 of *F1*, 1 of *F2*, 0 of *F4* and 1 of *F5*
F2	$q_B q_C + q_B (1-q_C) q_D$	$q_C + (1-q_C) q_D$	**2 routes**: branch 1 of *F2*, 1 of *F4* **or** branch 1 of *F2*, 0 of *F4* and 1 of *F5*
F3	q_B	1	1 branch
F4	q_C	1	1 branch
F5	q_D	1	1 branch

The structure of table 2.4 is similar to the structure of table 2.3. It summarizes the calculation steps for the probability of $po_{x_i}^0(q)$, the path from the selected node along the 0 branch to any terminal 1 vertex, excluding the probability of the selected node.

Table 2.4 Calculation of $po_{x_i}^0(q)$

Node	Calculations	$po_{x_i}^0(q)$	Comments
F1	$(1-q_A)q_B$	q_B	0 branch of *F1* and 1 of *F3*
F2	0	0	0 branch
F3	0	0	0 branch
F4	$(1-q_C)q_D$	q_D	0 branch of *F4* and 1 of *F5*
F5	0	0	0 branch

Ultimately, the criticality function $G_i(q)$ can be calculated for each variable, by adding together contributions from any nodes of the same variable (Table 2.5). For example, nodes *F2* and *F3* refer to the same variable *B*. Hence, the value of the criticality function for the variable *B* is obtained by summing the relevant terms of each node (Equation 2.19).

Table 2.5 Calculation of the Criticality Function, $G_i(q)$

Variable (Nodes)	Calculations	$G_i(q)$
A (*F1*)	$1\cdot\left[q_Bq_C+q_B(1-q_C)q_D-q_B\right]$	$G_A(q)=q_Bq_C+q_B(1-q_C)q_D-q_B$
B (*F2* and *F3*)	$q_A(q_C+(1-q_C)-0)+(1-q_A)(1-0)$	$G_B(q)=q_A(q_C+(1-q_C))+(1-q_A)$
C (*F4*)	$q_Aq_B(1-q_D)$	$G_C(q)=q_Aq_B(1-q_D)$
D (*F5*)	$q_Aq_B(1-q_C)(1-0)$	$G_D(q)=q_Aq_B(1-q_C)$

The system unconditional failure intensity $w_{sys}(t)$ is calculated by substituting the obtained value of $G_i(q)$ into equation 2.14.

2.7 Summary

The fault tree is a powerful tool suitable to represent the system failure and can be used to represent safety system design. The house events, incorporated in the fault tree structure, enable capability for multiple design options. The Binary Decision Diagram approach is an efficient technique for fault tree analysis. The process requires the fault tree to be converted into an alternative structure known as a Binary Decision Diagram. The main advantage of this method is that it produces the minimal cut sets of the fault tree more efficiently and can produce exact quantification measures. However, the conversion process from the fault tree to the BDD requires the basic events to be ordered. In some cases this ordering can cause complications.

CHAPTER 3

OPTIMIZATION TECHNIQUES

3.1 Introduction

There is no doubt today about the importance of optimization problems in various industries (aerodynamics, statistics, economics, engineering, etc.). The optimization process can be generally defined as getting the best of something under given conditions [Pun, 1969]. However, sometimes what is 'best' for one person is 'worst' for another, more often it is difficult to define the meaning of 'best'. The word *optimum* meaning 'best', is synonymous with 'most' or 'maximum' in the former case, and with 'least' or 'minimum' in the latter. The technical verb *optimize*, a stronger word than 'improve', means to achieve the optimum, and *optimization* refers to the act of optimizing [Wilde and Beightler, 1967]. Therefore, deciding how to design, build, regulate, or operate a physical, economic or safety system ideally encompasses the quantitative study of optima, methods for finding them, and involves three steps:

- Identifying accurately and quantitatively, how the system variables interact;
- Investigation of a single measure of system effectiveness expressible in terms of the system variables;
- Choosing the values of the system variables yielding optimum effectiveness.

Thus optimization and choice are closely related. Each optimization process requires the selection of a mathematical method to solve the optimization problem. Such methods are usually called *optimization techniques (mathematical programming techniques)*, or *algorithms*. Many methods of optimization theory have been known for centuries; however, the tedious and voluminous computations required prevented their practical application. The choice of optimization techniques has developed rapidly since the advent of the electronic computer in 1945. Perhaps the most outstanding example of the rapid development of optimization techniques occurred with the introduction of dynamic programming by Bellman in 1957 [Bellman, 2003] and of the maximum principle by Pontryagin in 1958 [Pontryagin, 1990].

This chapter provides an introduction to engineering optimization and optimum design. The aim is to propose a method or methods appropriate for effective and efficient application to the safety system optimization problem under consideration. The chapter opens with a statement of an optimization problem and follows with an overview of the main classical groups of optimization methods; and finishes with a section of some modern optimization techniques. At the end of the chapter the discussion of the possible application of the mentioned methods to the safety system optimization problem is provided.

3.2 The Optimization Problem and Objective Function

An optimization or a mathematical programming problem can be stated as follows [Rao, 1996]:

$$\text{Find } X = \begin{Bmatrix} x_1 \\ x_2 \\ \text{M} \\ x_n \end{Bmatrix} \text{ which minimizes (maximizes) } f(X)$$

Subject to the constraints

$$g_j(X) \le 0, \quad j = 1,2,...,m,$$

$$l_j(X) = 0, \quad j = 1,2,..., p, \tag{3.1}$$

Where X is an n-dimensional vector called the *design vector*, $f(X)$ is termed the *objective function,* and $g_j(X)$ and $l_j(X)$ are known as *inequality* and *equality* constraints, respectively. The number of variables n and the number of constraints m and (or) p need not be related in any way. The problem stated in equation 3.1 is called a *constrained optimization problem.*

Some optimization problems do not involve any constraints and can be stated as:

$$\text{Find } X = \begin{Bmatrix} x_1 \\ x_2 \\ \text{M} \\ x_n \end{Bmatrix} \text{ which minimizes (maximizes) } f(X). \qquad (3.2)$$

Such problems are called *unconstrained optimization problems.*

In general, for each safety system there is more than one acceptable design, and the purpose of optimization is to choose the best one from the many acceptable designs available. The criterion with respect to which the design is optimized, when expressed as a function of the design variables, is known as an *objective function*. The choice of objective function is governed by the nature of the problem. In some situations, there may be more than one criterion to be satisfied simultaneously. An optimization problem involving multiple objective functions is known as a *multi-objective programming problem*.

Each optimization problem requires the search for a minimum or maximum value of the function. However, minimums and maximums have a classification. An objective function of one variable $f(x)$ may have a *local* (*relative*) *minimum*, a *local* (*relative*) *maximum*, a *global* (*absolute*) *minimum* and a *global* (*absolute*) *maximum* at the point $x = x^*$.

3.3 Classical Groups of Optimization Methods

The classical groups of optimization methods involve *classical optimization techniques*, *linear, nonlinear, geometric, dynamic, integer* and *stochastic programming*.

Classical Optimization Techniques: The classical methods of optimization are useful in finding the optimum solution of continuous and differentiable functions. These methods are analytical and make use of the techniques of differential calculations in locating the optimum points [Rao, 1996]. The classical optimization problem is stated in equation 3.1. It is assumed that the constraints are independent; however, often it is not convenient from a practical point of view. The main reason for this is that the constraint

equations are nonlinear for most practical problems. Some practical problems involve objective functions that are not continuous and (or) differentiable. This also makes most classical techniques difficult to apply.

Linear Programming (LP): This is an optimization method applicable for the solution of problems in which the objective function and the constraints appear as linear functions of the decision variables. The LP type of optimization problem was first recognized in the 1930s by economists while developing methods for the optimal allocation of resources [Rao, 1996]. Nowadays LP is considered a revolutionary development that helps to make optimal decisions in complex situations. At least four Nobel Prizes were awarded for contributions related to LP. The main disadvantage of LP techniques is that they are very time-consuming for large scale optimization problems.

Nonlinear Programming: If the optimization problem involves an objective function and /or constraints that are too complicated to manipulate, it can not be solved by using the classical analytical methods. In this case *nonlinear programming* methods are often used. These techniques are often called *numerical methods* and are divided into the following groups: *elimination, interpolation, direct search, indirect search (descent), direct and indirect methods.* Most of these techniques can be applied only under certain conditions, which is the main disadvantage for practical applications.

Geometric Programming (GP): GP was developed by Duffin, Peterson and Zener [Duffin et al., 1967]. This method solves a class of nonlinear programming problems. This method differs from other optimization techniques. The main difference is that instead of finding optimal values of the design variables first, GP finds the optimal value of the objective function. Hence, this method is especially suitable for the problems where the optimal value of the objective function is the main subject of interest. Another important advantage of GP is that it often reduces a complicated optimization problem to one involving a set of simultaneous linear algebraic equations. However, the major disadvantage of this method is that it requires the objective function and the constraints in the form of *posynomials* (i.e. the sums of several components, each of those can be expressed as a power function of positive variables and constraints), which is hard to achieve for the majority of practical problems. A more

detailed explanation of the geometrical programming can be found in the reference [Rao, 1996].

Dynamic Programming: The dynamic programming technique represents or decomposes a multistage decision problem as a sequence of single-stage decision problems. Therefore, an *N*-variable problem can be represented as a sequence of *N* single-variable problems, which in most cases are easier to solve then the original problem. The decomposition to *N* single-variable problems is done in such a manner that the optimal solution of the original *N*-variable problem can be obtained from the optimal solutions of the *N* one-dimensional problems. The main disadvantage of the dynamic programming technique is a major drawback, known as *the curse of dimensionality* (i.e. exponential increase in volume) [Rao, 1996]. However, this technique is suitable for the solution of a wide range of complex problems in several areas of decision making.

Integer Programming: In all optimization techniques so far considered the design variables are assumed to be continuous, which can take a real value. However, in many engineering systems, certain design variables can take only discrete values. Also there are practical problems in which the fractional values of the design variables are neither practical nor physically meaningful. These difficulties can be avoided by using an integer programming techniques for the problem solution. Very little work has been done in the field of integer nonlinear programming. There are two popular methods of solution: the generalized penalty function method and the sequential linear integer (discrete) programming method. The detailed description of these techniques can be found in reference [Rao, 1996].

Stochastic Programming: This group of techniques is applied in situations where some or all of the parameters of the optimization problem are described by stochastic (random) variables rather that deterministic quantities. The sources of random variables may be several. They depend on the nature and type of the problem. For example, in the design of aircraft and rockets the actual load acting on the vehicle depends on the atmosphere conditions, which cannot be predicted precisely in advance [Rao, 1996]. Therefore, the loads should be treated as random variables in the design of such flight vehicles.

The basic idea of stochastic programming is to convert the stochastic problem into a equivalent deterministic problem, which is, therefore, solved by using familiar techniques, such as linear, nonlinear, geometric and dynamic programming.

3.4 Modern Heuristic Optimization Techniques

During the last decade several heuristic techniques have evolved and facilitated the solution of optimization problems that were previously difficult or impossible to solve. The most popular of these tools are briefly discussed in this section. They are: *genetic algorithms, evolutionary programming, random search, simulated annealing, great deluge, threshold accepting, tabu search* and the *particle swarm optimization method.*

Genetic Algorithms (GAs): GAs, developed initially by Holland in the 1970s [Holland, 1975], are search algorithms based on the principles of natural selection and genetics. A GA technique starts with a set of feasible solutions (population). Each solution corresponds to a chromosome. Solutions are selected from the population either randomly or according to their fitness (objective function value) and are combined to form new solutions (offspring). This process repeats until a stop criterion is satisfied, for example a maximum number of generations is reached. There are two basic genetic operators: crossover and mutation. A crossover operator denotes the place where two parents are split, then re-combined to form offspring, allowing beneficial genes on two different parents to be combined and, therefore, to produce in theory better solutions. Mutation is generally applied in a random manner. It occasionally introduces beneficial material into chromosomes [Davis, 1991].

Evolutionary Programming (EP): EP, originally conceived by Lawrence in 1960 [Lawrence, 1996], is a stochastic optimization strategy similar to GAs. There are three important ways in which EP differs from GAs. First, there is no constraint on the representation. Second, the mutation operator simply changes aspects of the solution according to the statistical distribution, and, the severity of mutations is often reduced as the global optimum is approached. Third, EP typically does not use any crossover as a genetic operator.

Random Search (RS): RS is one of the Monte Carlo methods. It serves as a baseline method of scheduling [Bettinger et al, 2002]. The method works through the following steps: random assignment of the initial solution and then fitness evaluation. If the stopping criterion is met, the best solution found during the search should be reported, otherwise the process repeats from the random assignment step. This method is one of the simplest of the Monte Carlo methods and is often less than adequate. However, RS can be applied with other optimization techniques to generate the initial data for very complicated problems.

Simulated Annealing (SA): SA is a search technique that began to be used in a widespread manner in the early 1980s [Bettinger et al., 2002]. The algorithm is based on a simulation of the cooling of materials in a heat bath. This process is known as annealing. Similar to RS, SA is a Monte Carlo approach that uses a local search in which a subset of solutions is explored by moving from one solution to a neighbouring solution. SA's major advantage over other methods is an ability to avoid becoming trapped at local minima. This optimization technique is very popular and could be applied in a wide variety of disciplines. However, the major difficulty in the implementation is that there is no obvious analogy for the temperature T (major component of the technique) with respect to a free parameter in the optimization problem. Furthermore, avoidance of capture in local minima is dependent on the "annealing schedule", the choice of initial temperature, how many iterations are performed at each temperature, and how much the temperature is decremented at each step as cooling proceeds.

Great Deluge Algorithm (GDA): GDA is a recently developed variant on simulated annealing. It was introduced by Dueck in 1993 [Dueck, 1993]. The algorithm is similar to SA in that only a single change is considered to a "current" solution. The resulting temporary solution is evaluated, and a decision is made whether or not to convert the temporary solution to the current solution. The form of the GDA consists of using a single parameter in the determination of whether or not to convert the temporary solution to the current one. The use of one parameter rather than two is advantageous. The GDA algorithm is believed to de-sensitize the SA algorithm thus leading to equally good results even when parameter estimation and formulation is poor.

Threshold Accepting (TA): TA is similar to both simulated annealing and the great deluge process. It was introduced by Dueck and Scheuer in 1990 [Dueck and Scheuer, 1990]. Similar to SA, TA also examines a single change to a current solution but uses a process which has a different set of acceptance rules. TA accepts every new proposed solution which is not sufficiently worse than the previous current solution. In contrast, in SA there is only a small probability that a worse proposed solution would replace the current solution. TA is a popular technique and could be applied to various safety problems.

Tabu Search (TS): The basic concept of *Tabu Search* (TS) is described by Glover in 1986 [Glover, 1990]. The overall approach is to avoid entrainment in cycles by forbidding or penalizing moves which take the solution, in the next iteration, to points in the solution space previously visited (hence "tabu") [Pukkala and Kurttila, 2005]. The method is still actively researched, and is continuing to evolve and improve. TS proceeds according to the supposition that there is no point in accepting a new (poor) solution unless it is to avoid a path already investigated. This insures new regions of a problems solution space will be investigated with the goal of avoiding local minima and ultimately finding the desired solution.

Particle Swarm Optimization (PSO): PSO, developed by Eberhart and Kennedy in 1995, is an exciting new methodology in evolutionary computation. It is similar to genetic algorithms in that the system is initialized with a population of random solutions. However, unlike other algorithms, each potential solution (*particle*) is also assigned a randomized velocity and flown through the problem hyperspace [Lee and El-Sharkawi, 2006]. Compared with genetic algorithms (GAs), the information sharing mechanism in PSO is significantly different. In GAs, chromosomes share information with each other. So the whole population moves like one group towards an optimal area. In PSO, only the best string gives out the information to others. It is a one-way information sharing mechanism. The evolution only looks for the best solution. Compared with GAs, all particles tend to converge to the best solution quickly even in the local version in most cases. The PSO algorithm has been found to be extremely effective in solving a wide range of engineering problems [Hu et al, 2004].

3.5 An Overview of Different Multi-Objective GAs

Among all modern optimization methods the multi-objective genetic (evolutionary) algorithms (MOGAs) are the most popular for the solution of multi-objective problems. Over the last decade a number of techniques from this group have been developed and successfully applied to different engineering problems. This section overviews the main methods in this category.

The Niched Pareto Genetic Algorithm (NPGA): This algorithm was proposed by Horn and Nafpliotis in 1993 [Horn and Nafpliotis, 1993]. It combines tournament selection and the concept of Pareto dominance. Firstly, two competing individuals and a comparison set of other individuals are picked at random from the initial population. Then if one of the competing individuals is dominated by any member of the set, and another is not, then the latter is chosen as the winner of the tournament. If both individuals are dominated or not dominated, the result of the tournament is decided by sharing. In order to reach a nice spread of solutions, most MOGAs employ some kind of niching. In most cases, niching is used as a secondary measure of fitness [Jensen, 2003] and is selected for reproduction. This procedure is called *phenotypic sharing* on the objective vector and is described in detail in Horn and Nafpliotis's study [Horn and Nafpliotis, 1993].

The Non-Dominated Sorted Genetic Algorithm (NSGA and NSGAII): This approach was developed by Srinivas and Deb in 1994 [Srinivas and Deb, 1994]. It is based on Goldberg's suggestions that fitness assignment must be carried out in several steps. Consequently, the main idea of NSGA is the ranking process executed before the selection operation [Dias and Vanconcelos, 2002]. The nondominated individuals in the population are first identified and then are assumed to constitute the first nondominated front with a large dummy fitness value. It must be considered that the same fitness value is assigned to all of them. Afterwards, the individuals of the first front are temporarily "ignored" and the same procedure is applied to the rest of population. Therefore, the individuals for the second nondominated front are identified. The process continues until the whole population is classified into nondominated fronts. The population is then reproduced according to the dummy fitness values. Over the years, NSGA was criticised mainly because of the following factors: high computational complexity of non-dominated sorting, lack of elitism, and need for specifying the sharing parameter.

In 2000 the NSGA approach was modified by Deb, etal. [Deb et al., 2002] and is known as Elitist Non-dominated Sorting Genetic Algorithm (NSGA-II). This new version compensates for the disadvantages of the first proposed method. Deb's work demonstrates that the NSGA-II approach has a better spread in its optimized solutions than the Pareto Archive Evolutionary Algorithm (PAES).

Pareto Archive Evolutionary Algorithm (PAES): This method was proposed by Knowles and Corne [Corne and Knowles, 2000] and modified by Grosan [Grosan et al., 2002]. In this approach one parent generates by mutation one offspring which is, afterwards, compared with the parent. Then there are three possible options: if the offspring dominates the parent it is accepted as the next parent and the iteration continues; if the parent dominates the offspring the offspring is discarded and a new offspring is generated by a new mutation; if the parent and the offspring do not dominate each other a comparison set of previously nondominated individuals is used. It is important to maintain population diversity among the Pareto front. Consequently, a new generated offspring is compared with the archive (a set of selected potentially fittest strings) to verify if it dominates any member of the archive: if yes, the new generated offspring enters the archive and is accepted as a new parent; if no, both parent and offspring are checked for their nearness with the solution of the archive; if the offspring resides in the least crowded region in the parameter space among the members of the archive, it is accepted as a parent and its copy is added to the archive.

Pareto-frontier Differential Evolution (PDE): The Differential Evolution approach was developed by Price and Storn in 1997 [Price and Storn, 1997] and applied by Babu and others to several multi-objective problems [Babu and Jehan, 2003]. It is a population based search algorithm and is an improved version of simple GAs. Simple GAs use binary coding for representing problem parameters. In contrast, PDE uses real coding of floating point numbers. Babu in his work indicates the following advantages of the PDE approach: simple structure, ease of use, speed and robustness. The PDE scheme is adaptive and simple. It ensures that mutation increments are automatically scaled to the correct magnitude. Furthermore, PDE uses a non-uniform crossover in that the parameter vectors of the offspring vector are inherited in unequal proportions from the parent vectors. Tournament selection is a method used for PDE reproduction: the offspring vector competes against one of its parents. The overall structure of this approach is similar to most other population based methods.

Pareto Envelope-based Selection Algorithm (PESA): This algorithm was proposed by Corne and Knowles in 2000 [Corne and Knowles, 2000]. This approach is a variant of a simple GA. It maintains two populations of chromosomes, with each chromosome storing optimization parameter values. The internal population contains a set of optimization parameters to be evaluated. The external population contains the partial Pareto front found thus far in the computation. The main goal of PESA is to create a new set of optimization parameters for evaluation by exploring areas of fitness space which are poorly represented in the Pareto front.

Dynamic Multi-objective Evolutionary Algorithm (DMOEA): This method was first proposed by Yen and Lu in 2002 in [Yen and Lu, 2002] in order to improve MOGA. In this approach, "instead of determining population size heuristically, a dynamic population growing and declining strategy are designed to help the algorithm allocate an optimal number of nondominated solutions for the final resulting Pareto set" [Lu, 2002]. The key techniques of DMOEA are: cell-based rank and density calculation scheme; population growing strategy; population declining strategy; objective space compression strategy.

Strength Pareto Evolutionary Approach (SPEA): This technique is one of the most popular among all major multi-objective genetic algorithms. SPEA was developed by Zitzler and Thiele in 1998 [Zitzler and Thiele, 1998]. According to Zitzler and Thiele, SPEA is similar to other multi-objective GAs as it: stores the Pareto-optimal solutions found so far externally; uses the concept of Pareto dominance in order to assign scalar fitness values to individuals; performs clustering to reduce the number of nondominated solutions stored without destroying the characteristics of the Pareto-optimal front. On the other hand, SPEA is unique. Zitzler and Thiele indicate this method's uniqueness in five respects:

1. It combines the above three techniques in a single algorithm;
2. The fitness of an individual is determined from the solution stored in the external Pareto set only;
3. Whether members of the population dominate each other is irrelevant;
4. All solutions in the external Pareto set participate in selection;
5. A new niching method is provided in order to preserve diversity in the population.

This method is Pareto-based and does not require any distance parameter. In 2001 Zitzler, Laumanns and Thiele proposed an improved version of this algorithm and named it SPEA2, which compensates the potential weaknesses of its predecessor. Both the SPEA and SPEA2 are discussed in detail in chapter 4.

3.6 Safety System Optimization

Attempting to optimize the design of engineered safety systems, the analyst is frequently faced with the demand of achieving several targets (e.g. low costs, high revenues, high reliability, low accident risks), some of which are often in conflict. At the same time, several requirements (e.g. maximum allowable cost, weight, volume etc.) should also be satisfied. This type of problem is usually solved by focusing the optimization on a single objective which may be a weighed combination of some of the targets of the design problem. During the last decade a number of engineers have applied various methods for different safety system optimizations. Among those methods genetic algorithms have been applied most often due to their simplicity and universality. In all cases they performed successfully. Details on this safety system optimization research are discussed in sections 3.6.1 and 3.6.2.

3.6.1 Safety System Optimization by Genetic Algorithms

In 1997 and 1999 Pattison described a design optimization scheme for systems that require a high likelihood of functioning on demand. A simple genetic algorithm (GA) had been chosen as the optimization method. The technique was applied to two safety systems: the simple High Integrity Protection System (HIPS) [Andrews and Pattison, 1997] and the Firewater Deluge System (FDS) on an offshore platform [Pattison, 1999]. For both systems the minimum unavailability has been achieved within one hundred generations with computation time of several hours.

In 2000 Cantoni [Cantoni et al, 2000] produced a simulation for optimal industrial plant design (choice of system layout and components) under conflicting safety and economic constraints. In this work an engineering analysis aimed at assessing the reliability,

availability and safety levels was coupled with an economic analysis, i.e. estimation of costs for plant downtime, maintenance and repair. The plant function is a complicated multivariate, non-linear function, which cannot be put explicitly in analytical form. Therefore, the authors presented an approach which couples the Monte Carlo simulation method for the evaluation of plant safety and economic performance, and the genetic algorithm for determining the optimal system design. The GA considers a population of chromosomes, each one encoding a different alternative design solution. For a given design solution, the Monte Carlo simulation evaluates the system performance over a specified mission time. This latter constitutes the objective function to be maximized by the GA through the evolution of the successive generations of the population. The results obtained by the combination of the GA and Monte Carlo simulation confirmed the good performance of the methodology implemented for both the system safety and economic objectives.

In 2001 Busacca [Busacca et al, 2001] presented an optimization of a safety system by GAs in which every target is considered as a separate objective to be optimized. Similarly to Cantoni [Cantoni et.al, 2000], Busacca considered the industrial plant. In this case the plant has two separate objectives:

1) the net profit drawn from system operation during the mission time, made from profit from plant operation, purchase and installation costs, repair costs and penalties during downtime due to missed delivery of agreed services;
2) and the reliability at mission time.

The multi-objective genetic algorithm (MOGA) had been implemented for this safety system optimization. The values of the parameters were chosen based on experience and trial-and-error tuning, so as to achieve proper convergence. The results show that the implemented genetic approach efficiently identifies the Pareto optimal solutions, i.e. the nondominated solutions.

The procedure was then applied to the standby system of nuclear power plant (NPP), consisting of three pumps and seven valves. The goal was to optimize the effectiveness of NPP with respect to three different criteria: mean availability, cost, and workers' time of exposure to radiation. As in the previous examples, these objective functions share some common contributions but present conflicting ones as well. The optimization

performed with respect to availability, economic and workers' safety objectives has shown the potential of the approach and the benefits which can be derived from a more informative multi-objective framework. However, the authors underline the fact, that Pareto optimality does not solve the decision problem. The decision maker is just provided with the whole spectrum of nondominated alternatives and their performances with respect to the objectives, and he must ultimately select the preferred one according to the preference values.

Similarly to Cantoni [Cantoni et al, 2000], in 2003 Marseguerra [Marseguerra et al, 2004] proposed the multi-objective optimization scheme for nuclear safety systems based on the effective coupling of genetic algorithms (MOGA) and Monte Carlo simulation. This technique was demonstrated on the Reactor Protection Instrumentation System (RPIS) of a pressurized water reactor. This safety system consists of analog channels, logic trains and trip breakers. Therefore, three decision variables of the optimization problem were considered. They are:

1) the surveillance test interval of the analog channels;
2) the surveillance test interval of the logic trains;
3) the allowable time in which a reduced safety margin of the protection system is acceptable, selected within (1, 50) hours.

Both test intervals were selected within the range (30, 365) days. The results were found to be physically reasonable. The approach was found valuable and provided the decision-maker with a useful tool for distinguishing those solutions that, besides being optimal with respect to the expected availability behaviour, give a high degree of confidence on the actual system performance.

Safety improvement of industrial installations leads to the optimal allocation of designs that use more reliable equipment, testing and maintenance activities to assure a high level of reliability, availability and maintainability for their safety-related systems. In 2004 Martorell [Martorell et al, 2004] considered a multiple-optimization problem, where the parameters of design, testing and maintenance act as decision variables. The authors summarized the problem formulation and fundamentals of two major groups of resolution alternatives:

- the traditional approach: the multi-objective problem is transformed into several single-objective problems, which are therefore solved by simple GAs;
- the alternative approach: when all objectives are considered and solved by MOGA. This approach results in a set of non-dominated solutions to make the final decision.

Both alternatives represented extreme options for the decision-making process. However, the best results were obtained by the SPEA2-based MOGA, which provides a better defined Pareto optimal front. Using this approach the search is performed without assigning a preference function and the final decision is taken a posteriori.

In 2006 Everson and Fieldsend [Everson and Fieldsend, 2006] introduced the multi-objective optimization of safety related and critical systems. They focused on the Short-Term Conflict Alert system (STCA), designed to raise a warning to air traffic controllers if there is a developing conflict between aircraft, giving them time to redirect the aircraft. For the system optimization the authors implemented a stochastic search algorithm MOEA, based on the GAs. The STCA has over 1500 parameters determining its behaviour. However, it was decided to optimize only 912 of them. The optimization time for this huge system took approximately 12 days. However, the method gave effective results, which are expected to be better after some future alterations to the traditional technique.

In 2008 Lin [Lin et al., 2008] developed a GA-based methodology for optimization of the inspection system for thin film transistor (TFT) liquid crystal display (LCD), which ensures high quality in an LCD production line. The proposed method was compared with the algorithms that use artificial parameter sets. Results indicated a good performance of the developed technique.

In 2009 Huang [Huang et al., 2009] suggested a GA-based method for optimization of unmanned crane routes, which would minimize the cycle time by calculating the most efficient combination of horizontal and vertical motions of container loading/unloading. The optimization criteria involved length, smooth degree and safety distance. The computational experiments testified the effectiveness of the algorithm and explored a new way to increase the efficiency of container loading/unloading process.

Various MOGAs have been proposed and applied since to different engineering system optimization problems since 1985. Over the last ten years, such methods are:

- *Multi-Objective Genetic Algorithm (MOGA)*: Yamachi [Yamachi et al., 2006] proposed to use a combination of *N*-version programming and MOGA for the engineering system reliability and total cost optimization and proved that the combined technique could slightly remove the roughness of Pareto solutions.
- *Non-dominated Sorting Genetic Algorithm 2 (NSGA2)*: Deb successfully tested NSGA2 on five different engineering problems [Deb et al., 2000)]. Dias and Vanconcelos [Dias and Vanconcelos (2002)] applied NSGA to two test problems and showed that it performs better that MOGA and can be applied to solve multi-objective problems in electromagnetics.
- *Pareto-frontier Differential Evolution (PDE)*: Babu and Jehan [Babu and Jehan, 2003] tested PDE on one simple general problem and on the engineering application of cantilever design problem. Results indicated that compared to the simple GA, PDE algorithm gives the exact optimum with less number of iterations.
- *Niched Pareto Genetic Algorithm 2 (NPGA2)*: Salazar [Salazar et al., 2006] demonstrated the use of NPGA2 to solve three types of reliability optimization problems: to find the optimal number of redundant components, find the reliability of components, and determine both their redundancy and reliability. The technique was successfully tested on four redundancy problems.
- *Pareto Envelope-based Selection Algorithm (PESA)*: Everingham [Everingham et al., 2003] successfully applied PESA to six image segmentation algorithms used in many practical computer vision systems.
- *Pareto Archive Evolutionary Strategy (PAES)*: Grosan [Grosan et al., 2002] proposed an improved version of PAES, which was tested on two engineering functions and performed better than the original PAES.
- *Strength Pareto Evolutionary Algorithm 2 (SPEA2)*:
  - Martorell [Martorell et al., 2006] successfully applied SPEA2 to the simultaneous optimization of periodic test interval and test planning performed on stand-by safety-related equipment.
  - Sareni [Sareni et al, 2004] applied SPEA2 to a number of test problems and proved that it performs as efficiently as NSGA2.

- Jensen [Jensen, 2003] compared SPEA2 performance to five other multi-objective techniques (i.e. NSGA2, DMOEA, PDE, PAES and PESA) by application to two example problems. Results showed that SPEA2 and NSGA2 performed better than other techniques.
- Zitler [Zitler et al. (1), 2001] and Kamiura [Kamiura et al., 2002] applied SPEA2 to two or three objectives In 2003 V. Khare [Khare et al., 2003] investigated MOGAs for their scalability with respect to the numbers of objectives from 2 to 8.

3.6.2 Safety System Optimization by Other Methods

A Multi-objective Design Optimization framework called Concurrent Subspace Design (CSD) had been applied to the design of an aircraft brake assembly in 1999 by Stelmack [Stelmack et al, 1999]. CSD is closely related to the Concurrent Subspace optimization method proposed by Sobieski [Sobieski, 1988]. The design optimization problem contained 14 design variables and 24 states. The initial database generated for this design problem contained 38 designs. One of them served as a starting point. Every other design was obtained by perturbing one of the 14 design variables away from its baseline value. The obtained results were encouraging. They demonstrate the ability of CSD framework to identify designs which are improved according to the objectives and requirements posed for a particular problem.

During the same time as Stelmack, Haimes [Haimes, 1999] completed his investigation of Los Angeles water system complexity and the misuse of modelling and optimization. In the maintenance of water distribution systems, different repair /replacement strategies for varying subsystems often have unexpected impacts on the overall system; the demands for the recourses and their appropriate allocations have a diverse impact on the system's reliability. This system multi-objective optimization was focused on five points: cost –benefit analysis (cost, benefit and risk objectives); expected value of risk; present value of money; reliability analysis; and the fallacy of optimization.

The linear optimization procedure has been applied to all objectives. Finally, the theoretical optimum solutions were found for each point. However, the authors stated that an optimum solution to a real-life problem depends on decision makers, the credibility of the database, etc. Therefore, a mathematical optimum to this model does not necessarily correspond to the optimum for the real-life water system.

In 2000 another group of engineers [Seward et al, 2000] presented an account of carrying out a hazard analysis to define the safety requirements for an autonomous robotic excavator (LUCIE). This particular safety problem differs from most others. Previously, hazard analysis was applied mostly to robotic manipulators, where the manipulator's environment is generally structured and can be controlled. However, in this case the system was required to interact directly with a natural environment: a building site. The safety manager (SM) is a target hardware LUCIE system processor which monitors the environment and ensures safe behaviour. Hence, the following three principal hazard definitions have been adopted from the SM hazard containment point of view:

1) Collision with an object on the surface.
2) Collision with an underground object.
3) Toppling of the excavator.

The problem was managed through a safety decision making process, which was implemented through a decision network based on the fault tree structures developed within the FTA. Furthermore, the decision network provided the method by which the excavator is returned to a safe state from an unsafe operating situation.

During the same time period as Seward, Terechine [Terechine, 2000] performed pulsed power system optimization for the Neutrino Factory (NF). The main idea of the system to use time dependent electrical field of an induction nature to correct 200 MeV muon energy spread after energy-time correlation developed in a drift space. The goal of this work was to identify the limits of choosing system parameters, and to optimize the system. Three system parameters were considered to be optimised: the power level, system reliability and operational cost. The author chose a linear optimization technique for the system optimization. The obtained results showed that optimization parameters are proportional to the length of the system. The increase of the length gives the

possibility to gain maximum power, increase system reliability and save in operation cost.

In 2003 Reif [Reif, 2003] presented the safety optimization method, based on the combination of fault tree analysis and mathematical techniques. The methodology can be described as follows:

Step 1. Carry out a fault tree analysis of the system hazards.
Step 2. Use a statistical distribution for failure probabilities.
Step 3. Estimate the costs of each hazard with a cost function.
Step 4. Perform mathematical optimization (usually probabilistic).

This method was illustrated on the height control system of the Elbtunnel in Hamburg. Two different hazards, the collision of overhigh vehicles with the tunnel entrance and the tripping of the false alarm, had been considered. The safety analysis had shown the benefit of the combination of all these methods. Formal verification had shown a design flow, which resulted in a safety gap. Safety optimization yielded optimal configuration values and made the system safer, led to design improvement and increased overall system quality.

In 2005 Acar [Acar and Haftka, 2005] proposed a probabilistic optimization method for the industrial safety system, in which the probability of failure calculation is confined to stress allowables and the stress distribution is condensed into a representative single value by an inverse transformation. The main goal of this method is to base the design on stress allowables whose probability distributions are more accurate. This method therefore serves as a way to take a first step for probability of failure calculations when probabilistic data is not scarce. The method is illustrated with an aircraft wing, horizontal tail and vertical tail system weight and safety optimization. Results show that probabilistic design renders about 0.77% weight saving for the same level of safety and 31% safety improvement while keeping the weight unchanged. The method was found to predict slightly lower (around 6%) failure probabilities when optimizing for weight and slightly higher (around 2%) failure probabilities when optimizing for safety.

At the same time Ortmeier [Ortmeier et al, 2005] reported on the safety analysis of a distributed and decentralized control of a railroad crossing: the radio-based level

crossing. The main points of optimization were safety and the mean cost. The authors decided to implement the mixture of the FTA technique for safety and a linear optimization procedure for the cost minimization. The cost function consists of two terms: one term for hazard costs and a second term directly dependent on the parameters themselves, for example, more reliable sensors are more expensive. The costs associated with the hazards were approximated by the sum of hazard probabilities. The results showed that the obtained optimum might not be the best solution from the economical point of view. However, the obtained optimum choice of parameters (cost and safety) not only allowed faster travel but also safer.

3.7 Conclusions

When reviewing the classical groups of optimization methods (discussed in section 3.3) the following points are noted with regards to their potential application:

1) The classical optimization techniques are analytical and can be applied only to objective functions that are continuous and differentiable. Therefore, this group of techniques can be hard to apply to safety system optimization problems as most do not satisfy these conditions. Another obvious disadvantage of classical optimization methods is that they are time-consuming.

2) Linear programming methods can be applied only to systems with linear objectives. The main disadvantage of these methods is that they are time-consuming for systems with a large number of design variables.

3) The majority of real safety systems involve the objective functions and constraints that are too complicated to manipulate. Therefore, the application of LP and Classical optimization techniques is difficult. In this case nonlinear programming methods (numerical methods) are helpful. However, these methods are efficient only if the initial interval of uncertainty is known.

4) Random search methods are not very efficient by themselves. However, they can be combined with more efficient techniques at the early stages of optimization to detect the region where the global minimum is likely to be found.

5) Geometric programming is suitable for safety system optimization, where the optimal value of the objective function (not the design variables) is the main subject of interest. However, this method is not applicable to the problems where the difference between the total number of terms appearing in the design constraints and the number of the design variables is less than one.

6) Dynamic programming is easy to implement. It is suitable for the solution of a wide range of complex optimization problems and, hence, can be successfully applied to any safety system optimization. The main disadvantage of this technique is a major drawback, known as the curse of dimensionality or state explosion problem.

7) In many safety systems certain design variables can take only discrete values. In this case integer programming should be applied or combined with the above techniques. Stochastic programming is applied in situations where some or all of the parameters of the optimization problem are described by random variables rather than deterministic quantities. For example, the atmosphere condition parameters affecting some of safety system's components cannot be predicted precisely in advance and, therefore, should be treated as random variables.

The modern heuristic optimization techniques, discussed in section 3.4, are certainly the most efficient and preferable for safety systems optimization. These methods have evolved to facilitate solving optimization problems that were previously difficult or impossible to solve. In practice all these techniques give good results in a short computation time. Nowadays one of the most powerful optimization method groups is GAs. Other efficient techniques are Great Deluge, Threshold Accepting and Particle Swarm Optimization.

During the last decade the MOGAs group of techniques became very popular and widely applicable for engineering problems with several optimization parameters. The SPEA technique and its improved version SPEA2 are the leaders in this group in terms of overall performance, i.e. the accuracy of the results, the search speed and complexity. These techniques are discussed in detail in the next chapter.

CHAPTER 4

GENETIC ALGORITHMS

4.1 Introduction

Genetic algorithms (GAs) have arisen from a need to model the biological processes of natural selection and population genetics. GAs are founded on Darwinian evolutionary principles and are global, parallel, adaptive heuristic search and optimisation methods. First described by John Holland in the 60s at the University of Michigan, GAs have been widely studied and experimented. The main theme of the research on genetic algorithms has been robustness [Holland, 1975]. The necessity is the balance between efficiency and efficacy for survival in many different environments. The system can perform its functions longer and better if higher levels of adaptations can be achieved. During the last few years GAs have been applied in a variety of areas. A significant contribution was achieved within safety systems engineering [Buckles and Petry 1992, Munoz 1997, Cantony 2000, Marseguerra 2000, Tsai 2001, Lapa 2003, Vinod 2004, Chen 2008]. GAs are attractive to problem solvers as they are not conducive to formal, rigorous, classical analysis. Moreover, GAs are well suited to solve multiobjective optimisation problems. During the past few years MOGAs were successfully applied to a number of systems safety and reliability problems [Elegbede 2003, Marseguerra 2004, Martorell 2005, Lin 2008]. Recently, some important multiobjective evolutionary algorithms (MOEAs) have been developed. Among these the Strength Pareto Evolutionary Algorithm (SPEA) and its improved version (SPEA2) seem to be the most effective techniques for multiobjective optimization problems.

4.2 Genetic Algorithm Differences from Traditional Methods

Improvement is the most important goal of optimisation. Consequently, optimisation seeks to improve performance towards some optimal point or points [Goldberg, 1989]. GAs surpass more traditional methods in some very fundamental ways. The main four differences are:

- Traditional methods work with the parameters themselves. On the other hand, GAs work with a coding of the parameter set.
- Traditional approaches start the search from a single point. GAs search from a population of points.
- GAs use objective function information. In contrast, traditional methods use derivates or other auxiliary knowledge.
- Traditional methods use deterministic rules. GAs are based on probabilistic rules.

With more traditional methods it is essential to analyse each of the parameters separately. Conversely, GAs require the initial parameter set of the optimisation problem to be coded as a finite-length string over some finite alphabet. Simple GAs use either binary or Gray coded strings [Pattison, 1999]. Both of these have a 2-character alphabet {0, 1}. The binary coding method is more popular among engineering system designers. It works in the following way: the right most bit of a typical 8-bit byte in the computer has the value 2 to the power 0, i.e. 2^0. The bit directly to the left is 2^1, the next 2^2 and so on. For example, for a string 11010:

$$11010 = 1\times 2^4 + 1\times 2^3 + 0\times 2^2 + 1\times 2^1 + 0\times 2^0 = 16 + 8 + 0 + 2 + 0 = 26.$$

In most optimisation methods some transition rules are used and move gingerly from a single point in the decision space to the next. This "point-to-point" process is dangerous as it is an ideal prescription to locate false peaks in multimodal search spaces. GAs avoid this problem by working with a database of points (population of strings) simultaneously climbing many peaks in parallel. Hence, the probability of finding a false peak is reduced.

In order to work properly many traditional techniques require much auxiliary information. As an example, gradient techniques need analytical or numerical derivatives to climb the current peak. "GAs are blind" – they only require objective function values associated with individual strings to perform an effective search for better and better solutions. This makes GAs a more canonical method than many other search schemes. However, the refusal to use specific information when it does exist can place an upper bound on the performance of the algorithm. The last main difference is

that GAs use probabilistic transition rules to complete their search. They use random choice as a tool to guide a search towards areas of the search space with likely improvement. All of the mentioned differences result in an advantage over the more commonly used techniques.

4.3 Simple Genetic Algorithms

Each individual within the initial population (*gene pool*) represents a particular solution (*chromosome, string, individual*) to the problem [Fleming and Purshouse, 2001]. This solution generally is expressed in some form of genetic code. A *Gene* is a variable contributing to the solution. The value of a gene is traditionally called an *allele*. Good solutions are selected and manipulated by *genetic operators* to achieve new, possibly better solutions [Chambers, 2001].

Figure 4.1 Schematic of the GAs

Figure 4.1 represents a schematic of the simple GA. Each individual within the gene pool is assigned a fitness value. This value expresses how "good" the solution is at solving the problem. Better solutions are assigned higher values that determine how successful the individual will be at propagating its code (its *genes*) to subsequent generations. A simple GA that produces good results in many practical problems is

composed of three operators: *reproduction*, *crossover* and *mutation*. In its general form, GA works through the following steps [Davis, 1991]:

Step 1. Creation of a random initial population of N potential solutions to the problem.

Step 2. Evaluation of each individual (chromosome) in terms of its fitness.

Step 3. Creation of a new chromosome by mating current chromosomes (parents).

Step 4. Applying genetic crossover, mutation and recombination operators to the parents to generate children.

Step 5. Deleting members of the population to make room for new chromosomes.

Step 6. Evaluation of the new chromosomes and placing them into the population.

Step 7. If the maximum number of generations is reached, stop and return the best chromosome; if not, go to step 3.

4.3.1 Genetic Operators

Reproduction. It is a process in which individual strings are copied according to their objective function values (fitness values). As a result, strings with a larger value have a higher probability of contributing one or more offspring in the next generation. This operator is an artificial version of natural selection, the Darwian survival of the fittest [Goldberg, 1989].

There are a number of ways the reproduction operator may be implemented in algorithmic form. As suggested by Goldberg, one of the easiest ways is to use a *biased roulette wheel selection method*. Each current string in the population has a roulette wheel slot sized in proportion to its fitness. Each time another offspring is required, a simple spin of the weighted roulette wheel yields the reproduction candidate. Hence, highly fit strings have a higher probability of being in the succeeding generation. If a string has been selected for reproduction, an exact copy of it is made. This string is then entered into a gene pool for further genetic operator action. The allocation space on the roulette-wheel is adapted for each string i according to the *selection function* p_i, which is calculated using equation 4.1.

$$p_i = \frac{f_i}{\sum_{i=1}^{N} f_j}, \quad j = 1,...,N . \tag{4.1}$$

where f_i is the fitness of string i.

For example, given the initial population of four strings 01100, 10001, 11100 and 00101. The values of objective (fitness) function for each of them is calculated according to the binary coding system described in section 4.2. These values give the percentage of population total fitness as shown in table 4.1.

Table 4.1 Sample Population of Strings and Fitness Values

No.	String	Fitness	% of Total
1	01100	12	19.4
2	10001	17	27.4
3	11100	28	45.1
4	00101	5	8.1
Total		**62**	**100**

Therefore, the first string (01100) has a fitness value of 12, which represents 19.4 percent of the total fitness. Consequently, string 1 is given 19.4 percent of the biased roulette wheel. As a result, each spin selects string 1 with probability 0.194.

In some GAs the inverse of the selection function (p_i^{-1}) is used to select chromosomes for deletion [Chambers, 2001]. The best chromosome is always preserved in the population (*elitist selection*).

Crossover. Goldberg points out that the mechanics of reproduction and crossover are surprisingly simple [Goldberg, 1989]. They involve random number generations, string copies, and some partial string exchanges. Crossover is a recombination operator and is characterised by the majority of authors as a distinguishing feature of the GA since it is the basic operator for producing new chromosomes.

Say, the chromosomes are selected in pairs (s_v, s_w), where $s_v = (v_1,...,v_n)$, $s_w = (w_1,...,w_n)$, $v_i, w_j \in \{0, 1\}$, and n is the number of elements in the chromosome. Then if $r \in [0, 1]$ is a random number (uniform distribution), *simple arithmetic crossover (one-point crossover)* can be defined as follows [Chambers, 2001]:

> Chromosomes s_v and s_w are crossed over at the k-th position. The resulting offsprings are: $s_v^{'} = (v_1,...,v_k, w_{k+1},..., w_n)$ and $s_w^{'} = (w_1,..., w_k, v_{k+1},..., v_n)$, where k is selected at random from $\{2,\ldots, n - 1\}$.

Mutation. Karr and Freeman [Karr and Freeman, 1999] describe mutation as an operation that provides a random element in the search process. This allows for various attributes of the candidate solutions to be occasionally altered. The operation of mutation begins by probabilistically selecting an individual from the population on the basis of its fitness [Koza, 1994]. A mutation point along the chromosome usually is chosen at random and the single character at that point is randomly changed. The altered chromosome is then copied into the next generation of the population.

Assuming that the chromosome is $s_v = (v_1,...,v_n)$ and $v_k^{\min}$ and $v_k^{\max}$ are the lower and upper bounds, respectively, on the parameter encoded by element k, $k \in \{1, 2,..., n\}$, mutation operators can be defined as [Chambers, 2001]:

1. Uniform mutation: a randomly selected element v_k, is replaced by $v_k^{'}$, which is a random number in the range $\left[v_k^{\min}, v_k^{\max}\right]$. The resulting chromosome is $s_v^{'} = (v_1,...,v_{k-1}, v_k^{'}, v_{k+1},...,v_n)$.

2. Multiple uniform mutation: uniform mutation of m randomly selected elements, where m is also selected at random from $\{1, 2,..., n\}$.

4.3.2 Genetic Algorithm Parameters

There are important global program variables that affect the operation of Genetic Algorithms. The most common GA parameters are:

- the population size,
- the maximum number of generations,
- the probability of crossover,
- the probability of mutation.

Greater population size and number of generations increase potentially the exploration of the search space and diversity from the onset. However, this is usually at the expense of efficiency and a balance for the particular problem must be attained.

The crossover rate is important as it controls the expected number of chromosomes to undergo the crossover operation per generation [Goldberg, 1989]. A higher crossover rate allows exploration of the solution space. Furthermore, it reduces the chances of settling for a false optimum. However, if this rate is too high a lot of computation time is wasted exploring unpromising regions of the solution space.

According to Holland [Holland, 1975], the function of the mutation rate is to control the rate at which each bit is changed to its opposing value. It is very important to choose a well-balanced mutation rate. If this rate is too low, it results in the fact that many useful genes are never tried out. In contrast, if it is too high, there will be many random perturbations, the offspring will start losing their resemblance to the parents. As a result, the algorithm will lose the ability to learn from the history of the search [Gen and Cheng, 1997]. Establishing the best values for the GA parameter set is the optimisation problem in itself.

4.4 Multiobjective Genetic Algorithms

4.4.1 Introduction

In the real world it is rare for any problem to concern only a single value or objective, usually there are several. Hence, multi-objective optimization problems should be solved. Computers have made rapid progress in hardware and software. Therefore, designers have begun to use computer simulations to aid decision-making for such structures as cars, airplanes, electric devices and safety systems. Usually, there is no

single optimal solution, but rather a set of alternative solutions. Consequently, one of the main goals of multi-objective optimization is to obtain a set of Pareto optimal solutions. Multiobjective Genetic Algorithms (MOGAs) have become immensely popular in recent years since they are multi point search methods and are particularly suitable to derive the Pareto optimal set [Zitler and Thiele, 1998]. Furthermore, they require little knowledge about the problem being solved. Another advantage is that MOGAs are easy to implement, robust, and inherently parallel. Due to their universality, these methods often take less time to find the optimal solution than other multiobjective approaches [Sbalzarini et al., 2000].

4.4.2 Dominance and Pareto-optimality

The main goal of multiobjective optimization is the search for acceptable solutions to problems that incorporate performance criteria. In most cases objectives are in competition with one another (*trade-off*). Improvement in one objective cannot be achieved without detriment to another. Figure 4.2 illustrates this situation [Purshouse and Fleming, 2001].

Figure 4.2 A Trade-off Between Two Competing Objectives

In order to treat simultaneously several objective functions, it is necessary to substitute the single-fitness based procedure employed in the single-objective GA for comparing

two proposals of solutions [Busacca et al., 2001]. The comparison of two chromosome-coded solutions with respect to several objectives may be achieved through the introduction of the concepts of *Pareto Optimality and Dominance*. Ivo Sbalzarini [Sbalzarini et al., 2000] defines them as follows:

Definition 4.1. Consider the following multiobjective optimization of the problem with *m* decision variables *x* (parameters) and *n* objectives *y* [Tillman et al., 1980]:

$$\text{Maximize} \quad y = f(x) = \left((f_1(x_1,...,x_m),...,f_n(x_1,...,x_m)\right) \tag{4.2}$$

where $x = (x_1,...,x_m) \in X$

$y = (y_1,...,y_n) \in Y$

and where

$\boldsymbol{x}$ – decision (parameter) vector,

$\boldsymbol{X}$ – parameter space,

$\boldsymbol{y}$ – objective vector,

$\boldsymbol{Y}$ – objective space.

A decision vector $a \in X$ is said to <u>*dominate*</u> a decision vector $b \in X$ (also written as $a \phi b$) if and only if:

$$\forall i \in \{1,...,n\} : f_i(a) \geq f_i(b) \wedge \exists j \in \{1,...,n\} : f_j(a) > f_j(b). \tag{4.3}$$

Additionally, it is said that *a* <u>*covers*</u> *b* ($a \underline{\phi} b$) if and only if $a \phi b$ *or* $f(a) = f(b)$.

Based on this convention, nondominated Pareto-optimal solutions are:

Definition 4.2. Let $a \in X$ be an arbitrary decision (parameter) vector.

(a) The decision vector *a* is said to be <u>*nondominated*</u> regarding a set $X' \subseteq X$ if and only if there is no vector in X' which dominates *a*; formally:

$$\neg\exists a' \in X' : a' \phi a. \tag{4.4}$$

(b) The decision (parameter) vector *a* is called <u>*Pareto-optimal*</u> if and only if *a* is nondominated regarding the whole parameter space *X*.

It is important to consider that:

1) Pareto-optimal parameter vectors cannot be improved in any objective without causing a degradation in at least one of the other objectives.
2) They represent in that sense *globally optimal* solutions.
3) The Pareto-optimal set does not necessary contain all Pareto-optimal solutions in X.

Definition 4.3. The set of objective vectors $f(a'), a' \in X'$ is called *Pareto-optimal front* or *Pareto-front*.

Consider the example shown in figure 4.2, Pareto optimality can be illustrated as Figure 4.3. The grey area of the left graph represents the entire dominated feasible region. The solid black curve on both graphs corresponds to the actual Pareto front. On the right graph the possible nondominated solutions from the Pareto front and dominated solutions from the feasible region are represented by circles and squares respectively. The dotted line shows the deviation between the actual and identified Pareto fronts.

Figure 4.3 Pareto Optimality

4.4.3 MOGA General Structure

Shaffer (1984) proposed genetic algorithms as multiobjective optimisers. However the first Pareto-based multiobjective genetic algorithm (MOGA) was developed by Fonsesa and Fleming in 1993 [Purshouse and Fleming, 2001]. Figure 4.4 represents one of the latest schematics of the MOGA [Chung and Alonso, 2004].

Figure 4.4 MOGA Schematic

As shown in figure 4.4, a random population is generated and their objective values are calculated like in the original GA. Then, to ensure all the nondominated individuals have the same level of reproductive potential, Goldberg's nondominated sorting procedure is implemented. Hence, the fitness value of each individual is determined based on the nondomination criterion rather than the objective function value itself. Based on the rank of nondominance, the population goes through the usual operators of the simple GA (selection, crossover and mutation) and checks if the nominal convergence among the population points is reached. If convergence isn't met, it returns to the function evaluation and nondominance ranking steps for the new generation,

other wise it goes to the reinitialization step. Two types of elitism are implemented in the reinitialization step. The first type is to pass the best solutions from the previous nominal convergence stage (the same elitism strategy as in the simple GA). The second type involves the storing of nondominated vectors produced from each cycle of the simple GA to an external file and inserting some of the best solutions generated so far as the reinitialized population for the simple GA. This process is applied at certain intervals for the intention of improving the nondominated solutions by getting closer to the true Pareto front or by getting a better distribution.

4.4.4 Pareto Ranking Procedure

The selection and replacement procedures of the MOGAs are based on Pareto ranking. This procedure is [Bussaca et al., 2001]:

1. All nondominated individuals in the current population are identified. These solutions are considered the best solutions, and assigned the *rank 1*.
2. Rank 1 solutions are virtually removed from the population and the next set of nondominated individuals are identified and assigned *rank 2*.
3. The process continues until every solution in the population has been ranked.

Figure 4.5 shows the Pareto ranking procedure for the small population with two objectives.

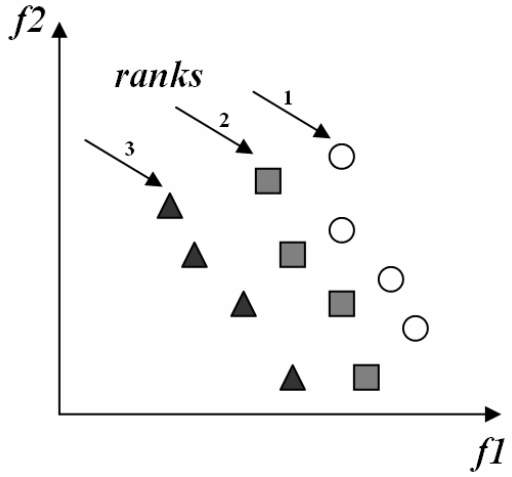


Figure 4.5 Population Ranking for a Problem of Maximization of Objective Functions *f1* and *f2*

It is important to notice that every solution belonging to the same rank class has to be considered equivalent to any other of the class, i. e. it has the same probability of the others to be selected as a parent and survive the replacement.

During the optimization search, an archive of a given number of nondominated solutions representing *the dynamic Pareto optimality surface* is recorded and updated.

At the end of each generation, nondominated solutions in the current population are compared with those already stored in the archive. Busacca [Busacca et al., 2001] suggests the following *archival rules*:

> **Rule 1.** If the new solution dominates existing members of the archive, those are removed and the new solution is added.
> **Rule 2.** If the new solution is dominated by any member of the archive, it is not stored.
> **Rule 3.** If the new solution neither dominates nor is dominated by any member of the archive then:
>
> - If the archive is not full, the new solution is stored.
> - If the archive is full, the new solution replaces the most similar one in the archive. An *Euclidian distance* based on the values of the fitness of the chromosomes can be used for this purpose:

Definition 4.4 If $x = (x_1, x_2, ..., x_n)$ and $y = (y_1, y_2, ..., y_n)$ are two *n*-dimensional vectors, their *Euclidean distance* $D(x, y)$ is given by the equation:

$$D(x,y) = x - y = \sqrt{(x_1 - y_1)^2 + (x_2 - y_2)^2 + ... + (x_n - y_n)^2} \ . \qquad (4.5)$$

The setup of an archive of nondominated solutions can also be exploited by introducing *an elitist parent's selection procedure* such that every solution in the archive is chosen once as a parent in each generation. If the population size *N* is too big, typically *N*/4 is used. The result of the optimization is constituted by the archive itself which gives the Pareto optimality region.

4.4.5 The Strength Pareto Evolutionary Algorithm (SPEA)

4.4.5.1 SPEA Overview

Among all major multiobjective evolutionary algorithms, the SPEA approach is one of the most popular. The algorithm was developed by Zitzler and Thiele in 1998 [Zitzler and Thiele, 1998]. The scheme of the algorithm is outlined in Figure 4.6. It can be described in the following way:

- First of all, the external Pareto set is updated: all nondominated individuals in the population are copied to the Pareto set.
- Subsequently, possibly dominated solutions are removed from it.
- If the number of externally stored Pareto solutions exceeds a given maximum, a reduced representation is computed by clustering.
- After fitness assignment individuals randomly picked out of the union of the population and Pareto set hold binary tournaments in order to fill the mating pool.
- Finally, crossover and mutation are applied to the population as usual.

Figure 4.6 The Strength Pareto Genetic Algorithm Scheme

4.4.5.2 The Basic Algorithm

There are 10 basic steps of the SPEA algorithm [Zitzler and Thiele, 1998]. They are:

Step 1. Generate random initial population P and create the empty external set of nondominated individuals P'.

Step 2. Evaluate the objective function for each individual in P in parallel.

Step 3. Copy nondominated members of P to P'.

Step 4. Remove solutions within P' which are covered by any other member of P'.

Step 5. If the number of externally stored nondominated solutions exceeds a given maximum N', prune P' by means of clustering. *Clustering* is a statistical method that obtains a set of elements and tries to clump similar elements together into "clusters". In hierarchical clustering the data are not portioned into a particular cluster in a single step. Instead, a series of partitions takes place, which may run from a single cluster containing all objects to n clusters each containing a single object. The *average linkage method* (the hierarchical clustering algorithm), which has proven to perform well on Pareto optimization, can be chosen:

Average linkage method [Morse, 1980] uses the average distance between all pairs of objects in cluster r and cluster s:

$$d(r,s) = \frac{1}{n_r n_s} \sum_{i=1}^{n} \sum_{j=1}^{n} dist(x_{ri}, x_{sj}), \tag{4.6}$$

where n_r and n_s are the sizes of the clusters r and s respectively.

At each stage of hierarchical clustering, the clusters r and s, for which $d(r,s)$ is minimum, are merged.

Step 6. Calculate the fitness of each individual in P as well as in P'. All individuals in P and P' are assigned a scalar fitness value. *First*, all members of the nondominated set P' are ranked. *Second*, the individuals in the population P are assigned their fitness value.

a) Each solution $i \in P'$ is assigned a real value $s_i \in [0,1)$, called *strength* (The strength of a Pareto solution is at the same time its fitness), s_i is

proportional to the number of population members $j \in P$ for which $i \underline{\phi} j$:

$$s_i = \frac{n}{N+1} = f_i, \tag{4.7}$$

Where n - the number of individuals in P that covered by *i;*
N - the size of *P;*
f_i - the fitness of *i.*

b) The fitness of an individual $j \in P$ is calculated by summing the strengths of all external nondominated solutions $i \in P'$ that cover *j*. Add one to this sum to guarantee that members of P' always have better fitness than members of P (note that the fitness is to be minimized):

$$f_i = 1 + \sum_{i,i \geq j} s_i, \; f_i \in [1, N). \tag{4.8}$$

- If all Pareto solutions have equal strengths, the fitness of an individual is completely determined by the number of Pareto points that cover it – that is identical to the Pareto ranking procedure considered before.

- However, in the case when the population is unbalanced the strengths come into play: the *stronger* a Pareto point the *less* fit the covered individuals.

Step 7. Select individuals from $P + P'$ (multiset union), until the mating pool is filled. The selection can be done using the following binary tournament procedure:

a) Randomly (uniformly distributed random numbers) select two individuals out of the union $P + P'$.

b) Copy the one with the better (i. e. lower for SPEA) fitness value to the mating pool.

c) If the mating pool is full, then stop, else go to step (a).

Step 8. Adapt step sizes of the members of the mating pool. Adaptation of the step sizes can be done using *the self-adaptive mutation method* [Hodson, 2001]:

Each element of P and P' is assigned an individual step size for every parameter, i. e. $\sum = diag\left(\sigma_i^2\right)$ is a diagonal matrix for each individual. The step

sizes of all members of the mating pool are then either increased by 50%, cut to half, or kept the same, each having a probability of 1/3.

Step 9. Apply recombination (crossover) and mutation to members of the mating pool in order to create a new population *P*.

Step 10. If the maximum number of generations is reached, then stop, else go to Step2.

4.4.6 SPEA2: Improved Strength Pareto Algorithm

In 2001 Zitzler, Laumanns and Thiele proposed an improved version of this algorithm and named it SPEA2, which compensates for the potential weaknesses of its predecessor.

Zitzler, Laumanns and Thiele in their work identify three main differences of SPEA2 in comparison to SPEA [Zitzler et al. (2), 2001]. They are:

1. An improved fitness assignment scheme is used, which takes into account for each individual how many individuals it dominates and it is dominated by: in SPEA individuals that are dominated by the same archive members have identical fitness values.
2. A nearest neighbour density estimation technique is incorporated which allows a more precise guidance of the search process: in SPEA if many individuals of the current generation are indifferent (do not dominate each other) no or very little information can be obtained from the partial order defined by the dominance relation. This situation often occurs in the presence of more than two objectives.
3. A new archive truncation method guarantees the preservation of boundary solutions: the clustering technique in SPEA is able to reduce the nondominated set without destroying its characteristics. However, it may loose outer solutions, those should be kept in the archive in order to obtain a good spread of nondominated solutions.

The suggested overall algorithm is as follows:

Step 1. *Initialization*: Generate an initial population and create the empty archive (external set).

Step 2. *Fitness assignment*: Calculate fitness values of individuals in initial population.

Step 3. *Environmental selection*: Copy all nondominated individuals to the archive. If its size exceeds the allowable size then reduce the archive by means of the truncation operator, otherwise fill the archive with dominated individuals from initial population. Important notice: the number of individuals contained in the archive is constant over time.

Step 4. *Termination*: If the maximum number of generations is reached or another stopping criterion is satisfied then set the nondominated set to the set of decision vectors represented by the nondominated individuals in the archive. Stop.

Step 5. *Mating selection*: Perform binary tournament selection with replacement on the archive in order to fill the mating pool.

Step 6. *Variation*: Apply recombination and mutation operators to the mating pool and set the archive to the resulting population. Increment generation counter and go to *Step2*.

Zitzler and others [Zitzler et al. (2), 2001] points out that in contrast to SPEA, SPEA2:

- Uses a fine-grained fitness assignment strategy which incorporates density information.
- The archive size is fixed (whenever the number of nondominated individuals is less than the predefined archive size, the archive is filled up by dominated individuals). With SPEA the archive size may vary over time.
- The clustering technique, which is invoked when the nondominated front exceeds the archive limit, has been replaced by an alternative truncation method which has similar features but does not loose boundary points.
- Only members of the archive participate in the mating selection process.

4.5 Summary

The GAs is a powerful group of the modern optimization techniques, which can be applied to most engineering optimization problems. In the case when several objectives are considered, as in this study, multi-objective genetic algorithms can be used. These methods are based on simple GA principles, are easy to apply and find a set of nondominated optimal solutions quickly. The SPEA2 technique is the leader of the group and, hence, is chosen for the safety system optimization in this study.

CHAPTER 5

HIPS SYSTEM ANALYSIS

5.1 Introduction

A safety system is an essential part of an industrial system as it operates to prevent the occurrence of certain conditions and their future development into a hazardous situation. Failure of such safety systems may have catastrophic consequences. Members of the workforce or the public could be injured or even killed. To minimise the likelihood of a hazardous situation, safety systems must be designed to maximise their availability. In this work the aim is to investigate a design optimisation scheme which yields an optimal safety system design by fully utilizing available resources.

5.2 General Structure of HIPS System

In this work the design optimisation scheme has been applied to a simple High-Integrity Protection System (HIPS). The main function of the HIPS is to prevent a high-pressure surge passing through it. Protection is provided for processing equipment whose pressure rating could be exceeded. The high pressure originates from a production well of a not normally manned offshore platform and the pieces of equipment to be protected are located downstream on the processing platform. Figure 5.1 represents the main features of the HIPS [Andrews and Pattison, 1999].

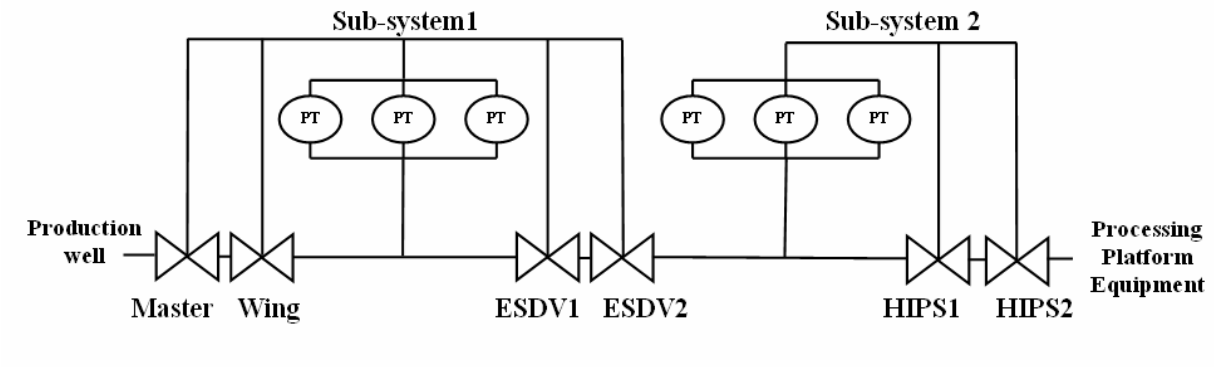


Figure 5.1 Structure of High-Integrity Protection System

The HIPS is divided into two separate subsystems. Sub-system 1 is the Emergency Shutdown or ESD sub-system. This is the first level of protection of the HIPS. The ESD system acts to close the Wing and Master valves together with any ESD valves that have been fitted when pressure in the pipeline exceeds the permitted value. This value is monitored using pressure transmitters (PT).

Sub-system 2 provides an additional level of protection. Inclusion of the high-integrity protection system incorporates this second level of redundancy. An important fact is that the latter sub-system is completely independent in operation. Its method of protection is the same as the ESD system.

5.2.1 Description of the Main Design Variables

The HIPS is a relatively simple system, yet there are a huge number of design options which can be considered. Ten main design variables describe this particular system. These variables, their description and evaluation limits are shown in table 5.1.

Table 5.1 HIPS Variables

Variable	Description	Value
θ_1, θ_2	Inspection intervals for subsystems 1 and 2	1week – 2 years
V	Valve type	1 or 2
P	Pressure transmitter type	1 or 2
N_1, N_2	Number of pressure transmitters fitted in subsystem 1 and 2 respectively	1 – 4 0 – 4
K_1, K_2	Number of pressure transmitters required to trip (activate) for subsystem 1 and 2 respectively	1 – N_1, 0 – N_2
E	Number of ESD valves fitted	0, 1, 2
H	Number of HIPS valves fitted	0, 1, 2

Two points to note are firstly that it is assumed that whatever valve type is selected all valves within the system are fitted as this type. This is true of the pressure transmitter type also. The second point is that the number of pressure transmitters required to activate the closure of valves on subsystem 1 or 2 is a function of the number installed (N_1, N_2).

The number of potential design variations considering just ten design variables is 45,158,400. It would be impractical to evaluate exhaustively each potential design. Furthermore, it is a complex task to understand the interaction between all the design variables and is practically impossible for any design engineer to do by hand. A technique is required to determine the 'best' design option in a more practical manner. This is to be achieved using a computerised optimisation algorithm.

5.2.2 Failure Data for HIPS Components and Design Limitations

Each hardware component of the HIPS can fail either in a dormant mode or spuriously. A dormant failure can be described as the inability of the component to carry out its desired task on demand. In contrast, spurious failure results from the component carrying out its desired function when its operation is not required. Table 5.2 shows the failure rate and mean repair time for each HIPS component in both dormant and spurious failure modes. This data will be used subsequently when calculating the unavailability and spurious trip probability of the HIPS.

Each combination of HIPS variables gives a new system design with its particular features such as cost, maintenance down-time, etc. The choice of system design is not unlimited. In this case, there are three limitations on the available resources. The total cost of the system must be less than one thousand units. The average time each year that the system resides in the down state due to preventative maintenance is a maximum of one hundred and thirty hours. If the number of times that a spurious system shutdown occurs is more than once per year then it is deemed unacceptable. Hardware costs for each component in the system as well as times taken to service each component at each maintenance test are shown in table 5.2.

Table 5.2 Component Failure Data

Component	Dormant Failures		Spurious Failures		Cost	Test Time (hours)
	Failure Rate (per hour)	Mean Repair Time (hours)	Failure Rate (per hour)	Mean Repair Time (hours)		
Wing Valve	1.14×10^{-5}	36.0	1×10^{-6}	36.0	100	12
Master Valve	1.14×10^{-5}	36.0	1×10^{-6}	36.0	100	12
HIPS Valve 1	5.44×10^{-6}	36.0	5×10^{-7}	36.0	250	15
HIPS Valve 2	1×10^{-5}	36.0	1×10^{-5}	36.0	200	10
ESDV Valve 1	5.44×10^{-6}	36.0	5×10^{-7}	36.0	250	15
ESDV Valve2	1×10^{-5}	36.0	1×10^{-5}	36.0	200	10
Solenoid Valve	5×10^{-6}	36.0	5×10^{-7}	36.0	20	5
Relay Contacts	0.23×10^{-6}	36.0	2×10^{-6}	36.0	1	2
Pressure Transmitter 1	1.5×10^{-6}	36.0	1.5×10^{-5}	36.0	20	1
Pressure Transmitter 2	7×10^{-6}	36.0	7×10^{-5}	36.0	10	2
Computer Logic	1×10^{-5}	36.0	1×10^{-5}	36.0	20	1

5.2.3 HIPS Cost Evaluation

There are two main categories of constraints: explicit and implicit. Explicit ones can be determined and easily evaluated from an explicit function of the design variables. In contrast, implicit constraints can only be evaluated by a full analysis of the system. Cost of the HIPS design is an explicit constraint and is represented by the following equations:

$$Cost = Cost(subsystem1)+Cost(subsystem2) \leq 1000 \tag{5.1}$$

$$Cost(subsystem1)= E(V_1C_{VE1} + V_2C_{VE2} + C_s) + N_1(P_1C_{P1} + P_2C_{P2}) + 261 \tag{5.2}$$

$$Cost(subsystem2)= H(V_1C_{VH1} + V_2C_{VH2} + C_s) + N_2(P_1C_{P1} + P_2C_{P2}) + 21 \tag{5.3}$$

where

$C_{V1} = C_{VE1} = C_{VH1}$ – cost of the valve type 1,

$C_{V2} = C_{VE2} = C_{VH2}$ – cost of the valve type 2,

C_{P1} – cost of the PT type 1,

C_{P2} – cost of the PT type 2,

C_s – cost of the solenoid valves.

The constant 261 (equation 5.2) for subsystem 1 is the additional cost for the wing and master valve, their solenoid valves, the computer and control relays. Subsystem 2 has a fixed cost of 21 units (equation 5.3) for the computer and control relays.

From equations (5.1) - (5.3) the cost of each system design is:

$$\textbf{Cost} = (E+H)(V_1C_{V1} + V_2C_{V2} + C_s) + (N_1 + N_2)(P_1C_{P1} + P_2C_{P2}) + 282 \qquad (5.4)$$

5.2.4 HIPS MDT Evaluation

Similarly to the cost for the HIPS, average MDT is calculated as a sum of the maintenance down time subsystem 1 and subsystem 2 for each potential design:

$$MDT = MDT(Subsystem1) + MDT(Subsystem2) \le 130, \qquad (5.5)$$

$$MDT(Subsystem1) = \frac{52}{Q_1}\left[E\left(V_1M_{VE1} + V_2M_{VE2} + M_S\right) + N_1\left(P_1M_{P1} + P_2M_{P2}\right) + 47\right], \qquad (5.6)$$

$$MDT(Subsystem2) = \frac{52}{Q_2}\left[H\left(V_1M_{VH1} + V_2M_{VH2} + M_S\right) + N_2\left(P_1M_{P1} + P_2M_{P2}\right) + 13\right]. \qquad (5.7)$$

Where

$M_{VE1} = M_{VH1} = M_{V1}$ – test time of the valve type 1,

$M_{VE2} = M_{VH2} = M_{V2}$ – test time of the valve type 2,

M_{P1} – test time of the pressure transmitter 1,

M_{P2} – test time of the pressure transmitter 2,

M_S – test time of the solenoid valve.

The expression $52/Q_2$ (equations 5.6 - 5.7) gives the number of times the system is down in a year. The constant 47 (equation 5.6) is the sum of the test times for the wing and master valve, their solenoids, the computer and control relay for subsystem 1. Similarly for subsystem 2, the sum of the test time for the computer and control relay is 13 units (equation 5.7).

From equations (5.5) – (5.7) MDT for each design can be calculated using equation 5.8:

$$\boldsymbol{MDT} = 52\left(h1 \cdot h3 + h2 \cdot h4 + h5\right), \tag{5.8}$$

where

$$h1 = V_1 M_{V1} + V_2 M_{V2} + M_S ,$$

$$h2 = P_1 M_{P1} + P_2 M_{P2} ,$$

$$h3 = \frac{E}{Q_1} + \frac{H}{Q_2},$$

$$h4 = \frac{N_1}{Q_1} + \frac{N_2}{Q_2},$$

$$h5 = \frac{47}{Q_1} + \frac{13}{Q_2}.$$

5.3 HIPS Analysis

The most important feature of each safety system is its ability to operate when the demand arises. Therefore, the objective is to minimise system unavailability, which means to minimise the probability of system failure on demand. Ideally, using the design alternatives, it is essential to determine which potential system design would produce the highest functionality.

In practice, certain factors need to be taken into account, for this application, the available resources. In section 5.2.2 the limitations on resources were defined as cost, maintenance effort and spurious frequency. The design options should improve the HIPS performance without violating the constraints. Consequently, the evaluation of each constraint is required in order to assess the overall desirability of each design option.

System design performance can be obtained by using the fault tree analysis method. Fault trees are used to quantify the system unavailability of each potential design. Constructing a fault tree for each design variation would be a time consuming task, hence, impractical. One way to solve this problem could be to use automatic fault tree construction methods, if they were available. An alternative is to resolve this difficulty by using house events. These enable the construction of a single fault tree capable of representing the causes of the system failure mode for every possible system design.

The top event of the HIPS unavailability fault tree represents the causes of the system failing to protect the processing equipment. The top event 'Safety system fails to protect' will occur if all (Wing, Master, ESD and HIPS) valves along the pipeline fail to close. In total the fault tree consists of 154 gates, 38 basic events representing component failures, and 40 house events representing design options. The unavailability fault tree construction process is discussed in appendix A (section A.1). Appendix C (section C.1) represents its detailed structure.

The spurious trip frequency for each design is also an implicit constraint that requires the use of fault tree analysis to assess its value. House events are again used to construct a fault tree capable of representing each potential design for this failure mode. The causal relationship 'HIPS fails spuriously' is represented by the sub-events 'Wing or Master Valve Fails Spuriously', 'ESD Subsystem Fails Spuriously' and 'HIPS Subsystem Fails Spuriously' related by 'OR' logic. The fault tree consists of 142 gates, 38 basic events and 40 house events. This fault tree construction details are shown in appendix A (section A.2). The detailed spurious trip fault tree structure is represented in appendix C (section C.2).

5.4 Review of Previous Work on HIPS

5.4.1 Introduction

In 1999 Rachel Pattison completed her work on safety systems optimization. In her thesis [Pattison, 1999] she analyzed the High Integrity Protection System (HIPS), which is described in section 5.2.

Pattison investigated the effectiveness of Genetic Algorithms to optimize HIPS availability. In her work she applied the SGA_C algorithm, which is a C-language translation and extension of the original Pascal Simple Genetic Algorithm (SGA) code developed by Goldberg [Goldberg, 1989]. This package was used as a framework to build the GA optimization software called GASSOP (Genetic Algorithm Safety System Optimization Procedure).

5.4.2 HIPS Data Structure

Each solution is a string of 1's and 0's representing a particular system design which depends on the values of the ten HIPS parameters (Table 5.1). Each parameter was allocated a particular length of the string depending on its maximum value. In total each string, representing all design variables, is 32 bits in length. Table 5.3 shows the chosen binary representation of each string parameter. The order of parameters in the string is as listed.

Table 5.3 Binary Representation of the HIPS Parameters

Variable	Description	Number of bits
θ_1, θ_2	Inspection intervals for subsystems 1 and 2	7
V, P	Valve and Pressure transmitter type	1
N_1, N_2, K_1, K_2	Number of pressure transmitters fitted and required in subsystem 1 and 2	3
E, H	Number of ESD and HIPS valves fitted	2

5.4.3 String Fitness Evaluation and Penalty Formulas

The system unavailability is the main optimization criterion. However, resources are not inexhaustible. Therefore, the following limits were considered:

- Cost $\leq$ 1000 units,
- Maintenance Down Time: MDT $\leq$ 130 hours,
- Spurious system failure: $F_{sys} \leq 1$ per year.

If these three parameters exceed their respective limits, the following penalty equations were implemented in the GASSOP:

1. The method utilised tries to form a direct relationship between cost and performance. Therefore, if cost exceeds its allocated limit by 100 units, i. e. 10% of the total budget (total cost 1000 units), a corresponding increase in

performance of at least 10% expected. In this case the assumption was made that a typical system cost would have an unavailability of about 0.002. Hence, if the cost increases to 1100 units, it is expected that system unavailability will decrease to 0.0018. String designs with excessive cost will not be adopted so the more the constraint violation the heavier the penalty. This is implemented using an exponential relationship of the form $y = x^{5/4}$. Therefore, the penalty function for excess cost, C_p, is:

$$C_p = \left(\frac{COST - 1000}{1000} \right)^{\frac{5}{4}} \times 0.002 . \tag{5.9}$$

2. If the MDT of a particular design (string) exceeds 130 hours, a contribution is made to the unavailability of the system in the term of a penalty. The respective penalty is:

$$M_p = \frac{(MDT - 130)}{8760}, \tag{5.10}$$

where 8760 is the number of hours per year.

3. The third constraint, excess spurious trip occurrence, is also related to cost. If a spurious trip occurs, production ceases hence a financial loss is incurred. It was assumed that the cost per hour for loss of production is 100 units. On average a spurious trip requires 36 hours to repair and only one such occurrence a year is acceptable. Using the cost penalty formula (5.9) the spurious trip penalty can be expressed as:

$$S_p = \left(\frac{Excess\cos t}{1000} \right)^{\frac{5}{4}} \times 0.002 . \tag{5.11}$$

Each penalty is subsequently added to the system unavailability. Hence, penalised system unavailability is:

$$Q'_{sys} = Q_{sys} + C_p + M_p + S_p . \tag{5.12}$$

The result is a sole fitness value for each design referred to as its penalised system unavailability Q'_{sys}.

5.4.4 GA Operators

Each string enters the selection procedure with an associated fitness value Q'_{sys} (equation 5.12). The smaller Q'_{sys}, the fitter the string and, therefore, the greater should be its chance of reproduction. In such cases, a possible approach is to base reproduction probability on availability concepts rather than unavailability. This, however, does not provide a useful measure of selection potential. The desirable solutions have unavailability values in the range 0 to 0.1 and hence availability values in the range 0.9 to 1.0. Pattison in her work solves this problem by applying a specialized conversion method, which allocates each string to one of three categories according to its fitness value (Table 5.4).

Table 5.4 Categories Represented in the Fitness Domain

Fitness value domain				
Upper limit Unavailability		**Category**		**Lower limit Unavailability**
1.0	≥	3	>	0.2
0.2	≥	2	>	0.1
0.1	≥	1	≥	0

Category 3 comprises of poor system designs (those with unavailability in the range 0.2-1.0). Each string is automatically allocated zero percent on the roulette wheel and, consequently, will be eliminated from the succeeding generation. Category 2 contains average merit designs (those with unavailability between 0.1 and 0.2). To preserve sensitivity each string is subtracted from the upper category limit of 0.2, and subsequently allocated some portion of a total of 5% of the roulette wheel. The first category strings are of ultimate interest and, therefore, the remaining 95% of the roulette wheel is allocated between each string in this category.

The standard crossover and mutation operators are used in the GASSOP program. These operators may produce an infeasible design. There are two possibilities: either a parameter value may be altered such that it lies outside its designated range or interactions between parameters may render the design infeasible. Pattison in her work implemented a design checking procedure after any parameter manipulation:

5.4.5 GASSOP Results

Pattison implemented GASSOP with the following parameter values:

1. Population of 20 strings.
2. Maximum number of generations 100.
3. Mutation rate 0.01.
4. Crossover rate 0.7.

In total 100 system evaluations were performed and gave the best results in generation 70 when the fittest string from the entire process arose (Table 5.5).

Table 5.5 Characteristics of the Best Design

Subsystem 1	No. of ESD valves	0
	No. of PTs	4
	No. of PTs to trip system	2
	MTI	45
Subsystem 2	No. of HIPS valves	2
	No. of PTs	1
	No. of PTs to trip system	1
	MTI	22
Valve type		2
PT type		1
MDT		127.74
Cost		822
Spurious trip		0.847
System unavailability		9.36e-4

5.4.6 Determining the Best GA Parameters

In the GASSOP algorithm there are four main GA parameters. They are: population size, maximum number of generations, crossover and mutation rates. Using GASSOP an analysis was carried out to investigate the effect of changing these parameter values. Regarding De Jong's test bed of five trial solutions [De Jong (2), 1975] and two criteria of goodness defined by Goldberg [Goldberg, 1989] a limited set of values for each parameter was chosen as follows:

Mutation rate:	0.1	0.01	0.001	
Crossover rate:	0.5	0.6	0.7	0.8
Population size:	10	20	50	

The results showed that larger populations lead to a better performance, i. e. obtain solutions with smaller unavailability. When the population size doubles from 10 to 20 strings the fitness value improves by 20%. An additional 18% improvement is achieved when the initial population is further increased to 50 strings. The mutation rate parameter implies that the largest rate, 0.1, leads to the generation of a fitter string. The crossover parameter again produced a slight bias toward the highest value. Following this investigation modifications were made to the genetic algorithm.

5.4.7 GA Modifications

After implementing GASSOP to the HIPS optimization some areas of improvement became apparent. The system unavailability plus penalty due to violation of cost, MDT and spurious trip frequency are important factors since an inaccurate calculation of any one will result in a reproduction probability that does not precisely reflect the fitness of the string.

MDT Modification: Three methods to improve the exploration of the range of values allowed by the MTI parameters have been explored. The best improvement of the results has been achieved with the method 2a: in addition to the MDT the system unavailability is evaluated for each MTI combination. Should any pair of test intervals

exceed 130 hours MDT the respective penalty is added. The test intervals resulting in the lowest penalised system unavailability are retained.

Modified Cost Penalty Formula: The cost penalty in the original GA is derived from equation (5.11). A base level in system performance (a system unavailability of 0.002) is assumed.

$$Cost\ Penalty = (Excess\ Cost / 1000)^{5/4} \times 0.002.$$

An alternative cost penalty, described in Pattison's work, takes into account both the cost violation and the system unavailability. This is achieved using a multiplying factor which, rather than being fixed, varies according to the system unavailability of the design. Hence, the modified cost penalty formula can be expressed as:

$$Cost\ Penalty = (Excess\ Cost / 1000)^{5/4} \times System\ Unavailability. \quad (5.13)$$

Results comparison show that altering the cost penalty formula enables a more detailed exploration around the border of the search space. The lack of final system designs from the original GA that portray a slight excess in either the cost or trip constraints implies that the penalty exerted is too great.

Modified Conversion Method: Each string receives a measure of its fitness (the string's penalised system unavailability) in the environment. The conversion method gives rise to reproduction probabilities in accordance with the fitness value of each string. The previously discussed conversion method (section 5.4.4) is rather crude. Its effectiveness is dependent on the range and distribution of fitness values being converted. For example, a population of values ranging between 0.009 and 0.4, produces a set of roulette wheel percentages with a fairly accurate proportional representation. A much fitter population (most systems having a fitness value less than 0.01) gravitates towards a set of almost equal percentages, although the actual fitness values vary quite markedly. The main disadvantage of the previously proposed method is that it is insensitive to small yet significant differences between the fitness values of the string. Additional problems occur when either a very high (or low) proportion of strings fall into a particular category.

Pattison investigates an alternative conversion method. It is able to cope with very diverse populations and is simultaneously able to show sensitivity to a highly fit set of strings. Initially nine categories are depicted. They cover the area of the fitness domain of importance (below 0.2). Furthermore, each category is assigned a particular weight. As the category gets fitter, its weight increases in size. The fundamental steps of the modified method are then as follows:

Step 1. Firstly each category is designated a particular percentage of the roulette wheel depending on:

1. The number of strings, *n*, of the total population, *N*, in the category.
2. The weight assigned to the category.

Step 2. The percentage allocated to each category is then distributed appropriately between the strings within. The method used must ensure that a system in a fitter category is given a greater percentage than a poorer design in a less fit category.

The results show that the modified conversion method generally improved the overall system performance. The original method is not able to cope adequately with highly fit populations of system designs. However, the new method is able to differentiate between the strings in the fitter population without loss of essential information during the conversion process.

5.4.8 Modified GA

In Pattison's work a modified GA was created with MDT modification method 2a, the modified cost penalty and modified fitness conversion method. 10 runs of GA with the population size, crossover rate, mutation rate and the number of generations equal to 20, 0.7, 0.01 and 100 respectively. Table 5.6 represents the characteristics of the best overall.

Table 5.6 Characteristics of the best Design

Subsystem 1	No. of ESD valves	0
	No. of PTs	2
	No. of PTs to trip system	1
	MTI	29
Subsystem 2	No. of HIPS valves	2
	No. of PTs	3
	No. of PTs to trip system	2
	MTI	32
Valve type		2
PT type		1
MDT		128.43
Cost		822
Spurious trip		0.717
System unavailability		6.57127e-4

The modified GA demonstrates the ability to find and explore the fittest areas of the search space by full usage of available MDT resources and through exploration of the boundary of the domain. The GA is able to differentiate between highly fit strings as the algorithm progresses and retention of the best design over latter generations is achieved.

5.5 Summary

The high integrity protection system is a relatively simple safety system. However, it has a large number of design options and the optimization tool is required in order to find the optimal system design within certain constraints. Previously the simple GA based technique and its modified version have been successfully applied to the HIPS optimization. However, in reality often more than one optimization criterion should be considered, for example, the constraints in this problem could be set as optimization objectives; and, therefore, the multi-objective technique should be applied to the system. The next chapter describes this research on the HIPS optimization by a developed SPEA2 based tool.

CHAPTER 6

HIPS OPTIMIZATION BY SPEA2

6.1 Program Structure

One of the research objectives was to investigate the effectiveness of the Improved Strength Pareto Evolutionary Algorithm (SPEA2), described in section 4.4.6, to optimise the High Integrity Protection System (HIPS) with regards to the unavailability, cost, MDT and spurious trip frequency. The C++ package [Deitel (2003), Mcgrath (2004)] was used to build the HIPS optimisation software called ISPEASSOP (Improved Strength Pareto Evolutionary Algorithm Safety System Optimization Procedure) [Borisevic and Bartlett, 2007(1)]. The schematic structure of the program is shown in figure 6.1.

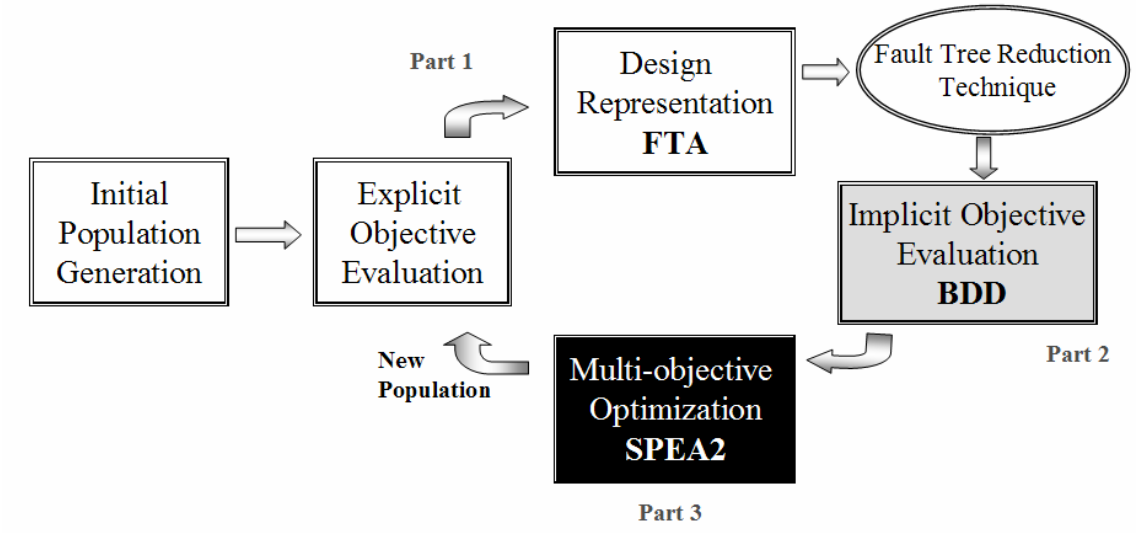


Figure 6.1 ISPEASSOP Schematic

There are three main parts of the ISPEASSOP program. In figure 6.1 the program part one, two and three components are coloured in white, grey and black respectively. Part one is responsible for the HIPS structure and explicit objective evaluation, part two is responsible for analysis using the Binary Decision Diagram method which calculates the HIPS unavailability and spurious trip frequency, and part three is an implemented SPEA2 algorithm for the HIPS optimisation. All these parts are discussed in detail in sections 6.2-6.4.

6.2 Program Part One Structure

The first part of the program produces the HIPS design architecture. The main steps of this part of the program are shown in figure 6.2. They can be also divided in two subgroups: the first is responsible for the population and data formation. This subgroup steps are: the generation of the initial population, parameter evaluation for each population individual and the regeneration of the infeasible region of parameter values, MDT and Cost parameter evaluation. The second subgroup is responsible for the HIPS structure modification according to each possible design. These steps involve: the setting of relevant house events and the HIPS fault tree reduction for each design. All mentioned steps are discussed in detail in sections 6.2.1 and 6.2.2.

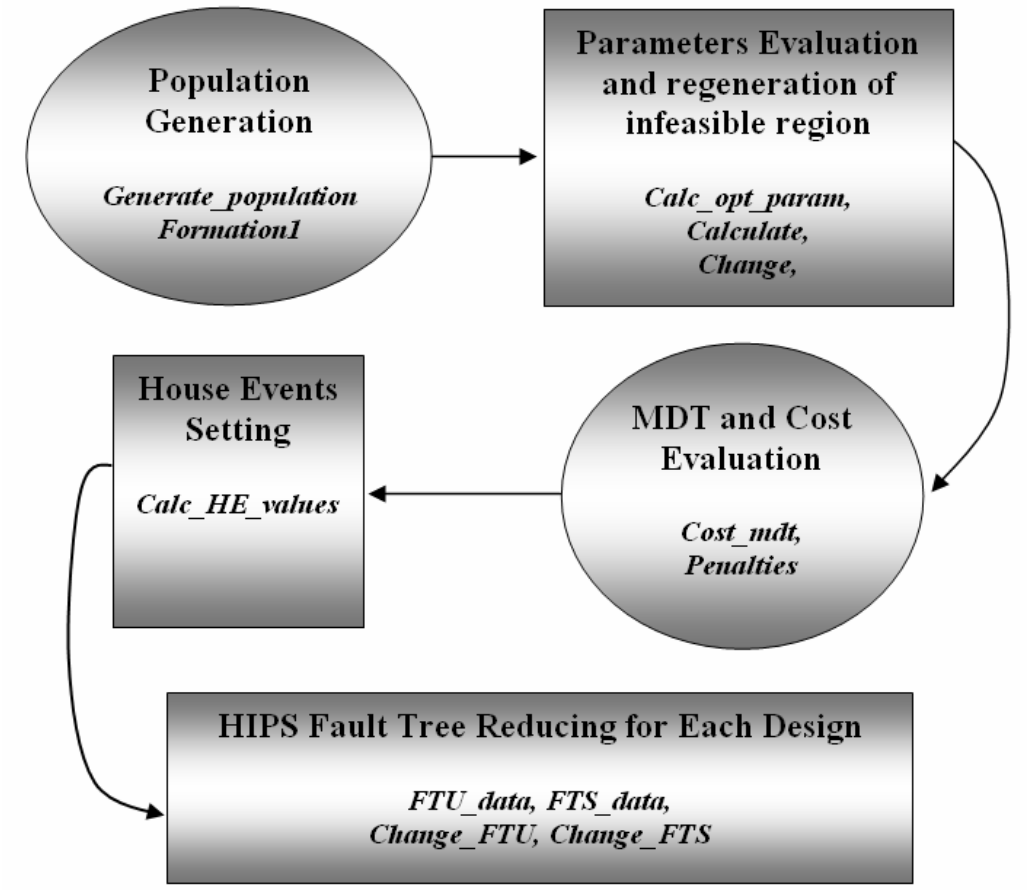


Figure 6.2 Program Part 1 Schematic

6.2.1 Coding and Initialisation

The main subroutines of the coding and initialisation part of the program are: *Generate_population, Formation1, Calc_opt_param, Cost_mdt* and *Penalties.* The subroutine *Generate_population* generates the initial random population. The number of strings for the initial population for a problem is not defined, thus, based on work carried out by Pattison [Pattison, 1999], initial research has used 20. Each string represents a particular system design depending on the values assigned to each of its 10 parameters (Table 5.1), where each parameter is calculated according to the binary coding system described in section 4.2. A review of these parameters and binary coding is given in table 6.1.

Table 6.1 HIPS Design Parameters and Binary Coding

Parameter	Description	Value	Binary Coding
$\theta_1,\ \theta_2$	Inspection intervals for subsystems 1 and 2	1week – 2 years	7 bits each
V	Valve type	1 or 2	1 bit
P	Pressure transmitter type	1 or 2	1 bit
$N_1,\ N_2$	Number of pressure transmitters fitted in subsystem 1 and 2	0 - 4	3 bits
$K_1,\ K_2$	Number of pressure transmitters required to trip (activate) for subsystem 1 and 2	0 - $N_1,\ N_2$	3 bits each
E	Number of ESD valves fitted	0, 1, 2	2 bits
H	Number of HIPS valves fitted	0, 1, 2	2 bits

Each parameter must be allocated a particular length of the string (Table 6.1), i.e. a particular number of bits, in order to accommodate the largest possible value in binary form. For example, the parameters governing the maintenance test interval for subsystems 1 and 2, θ_1 and θ_2 respectively, require 14 bits (7 bits each) of the total string to accommodate the maximum time span of 104 weeks each. In total, each string representing all design variables is 32 bits in length. It can be interpreted as a set of concatenated integers in binary form, as shown in figure 6.3.

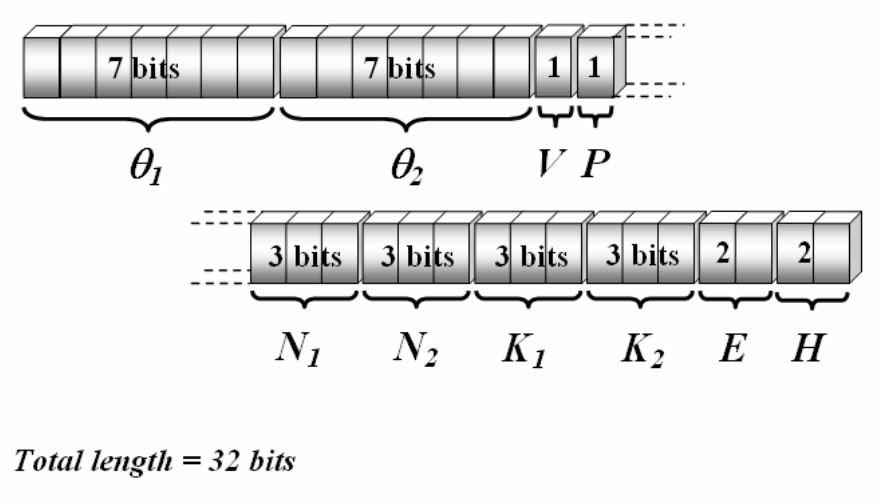


Figure 6.3 Binary Representation of Solution String

Subroutine *Calc_opt_param* calculates the numerical values for each of the 10 parameters from the initialized population. Procedures *Calculate*, *Change0*, *Change1* and *Change2* are executed during the *Calc_opt_param* subroutine. The decoding of the initialised binary coding system for each parameter is implemented in the procedure *Calculate.* For ease of computation, the order of parameters on the string was chosen as shown in figure 6.3. Each string parameter follows directly from the last, leaving no gaps. However, the restricted range of each parameter is not necessarily equivalent to its corresponding binary range defined by the allocated bit field length. For example, N_1 is the parameter governing the number of pressure transmitters in subsystem 1. It can obtain values in the range from 0 to 4, therefore 3 bits are allocated to this parameter. However, random initialisation of these 3 bits will decode to a value in the range from 0 to 7. This can result in three infeasible values of the parameter N_1. The parameters affected are: the number of pressure transmitters fitted in subsystem 1 and 2 (N_1, N_2), the number of transmitters required to trip subsystem 1 and 2 (K_1, K_2), the number of ESD valves fitted (*E*) and the number of HIPS valves fitted (*H*).

This problem is overcome by implementing procedures *Change0, Change1* and *Change2*:

1) Procedure *Change0* changes the value of the parameter which can't be equal to zero or exceeds the allowed maximum. For example, if the parameter values are in the range from 1 to 4 then if the generated parameter value is equal to 0, 5, 6

or 7 it is changed to 1, 2, 3 and 4 respectively to keep equal probabilities of occurrence for each parameter value. The same procedure checks and changes, if necessary, the values of any dependent parameters. For example, if the number of HIPS valves fitted, *H*, is equal to 0, the associated parameters: the number of pressure transmitters fitted in subsystem 2 (N_2), the number of pressure transmitters required to trip subsystem 2 (K_2) and the inspection interval for subsystem 2 (θ_2) are modified to zero.

2) Procedure *Change1* regenerates the value of the parameter which can be equal to zero, but can not exceed the fixed maximum. For example, the number of ESD valves fitted (*E*) and the number of HIPS valves fitted (*H*) can both obtain the value 0, but can not exceed the maximum value of 2.
3) Procedure *Change2* regenerates the values of parameters *E* (number of ESD valves) and *H* (number of HIPS valves) in the case when both parameters are equal to zero. The situation when both mentioned parameters are equal to zero is possible from the technical part of the HIPS design but it provides a trivial system design, which is not a subject of research interest.

The subroutine *Cost_mdt* calculates the cost and maintenance down time (MDT) for each potential HIPS design. The calculations are made according to equations 5.4 and 5.8. The subroutine *Formation1* is used only at the initial step of the program. Its main purpose is to format the initial population only of strings with cost less than 1000 units and MDT less than 130 hours. These limitations can be controlled only in the initial population. In the later steps of the optimization process genetic operators can produce new strings with parameters exceeding this limitation. In this case, the subroutine *Penalties* implements cost and MDT penalty formulas (equations 5.9 and 5.10).

6.2.2 HIPS Design Construction

One fault tree has been constructed to represent all system designs for the HIPS unavailability and one for spurious trip frequency, where house events have been added to allow for different options to be ‘turned on’ or ‘off’ (shown in appendix *A*). Therefore to develop the fault tree structures for each design, the values for each house event need

to be established. The main subroutines of the HIPS design construction part are: *House_event_data, Calc_HE_values, FTU_data, FTS_data* and *FT_reduction.* The subroutine *House_event_data* formats the array of HIPS house event names and uses the subroutine calc*_HE_values* which calculates the values of the relevant house events, i.e. 1 or 0, for each string according to the HIPS parameter values. Each HIPS design depends on the values of 40 house events. The meanings of the house events and their dependence on the parameter values are represented in tables A.1-A.4 (appendix *A*).

Subroutines *FTU-data* and *FTS-data* compose fault tree structures for the HIPS unavailability and spurious trip respectively. These structures are stored in the text files "*uft.txt*" and "*sft.txt*" (appendix C). Both files have a similar row-column structure. Each row represents a gate within the fault tree and the columns are related to:

- gate name,
- type of the gate (*AND/OR*),
- number of input gates,
- number of input events,
- number of house events (0 if there is no house events, otherwise 1),
- value of the gate (created only for computing purpose, initially considered to be equal to 2),
- array of input gates,
- array of input events.

Consider the example fault tree shown in figure 6.4.

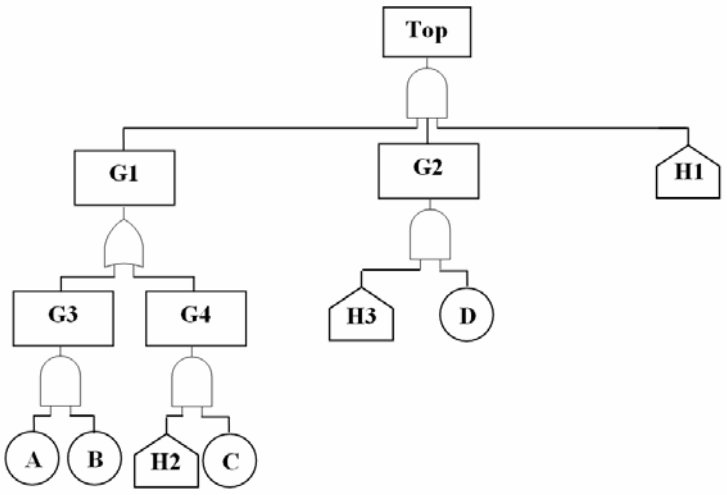


Figure 6.4 Example Fault Tree

Therefore, the initial "*example.txt*" file structure is:

Top	and	2	1	1	2	G1	G2	H1
G1	or	2	0	0	2	G3	G4	
G2	and	0	2	1	2	H3	D	
G3	and	0	2	0	2	A	B	
G4	and	0	2	1	2	H2	C	

The top event (Top) has gate type 'and', two input gates (*G1* and *G2*) and one input event (*H1*), which is a house event. The input gate *G1* has gate type 'or' and two input gates (*G3* and *G4*). The input gate *G2* is of 'and' type and has two input events (*H3* and *D*), where the input event *H3* is a house event and *D* is the basic event. The input gate *G3* is of 'and' type and has two basic input events (*A* and *B*). The last input gate, *G4*, has 'and' type and two input events: a basic event *C* and a house event *H2*.

The same subroutines also compose event data. The information about the events is stored in data files "ued.txt" and "sed.txt" for the HIPS unavailability and spurious trip respectively. The row-column structure of these files is:

- event name,
- failure rate (λ),
- mean time to repair (τ).

Subroutines *Change_FTU* and *Change_FTS* change the structure of HIPS unavailability and HIPS spurious trip fault trees respectively according to the values of the corresponding house events. A specialized fault tree reducing algorithm has been developed and implemented in subroutine *FT_Reduction* which contains the loop of subroutines *Reduce1*, *Reduce2* and *Reduce3*. It works from the bottom to the top of the fault tree. During this algorithm each gate is assigned a value of 0 or 1 depending on the gates structure. If the gate is 'activated' its value is set to 1, otherwise it is set to 0. As a result, all gates with a value of 0 are deleted from the fault tree. To illustrate the reducing process consider the general gate with house event structure as shown in figure 6.5, where $g_1,...,g_n$ are the input gates, h is the house event, $e_1,...,e_m$ the input events and $n,m \in Z_+$.

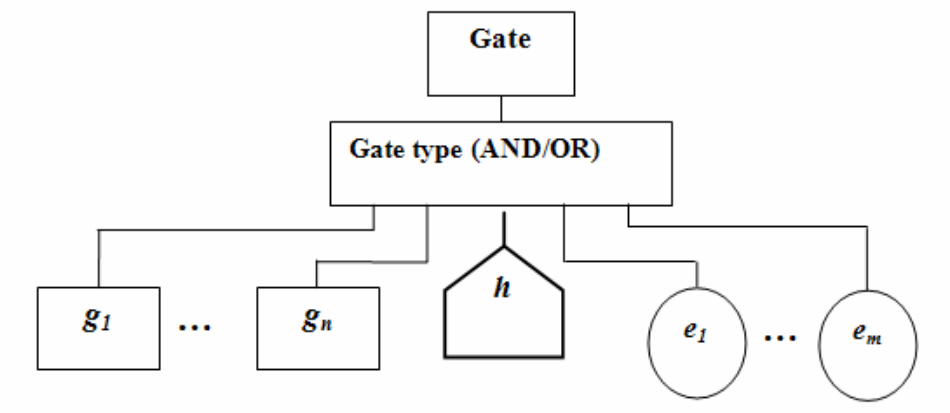


Figure 6.5 Gate with House Event Structure

The reducing algorithm was developed for both 'AND' and 'OR' gate types (Tables 6.2-6.3). For example, consider the 'AND' gate associated with the house event. Then the logic of the reducing algorithm can be described as follows: if the value of the house event is FALSE, the value of the analysed gate 'Gate' is automatically set to zero, hence this gate as a part of the fault tree is switched off. If the value of the house event is TRUE, then the following reduction depends on the number of input gates. If there are no input gates but:

- there is only one input event (different from the house event) then the 'Gate' is replaced by that event.
- there are several input events then the value of the 'Gate' is set to one, the house event is deleted from the line.

On the other hand, if there are input gates, the possible combinations are as follows:

- The 'Gate' value is set to zero if at least one input gate, $g_1,...,g_n$, value is equal to zero.
- If all input gates, $g_1,...,g_n$, values are equal to one then:
    1. if there is only one input gate and no input events, the 'Gate' is replaced by that input gate;
    2. otherwise, the 'Gate' value is set to one, the house event is deleted.

Similar logic has been implemented in the reduction of the 'OR' gate.

Table 6.2 Fault Tree Reducing Algorithm for AND gate

<table>
<tr><th colspan="6">Gate type is ‘AND’</th></tr>
<tr><th>‘FALSE’</th><th colspan="5">The house event’s value is ‘TRUE’</th></tr>
<tr><td rowspan="3">Gate value is set to 0</td><td colspan="2">There are no input gates
$n = 0$</td><td colspan="3">There are input gates
$n > 0$</td></tr>
<tr><td>There is one input event
$m = 1$</td><td>There are more than 1 input events
$m > 1$</td><td colspan="2">All input gates values are equal to 1</td><td>At least one input gate’s value is equal to 0</td></tr>
<tr><td>Gate is replacing by the input event (e_1)</td><td>Gate value is set to 1</td><td>There are no input events
$n + m = 1$</td><td>There are at least one input event
$n + m > 1$</td><td rowspan="2">Gate value is set to 0</td></tr>
<tr><td colspan="2"></td><td>Gate is replacing by the input gate (g_1)</td><td>Gate value is set to 1</td></tr>
</table>

Table 6.3 Fault Tree Reducing Algorithm for ‘OR’ Gate

<table>
<tr><th colspan="6">Gate type is ‘OR’</th></tr>
<tr><th>The house event’s (h) value is ‘TRUE’</th><th colspan="5">The house event’s (h) value is ‘FALSE’</th></tr>
<tr><td rowspan="3">Gate value is set to 1</td><td colspan="2">There are no input gates
$n = 0$</td><td colspan="3">There are input gates
$n > 0$</td></tr>
<tr><td>There is one input event
$m = 1$</td><td>There are more than 1 input events
$m > 1$</td><td colspan="2">At least one input gate’s value is equal to 1</td><td>All input gates values are equal to 0</td></tr>
<tr><td>Gate is replacing by the input event (e_1)</td><td>Gate value is set to 1</td><td>There are no input events
$n + m = 1$</td><td>There are at least one input event
$n + m > 1$</td><td rowspan="2">Gate value is set to 0</td></tr>
<tr><td colspan="2"></td><td>Gate is replacing by the input gate (g_1)</td><td>Gate value is set to 1</td></tr>
</table>

Subroutines *Reduce1*, *Reduce2* and *Reduce3* are the main fault tree reduction process subroutines. The subroutine *Reduce1* evaluates the fault tree gates with the simplest structure: the gate without house events or the gate with the house event whose value is 'FALSE'. The subroutine *Reduce2* evaluates all other gates, i.e. the gates with house event, whose value is 'TRUE'. The evaluation is similar to the algorithm described in tables 6.2 and 6.3. Finally, the subroutine *Reduce3* defines and deletes all gates which are not involved in the fault tree after reduction. All three reduction subroutines use four procedures: *Restructure, Restructure1*, *Restructure2* and *Restructure3*. The first procedure deletes the house event from the relevant information string and rearranges event data. The procedure *Restructure* is used by all other procedures. The procedure *Restructure1* is responsible for the fault tree reduction when the gate is replaced by the input event. The procedure *Restructure2* is used to replace the gate by the input gate. Finally, the procedure *Restructure3* deletes from the fault tree gates with the value 0.

Consider the example fault tree from figure 6.5 with the house event values:

H1 = "TRUE",

H2 = "FALSE",

H3 = "TRUE".

The reducing algorithm works from the bottom (gates *G3* and *G4*) to the top of the fault tree (Figure 6.6). The gate *G4*, highlighted by the dashed circle, has an input house event *H2*.

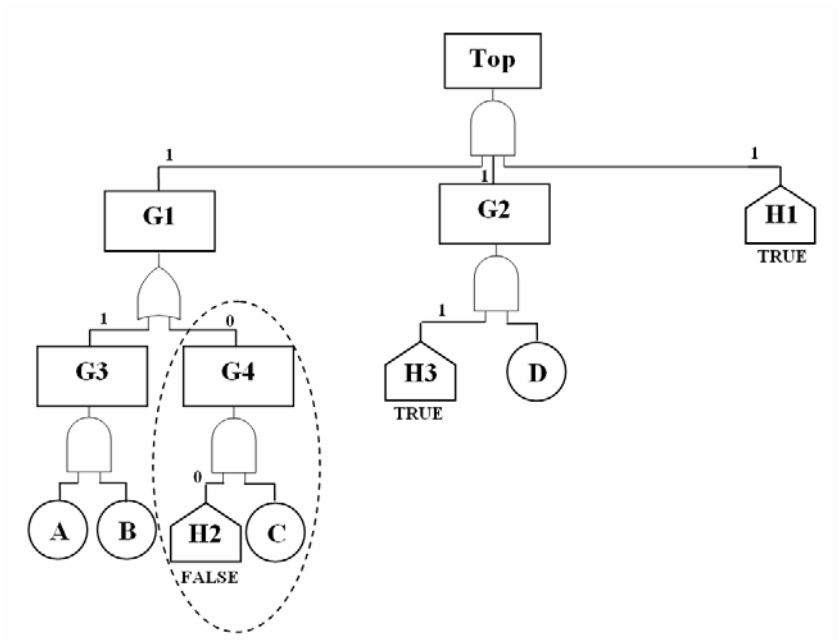


Figure 6.6 Fault Tree Reduction Example

The reduction process starts from the gate *G3*. The gate *G3* is of type 'AND'. It has 2 input events (*A* and *B*). Since both input events are basic and there are no house events, the gate *G3* is active and its value changes from 2 to 1. Therefore, the data file at this step is:

Top	and	2	1	1	2	G1	G2	H1
G1	or	2	0	0	2	G3	G4	
G2	and	0	2	1	2	H3	D	
G3	and	0	2	0	**1**	A	B	
G4	and	0	2	1	2	H2	C	

The gate *G4* is also an 'AND' gate, which has one input event (*C*) and a house event (*H2*). The value of *H2* is false, therefore, the value of *G4* is set to zero and is deleted from the fault tree structure. Hence, only one input gate is left in gate *G1*. According to table 6.3, this gate is replaced by the input gate *G3*. Therefore, the data file after reduction is:

Top	and	2	1	1	2	**G3**	G2	H1
~~G1~~	~~or~~	~~2~~	~~0~~	~~0~~	~~1~~	~~G3~~	~~**G4**~~	
G2	and	0	2	1	2	H3	D	
G3	and	0	2	0	1	A	B	
~~G4~~	~~and~~	~~0~~	~~2~~	~~1~~	~~**0**~~	~~H2~~	~~C~~	

Considering gate *G2* next, it has the house event, which is true, and one input event, *D*. Therefore, *G2* is replaced by its input event (Table 6.2). The data file at this step is:

Top	and	2	1	1	2	G3	**D**	H1
~~G2~~	~~and~~	~~0~~	~~2~~	~~1~~	~~**1**~~	~~H3~~	~~D~~	
G3	and	0	2	0	1	A	B	

Since the value of the house event *H1* is true the obtained fault tree after reduction is as shown in figure 6.7.

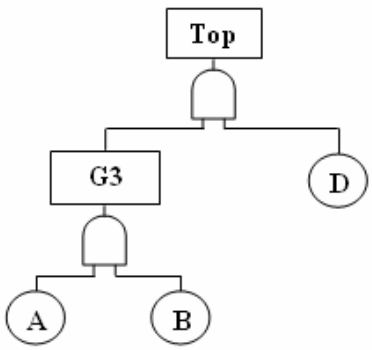


Figure 6.7 Example Fault Tree after Reduction

Therefore, the final structure of the data file is:

Top	and	2	1	0	1	G3	D
G3	and	0	2	0	1	A	B

6.3 Program Part Two Structure

The main goal of this part of the program is the fault tree unavailability and the spurious trip frequency calculation. For each HIPS design this task is achieved in four main steps, as shown in figure 6.8. The main routines within each step are also shown in figure 6.8. All steps are discussed in detail in sections 6.3.1- 6.3.2.

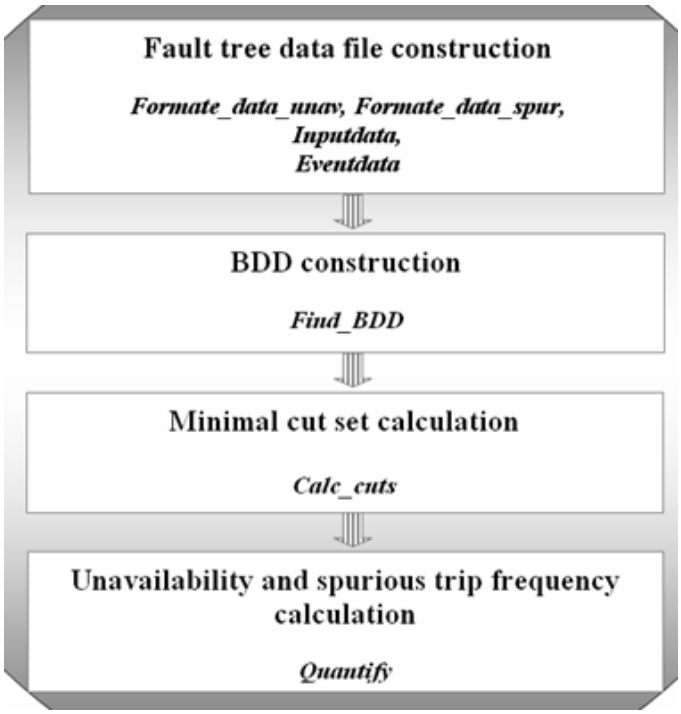


Figure 6.8 Program Part Two Schemata

6.3.1 Data File Structures

The program constructs the fault tree data files for both the HIPS unavailability and the spurious trip frequency according to each HIPS potential design. Subroutines *Formate_data_unav* and *Formate_data_spur* construct data files by using the reduced fault tree designs obtained in subroutines *Change_FTU* and *Change_FTS* and setting

the relevant failure data (failure rate and mean time to repair) for each component of the fault tree (data files "*ued.txt*" and "*sed.txt*" for the HIPS unavailability and spurious trip respectively).

The program generates four text data files for each HIPS design. Text file:

1) *unav.gsq* contains the general fault tree for the HIPS unavailability,
2) *unav.aqd* contains event data for the HIPS unavailability,
3) *spur.gsq* contains the general fault tree for the HIPS spurious trip,
4) *spur.aqd* contains event data for the HIPS spurious trip.

To illustrate the data file construction process consider an example reduced fault tree, as shown in figure 6.9, where the significant parameters, Q related to the system component unavailability and w to spurious failure frequency, are stated beneath each basic event.

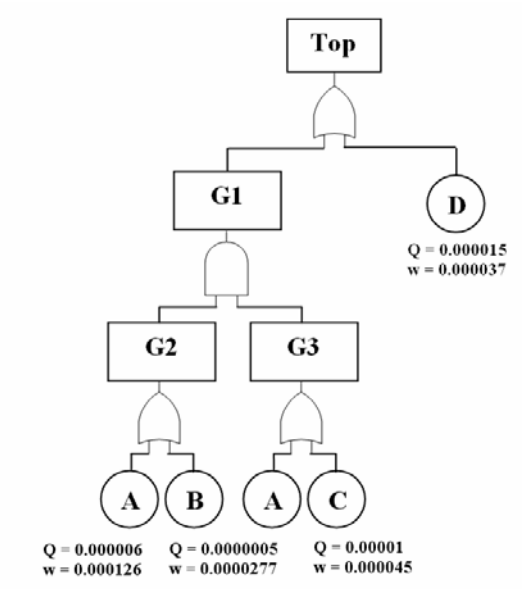


Figure 6.9 Fault Tree 'Example'

The "*.gsq*" file structure is arranged in rows and columns. Each row represents a gate within the tree and the columns relate to:

- gate name,
- type of gate (*AND/OR*),
- number of input gates,
- number of input events,
- array of input gates,
- array of input events.

Therefore, the '*Example.gsq*' fault tree structure file for the example fault tree (Figure 6.9) is:

TOP	or	1	1	G1	D
G1	and	2	0	G2	G3
G2	or	0	2	A	B
G3	or	0	2	A	C

Hence, the top gate 'TOP' is of type 'OR'. It has one input gate (*G1*) and one input event (*D*).

The "*.aqd*" file has a row-column structure. Each row represents a particular event and the columns relate to the following event data:

- event name,
- unavailability of the event $Q = \lambda\left(\tau + \frac{\theta}{2}\right)$,

  where λ is the failure rate (Table 5.2),

  θ is an inspection interval (obtained within the program),

  τ is the mean time to repair (Table 5.2),
- unconditional failure intensity $w = \lambda(1 - Q)$.

The unavailability and unconditional failure intensity calculation methods are chosen based on work by Pattison [Pattison, 1999]. Q and w are calculated within the program by using λ and τ stored in the data files "*ued.txt*" and "*sed.txt*" for the HIPS unavailability and spurious trip respectively. Hence, the '*Example.aqd*' event data file for the example fault tree (Figure 6.9) is:

A	0.000006	0.000126
B	0.0000005	0.0000277
C	0.00001	0.000045
D	0.000015	0.000037

Therefore, the element *A* has unavailability 0.000006 and unconditional failure intensity 0.000126.

Subroutines *Inputdata* and *Eventdata* read the fault tree data into arrays within the program. The structure of these data arrays is similar to the structure of the *".gsq*" and "*.aqd*" files.

6.3.2 Fault Tree Analysis - BDD Construction

To analyse the fault tree the latest binary decision diagram (BDD) technique has been used. Construction and analysis of the BDD is the second step of the program. The subroutine *Find_BDD* is a code for the If-Then-Else (**ite**) binary decision diagram construction method developed by Rauzy [Rauzy, 1993] (section 2.5.1). This subroutine is an improved version of the C++ program code developed by Remenyte-Prescott [Remenyte-Prescott, 2007]. The BDD structure is then used to obtain the cut sets of the fault tree (the subroutine *Calc_Cuts*). Top event and unconditional system failure intensity quantification is implemented in the subroutine *Quantify* using equations 2.14 – 2.20.

6.4 Program Part Three Structure

This part of the program is an implemented Improved Strength Pareto Optimization Approach (SPEA2), whose structure is shown in figure 6.10.

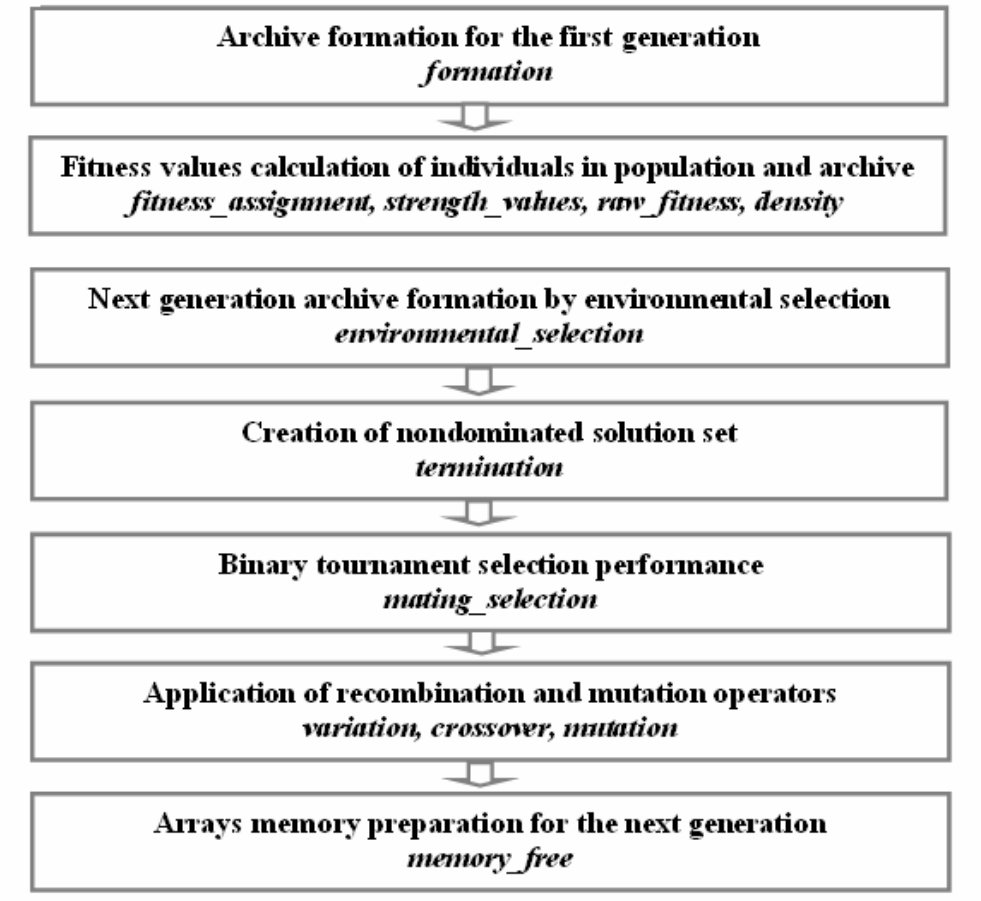


Figure 6.10 Program Part Three Structure

The algorithm (Figure 6.10) works through six steps, using the following inputs:

Input: N (population size) set to 20,

$\overline{N}$ (archive size) set to 20 (generally this variable can obtain values in the range from 1 to population size),

T (maximum number of generations) set to 100.

Output: A (resulting population) determined by the best solution strings set after the final generation.

There are six main steps of the optimization algorithm. Each step is described in detail after the summary of the whole optimization algorithm.

Step 1. *Initialization*: Generate an initial population P_0 and create the empty archive (external set) $\overline{P_0} = \theta$. Set the generation number $t = 0$.

Step 2. *Fitness assignment*: Calculate fitness values of individuals in P_t and $\overline{P_t}$.

Step 3. *Environmental selection*: Copy all nondominated individuals in P_t and $\overline{P_t}$ to $\overline{P_{t+1}}$. If size of $\overline{P_{t+1}}$ exceeds $\overline{N}$ then reduce $\overline{P_{t+1}}$ by means of the truncation operator, otherwise if size of $\overline{P_{t+1}}$ is less than $\overline{N}$ then fill $\overline{P_{t+1}}$ with dominated individuals in P_t and $\overline{P_t}$.

Step 4. *Termination*: If $t \geq T$ or another stopping criterion is satisfied then set A to the set of decision vectors represented by the nondominated individuals in $\overline{P_{t+1}}$. Stop.

Step 5. *Mating selection*: Perform binary tournament selection with replacement on $\overline{P_{t+1}}$ in order to fill the mating pool.

Step 6. *Variation*: Apply recombination and mutation operators to the mating pool and set P_{t+1} to the resulting population. Increment generation counter ($t = t + 1$) and go to *Step 2*.

The SPEA2 method requires an external memory (archive or Pareto set) at each iteration to keep the best solutions. According to the Environmental selection principle (section 4.4.6) the first generation archive $\overline{P}_1$ is equal to the initial population P_0. After

the initialization step the subroutine *formation* creates the first generation population by applying recombination and mutation operators to the initial population and formats all the necessary data arrays (cost, mdt, system spurious trip and system unavailability) for both the population and the archive before the first generation.

The fitness assignment is the second step of the optimization algorithm. It is implemented in the subroutine *fitness_assignment*. One of the main goals of this routine is to divide the population of strings into dominated and nondominated solution groups according to the following rules: since all system parameter values should be minimized and the MDT value should be close to the limit of 130, the string *a* dominates the sting *b* ($a \phi b$) if all *a* parameter values are equal to or smaller than *b* parameter values and at least one of parameter *a* value is smaller than the respective *b* parameter value, i.e.:

$$a \phi b = (f_i(a) \le f_i(b), \;\; i = 1,...,n,) \wedge (\exists k : f_k(a) < f_k(b)), \;\; k \in [1,n].$$

The string *a* is nondominated if there is no string in the population which dominates *a*. To avoid the situation that individuals dominated by the same archive members have identical fitness values, for each individual both dominating and dominated solutions are taken into account (section 4.4.2). In detail, each individual *i* in the archive $\overline{P}_t$ and the population P_t is assigned a strength value $S(i)$, representing the number of solutions it dominates (the subroutine *strength_values*), which is calculated as [Zitzler et al. (2), 2001]

$$S(i) = \left| \left\{ j \mid j \in P_t + \overline{P}_t \wedge i \phi j \right\} \right|,$$

where | | denotes the cardinality of a set, + stands for multiset union and the symbol ϕ corresponds to the Pareto dominance relation (Definition 4.1). Consider the example population and archive at the *t*-th iteration (Table 6.4):

Table 6.4 Example Population and Archive

String No.	Population, P_t				String No.	Archive, $\overline{P}_t$			
	Cost	**MDT**	***Q***	F_{sys}		**Cost**	**MDT**	***Q***	F_{sys}
1P	890	125.4	3.7e-4	0.819	**1A**	586	128.8	1e-6	0.418
2P	520	129.7	1.2e-7	0.112	**2A**	700	129.6	1.2e-6	0.215
3P	970	122.8	2.6e-5	0.414	**3A**	620	129.5	3.5e-6	0.318

Therefore, the strength value of the first string in the example population *S*(1P) is equal to zero, since there is not a string in the population or archive which would be dominated by this string. The second string of the population (2P) has the lowest cost, system unavailability and spurious trip values, and its MDT value is closest to the limit of 130 hours. Hence, it dominates all other strings both in the population and archive and its strength value *S*(2P) is set to 5. The third string (3P) dominates the string 2P in MDT, system unavailability and spurious trip values, however the total system cost is larger, than the other strings, therefore, the strength *S*(3P) is set to zero. The first string in the archive (1A) dominates only the string 1P, hence its strength value is set to 1. Strings 2A and 3A dominate strings 1P and 3P, therefore, *S*(2A) and *S*(3A) are equal to 2 (Table 6.5).

Table 6.5 Strength Values for Example Problem

String No.	Population, P_t	String No.	Archive, $\overline{P}_t$
	Strength value		Strength value
1P	0	**1A**	1
2P	5	**2A**	2
3P	0	**3A**	2

On the basis of the *S* values, the raw fitness $R(i)$ (the subroutine *raw_fitness*) of an individual *i* is calculated as:

$$R(i) = \sum_{j \in P_t + \overline{P}_t,\, j \phi i} S(j).$$

Hence the raw fitness is determined by the strengths of its dominators in both the archive and population. It is important to note that fitness is to be minimized, i.e., $R(i) = 0$ corresponds to a nondominated individual, while a high $R(i)$ value means that *i* is dominated by many individuals. Considering the example above, the string 1P is dominated by strings 2P, 1A, 2A and 3A. Hence, its raw fitness *R*(1P) is equal to the sum of four strength values:

$$R(1P) = S(2P) + S(1A) + S(2A) + S(3A) = 5 + 1 + 2 + 2 = 10.$$

The string 2P dominates all other strings, therefore its raw fitness is set to zero. The string 3P is dominated by strings 2P, 2A and 3A. Hence, *R*(3P) is equal to 9. Strings 1A,

2A and 3A are all dominated only by 2P, therefore, $R(1A) = R(2A) = R(3A) = 5$. Table 6.6 represents the results summary of raw fitness calculations for the example problem.

Table 6.6 Raw Fitness Results for Example Problem

String No.	Population, P_t	String No.	Archive, $\overline{P}_t$
	Raw Fitness, R_i		Raw Fitness, R_i
1P	10	**1A**	5
2P	0	**2A**	5
3P	9	**3A**	5

Although the raw fitness assignment provides a sort of niching mechanism based on the concept of Pareto dominance, it may fail when most individuals do not dominate each other. Hence, additional information is incorporated to discriminate between individuals having identical raw fitness values. The density estimation technique used in SPEA2 is an adaptation of the *k*-th nearest neighbour method [Silverman, 1986], where the density at any point is a decreasing function of the distance to the *k*-th nearest data point. In this problem the inverse of the distance to the *k*-th nearest neighbour is taken as a density estimate, i.e. for each individual *i* the distances to all individuals *j* in the archive and population are calculated as

$$\sigma_{ij} = \sqrt{(C(i)-C(j))^2 + (MDT(i)-MDT(j))^2 + (Q(i)-Q(j))^2 + (Fsys(i)-Fsys(j))^2},$$

where $C(i)$ is the cost of the *i*-th design, $Q(i)$ is the *i*-th designs system unavailability, *j* is from the interval [1,.., *n*], where *n* is the total number of individuals in the population and archive, with the condition that $i \neq j$. Obtained distances are stored in a list or matrix. For the example problem the distance matrix is shown in table 6.7. It is important to notice that the matrix is symmetrical.

Table 6.7 The Distance Matrix for Example Problem

i	**Distance values, σ_{ij}**					
1P	0	370.0257	80.04326	304.0193	190.0474	270.0316
2P	**370.0257**	0	450.053	66.00685	180.0001	100.0004
3P	**80.04326**	**450.053**	0	384.0469	270.0857	350.0641
1A	**304.0193**	**66.00685**	**384.0469**	0	114.003	34.00735
2A	**190.0474**	**180.0001**	**270.0857**	**114.003**	0	80.00013
3A	**270.0316**	**100.0004**	**350.0641**	**34.00735**	**80.00013**	0
j	**1P**	**2P**	**3P**	**1A**	**2A**	**3A**

After sorting the list in increasing order, the *k*-th element gives the distance sought, denoted as σ_i^k, where *k* is equal to the square root of the sample size:

$$k = \sqrt{N + \overline{N}},$$

thus, for the HIPS *k* is approximately equal to 6, since $N = \overline{N} = 20$. For the example problem *k* is equal to 2. The distance matrix (Table 6.7) after sorting in increasing order and reduction of distances between the same elements is given in table 6.8.

Table 6.8 The Distance Matrix after Sorting and Reduction

K	**Distance values, σ_i^k**					
1	80.04326	66.00685	80.04326	34.00735	80.00013	34.00735
2	**190.0474**	**100.0004**	**270.0857**	**66.00685**	**114.003**	**80.00013**
3	270.0316	180.0001	350.0641	114.003	180.0001	100.0004
4	304.0193	370.0257	384.0469	304.0193	190.0474	270.0316
5	370.0257	450.053	450.053	384.0469	270.0857	350.0641
I	**1P**	**2P**	**3P**	**1A**	**2A**	**3A**

Afterwards, the density $D(i)$ corresponding to i is defined by

$$D(i) = \frac{1}{\sigma_i^k + 2}.$$

Hence, the density values for the example problem are calculated with the distance values (σ_i^k) at $k = 2$ (Table 6.8). The obtained density results are shown in table 6.9.

Table 6.9 Density Values for Example Problem

String No.	**Population, P_t** **Density, D_i**	**String No.**	**Archive, $\overline{P}_t$** **Density, D_i**
1P	0.005207	**1A**	0.014704
2P	0.009804	**2A**	0.00862
3P	0.003675	**3A**	0.012195

Density value calculation is implemented in the subroutine *density*. In the denominator, two is added to ensure that its value is greater than zero. Finally, adding $D(i)$ to the raw fitness value $R(i)$ of an individual *i* yields its fitness $F(i)$:

$$F(i) = R(i) + D(i).$$

Therefore, the fitness values of the strings in the example problem are shown in table 6.10.

Table 6.10 Fitness Value for Example Problem

String No.	Population, P_t Fitness, F_i	String No.	Archive, $\overline{P}_t$ Fitness, F_i
1P	10.005207	1A	5.014704
2P	**0.009804**	2A	5.00862
3P	9.003675	3A	5.012195

The third step of the optimization algorithm is the environmental selection, which is implemented in the subroutine *environmental_selection*. During this step the first goal is to copy all nondominated individuals, i.e., those which have a fitness lower than one, from the archive and population to the archive of the next generation. In the example problem only the string 2P satisfies this condition and, hence, is a nondominated string. Therefore, the next generation archive ($\overline{P}_{t+1}$) is:

$$\overline{P}_{t+1} = \{2P\}\,.$$

If the total number of nondominated solutions is equal to the next generation archive size then the environmental selection step is completed. Otherwise, there can be two situations: either the archive is too small (in the example problem the archive size is fixed and equal to 3, however there is only one nondominated string, 2P) or too large. In the first case, the best dominated individuals in the previous archive and population are copied to the new archive. This is implemented by sorting the multiset $P_t + \overline{P}_t$ according to the fitness values and picking the best individuals from it. The sorted set of P_t and $\overline{P}_t$ without nondominated components for the example problem is shown in table 6.11.

Table 6.11 Sorted Set of P_t and $\overline{P}_t$

No.	String	Fitness, *F*
1	**2A**	**5.00862**
2	**3A**	**5.012195**
3	1A	5.014704
4	3P	9.003675
5	1P	10.005207

Therefore, the next generation archive $\overline{P}_{t+1}$ is extended with the fittest dominated strings 2A and 3A. Hence

$$\overline{P}_{t+1} = \{2P,\ 2A,\ 3A\}\ .$$

In the second case, when the size of the current nondominated set exceeds the fixed archive size, an archive truncation procedure is invoked which iteratively removes individuals from the nondominated set until its size is equal to the archive size. The truncation procedure works according to the following rule: the individual which has the minimum distance to another individual is chosen at each stage; if there are several individuals with minimum distance the tie is broken by considering the second smallest distance and so on.

The subroutine *termination* checks whether the maximum number of generations is reached or another stopping criterion is satisfied. If yes, i.e. $t = T$, the resulting set A is equal to the set of nondominated individuals in archive $\overline{P}_{t+1}$ and the optimization algorithm stops. For the example problem if t is the last generation, then $A = \{2P\}$.

The mating selection, implemented in the subroutine *mating_selection,* performs binary tournament selection with replacement on $\overline{P}_{t+1}$ in order to fill the mating pool. The selection is done by using the following procedure:

d) Random selection of two individuals of the archive.

e) Copying the one with the lower fitness value to the mating pool.

f) If the mating pool is full, then the algorithm stops, else everything repeats from step (a).

The final step of the optimization algorithm is the variation step, implemented in the subroutine *variation*. During this step the crossover and mutation operators are applied to the mating pool.

Crossover is a genetic operator producing new strings, which have some parts of both parent's genetic material. The SPEA2 uses the simplest form, single-point crossover. This process is carried out in the routine *crossover*. Crossover considers every other individual in the population and archive. Consider string j. A random number between 0

and 1 is generated. If this number lies below the crossover rate the mating process proceeds and string *j* is mated with string *j+1* about a randomly generated crossover point. In the other case, both strings remain unchanged and consideration is given to string *j+2*. As the population and the archive are determined using a probabilistic selection method, mating pairs are in effect randomly chosen.

The mutation process is implemented in the routine *mutation*. This is a random process where one allele of a gene is replaced by another to produce a new genetic structure. In the SPEA2 mutation is applied with uniform probability to the entire population of strings. Therefore, it is possible that a given binary string may be mutated at more than one point. Mutation occurs with a probability determined by the mutation rate.

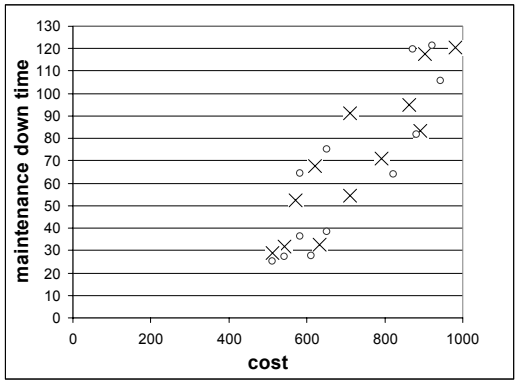
After the variation step the modified mating pool is set to the resulting population P_{t+1}. The generation counter is increased by one. The subroutine *memory_free* formats all necessary data arrays (archive, population arrays and optimization parameters array) for the next generation and the process repeats from the second optimization step.

6.5 Results

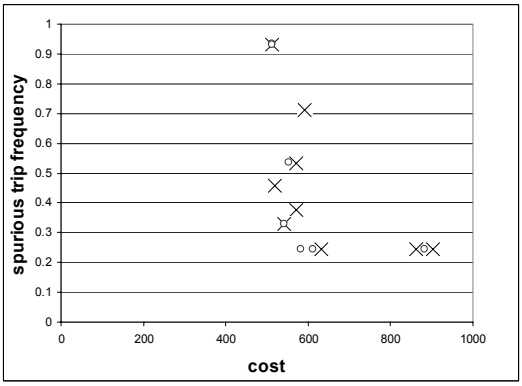
6.5.1 Optimization Schemes

Two different optimization schemes have been implemented to tailor the algorithm parameters for the HIPS system in order to evaluate the one that leads faster to the global optimal solution. In the first scheme a single population of 20 strings have been generated and run through 3000 generations with the crossover and mutation rates equal to 0.7 and 0.01 respectively. The second scheme was based on thirty different initial populations with only 100 generations for each run of the ISPEASSOP with the same crossover and mutation rates. The first scheme resulted in a single Pareto set of nondominated HIPS design options, on the other hand the second scheme gave 30 sets. A Pareto set obtained from the 20th run in the second scheme consisted of a larger number of nondominated solutions by most optimization parameter values and, therefore, has been chosen for comparison with the first optimization scheme. Due to a

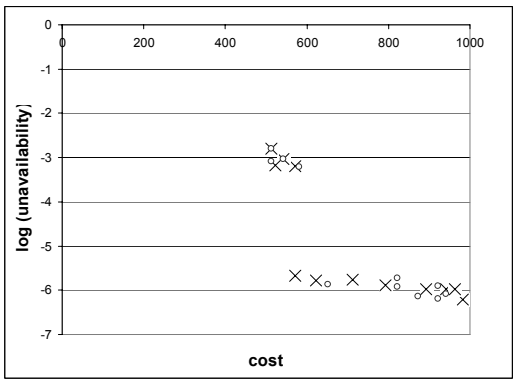
large number of optimization parameters the analysis of these results have been carried out for each group of two. Figures 6.11a - f show the comparison of resulting Pareto fronts in two dimensional space (where scheme 1 points are represented using a × and a ° for scheme 2).



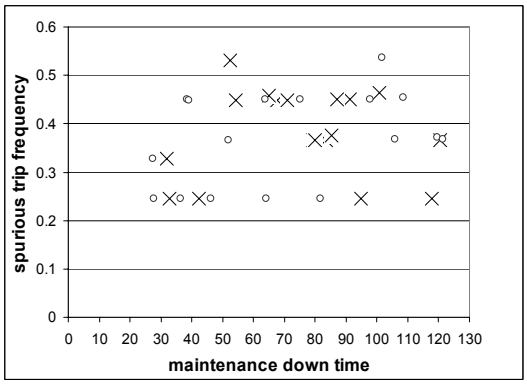
a) Cost and MDT Pareto front



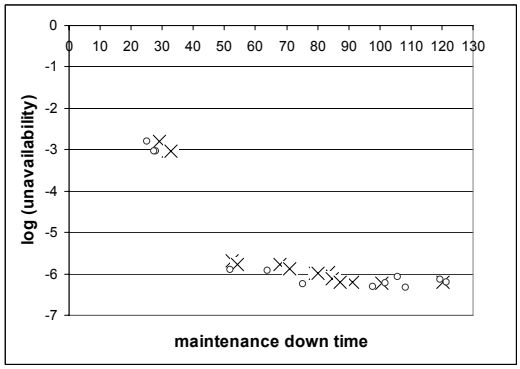
b) Cost and Fsys Pareto front



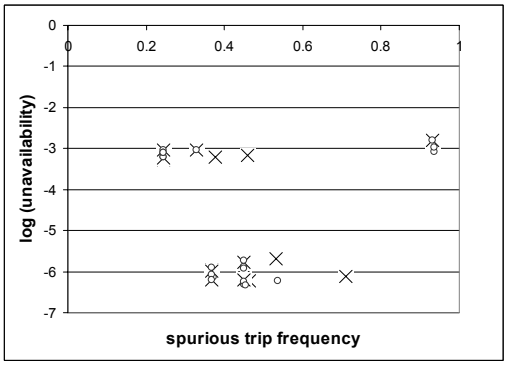
c) Qsys and Cost Pareto front



d) Fsys and MDT Pareto front



e) Qsys and MDT Pareto front



f) Qsys and Fsys Pareto front

Figure 6.11 Pareto Front in Two Dimensional Space

All figure 6.11 plots show that both optimization schemes produced very similar Pareto fronts, however the front obtained by the first scheme produces a larger number of the

boundary points (design solutions) due to a larger number of generations. Observation of the data itself shows that the 1st scheme produces up to 4 additional non-dominated designs. Figure 6.11a indicates that the maintenance down time values are directly proportional to the system cost values. Therefore, a reduction of maintenance hours or personnel would result in a lower system cost. Figure 6.11b illustrates that with an increase in cost comes a decrease in spurious trip frequency, however at just below 600 cost units further increase does not improve the reduction in inappropriate action of the system. Figure 6.11c shows the relationship between cost and unavailability. A log scale has been used to represent the unavailability due to the large deviation in values produced. As cost is increased the effect is to improve the unavailability values. This can be explained by the fact that the main cost contributors would be either the additional safety equipment or the increased number of maintenance hours, both resulting in the system availability improvement. Similar to the higher system cost, higher maintenance down time values lead to smaller system unavailability (Figure 6.11e). Figure 6.11d shows that there is no clear relationship between spurious trip frequency and maintenance down time due to the nature of spurious failures. Figure 6.11f shows that there is again not a strong relationship between spurious trip and unavailability, however values of 0.45 spurious trip occurrences per year produces the smallest unavailability.

6.5.2 Exhaustive Search Results

For a system required to work on demand the system unavailability is one of the most important optimization criterion. Therefore in order to check the performance of the schemes from section 6.5.1 in terms of system unavailability, an exhaustive search has been produced [Riauke and Bartlett, 2009 (1)]. Given the total number of potential HIPS designs equals 232257600, the whole exhaustive search required almost 38 hours. Evidence showed a difference between the value of the unavailability obtainable with constraints removed. Figure 6.12 represents the comparison of the smallest actual (with constraints removed) and feasible (obtained from the design, with optimization parameters within their limitations) for system unavailability obtained during the search when considering maintenance test intervals for sub-system 1.

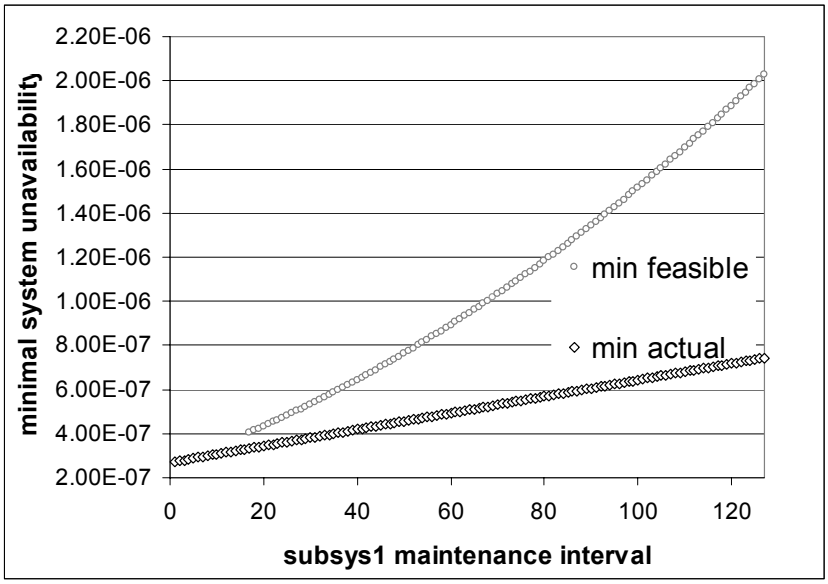


Figure 6.12 Comparison of the Actual and Feasible Minimal Qsys obtained by Exhaustive Search

As it might be seen from figure 6.12, the smallest actual system unavailability (*Qsys* = 2.884e-7) has been obtained in the first group ($\theta 1$ = 1 week), however the design corresponding to this value is infeasible due to a large value of maintenance down time (*MDT* = 2028 *h*). The value of the smallest system unavailability has the tendency to increase when the maintenance interval $\theta 1$ becomes larger. The smallest feasible unavailability value is *Qsys* = 4.051e-7. The difference between the actual and feasible values increases when the maintenance interval becomes larger.

The comparison of the best results obtained during experiments described in section 6.5.1 to the best HIPS design obtained by exhaustive search is shown in table 6.12. This table shows that both optimization schemes produced designs with unavailability values close to the one obtained by exhaustive search, however the second scheme resulted in a smaller value. Other optimization parameter values are similar in all cases. The first scheme produces smaller MDT (128.40 *h*) and system cost (632 units) values, on the other hand the second scheme resulted in smaller spurious trip frequency (0.45027 times). All three designs are very similar. In all cases the first subsystem has no ESD valves (*E*) and the subsystem 2 consists of only 1 HIPS valve (*H*). The dominated value for pressure transmitters fitted (*N1/N2*) and required (*K1/K2*) is 2. All designs are constructed of the first type of valve (*V*) and pressure transmitter (*P*) type. The maintenance interval for subsystem 1 ($\theta 1$) is close to 17, and the maintenance interval

$\theta 2$ varies from 60 to 127. Other optimization parameter values are also very similar for all three designs. Since the optimization technique searches for the optimal solutions in terms of four objectives, the likelihood of the resulting design with the smallest global unavailability is reduced. However, the obtained results prove the good performance of the developed tool, which produces a solution close to the global minimum in only 20 minutes (for 3000 system evaluations).

Table 6.12 Results Comparison

Subsystem		Exhaustive Search	1st Scheme	2nd Scheme
1	No. of ESD valves, E	0	0	0
	No. of PTs, N_1	2	2	3
	No. of PTs to trip, K_1	2	2	2
	Maintenance test interval, θ_1	17	19	18
2	No. of HIPS valves, H	1	1	1
	No. of PTs, N_2	2	2	3
	No. of PTs to trip, K_2	2	2	2
	Maintenance test interval, θ_2	127	60	93
Valve type (V)		1	1	1
PT type (P)		1	1	1
MDT		129.53	128.40	129.53
Cost		632	632	652
Spurious trip occurrence (F_{sys})		0.45045	0.45044	0.45027
System unavailability (Q_{sys})		4.051e-7	4.235e-7	4.143e-7

Looking at the unavailability values for each of the optimization schemes shows that in both experiments the majority of unavailability values are close to the feasible smallest value produced by the exhaustive search. The minimal unavailability values concentrate in the intervals [4.235e-7, 1e-6) and [4.143e-7, 6e-7) for the first and second schemes respectively. Despite the relatively small number of generations, the second optimization scheme provided larger diversity between potential HIPS designs and led to a smaller system unavailability.

6.5.3 Best ISPEASSOP Design

The ISPEASSOP was implemented with a generated population of 20 strings. A maximum of 100 generations was allowed along with a mutation rate of 0.01 and crossover rate 0.7. The first run of 100 generations of the ISPEASSOP gave the best

string set in generation 8 to 11. The fittest string occurs in the 8th generation (Table 6.13)

Table 6.13 The Best Design After the First Run of 100 Generations of the ISPEASSOP

Subsystem 1	No. of ESD valves	0
	No. of PTs	1
	No. of PTs to trip system	1
	MTI	25
Subsystem 2	No. of HIPS valves	1
	No. of PTs	3
	No. of PTs to trip system	3
	MTI	73
Valve type		1
PT type		2
MDT		129.7008
Cost		592
Spurious trip		0.45468
System unavailability		4.503e-7

6.5.4 ISPEASSOP Accuracy Comparison with FAULTREE+

To compare the accuracy of the ISPEASSOP program with the conventional Kinetic Tree Theory approach of FAULTTREE+, the best design (Table 6.13) and two more designs were analysed. Table 6.14 shows the parameter set of each considered design, where the final rows state the corresponding system unavailability and spurious trip occurrence using both techniques. In addition, the cost and maintenance down time of each design are provided.

Table 6.14 Comparison of Quantification Results of Three HIPS designs

Design Variables	Design1	Design2	Best Design
E	2	0	0
K_1 / N_1	2 / 3	4 / 4	1 / 1
H	0	1	1
K_2 / N_2	0 / 0	3 / 4	3 / 3
V	1	2	1
P	2	1	2
θ_1 / θ_2	108 / 71	29 / 104	25 / 73
MDT	54.2989	107.4483	129.7008
Cost	852	662	592
F_{sys} (ISPEASSOP)	**0.251831**	**0.531678**	**0.454681**
F_{sys} (FAULTTREE+)	**0.251850**	**0.531692**	**0.454644**
Q_{sys} (ISPEASSOP)	**8.260012e-4**	**1.900385e-6**	**4.50319e-7**
Q_{sys} (FAULTTREE+)	**8.260e-4**	**1.899e-6**	**4.503e-7**

Data comparison (Table 6.14) shows that the ISPEASSOP program produces accurate results. The insignificant discrepancy occurs due to programming language difference.

6.5.5 Results Comparison and Conclusions

Similar to the Pattison work, in total 10 runs, 100 generations each, of the ISPEASSOP were made. Therefore, 1000 system evaluations were performed in determining the best design. The running time of the program was an order of minutes. The best designs from each run of the program are shown in table 6.15.

Table 6.15 Characteristics of the Best Designs of ISPEASSOP after 10 Runs of 100 Generations

Run No.	Cost	MDT	F_{sys}	Q_{sys}
1	592	129.7008	0.455	4.5e-7
2	512	129.6974	0.332	8.33e-4
3	582	128.7361	0.324	6.8e-4
4	922	128.2273	0.718	1e-6
5	882	129.1590	0.166	1e-6
6	992	129.2523	0.552	1e-6
7	852	128.3286	0.245	6.55e-4
8	542	128.9881	0.324	8.45e-4
9	872	129.9032	0.377	1e-6
10	862	129.7309	0.999	1e-6
Average values	**761**	**129.1724**	**0.449**	**3.01e-4**
GASSOP Best result	**822**	**128.43**	**0.717**	**7.6e-4**

Design Variables										
Run No.	***Q1***	***Q2***	***V***	***P***	***N1***	***N2***	***K1***	***K2***	***E***	***H***
1	25	73	1	2	1	3	1	3	0	1
2	27	105	2	2	1	0	1	0	1	0
3	64	9	2	1	4	0	3	0	1	0
4	33	96	1	1	2	3	1	3	1	1
5	42	53	1	2	4	2	4	1	1	1
6	34	90	2	2	2	3	2	2	1	2
7	40	91	1	2	3	0	3	0	2	0
8	27	118	2	1	2	0	2	0	1	0
9	26	124	1	2	3	2	3	2	0	2
10	42	46	1	2	2	2	1	2	1	1

Pattison [Pattison, 1999] applied GASSOP program to the HIPS optimization (section 5.4). The best results were obtained after several modifications of the program routines (section 5.4.7). Table 6.15 shows the average values of optimization parameters produced by ISPEASSOP after 10 runs of the program are better than the best results produced by the simple GA (GASSOP). The comparison of the best designs obtained by the modified GASSOP and ISPEASSOP programs with the same parameters (100 generations, 0.01 mutation rate and 0.7 crossover rate) after 10 runs is represented in table 6.16.

Table 6.16 Results Comparison

		GASSOP	ISPEASSOP
Subsystem 1	No. of ESD valves	0	0
	No. of PTs	2	1
	No. of PTs to trip system	1	1
	MTI	29	25
Subsystem 2	No. of HIPS valves	2	1
	No. of PTs	3	3
	No. of PTs to trip system	2	3
	MTI	32	73
Valve type		2	1
PT type		1	2
MDT		**128.43**	**129.7008**
Cost		**822**	**592**
Spurious trip		**0.717**	**0.455**
System unavailability		**7.6e-4**	**4.5e-7**

Table 6.16 shows that the SPEA2 optimization algorithm, implemented in the ISPEASSOP program, gives better results. All the HIPS optimization parameters were improved. The available MDT resources are fully used (MDT is very close to 130 hours), the total system cost is smaller (price reduction is 230 units) as well as the spurious trip occurrence (approximately 1.5 times smaller) and system unavailability values (1690 times smaller). The result improvement can be explained by the fact, that multi-objective search of SPEA2 algorithm leads to nondominate solutions faster than simple GAs when the number of generations is relatively small (i.e. only 100 generations).

In both programs (GASSOP and ISPEASSOP) the maximum number of generations is equal to 100. The fittest design of GASSOP is achieved only in the 70th generation. In contrast, in all 10 runs of 100 generations of ISPEASSOP the fittest strings occurred in the first 10 generations. Consequently, the ISPEASSOP program requires less computer

memory recourses, which is an important advantage for the large safety systems optimization.

6.6 Program Modification and Results

The obtained results (Table 6.15 and 6.16) show that the SPEA2 optimization algorithm with chosen parameter values (100 generation, 0.7 crossover rate and 0.01 mutation rate) produces improved results compared to the modified simple Genetic algorithm. However, the modification of some optimization parameter values and some parts of the program could possibly improve the method further [Bartlett and Riauke, 2009]. The potential list of modifications would be to use:

- different main program parameter values:
    a) population size,
    b) crossover rate,
    c) mutation rate;
- different crossover procedure (not a single-point),
- different methods for changing infeasible parameter parts to the feasible ones (regeneration).

All the mentioned modifications have been investigated and are discussed in detail in sections 6.6.1-6.6.5.

6.6.1 Crossover and Mutation Rates Modification

According to De Jong's [De Jong (1), 1975] research of effectiveness of these parameters towards the GAs, the optimal values for the crossover rate appeared to be in the range from 0.5 to 0.9. The interval for the effective mutation rate values is (0, 0.2). Hence, investigation was produced with a limited set of values for each parameter (5 values for crossover rate and 3 values for mutation rate). These values were:

Crossover Rate: 0.5 0.6 0.7 0.8 0.9

Mutation Rate: 0.1 0.01 0.001

In total there are 15 possible combinations of these parameters. Hence, 15 runs of the ISPEASSOP program were carried out. Table 6.17 represents the fittest strings obtained for each combination of crossover and mutation rates. Table 6.18 shows the system designs for each combination from table 6.17 respectively.

Table 6.17 Best Results for Different Combinations of Values of Crossover and Mutation Rates

Experiment No.	Crossover Rate	Mutation Rate	Best Design			
			Cost	MDT	F_{sys}	Q_{sys}
1	0.5	0.1	592	121.2343	0.454393	8.17568e-7
2	0.5	0.01	592	129.9834	0.245132	5.05000e-4
3	0.5	0.001	582	124.8448	0.327878	5.05000e-4
4	0.6	0.1	592	125.4809	0.454405	8.00796e-7
5	0.6	0.01	582	124.8448	0.327878	5.05000e-4
6	0.6	0.001	572	128.4310	0.375863	5.80712e-4
7	0.7	0.1	612	127.8400	0.841392	5.53339e-7
8	**0.7**	**0.01**	**592**	**129.7008**	**0.454681**	**4.50419e-7**
9	0.7	0.001	582	124.8448	0.327878	5.05000e-4
10	0.8	0.1	512	124.9472	0.332248	8.49826e-4
11	0.8	0.01	592	129.9834	0.245132	5.05000e-4
12	0.8	0.001	592	121.2343	0.454393	8.17568e-7
13	0.9	0.1	942	129.6209	0.368503	8.87894e-7
14	0.9	0.01	592	129.9834	0.245132	5.05000e-4
15	0.9	0.001	592	129.9834	0.245132	5.05000e-4

Table 6.18 Best System Designs for Different Combinations of Values of Crossover and Mutation Rates

No.	Q1	Q2	E	H	N1 / K1	N2 / K2	V	P
1	25	105	0	1	1 / 1	3 / 3	1	2
2	29	108	1	0	2 / 2	0 / 0	1	1
3	29	104	1	0	4 / 4	0 / 0	2	1
4	24	105	0	1	1 / 1	3 / 3	1	2
5	29	104	1	0	4 / 4	0 / 0	2	1
6	29	104	1	0	1 / 1	0 / 0	1	1
7	25	65	0	1	1 / 1	2 / 1	1	1
8	**25**	**73**	**0**	**1**	**1 / 1**	**3 / 3**	**1**	**2**
9	29	104	1	0	4 / 4	0 / 0	2	1
10	28	111	1	0	1 / 1	0 / 0	1	2
11	29	108	1	0	2 / 2	0 / 0	1	1
12	25	105	0	1	1 / 1	3 / 3	1	2
13	25	107	0	2	2 / 2	4 / 2	1	1
14	29	108	1	0	2 / 2	0 / 0	1	1
15	29	108	1	0	2 / 2	0 / 0	1	1

Table 6.17 represents the nondominated set of optimal strings. There are three optimal solutions among the obtained possible system designs with parameter values which are the best according to the considered limitations. The first one is produced by the

combination of 0.7 crossover rate and 0.01 mutation rate (8). Combination 0.8 and 0.1 gives the second. Finally, the third optimal solution is produced by several combinations (2, 11, 14 and 15). The second design gives the smallest cost (512 units). However, the unavailability of the first design is the smallest of all obtained solutions (Q_{sys} = 4.504e-7). On the other hand, the third design benefits from the smallest spurious system failure (F_{sys} = 0.245132) and MDT closest to the limit of 130 hours (MDT = 129.9834). The difference between the values of MDT and system spurious failure and cost in all cases is not significant, therefore, the first solution can be assumed to be the best, since its unavailability is smaller.

6.6.2 Modification of the Population Size

The populations of 5, 10, 20 and 40 strings have been investigated by running the ISPEASSOP program with 0.7 crossover rate and 0.01 mutation rate, since this combination of rates gives the perceived best solution. The summary of these results is given in table 6.19. Table 6.20 shows the average and best design parameter values for each population size respectively.

Table 6.19 Average and Best Design Parameter Values for the Different Population Size

	Population of 5 strings			
	Cost	**MDT**	**F_{sys}**	**Q_{sys}**
Average	604	112.235	0.48179	2.3685e-4
Best	562	127.881	0.66345	7.6799e-7
	Population of 10 strings			
	Cost	**MDT**	**F_{sys}**	**Q_{sys}**
Average	748	118.462	0.42898	9.2653e-5
Best	582	124.845	0.32788	5.0504e-4
	Population of 20 strings			
	Cost	**MDT**	**F_{sys}**	**Q_{sys}**
Average	597	120.7182	0.37532	6.9379e-5
Best	**592**	**129.7008**	**0.45468**	**4.5042e-7**
	Population of 40 strings			
	Cost	**MDT**	**F_{sys}**	**Q_{sys}**
Average	657	123.5718	0.34232	5.3651e-5
Best	512	126.8485	0.33222	8.9231e-4

Table 6.20 Best Design Parameter Values for the Different Population Size

Population size	**Q1**	**Q2**	**E**	**H**	**N1 / K1**	**N2 / K2**	**V**	**P**
5 strings	27	45	0	1	2 / 2	1 / 1	2	1
10 strings	29	104	1	0	4 / 4	0 / 0	2	1
20 strings	25	73	0	1	1 / 1	3 / 3	1	2
40 strings	33	26	1	0	1 / 1	0 / 0	2	2

As it might be expected, the larger population has led to a better performance. Each time when the population size doubles, the average values of MDT, system spurious failure and unavailability are improved. However, the best design parameters values change chaotically. The largest population gives the smallest cost. The smallest spurious system failure occurs in the population of 10 strings. The population of 20 strings produces the smallest unavailability and fully utilized available recourses by MDT, which is closest to the limit of 130 hours. Similar to the results from section 6.6.1, all optimal designs are very close to each other. However, only the population of 20 strings produces the significantly smaller unavailability. Therefore, the population size equal to 20 strings can be assumed to be the best.

6.6.3 Modification of the Parameter Evaluation Scheme

There are four HIPS design parameters (number of pressure transmitters fitted and required for subsystems 1 and 2, i.e. N_1, N_2, K_1, K_2 respectively) which require closer consideration during their evaluation. Each of these parameters is represented by three string elements. According to the binary calculation system this memory allocation gives 8 possible values from the interval [0, 7]. However, only five values (0, 1, 2, 3 and 4) are feasible for the parameters N_2 and K_2, and only four values (1, 2, 3 and 4) are feasible for parameters N_1 and K_1 due to the HIPS structure. In the initial ISPEASSOP version these parameters evaluation scheme is shown in figure 6.13.

```
allowed_max := 4
allowed_min := 1
If    ((Parameter_value < allowed_min) or
            (parameter_value > allowed-maximum))
Then
   //start of then
            If  (parameter_value = 0) then  parameter_value := 1;
            If  (parameter_value = 5) then  parameter_value := 2;
            If  (parameter_value = 6) then  parameter_value := 3;
            If  (parameter_value = 7) then  parameter_value := 4;
   //end of then
```

Figure 6.13 Initial Parameter Evaluation Procedure

The modified parameter evaluation scheme is shown in Figure 6.14. If the parameter value is infeasible, the part of the string responsible for this parameter is regenerated and the parameter value is recalculated. The process stops only when the new value is from the feasible region.

```
allowed_max := 4
allowed_min := 1

While ((Parameter_value < allowed_min) or
        (parameter_value > allowed-maximum))

  //start of while
      Regenerate (part_of_string);
      Recalculate (parameter_value);
  //end of while
```

Figure 6.14 Modified Parameter Evaluation Procedure

Different size populations were tested with both parameter evaluation schemes. The summary of the results is shown in table 6.21. Table 6.22 represents the best designs obtained by both the initial and modified ISPEASSOP program versions.

Table 6.21 Comparison of Average and Best Designs after Modification of Parameter Estimation Procedure for Different Population Size

Population size	Estimation technique	Parameter Values	Cost	MDT	F_{sys}	Q_{sys}
5 strings	Initial	Average	**604**	**112.235**	**0.48179**	**2.3685e-4**
		Best	562	127.881	0.66345	7.6799e-7
	Modified	Average	**644**	**75.055**	**0.48311**	**6.2775e-5**
		Best	866	112.583	0.46344	6.7555e-7
10 strings	Initial	Average	**748**	**118.462**	**0.42898**	**9.2653e-5**
		Best	582	124.845	0.32788	5.0504e-4
	Modified	Average	**627**	**122.458**	**0.50899**	**8.0200e-7**
		Best	782	127.966	0.75463	7.3285e-7
20 strings	Initial	Average	**597**	**120.718**	**0.37532**	**6.9379e-5**
		Best	592	129.701	0.45468	4.5042e-7
	Modified	Average	**587**	**121.776**	**0.45439**	**6.1756e-7**
		Best	842	129.883	0.45059	1.0986e-6
40 strings	Initial	Average	**657**	**123.572**	**0.34232**	**5.3651e-5**
		Best	512	126.849	0.33222	8.9231e-4
	Modified	Average	**600**	**125.105**	**0.46006**	**6.6664e-7**
		Best	552	129.037	0.53719	1.3263e-6

Table 6.22 Best Designs with Modified Parameter Estimation Technique

Population size	Best Design	Q1	Q2	E	H	N1 / K1	N2 / K2	V	P
5 strings	Initial	27	45	0	1	2 / 2	1 / 1	2	1
	Modified	43	72	1	1	2 / 2	2 / 2	1	2
10 strings	Initial	29	104	1	0	4 / 4	0 / 0	2	1
	Modified	29	104	1	1	2 / 2	1 / 1	2	1
20 strings	Initial	25	73	0	1	1 / 1	3 / 3	1	2
	Modified	24	111	0	2	3 / 2	3 / 3	2	1
40 strings	Initial	33	26	1	0	1 / 1	0 / 0	2	2
	Modified	27	54	0	1	1 / 1	4 / 3	2	2

The investigation results (Table 6.21) show that the best designs were obtained for the population of 20 strings. The best values of the parameters are coloured. Both optimal solutions, called “best”, are very close to each other. However, the average results, produced by the modified method are better in terms of three optimization parameter values (cost, MDT and unavailability). This method produces significantly smaller average unavailability.

6.6.4 Modification of the Crossover Procedure

There are many different types of the crossover operator (section 4.3.1). Initially the single-point crossover was implemented for the HIPS optimization. This procedure works through the following scheme:

Step 1. The random number is generated.

Step 2. If the generated number is smaller than the crossover rate, the pair of population strings *j* and *j+1* are crossed at the randomly chosen position. If not, step one repeats for the pair of strings *j+1* and *j+2*.

Step 3. If population end is not reached the process repeats from step one for the next string in the population.

During the investigation process three additional crossover procedures were created. The modified method is similar to the single-point crossover. The main difference

appears at the third step, when consideration is given to the second parent string from the pair. This string can again participate in crossover as the first parent. Two-point and Three-point crossover methods first generate respectively two and three random positions of the string. Then parent strings are crossed at those points. All methods were applied to the different size populations (5, 10, 20 and 40 strings). Table 6.23 shows the summary of the results. The best designs are represented in table 6.24.

Table 6.23 Comparison of Average and Best Design Values for Different Crossover Types

Population Size	Parameter values	Crossover	Cost	MDT	F_{sys}	Q_{sys}
5 strings	Average	Single	**604**	**112.235**	**0.48179**	**2.3685e-4**
		Modified	**627**	**66.158**	**0.31818**	**5.7704e-4**
		2-point	**652**	**67.929**	**0.41680**	**3.0968e-4**
		3-point	**787**	**78.685**	**0.54929**	**6.2900e-5**
	Best	Single	562	127.881	0.66345	7.6799e-7
		Modified	782	79.166	0.64713	1.2419e-6
		2-point	782	83.288	0.64641	1.1553e-6
		3-point	862	83.283	0.46426	1.0299e-6
10 strings	Average	Single	**748**	**118.462**	**0.42898**	**9.2653e-5**
		Modified	**632**	**123.819**	**0.37788**	**3.0799e-4**
		2-point	**787**	**124.093**	**0.45118**	**8.8300e-7**
		3-point	**572**	**120.795**	**0.48523**	**8.7900e-7**
	Best	Single	582	124.845	0.32788	5.0504e-4
		Modified	562	129.814	0.24930	8.5832e-4
		2-point	882	125.393	0.44997	9.0974e-7
		3-point	592	121.234	0.45439	8.1757e-7
20 strings	Average	Single	**597**	**120.718**	**0.37532**	**6.9379e-5**
		Modified	**606**	**123.597**	**0.36355**	**4.0280e-4**
		2-point	**589**	**122.052**	**0.55891**	**8.5400e-7**
		3-point	**571**	**121.445**	**0.63578**	**8.3100e-7**
	Best	Single	592	129.701	0.45468	4.5042e-7
		Modified	562	129.983	0.24930	8.5832e-4
		2-point	632	126.961	0.58123	7.9337e-7
		3-point	542	123.332	0.53717	9.7937e-7
40 strings	Average	Single	**657**	**123.572**	**0.34232**	**5.3651e-5**
		Modified	**838**	**122.165**	**0.41504**	**1.2060e-6**
		2-point	**608**	**124.891**	**0.48490**	**8.7200e-7**
		3-point	**845**	**123.088**	**0.38133**	**1.2870e-6**
	Best	Single	512	126.849	0.33222	8.9231e-4
		Modified	622	128.818	0.79354	6.2559e-7
		2-point	582	128.267	0.66326	7.0493e-7
		3-point	852	128.245	0.37570	1.4473e-6

Table 6.24 Best Designs Comparison for Different Crossover Types

Population size	Best Design	Q1	Q2	E	H	N1 / K1	N2 / K2	V	P
5 strings	Single	27	45	0	1	2 / 2	1 / 1	2	1
	Modified	69	63	1	1	4 / 2	2 / 2	2	2
	2-point	64	63	1	1	4 / 2	2 / 2	2	2
	3-point	70	63	1	1	2 / 2	2 / 2	1	2
10 strings	Single	29	104	1	0	4 / 4	0 / 0	2	1
	Modified	29	111	1	0	1 / 1	0 / 0	1	2
	2-point	29	72	0	2	4 / 4	4 / 3	2	1
	3-point	25	105	0	1	1 / 1	3 / 3	1	2
20 strings	Single	25	73	0	1	1 / 1	3 / 3	1	2
	Modified	29	108	1	0	1 / 1	0 / 0	1	2
	2-point	25	77	0	1	3 / 2	1 / 1	1	1
	3-point	24	103	0	1	1 / 1	3 / 3	2	2
40 strings	Single	33	26	1	0	1 / 1	0 / 0	2	2
	Modified	28	44	0	1	2 / 1	4 / 4	2	1
	2-point	27	45	0	1	1 / 1	3 / 3	2	1
	3-point	26	98	0	2	1 / 1	2 / 2	1	2

As it was expected the new crossover methods produced poor results for the smallest population since it is not diverse. However, each time when the population size doubles the produced results are significantly improved. On the other hand, the obtained results are quite chaotic. In table 6.24 the best values of the optimization parameters are highlighted. For all three populations (10, 20 and 40 strings) the modified crossover method worked best since it produced the largest number of the best parameter values. In the population of 10 strings this technique resulted in three best values (cost, MDT and spurious system failure). However, 3-point crossover benefited from the smallest unavailability. In the population of 20 strings the best unavailability is produced by single crossover. On the other hand, the modified method gives two best parameter values (MDT and spurious system failure). In the population of 40 strings the single and modified crossover methods produce an equal number of the best parameter values. However, this time the modified method resulted in the smallest system unavailability.

6.6.5 Chosen Optimization Scheme

According to the investigation results (Tables 6.19 - 6.24) the following modified optimization scheme was chosen:

- Population of 20 strings;
- 0.7 crossover rate;
- 0.01 mutation rate;
- 100 generations;
- Modified parameter estimation procedure;
- Modified crossover operator.

Table 6.25 and 6.26 show the comparison of the best design parameter values obtained by the initial and modified ISPEASSOP.

Table 6.25 Best Design Characteristics

Population size	Estimation technique	Parameter Values	Cost	MDT	F_{sys}	Q_{sys}
20 strings	Initial	Average	**597**	**120.718**	**0.37532**	**6.9379e-5**
		Best	592	129.701	0.45468	4.5042e-7
	Modified	Average	**570**	**126.702**	**0.32278**	**5.0000e-4**
		Best	592	129.983	0.24513	5.0500e-5

Table 6.26 Best Designs Comparison

Best Design	Q1	Q2	E	H	N1 / K1	N2 / K2	V	P
Initial	25	73	0	1	1 / 1	3 / 3	1	2
Modified	29	108	1	0	2 / 2	0 / 0	1	1

All solutions (table 6.25 and 6.26) are nondominated. The initial version of the program gives smaller unavailability for both the best and average results. However, the modified ISPEASSOP produces better values for the system cost, MDT and spurious system failure. The average cost was reduced by 27 units. The average MDT was improved and is reasonably closer to the limit of 130 hours. The spurious system failure of the new best design is two times smaller. The difference between the initial and modified unavailability is 0.00005. If this difference is insignificant for the potential decision maker then the modified program results can be assumed to be better. Otherwise, the initial version of the ISPEASSOP program produces the best system design.

6.7 Summary

- This chapter has discussed the application of the developed optimization scheme to the high integrity protection system (HIPS). The suggested optimization tool is based on the combination and integration of three well known techniques: the fault tree analysis, the BDD method and the SPEA2 optimization algorithm.

- The ISPEASSOP program, created for the HIPS optimization, consists of three parts. Part one produces the unavailability and spurious trip fault trees for each potential HIPS design. Program part two is responsible for analysis using the BDD method which calculates the HIPS unavailability and spurious trip frequency. The final part of the program incorporates the SPEA2 technique to the optimization process, which searches for the fittest potential HIPS designs.

- To test the developed technique performance the following optimization parameter values have been chosen: the initial population of 20 strings, 100 generations, 0.7 and 0.01 for the crossover and mutation rate respectively.

- The ISSPEASSOP results were compared to those produced by the simple GA based optimization tool. The SPEA2 based technique performed better for the relatively small number of generations and led to the optimal solution quicker.

- The ISPEASSOP programs high speed in comparison to the GA based program can be explained by the effective use of the developed fault tree reduction technique, which reduces the BBD size for each potential system design and, hence, less time is required to run the program. Additional speed reduction is achieved by the advanced features of the SPEA2 algorithm.

- The ISSPEASSOP accuracy has been tested with the FaultTree + software. Both programs produce similar results. The small discrepancy occurs due to the programming language difference.

- The HIPS has been optimized by ISPEASSOP with different combinations of crossover and mutation rate in order to test the developed technique efficiency. Five different mutation rates and three crossover rates have been considered. Results showed that the initial combination (i.e. 0.7 crossover rate and 0.01 mutation rate) produce the best results.

- The chosen combination of crossover and mutation rate has been tested with four different population sizes. Results showed that the population of 20 strings is sufficient enough for a good SPEA2 approach performance.

- To enhance the developed optimization tool further several modifications to the program structure have been suggested. Two optimization parameter evaluation schemes and four different crossover procedures have been investigated. The best results were obtained by the modified parameter evaluation scheme, based on the regeneration of all infeasible parameter values; and the modified crossover procedure similar to the single-point crossover technique.

- The exhaustive search has been performed for the HIPS in order to test if the developed optimization tool finds optimal designs with system unavailability values close to this parameter global minimum, since this optimization parameter is considered to be the most important. Results showed that all optimal solutions obtained after 100 generations have the system unavailability value close to its global minimum.

- Based on the tests results, the modified safety system optimization scheme has been suggested. It has been applied to the HIPS and produced better results than the initial optimization tool version.

- The HIPS is a relatively small safety system and application to a larger system should be made in order to test the scalability of the method and confirm its suitability in this domain.

CHAPTER 7

FDS OPTIMIZATION BY SPEA2

7.1 Introduction

The Improved Strength Pareto Evolutionary Algorithm (SPEA2) has been applied to the High Integrity Protection System (HIPS). However, the HIPS is a relatively simple example of a safety system. Many systems are much more complex and have a much larger number of design variables. Therefore, in this chapter the effectiveness of the SPEA2 is tested on the larger and more detailed safety system, the Firewater Deluge System (FDS) [Borisevic and Bartlett 2007 (2), Riauke and Bartlett 2008].

Section 7.2 first introduces and describes the main features of the deluge system design used on an offshore platform. Section 7.3 provides the safety system analysis. The review of this system optimization by the single GA based technique is provided in section 7.4. The SPEA2 implementation to the FDS system with results and comparative analysis is discussed in section 7.5, followed by the summary (section 7.6).

7.2 General Structure of the FDS system

The Firewater Deluge System works to supply, on demand, water and foam at a controlled pressure to a specific area on the platform, protected by a deluge system [Andrews and Bartlett, 2003]. The main features of the deluge system are shown in figure 7.1. The FDS comprises a deluge skid, firewater pumps, with associated equipment and ringmains, and aqueous film-forming foam (AFFF) pumps, with associated equipment and ringmains. The description of the main parts of the FDS is provided in section 7.2.1. Section 7.2.2 represents the system design variables and limitations. The FDS failure events and data are discussed in section 7.2.3.

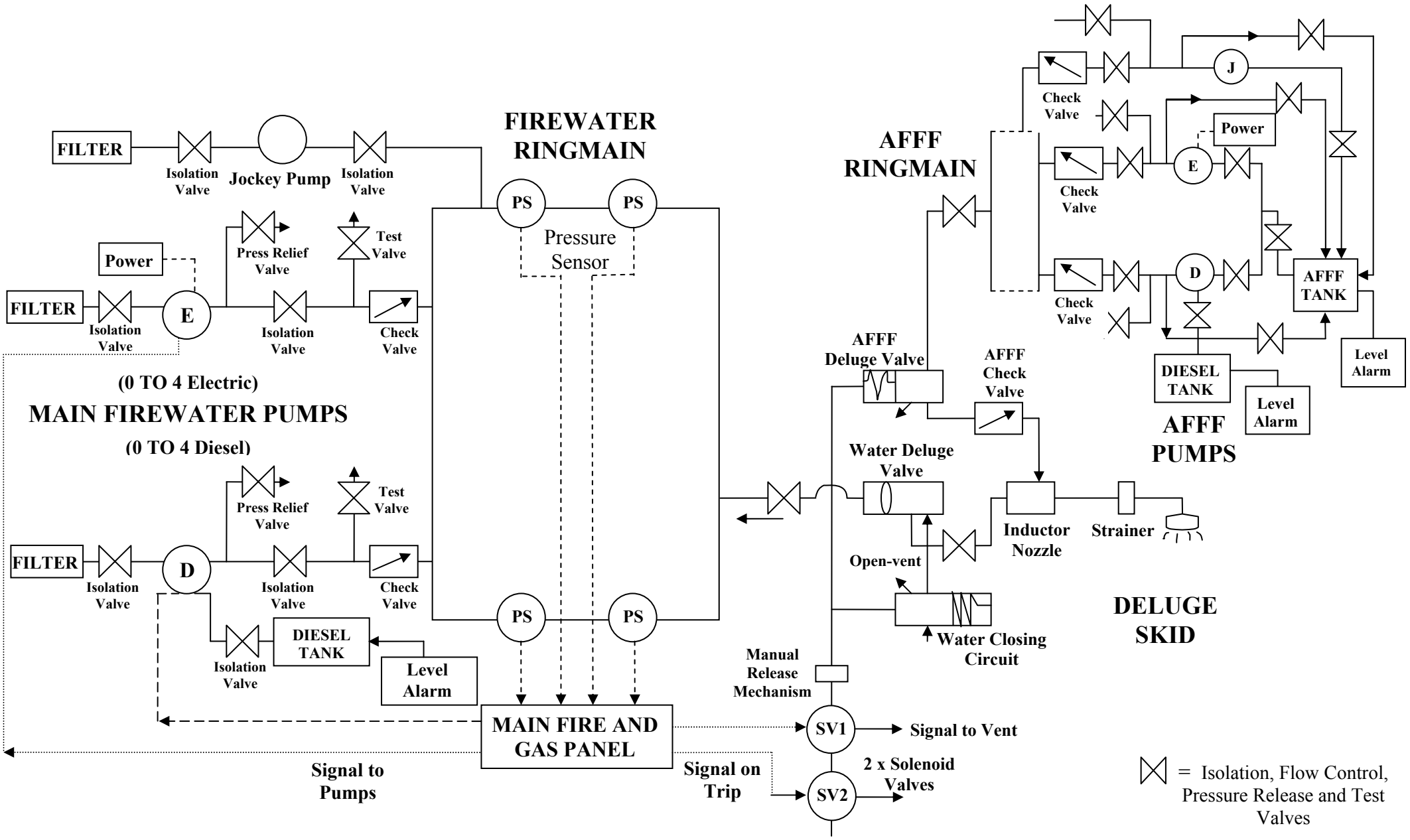


Figure 7.1 The Firewater Deluge System

7.2.1 Main Parts of the FDS

The Deluge System. The deluge valve set with all associated equipment is mounted on a fabricated steel framework called a skid. Skids are situated on the processing platform where an incident can occur. In this situation the associated equipment acts to spray water onto the affected area. The three main elements of the deluge valve set are the main distribution line, a water closing circuit and a control air circuit.

The system can be operated either manually by opening the systems local manual release valve on the skid or automatically: the main fire and gas panel (MFGP) gives the signal to the solenoid valves to de-energize and open thus releasing air pressure from the control air circuit. After the pressure drop the valmatic release valve opens and the water from the water closing circuit runs to drain. This process results in the fall of pressure on the deluge valve diaphragm. When the pressure on the diaphragm has fallen sufficiently, the firewater main pressure, acting on the underside of the deluge valve clack, overcomes the load imposed by the diaphragm. This allows water to flow into the distribution pipes, through the nozzle and onto the hazard.

The deluge valve set is also fitted with an aqueous film-forming foam (AFFF) supply line. Instrument air pressure maintains the valmatic release valve and AFFF valve closed. The AFFF valve and valmatic release valve open simultaneously when the air pressure drops in the control air circuit. This reaction is caused by the de-energising of the solenoid valves. As the water flows through the foam inductor in the main distribution line, foam concentrate is induced from the AFFF line via the foam proportioner. As a result, the solution of water and approximately 3% foam then feed into the distribution network, through the nozzles and onto the hazard.

Firewater Supply and Distribution System. The deluge systems are connected to a pressurised ringmain network. The jockey pump maintains the ringmain pressure by drawing water from the sea. The pressure transducers detect the falling pressure and subsequently send the signal to the MFGP, which activates the firewater pumps to supply water direct from the sea at sufficient pressure to meet the deluge requirement. In inactive standby the pumps remain not needed. Both pumps can be started manually at the fire control panel.

There are two sets of fire pumps: one set is powered from the main electric power plant and the other from their own dedicated diesel engines. The diesels have a day tank, which provides a 24 hour supply. The tank is fitted with a low level alarm, giving a signal in the central control room.

AFF Supply and Distribution System. The foam concentrate is stored in a stainless steel tank and is distributed through a stainless steel ringmain network. Similar to the firewater supply and distribution system, the tank has a low level alarm fitted, sounding in the central control room.

The foam system is kept at approximately the same pressure as the firewater system by a continuously running air driven jockey pump. There are two types of AFFF pumps: one supplied from the platform power plant, the other are diesel driven. When any firewater pump starts to supply foam at sufficient pressure to meet design requirements, the AFFF pumps start automatically.

It should be noted that the pumps not needed remain in standby. The diesel supply to the firewater diesel pumps is separate from that of the AFFF diesel pumps.

7.2.2 FDS Design Variables and Limitations

The overall FDS system can be represented by 17 design variables. These variables, their value and possible designer options are described in table 7.1.

It is important to notice that similar to the AFFF system pumps all pumps in the firewater system are to be of the same capacity. In addition, electric and diesel pumps of 100% capacity in the firewater system are of one type only, as are both 100% and 50% pumps in the AFFF system.

Table 7.1 FDS Design Variables

Variable	Description	Value
N	Number of pressure transmitters fitted on the ringmain	1, 2, 3, 4
K	Number of pressure transmitters required to trip	$1 - N$
P	Pressure transmitter type	1, 2 or 3
F_E	Number of the electrically powered firewater pumps	0 – 4
F_D	Number of the diesel firewater pumps (the total number of pumps $F = F_E + F_D : 1 \leq F \leq 8$)	0 – 4
F_P	The percentage of the capacity of the firewater pumps	100%, 50% or 33.3%
F_T	The pump type (for the 50% and 33.3% capacity pumps only)	1 or 2
A_E	Number of the electrically powered AFFF pumps	0 – 2
A_D	Number of the diesel AFFF pumps (the total number of pumps $A = A_E + A_D : 1 \leq A \leq 4$)	0 – 2
A_P	The percentage of the capacity for the AFFF pumps	100% or 50%
W	Water deluge valve type	1, 2 or 3
D	AFFF deluge valve type	1, 2 or 3
C	Type of the materials for certain purpose	1 or 2
θ_P	Maintenance test interval for the firewater and AFFF pump system	1 to 28 days
θ_R	Maintenance test interval for the ringmain	1 to 24 weeks
θ_D	Maintenance test interval for the deluge skid (in 3 monthly intervals only)	3 to 18 months
θ_{PM}	Preventative maintenance on components of wear-out type (in 3 monthly intervals only)	3 to 18 months

The FDS system must be tested at regular intervals and any failures found must be repaired. Some of the components are of wear-out type. Therefore, the preventative maintenance must be carried out at regular intervals. The expected cost of system testing, repairs and maintenance effort can be predicted by the knowledge of the components comprising the FDS. The life cycle cost is yielded by the initial cost and the cost of maintaining. The choice of design is not restricted; however, some limitations have been placed on the design. The summary of these limitations is shown in table 7.2. The system failure events and data are discussed in detail in appendix B (section B.1).

Table 7.2 The FDS Design Limitations

Limitation	Maximum Value (per year)
Total life cycle cost (*i.e. the sum of the initial cost and total cost of maintenance effort*)	< 125 000 units
Total cost of testing the system	< 20 500 units
Total cost of preventative maintenance effort	< 13 500 units
Total cost of maintenance effort (*i.e. the sum of the cost of corrective maintenance due to repair of dormant and spurious failure, total cost of testing the system and total cost of preventative maintenance*)	< 44 000 units
Acceptable number of times that a spurious system shutdown occurs	< 0.75 times

7.3 Safety System Analysis

The main goal of the FDS system is to mitigate the consequences of jet and pool fires. Furthermore, the system is designed to reduce overpressures in the event of an explosion. Failure in the event of a hydrocarbon release could result in fatalities. Therefore, the objective of optimization is to produce a safe, reliable FDS design with consideration to the available recourses.

This section provides the FDS life cycle cost evaluation scheme (section 7.3.1). Construction of the system unavailability and spurious trip fault trees calculation method are discussed in section 7.3.2.

7.3.1 FDS Life Cycle Cost Evaluation

The FDS life cycle cost is an important system optimization parameter due to the constraints imposed on the FDS. The important component of the life cycle cost is the initial cost to build the FDS (section 7.3.1.1). However, the system running costs must also be taken into account. These costs include only the maintenance activity, including the cost of corrective maintenance to repair any problem highlighted by system testing (section 7.3.1.2), the cost of preventative maintenance carried out at regular intervals on components that exhibit wear-out (section 7.3.1.3) and the cost of system testing at regular intervals (section 7.3.1.4). All these costs are evaluated over a period of 1 year,

i.e. 8760 hours. Data regarding the costs associated with each component is specified in appendix B (tables B.3, B.6, B.7, B.8 and B.10).

7.3.1.1 Initial Cost

Each component has an initial purchase cost and a storage cost. The storage cost depends on the number of spare items stored and the cost to store each item. Therefore, the FDS system initial cost (*SIC*), including the storage cost, is calculated as:

$$SIC = ICFPL + ICAPL + ICR + ICDS\,, \tag{7.1}$$

where *ICFPL* denotes the initial cost of the firewater pumps and lines, *ICAPL* is the initial cost of the AFFF pumps and lines, *ICR* is the initial cost of the ringmain and, finally, *ICDS* denotes the initial cost of the diesel supply.

Considering the *ICFPL* further, it can be expressed as:

$$ICFPL = \left[F_E\left(5400 + FE_{IS}\right)\right] + \left[F_D\left(5400 + FD_{IS}\right)\right] + 2600\,, \tag{7.2}$$

where F_E is the number of the electrically powered firewater pumps, 5400 is the sum of the initial plus storage cost of each fixed component on an electric pump line, FE_{IS} and FD_{IS} are the initial plus storage cost of the particular electric and diesel pump chosen (i.e. 100%, 50% or 33.33%, type 1 or 2) respectively. F_D denotes the number of diesel driven pump lines. 2600 is the sum of the initial cost of the electric and diesel supplies.

The *ICALP* expression is similar to *ICFPL* and is given by:

$$ICAPL = \left[A_E\left(6400 + AE_{IS}\right)\right] + \left[A_D\left(6400 + AD_{IS}\right)\right] + 3600\,, \tag{7.3}$$

where A_E is the number of the electrically powered AFFF pumps, 6400 is the sum of the initial plus storage cost of each fixed component on an electric or diesel pump line

(it differs from the firewater pump due to an additional isolation valve), AE_{IS} and AD_{IS} are the initial plus storage cost of the particular electric and diesel pump chosen. A_D denotes the number of the diesel driven pumps in the AFFF system. 3600 is the sum of the initial cost of the electric supply, diesel supply and AFFF tank isolation valve.

The only fixed component of the ringmain is the fire pump selector unit (FSU), Therefore, ICR is calculated as

$$ICR = 2200 + P_{IS}, \tag{7.4}$$

where 2200 is the initial plus storage cost of the FSU and P_{IS} is the initial plus storage cost of the particular pressure sensor chosen.

The final components of the system initial cost is *ICDS*, given by

$$ICDS = 7350 + VRV_{IS} + N_{IS} + IN_{IS} + WV_{IS} + AV_{IS}, \tag{7.5}$$

where 7350 is the sum of the initial pus storage cost of the fixed component. The latter terms (valmatic release valve (VRV), nozzle (N), inductor nozzle (IN), water deluge valve (WV) and AFFF deluge valve (AV)) correspond to the initial pus storage cost of the particular type of variable components chosen.

Therefore, the initial system cost (SIC) can be expressed as

$$\begin{aligned} SIC = {} & \left[F_E\left(5400 + FE_{IS}\right) + F_D\left(5400 + FD_{IS}\right) + 2600\right] + \\ & + \left[A_E\left(6400 + AE_{IS}\right) + A_D\left(6400 + AD_{IS}\right) + 3600\right] + \\ & + \left[2200 + P_{IS}\right] + \\ & + \left[7350 + VRV_{IS} + N_{IS} + IN_{IS} + WV_{IS} + AV_{IS}\right]. \end{aligned} \tag{7.6}$$

7.3.1.2 Corrective Maintenance Cost

The corrective maintenance cost of each component depends on the expected number of failures and the cost to repair each failure. Consider component *i*. The corrective maintenance of this component (CM_i) can be calculated as:

$$CM_i = \left(W_i^D + W_i^S\right)\left(N_R \cdot C_{HR} + C_{SR}\right), \tag{7.7}$$

where W_i^D and W_i^S are the expected number of dormant and spurious failures for the component *i* over the one year time period, i.e. 8760 hours. N_R, C_{HR} and C_{SR} denote the number of hours of manual work required to test the component, the cost per hour of manual work to repair failure and the cost of spares for each repair carried out respectively.

The unconditional failure intensity (w_i) for the non wear-out type component *i* is given by

$$w_i = \lambda_i(1 - q_i). \tag{7.8}$$

Hence, the expected number of failures during the one year period time (8760 hours) is determined by

$$W(0,8760) = \int_0^{8760} w_i(u)du. \tag{7.9}$$

The failure rate for the wear-out type component *i* is time dependent and, therefore, is defined by the Weibull distribution. Hence, the expected number of failures per year is:

$$W_i(0,8760) = \left(\frac{8760}{\theta_{PM}}\right)\int_0^{\theta_{PM}} \left(\frac{\beta_i}{\eta_i}\right)\left(\frac{t}{\eta_i}\right)(1 - q_i)dt\ , \tag{7.10}$$

where θ_{PM} is converted to hours, and β_i, η_i are Weibull distribution parameters (Appendix B, Tables B.8 and B.11).

The total FDS corrective maintenance cost (*SCMC*) is calculated in a similar manner to the initial cost. Therefore,

$$SCMC = CMCFPL + CMCAPL + CMCR + CMCDS\,, \tag{7.11}$$

where *CMCFPL* is the corrective maintenance cost of the firewater pumps and lines, *CMCAPL* denote the corrective maintenance cost of the AFFF system pumps and lines, *CMCR* and *CMCDS* are the corrective maintenance costs of the ringmain and the deluge skid respectively.

CMCFPL is formulated in a similar manner to that of the *ICFPL* (Equation 7.2). Hence,

$$\begin{aligned} CMCFPL = &\left[F_E\left(Fix + \left(W_{FE}^{D} \cdot FE_{CM} rep\right)\right)\right] + \\ &+ \left[F_D\left(Fix + \left(W_{FD}^{D} \cdot FD_{CM} rep\right)\right)\right] + \\ &+ \left[\mathrm{W}_{\mathrm{ES}}^{\mathrm{D}} \cdot 290\right] + \left[\left(\mathrm{W}_{\mathrm{DIV}}^{\mathrm{D}} \cdot 400\right) + \left(\mathrm{W}_{\mathrm{LA}}^{\mathrm{D}} \cdot 290\right)\right], \end{aligned} \tag{7.12}$$

where

$$Fix = \left(W_{CV}^{D} \cdot 840\right) + \left(W_{PRV}^{D} \cdot 790\right) + \left(W_{SV}^{D} \cdot 1120\right) + \left(W_{FB}^{D} \cdot 410\right) + \left(W_{IV}^{D} \cdot 740\right).$$

The calculation of *CMCAPL* has the same principles as CMCFPL, i.e.:

$$\begin{aligned} CMCFPL = &\left[A_E\left(Fix + \left(W_{AE}^{D} \cdot AE_{CM} rep\right)\right)\right] + \\ &+ \left[A_D\left(Fix + \left(W_{AD}^{D} \cdot AD_{CM} rep\right)\right)\right] + \\ &+ \left[\mathrm{W}_{\mathrm{ES}}^{\mathrm{D}} \cdot 290\right] + \left[\left(\mathrm{W}_{\mathrm{DIV}}^{\mathrm{D}} \cdot 400\right) + \left(\mathrm{W}_{\mathrm{LA}}^{\mathrm{D}} \cdot 290\right)\right] + \left[W_{AIV}^{D} \cdot 740\right], \end{aligned} \tag{7.13}$$

where

$$Fix = \left(W_{CV}^{D} \cdot 840\right) + \left(W_{PRV}^{D} \cdot 790\right) + \left(W_{SV}^{D} \cdot 1120\right) + \left(W_{FB}^{D} \cdot 410\right) + \left(W_{IV}^{D} \cdot 740\right).$$

The corrective maintenance cost of the ringmain (*CMCR*) is given by

$$CMCR = \left(W_{FSU}^{D} \cdot 1280\right) + \left[\left(W_{PT}^{D} + W_{PT}^{S}\right) \cdot PT_{CM} rep\right], \tag{7.14}$$

where W^D and W^S are the expected number of dormant and spurious failures respectively, since the pressure transmitters fail in two modes (dormant and spurious failure).

The corrective maintenance of the deluge skid (*CMCDS*) involves the summation of the corrective maintenance cost of each associated component. Taking into account that the solenoid and valmatic valves can fail in two modes (dormant and spurious), the *CMCDS* is expressed as:

$$\begin{aligned} CMCDS = & \left(560 \cdot \left(W_{SV1}^{D} + W_{SV1}^{S}\right) + 485 \cdot \left(W_{SV2}^{D} + W_{SV2}^{S}\right)\right) + \\ & + \left(660 \cdot W_{MRM}^{D} + 410 \cdot W_{S}^{D} + 2 \cdot 740 \cdot W_{IV}^{D} + 840 \cdot W_{CV}^{D}\right) + \\ & + \left(\left(\mathrm{W}_{\mathrm{VRV}}^{\mathrm{D}} + \mathrm{W}_{\mathrm{VRV}}^{\mathrm{S}}\right) \cdot VRV_{CM}rep + W_{N}^{D} \cdot N_{CM}rep + W_{ID}^{D} \cdot IN_{CM}rep\right) + \\ & + \left(\mathrm{W}_{\mathrm{WV}}^{\mathrm{D}} \cdot \mathrm{WV}_{\mathrm{CM}}\mathrm{rep} + \mathrm{W}_{\mathrm{AV}}^{\mathrm{D}} \cdot \mathrm{AV}_{\mathrm{CM}}\mathrm{rep}\right). \end{aligned} \tag{7.15}$$

7.3.1.3 Preventative Maintenance Cost

The preventative maintenance is required only for the wear-out type components. The preventative maintenance cost per year of each component depends on the number of times preventative maintenance is carried out in the year and the cost per effort. The total FDS preventative maintenance cost (*SPMC*) is, therefore, calculated as the sum of the preventative maintenance costs (PMC_i) incurred by each component *i*. PMC_i is given by

$$PMC_i = \left(\frac{8760}{\theta_{PM}}\right)\left(\left(N_P \cdot C_{HP}\right) + C_{SP}\right), \tag{7.16}$$

where θ_{PM} is converted to hours, N_P is the number of hours manual work required to carry out preventative maintenance, C_{HP} denotes the cost per hour of manual work to carry out preventative maintenance, and C_{SP} is the cost of spares each time preventative maintenance is undertaken.

Therefore, the *SPMC* is calculated as:

$$SPMC = \left(\frac{8760}{\theta_{PM}}\right)\left(F_E \cdot FE_{PM}rep + F_D \cdot FD_{PM}rep\right) + \\ + \left(\frac{8760}{\theta_{PM}}\right)\left(A_E \cdot AE_{PM}rep + A_D \cdot AD_{PM}rep\right), \tag{7.17}$$

where F_E and A_E represent the number of electrically powered firewater and AFFF pumps respectively, F_D and A_D denote the number of diesel driven firewater and AFFF pumps respectively. $FE_{PM}rep$, $FD_{PM}rep$, $AE_{PM}rep$ and $AD_{PM}rep$ are the costs of preventative maintenance of the particular firewater electric and diesel pump type chosen and the AFFF electric and diesel pump type chosen respectively.

7.3.1.4 Testing Cost

System tests are carried out on each pump line (θ_P), the distribution network (θ_R) and deluge skid (θ_D). The cost of testing must only be considered once per group of components, since a pump line test examines the pump and all other elements on that line simultaneously, and a single ringmain and deluge skid test examines all associated components. It is assumed that the simultaneously tested components require the same specialized labour (C_{HT}) and the same test time (H_T) as all other elements. Therefore, the FDS testing cost (*STC*) can be evaluated as:

$$STC = TCFPL + TCAPL + TCR + TCDS, \tag{7.18}$$

where *TCFPL*, *TCAPL*, *TCR* and *TCDS* denote the cost of testing the firewater pumps and lines, the cost of testing the AFFF pumps and lines, the cost of testing the ringmain and the deluge skid respectively.

Each component of the firewater pump lines has a test time of 2 hours (H_T = 2) and a cost of 30 units per hour ($C_{HT} = 30$). The cost per hour to test the electricity supply is 45 units. Therefore, the testing cost of the firewater pumps and lines (*TCFPL*) can be calculated as the sum of the cost to test the electricity supply per year and the total cost of all pump lines per year, i.e.:

$$TCFPL = F\left[\frac{8760}{\theta_P}(2\cdot 30)\right]+\left[\frac{8760}{\theta_P}(2\cdot 45)\right]= \\ =\frac{8760}{\theta_P}\cdot(60\cdot F\ +90), \tag{7.19}$$

where F is the total number of the firewater pumps.

Using similar principles, *TCAPL* is given by

$$TCAPL=\frac{8760}{\theta_P}\cdot(60\cdot A\ +90). \tag{7.20}$$

The cost of testing the ringmain per year is

$$TRC=\frac{8760}{\theta_R}\cdot 45\,. \tag{7.21}$$

The cost of the deluge skid per year is

$$TCDS=\frac{8760}{\theta_D}\cdot 60\,. \tag{7.22}$$

Therefore,

$$STC=\frac{8760}{\theta_P}\cdot(60\cdot F\ +90+60\cdot A\ +90)+\frac{8760}{\theta_R}\cdot 45+\frac{8760}{\theta_D}\cdot 60= \\ =8760\cdot\left[\frac{60\cdot(F\ +A\ )+180}{\theta_P}+\frac{45}{\theta_R}+\frac{60}{\theta_D}\right]. \tag{7.23}$$

7.3.1.5 Life Cycle Cost

The FDS total life cycle cost is evaluated by summing equations (7.6), (7.11), (7.19) and (7.23), i.e.

$$LCC = SIC + SCMC + SPMC + STC\ . \tag{7.24}$$

The last three terms of equation 7.24 are time dependent and, therefore, their sum gives the cost due to total maintenance effort (*TMEC*). Hence, the life cycle cost can be expressed as:

$$LCC = SIC + TMEC\ , \tag{7.25}$$

where $TMEC = SCMC + SPMC + STC$.

7.3.2 FDS Fault Tree Construction

Unavailability fault tree construction. The top event 'Firewater Deluge System Fails to Protect' represents the causes of the firewater deluge system unavailability. There are three main reasons for the top event to occur. Either the firewater or AFFF pump mechanisms are not activated, the AFFF pumps themselves fail or the water or foam deluge systems fail. The first reason, i.e. failure to initiate the firewater and AFFF pump mechanisms, occurs if both automatic and manual interventions fail. The manual start of the system fails if either the push button on the MFGP fails or if the operator fails to push the button. An automatic start fails if either the fire pump selector unit fails or the low pressure sensing on the firewater ringmain fails. Failure of the low pressure sensing depends on the number of pressure transmitters fitted (N) and the number of pressure transmitters required to trip the system (K).

Failure of the AFFF or water deluge skid occurs if either events 'Failure of the water deluge skid' or 'Failure of the AFFF Deluge Skid' occur. The possible reasons for the event 'Failure of the water deluge skid' to occur are: the water spray isolation valves fail, the strainer nozzle becomes blocked or the deluge valve fails to open. Further development of the event 'The water deluge valve fails to open' involves two scenarios connected by OR logic, i.e. events that restrict activation of the deluge valve and failure of the deluge valve itself. 'Failure to activate the water deluge valve' can be caused by the failure of the signal to the solenoids, by the solenoid valves remaining energized or by the failure of the valmatic release valve. In a similar manner the event 'Failure of the AFFF deluge skid' is developed. The fault tree consists of 618 gates, 50 basic events

and 59 house events. The fault tree construction is discussed in appendix B (section B.2.1). The detailed fault tree structure is represented in appendix D.

Spurious trip fault tree construction. According to the FDS system limitations a number of spurious system occurrences is permitted, i.e. $F_{sys} < 0.75$ (Table 7.2). Hence, the spurious activation of the FDS must be established by developing the specific fault tree to quantify causes of this failure mode. The top event 'Firewater deluge system fails spuriously' occurs if the solenoid valves fail spuriously, the valmatic release valve opens spuriously or the signal from the main fire and gas panel to the solenoid valves is interrupted. The latter event occurs as a result of spurious activation of the ringmain pressure sensors.

Constant failure rates are ascribed to all components from the FDS spurious trip fault tree. Furthermore, spurious failures are instantaneously revealed and repair initiated, hence the probability of failure of each basic event is independent of its associated maintenance test interval. Similarly to the unavailability fault tree, a single spurious trip fault tree with incorporated house events is formed to analyze each potential FDS design. After setting the house events, the resulting fault tree is converted to its BDD and the spurious s trip frequency is calculated within SPEA2 source code. The fault tree consists of 61 gates, 16 basic events and 13 house events. The fault tree construction is discussed in appendix B (section B.2.2). The detailed fault tree structure is shown in appendix D.

7.4 Review of Previous Work on FSD

In 1999 Pattison [Pattison, 1999] optimized the firewater deluge system by the simple genetic algorithm. In her work she applied the SGA_C algorithm, which is a C-language translation and extension of the original Pascal Simple Genetic Algorithm (SGA) code developed by Goldberg [Goldberg 1989]. This package was used to build the GA optimization software called GASSOP (Genetic Algorithm Safety System Optimization Procedure). The brief description of this software and its implementation to the high integrity protection system optimization are given in section 5.4. GASSOP has been

applied to the FDS with some modifications. These modifications and optimization results are briefly discussed in this section.

Penalty Derivation and String Fitness Evaluation. One of the main routines of the GASSOP software is the routine *Fitness*, which obtains a fitness value for each possible system design. The main components for the fitness evaluation are: spurious trip frequency (F_{sys}), system unavailability (Q_{sys}), penalised system unavailability (Q'_{sys}).

Two further data structures, associated with each individual, are called *lifeparts* and *penparts*. The *lifeparts* data structure is responsible for the system cost of each possible design and consists of the following components: life cycle cost per year of the FDS (*LCC*), cost per year due to system testing (*STC*), cost per year due to preventative maintenance (*SPMC*), cost per year due to corrective maintenance (*SCMC*), cost per year due to total maintenance effort (*TMEC*).

The FDS available resources are not inexhaustible (limitations are provided in table 7.2). Data structure *penparts* consists of the penalties applied to each design if any of parameters from table 7.2 exceed their respective limits. The penalty due to excess life cycle cost (LCC_P) is given by equation 7.26,

$$LCC_P = \left(\frac{excessLCC \cdot 100}{125000} \right)^{\frac{9}{8}} \cdot Q_{sys}, \tag{7.26}$$

where the first term expresses the percentage by which the life cycle cost exceeds its constraint and the denominator is the life cycle cost maximum allowed value (Table 7.2). The system unavailability of the considered design is then multiplied by the respective percentage excess to establish the appropriate penalty. Penalties due to excess system testing cost (STC_P), preventative maintenance cost ($SPMC_P$) and total maintenance effort ($TMEC_P$) are calculated in similar manner to the LCC_P (Equations 7.27 – 7.29).

$$STC_P = \left(\frac{excessSTC \cdot 100}{20500} \right)^{\frac{9}{8}} \cdot Q_{sys}. \tag{7.27}$$

$$SPMC_P = \left(\frac{excessSPMC \cdot 100}{13500} \right)^{\frac{9}{8}} \cdot Q_{sys}. \tag{7.28}$$

$$TMEC_P = \left(\frac{excessTMEC \cdot 100}{44000} \right)^{\frac{9}{8}} \cdot Q_{sys} . \tag{7.29}$$

It is important to notice, that equations 7.27-7.29 are not independent. Excess life cycle cost affects one or more other constraints. Moreover, both system testing and preventative maintenance comprise a proportion of total maintenance effort. The exponential relationship, i.e. $y = x^{\frac{9}{8}}$, has been chosen to compensate for this interaction to a certain extent [Pattison, 1999].

Occurrence of the spurious trip causes financial loss due to the ceasing of production on the processing platform. As a result the spurious trip constraint violation is expressed in terms of excess cost. Therefore, the spurious trip penalty (S_P) is given by

$$S_P = \left(\frac{Excess\ \cos t \cdot 100}{203000} \right)^{\frac{9}{8}} \cdot Q_{sys} . \tag{7.30}$$

Each penalty is subsequently added to the system unavailability to give the total penalized system unavailability value for each possible FDS design, i.e.

$$Q'_{sys} = Q_{sys} + LCC_P + STC_P + SPMC_P + TMEC_P + S_P . \tag{7.31}$$

Modifications of the Genetic Operators. The fitness conversion method used in the selection process for the HIPS system (Section 5.4) has been modified due to the complexity of the FDS system. The range of fitness values that represent a design is larger, hence, an extra category has been incorporated and the bounds of each category have been modified. The new method consists of ten categories with values from the interval from 0 to 0.4.

Since the search space occupied by the maintenance test interval (MTI) parameters is comparatively large, it was decided [Pattison, 1999] to store the MTI parameters on a separate binary string (referred to as string 2) and, hence, isolate the action of genetic operators (crossover and mutation) on these variables. All other parameters are stored on string 1, which is affected by crossover and mutation in the normal manner. After each modification, the parameter set of the possible FDS designs is checked for feasibility. Each check follows any design modification, leading to the system

unavailability, spurious trip frequency calculation and, hence, life cycle costs and penalised system unavailability re-evaluation. As a result, the fittest design is selected to enter the new population.

Results. The GASSOP has been tested 10 times with a population of 20 strings and 100 generations. The mutation and crossover rates were selected as 0.01 and 0.7 respectively. Each run required several hours. The best design arose in the 2nd run and produced a system unavailability of 1.263×10^{-2}. Therefore, this design is over 98.73% available. Table 7.3 shows the characteristics of this design.

Table 7.3 Characteristics of the Best FDS Designs

General			*Firewater Supply and Distribution System*			
K	***N***	***P***	F_E	***F***	F_P	F_T
1	3	1	3	6	50%	2
AFFF Supply and Distribution System			*Valve and Material Types*		
A_E	***A***	A_P	***W***	***D***	***C***
1	2	100%	3	3	2
Maintenance Intervals					
θ_P		θ_R	θ_D		θ_{PM}
18		1	3		18
Optimization Parameter Values						
STC	***SPMC***	***TMEC***	***LCC***	F_{sys}	Q_{sys}	Q'_{sys}
8759.3	11123.8	29640.7	120386	0.403	1.263e-2	1.263e-2

Ten runs of the program resulted in designs with similar parameter values [Pattison, 1999]. Considering the deluge system, both the water and AFFF deluge valve are predominantly of type 3. The pipe work is consistently of type 2, i.e. the non-corrosion resistant material. The combination of 3 firewater pumps, 1 electrically driven and 2 diesel driven is dominating for the firewater supply and distribution system.

Considering the AFFF pump system, two combinations repeatedly arise. They are: two 100% pumps (1 electric and 1 diesel) and four 50% pumps (2 electric and 2 diesel). It

can be noticed, that the balance in the number of electric to diesel pumps leads to the fittest designs (particularly regarding those of 50% capacity).

The pressure transmitters are predominantly of type 1, thus, preventing the number of spurious trip occurrences from exceeding its limit of 0.75 per year. A strong pattern arises in the values assigned to the maintenance test interval parameters and the total life cycle cost of the system.

The maintenance test interval for the ringmain is set as 1 week for all but one of the best designs. The deluge skid is consistently tested at 3 monthly intervals. The test interval between preventative maintenance, conversely, tends to be at the higher end of its range, i.e. 15 to 18 months. Greater variation exists regarding θ_P.

The total life cycle cost of each of the best designs approaches the limit of 125000 units. Hence, the majority of the best designs utilise all of the available recourses.

7.5 SPEA2 Implementation to FDS

The Improved Strength Pareto Evolutionary Algorithm Safety System Optimization Procedure (ISPEASSOP), described in chapter 6, has been applied to the FDS system. The framework of the program is predominantly the same as for the HIPS. The main differences occur in coding and initialization subroutines *Generate_population, Calc_opt_param*, *Cost_mdt* and *Penalties* (section 6.2.1), and in the FDS design construction subroutines *House_event_data, Calc_HE_values, Formate_data_unav* and *Formate_data_spur* (section 6.2.2). All optimization subroutines were slightly modified due to the extension of the number of optimization parameters. All differences and modifications made to the ISPEASSOP are discussed in sections 7.5.1 – 7.5.3.

7.5.1 Modifications for Coding and Initialization

Each potential FDS design is described by 17 parameters (Table 7.1). The subroutine *Generate_initial_population* produces the initial population of 20 strings, 43 bits in length. The number of bits allocated to and the order in which each parameter is stored is shown in table 7.4.

Table 7.4 The FSD Parameter Set Binary Representation

Variable	Description	Values	No. of bits
N	Number of pressure transmitters fitted on the ringmain	1, 2, 3, 4	3
K	Number of pressure transmitters required to trip	1 - N	3
P	Pressure transmitter type	1, 2 or 3	2
F_E	Number of the electrically powered firewater pumps	0 – 4	3
F_D	Number of the diesel firewater pumps	0 – 4	3
F_P	The percentage of the capacity of the firewater pumps	100, 50, 33.3%	2
F_T	The pump type	1 or 2	1
A_E	Number of the electrically powered AFFF pumps	0 – 2	2
A_D	Number of the diesel AFFF pumps	0 – 2	2
A_P	The percentage of the capacity for the AFFF pumps	100% or 50%	1
W	Water deluge valve type	1, 2 or 3	2
D	AFFF deluge valve type	1, 2 or 3	2
C	Type of the materials for certain purpose	1 or 2	1
θ_P	Maintenance test interval for the firewater and AFFF	1 to 28 days	5
θ_R	Maintenance test interval for the ringmain	1 to 24 weeks	5
θ_D	Maintenance test interval for the deluge skid	3 to 18 months	3
θ_{PM}	Preventative maintenance on components of wear-out type	3 to 18 months	3

The subroutine *Calc_opt_parameters* evaluates the values of all 17 design parameters and changes the infeasible parts of the string according to the following rules:

1. While $N < 1$ or $N > 4$, regenerate N;
2. While $K = 0$ or $K > N$, regenerate K;
3. If $P = 0$, regenerate P;
4. If $F_p = 0$, regenerate F_p, value 1 refers to 100%, 2 to 50% and 3 to 33.33%;
5. If $F_E > 4$, regenerate F_E;
6. If $F_D > 4$, regenerate F_D;
7. F is the total number of the firewater pumps:

   If $F_P = 1$ (100%), F values are in the range 1 – 8,

   If $F_P = 2$ (50%), F values are in the range 2 – 8,

   If $F_P = 3$ (33.33%), F values are in the range 3 – 8.

Hence,

If $F_P = 1$ and $F = F_E + F_D < 1$, while $F_E + F_D < 1$ regenerate F_E and F_D;

If $F_P = 2$ and $F = F_E + F_D < 2$, while $F_E + F_D < 2$ regenerate F_E and F_D;

If $F_P = 3$ and $F = F_E + F_D < 3$, while $F_E + F_D < 3$ regenerate F_E and F_D.

8. $F_T = 0$ refers to type 1 and $F_T = 1$ refers to type 2.
9. $A_p = 0$ refers to 100%, $A_p = 1$ refers to 50%.
10. If $F_E = 3$, regenerate F_E;
11. If $F_D = 3$, regenerate F_D;
12. A is the total number of the AFFF pumps:

    If $A_P = 0$ (i.e. 100%), A values are in the range 1 – 4,

    If $A_P = 1$ (i.e. 50%), A values are in the range 2 – 4.

    Hence,

    If $A_P = 0$ and $A = A_E + A_D < 1$, while $A_E + A_D < 1$ regenerate A_E and A_D;

    If $A_P = 1$ and $A = A_E + A_D < 2$, while $A_E + A_D < 2$ regenerate A_E and A_D.

13. If $W = 0$, regenerate W;
14. If $D = 0$, regenerate D;
15. $C = 0$ and C = 1 refer to the old and new type of material respectively.
16. While $Q_P = 0$ or $Q_P > 28$, regenerate Q_P;
17. While $Q_R = 0$ or $Q_R > 24$, regenerate Q_R;
18. While $Q_D = 0$ or $Q_D = 7$, regenerate Q_D;
19. While $Q_{PM} = 0$ or $Q_{PM} = 7$, regenerate Q_{PM}.

It is important to note, that an integer in the range 1 to 6 is generated for parameters Q_D and Q_{PM}. The resulting value is subsequently multiplied by 3, so that each parameter is assigned a value in accordance with its feasible range, i.e. 3 to 18 months in 3 month intervals.

Since the maintenance down time (MDT) is not an optimization parameter of the FDS, the subroutine *Cost_mdt* from the initial version of the ISPEASSOP was changed by the subroutine *Cost_calc*. The new subroutine evaluates the initial FDS cost, the total cost of maintenance effort (SCMC), the total cost of preventative maintenance effort (SPMC), the total testing cost (STC) and the total life cycle cost of the system (LCC).

The calculation of these costs is explained in detail in section 7.3.1. The subroutine *Penalties* evaluates penalties for each potential system design by using equations 7.24-7.29. Each penalty is, therefore, added to the system unavailability.

7.5.2 Modifications of Design Construction

The total number of house events in the FDS unavailability fault tree is 59. The routine *House_event_data* formats the house event array, which contains the values of each house event for all population strings. This routine is based on the subroutine *Calc_HE_values,* which calculates house events values for each potential design in accordance with the optimization parameter values (Table 7.4). Table 7.5 represents the house event values evaluation rules.

Table 7.5 House Event Values Evaluation Rules

House Event	Description	Evaluation Rule
h1s	1 PT is fitted	TRUE if $N = 1$
h2s	2 PT are fitted	TRUE if $N = 2$
h3s	3 PT are fitted	TRUE if $N = 3$
h4s	4 PT are fitted	TRUE if $N = 4$
h1ts	1 PT is required to trip	TRUE if $K = 1$
h2ts	2 PT are required to trip	TRUE if $K = 2$
h3ts	3 PT are required to trip	TRUE if $K = 3$
h4ts	4 PT are required to trip	TRUE if $K = 4$
hpt1	PT type 1 is fitted	TRUE if $P = 1$
hpt2	PT type 2 is fitted	TRUE if $P = 2$
hpt3	PT type 3 is fitted	TRUE if $P = 3$
h1p	1 firewater pump is fitted	TRUE if $F = F_E + F_D = 1$
h2p	2 firewater pumps are fitted	TRUE if $F = F_E + F_D = 2$
h3p	3 firewater pumps are fitted	TRUE if $F = F_E + F_D = 3$
h4p	4 firewater pumps are fitted	TRUE if $F = F_E + F_D = 4$
h5p	5 firewater pumps are fitted	TRUE if $F = F_E + F_D = 5$
h6p	6 firewater pumps are fitted	TRUE if $F = F_E + F_D = 6$
h7p	7 firewater pumps are fitted	TRUE if $F = F_E + F_D = 7$
h8p	8 firewater pumps are fitted	TRUE if $F = F_E + F_D = 8$
he0	no electric pumps are fitted	TRUE if $F_E = 0$
he1	1 electric pump is fitted	TRUE if $F_E = 1$
he2	2 electric pumps are fitted	TRUE if $F_E = 2$
he3	3 electric pumps are fitted	TRUE if $F_E = 3$
he4	4 electric pumps are fitted	TRUE if $F_E = 4$

Table 7.5 …continued

he_no1	electric pump number 1 is fitted	TRUE if F_E = 1, 2, 3 or 4
he_no2	electric pump number 2 is fitted	TRUE if F_E = 2, 3 or 4
he_no3	electric pump number 3 is fitted	TRUE if F_E = 3 or 4
he_no4	electric pump number 4 is fitted	TRUE if F_E = 4
hd_no1	diesel pump number 1 is fitted	TRUE if F_D = 1, 2, 3 or 4
hd_no2	diesel pump number 2 is fitted	TRUE if F_D = 2, 3 or 4
hd_no3	diesel pump number 3 is fitted	TRUE if F_D = 3 or 4
hd_no4	diesel pump number 4 is fitted	TRUE if F_D = 4
h501	50% capacity elec. Pump type 1 is fitted	TRUE if F_P= 2 and F_T = 1
h502	50% capacity elec. Pump type 2 is fitted	TRUE if F_P= 2 and F_T = 2
h331	33.3% capacity elec. pump type 1 is fitted	TRUE if F_P= 3 and F_T = 1
h332	33.3% capacity elec. pump type 2 is fitted	TRUE if F_P= 3 and F_T = 2
ha1p	1 AFFF pump is fitted	TRUE if $A = A_E + A_D = 1$
ha2p	2 AFFF pumps are fitted	TRUE if $A = A_E + A_D = 2$
ha3p	3 AFFF pumps are fitted	TRUE if $A = A_E + A_D = 3$
ha4p	4 AFFF pumps are fitted	TRUE if $A = A_E + A_D = 4$
hae0	0 AFFF electric pumps are fitted	TRUE if A_E = 0
hae1	1 AFFF electric pumps is fitted	TRUE if A_E = 1
hae2	2 AFFF electric pumps are fitted	TRUE if A_E = 2
hae_no1	AFFF electric pump number 1 is fitted	TRUE if A_E = 1 or 2
hae_no2	AFFF electric pump number 2 is fitted	TRUE if A_E = 2
had_no1	AFFF diesel pump number 1 is fitted	TRUE if A_D = 1 or 2
had_no2	AFFF diesel pump number 2 is fitted	TRUE if A_D = 2
ha100	AFFF 100% capacity pumps are fitted	TRUE if A_P = 0, i.e. 100%
ha50	AFFF 50% capacity pumps are fitted	TRUE if A_P = 1, i.e. 50%
ha501	50% capacity electric AFFF pump type 1 is fitted	TRUE if A_P = 1 and F_T = 1
ha502	50% capacity electric AFFF pump type 2 is fitted	TRUE if A_P = 1 and F_T = 2
hav1	AFFF valve type 1 is fitted	TRUE if D = 1
hav2	AFFF valve type 2 is fitted	TRUE if D = 2
hav3	AFFF valve type 3 is fitted	TRUE if D = 3
h_old	old material element is fitted	TRUE if C = 1
h_new	new material element is fitted	TRUE if C = 2
hwv1	water spray valve type 1 is fitted	TRUE if W = 1
hwv2	water spray valve type 2 is fitted	TRUE if W = 2
hwv3	water spray valve type 3 is fitted	TRUE if W = 3

The subroutines *Formate_data_unav* and *Formate_data_spur* are responsible for the FDS unavailability and spurious trip fault trees construction for each potential design. This process involves data file formation for each basic event of the fault tree. Since the FDS structure is more complicated compared to the high integrity protection system (HIPS), the initial data structure, initially composed for the HIPS elements, has been modified. These structures are stored in the text files "ued.txt" and "sed.txt" for the unavailability and spurious trip fault trees respectively. Each row represents a basic event and the columns are related to:

- event name,
- type of data (necessary for the BDD construction),
- failure rate (λ),
- mean time to repair (τ),
- unavailability for the events caused by human error (q),
- β parameter for Weibull distribution,
- η parameter for Weibull distribution,
- calculation type (*Calctype*). This parameter identifies the calculation type for each event unavailability (Q) and unconditional failure intensity (w) for later optimization steps. *Calctype* values are integer from the interval [1, 3], i.e.:

  1) if the component is of 'non-wear-out' type, the *Calctype* = 1. Hence,

  unavailability of the component $Q = \lambda\left(\tau + \frac{\theta}{2}\right)$,

  where θ is an inspection interval (obtained within the program), and

  unconditional failure intensity $w = \lambda(1 - Q)$;

  2) if the event occurrence is caused by human error, the $Calctype = 2$:

  $Q = q$;

  3) if the component is of 'wear-out' type, the *Calctype* = 3. In this case the component unavailability is calculated by Weibull distribution, i.e.:

$$Q(t) = 1 - \exp\left(-\left(\frac{t}{\eta}\right)^{\beta}\right), \eta > 0, \beta > 0,$$

and unconditional failure intensity is

$$w(t) = \lambda(t)(1 - Q),$$

where $\lambda(t)$ is a hazard rate function (equation 7.1), β and η are Weibull distribution parameters, and t is equal to 1 year.

- number of hours manual work required to test the component (H_T),
- cost per hour of manual work to test the component (C_{HT}),
- number of hours manual work required to repair the component (C_R),
- cost per hour of manual work to repair failure (C_{HR}),
- cost of spares for each repair carried out (C_{SR}),
- number of hours manual work required for preventative maintenance (H_P),
- cost per hour of manual work for preventative maintenance (C_{SP}),
- number of spares stored (N_S),
- storage cost per component (C_S),
- initial cost (C_I),

The data for each basic event is taken from tables B.3, B.7, B.8, B.9 and B.11 (Appendix B).

7.5.3 Simple GAs and SPEA2 Results Comparison

Similar to GASSOP [Pattison, 1999], the modified ISPEASSOP program was implemented to the FDS optimization with a generated population of 20 strings. In order to compare results, in total 10 runs, 100 generations each, of the modified ISPEASSOP were made. A maximum of 100 generations was allowed along with a mutation rate of 0.01 and crossover rate 0.7. The running time of the program was an order of minutes. Table 7.6 represents the best 10 designs of the FDS obtained after each run and table 7.7 shows the optimization parameter values for each of these designs.

Table 7.6 Best Designs of the FDS after 10 Runs of the ISPEASSOP

Design Variables	ISPEASSOP Run Number									
	1	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**
K/N	1 / 3	1 / 1	1 / 1	3 / 3	1 / 1	2 / 3	2 / 3	1 / 3	2 / 2	2 / 2
P	2	2	1	3	1	2	2	2	3	3
Firewater Supply and Distribution System										
F_E/F	3 / 4	0 / 1	1 / 4	3 / 4	3 / 4	2 / 3	2 / 3	3 / 4	1 / 3	1 / 4
F_P	33.3%	100%	33.3%	33.3%	33.3%	50%	50%	33.3%	33.3%	33.3%
F_T	2	2	2	2	2	2	2	2	1	1
AFFF Supply and Distribution System										
A_E/A	1 / 1	1 / 1	0 / 1	1 / 1	1 / 1	1 / 1	1 / 1	1 / 1	2 / 2	0 / 2
A_P	100%	100%	100%	100%	100%	100%	100%	100%	50%	50%
Valve and Material Types										
W	3	3	2	3	3	2	2	3	1	2
D	3	3	2	3	3	2	2	3	3	2
C	0	0	0	0	0	0	0	0	0	0
Maintenance Intervals										
θ_P	23	15	28	23	16	28	28	23	13	21
θ_R	18	4	5	18	6	4	20	18	14	6
θ_D	15	15	18	15	18	15	15	15	6	9
θ_{PM}	18	9	15	18	12	9	9	18	9	15

Table 7.7 Optimization Parameter Values for Designs from Table 7.6

	STC	$SPMC$	$SCMC$	LCC	F_{sys}	Q_{sys}	Q'_{sys}	$Q'_{sys}-Q_{sys}$
1	7796.4	5813.0	27606.2	102966	0.5593	9.321E-03	**9.321E-03**	0
2	7935.3	6434.8	34240.6	**85561**	0.3220	1.012E-02	1.012E-02	0
3	6767.0	6922.8	29052.2	103342	0.2602	1.078E-02	1.078E-02	0
4	7796.4	5813.0	27606.3	102666	**0.2002**	9.321E-03	**9.321E-03**	0
5	11381.6	8719.4	18403.5	99755	0.2613	9.436E-03	9.436E-03	0
6	6892.4	12862.9	33277.8	117533	0.2033	1.081E-02	1.081E-02	0
7	6423.1	12862.9	33277.8	117064	0.2033	1.078E-02	1.078E-02	0
8	7796.4	5813.0	7606.2	102966	0.5593	9.321E-03	**9.321E-03**	0
9	13766.2	10368.7	9396.6	93582	0.2020	1.360E-02	1.360E-02	0
10	9857.9	7470.3	18265.2	103643	0.2022	1.284E-02	1.284E-02	0

There are three main FDS optimization parameters. They are the total life cycle cost (LCC), the spurious trip frequency (F_{sys}) and the penalized system unavailability (Q'_{sys}). Results (Table 7.7) show that all best designs satisfy the limitations from table 7.2, since the difference between the original system unavailability (Q_{sys}) and the penalized system unavailability is equal to zero for all obtained designs. The lowest life cycle cost (LCC = 85561) was produced by run 2. The smallest spurious trip frequency appears in run 4. The smallest system unavailability was obtained in runs 1, 4 and 8. Therefore, the fourth design can be considered as the best after 10 runs, since it dominated other designs in terms of the system unavailability and the spurious trip frequency.

Figures 7.2, 7.3 and 7.4 represent the graphical comparison of the firewater deluge system LCC, F_{sys} and Q'_{sys} obtained by the simple GAs [Pattison, 1999] and SPEA2 optimization algorithms respectively (Table 7.7). The circles on each graph correspond to the fittest design parameter values obtained by SPEA2. The dotted and solid lines indicate the maximum and minimum parameter values obtained by simple GAs respectively.

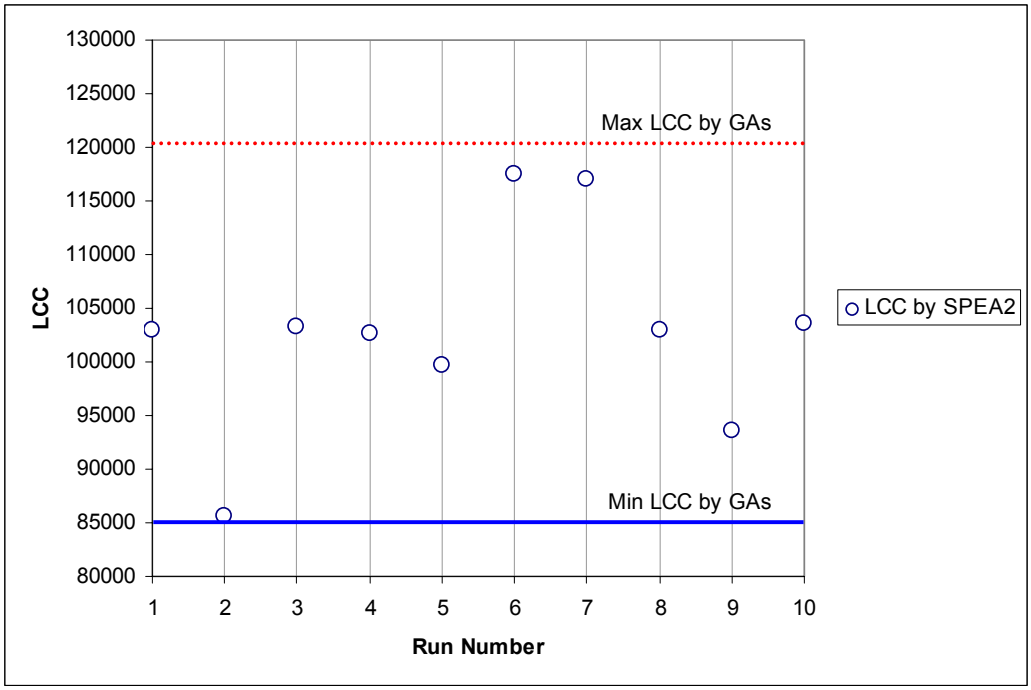


Figure 7.2 Comparison of the LCC obtained by GAs and SPEA2

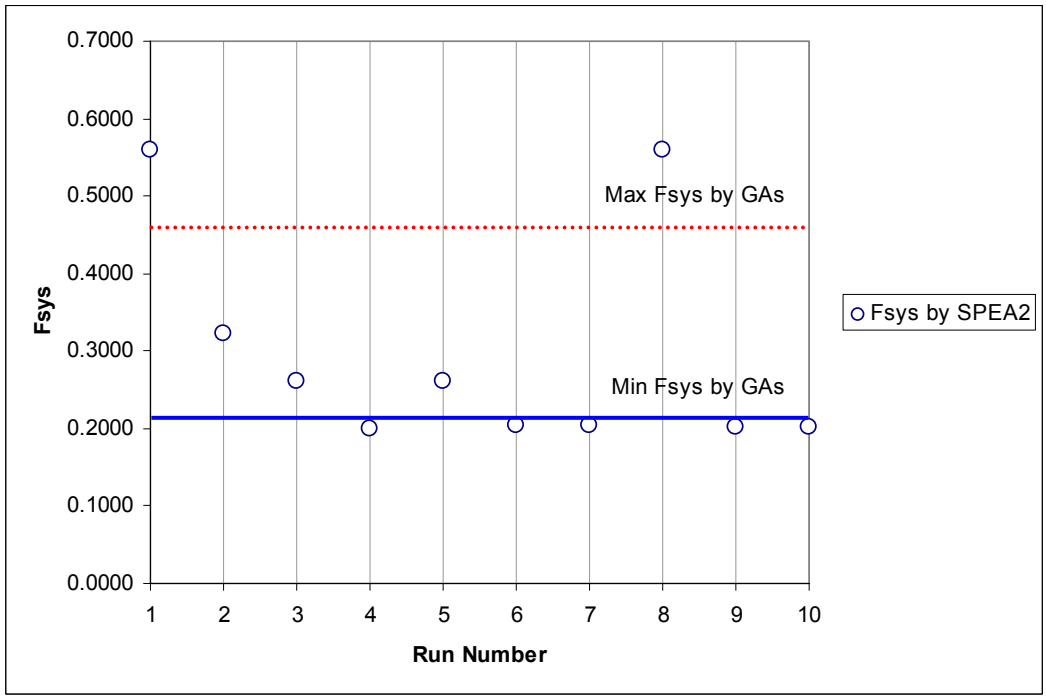


Figure 7.3 Comparison of the F_{sys} obtained by GAs and SPEA2

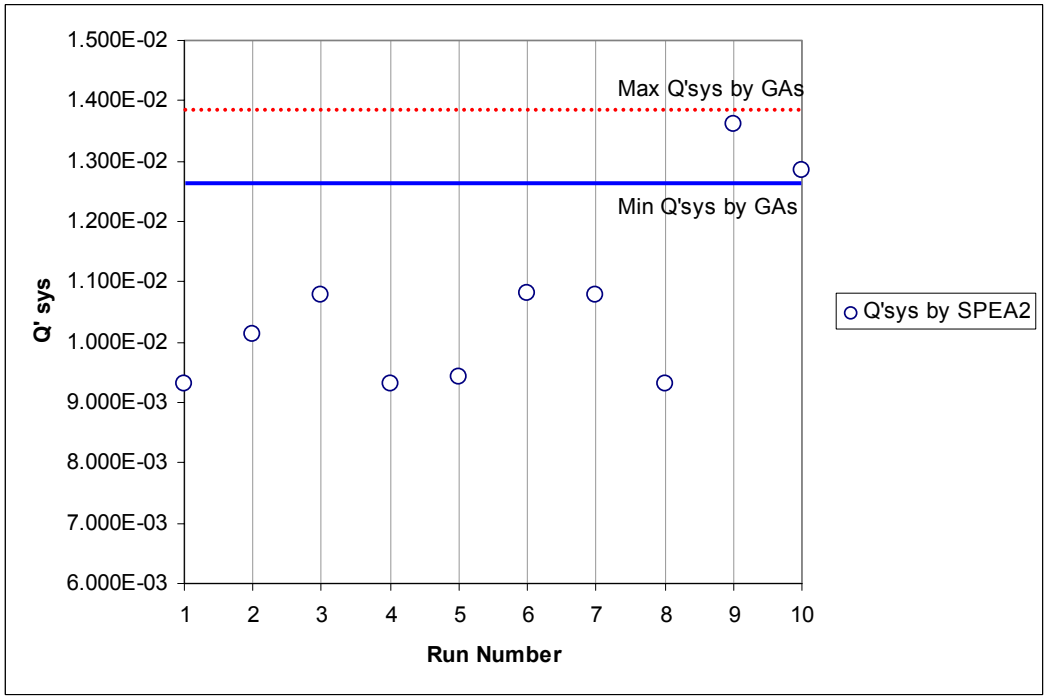


Figure 7.4 Comparison of the Q'_{sys} obtained by GAs and SPEA2

Figures 7.2-7.4 show that, similar to the HIPS system optimization, the SPEA2 algorithm, implemented to the FDS system, finds nondominated solutions faster than the simple GAs for a relatively small number of generations. The life cycle cost (Figure 7.2) is in the same range of values as for simple GAs for all designs. Five runs resulted

in a 10% smaller spurious trip frequency compared to the simple GAs results (Figure 7.3), however, two designs have 25% higher spurious trip frequency. The penalized system unavailability obtained by the SPEA2 method is 15% smaller for the majority of designs. Since the smallest value of this parameter has been a goal for the simple GA, such difference indicates that the SPEA2 based technique is more suitable for the complex safety system optimization with a relatively small number of generations.

7.6 Summary

- The modified SPEA2 based optimization tool, discussed in previous chapter, has been modified further for the application to the firewater deluge system (FDS). The following modifications have been done:

    a) the new routine for the house event logic interpretation has been incorporated into the program structure;

    b) the Weibul distribution has been used to represent failures of the wear out type components in the system unavailability and spurious trip frequency evaluation routines;

    c) the SPEA2 search has been amended to reflect the new number of optimization parameters;

    d) additional limitations to the system design have been incorporated for all maintenance costs (i.e. life cycle cost per year, cost per year due to system testing, cost per year due to preventative maintenance, cost per year due to corrective maintenance, cost per year due to total maintenance effort);

    e) the life cycle cost and maintenance down time calculation procedures have been changed in order to match the FDS features.

- The modified tool has been successfully applied to the FDS and, similar to the HIPS optimization case, found optimal designs faster than the simple GA.

- Another important advantage of the SPEA2 method is that it is much faster and requires less computer resources comparing to the simple GA. One run of the GASSOP program takes several hours. On the other hand, one run of the modified ISPEASSOP takes only 12 minutes. Such a difference can be explained by the advanced features of the SPEA2 method and the developed fault tree reduction technique which reduces the BBD size for each potential system design and, hence, results in a shorter program running time.

- It is not envisaged that increasing number of constraints or design options will restrict the method and, hence, the developed technique has a potential for application to any safety system optimization. Further program improvements are discussed in chapter 8.

CHAPTER 8

OPTIMIZATION OF HIPS WITH DEPENDENCIES

8.1 Introduction

Previous chapters illustrate two examples of offshore safety system optimization by the developed algorithm. This algorithm comprises such well-known techniques as fault tree analysis and the binary decision diagram method. The use of these methods is adequate only if all basic events occur independently, since the techniques are not able to take into account the dependencies. It is rare that real safety systems consist of just independent components in terms of failure or (and) repair. Failure to identify the dependency in the system would result in an incorrect system unavailability and failure frequency prediction. Therefore, an appropriate modelling technique is required to overcome the problems caused by dependency between components during the system unavailability and failure intensity evaluation process.

This chapter introduces the modified optimization technique. The new methodology enables optimization of safety systems with dependencies by effective use of the Markov modelling tool [Andrews and Moss, 2002]. Different component dependency types are overviewed in section 8.2. Section 8.3 discusses the system dependency modelling technique and also details the code produced to carry out the optimization method. The technique application to the HIPS is illustrated in section 8.4 [Riauke and Bartlett, 2009 (2)]. The discussion of the results and their potential improvement is provided at the end of the chapter.

8.2 Dependency Types

Prior to the optimization process, the safety system dependency groups should be identified and numbered. The group number is important since it plays an identity tag role in the initial stage of analysis. This section overviews the main types of dependency which are frequently encountered in many safety systems [Sun, 2006].

Maintenance Dependency: This type of dependency is common for all safety systems. It arises from the situation where one maintenance engineer or a team of engineers has to take responsibility for a group of components usually of the same or similar type. If several components from the same maintenance group fail subsequently, only one of them goes through the repair process. Others wait in a queue for repair until the engineer has restored the first component. The queue may be random or prioritized if some failures are more critical than others. In both cases the queuing affects repair times and, hence, the probability of failure of other components. Therefore, the maintenance dependency affects the whole system and influences the system unavailability and failure intensity.

Standby Dependency: Redundancy or diversity is commonly used in most safety system designs in order to reduce the chance of system failure. It can be achieved by use of standby systems. These systems are activated by the primary system failure and take over its duty. The likelihood of the standby component failure increases as it experiences the operational load due to the failure of the primary operating component. Therefore, it is important to take into account the standby dependency for the correct system unavailability and failure intensity prediction.

Secondary-failure Dependency: This dependency exists in systems designed of components whose failures may be caused by their *primary* or *secondary* sources. If the component fails under expected operating conditions its failure is classified as primary. On the other hand, the secondary failure is caused by failures of other system components. Frequently, the secondary failure results in a change of the working environment of the primary component and this change causes the primary component to fail.

Initiator-enabler Dependency: The initiating event is the event which every safety system is designed to mitigate. Hence, the occurrence of such an event activates the safety system. The failure of the safety system is a general enabling event. The occurrence of the enabling event produces conditions under which the initiating event causes a hazard. For a complex system, it is a difficult task to define initiator-enabler dependency between system components.

Test Dependency: Regular inspection is essential for every safety system in order to reveal dormant failures of its components. Some safety systems may be divided into relatively independent sub-systems. In this case the inspection is carried out not only for individual components but for the whole sub-system. The inspection interval can be the same or different for the sub-system and its components. The difference in these intervals results in the test dependency between sub-system components, since for some components dormant failures may be revealed during the sub-system inspection. Therefore, this affects the components actual downtime. The test dependency influences only the individual component failure probability and has no impact on the rise of statistical dependency between component failures. However, it is still important to identify and specify this dependency for accurate system unavailability and failure intensity prediction.

8.3 System Dependency Modelling Technique

In 2006 Huiling Sun [Sun, 2006] proposed a new system dependency modelling technique. The method includes two main steps. The first is fault tree simplification and modularization in order to obtain independent modules containing specific dependency groups. Then these modules go through Markov analysis [Andrews and Moss, 2002]. As a result, the conventional fault tree structure is maintained. This methodology has been integrated into the optimization algorithm for safety systems with dependencies in order to enable the use of fault tree analysis and binary decision diagrams, since these techniques are the major parts of the developed optimization tool. This section provides an overview of the main aspects of this method with accent on the fault tree modularization technique (section 8.3.1) and Markov modelling (section 8.3.2).

8.3.1 Fault Tree Modularization

One of the main parts of the dependency modelling technique is Markov analysis of the dependent parts of the fault tree. However, one of the weaknesses of the Markov method is that the model size grows exponentially with the number of components. Therefore, it is essential to simplify the fault tree structure and divide it into the smallest

independent parts containing separated dependency groups. The whole process can be broken into the following steps [Sun, 2006]:

Step 1. Reorganization of the dependency information: This step is essential since the overlap between different dependencies may exist in the systems containing more than one dependency group. In this case, the same event may appear in more than one record in the dependency file. Therefore, it is important to ensure that all events which share a dependency relationship with the event, repeated in several dependency categories, will be included in the same *serial*. The resulting record no longer represents a single dependency relationship and the original dependency group number is no longer an appropriate identity tag. Consequently, a dependency serial number is allocated to distinguish the group of dependent events.

Step 2. Fault tree simplification: This step is based on the reduction technique used in the Faunet code [Platz and Olsen, 1976]. It provides a good framework to reduce the fault tree to its minimal form. Three main stages of the fault tree simplification method are *contraction*, *extraction* and *factorisation*. During the contraction stage gates of the same type (i.e. OR or AND) are contracted to form a single gate. The extraction stage is used to identify and extract the common event (factor) from the gate structure. During the factorisation stage pairs of independent events that always occur together in the same gate type are identified and combined to form a single complex event.

Step 3. Dependency information formation: This step allocates the dependency serial numbers for each gate in the fault tree structure. The dependency of each gate is defined by a list of all dependency serial numbers the gate basic events belong to.

Step 4. Combination of dependent events: The purpose of this step is to restructure the fault tree in order to separate the events from the same serial into separate branches. Using the information generated in the previous step, each gate is examined in turn. The combined new gates lead to a fault tree structure with the smallest independent sub-trees for each dependency.

Step 5. Fault tree modularization: This step is based on the algorithm developed by Rauzy [Rauzy and Dutuit, 1996]. The algorithm performs a step-by-step traversal

recording each gate and event of the fault tree. It fixes the step number at which the first, second, and final visits to each node were made. Each fault tree part is traversed only once. Therefore, when the gates appear more than once in the tree, only its first appearance is traversed completely. All other appearances of these gates are treated like basic events. In order to ensure that dependent basic events featuring the same dependency serial will end up in the same module, they are treated as a single basic event with the same label during the first traversal. All events from the same dependency group will then be replaced with a symbol characterising the particular dependency group.

The principle of the modularization algorithm is that if the first visit step number to any of the gate descendants is smaller than the first visit step number to this gate, then such a descendant must also occur beneath another gate in the fault tree structure. Similarly, if the last visit to any of the gate descendants is greater than the second visit number to the gate, then again it must occur elsewhere in the tree. Therefore, the gate can be classified as a module only if it satisfied two conditions:

- the first visit to each descendant is after the first visit to the gate;
- the last visit to each descendant is before the second visit to the gate.

The second pass through the fault tree structure is needed in order to assess these conditions. The maximum (Max) of the last visits and the minimum (Min) of the first visits of all the descendants for each gate will be obtained based on the result of the first traversal. After this search all modules are identified and tagged for future analysis.

Step 6. Dependency information update: By this step all independent sub-trees (modules) are identified and tagged. However, not all of them can go through the Markov analysis. It is essential that each module has a mutual dependency, i.e. all its dependant descendants should belong to the same dependency serial. This step is designed to collect and store such information.

Step 7. Re-modularization of each dependency relationship: The previous step identifies all modules with mutual dependency. This step checks if the module is minimal and, hence, satisfies the final condition to be analyzed by the Markov method. Firstly, it has

to be determined whether a certain module is already the smallest one for the given dependency. This is accomplished by using the dependency information provided by the preceding step. If any module contains a given dependency serial number and its mutual dependency includes the given dependency serial, it can be concluded that this module is already the smallest module and does not need further processing. If the module does not satisfy this condition, the search for the smallest independent sub-tree of the given dependency serial continues within this module. The search process is divided into three stages. To illustrate this technique consider the example module in figure 8.1. The module (*Gate1*) has two dependency serials: the serial one is represented by events *A* and *B* coloured in grey, the serial two consists of events *D* and *E* with a striped pattern.

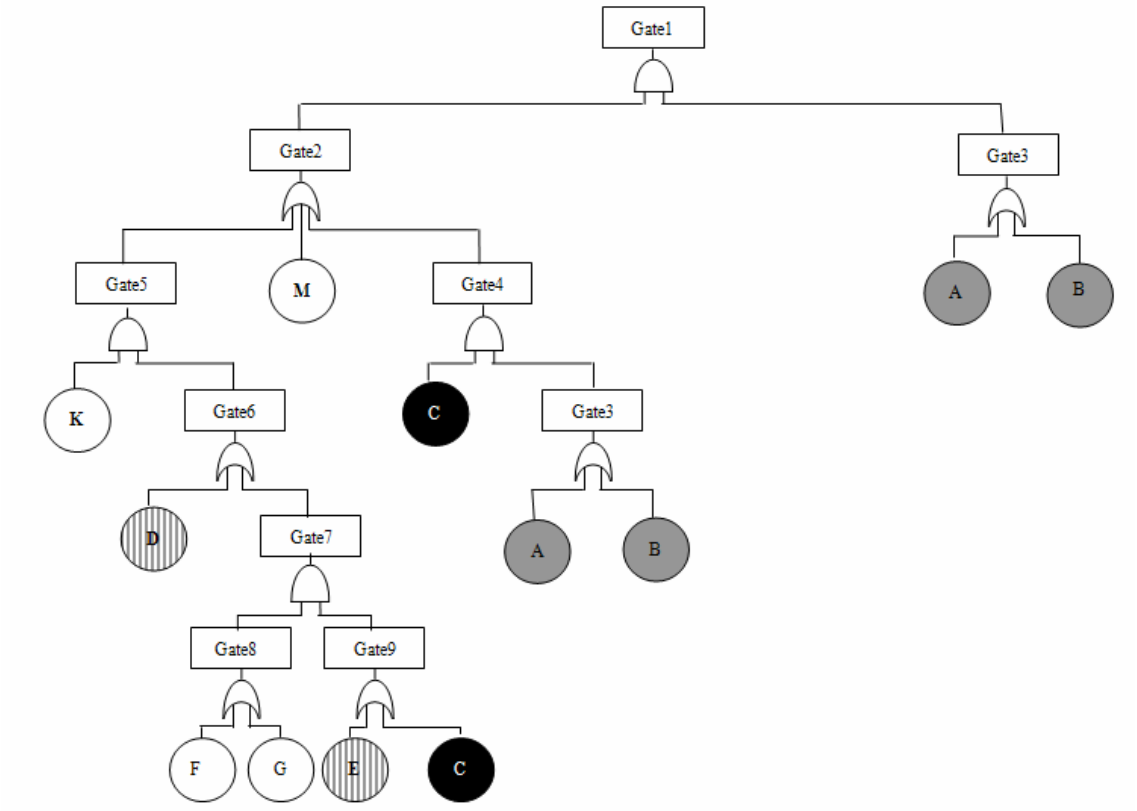


Figure 8.1 Example Module

Stage 1. Traverse the module from its top event until the descendant gate with the same mutual dependency number is found. Regarding the example module *Gate1* from figure 8.1, the path for the dependency serial one is: *Gate1, Gate3*. The *Gate3* is a minimal module (labelled as *Mod1*), since all its descendants occur only under this gate in the module *Gate1*. Therefore, the algorithm stops for the dependency serial one at this point. The path for the dependency serial two will be: *Gate1*,

Gate2, *Gate5* and *Gate6*. The gate *Gate6* is the first gate in this path which mutual dependency includes dependency serial two.

Stage 2. The gate, found in stage 1, would lead to the smallest independent sub-tree for the given dependency serial if this gate has been identified as module. If the gate is not a module, some of its descendants occur elsewhere in the initial module and, therefore, prevent it from being a module. Hence, at this stage all those preventing components should be identified. For example, the event *C* (coloured in black) is the preventing element under the *Gate6*, since it occurs only once under the *Gate6* but occurs twice in module *Gate1*.

Stage 3. After all preventing elements have been detected, it is essential to identify a new module which includes those preventing components. If the new potential module contains any preventing elements, the procedure repeats until the independent module is found. Considering the example from figure 8.1, the path upwards the preventing component *C* is: *C*, *Gate4* and *Gate2*. The search stops at the *Gate2*, since it is the first gate which also appears in the descending path for the *Gate6*. It indicates that *Gate4* and *Gate5* are both immediate descendants of *Gate2* and that *Gate4* contains the preventing component *C*. Hence, the potential module is now updated to include the combination of *Gate5* and *Gate4*. In the new potential module no preventing elements have been identified. Therefore, the combination of *Gate4* and *Gate5* is a new module, labelled as *Mod2*, and is a smallest for dependency serial two. The resulting example module structure is shown in figure 8.2.

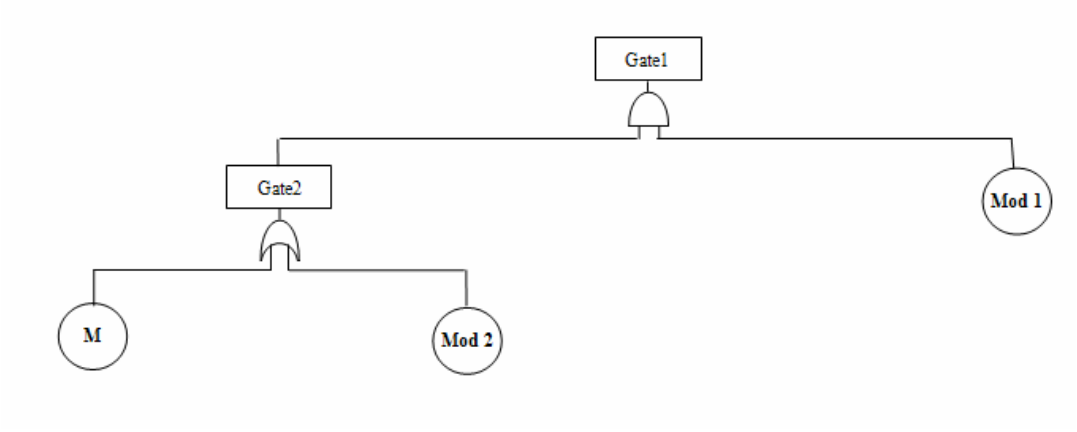


Figure 8.2 Resulting Example Module Structure

When the fault tree modularization into independent sections is finished, the analysis of these independent sub-trees will be carried out by the Markov method, if the section contains dependent components; or by the binary decision diagram, if all sub-tree components are independent. If the dependent component groups could not be separated even after the use of the suggested technique, the whole fault tree is treated as a module and goes through the Markov analysis.

8.3.2 Markov Model Generation

Incorporation of the Markov method into conventional fault tree analysis is an alternative solution to the assessment of systems which contain a dependency relationship. The application of the Markov method is realized through the generation of the Markov model. It represents the characteristics of the particular dependency relationship included in the independent fault tree module.

A complete Markov model is composed of three elements: a list of system states, the transition between these states and corresponding transition rate. Consider the example from figure 8.3.

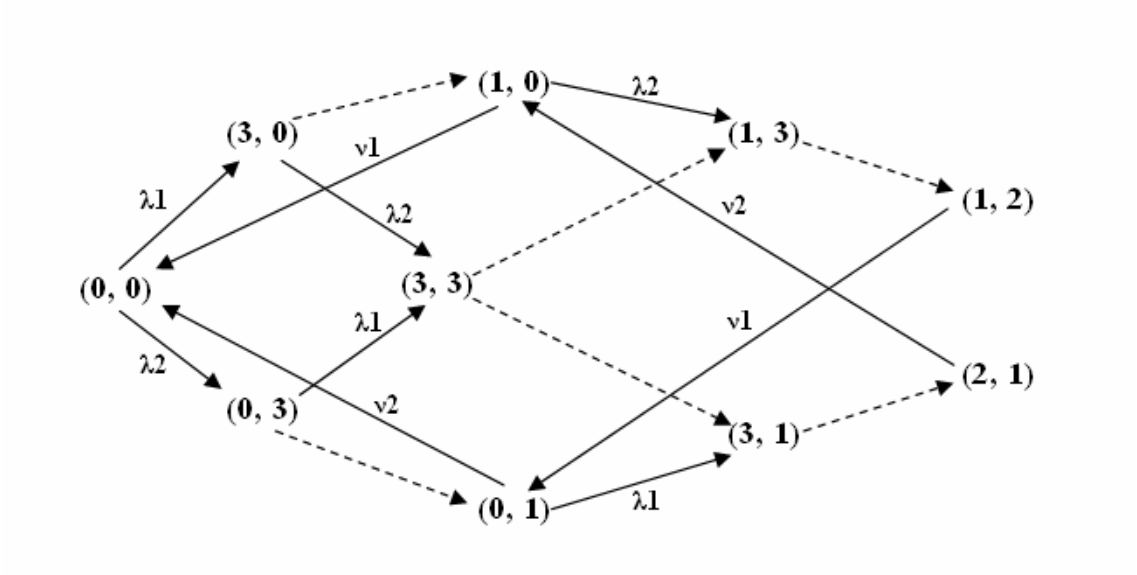


Figure 8.3 Example Markov Model

Figure 8.3 represents a Markov model for the module which contains only two dependent system components defined as (x, y). Each component may be in one of four

possible states, i.e. 0, 1, 2, or 3. State 0 corresponds to the element working condition. State 1 means that the component is failed, however, its failure has been revealed and the component is currently under repair. Similarly to state 1, state 2 means that the component has failed and this failure is already revealed, however the component waits in the queue for repair. State 3 corresponds to an unrevealed element failure. Four individual component states result in ten possible system states for the Markov model. The transitions between these states are represented by arrows. Solid arrows indicate failure/repair of one of the basic system components. The corresponding transition rate is the failure rate or repair rate of the relevant component. A basic assumption of the analysis method is that only one transition is possible in a small time interval. For example, the system transits from state (0, 0), when both components are in the working condition, to the state (3, 0), when the first component is in failure condition, with transition rate λ_1 corresponding to the first basic event failure rate.

The dotted arrows represent the interval between the actual component failure time and the time this failure is revealed. Unlike other state transitions which are caused by component failure or repair, the occurrence of these inspection transitions are mandatory and instantaneous at the specific points of time. The transitions represented by the dotted arrows will not occur until the inspection is performed.

The manual development of the Markov model, due to their size, can be error-prone. Therefore, it should be generated automatically. In order to represent the correct dependency relationship, the unique Markov model should be developed for each dependency type [Sun, 2006]. This section provides an example of Markov model generation for the module with maintenance dependency between components.

Markov model generation for the maintenance dependency: Consider the example module from figure 8.4. Assume that there is only maintenance dependency between basic events *A* and *B*.

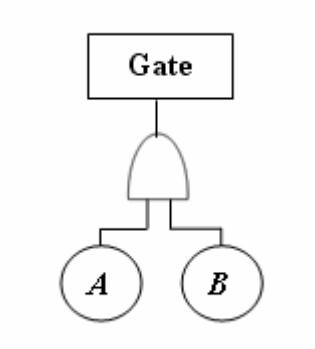


Figure 8.4 Example Module

The development of the Markov model requires an initial system state. In most cases, this state is configured with each component working normally, i.e. (0. 0). It means that in the initial system state no basic events have occurred. With the initial state established, the Markov model can be gradually developed by systematically considering the transitions that make each component state change. The Markov model is complete when all possible transitions from the generated system states result in states which are already included in the module. Figure 8.3 represents the general Markov module scheme for the example module. Depending on the maintenance time interval values for the components *A* and *B*, i.e. θ_A and θ_B, this general scheme may produce three different Markov models for the example module. The main difference occurs in the transitions from the system state (3, 3), when both components are in the unrevealed failure condition:

Case 1. If θ_A is smaller than θ_B, component *A* failure will be revealed first and it will go through the repair process. Therefore, the system transition path will be:

$$(3, 3) \rightarrow (1, 3) \rightarrow (1, 2).$$

Case 2. If θ_A is greater than θ_B, component *B* failure will be revealed first and the system transition path will be:

$$(3, 3) \rightarrow (3, 1) \rightarrow (2, 1).$$

Case 3. If θ_A is equal to θ_B, both component failures will be revealed at the same time. In this case, the safety system maintenance engineer will choose the more important component to be repaired first. If components have no clear repair priority, it is assumed that they will be repaired in the same order as they appear in the module, i.e. from left to

right. Hence, the transition sequence will be similar to case 1 or case 2 depending on that order.

When the number of the module components is greater than two, the complexity of this scheme grows rapidly. Therefore, a systematic approach is needed. Sun [Sun, 2006] suggested the following algorithm to ensure that each possible state transition is taken into consideration when two or more dormant (unrevealed) failures exist in the system:

> **Step1**. Calculate the common multiple θ_{com} for the inspection interval of the basic events which in the current state are tagged with unrevealed failure code '3'.
>
> **Step2**. Consider the first basic event with the current state '3'. Start with its inspection interval θ_i. Establish the possible state transitions when time point $T = \theta_i,\ 2\theta_i,\ 3\theta_i, \ldots,\ \theta_i$. At each of these points of time, make the new state of the basic event to be revealed, i.e. change from '3' to '1'. Meantime record the time condition for each transition.
>
> **Step 3**. Consider the next basic event with state '3'. Repeat step 2 until every basic event with state '3' has been examined. Points of time which have been considered with the previous basic events will be skipped.

The generation algorithms for other dependency types are discussed in chapter 9.

8.3.3 Quantitative Analysis for Safety Systems with Dependencies

This section discusses how the quantitative analysis is carried out in the identified modules using different techniques and how each module results are interpreted within the overall fault tree structure in order to obtain the final system unavailability and failure intensity.

After the fault tree modularization and Markov model generation for each independent module, all Markov models should be quantified in order to obtain the relevant reliability parameters. The Markov model provides the exhaustive list of all possible system states. The module can experience different system states at different times during its working period. The total probability of the system residing in all of these

states is equal to 1. The system states can be split in two groups: those states in which the system functions normally; and states which lead to the top event occurrence due to specific failure. Hence, the module failure probability can be obtained by summing up the probabilities of all system states belonging to the second category. Alternatively, the probability of the module functioning is equal to the sum of the first category states probabilities.

Consider a general Markov model consisting of n system states, of which states 1, 2,…, k are working system states and states k + 1, k +2, …, n are failed system states. Therefore, the module failure probability, $Q_M(t)$, can be expressed by equation 8.1 [Andrews and Moss, 2002]:

$$Q_M(t) = \sum_{i=k+1}^{n} Q_i(t), \tag{8.1}$$

where $Q_i(t)$ is the probability of system residing in state i at time t. Considering the probability that the module is residing in state i at time $t+dt$, two situations should be taken into account: first, the module could be in state j at time t and, hence, transit to state i during time interval dt; second, the module could be in state i at time t and it remains in the same state during the interval dt. Therefore:

$$Q_i(t+dt) = \sum_{\substack{j=1 \\ j\neq i}}^{n} Q_j(t)\cdot a_{ji}dt + Q_i(t)\cdot\left[1-\sum_{\substack{i=1 \\ i\neq j}}^{n} a_{ij}dt\right]. \tag{8.2}$$

Since $dt \to 0$, equation 8.2 can be simplified. Its matrix form is shown in equation 8.3.

$$\left[\frac{dQ_1}{dt}, \frac{dQ_2}{dt}, ..., \frac{dQ_n}{dt}\right] = \left[Q_1(t), Q_2(t), ..., Q_n(t)\right]\cdot\left[A\right], \tag{8.3}$$

where matrix [A] is the state transition matrix with elements a_{ij} corresponding to transition rate from state i to state j, and $a_{ii} = -\sum_{\substack{i=1 \\ i\neq j}}^{n} a_{ij}$. This matrix is formulated directly from the Markov model using the following rules:

Rule 1. [*A*] is a square matrix with the number of components equal to the number of states in the model.

Rule 2. All rows summate to zero.

Rule 3. Component a_{ij} represents the transition rate from state *i* to state *j*.

Rule 4. A diagonal element a_{ii} is the transition rate out of state *i* (always a negative value).

The Markov model has two types of solution: the *steady-state* and the *transient*. When the system is in the steady-state condition, the probability of being in any other state will not change with time. These probabilities are independent from the initial system state. Hence, the steady-state solution for each state in the Markov model can be obtained by equation 8.4:

$$Q_i(\infty) = \frac{\begin{vmatrix} a_{11} & a_{12} & \dots & a_{1,n-1} & 0 \\ & & & & \text{M} \\ a_{i1} & a_{i2} & \dots & a_{i,n-1} & 1 \\ & & & & \text{M} \\ a_{n1} & a_{n2} & \dots & a_{n,n-1} & 0 \end{vmatrix}}{\begin{vmatrix} a_{11} & a_{12} & \dots & a_{1,n-1} & 1 \\ & & & & \text{M} \\ a_{i1} & a_{i2} & \dots & a_{i,n-1} & 1 \\ & & & & \text{M} \\ a_{n1} & a_{n2} & \dots & a_{n,n-1} & 1 \end{vmatrix}}. \tag{8.4}$$

The module failure probability, $Q_M(\infty)$, therefore, can be given by equation 8.5:

$$Q_M(\infty) = \frac{\begin{vmatrix} a_{11} & a_{12} & \dots & a_{1,n-1} & 0 \\ & & & & \text{M} \\ a_{i1} & a_{i2} & \dots & a_{i,n-1} & 1 \\ & & & & \text{M} \\ a_{n1} & a_{n2} & \dots & a_{n,n-1} & 1 \end{vmatrix}}{\begin{vmatrix} a_{11} & a_{12} & \dots & a_{1,n-1} & 1 \\ & & & & \text{M} \\ a_{i1} & a_{i2} & \dots & a_{i,n-1} & 1 \\ & & & & \text{M} \\ a_{n1} & a_{n2} & \dots & a_{n,n-1} & 1 \end{vmatrix}}. \tag{8.5}$$

The transient solution provides the probability of the system residing in any of the states at any specific point of time t by progressing from the initial system state in insignificant time steps, dt. In this case, starting from time $t = 0$ and using equation 8.3, the state probabilities at discrete time points with constant interval dt can be obtained as:

$$\begin{aligned} &[Q_1(dt), Q_2(dt), \dots, Q_n(dt)] = [Q_1(0), Q_2(0), \dots, Q_n(0)] \cdot [P], \\ &[Q_1(2dt), Q_2(2dt), \dots, Q_n(2dt)] = [Q_1(dt), Q_2(dt), \dots, Q_n(dt)] \cdot [P], \\ &\qquad \dots \\ &[Q_1(mdt), Q_2(mdt), \dots, Q_n(mdt)] = [Q_1((m-1)dt), Q_2((m-1)dt), \dots, Q_n((m-1)dt)] \cdot [P], \end{aligned} \tag{8.6}$$

where [P] is a square transition probability matrix with components $p_{ij} = a_{ij} \cdot dt$, for $i \neq j$; and $p_{ii} = 1 + a_{ii} \cdot dt$, where $a_{ii} = -\sum_{\substack{i=1, \\ i \neq j}}^{n} a_{ij}$.

The states with dormant failures require additional assumptions. Consider the transition from state i to state j that occurs after the regular inspection interval θ. Then if $dt = \theta, 2\theta, \dots, m\theta$:

$$\begin{aligned} &Q_j'(dt) = Q_i(dt) + Q_j(dt), \\ &Q_i'(dt) = 0, \end{aligned} \tag{8.7}$$

where $Q^{/}(dt)$ represents the updated state probability at the point in time immediately following the inspection.

The module unconditional failure intensity, $w_M(t)$, for the general Markov model can be expressed as:

$$w_M(t) = \sum_{i=1}^{k} Q_i(t) \cdot \left(\sum_{j=k+1}^{n} a_{ij}(t) \right), \tag{8.8}$$

where state *i* corresponds to the working module state and $a_{ij}(t)$ is the transition rate from state *i* to state *j* in which the module is failed.

8.4 HIPS with Dependency Optimization

The dependency overcoming techniques (FT modularization and Markov Analysis), discussed in section 8.3, have been incorporated into the developed ISPEASSOP optimization tool. The ISPEASSOP modified version has been applied to the HIPS optimization. This section provides the details of this application. The schematic structure of the modified ISPEASSOP program is shown in figure 8.5.

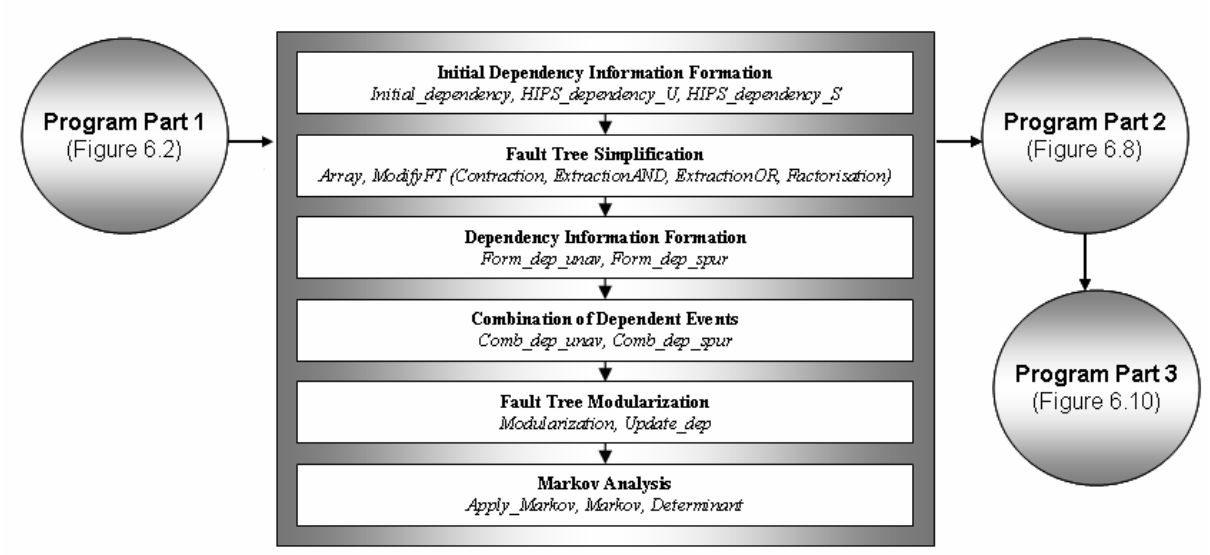


Figure 8.5 Modified ISPEASSOP Program Schematic Structure

In figure 8.5 program parts one, two and three refer to the ISPEASSOP program parts previously discussed in chapter 6 and shown in figures 6.2, 6.2 and 6.10 respectively. Sections 8.4.1 and 8.4.2 describe the new part of the program.

8.4.1 HIPS Dependency Groups

In the HIPS the only dependency type present is the maintenance dependency. In total eight dependency groups have been identified for the general HIPS structure, four for each fault tree (i.e. unavailability and spurious trip). These groups are summarised in table 8.1. It is assumed that each group is maintained by one engineer.

Table 8.1 HIPS Dependency Groups

Group No	Event Name	Component Description	Fault Tree Type
1	pt11, pt12, pt13, pt14, pt15, pt16, pt17, pt18	Pressure transmitters of type 1	Unavailability
2	pt21, pt22, pt23, pt24, pt25, pt26, pt27, pt28	Pressure transmitters of type 2	
3	wv, mv, esd11, esd12, hips11, hips12	Subsystem 1: wing valve, master valve, ESD valves of type 1. Subsystem 2: HIPS valves of type 1	
4	wv, mv, esd21, esd22, hips21, hips22	Subsystem 1: wing valve, master valve, ESD valves of type 2. Subsystem 2: HIPS valves of type 2	
5	spt11, spt12, spt13, spt14, spt15, spt16, spt17, spt18	Pressure transmitters of type 1	Spurious Trip
6	spt21, spt22, spt23, spt24, spt25, spt26, spt27, spt28	Pressure transmitters of type 2	
7	wvs, mvs, sesd11, sesd12, ships11, ships12	Subsystem 1: wing valve, master valve, ESD valves of type 1. Subsystem 2: HIPS valves of type 1	
8	wvs, mvs, sesd21, sesd22, ships21, ships22	Subsystem 1: wing valve, master valve, ESD valves of type 2. Subsystem 2: HIPS valves of type 2	

The dependency data from table 8.1 is stored in the text file "*HIPSdept.txt*". This file has a row-column structure. Each row represents a dependency group and the columns are related to:

- dependency group name,
- group dependency type (*mt* for maintenance dependency type),
- number of dependent components,
- number of maintenance engineers for this group,
- array of dependent components.

The dependency information from table 8.1 is formed in the *Initial_dependency* routine. The program reads the data from the file *"HIPSdep.txt"* into the data array *DP*, which structure is similar to the initial data file.

8.4.2 Technique Application to HIPS Optimization

To demonstrate the application of the modified optimization tool to the HIPS, consider an example possible HIPS design, shown in figure 8.6.

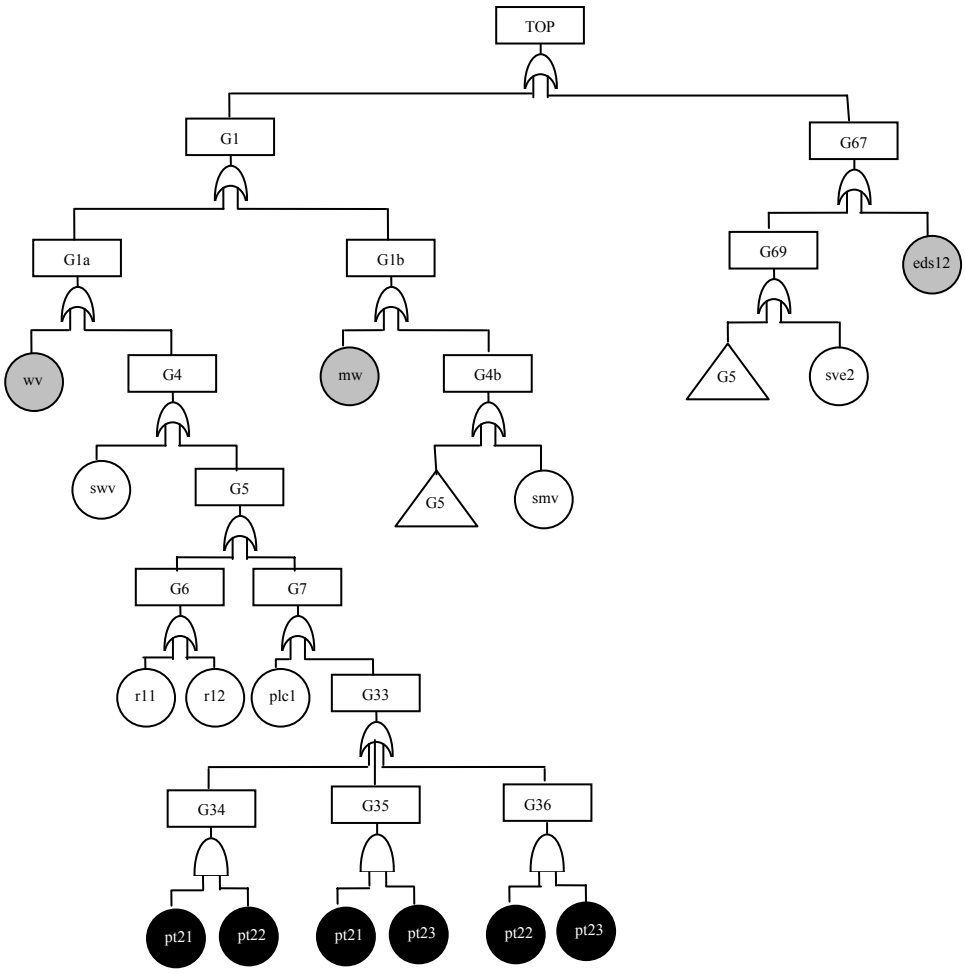


Figure 8.6 Example HIPS Design

Step 1. Dependency Information Formation: The subroutines *HIPS_dependency_U* and *HIPS_Dependency_S* identify the dependency groups and number of dependent components for each potential HIPS design by comparing the design fault tree events to the information stored in the dependency array *DP*. The example HIPS design contains only two dependency groups from table 8.1. These groups are:

Group 2: pt21, pt22, pt23.
Group 3: wv, mv, esd12.

In figure 8.3 the second and third dependency groups components are coloured in black and grey respectively. In this example there are only two dependency serials: serial one and two are equal to the dependency group 2 and 3 respectively.

Step 2. Fault Tree Simplification: The first step of the fault tree simplification process starts with the contraction technique, implemented in the subroutine *Contraction*. In the example HIPS design fault tree, shown in figure 8.6, all gates except for *G34*, *G35* and *G36* have the same type as the *Top* gate and, hence, should be eliminated from the fault tree structure. The resulting fault tree is shown in figure 8.7.

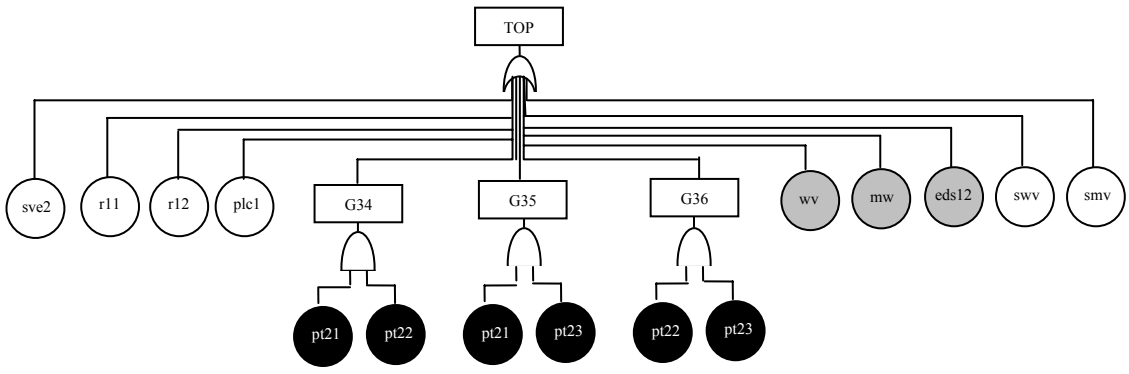


Figure 8.7 Example HIPS Design FT after Contraction

The algorithm for the contraction step is shown in figure 8.8.

```
Contraction
{
    go through all gates in the FT structure:
    {
        IF (the gate has input gates)
           FOR (the first to the last input gate)
                check the input gate type;

        IF (all input gate types are equal to the gate type)
         {
            replace the gate with its input gates;
            remove the gate from the FT structure;
          }

        check the FT structure for repeated gates and events;
        delete repeated components if necessary.
    }
}
```

Figure 8.8 The Algorithm for the Contraction Step

The contraction step is followed by the extraction step, implemented in the routine *Extraction*, which includes the subroutines *ExtractionAND* and *ExtractionOR* for AND and OR gate types respectively. In this step the common events are extracted from the same type gates. The program analyses the fault tree structure from left to right, hence, the first two gates to be considered are *G34* and *G35*. These gates are of the same type (AND) and have a common event *pt21*. Hence, the subroutine *ExtractionAND*, will transform the example fault tree structure as shown in figure 8.9. The extraction procedure stops at this point, since there are no more input gates from the same gate with common events.

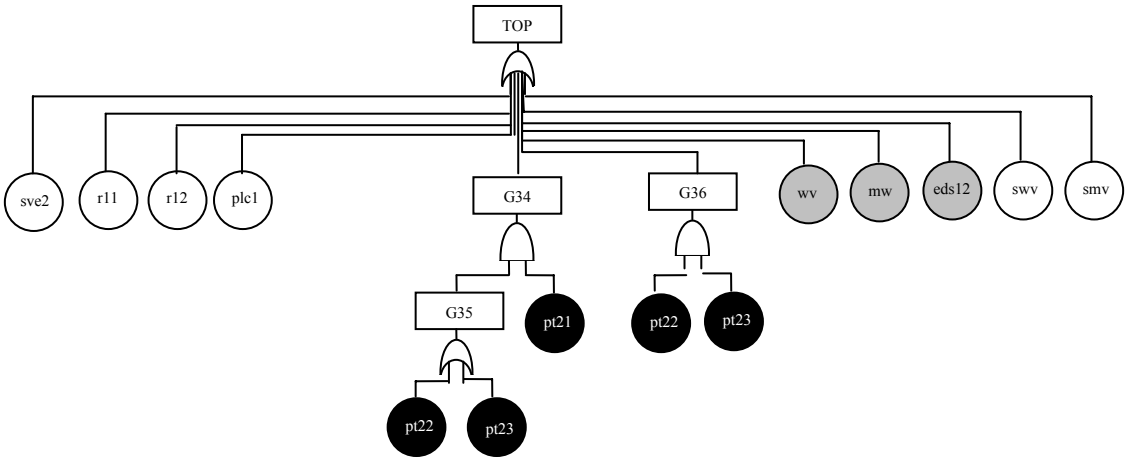


Figure 8.9 Example HIPS Design FT after Extraction

The algorithm for the extraction step is shown in figure 8.10.

```
ExtractionOR/AND
{
      go through all gates in the the FT structure:
            IF (the gate type is "OR"/ "AND")
                IF (the gate has two or more input gates)
                    FOR (i from the first to the last input gate)
                          FOR (j from the first to the last input gate)
                                IF (i and j are of the same type)
                                {
                                    IF (gates have identical input components)
                                        create a new gate for these components;
                                        IF (the new gate has only two input events)
                                            IF (the gate i / j  have only two input events)
                                                replace the gate i / j by the newly created gate;
                                                delete the gate i / j from the FT structure;
                                            ELSE  amend the gate i / j structure;
                                }
      check the logic of the amended FT;
}
```

Figure 8.10 The Algorithm for the Extraction Step

Factorization is the third and final step of the fault tree simplification algorithm and is implemented in the subroutine *Factorisation*. The program searches for the independent fault tree events, which appear once in its structure or several times but only as the descendants of the same gate. In the example FT such events are: *sve2*, *r11*, *r12*, *plc1*, *swv* and *smv*. Each pair of the independent events, found first, is changed by a new event. The new event names are generated in the subroutine *Array*. The search continues till all independent events are grouped where appropriate. Therefore, in the example HIPS fault tree the following structural amendments should be done:

sve2 and *r11* are changed by *Fact1*,
r12 and *plc1* are changed by *Fact2,*
swv and *smv* are changed by *Fact3,*
Fact1 and *Fact2* are changed by *Fact4,*
and *Fact3* and *Fact4* are changed by *Fact5*.

Every time the events are grouped their reliability data (unavailability and failure frequency) is combined according to the gate type within the subroutine *Array*. The overall algorithm for the factorisation procedure is shown in figure 8.11.

```
Factorisation
{
    go through all gates in the FT structure:

        IF (the gate has two or more input events)
        {
            FOR (each pair of input events)
            {
                IF (events are independent)
                {
                    IF (events occur together everywhere else in the FT structure)
                    {
                        create a new event; // (fact1, …, factn)
                        evaluate reliability data for the new event;
                        amend FT structure;
                    }
                }
            }
        }
}
```

Figure 8.11 The Algorithm for the Factorisation Step

The resulting example HIPS design fault tree after this step is shown in figure 8.12.

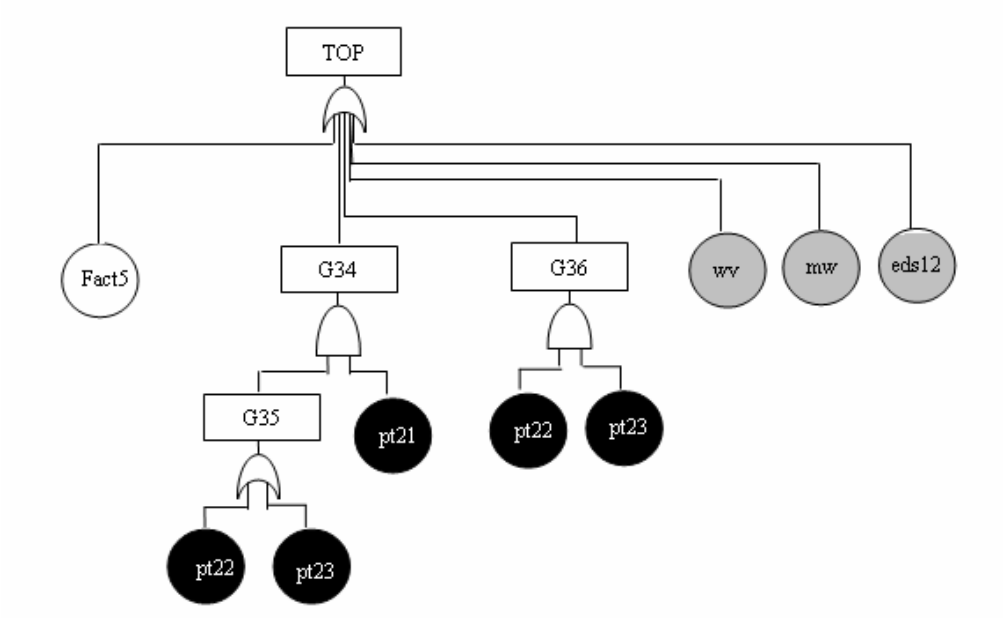


Figure 8.12 Example HIPS Design FT after Factorization

The discussed FT simplification subroutines (contraction, extraction and factorization) are incorporated to the routine *ModifyFT*, which is applied to each potential HIPS design in both the unavailability and spurious trip fault trees.

Step 3. Dependency Information Formation: After the fault tree simplification the next important step is the allocation of each gate dependency serial numbers. The subroutines *Form_dep_unav* and *Form_dep_spur* go through unavailability and spurious trip fault tree structures and form the dependency information for each gate. Table 8.2 shows the results for the example HIPS design.

Table 8.2 Example HIPS Design FT Gate Dependency Serials

Gate Name	Serial Number
Top	1, 2
G34	1
G35	1
G36	1

Step 4. Combination of Dependent Events: The final fault tree restructure step before modularization is the combination step, implemented in subroutines *Comb_dep_unav* and *Comb_dep_spur* for unavailability and spurious trip fault trees respectively. The program examines the fault tree structure and groups the same dependency serial events into separate branches. During this process additional gates are added to the fault tree structure. These gates do not affect the logic of the fault tree, since they have the same type as the gates which contain the dependent events.

For the example HIPS design, two gates (*Comb1* and *Comb2*) should be added to separate the dependency serial 1 and 2 components from other fault tree components (Figure 8.14). Figure 8.13 shows the general algorithm for the combination technique.

```
Combination
{
    go through all gates in the FT structure:
        IF (gate has several dependency serials)
        {
            group its dependent input gates and/or events by the dependency serial;
            create a new gate for each dependency group;
            amend the gate structure;
            add new gates to the FT structure;
        }
    check the logic of the FT structure;
}
```

Figure 8.13 The Algorithm for the Combination Technique

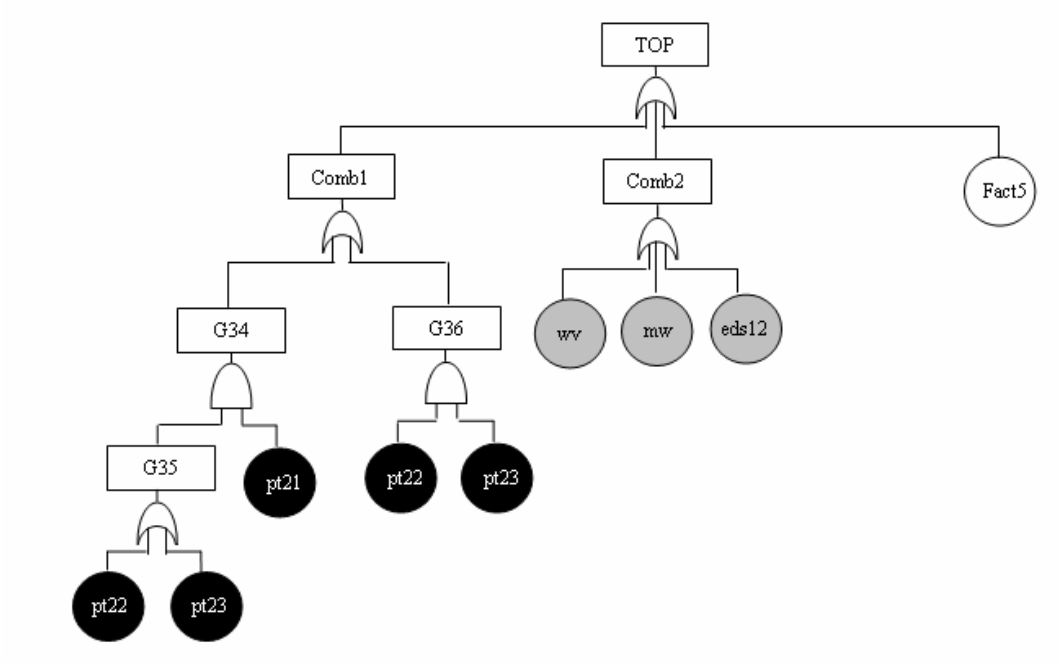


Figure 8.14 Example HIPS Design FT after Combination

Step 5. Fault Tree Modularization: The modularization algorithm is implemented in the program routine *Modularization*. The search starts from the top of the fault tree and goes to the bottom in the left to right direction. During the search the first, second and last visit numbers to all components are fixed in a new array. Then for each gate the first minimal and the last maximum visit number to any of its descendant is calculated from the search results.

If the gate is a module (i.e. satisfies both conditions discussed in section 8.3.1) the first indicator (*indicator1*) is set to 1, otherwise its value is set to zero. If the gate is a module, the subroutine *Update_dep* checks if this module contains only one dependency serial and is minimal. If these two conditions are satisfied the second indicator (*indicator2*) is set to 1, otherwise, its value is equal to zero. If indicator 2 is equal to one, it is assumed that the gate is the independent module and can go through Markov analysis. Figure 8.15 shows the generalized algorithm for the modularization step.

```
Modularization
{
    FOR (the top gate to the bottom gate):
                Fix the first, second and last visit numbers to all gates and events;
        IF (the gate satisfies both conditions discussed in section 8.3.1)
                set indicator 1 to 1;
            ELSE set indicator 1 to 0;
    Go through all gates in the FT structure:
                IF (the gate is a module, i.e. indicator1 = 1)AND(the module is minimal)
                AND(the module has only one dependency type)
                    set indicator2 to 1;
                ELSE set indicator2 to 0;
            IF (indicator2 = 1) module is independent and will go through the Markov analysis;
            ELSE apply the algorithm from section 8.3.1 (step 7);
}
```

Figure 8.15 The Algorithm for the Modularization Step

Table 8.3 shows the modularization results for the example HIPS design fault tree from figure 8.10.

Table 8.3 Example HIPS Design FT Modularization Results

Gate Name	1st Visit Number	2nd Visit Number	Last Visit Number	Total No of Visits	First Min Visit No	Last Max Visit No	Indicator 1	Indicator 2
Top	1	21	21	2	2	20	1	0
Comb1	2	14	14	2	3	13	1	1
G34	3	9	9	2	4	12	0	0
G35	4	7	7	2	5	12	0	0
G36	10	13	13	2	5	12	0	0
Comb2	15	19	19	2	26	28	1	1
Event Name								
spt22	5	12	12	2	-	-	-	-
spt23	6	11	11	2	-	-	-	-
spt21	8	8	8	1	-	-	-	-
esd12	16	16	16	1	-	-	-	-
Mv	17	17	17	1	-	-	-	-
Wv	18	18	18	1	-	-	-	-
Fact5	20	20	20	1	-	-	-	-

It can be seen from table 8.3 that only gates *Top*, *Comb1* and *Comb2* satisfy the module conditions (Indicator 1 is equal to 1). However, only gates *Comb1* and *Comb2* are minimal and have only one dependency serial each. Therefore, the indicator 2 value is equal to 1 for both gates and they are assumed to be the final independent modules, which will go through Markov analysis.

Markov Analysis: The subroutine *Apply_Markov* identifies the independent modules in the fault tree and applies the Markov models to them (Figure 8.17). The Markov models are generated by the subroutine *Markov* for each module separately. The quantification of the model is performed by the subroutine *Determinant*. When the Markov modelling for the independent module is complete (i.e. the module unavailability and spurious failure frequency are evaluated), the module is replaced by a new event, generated in the subroutine *Array*. Hence, in the example HIPS design fault tree the gates *Comb1* and *Comb2* will be replaced by the events *Mark1* and *Mark2*. The reliability data from Markov analysis for these new events will be added to the common HIPS event data array. The final structure of the example HIPS design fault tree is shown in figure 8.16. This final structure is dependency free and, therefore, can be analysed by conventional FT and BDD techniques.

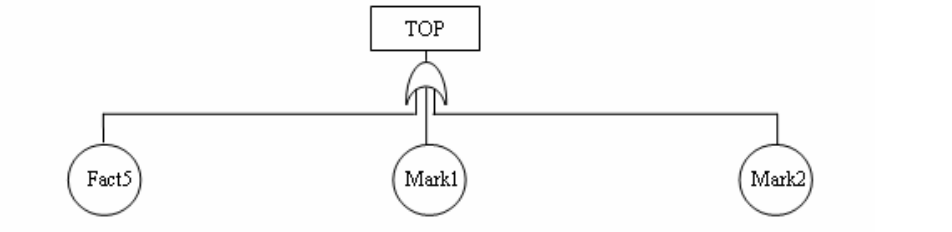


Figure 8.16 Example HIPS Design FT after Markov Analysis

```
Apply_Markov (maintenance dependency type)
{    FOR (the top gate to the bottom gate)
       {
            IF (the gate is an independent module)
            {
               evaluate the number of components for the Markov model; //i.e. n
               establish the initial state; // i.e. (0_1, …,0_n);
               generate all other states by using the algorithm from section 8.3.2;
               FOR (the first  to the last model state)
               {
                     evaluate the module (system) state in this model state;
                     IF (the module fails in this model state) set the state indicator to 1;
                     ELSE set the state indicator to 0;
               }
               FOR (the first to the last state indicator)
                     IF (the state indicator = 1)//state corresponds to the module failure
                          Include the state to the list of states corresponding to the module failure;

               Evaluate the module unavailability (Q_m) and frequency (w_m); // equation 8.4-8.8
               create a new event and record the Q_m and w_m in the event data array;
               replace the independent module by the newly created event in the FT;
               }
       }
    check the logic of the FT structure;
}
```

Figure 8.17 The Algorithm for the Markov Model Generation

8.4.3 Results

To test the difference between the results from the optimization of the HIPS with and without dependencies, the best ten designs obtained after ten runs of the original ISPEASSOP (chapter 6) have been tested with the modified version of the program. Table 8.4 shows the comparison of the results. The corresponding design parameter values are provided in table 8.5.

Table 8.4 Comparison of the Results for Original ISPEASSOP and ISPEASSOP with Dependencies

Run No.	Cost		MDT		F_{sys}		Q_{sys}	
	Original	With Depend.	Original	With Depend.	Original	With Depend.	Original	With Depend.
1	592		129.7008		0.455	0.476	4.50e-7	5.32e-7
2	512		129.6974		0.332	0.389	8.33e-4	9.26e-4
3	582		128.7361		0.324	0.350	6.80e-4	7.25e-4
4	922		128.2273		0.718	0.766	1.00e-6	8.20e-6
5	882		129.1590		0.166	0.235	1.00e-6	1.60e-6
6	992		129.2523		0.552	0.612	1.00e-6	8.04e-6
7	852		128.3286		0.245	0.295	6.55e-4	1.06e-3
8	542		128.9881		0.324	0.387	8.45e-4	9.01e-4
9	872		129.9032		0.377	0.437	1.00e-6	5.93e-6
10	862		129.7309		0.999	0.999	1.00e-6	8.88e-6

Table 8.5 Design Parameter Values for Table 8.4

Run No.	*Q1*	*Q2*	*V*	*P*	*N1*	*N2*	*K1*	*K2*	*E*	*H*
1	25	73	1	2	1	3	1	3	0	1
2	27	105	2	2	1	0	1	0	1	0
3	64	9	2	1	4	0	3	0	1	0
4	33	96	1	1	2	3	1	3	1	1
5	42	53	1	2	4	2	4	1	1	1
6	34	90	2	2	2	3	2	2	1	2
7	40	91	1	2	3	0	3	0	2	0
8	27	118	2	1	2	0	2	0	1	0
9	26	124	1	2	3	2	3	2	0	2
10	42	46	1	2	2	2	1	2	1	1

It can be seen from table 8.4 that implementation of maintenance dependency for the HIPS components resulted on average in 19% higher system unavailability and 15% higher spurious trip frequency values for all designs. These changes can be explained by

the increase of repair times for individual components due to the implemented dependency.

8.5 Summary

- This chapter has suggested and discussed the main dependency types, which may exist between the safety system component failures. These types are: maintenance, standby, secondary-failure, initiator-enabler and test dependency.

- One dependency type with eight dependency groups have been identified for the example HIPS system. The following modifications have been done to the developed optimization tool, discussed in chapter 6, in order to allow the use of fault trees and binary decision diagrams for analysis:

    a) Dependency information have been formed for the HIPS system;

    b) The fault tree simplification and modularization methods have been incorporated into the program structure in order to allocate the smallest independent sections in the fault tree.

    c) The Markov model generation algorithm for the maintenance dependency type has been developed and applied to the identified smallest independent modules with dependent components.

- The new technique has been applied to the HIPS system. Comparison of the results, obtained by the original optimization tool with the assumption that the system components are all independent and those, obtained by the suggested optimization technique for the system with dependencies, shows that for all potential original HIPS designs the system unavailability and spurious failure frequency have been underestimated. Therefore, it is important to identify all system dependencies for more accurate system unavailability and spurious failure frequency prediction.

- The main weakness of the optimization tool for systems with dependencies is its running time. One run of the original program for independent system components

is in order of minutes, however, the modified version requires several hours, due to the complexity of Markov analysis for a large number of system components even after the fault tree modularization.

- Potentially the Markov model size for maintenance dependency type could be reduced by increasing the number of maintenance engineers. This change would result in the dependent component elimination from the model for each additional engineer. This factor could be easily implemented into the optimization code.

- The HIPS is a small safety system with a single dependency type existing between its components failures, hence, the developed optimization scheme should be tested on a larger safety system with a wider range of dependencies.

CHAPTER 9

OPTIMIZATION OF FDS WITH DEPENDENCIES

9.1 Introduction

The previous chapter describes the optimization of the high integrity protection system with dependencies existing between its components. The HIPS is a relatively simple safety system and includes only maintenance dependency. In more complex systems two or more dependency types may exist. Therefore, a generalized technique should be developed in order to analyse different combinations of dependencies in the system. This chapter describes the firewater deluge system (FDS) optimization by the developed optimization tool. Section 9.2 discusses the FDS dependency groups. Section 9.3 introduces the Markov model generation algorithms for all dependency types. Results comparison is shown in section 9.4. The discussion of the results and the program potential application to any safety system is represented in section 9.5. The chapter finishes with the summary (section 9.6).

9.2 FDS Dependency Groups

Three main dependency types have been identified for the FDS components. The maintenance and test dependencies are relevant to the static system phase and exist due to the system maintenance features and the inspection carried out on each pump stream. The standby dependency exists in the dynamic system phase between the electric and diesel firewater and AFFF pumps. In total 26 dependency groups have been formed. Tables 9.1, 9.2 and 9.3 show the structure of maintenance, stand-by and test dependency groups respectively. The groups represented in these tables consist of all possible dependent events. However, each design will have a particular combination of events based on the equipment type and the design configuration.

Table 9.1 Maintenance Dependency Groups for FDS System

Dependency Group No	Events	Description
1	PT11/2/3/4 or PT21/2/3/4 or PT31/2/3/4	Pressure sensors 1-4 (type 1-3)
2	FB01, IVB 011, IVB 012	Firewater jockey pump with filter and associated isolation valves
3	E_100_1/2/3/4, E1_50 1/2/3/4, E2_50 1/2/3/4, E1_33 1/2/3/4, E2_33 1/2/3/4, D_100 1/2/3/4, D1_50 1/2/3/4, D2_50 1/2/3/4, D1_33 1/2/3/4, D2_33 1/2/3/4	All electric and diesel firewater pumps
4	IVB 1/2/3/4, CVB 1/2/3/4	All electric firewater valves
5	IVB 5/6/7/8 and CVB 5/6/7/8	All diesel firewater valves
6	APT11/2/3 or APT21/2/3 or APT31/2/3	AFFF ringmain components
7	FB 02, IVB 021 and IVB 022	AFFF jockey pump with filter and associated valves
8	AE_100 1/2, AE_50 1/2, AD_100 1/2, AD_50 1/2	All electric and diesel AFFF pumps
9	IVB 9/10, CVB 9/10	All electric AFFF valves
10	IVB 11/12, CVB 11/12	All diesel AFFF valves
11	WBS, WBN or WBO, WIVB or AIVB, AV1/2/3, ACVB, WV1/2/3	*Deluge skid components:* strainer, inductor nozzle, butterfly valve, AFFF deluge valve, AFFF check valve, water deluge valve
12	SV1, SV2	Solenoid valve 1 and 2

Table 9.2 Standby Dependency Groups for FDS System

Dependency Group No	Events	Description
13	E_100_1/2/3/4, E1_50 1/2/3/4, E2_50 1/2/3/4, E1_33 1/2/3/4, E2_33 1/2/3/4 D_100 1/2/3/4, D1_50 1/2/3/4, D2_50 1/2/3/4, D1_33 1/2/3/4, D2_33 1/2/3/4	Electric firewater pump 1-4 fails when functioning
14	AE_100 1/2, AE_50 1/2, AD_100 1/2, AD_50 1/2	Electric AFFF pump fails when functioning

Table 9.3 Test Dependency Groups for FDS System

Dependency Group No	Events	Description
15	FB1, IVC11, IVC12, PRVO1, SVO1, DVO, ESF	Elect pump1 with filter and associated equipment: isolation valves 11-12, press relief valve 1, test valve 1, test line and power supply
16	FB2, IVC21, IVC22, PRVO2, SVO2, DVO, ESF	Elect pump2 with filter and associated equipment: isolation valves 21-22, press relief valve 2, test valve 2, test line and power supply
17	FB3, IVC31, IVC32, PRVO3, SVO3, DVO, ESF	Elect pump3 with filter and associated equipment: isolation valves 31-32, press relief valve 3, test valve 3, test line and power supply
18	FB4, IVC41, IVC42, PRVO4, SVO4, DVO, ESF	Elect pump4 with filter and associated equipment: isolation valves 41-42, press relief valve 4, test valve 4, test line and power supply
19	FB5, IVC51, IVC52, PRVO5, SVO5, DVO, DIVC	Diesel pump 1 with filter and associated equipment: isolation valves 51-52, press relief valve 5, test valve 5, test line, diesel supply
20	FB6, IVC61, IVC62, PRVO6, SVO6, DVO, DIVC	Diesel pump 2 with filter and associated equipment: isolation valves 61-62, press relief valve 6, test valve 6, test line, diesel supply
21	FB7, IVC71, IVC72, PRVO7, SVO7, DVO, DIVC	Diesel pump 3 with filter and associated equipment: isolation valves 71-72, press relief valve 7, test valve 7, test line, diesel supply
22	FB8, IVC81, IVC82, PRVO8, SVO8, DVO, DIVC	Diesel pump 4 with filter and associated equipment: isolation valves 81-82, press relief valve 8, test valve 8, test line, diesel supply
23	FB9, IVC91, IVC92, PRVO9, SVO9, ATIVC	Electric AFFF pump1with filter and associated equipment: isolation valves 91-92, press relief valve 9, test valve 9, electric supply to AFFF
24	FB10, IVC101, IVC102, PRVO10, SVO10, ATIVC	Electric AFFF pump2with filter and associated equipment: isolation valves 101-102, press relief valve 10, test valve 10, electric supply to AFFF
25	FB11, IVC111, IVC112, PRVO11, SVO11, ADIVC	Diesel AFFF pump1 with filter and associated equipment: isolation valves 111-112, press relief valve11, test valve11
26	FB12, IVC121, IVC122, PRVO12, SVO12, ADIVC	Diesel AFFF pump2 with filter and associated equipment: isolation valves 121-122, press relief valve12, test valve12

The dependency groups from tables 9.1 – 9.3 are stored in the *"FDSdep.txt"* file, which has identical structure to the *"HIPSdep.txt"* file (section 8.4.1). The initial dependency arrays are formed in the *Initial_dependency* routine.

9.3 Markov Analysis for FDS

The Markov model generation for the maintenance dependency type has been described in chapter 8 (section 8.3.2). In order to perform the Markov analysis for dependent components of the FDS system, additional routines have been added to the program structure for the test and standby dependency types. Sections 9.3.1 and 9.3.2 discuss the Markov model generation for the test and standby dependency types respectively.

9.3.1 Markov Model Generation for Test Dependency Type

The test dependency exists in the FDS system due to the discrepancy between a common inspection interval for a group of components and the inspection interval for individual components. This particular type of dependency has no effect on the interaction between component failures and does not change the way component failures contribute to the system failure. Hence, this dependency existence does not require the Markov method. The numerical solution shown in equation 9.1 can be used instead. However, the Markov model should account for the existence of this dependency type.

$$q_i(t) = 1 - e^{-\lambda\left(t - \max\left(n_1\theta_i, n_2\theta_m\right)\right)}, \tag{9.1}$$

where:

t is an actual time;

θ_i is the individual inspection interval for component *i*;

θ_m is the common inspection interval for the groups of components;

$\max(n_1\theta_i, n_2\theta_m)$ means that the component failure probability is considered after the most recent inspection (either the individual or common).

All groups containing the test dependency (Table 9.3) consist of six or seven components. The representing Markov model for each of these groups would exceed 500 states. It would be an impractical task to illustrate the model of this size as an example. Hence, consider a simple module consisting of only two components: *FB* (corresponding to the firewater pump with filter) and *IV* (corresponding to isolation valve on this pump line). Each component may be in one of three states: the working state '0', the unrevealed failure state '3' and the revealed failure state '1'. The resulting Markov model is shown in figure 9.1.

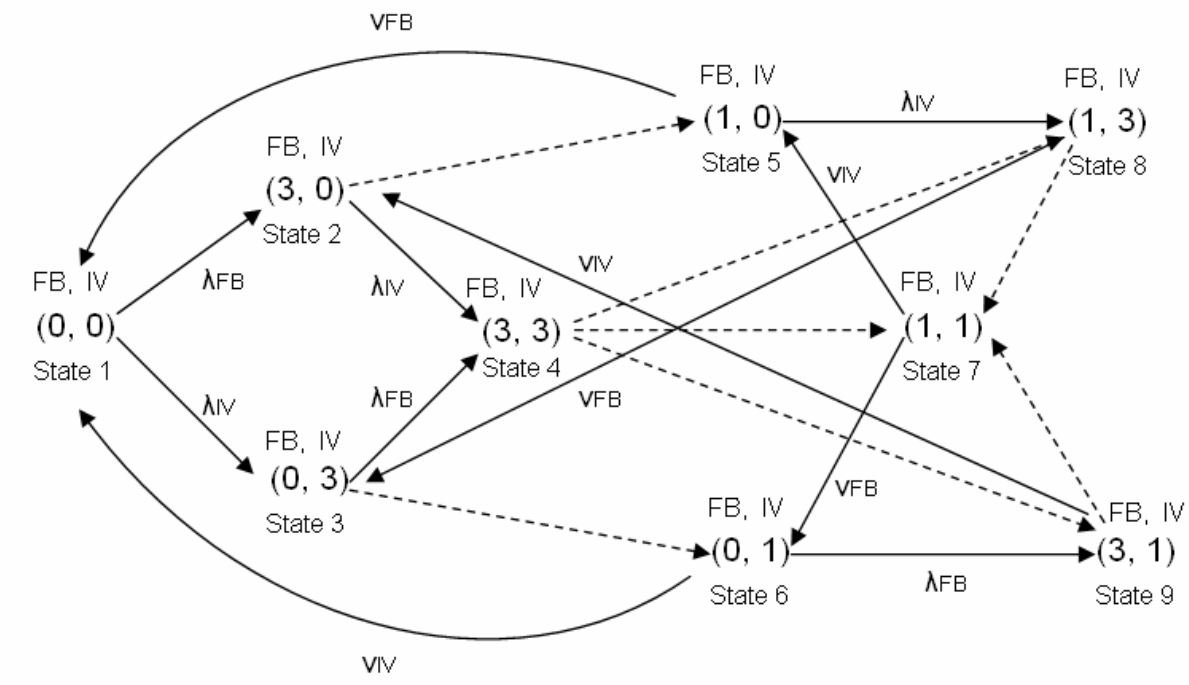


Figure 9.1 Markov Model for Example Module with Test Dependency

In figure 9.1 the time condition for the state with unrevealed failures to transit to the state with revealed failures can be expressed by equation 9.2:

$$T = k\theta_{system}\,, \tag{9.2}$$

where k is an integer number and θ_{system} corresponds to the system common test interval. Each time condition represents the discovery of the dormant failure and, therefore, causes the transition between states. Table 9.4 summarises the time conditions for the example Markov model from figure 9.1.

Table 9.4 Time Conditions for Markov Model from Figure 9.1

Transition from	Transition to	Time Condition
State 2	State 5	2 θ
State 3	State 6	3 θ
State 4	State 7	6 θ or 5 θ
State 4	State 8	2 θ
State 4	State 9	3 θ
State 8	State 7	3 θ or 5 θ
State 9	State 7	2 θ or 5 θ

The time conditions from table 9.4 have been established by using the algorithm discussed in chapter 8 (section 8.3.2). It can be noticed from table 9.4 that for the time condition $T = 5\theta$ transitions from states 4, 8 and 9 to state 7 can take place. This situation is possible only for the systems with test dependency between components. Since states 4, 8 and 9 correspond to the top event occurrence, at the test time equal to 5θ both dormant failures will be revealed. Therefore, by adding this time interval to the initial list of time conditions the Markov model will include the test dependency. The general algorithm for the Markov model generation for the test dependency is shown in figure 9.2. This algorithm is implemented in the subroutine *Markov_test_dep*.

```
Markov_test_dep
{
    establish the initial state; // (0, 0, …, 0)
    go through all existing states:
        go through all individual component states
            IF  (individual component state is equal to 0)
                set the new state to 1 or 3; //depending on failure type
            ELSE
                IF (individual component state is equal to 1)
                    set the new state to 0;
                ELSE
                    evaluate system time conditions;
                    IF (the top event occurs in the current system state)
                        add the time condition T=kθ
                IF (newly created state does not exist in the state list)
                    add new state to the state list;
                    record the transition rate:
}
```

Figure 9.2 Markov Model Generation Algorithm for Test Dependency Type

9.3.2 Markov Model Generation for Standby Dependency Type

The standby approach is commonly used in most safety systems structure to enhance the system ability to tolerate some key component failures and to ensure its continuous operation. In the FDS the standby dependency type exists between electric and diesel driven pumps, in both the firewater and the AFFF subsystems (Table 9.2). To illustrate the Markov model generation for this dependency type, assume that the AFFF subsystems consists of 1 electric and 1 diesel AFFF pump both of type 1 with a 100% capacity. Hence, the fourteenth dependency group (Table 9.2) will be represented by two components: *AE*_100 1 and *AD*_100 1. The electric pump (*AE*_100 1) is a duty pump. When it fails the standby diesel driven pump (*AD*_100 1) takes its duty. It is assumed that the failure of the duty pump (electric or diesel driven) is revealed at once. The standby pump will be checked for possible failures immediately after the duty pump failure. Therefore, the electric duty pump may be in one of two states: '0' corresponding to the pump working state and '1' corresponding to the pump revealed failure. On other hand, the diesel driven standby pump may be in one of three possible states: '0', '1' and '3' corresponding to the 'failed unrevealed' state.

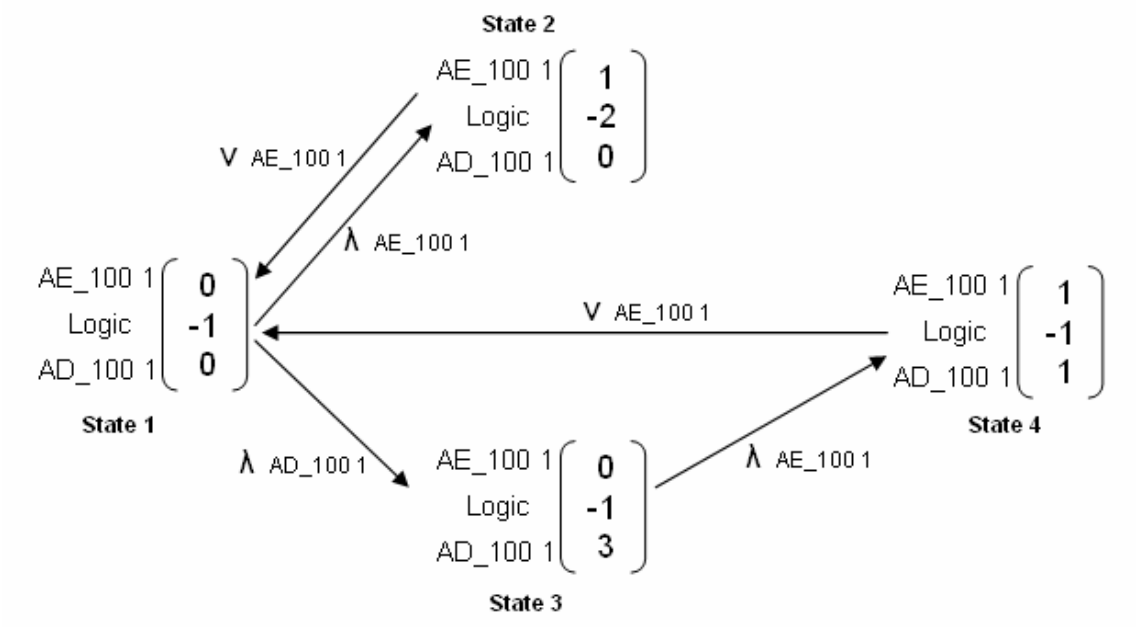


Figure 9.3 Markov Model for Example Module form Figure 9.4

It can be noticed from figure 9.3 that all states in the Markov model consist of three components. The additional component (*Logic*) is incorporated in the model structure to

represent the function of the standby diesel driven pump. The state '-1' corresponds to the standby state. On the other hand, the state '-2' corresponds to the duty function and indicates the failure of the electric pump.

The Markov model generation for the standby dependency type was incorporated in the program structure in the subroutine *Markov_standby_dep*. The general algorithm structure is shown in figure 9.4.

```
Markov_standby_dep
{
    establish the initial state; // (0, 0, …, -1, 0, …, 0)
    go through all existing states:
        IF  (the overall module state is 1) //failure state
            repair all revealed failures at once;
            IF (the new state is not in the state list)
                add to the state list;
                record the state transition rate;
        ELSE o through all existing states:
        {
        Case1:
            set a new state as 1 or 3;
            IF (the duty pump fails)
                IF (the standby component is available)
                    set the state of corresponding event to -2;
        Case2:
            set the new state to 0;
            IF (the duty pump is in state 0)
                set the new state to '-1';

        IF (the new state is not in the state list)
                add to the state list;
                record the state transition rate;

        }
}
```

Figure 9.4 Markov Model Generation Algorithm for Standby Dependency Type

9.4 Results

To test the difference between the results from the optimization of the FDS with and without dependencies, the best ten designs obtained after ten runs of the original program (chapter 7) have been tested with the modified version of the program. Table

9.5 shows the comparison of the results. The corresponding design parameter values are provided in table 9.6.

Table 9.5 Comparison of the Results for Original ISPEASSOP and ISPEASSOP with Dependencies

Run	Program Version	*STC*	*SPMC*	*SCMC*	*LCC*	F_{sys}	$Q_{sys} = Q'_{sys}$
1	Original	7796.4	5813.0	27606.2	102966	0.5593	9.321E-03
	Modified					0.6001	1.234E-02
2	Original	7935.3	6434.8	34240.6	85561	0.3220	1.012E-02
	Modified					0.3340	1.300E-02
3	Original	6767.0	6922.8	29052.2	103342	0.2602	1.078E-02
	Modified					0.4010	1.124E-02
4	Original	7796.4	5813.0	27606.3	102666	0.2002	9.321E-03
	Modified					0.2320	1.234E-02
5	Original	11381.6	8719.4	18403.5	99755	0.2613	9.436E-03
	Modified					0.3000	1.151E-02
6	Original	6892.4	12862.9	33277.8	117533	0.2033	1.081E-02
	Modified					0.2400	1.133E-02
7	Original	6423.1	12862.9	33277.8	117064	0.2033	1.078E-02
	Modified					0.2400	1.523E-02
8	Original	7796.4	5813.0	7606.2	102966	0.5593	9.321E-03
	Modified					0.6001	1.234E-02
9	Original	13766.2	10368.7	9396.6	93582	0.2020	1.360E-02
	Modified					0.2305	1.472E-02
10	Original	9857.9	7470.3	18265.2	103643	0.2022	1.284E-02
	Modified					0.2235	1.441E-02

Table 9.6 Design Parameter Values for Table 9.5

Design Variables	Run Number									
	1	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**
K / N	1 / 3	1 / 1	1 / 1	3 / 3	1 / 1	2 / 3	2 / 3	1 / 3	2 / 2	2 / 2
P	2	2	1	3	1	2	2	2	3	3
Firewater Supply and Distribution System										
F_E / F	3 / 4	0 / 1	1 / 4	3 / 4	3 / 4	2 / 3	2 / 3	3 / 4	1 / 3	1 / 4
F_P	33.3%	100%	33.3%	33.3%	33.3%	50%	50%	33.3%	33.3%	33.3%
F_T	2	2	2	2	2	2	2	2	1	1
AFFF Supply and Distribution System										
A_E / A	1 / 1	1 / 1	0 / 1	1 / 1	1 / 1	1 / 1	1 / 1	1 / 1	2 / 2	0 / 2
A_P	100%	100%	100%	100%	100%	100%	100%	100%	50%	50%
Valve and Material Types										
W	3	3	2	3	3	2	2	3	1	2
D	3	3	2	3	3	2	2	3	3	2
C	0	0	0	0	0	0	0	0	0	0
Maintenance Intervals										
θ_P	23	15	28	23	16	28	28	23	13	21
θ_R	18	4	5	18	6	4	20	18	14	6
θ_D	15	15	18	15	18	15	15	15	6	9
θ_{PM}	18	9	15	18	12	9	9	18	9	15

It can be seen from table 9.5 that the main difference appears in the spurious trip frequency and the system unavailability values. Figures 9.5 and 9.6 show the comparison of these parameter values for designs obtained by the initial and the modified ISSPEASSOP program versions respectively.

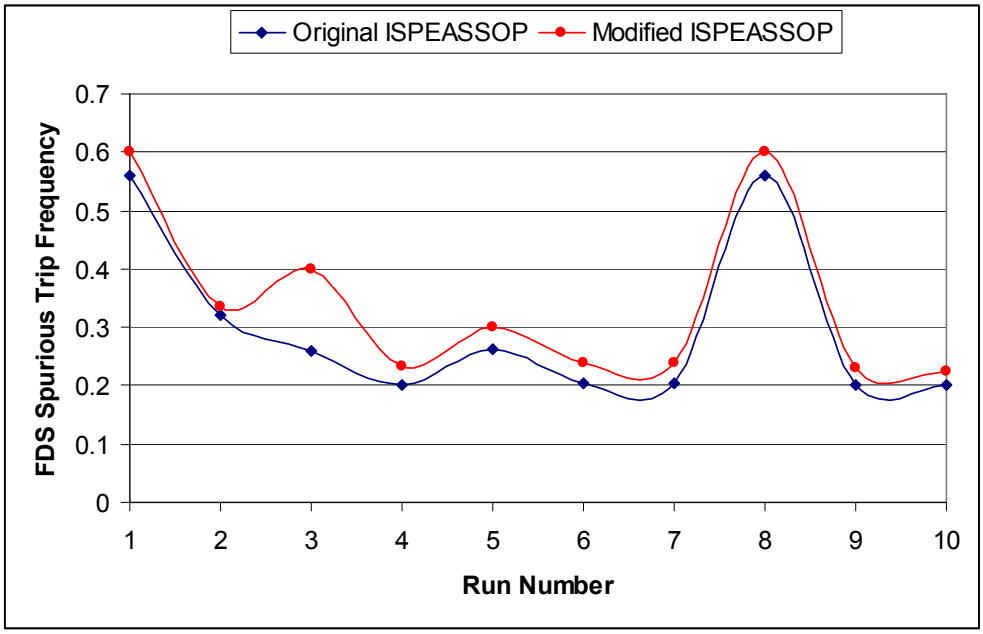


Figure 9.5 Comparison of the FDS Spurious Trip Frequency

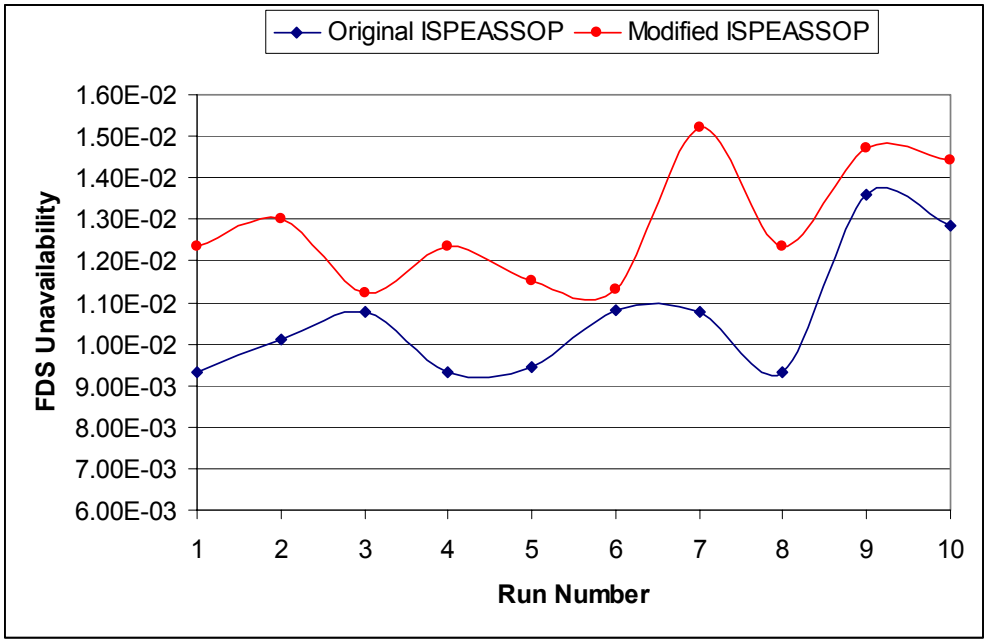


Figure 9.6 Comparison of the FDS Unavailability

Figures 9.5 and 9.6 show that, similarly to the HIPS (Chapter 8), the system unavailability and spurious trip frequency values have increased on average by 20% and 15% respectively for the potential designs of the FDS with dependencies mainly due to increased repair time for dependent components. However, the penalised system unavailability is equal to the system unavailability for all designs, i.e. all optimization parameter values are within their limits. The running time of the program increased from 12 minutes to 52 hours for the system with dependencies due to the complexity of the Markov analysis.

9.5 Discussion

The generation of the Markov models for the maintenance, standby and test dependency types has already been discussed. In addition to these types, the secondary_failure and initiator_enabler dependencies may exist among the safety system components (Chapter 8). The Markov model generation for these dependency types can be easily incorporated into the developed optimization tool.

Secondary failure Dependency: Consider the example module from figure 9.7. Assume that the secondary-failure dependency relationship exists between basic events: *b* and *d*, coloured in grey. In this example, the basic event *d* represents a primary failure, while basic event *b* corresponds to the secondary failure.

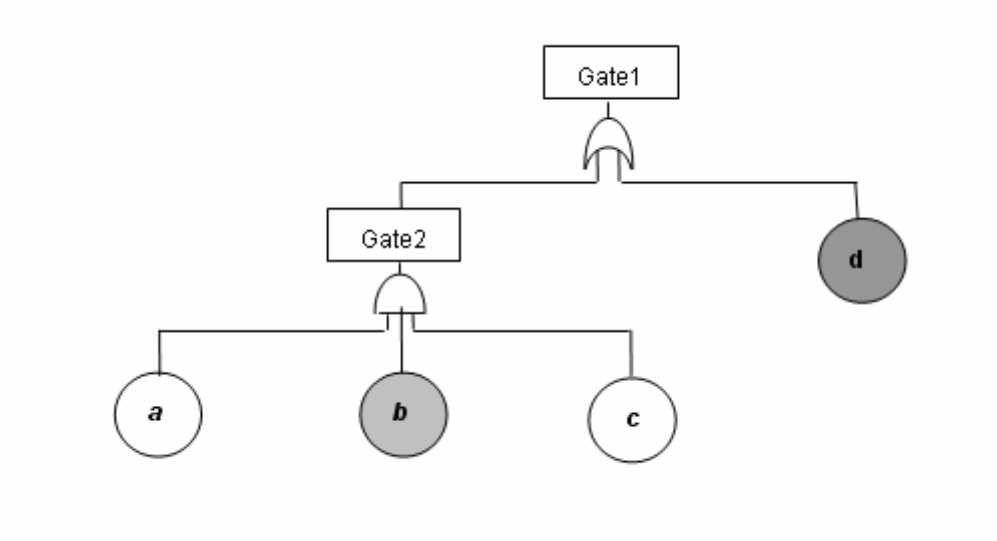


Figure 9.7 Example Module for Secondary_failure Dependency Type

Assuming that basic events *d* and *b* represent a revealed (i.e. 1) and dormant failure (i.e. 3) respectively, the resulting Markov model of the example module is shown in figure 9.8. The model structure is based on the assumption that no component failures will occur after the primary failure is revealed.

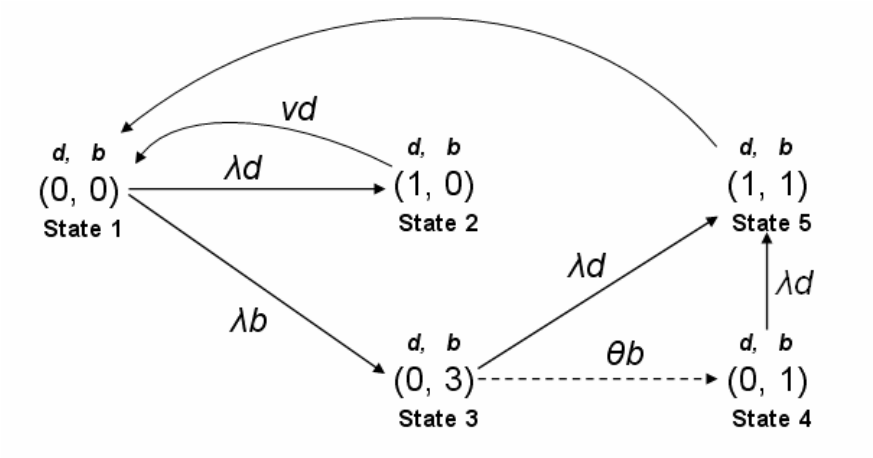


Figure 9.8 Example Markov Model for the Secondary_failure Dependency Type

The rules for the model construction can be described as follows [Sun, 2005]:

- The occurrence of the secondary failure in the system results in the failure of the primary component. Under this circumstance, the primary failure event should be considered as "occurred" and its repair process should start immediately. In figure 9.8, transitions from state 3 to state 5 and from state 4 to state 5 both reflect such a feature.

- The repair process involving the secondary_failure dependency type needs extra attention. When the failure of the primary component occurs and gets revealed, the secondary components will be inspected immediately for any potential failures. Therefore, the dormant failures of the secondary events will be revealed at this stage. Transition from state 3 to state 5 represents this process. All components with revealed failures will go through the repair process simultaneously and the system will be restored to the initial state when the repair of all its components is completed. In this case, the corresponding transition rate is the inverse of the maximum of the mean time to repair (τ) of all the failed

components. For example, the rate of the transition from state 5 to state 1 can be expressed as:

$$\tau_{5-1} = \frac{1}{\max(\tau_d, \tau_b)}, \tag{9.3}$$

where τ_d and τ_b are the mean time to repair component *d* and *b* respectively.

The overall algorithm for the Markov model generation for the secondary_failure dependency type is suggested in figure 9.9.

```
Markov Model Generation (secondary_failure dependency type)
{
    establish the initial state; // (0, 0, …, 0)

    go through all existing states:
      IF  (the primary failure is revealed (i.e. state is equal to 1))
        IF (the secondary failure event has occurred)
        {
          record the transition from the current state to the initial state;
          evaluate the transition rate accordingly;
        }
        ELSE
        {
          set the new state for the primary failure event to 0;
          set the state for the secondary failure event to 1; //initially is equal to 3
          IF (newly generated state doesn't exist in the state list)
          {
            add this state to the state list;
            record the transition rate:
          }
        }
      ELSE (go through all existing states)
          IF (current state is equal to 0)
            set new state to 1 or 3; //depending on failure
          IF (secondary failure occurs in the resulting system state)
            set new primary failure event state to '1' or '3'
          ELSE
            IF (current state is equal to 1)
              set new state to 0;
          IF (newly created state does not exist in the state list)
            add new state to the state list;
            record the transition rate:
}
```

Figure 9.9 The Algorithm for the Markov Model Generation for the Secondary_failure Dependency Type

Initiator enabler Dependency: To illustrate the Markov model generation for this dependency type, consider the example module from figure 9.10. The fault tree structure represents the potential damage of the compressor due to the failure of the

solenoid valves *S1* and *S2* to close on demand and, hence, cut the liquid supply to the compressor. Assuming that the abnormally high level of liquid can occur in the system (event *A*), such failure of the solenoid valves and the failure of the compressor to shutdown on demand (event *C*) will result in the compressor damage.

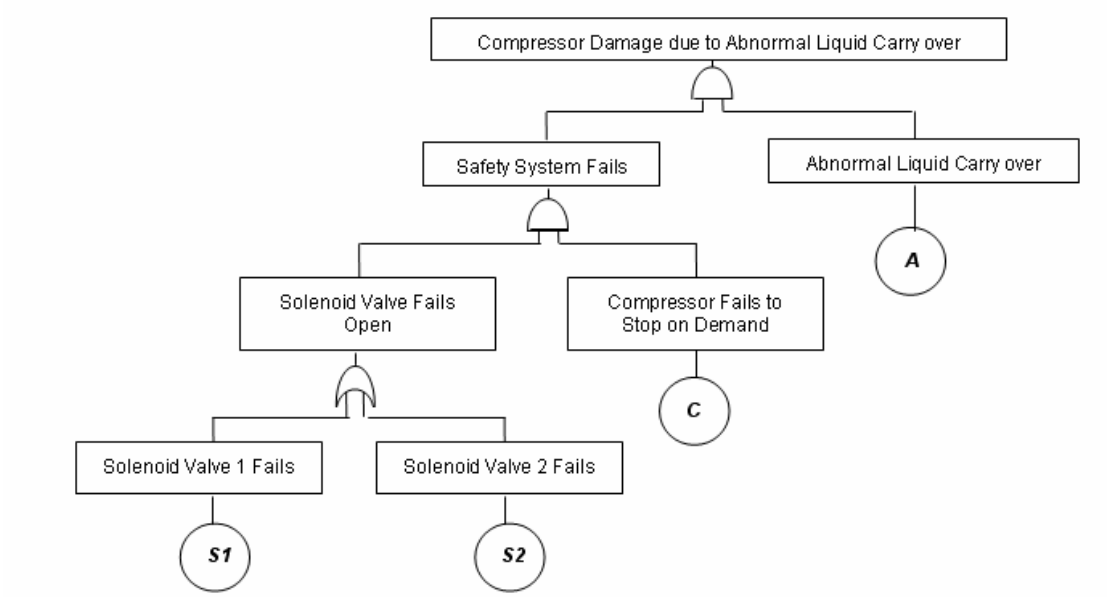


Figure 9.10 Example Module for Initiator_enabler Dependency Type

In the system shown in figure 9.10, the initiator is the abnormal liquid carry over event *A*. Events *C*, *S1* and *S2* are three enabling events featuring different reliability characteristics. The resulting Markov model for this example module consists of 54 states. Since the diagrammatic form of the model could not be represented clearly due to its size, the model structure is represented in table 9.7.

Table 9.7 Resulting Markov Model for Example Module from Figure 9.10

State	*A*	*C*	*S1*	*S2*	Mod. State	State	*A*	*C*	*S1*	*S2*	Mod. State
1	0	0	0	0	0	**28**	0	3	1	0	-2
2	1	0	0	0	-1	**29**	0	0	1	3	0
3	0	3	0	0	0	**30**	0	0	1	1	0
4	0	0	3	0	0	**31**	0	3	0	1	-2
5	0	0	0	3	0	**32**	0	0	3	1	0
6	1	0	3	0	-1	**33**	1	0	1	3	-1
7	1	0	0	3	-1	**34**	1	0	1	3	-1
8	1	3	0	0	-1	**35**	1	0	3	1	-1

Table 9.7 Continued

State	*A*	*C*	*S1*	*S2*	Mod. State	State	*A*	*C*	*S1*	*S2*	Mod. State
9	0	1	0	0	0	**36**	1	1	3	0	-1
10	0	3	3	0	-2	**37**	1	1	0	3	-1
11	0	3	0	3	-2	**38**	1	1	1	0	-1
12	0	0	1	0	0	**39**	1	3	3	3	-1
13	0	0	3	3	0	**40**	1	1	0	1	-1
14	0	0	0	1	0	**41**	0	1	3	3	-2
15	1	0	0	0	-1	**42**	1	1	1	3	1
16	1	0	3	3	-1	**43**	0	1	1	3	-2
17	1	0	0	1	-1	**44**	1	1	1	1	1
18	1	1	0	0	-1	**45**	0	1	1	1	-2
19	1	3	3	0	-1	**46**	1	1	3	1	1
20	1	3	0	3	-1	**47**	0	1	3	1	-2
21	0	1	3	0	-2	**48**	0	3	1	3	-2
22	0	1	0	3	-2	**49**	0	3	1	1	-2
23	1	1	1	0	1	**50**	0	3	3	1	-2
24	0	1	1	0	-2	**51**	1	1	3	3	-1
25	0	3	3	3	-2	**52**	1	1	1	3	-1
26	1	1	0	1	1	**53**	1	1	1	1	-1
27	0	1	0	1	-2	**54**	1	1	3	1	-1

In the example Markov model (Table 9.7) values '-1' and '-2' are used along with '0' and '1' to represent the module state. Both new values indicate that the top event has not occurred yet, however, they both have more implication than the working state '0'. These states indicate a different order of occurrence between the initiating and enabling event [Sun, 2005]. The state '-1' refers to the situation where the initiating event occurs prior to the general enabling event (for example, the transition from the state 3 to the state 8 in Table 9.7). On the other hand, the module state '-2' corresponds to the situation where the enabling event occurs prior to the initiating event. For example, in the state 10 (Table 9.7), the occurrence of the basic events *C* and *S1* result in the occurrence of the enabling event *A*. Since the initiating event has not occurred in this state, the module state of the state 10 is set to '-2'. The overall algorithm for the Markov

model generation for the initiator_enabler dependency type among the system components is shown in figure 9.11 [Sun, 2005].

```
Markov Model Generation (initiator_enabler dependency type)
{
    establish the initial state; // (0, 0, …, 0)
    go through all existing states:
       IF  (the primary failure is revealed (i.e. state is equal to 1))
          IF (the secondary failure event has occurred)
          {
             record the transition from the current state to the initial state;
             evaluate the transition rate accordingly;
          }
          ELSE
          {
              set the new state for the primary failure event to 0;
              set the state for the secondary failure event to 1; //initially is equal to 3
              IF (newly generated state doesn't exist in the state list)
              {
                  add this state to the state list;
                  record the transition rate:
               }
          }
       ELSE (go through all existing states)
              IF (current state is equal to 0)
                 set new state to 1 or 3; //depending on failure
              ELSE
                 IF (current state is equal to 1)
                    set new state to 0;
                 IF (the event is initiating event with a state '1')
                     reveal all dormant failures of enabling events;
                 ELSE algorithm represented in chapter 8 (section 8.3);
              IF (newly created state does not exist in the state list)
                 add new state to the state list;
                 record the transition rate:
}
```

Figure 9.11 Markov Model Generation for Initiator_Enabler Dependency Type

Combination of Secondary_failure and Initiator_enabler Dependency Types

The initiator-enabler dependency can occur in a system together with a secondary-failure dependency. This can happen when component failures corresponding to the initiator-enabler dependency contribute to the causes of the secondary failure. In this case, all relevant component failures will have to be investigated in the same model in order to obtain the accurate system reliability parameters. Figure 9.12 shows the general possible algorithm for the Markov model generations for the situation when the

secondary failure dependency should be solved together with the initiator-enabler dependency [Sun, 2005].

```
Markov (secondary_failure & initiator_enabler dependency types)
{
    establish the initial state; // (0, 0, …, 0)
    go through all existing states:
      IF  (the primary failure is revealed (i.e. state is equal to 1))
        IF (the secondary failure event has occurred)
        {
          record the transition from the current state to the initial state;
          evaluate the transition rate accordingly;
        }
        ELSE
        {
          set the new state for the primary failure event to 0;
          set the state for the secondary failure event to 1; //initially is equal to 3
          IF (newly generated state doesn't exist in the state list)
          {
            add this state to the state list;
            record the transition rate:
          }
        }
      ELSE (go through all existing states)
          IF (current state is equal to 0)
            set new state to 1 or 3; //depending on failure
          IF (secondary failure occurs in the resulting system state) //init_enab. dependency
            set new primary failure event state to '1' or '3'
          ELSE
            algorithm from chapter 8 (section 8.3);
            IF (current state is equal to 1)
              set new state to 0;
          IF (newly created state does not exist in the state list)
            add new state to the state list;
            record the transition rate:
}
```

Figure 9.12 Markov Model Generation Algorithm for the Mixture of the Initiator-Enabler and Secondary-Failure Dependency Types

9.6 Summary

- Three dependency types have been identified for the FDS component failures. They are: maintenance, test and standby dependencies. The Markov model generation algorithms have been developed for the additional dependency types and incorporated into the program structure.

- The modified developed tool has been successfully applied to the FDS with dependencies optimization. Results show that, similar to the HIPS, the FDS system unavailability and the spurious trip frequency increased for all potential system designs. Therefore, it is imperative to incorporate the analysis of the dependent components in every safety system optimization process.

- The Markov model generation algorithms have been suggested for all other main dependency types, i.e. secondary-failure and initiator-enabler dependencies. This gives the developed technique the potential to be applicable to any type of the safety system. The program is flexible, additional dependency types and related Markov models can be easily incorporated in its structure.

- The main disadvantage of the developed tool is the running time of the program which increases rapidly for larger systems due to the complexity of the Markov analysis. Compared to the HIPS, the program running time for the FDS system optimization is ten times greater.

- For the majority of safety systems several dependency relationships may exist in the system at the same time. Ignorance of any dependency type would result in an incorrect Markov model. Hence, all dependency types, existing between the system components failures, should be integrated into the model structure.

- Each combination of different dependency types requires a unique Markov model generation algorithm. It is impossible to go through every possible combination of different dependency relationships. Therefore, another disadvantage of the developed technique is that the program should be amended for each safety system in accordance with its dependency information.

- The developed optimization tool is based on assumption that all system components are repairable. No credit is given to the safety systems with non-repairable structure. Therefore, further modifications to the program code should be considered to consider systems with these characteristics.

CHAPTER 10

CONCLUSIONS AND FUTURE WORK

10.1 Introduction

The performed research has achieved the aims discussed in the objectives section of chapter one:

1) *The developed optimization scheme*: the developed safety system design optimization scheme combines the advantages of fault tree analysis (FT) for system failure logic representation, binary decision diagrams (BDD) for system design quantification and the Improved Strength Pareto Evolutionary Algorithm (SPEA2) for system design optimization. These have been integrated into an automated tool. House events are incorporated into the fault tree structure to allow the use of a single FT for all possible system designs.

   The conversion of the general system FT to the BDD for each design would be a time-consuming task. To overcome this problem, the fault tree reduction technique has been developed and implemented into the program structure. It minimises the FT structure for each possible design in accordance with the house events logic and, hence, reduces the corresponding BDD size. This results in the reduction of the overall analysis time.

   To find an optimal design a process is required which considers a number of design variables. Multi-Objective Genetic Algorithms (MOGAs) are a group of techniques which allow this type of parallel processing. The developed optimization tool incorporates the SPEA2 method. It is a relatively recent evolutionary technique for finding or approximating the optimal solution set for multi-objective optimization problems and has shown very good performance in comparison to other multi-objective genetic algorithms. This addition to the developed methodology permits the consideration of not only a primary objective, i.e. availability of the system, but

caters for all critical factors imperative to obtain an optimal system design and, hence, performance.

2) *High Integrity Protection System (HIPS) optimization*: the performance of the developed optimization tool has been tested on the example HIPS. There are three main parts of the developed ISPEASSOP program. Part one is responsible for the HIPS structure, part two is responsible for analysis using the Binary Decision Diagram method which calculates the HIPS unavailability and spurious trip frequency, and part three is an implemented SPEA2 algorithm for the HIPS optimisation.

   - The results produced by the ISPEASSOP were compared to those obtained by the simple GA based techniques. The SPEA2 based optimization tool finds the optimal solution quicker.

   - This application highlighted potential areas of improvement. The following modifications have been made to the program structure:

     - *Modification of the Design Parameter Evaluation Scheme*: the modified scheme ensures that all design parameters are from feasible regions. This made the feasible solution search faster.

     - *Modification of the Crossover Procedure*: the modified method is similar to the single-point crossover. The main difference appears when consideration is given to the second parent string from the pair. This string can again participate in crossover as the first parent. This amendment resulted in a larger variety of the potential designs and, hence, improved the overall technique performance.

3) *Firewater Deluge System (FDS) optimization*: the modified optimization tool has been tested on a more complex system (FDS) in order to explore its potential. The system has more design variables and increased complexity with additional resources. The following modifications have been done to the ISPEASSOP program:

- The new routine for the house event logic interpretation has been developed in order to represents the FDS features.
- The Weibull distribution has been used to represent failures of the wear out type components in the system unavailability and spurious trip frequency evaluation routines.
- The SPEA2 search has been amended to reflect the new number of optimization parameters.
- Additional limitations to the system design have been incorporated for all maintenance costs (i.e. life cycle cost per year, cost per year due to system testing, cost per year due to preventative maintenance, cost per year due to corrective maintenance, cost per year due to total maintenance effort).
- The life cycle cost and maintenance down time calculation procedures have been changed in accordance to the FDS operation features.

The results have been compared to those obtained by the simple GA. Similar to the HIPS, the SPEA2 based technique led to the optimal solution quicker. In addition, the running time of the program has been reduced from several hours to twelve minutes due to the discussed advanced features present in the developed technique.

4) *Optimization of safety systems with dependencies:* A further attempt on improving the applicability of the developed optimization technique has been carried out by investigating the dependency relationships among the safety system components. The Markov method allows the analysis of systems with dependencies. However, the applicability of this method is restricted for large systems by the state-space explosion problem. This problem has been solved by identifying the smallest independent sections (modules) in the fault tree structure. Therefore, the BDD and the Markov analysis are applied in a combined way to improve the analysis efficiency. The BDD method is applied to modules which contain no dependency. On the other hand, the Markov analysis is applied to modules which contain dependent components.

Different types of dependency have been identified for general safety system components. The algorithms for the Markov model generation have been established for all these dependency types.

- In order to test the modified technique performance on the HIPS and FDS, the following modifications have been made to the program:
    - Dependency information has been formed for the HIPS and FDS system component failures;
    - The fault tree simplification and modularization methods have been incorporated into the program structure in order to allocate the smallest independent sections in the fault tree.
    - The Markov model generation algorithms for relevant dependency types have been developed and applied to the identified smallest independent modules with dependent component failures.

Results showed that the previously obtained optimization parameter values were underestimated for both systems. Hence, in order to obtain accurate results it is imperative to take into account all dependency relationships existing between the safety system component failures. On the other hand, the program running time has increased for both systems from several minutes to several hours due to the complexity of the Markov analysis.

10.2 Conclusions

The research has led to the following conclusions:

1) The developed automated fault tree reduction technique proved to be effective for obtaining the fault tree structures for each potential system design and, hence, reducing the BDD size. This results in the reduction of the program running time.

2) In the case when several objectives are considered, as in this research, multi-objective genetic algorithms (MOGAs) can be used. These methods are based on

simple GAs principles, are easy to apply and find a set of nondominated optimal solutions quickly. The Improved Strength Pareto Evolutionary Algorithm (SPEA2) has been incorporated in the optimization tool and was successfully tested on the High Integrity Protection System (HIPS).

- Results showed that the SPEA2-based optimization technique performs better than the simple GA-based tool for the relatively small number of generations and leads to the optimal solution quicker.

- Constraints on the available resources do not prohibit application of the SPEA2 method. Any implicit or explicit constraints can be incorporated in the program structure. Standard SPEA2 can find results quickly, but modifications to GA parameters can improve the efficiency of the search procedure.

3) The modified technique was successfully tested on a Firewater Deluge System (FDS). The use of a Weibull distribution to model the relevant FDS components within the modified SPEA2-based approach enabled consideration of the components of wear-out type. Similar to the HIPS, the developed technique performed better than the simple GA based optimization tool.

4) The Markov method can be used for analysis of the independent modules, identified for each dependency group, in the fault tree structure. The integration of this method into the developed optimization tool enhances the range of systems in which optimization can occur.

   - Results comparison showed that the modified optimization tool with incorporated dependency analysis provided on average 19% higher safety system unavailability and 15% higher spurious trip frequency values compared to those obtained by the initial program version. Hence, the dependency information incorporation in the optimization process is essential for the optimization tool performance quality and accuracy.

   - The developed optimization tool has been applied to two example safety systems; however, the procedure could equally be applied to optimise the performance of any repairable safety system, whose failure causes can be

represented by a fault tree. The Markov model generation for other dependency types has been suggested and can be incorporated in the program structure.

10.3 Future Work

The research has led to the following potential areas of investigation:

- *Automated Markov model generation algorithm:* to develop an automated algorithm for the Markov model generation for any combination of dependencies in the safety system in order to enhance the developed tool application.

- *Markov model size reduction:* to find additional ways of dealing with the Markov model size or replace this method with an alternative technique (for example, the Bayesian Belief Networks since this method can be applied to systems with dependent component failures and potentially might reduce the analysis time); test and compare the new technique performance with the original tool.

- *Automated house event logic interpretation algorithm:* to develop an automated algorithm for the house event logic interpretation in order to simplify the tool application to safety system optimization. This could be possibly achieved by creating a data file representing the system component dependency on the house event values.

- *Developed tool efficiency and accuracy further improvement:* to apply the developed optimization technique to a variety of different safety systems (i.e. in a different industrial domain) to test its performance and develop, if necessary, modifications in order to improve the tool efficiency and accuracy.

- *Application to non-repairable and phase mission safety systems:* to modify the developed tool for non-repairable and phase mission system optimization, in order to explore the scope for its application.

BIBLIOGRAPHY

[Andersen, 1998] H. R. Andersen, "*An Introduction to Binary Decision Diagram*", Lecture notes for 49285 Advanced Algorithms E97, Technical University of Denmark, p.36, 1998.

[Andrews and Pattison, 1997] J. D. Andrews, R. L. Pattison, "*Optimal Safety System Performance*", Proceedings Annual Reliability and Maintainability Symposium, pp. 76-83, 1997.

[Andrews and Pattison, 1998] J. D. Andrews, R. L. Pattison, "*Genetic Algorithms in Optimal Safety System Design*", Proc. Instn. Mech. Engrs., Vol. 213, pp. 187-197, 1999.

[Andrews and Moss, 2002] J. D. Andrews, T. R. Moss, "*Reliability and Risk Assessment*", Second Edition, Professional Engineering Publishing, 2002.

[Andrews and Bartlett, 2003] J. D. Andrews, L. M. Bartlett, "*Genetic Algorithm Optimization of a Firewater Deluge System*", Quality and Reliability Engineering International, John Willey & Sons, pp. 39-52, 2003.

[Acar and Haftka, 2005] E. Acar, R. T. Haftka, "*Reliability Based Aircraft Structural Design Optimization with Uncertainty about Probability Distributions*", 6th World Congress of Structural and Multidisciplinary Optimization, Brazil, pp. 1-10, 2005.

[Babu and Jehan, 2003] B. V. Babu, M. M. L. Jehan, "*Differential Evolution for Multi-Objective Optimization*", Chemical Engineering Department, India, 2003.

[Bartlett and Riauke, 2009] L. M. Bartlett, J. Riauke. *Multi-objective Offshore Safety System Design Optimization*, paper has been submitted for publication in International Journal of Performability Engineering (IJPE).

[Buckles and Petry, 1992] B. P. Buckles, F. E. Petry, "*Genetic Algorithms*", IEEE Computer Society Press, 1992.

[Bellman, 2003] R. E. Bellman."*Dynamic Programming*", Dover Publications, Mineola, US, p. 388, 2003.

[Bettinger et al, 2002] P. Bettinger, D. Graetz, K. Boston, J. Sessions & W. Chung, "*Eight Heuristic Planning Techniques Applied to Three Increasingly Difficult Wildlife Planning Problems*", Silva Fennica 36(2): pp. 561-584.

[Borisevic and Bartlett, 2007 (1)] J. Borisevic, L. M. Bartlett. *Safety System Optimization by Improved Strength Pareto Evolutionary Algorithm (SPEA2)*, proceedings of the 17th AR^2TS, pp. 38-49, 2007.

[Borisevic and Bartlett, 2007 (2)] J. Borisevic, L. M. Bartlett. *Genetic algorithm based multi-objective optimization of a Firewater Deluge System*, Risk, Reliability and Societal Safety, Aven & Vinnem, Taylor & Francis Group, London, ISBN 978-0-415-44786-7, pp. 107-114, 2007.

[Busacca et al., 2001] P. G. Busacca, M. Marseguerra, E. Zio, "*Multiobjective Optimization by Genetic Algorithms: Application to safety systems*", Reliability Engineering and System Safety 72, pp. 59-74, 2001.

[Cantoni et al, 2000] M. Cantoni, M. Marseguerra, E. Zio, "*Genetic Algorithms and Monte Carlo Simulation for Optimal Plant Design*", Reliability Engineering and System Safety 68, pp. 29-38, 2000.

[Chamblers, 2001] L. Chambers, "*The practical handbook of Genetic Algorithms Applications*", Chapman & Hall/CRR, 2001.

[Chen and Hu, 2008] Y. Chen and K. Hu. "*Optimal Design of Steel Portal Frames Based on Genetic Algorithms"*, Journal of Frontiers of Architecture and Civil Engineering in China, Vol. 2, pp. 318-322, 2008.

[Chung and Alonso, 2004] H. S. Chung, J. J. Alonso, "*Multiobjective Optimization Using Approximation Model-Based Genetic Algorithms*", 10th AIAA/ISSMO Symposium on Multidisciplinary Analysis and Optimization, VA 20191-4344, American Institute of Aeronautics and Astronautics, New York, p. 16, 2004.

[Corne and Knowles, 2000] D. W. Corne, J. D. Knowles, "*The Pareto-Envelope Based Selection Algorithm for Multiobjective Optimization*". In Proceedings of the Sixth International Conference on Parallel Problem Solving from Nature, Springer-Verlag, Berlin, pp 839-848, 2000.

[Crosetti, 1970] P. A. Crosetti, "*Fault Tree Analysis with Probability Evaluation*", IEEE Nuclear Power systems Symposium, Nov, pp. 465-471, 1970.

[Davis, 1991] L. Davis, "*Handbook of Genetic Algorithms*", Van Nostrand Reinhold, 1991.

[Deb et al., 2000] K. Deb, S. Agrawal, A. Pratap, T. Meyarivan, "*A Fast Elitist Non-Dominated Sorting Genetic Algorithm for Multi-Objective Optimization: NSGA-II*", Report No. 200001, Kanpur Genetic Algorithms Laboratory (KAnGAL), India, 2000.

[Deitel, 2003] H. M. Deitel, P. J. Deitel, "*C++ How to Program*", Pearson Education, 2003.

[De Jong (1), 1975] K. A. De Jong, "*Analysis of the behaviour of a class of genetic adaptive systems*", PhD thesis, Dept. of Computer and Communications Sciences, University of Michigan, Ann Arbour, 1975.

[De Jong (2), 1975] K. A. De Jong, "*Genetic Algorithms: a 10 Year Perspective*", Conference on Gas and their Applications, 1975.

[Dhillon, 1978] B. S. Dhillon, "*Bibliography of Literature of Fault Trees*", Microelectronics and Reliability, Vol. 17, pp. 501-503, 1978.

[Dias and Vasconcelos, 2002] A. H. F. Dias, J. A. de Vasconcelos, "*Multiobjective Genetic Algorithms Applied to Solve Optimization Problems*", IEEE Transactions on Magnetics, Vol. 38, No. 2, pp. 1133-1136, 2002.

[Dueck, 1993] G. Dueck. "*New Optimization Heuristics The Great Deluge Algorithm and the Record-to-Record Travel*", Journal of Computational Physics, Volume 104, Issue 1, p. 86-92, 1993.

[Dueck and Scheuer, 1990] G. Dueck and T. Scheuer. "*Treshold Accepting: A General Purpose Optimization Algorithm Appearing Superior to Simulated Annealing*", Journal of Computational Physics, Volume 90, Issue 1, pp. 161-175, 1990.

[Duffin et al., 1967] R. J. Duffin, E. Peterson, C. Zener, "*Geometric programming*", Wiley, New York, 1967.

[Elegbede and Adjallah, 2003] C. Elegbede, K. Adjallah, "*Availability Allocation to Repairable Systems with Genetic Algorithms: a Multi-objective Formulation*", Reliability Engineering and System Safety 82, pp. 319-330, 2003.

[Everinhgam et al., 2003] M. Everinhgam, H. Muller, B. Thomas, "*Evaluating Image Segmentation Algorithms Using the Pareto Front*", Report sponsored by Hewlett-Packard Research Laboratories, Dep. of Computer Science, University of Bristol, 2003.

[Everson and Fieldsend, 2006] R. M. Everson, J. E. Fieldsend, "*Multiobjective Optimization of Safety Related Systems: An Application to Short-Term conflict Alert*", IEEE Transactions on Evolutionary Computation, University of Exeter, UK, pp. 1-12, 2006.

[Fleming and Purshouse, 2001] P. J. Fleming, R. C. Purshouse, "*Genetic Algorithms in Control System Engineering*", Research Report No. 789, Dep. of Automatic Control and Systems Engineering, University of Sheffield, 2001.

[Gen and Cheng, 1997] M. Gen, R. Cheng, "*Genetic Algorithms and Engineering Design*", John Willey & Sons, 1997.

[Glover, 1990] F. Glover, "*Tabu Search: A Tutorial*", *Interfaces*, 20(4): 74-94, 1990.

[Goldberg, 1989] D. E. Goldberg, "*Genetic Algorithms in Search. Optimization and Machine Learning*", Addison-Wesley Publishing Company, 1989.

[Grosan et al., 2002] C. Grosan, M. Oltean, D. Dumitrescu, M. Oltean, "*A Modified PAES Algorithm Using Adaptive Representation of Solutions*", Report, Faculty of Mathematics and Computer Science, University of Cluj Napoca, Romania, 2002.

[Greiner et al., 2003] D. Greiner, B. Galvan, G. Winter, "*Safety Systems Optimum Design by Multicriteria Evolutionary Algorithms*", Springer-Verlag, EMO 2003, LNCS 2632, pp. 722-736, 2003.

[Haimes, 1999] Y. Y. Haimes, "*Water System Complexity and the Misuse of Modeling and Optimization*", Centre for Risk Management for Engineering Systems, University of Virginia, pp. 36-41, 1999.

[Hodson, 2001] R. J. W. Hodson, "*Memetic Algorithm Approach to Thin-Film Optical Coating Design*", Report, Dep. of Physics, University of Ottawa, Canada, 2001.

[Holland, 1975] J. Holland, "*Adaptation in Natural and Artificial Systems*", University of Michigan Press, 1975.

[Horn and Nafpliotis, 1993] J. Horn, N. Nafpliotis, "*Multiobjective Optimization Using the Niched Pareto Genetic Algorithms*", IlliGAL Report 93005, Illinois Genetic Algorithms Laboratory, University of Illinois, Urbana, Champaign, 1993.

[Hu et al, 2004] X. Hu, R. Eberhart, Y. Shi., "*Recent advances in particle swarm*", IEEE Congress on Evolutionary Computation 2004, Portland, Oregon, USA.

[Huang et al., 2009] Y. Huang, C. Liang and Y. Yang. "*The Optimum Route Problem by Genetic Algorithm for Loading/Unloading of Yard Crane*", Computers & Industrial Engineering, Vol. 56, Issue 3, pp. 993-1001, 2009.

[Jensen, 2003] M. T. Jensen, "*Reducing the Run-Time Complexity of Multiobjective EAs: The NSGA-II and Other Algorithms*", IEEE Transactions on Evolutionary Computation, vol. 7, No. 5, pp. 503-515, October 2003.

[Kamiura et al., 2002] J. Kamiura, T. Hiroyasu, M. Miki, S. Watanabe, "*MOGADES: Multi-objective Genetic Algorithm with Distributed Environment Scheme*", Proceedings of the 2nd International Workshop on Intelligent Systems Design and Applications: ISDA-02, pp. 143-148, 2002.

[Karr and Freeman, 1999] C. L. Karr, L. M. Freeman, "*Industrial Applications of Genetic Algorithms*", CRC Press, 1999.

[Khare et al., 2003] V. Khare, X. Yao, K. Deb, "*Performance Scaling of Multiobjective Evolutionary Algorithms*", Springer-Verlag Berlin Heidelberg, LNCS 2632, pp. 376-390, 2003.

[Koza, 1994] J. R. Koza, "*Genetic Programming*", The MIT Press, 1994.

[Lapa et al., 2003] C. M. F. Lapa, C. M. N. A. Pereira, P. F. Frutuoso e Melo, "*Surveillance Test Policy Optimization Through Genetic Algorithms Using Non-periodic Intervention Frequencies and Considering Seasonal Constraints*", Reliability Engineering and System Safety 81, pp. 103-109, 2003.

[Lawrence, 1996] J. F. Lawrence, T. Back and P. J. Angeline. "*Evolutionary Programming V: Proceedings of the Fifth Annual Conference on Evolutionary Programming*", MIT Press, p. 504, 1996.

[Lee and El-Sharkawi] K. Y. Lee, M. A. El-Sharkawi, "*Modern Heuristic Optimization Techniques with Applications to Power Systems*", Willey-IEEE Press, 2006.

[Lin et al., 2008] C. S. Lin, Y. C. Liao, Y. L. Lay, K. C. Lee, M. S. Yeh. "*High-Speed TFT LCD Defect-Detection System with Genetic Algorithm*", Assembly Automation Journal, Vol. 28, pp. 69-76, 2008.

[Lu, 2002] H. Lu, "*Dynamic Population Strategy Assisted Particle Swarm Optimization in Multiobjective Evolutionary Algorithm Design*", Report sponsored by IEEE Neutral Network Society, School of Electrical and Computer Engineering, Oklahoma State University, 2002.

[Marseguerra et al., 2004] M. Marseguerra, E. Zio, L. Podofillini, "*A Multiobjective Genetic Algorithm Approach to the Optimization of the Technical Specifications of a nuclear safety system*", Reliability Engineering and System Safety 84, pp. 87-99, 2004.

[Martorell et al., 2004] S. Martorell, A. Sanchez, S. Carlos, V. Serradell, "*Alternatives and Challenges in Optimizing Industrial Safety Using Genetic Algorithms*", Reliability Engineering and System Safety 86, pp. 25-38, 2004.

[Martorell et al., 2005] S. Martorell, J. F. Villanueva, S. Carlos, Y. Nebot, A. Sanchez, J. L. Pitarch, V. Serradell, "*RAMS + C Informed Decision-making with Application to Multi-objective Optimization of Technical Specifications and Maintenance Using Genetic Algorithms*", Reliability Engineering and System Safety 87, pp. 65-75, 2005.

[Martorell et al., 2006] S. Martorell, S. Carlos, J. F. Villanueva, A.I. Sanchez, B. Galvan, D. Salazar, M. Cepin, "*Use of Multiple Objective Evolutionary Algorithms in Optimizing Surveillance Requirements*", Reliability Engineering and System Safety 91, pp. 1027-1038, 2006.

[Morse, 1980] J. N. Morse, "*Reducing the size of the nondominated set: Pruning by clustering*". Computer and Operations research, 7(1-2), pp. 55-66, 1980.

[Munoz et al., 1997] A. Munoz, S. Martorell, V. Serradell, "*Genetic Algorithms in Optimizing Surveillance and Maintenance of Components*", Reliability Engineering and System Safety 57, pp. 107-120, 1997.

[Ortmeier et al, 2005] F. Ortmeier, W. Reif, G. Schellhorn, "*Safety Optimization of a Radio-based Railroad Crossing*", University of Augsburg, Germany, 2005.

[Pattison, 1999] R. Pattison, "*Safety System Design Optimization*", PhD thesis, Dept. of Mathematical Sciences, Loughborough University, 1999.

[Platz and Olsen, 1976] O. Platz, J.V. Olsen, "*FAUNETL: A Program Package for the Evaluation of Fault Trees and Networks*", Riso Report No 348, DK-4000, Roskilde Denmark, 1976.

[Pontryagin, 1990] L. S. Pontryagin. "*Optimal Control and Differential Games*", American Mathematical Society, p.278, 1990.

[Price and Storn, 1997] K. Price, R. Storn, "*Differential Evolution – A simple evolution strategy for fast optimization*", Dr. Dobb's Journal, 22(4), pp. 18-24 and 78, 1997.

[Pukkala and Kurttila, 2005] T. Pukkala, M. Kurttila, "*Examining the performance of six heuristic optimisation techniques in different forest planning problems*", Silva Fennica 39(1), pp. 67–80, 2005.

[Pun, 1969] L. Pun, "*Introduction to Optimization Practice*", John Wiley & Sons, 1969.

[Purshouse and Fleming, 2001] R. C. Purshouse, P. J. Fleming, "*The Multiobjective Gebetic Algorithm Applied to Behchmark Problems in Analysis*", Research Report No. 796, Dep. of Automatic Control and System Engineering, University of Shieffield, 2001.

[Rao, 1996] S. S. Rao, "*Engineering Optimization. Theory and Practice. Third Edition*", John Wiley & Sons, 1996.

[Rauzy, 1993] A. Rauzy, "*New Algorithm for Fault Tree Analysis*", Reliability Engineering and System Safety, Vol. 40, pp. 203-211, 1993.

[Rauzy and Dutuit, 1996] A. Rauzy, Y. Dutuit, "*A LinearTime Algorithm to Find Modules in Fault Trees*", IEEE Trans Reliability, Vol. 45, No. 3, pp. 422-425, 1996.

[Reif, 2003] F. O. W. Reif, "*Safety Optimization: A Combination of Fault Tree Analysis and Optimization Techniques*", Ausburg, pp. 1-8, 2003.

[Remenyte-Prescott, 2007] R. Remenyte-Prescott. "*System Failure Modelling Using Binary Decision Diagrams*", PhD thesis, AAE Department, Loughborough University, 2007.

[Riauke and Bartlett, 2008] J.Riauke, L.M. Bartlett, *An Offshore Safety System Optimization using a SPEA2 based approach*, Proc. IMechE Vol. 222 Part O: J. Risk and Reliability, JRR113, pp. 271-282, 2008.

[Riauke and Bartlett, 2009 (1)] J. Riauke, L. M. Bartlett. *Safety System Design Optimization using Multi-objective Genetic Algorithm,* International Journal of Reliability and Safety (IJRS), Vol. 3, No. 4, pp. 397-412, DOI: 10.1504/IJRS.2009.028584, 2009.

[Riauke and Bartlett, 2009 (2)] J. Riauke, L. M. Bartlett. *An Integrated Design optimization Approach for Systems with Dependencies*, proceedings of the 18th AR^2TS, pp. 455-466 , 2009.

[Salazar et al., 2006] D. Salazar, C. M. Rocco, B. J. Galvan, "*Optimization of Constrained Multiple-objective Reliability Problems Using Evolutionary Algorithms*", Reliability Engineering and System Safety 91, pp. 1057-1070, 2006.

[Sareni et al., 2004] B. Sareni, J. Regnier, X. Roboam, "*Recombination and Self-Adaptation in Multi-objective Genetic Algorithms*", Springer-Verlag Berlin Heidelberg, LNCS 2936, pp. 115-126, 2004.

[Sbalzarini et al., 2000] I. F. Sbalzarini, S. Muller, P. Koumoutsakos, "*Multiobjective optimization using evolutionary algorithms*", Center for Turbulence Research, Proceedings of the Summer program 2000, pp. 63-74, 2000.

[Seward et al, 2000] D. Seward, C. Pace, R. Morrey, I. Sommerville, "*Safety Analysis of Autonomous Excavator Functionality* ", Reliability Engineering and System Safety 70, pp. 29-39, 2000.

[Shi and Lee, 1997] Y. Shi, H. Lee, "*A Binary Integer Linear Program with Multi-Criteria and Multi-Constraint Levels* ", Computer and operational Research, Vol. 24, No. 3, pp. 259-273, 1997.

[Sobieski, 1988] J. Sobieski, "*Optimization by Decomposition: A Step from Hierarchic to Non-Hierarchic Systems*", NASA Conference Publication 3031, Part1, Second NASA/Air Force Symposium on Recent Advances in Multidisciplinary Analysis and Optimization, Hampton, Virginia, September 1988.

[Stelmack et al, 1999] M. A. Stelmack, S. M. Batill, B. C. Beck, "*Design of an Aircraft Brake Component using an Interactive Multidisciplinary Design Optimization Framework*", Department of Aerospace and Mechanical Engineering, University of Notre Dame, Notre Dame, Indiana, pp. 1-9.

[Sun, 2006] H. Sun, "*System Dependency Modelling*", PhD thesis, Dept. of Mathematical Sciences, 2006.

[Tillman et al., 1980] F. A. Tillman, C. L. Hwang, W. Kuo, "*Optimization of System Reliability*", Industrial Engineering, Marcel Dekker, 1980.

[Tsai et al, 2001] Y. T. Tsai, K. S. Wang, H. Y. Teng, "*Optimizing Preventive Maintenance for Mechanical Components Using Genetic Algorithms*", Reliability Engineering and System Safety 74, pp. 89-97, 2001.

[Vinod et al., 2004] G. Vinod, H. S. Kushwaha, A. K. Verma, A. Srividya, "*Optimization of ISI Interval Using Genetic Algorithms for Risk Informed In-Service Inspection*", Reliability Engineering and System Safety 86, pp. 307-316, 2004.

[Watson, 1969] H. A. Watson*, "Microwave Semiconductor Devices and Their Circuit Applications",* McGraw-Hill, p.617, 1969.

[Wilde and Beightler, 1967], D. J. Wilde, C. S. Beightler, "*Foundations of Optimization*", Prentice-Hall, inc. Englewood Cliffs, N. J., 1967.

[Yamachi et al., 2006] H. Yanachi, Y. Tsujimura, Y. Kambayashi, H. Yamamoto, "*Multi-objective Genetic Algorithm for Solving N-version Program Design Problem*", Reliability Engineering and System Safety, pp. 1083-1094, 2006.

[Yen and Lu, 2002] G. G. Yen, H. Lu, "*Dynamic Population Size in Multiobjective Evolutionary Algorithm*", in Proceedings 9th IEEE Cong. Evol. Comput., pp.1648-1653, 2002.

[Zitzler et al. (1), 2001] E. Zitzler, K. Deb, L. Thiele, "*Comparison of Multiobjective Evolutionary Algorithms: Empirical Results*", Evolutionary Computation 8(2), pp. 173-195, 2001.

[Zitzler et al. (2), 2001] E. Zitzler, M. Laumanns, L. Thiele, "*SPEA2: Improving the Strength Pareto Evolutionary Algorithm*", Computer Engineering and Communication Network Lab (TIK), Swiss Federal Institute of Technology, TIK-Report No. 103, 2001.

[Zitzler and Thiele, 1998] E. Zitzler, L. Thiele, "*An Evolutionary Algorithm for multiobjective Optimization: The Strength Pareto Approach*", Computer Engineering and Communication Network Lab (TIK), Swiss Federal Institute of Technology, TIK-Report No. 43, 1998.

APPENDIX A

A.1 HIPS Unavailability Fault Tree Structure

The top event of the HIPS unavailability fault tree represents the causes of the system failing to protect the processing equipment. The top event 'Safety system fails to protect' will occur if all (Wing, Master, ESD and HIPS) valves along the pipeline fail to close. Consequently, sub-events 'Wing and Master valves fail to protect' (*G1*), 'ESD valves fail to protect' (*G2*) and 'HIPS valves fail to protect' (*G3*) related by AND logic are immediate, necessary and sufficient. The structure of the top gate is shown below in figure A.1.

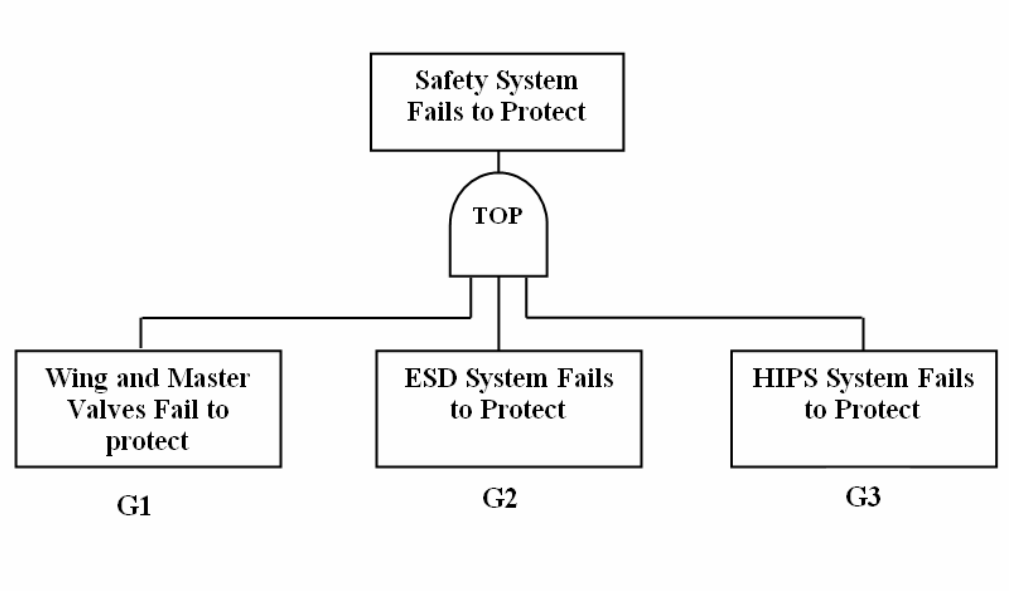


Figure A.1 TOP Event of HIPS Unavailability Fault Tree

Each valve is a fail safe type which can be defined as 'air to open'. The principle of each valve working can be described as follows: when the pressure in the pipeline is at an appropriate level, the solenoid of the valve is maintained by computer logic in an energised state, the pneumatic line remains pressurised and the associated actuator retains the valve in the open state. When pressure in the pipeline increases, pressure transmitters detecting that fact transmit a signal to the computer. In the case when the pressure increase exceeds the acceptable level the function of the computer is to cause the circuit of the output channel to the solenoid to open. This circuit can be broken by two relay contacts which introduce a level of redundancy. As a result, the solenoid is

de-energised and a vent valve activated. Consequently, pressure drops to the actuator, causing the valve to close.

A.1.1 Main Sub-events Structure

In this section the principle structure of each of the HIPS unavailability fault tree sub-events is described. These are: 'Wing and Master Valves Fail to Protect', 'ESD System Fails to Protect' and 'HIPS System Fails to Protect'.

Sub-event 1. 'Wing and Master Valves Fail to Protect'

The Wing and Master valves, their solenoids, the relay contacts and the computer logic are fixed components of subsystem 1 and, consequently, they constitute part of each potential design variation.

If either the Wing or Master valves close, it will protect against a high-pressure surge in the pipeline. However, the sub-event 'Wing and Master valves fail to protect' will occur only if both mentioned valves fail.

Failure of the Wing valve can be defined as: the Wing valve fails to protect if either the Wing valve fails itself to protect, or the pneumatic line to the actuator of the Wing valve remains pressurised. Basic event '*WV*' is used in the fault tree (Figure A.2) to describe the situation when the Wing valve itself fails. The pneumatic line remains pressurised due to failure of the solenoid valve (basic event '*SWV*') or is a result of the solenoid staying energised.

The alternative fault tree branch 'Master valve fails to protect' is developed in similar manner to the Wing valve. Figure A.2 shows the intermediate fault tree structure of the 'Wing valve Fails to Protect'.

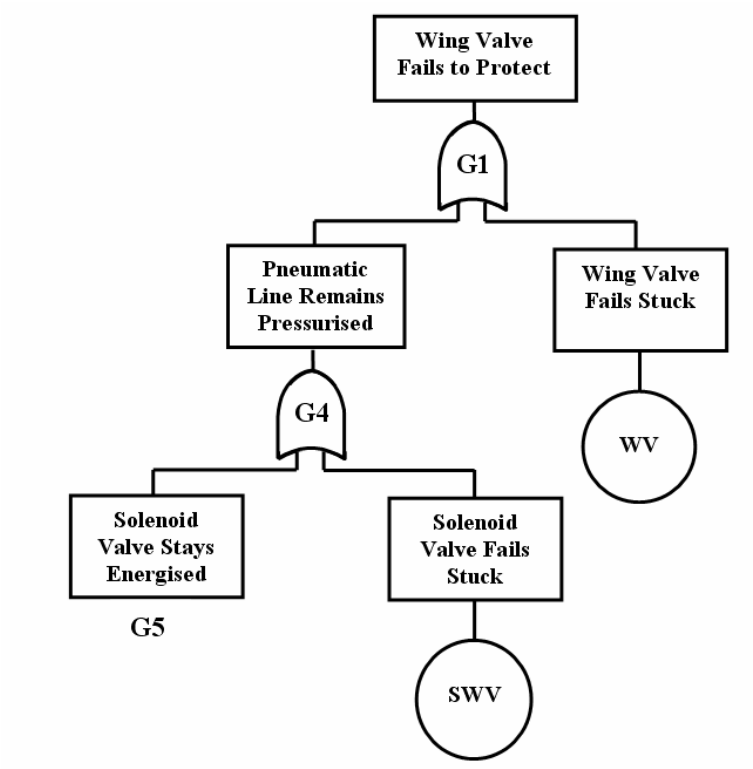


Figure A.2 Structure of the Branch 'Wing Valve Fails to Protect'

Each of the Wing and Master valves has solenoids. Therefore, the event 'Solenoid stays energised' is common to branches developed for both valves. In figure A.2 this event is shown as gate 5 (*G5*). It represents a failure in the flow of fault logic from detection of increased pressure by the pressure transmitters to de-energising of the solenoids of the Wing and Master valves. Therefore, the solenoid stays energised if both relay contacts fail to break the circuit from the computer. Basic event '*R1/1*' defines the failure of relay contact 1, analogically, basic event '*R1/2*' defines the failure of relay contact 2. Also the solenoid stays energised if the computer fails to send the trip. The latter happens if the computer does not receive input to signify an increase of pressure, i.e. if the trip signal to subsystem 1 is not received or if the computer logic fails (basic event '*PLC1*'). Figure A.3 shows the causal relationship from gate 5 (*G5*) to 'Subsystem 1 fails to indicate trip'.

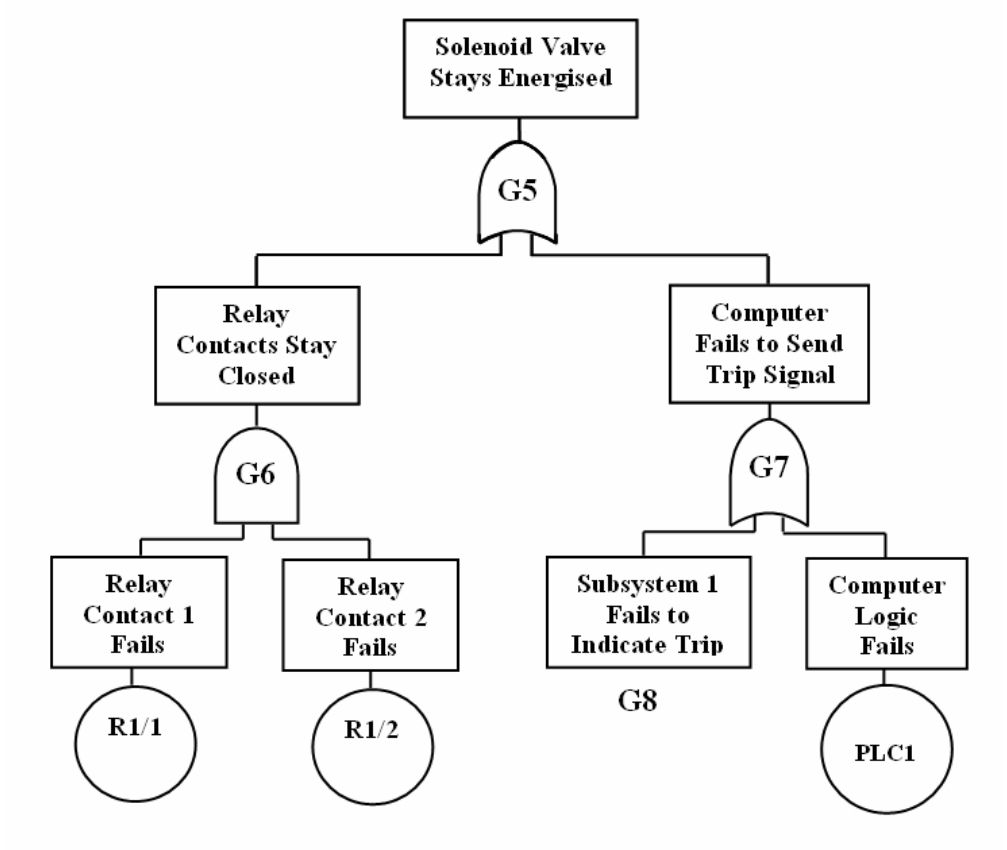


Figure A.3 Structure of the Branch 'Solenoid Valve Stays Energised'

The pressure transmitters are used to monitor the system pressure level, hence, are very important for the safety system. Failure of the pressure transmitters of subsystem 1 to detect a pressure beyond the acceptable limit prevents action of the computer logic associated with subsystem 1 and, as a result, any subsequent events.

The number of pressure transmitters for subsystem 1 is N_1. It is a design variable with range 1 to 4. According to the value of this parameter the structure of the fault tree will alter. Therefore, 'Subsystem 1 Trip Signal Not Received' is dependent upon whether 1 pressure transmitter is installed, or 2, or 3, or 4. Associated with each option is a House Event. Table A.1 introduces the house events used to model these structural changes.

The house event obtains a 'TRUE' value, if the condition holds. Therefore, the event is assigned a probability of 1. Otherwise, the probability is 0. As an example, if the design is such that subsystem 1 has 2 pressure transmitters fitted, house events *EN2*, *NEN1*, *NEN3* and *NEN4* would be assigned a probability of 1 and the rest set to 0.

Table A.1 House Events for Pressure Transmitters in Subsystem1

House Event	Description for Subsystem 1
EN1	1 pressure transmitter is fitted
EN2	2 pressure transmitters fitted
EN3	3 pressure transmitters fitted
EN4	4 pressure transmitters fitted
NEN1	Number of fitted pressure transmitters is not 1
NEN2	Number of fitted pressure transmitters is not 1
NEN3	Number of fitted pressure transmitters is not 1
NEN4	Number of fitted pressure transmitters is not 1

To help to reduce the number of branches within the fault tree the use of 'No Pressure Transmitters fitted' options are included, as shown in figure A.4. Hence, the sub-event 'Subsystem 1 Fails to Indicate Trip' has four inputs connected with an AND gate: inputs are either activated as failure of that branch or not a number of pressure transmitters fitted.

Consider the design such that subsystem 1 has 2 pressure transmitters fitted. House events *NEN1*, *NEN3* and *NEN4* would be set to TRUE with probability of 1, hence, *NEN2* would be set to zero, thus activation of channel 2 would be dependent on the sub-event 'Fail to indicate trip 2 PT fitted'.

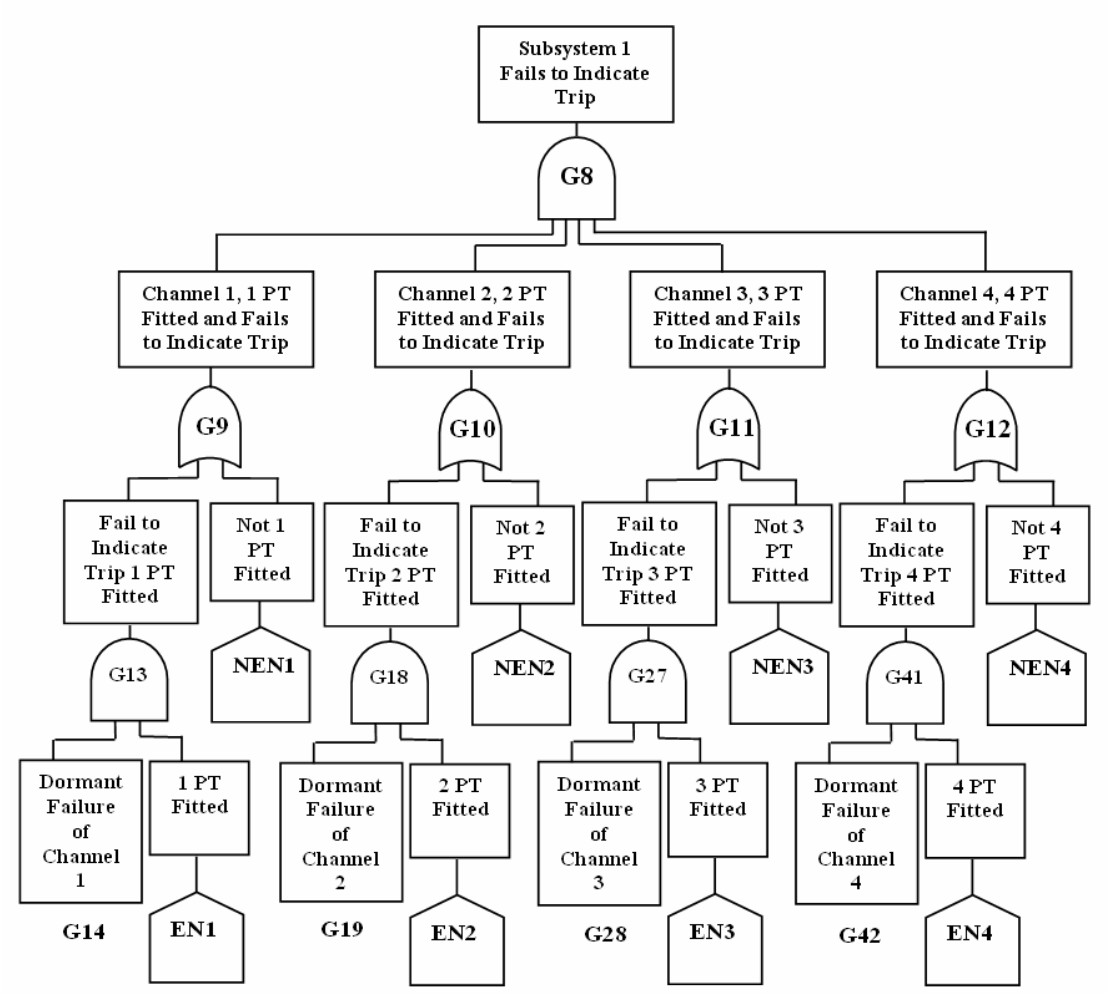


Figure A.4 Structure of the Branch 'Subsystem 1 Fails to Indicate Trip'

The number of pressure transmitters in subsystem 1 is only one of the variables that must be considered. Another important variable is the number of pressure transmitters required to trip the system (K_1). This variable can obtain a value in the range 1 to N_1. Furthermore, consideration must be given to the type of pressure transmitter (P_1 or P_2). These parameters give new possible variations, and must be modelled in each channel. Additional house events to model further structural changes are defined in table A.2:

Table A.2 Additional House Events for Pressure Transmitters in Subsystem1

House Event	Description
EK1	1 pressure transmitter is required to trip subsystem 1
EK2	2 pressure transmitter is required to trip subsystem 1
EK3	3 pressure transmitter is required to trip subsystem 1
EK4	4 pressure transmitter is required to trip subsystem 1
P11	Pressure transmitter type is 1
P12	Pressure transmitter type is 2

To illustrate how the fault tree for each dormant failure channel is constructed, channel 1 is explained. As only one PT is fitted there must be the same number to trip. Therefore, the K_1 value must be 1. A house event is included to account for this option. For the other channels, this section would include options of 1 PT to trip or 2 PT to trip, or 3 or 4 with corresponding house events. The development of channel 1 is shown below in figure A.5.

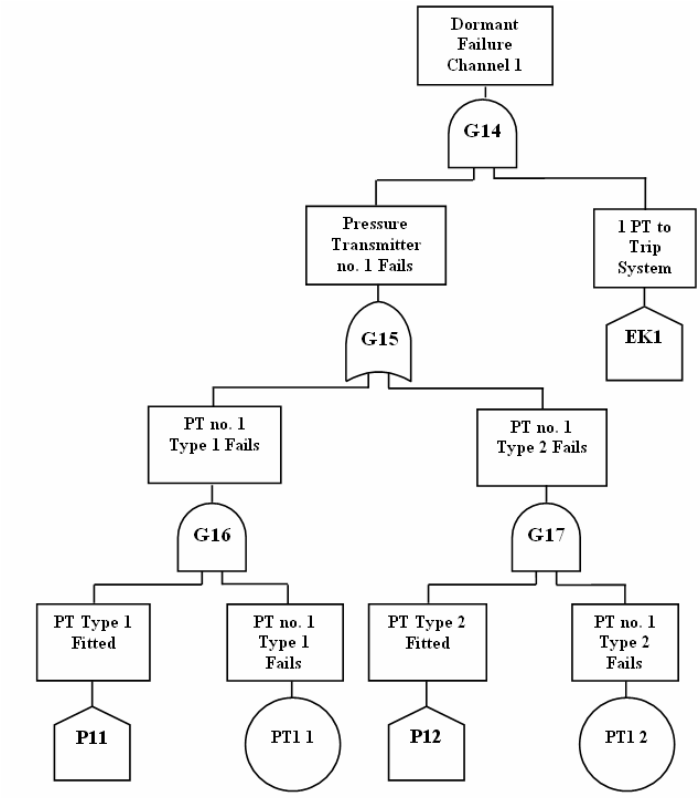


Figure A.5 Structure of the Branch 'Dormant Failure of Channel 1'

To analyse 'Pressure transmitter No1 fails' it is dependent on the type of PT fitted. The options are that type 1 is fitted or type 2. For each 'Type' failure, it is due to both the house event being set to TRUE and the pressure transmitter itself failing. For type 1 failure this is house event '*P11*' ('Pressure transmitter type 1 is fitted') AND basic event '*PT11*' which means the failure of pressure transmitter number 1 type 1 occurs. Type 2 has the same format. It is assumed, that if more that one pressure transmitter is fitted in the system, their type is the same.

Similarly, channel 2 models the system design with 2 pressure transmitters. Therefore, K_1 can be either 1 or 2. One pressure transmitter acts as a redundant component. Hence, to fail to trigger channel 2 both PT must fail (hence the AND gate, *G22*). When $K_1 = 1$ one pressure transmitter is required to trip the system. The failure when $K_1 = 2$, means that either PT can fail (hence the OR gate, *G26*). Further development of channel 2 is shown in figure 5.6.

Likewise, channel 3 must model the system design with three pressure transmitters. Hence, K_1 can be equal to 1, 2 or 3. Possible combinations are either 3 from 3, any combination of 2 from 3, or failure of 1 pressure transmitter alone. They will trigger dormant failure of channel 3, where $K_1 = 1$, $K_1 = 2$ or $K_1 = 3$ respectively. In a similar manner, additional basic events '*PT31*' and '*PT32*' are introduced. They represent failure of pressure transmitter number 3 either type 1 or 2. Analogically, channel 4 must account for 1 through to 4 pressure transmitters being able to trip the system. The development of this section is similar to the ones already discussed.

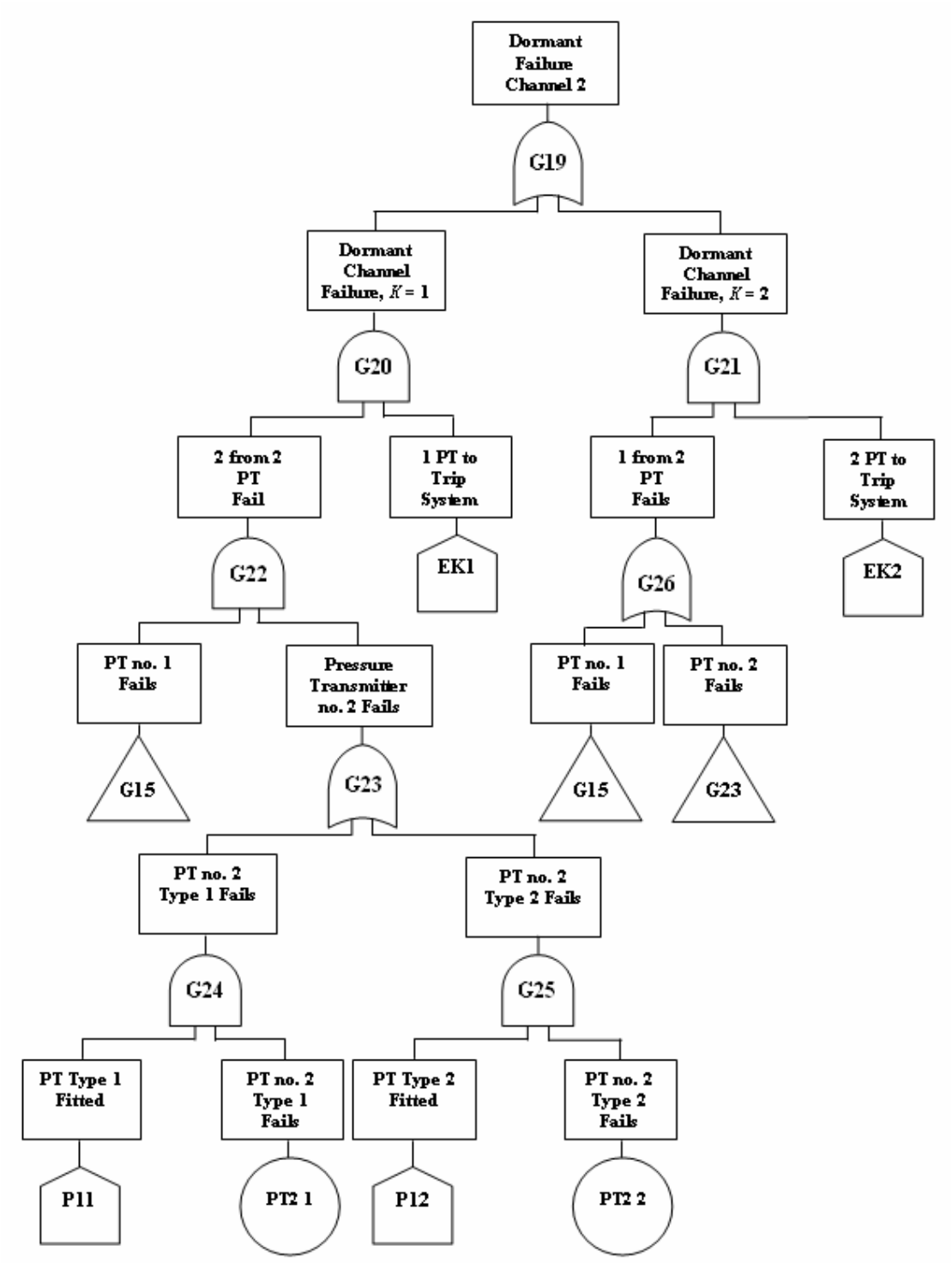


Figure A.6 Structure of the Branch 'Dormant failure of channel 2'

<u>Sub-event 2.</u> 'ESD Subsystem Fails to Protect'

This sub-event heads the second branch of the HIPS fault tree in figure A.1. The event 'ESD valves fail to protect' is developed in a very similar manner to the Wing and Master valves. However, ESD valves are not a fixed part of the system design. That is why some structural differences arise. The number of ESD valves (*E*) is a variable ranging from 1 to 2. House events are included to switch on and off relevant options (Table A.3).

Table A.3 House events for ESD valves in Subsystem1

House Event	Description
E1	Subsystem 1 has 1 ESD valve fitted
E2	Subsystem 1 has 2 ESD valve fitted
NE1	The number of ESD valves for subsystem 1 is not 1
NE2	The number of ESD valves for subsystem 1 is not 2
V1	ESD valve type 1 is fitted
V2	ESD valve type 2 is fitted

The ESD subsystem fails to protect if both ESD valve 1 and ESD valve 2 fail to protect. Failure of each ESD valve can occur in two cases: if the ESD valve is not fitted or the ESD valve is fitted and fails (shown in figure A.7).

The 'ESD valve fails to close' if either the ESD valve itself fails stuck or the pneumatic line to the valve remains pressurised. Pressurisation of the pneumatic line occurs due to failure of the solenoid of the ESD valve or the solenoid staying energised. Furthermore, failure of the ESD valve depends directly on the valve type fitted which can be either V_1 or V_2. The casual relationship, including the representation with house events, is shown in figure A.8.

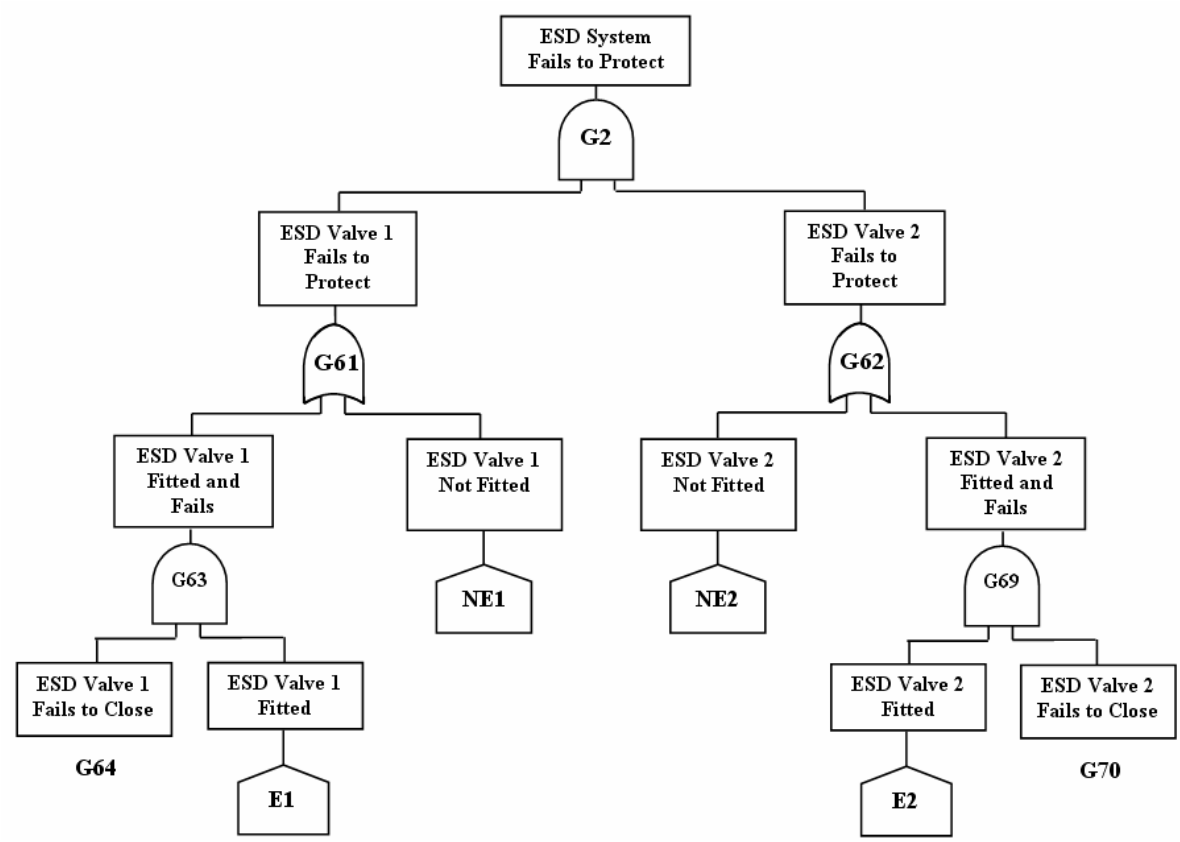


Figure A.7 Structure of the Branch 'ESD System Fails to Protect'

Computer logic, relay contacts and any pressure transmitters fitted is utilised by the passage of fault tree logic from detection of increased pressure to de-energising the solenoid.

These basic events are similar to those which transmit fault logic and trigger the wing and master valve to close. The event 'Solenoid stays energised' is, therefore, identical to that described in the previous section. Its branching structure is merely replicated within the ESD structure of the tree. The fault tree structure representing the failure of ESD valve 1 to close is shown in figure A.8.

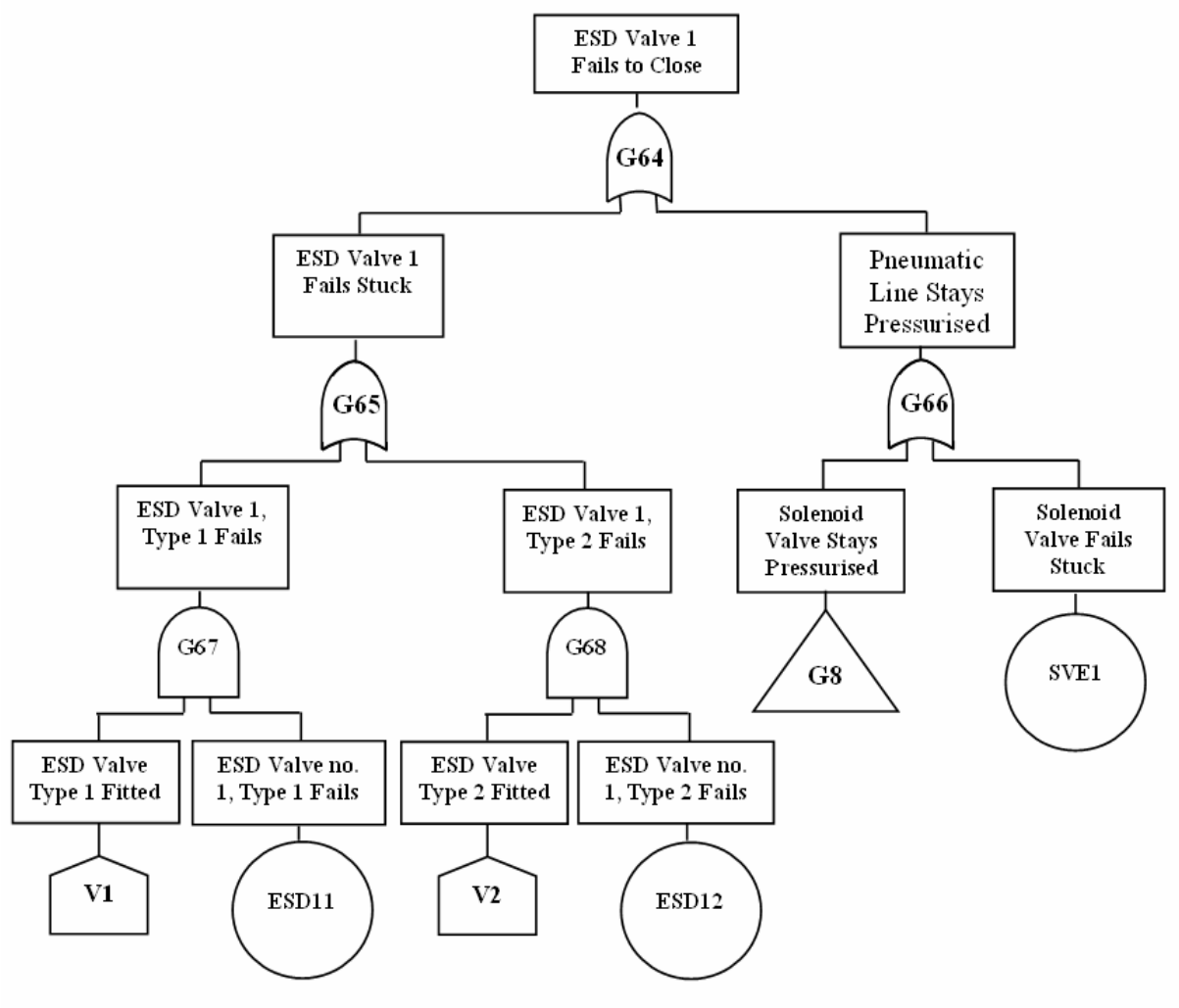


Figure A.8 Structure of the Branch 'ESD Valve 1 Fails to protect'

<u>Sub-event 3.</u> 'HIPS Fails to Protect'

The HIPS and ESD subsystems are identical in their working manner. Similarly to the ESD subsystem, the HIPS subsystem fails if all fitted HIPS valves fail. However, the HIPS system is completely independent in its operation to the ESD, Wing or Master valves. Variable *H* identifies the number of HIPS valves ranges from 1 to 2. Table A.4 represents the house events, which are included to switch on and off relevant parts of the tree:

Table A.4 House Events for HIPS Valves in Subsystem 1

House Event	Description
H1	Subsystem 1 has 1 HIPS valve fitted
H2	Subsystem 1 has 2 HIPS valve fitted
NH1	The number of HIPS valves for subsystem 1 is not 1
NH2	The number of HIPS valves for subsystem 1 is not 2
HV1	Fitted HIPS valve type is 1
HV2	Fitted HIPS valve type is 2

Failure logic for the HIPS system completes the fault tree development. It is developed in an identical manner to that described for the ESD subsystem, with all components and computer logic and pressure transmitters being distinct. Calculations of the unavailability are then achieved by setting the corresponding house events and using the dormant failure and repair data (Table 5.2). The calculation procedure is the standard quantification approach used for fault trees.

A.2 HIPS Spurious Frequency

A.2.1 HIPS Spurious Trip Fault Tree Structure

Consideration is given to a second system failure mode, spurious activation of the HIPS, due to the constraint limiting the number of spurious system trips permitted. This culminating event requires a specific fault tree to be developed. If any one of the valves included along the pipeline closes the top event will occur.

Each valve is an ‘air to open’ safety type. The casual relationship ‘HIPS fails spuriously’ is shown in figure A.9. The immediate, necessary, and sufficient sub-events to the top event are ‘Wing or Master Valve Fails Spuriously’ (*G1*), ‘ESD Subsystem Fails Spuriously’ (*G2*) and ‘HIPS Subsystem Fails Spuriously’ (*G3*). They all are related by OR logic.

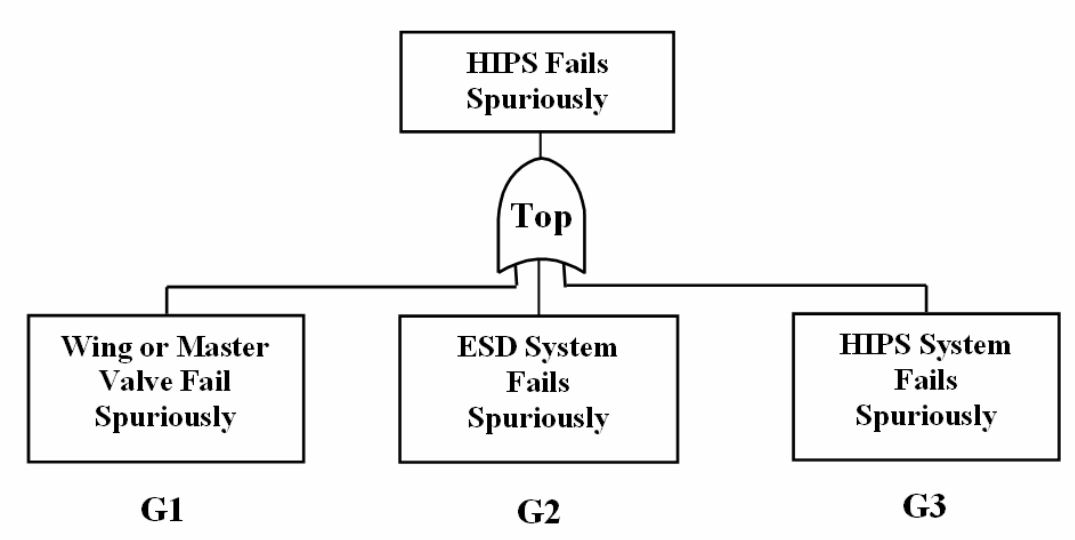


Figure A.9 Structure of the Top Event of HIPS Spurious Trip Fault Tree

Obviously, the main difference between the HIPS spurious trip fault tree (Figure A.9) and the HIPS unavailability fault tree (Figure A.1) is the lack of redundancy in the first level of the tree structure. It is important to notice that house events incorporated in the HIPS spurious trip fault tree structure and the house events from the first level of the HIPS unavailability fault tree are consistent with each other. Furthermore, the structural characteristics of the design remain the same in each case. However, the failure modes of the sub-events are different.

A.2.2 HIPS Spurious Trip Fault Tree Main Sub-events Structure

Sub-event 1. 'Wing or Master Valve Fails Spuriously'

Spurious system failure occurs if either wing or master valve fails spuriously. Therefore, these are the next level intermediate events. Consider the sub-event 'Wing Valve Closes Spuriously'. It closely resembles the event of wing valve failing stuck in the system unavailability fault tree. The components and their relationship in the branch are almost the same, with the only difference being the failure mode, that of spurious activation. Moreover, spurious action of either relay contact instigates the flow logic and trips the system (Figure A.10).

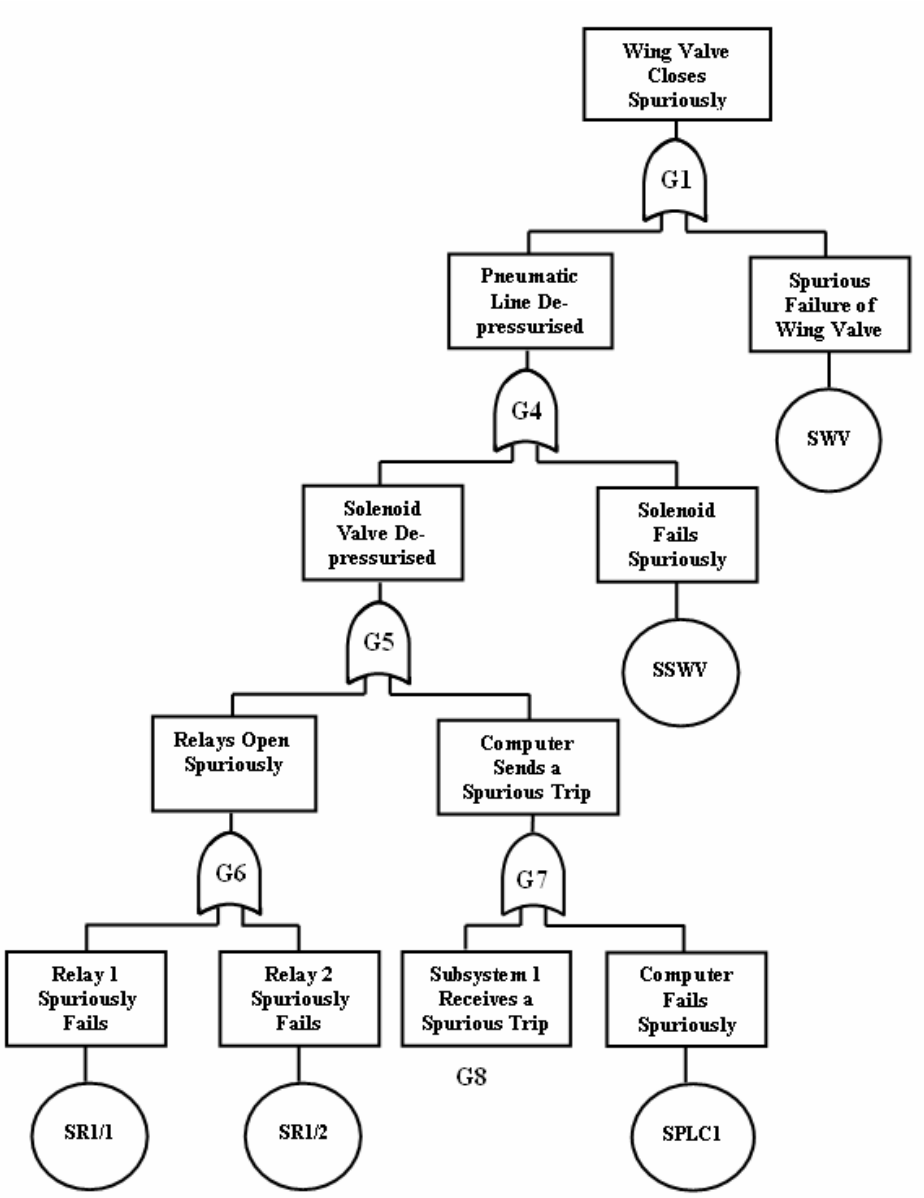


Figure A.10 Structure of the Branch 'Wing Valve Fails Spuriously'

Structural differences arise in fault tree gate *G8*. Sub-events are again portioned into 4 separate channel failures. Channel 1 indicates inclusion of 1 pressure transmitter and so on. OR logic associates the channels and spurious channel failure occurs if the relevant number of pressure transmitters are fitted, as defined by house events *EN1*, *EN2*, *EN3* or *EN4*, and the relevant trip combinations occurs, i.e. house events *EK1*, *EK2*, *EK3* or *EK4*. From figure A.11 it is clear that the mutually exclusive house event pairings, e.g. *NEN1* and *EN1*, are not required.

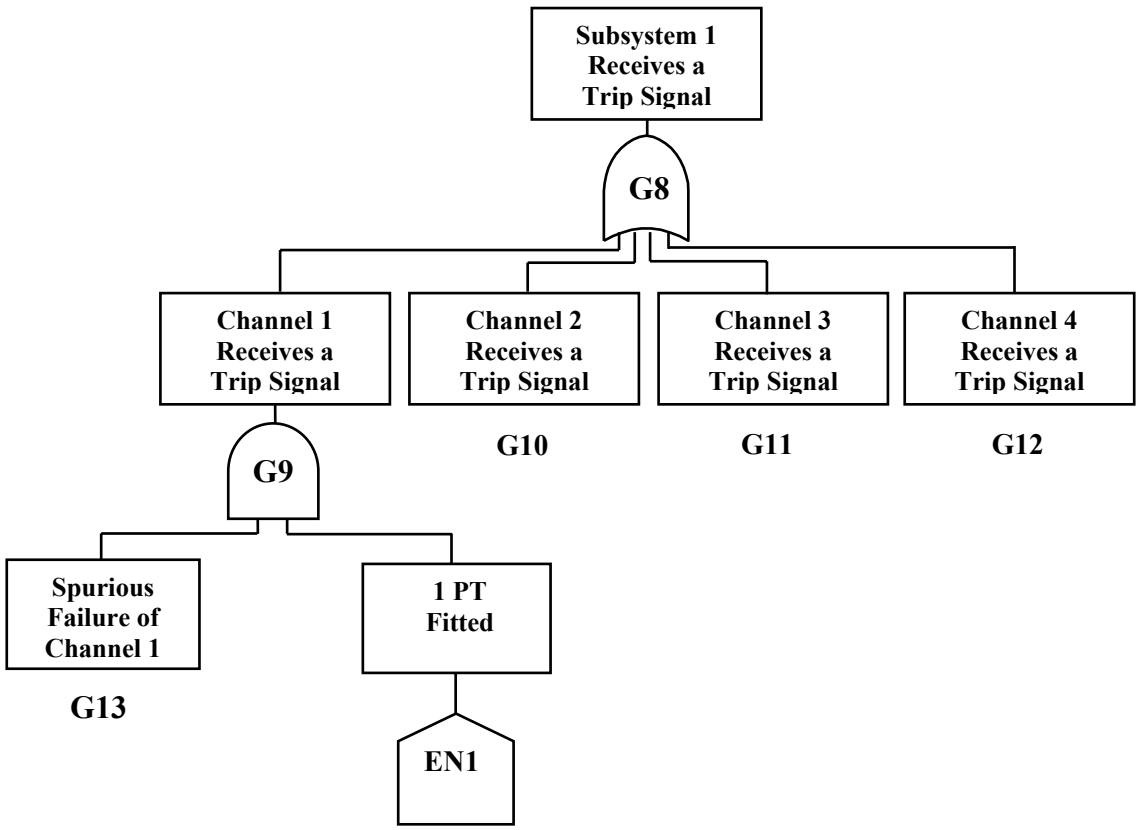


Figure A.11 Structure of the Branch 'Subsystem 1 Receives a Trip Signal'

Further structural differences can be found within each channel. Consider channel 2, which can have 1 or two fitted pressure transmitters. Consequently, the number of required pressure transmitters K_1 in this case can be either 1 or 2. Spurious failure of each transmitter will initiate spurious system action, if only 1 pressure transmitter is required to trip the system. In contrast, if 2 pressure transmitters must register an increase in pressure, spurious failure of both is required to cause spurious system failure. The development of other channels is similar to those discussed before.

Sub-events 2 and 3. 'ESD and HIPS Subsystem Fail Spuriously'

The ESD subsystem will fail spuriously if any of the included ESD valves fail. The development of the resulting intermediate events such as 'ESD Valve 1 Closes Spuriously' or 'ESD Valve 2 Closes Spuriously' is similar to the structure in the system unavailability fault tree. Furthermore, the failure of the HIPS in its structure mimics the relative one from the unavailability fault tree as well.

APPENDIX B

B.1 The FDS Failure Events and Data

There are two main types of the FDS events. The type 'HE' states that the event is a human error. On the other hand, type 'CO' denotes that the event is a component failure. The 'wear-out' components are denoted by 'W'. In contrast, 'NW' states that the component is of 'non-wear-out' type. It is important to notice, that preventative maintenance is only carried out on components of wear-out type.

The system is checked for corrosion build-up. Consequently, the corrosion resistant components are introduced, where 'n' and 'o' correspond to the non-corrosion resistant and corrosion resistant materials respectively. The notation key to the FDS data is expressed in table B.1.

Table B.1 The FDS Data Notation

Notation	Description
λ_D	Dormant failure rate
λ_S	Spurious failure rate
τ_D	Dormant mean time to repair
τ_S	Spurious mean time to repair
H_T	Number of hours manual work required to test the component
C_{HT}	Cost per hour of manual work to test the component
C_R	Number of hours manual work required to repair the component
C_{HR}	Cost per hour of manual work to repair failure (dormant or spurious)
C_{SR}	Cost of spares for each repair carried out (dormant or spurious)
H_P	Number of hours manual work required to carry out preventative maintenance
C_{SP}	Cost of spares each time preventative maintenance is undertaken
C_{HP}	Cost per hour of manual work to carry out preventative maintenance
N_S	Number of spares stored
C_S	Storage cost per component
C_I	Initial cost

Failure Events and Data of the Deluge System. Events considered in the reliability assessment of the deluge skid are specified in table B.2.

Table B.2 Failure Events of the Deluge Skid

Event Name	Event Description	Event Type	Rates
SI	Failure of MFGP to correctly select and send a close signal to the solenoid valve.	CO	NW
WBS	Strainer, located upstream of the water deluge valve, blocked.	CO	NW
WBN (old)	Deluge nozzle on the waterspray system blocked, old type material.	CO	NW
WBN (new)	Deluge nozzle on the waterspray system blocked, new type material.	CO	NW
WIVB	Blockage of the locked open butterfly valve, one upstream and one downstream of the water deluge valve.	CO	NW
WIVO	Operator leaves the normally locked open butterfly valve in the shut position (one upstream and one downstream of the water deluge valve).	HE	-
WV1	Water deluge valve type 1 fails to open	CO	NW
WV2	Water deluge valve type 2 fails to open	CO	NW
WV3	Water deluge valve type 3 fails to open	CO	NW
MRM	Manual release mechanism fails to dump instrument air.	CO	NW
SV1 SV2	Solenoid activated valve fails to dump instrument air on receipt of the signal form the MFGP.	CO	NW
WVR (old)	Valmatic relief valve sticks closed on activation, old type material.	CO	NW
WVR (new)	Valmatic relief valve sticks closed on activation, new type material.	CO	NW
AIVB	Normally locked open butterfly valve on AFFF distribution line blocked (only one isolation valve on the AFFF line).	CO	NW
AIVC	Operator leaves the normally locked open butterfly valve on the AFFF distribution line in the shut position.	HE	-
AINB (old)	The foam supply into the firewater distribution line is blocked by inductor nozzle, old type material.	CO	NW
AINB (new)	The foam supply into the firewater distribution line is blocked by inductor nozzle, new type material.	CO	NW
ACVB	The check valve in the AFFF injection line is blocked	CO	NW
AV1	AFFF deluge valve type 1 fails to open on demand	CO	NW
AV2	AFFF deluge valve type 2 fails to open on demand	CO	NW
AV3	AFFF deluge valve type 3 fails to open on demand	CO	NW

Table B.3 provides the failure and repair data, maintenance effort and costs associated with each component. It can be noticed from this table, that human error events only require specification of the probability of occurrence. In addition, SV1, SV2 and WVRF are the only components that fail spuriously.

Table B.3 Data Associated with Events from Table B.2

Event	λ_D	τ_D	λ_S	τ_S	H_T	C_{HT}	C_R	C_{HR}	C_{SR}	N_S	C_S	C_I
SI	2e-7	6e-6	-	-	-	-	-	-	-	-	-	-
WBS	2.8e-5	1.2e-5	-	-	2	30	12	30	50	4	75	100
WBN (o)	3e-5	1.2e-5	-	-	2	30	12	30	100	3	300	1000
WBN (n)	5e-6	1.2e-5	-	-	2	30	12	30	300	3	300	3000
WIVB	1.8e-6	1.8e-6	-	-	2	30	18	30	200	2	300	400
WIVO	$Q = 0.01$											
WV1	4.0e-5	1.8e-5	-	-	2	30	18	30	200	2	200	400
WV2	3.5e-5	1.8e-5	-	-	2	30	18	30	250	2	200	500
WV3	2.8e-5	1.8e-5	-	-	2	30	18	30	300	2	200	600
MRM	1e-5	1.2e-5	-	-	2	30	12	30	300	1	300	600
SV1	3e-6	1.2e-5	3e-6	1.2e-5	2	30	12	30	200	2	300	400
SV2	2e-5	1.2e-5	2e-5	1.2e-5	2	30	12	30	125	2	300	250
WVR (o)	5e-6	1.2e-5	5e-6	1.2e-5	2	30	12	30	300	1	300	600
WVR (n)	2e-6	1.2e-5	2e-6	1.2e-5	2	30	12	30	450	1	300	900
AIVB	1.8e-5	1.8e-5	-	-	2	30	18	30	200	2	300	400
AIVC	$Q = 0.01$											
AINB (o)	3e-5	1.2e-5	-	-	2	30	12	30	100	3	300	1000
AINB (n)	5e-6	1.2e-5	-	-	2	30	12	30	300	3	300	3000
ACVB	2.5e-5	1.8e-5	-	-	2	30	18	30	300	2	300	600
AV1	4e-5	1.8e-5	-	-	2	30	18	30	150	2	150	300
AV2	3.5e-5	1.8e-5	-	-	2	30	18	30	150	2	150	300
AV3	2.8e-5	1.8e-5	-	-	2	30	18	30	250	2	150	500

Failure Events and Data of the Firewater Supply and Distribution System. Failure events considered in the reliability assessment of the firewater supply and distribution are specified in tables B.4, B.5 and B.6. Table B.4 provides a list of the events associated with each fire pump and, therefore, constitute a single pump line (Chapter 7, Figure 7.1).

Table B.4 Events of Each Firewater Pump

Event Name	Description	Event Type	Rates
FB	The pump, including seawater filter, is blocked by debris.	CO	NW
IVB	Firewater pump isolation valve being blocked. The butterfly isolation valve operates on the header from pump to ringmain.	CO	NW
IVC	The firewater pump isolation valve is left closed after pump test.	HE	-
PRVO	Pressure relief valve on header from pump to ringmain fails open.	CO	NW
SVO	The flow control valve fails open on demand (used to dump excess flow from the pump to ringmain).	CO	NW
DVO	Test line, used to dump flow from firewater pumps overboard during test, is left open after completion.	HE	-
CVB	Check valve on header between the pump and ringmain blocked.	CO	NW

It is important to notice, that the electricity supply (ESF) is global to all electric pumps. In addition, a single diesel tank supplies all fitted firewater diesel pumps. Table B.5 specifies the events associated with the electric and diesel supply and the different available pump types. Failure events associated with the distribution network are considered in table B.6.

Table B.5 Failure Events Associated with the Firewater Pumps

Event Name	Description	Event Type	Rates
ESF	Failure of electricity supply to electric driven firewater pumps.	CO	NW
DIVB	Diesel engine supply is blocked.	CO	NW
DIVC	Diesel supply is inadvertently left isolated after maintenance.	HE	-
LAF	Diesel tank level switch fails to signal flow level to control room.	CO	NW
OAF	Operator fails to notice or misinterprets the low level tank alarm.	HE	-
E_100	Failure of electric pump with 100% capacity.	CO	W
D_100	Failure of diesel pump with 100% capacity.	CO	W
E1_50	Failure of electric pump type 1 with 50% capacity.	CO	W
E2_50	Failure of electric pump type 2 with 50% capacity.	CO	W
D1_50	Failure of diesel pump type 1 with 50% capacity	CO	W
D2_50	Failure of diesel pump type 2 with 50% capacity.	CO	W
E1_33	Failure of the electric pump type 1 with 33.33% capacity.	CO	W
E2_33	Failure of the electric pump type 2 with 33.33% capacity.	CO	W
D1_33	Failure of the diesel pump type 1 with 33.33% capacity.	CO	W
D2_33	Failure of the diesel pump type 2 with 33.33% capacity.	CO	W
E/DM	Probability of maintenance being carried out on a pump.	-	-

Table B.6 Failure Events Associated with the Distribution Network

Event Name	Description	Event Type	Rates
FSU	Failure of fire pump selector unit to initiate start of the standby pump in sequence, on detection of failure of duty pump/pumps to restore ringmain pressure.	CO	NW
OE	Designated duty pump/pumps inadvertently left in a mode other than auto start at the end of the test.	HE	-
PBF	Manual push button in the control room failing to initiate pump start when pressed.	CO	NW
PT1	Failure of ringmain low pressure sensor type 1 to indicate low ringmain pressure.	CO	NW
PT2	Failure of ringmain low pressure sensor type 2 to indicate low ringmain pressure.	CO	NW
PT3	Failure of ringmain low pressure sensor type 3 to indicate low ringmain pressure.	CO	NW

Tables B.7, B.8 and B.9 give the associated component data for the events specified in tables B.4, B.5 and B.6 respectively. The notation key for these tables is supplied in table B.1.

Table B.7 Data Associated with the Events of Each Firewater Pump

Event	λ_D	τ_D	H_T	C_{HT}	C_R	C_{HR}	C_{SR}	N_S	C_S	C_I
FB	2.8e-5	1.2e-5	2	30	12	30	50	4	150	100
IVB	1.8e-5	1.8e-5	2	30	18	30	400	2	300	400
IVC	Q = 0.01									
PRVO	1.2e-5	1.8e-5	2	30	18	30	250	2	300	500
SVO	1.8e-5	2.4e-5	2	30	24	30	400	2	300	800
DVO	Q = 0.01									
CVB	2.5e-5	1.8e-5	2	30	18	30	300	2	300	500

The pumps are of wear-out type, therefore, the Weibull distribution is used. This distribution is chosen because in contrast to the exponential distribution Weibull is able to model increasing and decreasing failure rates. Thus, lending itself to the first (wear-in) and last (wear-out) phases of the bathtub curve in addition to the useful life period [Pattison, 1999]. It is characterised by a hazard rate function of the form given in equation B.1,

$$\lambda(t) = \frac{\beta}{\eta}\left(\frac{t}{\eta}\right)^{\beta-1}, \ \eta > 0, \ \beta > 0, \ t \geq 0, \tag{B.1}$$

where β is referred to as the shape parameter and η is the scale parameter (or characteristic life), which influences both the mean and spread of the distribution. Modifying the value of β has a dramatic effect on the probability density function $f(t)$ (Equation B.2):

$$f(t) = \lambda(t)\exp\left(-\left(\frac{t}{\eta}\right)^{\beta}\right). \tag{B.2}$$

When $\beta < 1$, the hazard rate applies to the burn-in phase. On the other hand, for $\beta = 1$ the hazard rate is constant and the distribution is identical to the exponential. When $\beta > 1$, the hazard rate applies to the wear-out phase. For $\beta \geq 3$, the probability density function tends toward a normal distribution, thus portraying symmetry. The value of the Weibull parameters (β and η) are specified in table B.8. In addition, PBF is a component failure with its probability of occurrence specified directly.

Table B.8 Failure Events Data for the Firewater Pumps

Event	λ_D	τ_D	H_T	C_{HT}	C_R	C_{HR}	C_{SR}	H_P	C_{HP}	C_{SP}	N_S	C_S	C_I
ESF	5e-5	2e-6	2	45	2	45	200	-	-	-	-	-	1000
DIVB	3e-5	8e-6	2	30	8	30	200	-	-	-	2	300	400
DIVC	Q = 0.01												
LAF	3e-5	6e-6	2	30	6	30	100	-	-	-	2	200	200
OAF	Q = 0.01												
Event	β	η	H_T	C_{HT}	C_R	C_{HR}	C_{SR}	H_P	C_{HP}	C_{SP}	N_S	C_S	C_I
E_100	2	16667	2	30	72	30	1500	72	30	300	1	1000	3000
D_100	2	14035	2	30	72	30	1450	72	30	290	1	1000	2900
E1_50	3/2	22857	2	30	48	30	900	48	30	180	2	900	1800
E2_50	3/2	26667	2	30	48	30	1000	48	30	200	2	900	2000
D1_50	3/2	20000	2	30	48	30	750	48	30	150	2	900	1500
D2_50	3/2	22857	2	30	48	30	900	48	30	180	2	900	1800
E1_33	3/2	32000	2	30	36	30	600	48	30	120	2	800	1200
E2_33	3/2	40000	2	30	36	30	700	48	30	140	2	800	1400
D1_33	3/2	28571	2	30	48	30	500	48	30	100	2	800	1000
D2_33	3/2	33333	2	30	48	30	550	48	30	110	2	800	1100
E/DM	Q = 0.04												

Table B.9 Failure Events Data for the Distribution Network

Event	λ_D	τ_D	λ_S	τ_S	H_T	C_{HT}	C_R	C_{HR}	C_{SR}	N_S	C_S	C_I
FSU	8e-6	2.4e-5			1	45	24	45	200	1	200	2000
OE	$Q = 0.01$											
PBF	$Q = 0.01$											
PT1	7e-6	4e-6	7e-6	4e-6	1	45	4	45	50	2	100	500
PT2	1.4e-5	4e-6	1.4e-5	4e-6	1	45	4	45	20	2	100	200
PT3	2.1e-5	4e-6	2.1e-5	4e-6	1	45	4	45	10	2	100	100

Failure Events and Data of the AFFF Supply and Distribution System. Failure events of each AFFF pump line are identical to those described in table B.3, where the associated component data is supplied in table B.6. Table B.9 provides further events involved in the AFFF supply and distribution system.

Table B.10 Failure Events Associated with AFFF Supply and Distribution

Event Name	Description	Event Type	Rates
ATIVB	Normally locked open ball valve on AFFF tank outlet blocked.	CO	NW
ATIVC	AFFF supply left isolated after maintenance.	HE	-
AESF	Failure of electric supply to the electric driven AFFF pumps.	CO	NW
ADIVB	Normally locked open ball valve on diesel tank outlet blocked.	CO	NW
ADIVC	Diesel supply to AFFF pumps left isolated after maintenance.	CO	NW
ALAF	Diesel tank level switch fails to signal low level to control room.	CO	NW
AOAF	Operator fails to notice AFFF diesel tank low level alarm.	HE	-
AE_100	Failure of AFFF electric pump with 100% capacity.	CO	W
AD_100	Failure of AFFF diesel pump with 100% capacity.	CO	W
AE_50	Failure of AFFF electric pump with 50% capacity.	CO	W
AD_50	Failure of AFFF diesel pump with 50% capacity.	CO	W
AE/DM	Probability of maintenance being carried out on an AFFF pump.	CO	W

Table B.11 describes the data associated with each of the events from table B.10. Similar to the firewater pumps, the AFFF pumps are of wear-out type. Hence, the Weibull distribution is used and Weibull parameters (β and η) are specified.

Table B.11 Failure Event Data for the AFFF Supply and Distribution System

Event	λ_D	τ_D	H_T	C_{HT}	C_R	C_{HR}	C_{SR}	H_P	C_{HP}	C_{SP}	N_S	C_S	C_I
ADIVC	Q = 0.01												
ATIVC	Q = 0.01												
ALAF	3e-5	6e-6	2	30	6	30	100	-	-	-	2	200	200
ATIVB	1.8e-5	1.8e-5	2	30	18	30	200	-	-	-	2	300	400
AESF	5e-5	2e-6	2	45	2	45	200	-	-	-	-	-	1000
ADIVB	3e-5	8e-6	2	30	8	30	200	-	-	-	2	300	400
AOAF	Q = 0.01												
Event	β	η	H_T	C_{HT}	C_R	C_{HR}	C_{SR}	H_P	C_{HP}	C_{SP}	N_S	C_S	C_I
AE_100	2	16667	2	30	72	30	750	72	30	150	1	800	1500
AD_100	2	14035	2	30	72	30	725	72	30	145	1	800	1450
AE_50	3/2	22857	2	30	48	30	450	48	30	90	2	600	900
AD_50	3/2	20000	2	30	48	30	375	48	30	75	2	600	750
AE/DM	Q = 0.04												

B.2 FDS Fault Tree Construction

B.2.1 FDS Unavailability Fault Tree Construction

The top event 'Firewater Deluge System Fails to Protect' represents the causes of the firewater deluge system unavailability. There are three main reasons for the top event to occur. The first reason, i.e. failure to initiate the firewater and AFFF pump mechanisms, occurs if both automatic and manual interventions fail. The manual start of the system fails if either the push button on the MFGP fails or if the operator fails to push the button. An automatic start fails if either the fire pump selector unit fails or the low pressure sensing on the firewater ringmain fails. Failure of the low pressure sensing depends on the number of pressure transmitters fitted (N) and the number of pressure transmitters required to trip the system (K). House events are included in the fault tree to model each possible design alternative in a similar manner to the high integrity pressure system (Appendix A). Figure B.1 represents the initial steps of the FDS system unavailability fault tree construction.

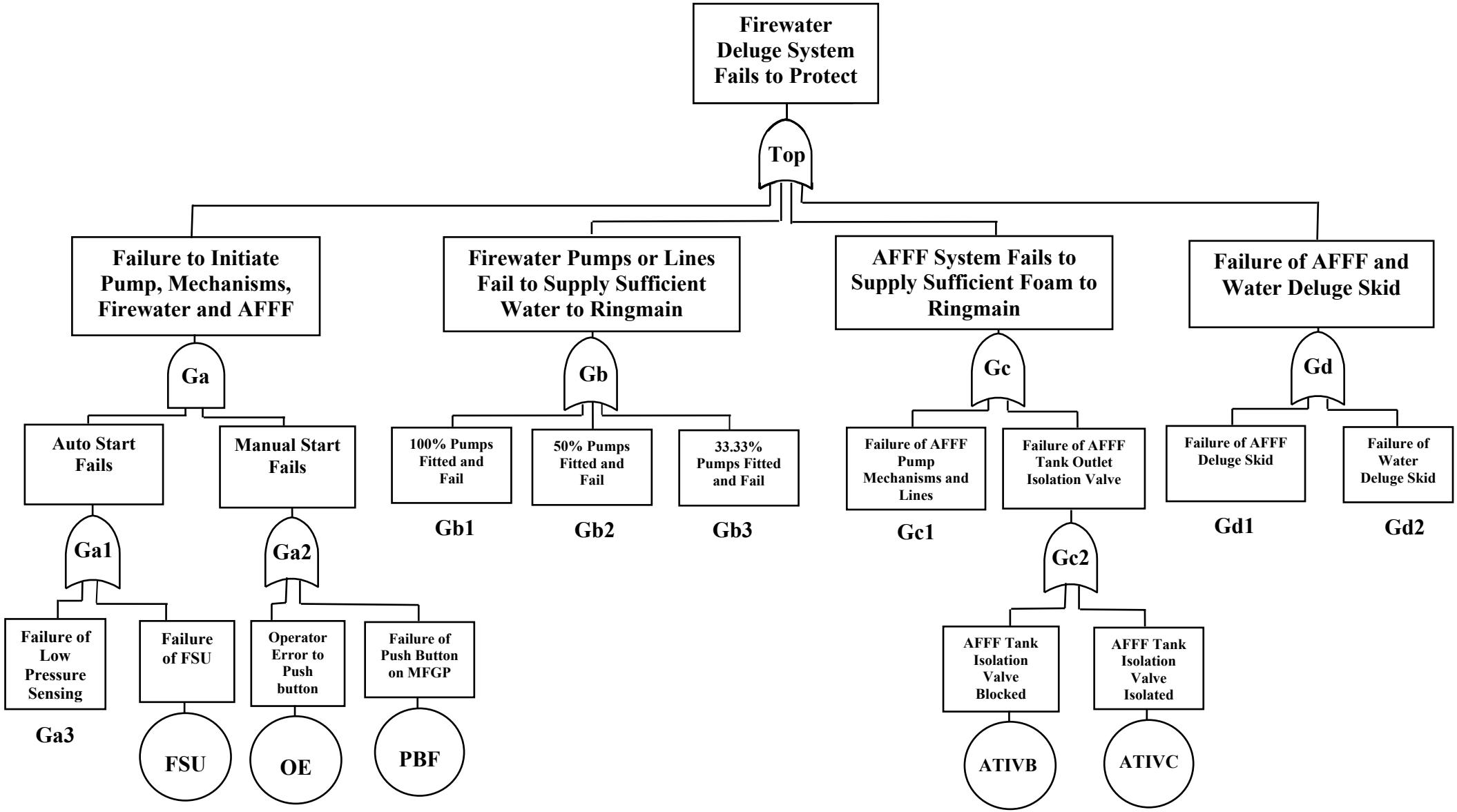


Figure B.1 Firewater Deluge System Fails to Protect

Failure of the firewater pumps and lines means that the FDS system fails to supply sufficient water to the ringmain due to the failure of the firewater pump mechanisms or lines to supply 100% pressure. As it is seen from figure B.1, the event 'Failure of the firewater pumps and lines' will occur if either the firewater pumps are of 100% capacity and fail, if the firewater pumps are of 50% capacity and fail or if the firewater pumps are of 33.33% capacity and fail.

The system may have 1 to 8 firewater pumps of 100% capacity. Therefore, the gate 'Gb1' has 8 sub-events, where the left most branch considers that 'One 100% pump only is fitted and fails', the next that 'Two 100% pumps are fitted and fail', and so on. Each of these 8 sub-events is further developed. For example, consider the event "Two 100% pumps are fitted and fail". Two fitted pumps give three possible combinations of the electric and diesel pumps. Each of these combinations is further developed to specify the events that will lead to the future system failure. Figure B.2 represents this casual relationship.

The house event H2P (Figure B.2) is set to true if two pumps are fitted. The house events HE0, HE1 and HE2 indicate the number of electric pumps fitted, i.e. HE0 is true if there are no electric pumps, HE1 and HE2 are true if 1 and 2 electric pumps are fitted respectively.

Pump failure occurs if the pump itself fails or if components of the pump line fail. Therefore, the gate 'Gb126', i.e. 'Electric Pump No. 1 Fails', can be depicted as shown in figure B.3. 'E1', appended to the failure events, specifies that the component occurs in electric pump line number 1 alone.

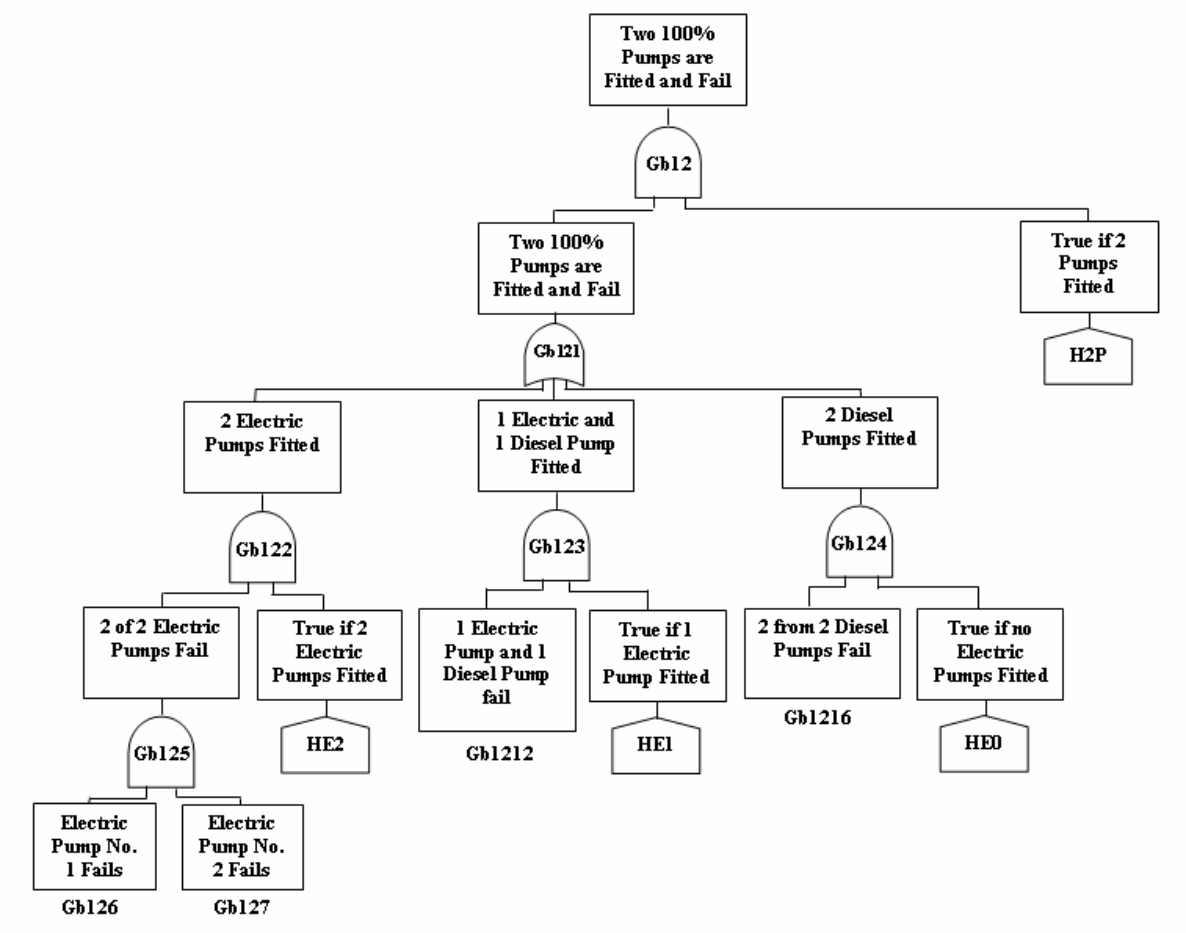


Figure B.2 Development of the Branch 'Two 100% Pumps are Fitted'

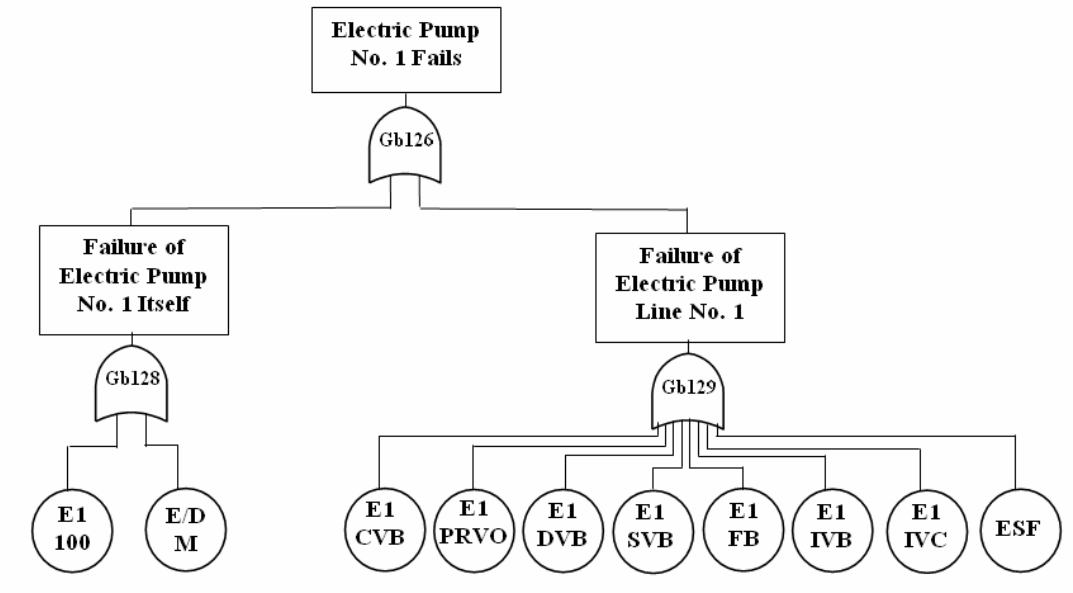


Figure B.3 Development of the Branch 'Electric Pump No. 1 Fails'

The event '50% Pumps are Fitted and Fail', i.e. gate 'Gb2', is developed in a similar manner to the gate 'Gb1'. The main difference is that for the gate 'Gb2' at least 2 pumps must be fitted to ensure 100% pressure attained. Therefore, the sub-event 'One 50% pump is fitted and fails' is

infeasible. The 50% pumps branch is a little more complex than 100% due to the complexity of the combinations of pumps resulting in overall pump failure. For example, consider the next two levels in the tree structure below the event 'Three 50% Pumps Fail' (Figure B.4).

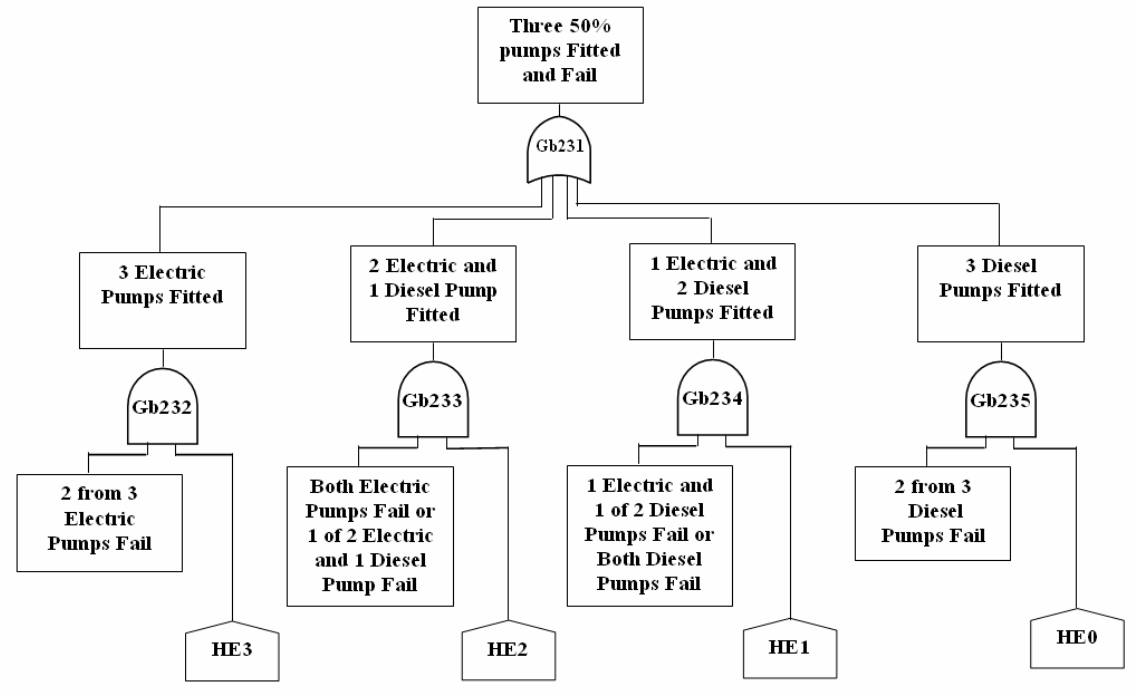


Figure B.4 Development of the Branch 'Three 50% Pumps Fail'

The development of the event '33.33% pumps are fitted and fail' is still more complex. However, the construction process follows a similar pattern. The 33.33% pumps branch requires at least 3 pumps to be fitted in the system to ensure 100% pressure, thus, only 6 sub-events constitute the level immediately below the gate 'Gb3'.

Consider now the AFFF pumps and lines. Failure of the AFFF pump system fails to supply sufficient foam to the ringmain as a result of failure to the AFFF pump mechanisms or lines or isolation of the AFFF tank (Chapter 7, Figure 7.1). Development of the gate 'Gc1' is similar to that of the firewater pump system. However, the pumps of 33.33% capacity are not considered for this part of the FDS system.

Failure of the AFFF or water deluge skid occurs if either events 'Failure of the water deluge skid' or 'Failure of the AFFF Deluge Skid' occur. The possible reasons for the event 'Failure of the water deluge skid' to occur are: the water spray isolation valves fail, the strainer nozzle becomes blocked or the deluge valve fails to open. Further development of the event 'The water

deluge valve fails to open' involves two scenarios connected by OR logic, i.e. events that restrict activation of the deluge valve and failure of the deluge valve itself. 'Failure to activate the water deluge valve' can be caused by the failure of the signal to the solenoids, by the solenoid valves remaining energized or by the failure of the valmatic release valve. In a similar manner the event 'Failure of the AFFF deluge skid' is developed. The main difference is that the blocked nozzle is replaced by blockage of the inductor nozzle and the strainer by a blocked AFFF check valve in the sequence of previously discussed events. The full version of the FDS Unavailability fault tree is described in appendix.

B.2.2 FDS Spurious Trip Fault Tree Construction

According to the FDS system limitations a number of spurious system occurrences is permitted, i.e. $F_{sys} < 0.75$ (Table 7.2). Hence, the spurious activation of the FDS must be established by developing the specific fault tree to quantify causes of this failure mode.

The top event 'Firewater deluge system fails spuriously' occurs if the solenoid valves fail spuriously, the valmatic release valve opens spuriously or the signal from the main fire and gas panel (MFGP) to the solenoid valves is interrupted. The latter event occurs as a result of spurious activation of the ringmain pressure sensors. Figure B.5 shows the casual relationship of the events directly causing the top event. The full fault tree structure is represented in appendix D.

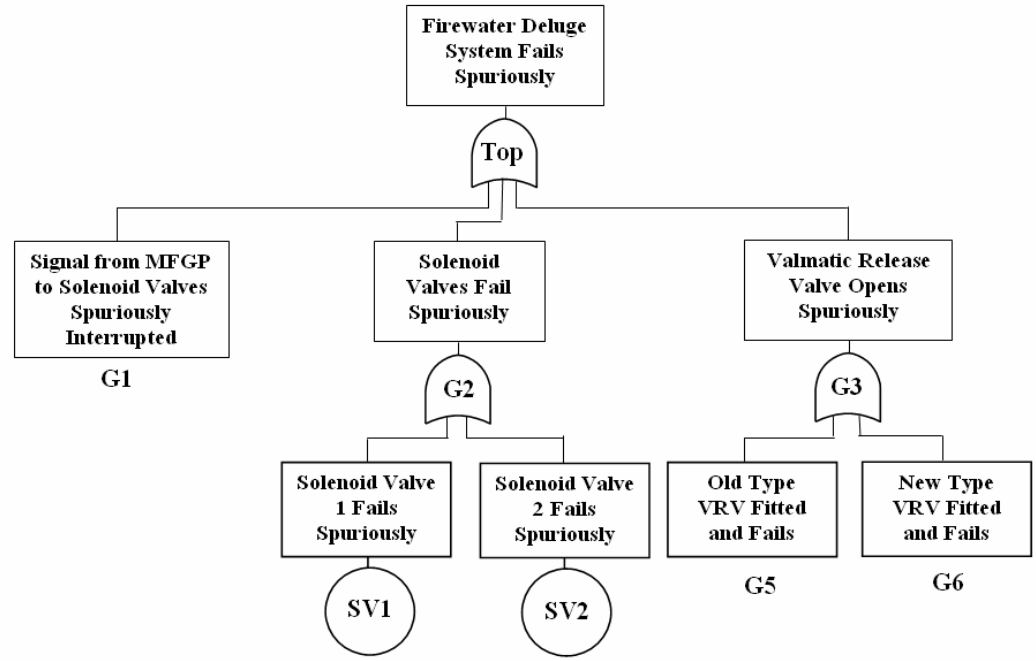


Figure B.5 Development of the Spurious Trip Fault Tree

APPENDIX C

Table C.1 HIPS Unavailability Fault Tree Structure

Gate No	Gate Name	Gate Type	Input Gate Number	Input Event Number	House Event Number	Input Gate List	Input Event List
1	top	and	3	0	0	g1, g2, g3	
2	g1	and	2	0	0	g1a, g1b	
3	g1a	or	1	1	0	g4	Wv
4	g1b	or	1	1	0	g4a	Mv
5	g2	and	2	0	0	g61, g62	
6	g3	and	2	0	0	g90, g91	
7	g4	or	1	1	0	g5	Svw
8	g4a	or	1	1	0	g5	Svm
9	g5	or	2	0	0	g6, g7	
10	g6	and	0	2	0		r1/1, r1/2
11	g7	or	1	1	0	g8	plc1
12	g8	and	4	0	0	g9, g10, g11, g12	
13	g9	or	1	1	1	g13	nen1
14	g10	or	1	1	1	g18	nen2
15	g11	or	1	1	1	g27	nen3
16	g12	or	1	1	1	g41	nen4
17	g13	and	1	1	1	g14	en1
18	g14	and	1	1	1	g15	ek1
19	g15	or	2	0	0	g16, g17	
20	g16	and	0	2	1		p11, pt11
21	g17	and	0	2	1		p12, pt21
22	g18	and	1	1	1	g19	en2
23	g19	or	2	0	0	g20, g21	
24	g20	and	1	1	1	g22	ek1
25	g21	and	1	1	1	g26	ek2
26	g22	and	2	0	0	g15, g23	
27	g23	or	2	0	0	g24, g25	
28	g24	and	0	2	1		p11, pt12
29	g25	and	0	2	1		p12, pt22
30	g26	or	2	0	0	g15, g23	
31	g27	and	1	1	1	g28	en3
32	g28	or	3	0	0	g29, g30, g31	
33	g29	and	1	1	1	g32	ek1
34	g30	and	1	1	1	g36	ek2
35	g31	and	1	1	1	g40	ek3
36	g32	and	3	0	0	g15, g23, g33	
37	g33	or	2	0	0	g34, g35	
38	g34	and	0	2	1		p11, pt13
39	g35	and	0	2	1		p12, pt23
40	g36	or	3	0	0	g37, g38, g39	
41	g37	and	2	0	0	g15, g23	
42	g38	and	2	0	0	g15, g33	
43	g39	and	2	0	0	g23, g33	
44	g40	or	3	0	0	g15, g23, g33	
45	g41	and	1	1	1	g42	en4
46	g42	or	4	0	0	g43, g44, g45, g46	

Gate No	Gate Name	Gate Type	Input Gate Number	Input Event Number	House Event Number	Input Gate List	Input Event List
47	g43	and	1	1	1	g47	ek1
48	g44	and	1	1	1	g52	ek2
49	g45	and	1	1	1	g58	ek3
50	g46	and	1	1	1	g59	ek4
51	g47	and	4	0	0	g15, g23, g33, g48	
52	g48	or	2	0	0	g49, g50	
53	g49	and	0	2	1		p11, pt14
54	g50	and	0	2	1		p12, pt24
55	g52	or	4	0	0	g53, g54, g56, g57	
56	g53	and	3	0	0	g15, g23, g33	
57	g54	and	3	0	0	g15, g23, g48	
58	g56	and	3	0	0	g23, g33, g48	
59	g57	and	3	0	0	g15, g33, g48	
60	g58	or	6	0	0	g58a, g58b, g58c, g58d, g58e, g58f	
61	g58a	and	2	0	0	g15, g23	
62	g58b	and	2	0	0	g15, g33	
63	g58c	and	2	0	0	g15, g48	
64	g58d	and	2	0	0	g23, g33	
65	g58e	and	2	0	0	g23, g48	
66	g58f	and	2	0	0	g33, g48	
67	g59	or	4	0	0	g15, g23, g33, g48	
68	g61	or	1	1	1	g63	ne1
69	g62	or	1	1	1	g69	ne2
70	g63	and	1	1	1	g64	e1
71	g64	or	2	0	0	g65, g66	
72	g65	or	2	0	0	g67, g68	
73	g66	or	1	1	0	g5	sve1
74	g67	and	0	2	1	v1	esd11
75	g68	and	0	2	1	v2	esd21
76	g69	and	1	1	1	g70	e2
77	g70	or	2	0	0	g71, g72	
78	g71	or	2	0	0	g73, g74	
79	g72	or	1	1	0	g5	sve2
80	g73	and	0	2	1		v1, esd12
81	g74	and	0	2	1		v2, esd22
82	g90	or	1	1	1	g92	nh1
83	g91	or	1	1	1	g102	nh2
84	g92	and	1	1	1	g93	h1
85	g93	or	2	0	0	g94, g95	
86	g94	or	2	0	0	g96, g97	
87	g95	or	1	1	0	g98	svh1
88	g96	and	0	2	1		hv1, hips11
89	g97	and	0	2	1		hv2, hips21
90	g98	or	2	0	0	g99, g100	
91	g99	or	1	1	0	g101	plc2
92	g100	and	0	2	0		r2/1, r2/2
93	g101	and	4	0	0	g301, g302, g303, g304	
94	g102	and	1	1	1	g103	h2

Gate No	Gate Name	Gate Type	Input Gate Number	Input Event Number	House Event Number	Input Gate List	Input Event List
95	g103	or	2	0	0	g104, g105	
96	g104	or	2	0	0	g106, g107	
97	g105	or	1	1	0	g98	svh2
98	g106	and	0	2	1		hv1, hips12
99	g107	and	0	2	1		hv2, hips22
100	g114	or	6	0	0	g348, g349, g350, g351, g352, g353	
101	g301	or	1	1	1	g305	nhn1
102	g302	or	1	1	1	g310	nhn2
103	g303	or	1	1	1	g319	nhn3
104	g304	or	1	1	1	g333	nhn4
105	g305	and	1	1	1	g306	hn1
106	g306	and	1	1	1	g307	hk1
107	g307	or	2	0	0	g308, g309	
108	g308	and	0	2	1		p21, pt15
109	g309	and	0	2	1		p22, pt25
110	g310	and	1	1	1	g311	hn2
111	g311	or	2	0	0	g312, g317	
112	g312	and	1	1	1	g313	hk1
113	g313	and	2	0	0	g307, g314	
114	g314	or	2	0	0	g315, g316	
115	g315	and	0	2	1		p21, pt16
116	g316	and	0	2	1		p22, pt26
117	g317	and	1	1	1	g318	hk2
118	g318	or	2	0	0	g307, g314	
119	g319	and	1	1	1	g320	hn3
120	g320	or	3	0	0	g321, g322, g323	
121	g321	and	1	1	1	g324	hk1
122	g322	and	1	1	1	g328	hk2
123	g323	and	1	1	1	g332	hk3
124	g324	and	3	0	0	g307, g314, g325	
125	g325	or	2	0	0	g326, g327	
126	g326	and	0	2	1		p21, pt17
127	g327	and	0	2	1		p22, pt27
128	g328	or	3	0	0	g329, g330, g331	
129	g329	and	2	0	0	g307, g314	
130	g330	and	2	0	0	g307, g325	
131	g331	and	2	0	0	g314, g325	
132	g332	or	3	0	0	g307, g314, g325	
133	g333	and	1	1	1	g334	hn4
134	g334	or	4	0	0	g335, g336, g337, g338	
135	g335	and	1	1	1	g339	hk1
136	g336	and	1	1	1	g340	hk2
137	g337	and	1	1	1	g114	hk3
138	g338	and	1	1	1	g354	hk4
139	g339	and	4	0	0	g307, g314, g325, g345	
140	g340	or	4	0	0	g341, g342, g343, g344	
141	g341	and	3	0	0	g307, g314, g325	

Gate No	Gate Name	Gate Type	Input Gate Number	Input Event Number	House Event Number	Input Gate List	Input Event List
142	g342	and	3	0	0	g307, g314, g345	
143	g343	and	3	0	0	g307, g325, g345	
144	g344	and	3	0	0	g314, g325, g345	
145	g345	or	2	0	0	g346, g347	
146	g346	and	0	2	1		p21, pt18
147	g347	and	0	2	1		p22, pt28
148	g348	and	2	0	0	g307, g314	
149	g349	and	2	0	0	g307, g325	
150	g350	and	2	0	0	g307, g345	
151	g351	and	2	0	0	g314, g325	
152	g352	and	2	0	0	g314, g345	
153	g353	and	2	0	0	g325, g345	
154	g354	or	4	0	0	g307, g314, g325, g345	

Table C.2 HIPS Spurious Trip Fault Tree Structure

Gate No	Gate Name	Gate Type	Input Gate Number	Input Event Number	House Event Number	Input Gate List	Input Event List
1	top	or	3	0	0	g1, g2, g3	
2	g1	or	2	0	0	g1a, g1b	
3	g1a	or	1	1	0	g4	wvs
4	g1b	or	1	1	0	g4b	mvs
5	g2	or	2	0	0	g60, g61	
6	g3	or	2	0	0	g80, g81	
7	g4	or	1	1	0	g5	ssvw
8	g4b	or	1	1	0	g5	ssvm
9	g5	or	2	0	0	g6, g7	
10	g6	or	0	2	0		sr1/1, sr1/2
11	g7	or	1	1	0	g8	splc1
12	g8	or	4	0	0	g9, g10, g11, g12	
13	g9	and	1	1	1	g13	en1
14	g10	and	1	1	1	g17	en2
15	g11	and	1	1	1	g25	en3
16	g12	and	1	1	1	g38	en4
17	g13	and	1	1	1	g14	ek1
18	g14	or	2	0	0	g15, g16	
19	g15	and	0	2	1		p11, spt11
20	g16	and	0	2	1		p12, spt21
21	g17	or	2	0	0	g18, g19	
22	g18	and	1	1	1	g20	ek1
23	g19	and	1	1	1	g24	ek2
24	g20	or	2	0	0	g14, g21	
25	g21	or	2	0	0	g22, g23	
26	g22	and	0	2	1		p11, spt12

Gate No	Gate Name	Gate Type	Input Gate Number	Input Event Number	House Event Number	Input Gate List	Input Event List
27	g23	and	0	2	1		p12, spt22
28	g24	and	2	0	0	g14, g21	
29	g25	or	3	0	0	g26, g27, g28	
30	g26	and	1	1	1	g29	ek1
31	g27	and	1	1	1	g33	ek2
32	g28	and	1	1	1	g37	ek3
33	g29	or	3	0	0	g14, g21, g30	
34	g30	or	2	0	0	g31, g32	
35	g31	and	0	2	1		p11, spt13
36	g32	and	0	2	1		p12, spt23
37	g33	or	3	0	0	g34, g35, g36	
38	g34	and	2	0	0	g14, g21	
39	g35	and	2	0	0	g14, g30	
40	g36	and	2	0	0	g21, g30	
41	g37	and	3	0	0	g14, g21, g30	
42	g38	or	4	0	0	g39, g40, g41, g42	
43	g39	and	1	1	1	g43	ek1
44	g40	and	1	1	1	g47	ek2
45	g41	and	1	1	1	g54	ek3
46	g42	and	1	1	1	g59	ek4
47	g43	or	4	0	0	g14, g21, g30, g44	
48	g44	or	2	0	0	g45, g46	
49	g45	and	0	2	1		p11, spt14
50	g46	and	0	2	1		p12, spt24
51	g47	or	6	0	0	g48, g49, g50, g51, g52, g53	
52	g48	and	2	0	0	g14, g21	
53	g49	and	2	0	0	g14, g30	
54	g50	and	2	0	0	g14, g44	
55	g51	and	2	0	0	g21, g30	
56	g52	and	2	0	0	g21, g44	
57	g53	and	2	0	0	g30, g44	
58	g54	or	4	0	0	g55, g56, g57, g58	
59	g55	and	3	0	0	g14, g21, g30	
60	g56	and	3	0	0	g14, g21, g44	
61	g57	and	3	0	0	g14, g30, g44	
62	g58	and	3	0	0	g21, g30, g44	
63	g59	and	4	0	0	g14, g21, g30, g44	
64	g60	and	1	1	1	g62	e1
65	g61	and	1	1	1	g67	e2
66	g62	or	2	0	0	g63, g64	
67	g63	or	2	0	0	g65, g66	
68	g64	or	1	1	0	g5	ssve1
69	g65	and	0	2	1		v1, sesd11
70	g66	and	0	2	1		v2, sesd21
71	g67	or	2	0	0	g68, g69	
72	g68	or	2	0	0	g70, g71	
73	g69	or	1	1	0	g5	ssve2
74	g70	and	0	2	1		v1, sesd12
75	g71	and	0	2	1		v2, sesd22

Gate No	Gate Name	Gate Type	Input Gate Number	Input Event Number	House Event Number	Input Gate List	Input Event List
76	g80	and	1	1	1	g82	h1
77	g81	and	1	1	1	g145	h2
78	g82	or	2	0	0	g83, g84	
79	g83	or	2	0	0	g86, g85	
80	g84	or	1	1	0	g87	ssvh1
81	g85	and	0	2	1		hv2, ships21
82	g86	and	0	2	1		hv1, ships11
83	g87	or	2	0	0	g88, g89	
84	g88	or	1	1	0	g90	splc2
85	g89	or	0	2	0		sr2/1, sr2/2
86	g90	or	4	0	0	g91, g92, g93, g94	
87	g91	and	1	1	1	g95	hn1
88	g92	and	1	1	1	g99	hn2
89	g93	and	1	1	1	g107	hn3
90	g94	and	1	1	1	g103a	hn4
91	g95	and	1	1	1	g96	hk1
92	g96	or	2	0	0	g97, g98	
93	g97	and	0	2	1		p21, spt15
94	g98	and	0	2	1		p22, spt25
95	g99	or	2	0	0	g100, g101	
96	g100	and	1	1	1	g102	hk1
97	g101	and	1	1	1	g106	hk2
98	g102	or	2	0	0	g96, g103	
99	g103	or	2	0	0	g104, g105	
100	g103a	or	4	0	0	g123, g124, g125, g126	
101	g104	and	0	2	1		p21, spt16
102	g105	and	0	2	1		p22, spt26
103	g106	and	2	0	0	g96, g103	
104	g107	or	3	0	0	g108, g109, g110	
105	g108	and	1	1	1	g111	hk1
106	g109	and	1	1	1	g118	hk2
107	g110	and	1	1	1	g122	hk3
108	g111	or	3	0	0	g96, g103, g112	
109	g112	or	2	0	0	g113, g114	
110	g113	and	0	2	1		p21, spt17
111	g114	and	0	2	1		p22, spt27
112	g118	or	3	0	0	g119, g120, g121	
113	g119	and	2	0	0	g96, g103	
114	g120	and	2	0	0	g96, g112	
115	g121	and	2	0	0	g103, g112	
116	g122	and	3	0	0	g96, g103, g112	
117	g123	and	1	1	1	g127	hk1
118	g124	and	1	1	1	g131	hk2
119	g125	and	1	1	1	g138	hk3
120	g126	and	1	1	1	g143	hk4
121	g127	or	4	0	0	g96, g103, g112, g128	
122	g128	or	2	0	0	g129, g130	
123	g129	and	0	2	1		p21, spt18

Gate No	Gate Name	Gate Type	Input Gate Number	Input Event Number	House Event Number	Input Gate List	Input Event List
124	g130	and	0	2	1		p22, spt28
125	g131	or	6	0	0	g132, g133, g134, g135, g136, g137	
126	g132	and	2	0	0	g96, g103	
127	g133	and	2	0	0	g96, g112	
128	g134	and	2	0	0	g96, g128	
129	g135	and	2	0	0	g103, g112	
130	g136	and	2	0	0	g103, g128	
131	g137	and	2	0	0	g112, g128	
132	g138	or	4	0	0	g139, g140, g141, g142	
133	g139	and	3	0	0	g96, g103, g112	
134	g140	and	3	0	0	g96, g103, g128	
135	g141	and	3	0	0	g96, g112, g128	
136	g142	and	3	0	0	g103, g112, g128	
137	g143	and	4	0	0	g96, g103, g112, g128	
138	g145	or	2	0	0	g146, g147	
139	g146	or	2	0	0	g148, g149	
140	g147	or	1	1	0	g87	ssvh2
141	g148	and	0	2	1		hv1, ships12
142	g149	and	0	2	1		hv2, ships22

APPENDIX D

Table D.1 FDS Unavailability Fault Tree Structure

Gate No	Gate Name	Gate Type	Input Gate No	Input Event No	House Event No	Input Gate List	Input Event List
1	top	or	4	0	0	ga, gb, gc, gd	
2	ga	and	2	0	0	ga1, ga2	
3	ga1	or	1	1	0	ga3	fsu
4	ga2	or	0	2	0		oe, pbf
5	ga3	or	4	0	0	gin6, gin7, gin8, gin9	
6	gin6	and	1	1	1	gin10	h1s
7	gin7	and	1	1	1	gin15	h2s
8	gin8	and	1	1	1	gin24	h3s
9	gin9	and	1	1	1	gin38	h4s
10	gin10	and	1	1	1	gin11	h1ts
11	gin11	or	3	0	0	gin12, gin13, gin14	
12	gin12	and	0	2	1		hpt1, pt11
13	gin13	and	0	2	1		hpt2, pt12
14	gin14	and	0	2	1		hpt3, pt13
15	gin15	or	2	0	0	gin16, gin17	
16	gin16	and	1	1	1	gin18	h1ts
17	gin17	and	1	1	1	gin23	h2ts
18	gin18	and	2	0	0	gin11, gin19	
19	gin19	or	3	0	0	gin20, gin21, gin22	
20	gin20	and	0	2	1		hpt1, pt21
21	gin21	and	0	2	1		hpt2, pt22
22	gin22	and	0	2	1		hpt3, pt23
23	gin23	or	2	0	0	gin11, gin19	
24	gin24	or	3	0	0	gin25, gin26, gin27	
25	gin25	and	1	1	1	gin28	h1ts
26	gin26	and	1	1	1	gin33	h2ts
27	gin27	and	1	1	1	gin37	h3ts
28	gin28	and	3	0	0	gin11, gin19, gin29	
29	gin29	or	3	0	0	gin30, gin31, gin32	
30	gin30	and	0	2	1		hpt1, pt31
31	gin31	and	0	2	1		hpt2, pt32
32	gin32	and	0	2	1		hpt3, pt33
33	gin33	or	3	0	0	gin34, gin35, gin36	
34	gin34	and	2	0	0	gin11, gin19	
35	gin35	and	2	0	0	gin11, gin29	
36	gin36	and	2	0	0	gin19, gin29	
37	gin37	or	3	0	0	gin11, gin19, gin29	
38	gin38	or	4	0	0	gin39, gin40, gin41, gin42	
39	gin39	and	1	1	1	gin43	h1ts
40	gin40	and	1	1	1	gin48	h3ts
41	gin41	and	1	1	1	gin54	h2ts
42	gin42	and	1	1	1	gin58	h4ts
43	gin43	and	4	0	0	gin11, gin19, gin29, gin44	
44	gin44	or	3	0	0	gin45, gin46, gin47	
45	gin45	and	0	2	1		hpt1, pt14
46	gin46	and	0	2	1		hpt2, pt24

Gate No	Gate Name	Gate Type	Input Gate No	Input Event No	House Event No	Input Gate List	Input Event List
47	gin47	and	0	2	1		hpt3, pt34
48	gin48	or	6	0	0	gin49, gin50, gin51, gin52, gin53, lg0	
49	gin49	and	2	0	0	gin11, gin19	
50	gin50	and	2	0	0	gin11, gin29	
51	gin51	and	2	0	0	gin11, gin44	
52	gin52	and	2	0	0	gin19, gin29	
53	gin53	and	2	0	0	gin19, gin44	
54	gin54	or	4	0	0	gin55, gin56, gin57, lg01	
55	gin55	and	3	0	0	gin11, gin19, gin29	
56	gin56	and	3	0	0	gin11, gin19, gin44	
57	gin57	and	3	0	0	gin19, gin29, gin44	
58	gin58	or	4	0	0	gin11, gin19, gin29, gin44	
59	lg0	and	2	0	0	gin29, gin44	
60	lg01	and	3	0	0	gin11, gin29, gin44	
61	gb	or	3	0	0	gb1, gb2, gb3	
62	gb1	or	8	0	0	gb11, gb12, gb13, gb14, gb15, gb16, gb17, gb18	
63	gb2	or	7	0	0	gb22, gb23, gb24, gb25, gb26, gb27, gb28	
64	gb3	or	6	0	0	b3tg1, b4tg1, b5tg1, b6tg1, b7tg1, b8tg1	
65	gb11	and	1	1	1	gb111	h1p
66	gb12	and	1	1	1	gb121	h2p
67	gb13	and	1	1	1	gb131	h3p
68	gb14	and	1	1	1	gb141	h4p
69	gb15	and	1	1	1	gb151	h5p
70	gb16	and	1	1	1	gb161	h6p
71	gb17	and	1	1	1	gb171	h7p
72	gb18	and	1	1	1	gb181	h8p
73	gb22	and	1	1	1	gb221	h2p
74	gb23	and	1	1	1	gb231	h3p
75	gb24	and	1	1	1	gb241	h4p
76	gb25	and	1	1	1	gb251	h5p
77	gb26	and	1	1	1	gb261	h6p
78	gb27	and	1	1	1	gb271	h7p
79	gb28	and	1	1	1	gb281	h8p
80	gb111	or	2	0	0	gb112, gb113	
81	gb112	and	1	1	1	gb126	he1
82	gb113	and	1	1	1	gb1213	he0
83	gb121	or	3	0	0	gb122, gb123 gb124	
84	gb122	and	1	1	1	gb125	he2
85	gb123	and	1	1	1	gb1212	he1
86	gb124	and	1	1	1	gb1216	he0
87	gb125	and	2	0	0	gb126, gb127	
88	gb126	or	2	0	0	gb128, gb129	
89	gb127	or	2	0	0	gb1210, gb1211	
90	gb128	and	1	1	1	b1g7	heno1
91	gb129	or	0	9	0		cvb1, prvo1, dvo, svo1, fb01, ivb021,

Gate No	Gate Name	Gate Type	Input Gate No	Input Event No	House Event No	Input Gate List	Input Event List
							ivc11, ef, m
92	gb131	or	4	0	0	gb132, gb133, gb134, gb135	
93	gb132	and	1	1	1	gb136	he3
94	gb133	and	1	1	1	gb1311	he2
95	gb134	and	1	1	1	gb1312	he1
96	gb135	and	1	1	1	gb1313	he0
97	gb136	and	3	0	0	gb126, gb127, gb137	
98	gb137	or	2	0	0	gb138, gb1310	
99	gb138	and	1	1	1	b1g7	heno3
100	gb141	or	5	0	0	gb142, gb143, gb144, gb145, gb146	
101	gb142	and	1	1	1	gb147	he4
102	gb143	and	1	1	1	gb1411	he3
103	gb144	and	1	1	1	gb1412	he2
104	gb145	and	1	1	1	gb1413	he1
105	gb146	and	1	1	1	gb1414	he0
106	gb147	and	4	0	0	gb126, gb127, gb137, gb148	
107	gb148	or	2	0	0	gb149, gb1410	
108	gb149	and	1	1	1	b1g7	heno4
109	gb151	or	4	0	0	gb152, gb153, b5hg5, gb154	
110	gb152	and	1	1	1	gb155	he4
111	gb153	and	1	1	1	gb156	he3
112	b5hg5	and	1	1	1	gb157	he2
113	gb154	and	1	1	1	gb159	he1
114	gb155	and	5	0	0	gb126, gb127, gb137, gb148, gb1213	
115	gb156	and	5	0	0	gb126, gb127, gb137, gb1213, gb1217	
116	gb157	and	5	0	0	gb126, gb127, gb1213, gb1217, gb1314	
117	gb159	and	5	0	0	gb126, gb1213, gb1217, gb1314, gb1415	
118	gb161	or	3	0	0	gb162, gb163, gb164	
119	gb162	and	1	1	1	gb165	he4
120	gb163	and	1	1	1	gb166	he3
121	gb164	and	1	1	1	gb167	he2
122	gb165	and	6	0	0	gb126, gb127, gb137, gb148, gb1213, gb1217	
123	gb166	and	6	0	0	gb126, gb127, gb137, gb1213, gb1217, gb1314	
124	gb167	and	6	0	0	gb126, gb127, gb1213, gb1217, gb1314, gb1415	
125	gb171	or	2	0	0	gb172, gb175	
126	gb172	and	1	1	1	gb174	he4
127	gb174	and	7	0	0	gb126, gb127, gb137, gb148, gb1213, gb1217, gb1314	
128	gb175	and	1	1	1	gb176	he3

Gate No	Gate Name	Gate Type	Input Gate No	Input Event No	House Event No	Input Gate List	Input Event List
129	gb176	and	7	0	0	gb126, gb127, gb137, gb1213, gb1217, gb1314, gb1415	
130	gb181	and	8	0	0	gb126, gb127, gb137, gb148, gb1213, gb1217, gb1314, gb1415	
131	gb1210	and	1	1	1	b1g7	heno2
132	gb1211	or	0	9	0		cvb1, prvo1, dvo, svo1, fb1, ivb1, ivc12, ef, m
133	gb1212	and	2	0	0	gb126, gb1213	
134	gb1213	or	2	0	0	gb1214, gb1215	
135	gb1214	and	1	1	1	b1g10	hdno1
136	gb1215	or	1	7	0	gdfsr1	cvb2, prvo2, dvo, svo2, fb02, ivb022, ivc21
137	gb1216	and	2	0	0	gb1213, gb1217	
138	gb1217	or	2	0	0	gb1218, gb1219	
139	gb1218	and	1	1	1	b1g10	hdno2
140	gb1219	or	1	7	0	gdfsr1	cvb2, prvo2, dvo, svo2, fb2, ivb2, ivc22
141	gb1310	or	0	9	0		cvb3, prvo3, dvo, svo3, fb3, ivb3, ivc31, ef, m
142	gb1311	and	3	0	0	gb126, gb127, gb1213	
143	gb1312	and	3	0	0	gb126, gb1213, gb1217	
144	gb1313	and	3	0	0	gb1213, gb1217, gb1314	
145	gb1314	or	2	0	0	gb1315, gb1316	
146	gb1315	and	1	1	1	b1g10	hdno3
147	gb1316	or	1	7	0	gdfsr1	cvb3, prvo3, dvo, svo3, fb3, ivb3, ivc32
148	gb1410	or	0	9	0		cvb4, prvo4, dvo, svo4, fb4, ivb4, ivc41, ef, m
149	gb1411	and	4	0	0	gb126, gb127, gb137, gb1213	
150	gb1412	and	4	0	0	gb126, gb127, gb1213, gb1217	
151	gb1413	and	4	0	0	gb126, gb1213, gb1217, gb1314	
152	gb1414	and	4	0	0	gb1213, gb1217, gb1314, gb1415	
153	gb1415	or	2	0	0	gb1416, gb1417	
154	gb1416	and	1	1	1	b1g10	hdno4
155	gb1417	or	1	7	0	gdfsr1	cvb4, prvo4, dvo, svo4, fb4, ivb4, ivc42
156	b1g7	or	0	2	0		pfe100, m
157	b1g10	or	0	2	0		pfd100, m
158	gdfsr1	or	1	1	0	gdfsr2	ef
159	gdfsr2	or	0	4	0		tivc, tivb, laf, oaf

Gate No	Gate Name	Gate Type	Input Gate No	Input Event No	House Event No	Input Gate List	Input Event List
160	gb221	or	3	0	0	gate1, gate2, gate3	
161	gb226	or	2	0	0	gb228, gb229	
162	gb227	or	2	0	0	gb2215, gb2216	
163	gb228	and	1	1	1	gb2210	heno1
164	gb229	or	0	9	0		cvb5, prvo5, dvo, svo5, fb5, ivb5, ivc51, ef, m
165	gb231	or	4	0	0	gate7, gate8, gate9, gate10	
166	gb237	or	2	0	0	gb238, gb239	
167	gb238	and	1	1	1	gb2210	heno3
168	gb239	or	0	9	0		cvb5, prvo5, dvo, svo5, fb5, ivb5, ivc52, ef, m
169	gb241	or	5	0	0	gate27, gate28, gate29, gate30, gate31	
170	gb248	or	2	0	0	gb249, gb2410	
171	gb249	and	1	1	1	gb2210	heno4
172	gb251	or	4	0	0	gate53, gate54, gate55, gate56	
173	gb261	or	3	0	0	gate61, gate62, gate63	
174	gb271	or	2	0	0	gate100, gate101	
175	gb281	or	8	0	0	gate117, gate118, gate119, gate120, gate121, gate122 gate123, gate124	
176	gb2210	or	2	0	0	gb2211, gb2212	
177	gb2211	and	1	1	1	gb2213	h501
178	gb2212	and	1	1	1	gb2214	h502
179	gb2213	or	0	2	0		pfe501, m
180	gb2214	or	0	2	0		pfe502, m
181	gb2215	and	1	1	1	gb2210	heno2
182	gb2216	or	0	9	0		cvb6, prvo6, dvo, svo6, fb6, ivb6, ivc61, ef, m
183	gb2218	or	2	0	0	gb2219, gb2220	
184	gb2219	and	1	1	1	gb2221	hdno1
185	gb2220	or	1	7	0	gdfsr1	cvb6, prvo6, dvo, svo6, fb6, ivb6, ivc62
186	gb2221	or	2	0	0	gb2222, gb2223	
187	gb2222	and	1	1	1	gb2224	h501
188	gb2223	and	1	1	1	gb2225	h502
189	gb2224	or	0	2	0		pfd501, m
190	gb2225	or	0	2	0		pfd502 m
191	gb2227	or	2	0	0	gb2228, gb2229	
192	gb2228	and	1	1	1	gb2221	hdno2
193	gb2229	or	1	7	0	gdfsr1	cvb7, prvo7, dvo, svo7, fb7, ivb, ivc71
194	gb2317	or	2	0	0	gb2318, gb2319	
195	gb2318	and	1	1	1	gb2221	hdno3

Gate No	Gate Name	Gate Type	Input Gate No	Input Event No	House Event No	Input Gate List	Input Event List
196	gb2319	or	1	7	0	gdfsr1	cvb7, prvo7, dvo, svo7, fb7, ivb7, ivc72
197	gb2410	or	0	9	0		cvb8, prvo8, dvo, svo8, fb8, ivb8, ivc81, ef, m
198	gb2421	or	2	0	0	gb2422, gb2423	
199	gb2422	and	1	1	1	gb2221	hdno4
200	gb2423	or	1	7	0	gdfsr1	cvb8, prvo8, dvo, svo8, fb8, ivb8, ivc82
201	b3tg1	and	1	1	1	b3tg2	h3p
202	b4tg1	and	1	1	1	b4tg2	h4p
203	b5tg1	and	1	1	1	b5tg2	h5p
204	b6tg1	and	1	1	1	b6tg2	h6p
205	b7tg1	and	1	1	1	b7tg2	h7p
206	b8tg1	and	1	1	1	b8tg2	h8p
207	b3tg2	or	4	0	0	b3tg4, b3tg5, b3tg6, b3tg7	
208	b3tg4	and	1	1	1	b3tg8	he3
209	b3tg5	and	1	1	1	b3tg20	he2
210	b3tg6	and	1	1	1	b3tg29	he1
211	b3tg7	and	1	1	1	b3tg31	he0
212	b3tg8	or	3	0	0	b3tg9, b3tg10, b3tg11	
213	b3tg9	or	2	0	0	b3tg12, gb229	
214	b3tg10	or	2	0	0	b3tg18, gb2216	
215	b3tg11	or	2	0	0	b3tg19, gb239	
216	b3tg12	and	1	1	1	b3tg13	heno1
217	b3tg13	or	2	0	0	b3tg14, b3tg15	
218	b3tg14	and	1	1	1	b3tg16	h331
219	b3tg15	and	1	1	1	b3tg17	h332
220	b3tg16	or	0	2	0		pfe331, m
221	b3tg17	or	0	2	0		pfe332, m
222	b3tg18	and	1	1	1	b3tg13	heno2
223	b3tg19	and	1	1	1	b3tg13	heno3
224	b3tg20	or	2	0	0	b3tg21, b3tg22	
225	b3tg21	or	2	0	0	b3tg9, b3tg10	
226	b3tg22	or	2	0	0	b3tg23, gld1	
227	b3tg23	and	1	1	1	b3tg24	hdno1
228	b3tg24	or	2	0	0	b3tg25, b3tg26	
229	b3tg25	and	1	1	1	b3tg27	h331
230	b3tg26	and	1	1	1	b3tg28	h332
231	b3tg27	or	0	2	0		pfd331, m
232	b3tg28	or	0	2	0		pfd332, m
233	b3tg29	or	2	0	0	b3tg9, b3tg30	
234	b3tg30	or	2	0	0	b3tg22, b3tg32	
235	b3tg31	or	3	0	0	b3tg22, b3tg32, b3tg34	
236	b3tg32	or	2	0	0	b3tg33, gld2	
237	b3tg33	and	1	1	1	b3tg24	hdno2
238	b3tg34	or	2	0	0	b3tg35, gld3	
239	b3tg35	and	1	1	1	b3tg24	hdno4
240	b4tg2	or	5	0	0	b4tg3, b4tg4, b4tg5, b4tg6, b4tg7	

Gate No	Gate Name	Gate Type	Input Gate No	Input Event No	House Event No	Input Gate List	Input Event List
241	b4tg3	and	1	1	1	b4tg8	he4
242	b4tg4	and	1	1	1	b4tg11	he3
243	b4tg5	and	1	1	1	b4tg14	he2
244	b4tg6	and	1	1	1	b4tg18	he1
245	b4tg7	and	1	1	1	b4tg21	he0
246	b4tg8	or	6	0	0	lg1, lg2, lg3, lg4, lg5, lg6	
247	b4tg9	or	2	0	0	b4tg10, gb2410	
248	b4tg10	and	1	1	1	b3tg13	heno4
249	b4tg11	or	2	0	0	b4tg12, b4tg13	
250	b4tg12	or	3	0	0	lg13, lg14, lg15	
251	b4tg13	and	2	0	0	b3tg8, b3tg22	
252	b4tg14	or	3	0	0	b4tg15, b4tg16, b4tg17	
253	b4tg15	and	2	0	0	b3tg9, b3tg10	
254	b4tg16	and	2	0	0	b3tg21, b3tg30	
255	b4tg17	and	2	0	0	b3tg22, b3tg32	
256	b4tg18	or	2	0	0	b4tg19, b4tg20	
257	b4tg19	and	2	0	0	b3tg9, b3tg31	
258	b4tg20	or	3	0	0	lg16, lg17, lg18	
259	b4tg21	or	6	0	0	lg7, lg8, lg9, lg10, lg11, lg12	
260	b4tg22	or	2	0	0	b4tg23, gld4	
261	b4tg23	and	1	1	1	b3tg24	hd_no4
262	b5tg2	or	4	0	0	b5tg3, b5tg4, b5tg5, b5tg6	
263	b5tg3	and	1	1	1	b5tg7	he4
264	b5tg4	and	1	1	1	b5tg10	he3
265	b5tg5	and	1	1	1	b5tg14	he2
266	b5tg6	and	1	1	1	b5tg18	he1
267	b5tg7	or	2	0	0	b5tg8, b5tg9	
268	b5tg8	or	4	0	0	lg19, lg20, lg21, lg22	
269	b5tg9	and	2	0	0	b4tg8, b3tg22	
270	b5tg10	or	3	0	0	b5tg11, b5tg12, b5tg13	
271	b5tg11	and	3	0	0	b3tg9, b3tg10, b3tg11	
272	b5tg12	and	2	0	0	b4tg12, b3tg30	
273	b5tg13	and	2	0	0	b3tg8, b4tg17	
274	b5tg14	or	3	0	0	b5tg15, b5tg16, b5tg17	
275	b5tg15	and	2	0	0	b4tg15, b3tg31	
276	b5tg16	and	2	0	0	b3tg21, b4tg20	
277	b5tg17	and	3	0	0	b3tg22, b3tg32, b3tg34	
278	b5tg18	or	2	0	0	b5tg19, b5tg20	
279	b5tg19	and	2	0	0	b3tg9, b4tg21	
280	b5tg20	or	4	0	0	lg23, lg24, lg25, lg26	
281	b6tg2	or	3	0	0	b6tg3, b6tg4, b6tg5	
282	b6tg3	and	1	1	1	b6tg6	he4
283	b6tg4	and	1	1	1	b6tg10	he3
284	b6tg5	and	1	1	1	lb1	he2
285	b6tg6	or	3	0	0	b6tg7, b6tg8, b6tg9	
286	b6tg7	and	4	0	0	b3tg9, b3tg10, b3tg11, b4tg9	
287	b6tg8	and	2	0	0	b5tg8, b3tg30	

Gate No	Gate Name	Gate Type	Input Gate No	Input Event No	House Event No	Input Gate List	Input Event List
288	b6tg9	and	2	0	0	b4tg8, b4tg17	
289	b6tg10	or	3	0	0	b6tg11, b6tg12, b6tg13	
290	b6tg11	and	2	0	0	b5tg11, b3tg31	
291	b6tg12	and	2	0	0	b4tg12, b4tg20	
292	b6tg13	and	2	0	0	b3tg8, b5tg17	
293	b7tg2	or	2	0	0	b7tg3, b7tg4	
294	b7tg3	and	1	1	1	b7tg5	he4
295	b7tg4	and	1	1	1	lb19	he3
296	b7tg5	or	3	0	0	b7tg6, b7tg7, b7tg8	
297	b7tg6	and	5	0	0	b3tg9, b3tg10, b3tg11, b4tg9, b3tg31	
298	b7tg7	and	2	0	0	b5tg8, b4tg20	
299	b7tg8	and	2	0	0	b4tg8, b5tg17	
300	b8tg2	and	1	1	1	b8tg3	he4
301	b8tg3	or	3	0	0	b8tg4, b8tg5, b8tg6	
302	b8tg4	and	5	0	0	b3tg9, b3tg10, b3tg11, b4tg9, b4tg21	
303	b8tg5	and	2	0	0	b5tg8, b5tg20	
304	b8tg6	and	5	0	0	b4tg8, b3tg22, b3tg32, b3tg34, b4tg22	
305	gld1	or	1	7	0	b3tg24	cvb9, prvo9, dvo, svo9, fb9, ivb9, ivc91
306	gld2	or	1	7	0	b3tg24	cvb9, prvo9, dvo, svo9, fb9, ivb9, ivc92
307	gld3	or	1	7	0	b3tg24	cvb10, prvo10, dvo, svo10, fb10, ivb10, ivc101
308	gld4	or	1	7	0	b3tg24	cvb10, prvo10, dvo, svo10, fb10, ivb10, ivc102
309	lg1	and	2	0	0	b3tg9, b3tg10	
310	lg2	and	2	0	0	b3tg9, b3tg11	
311	lg3	and	2	0	0	b3tg9, b4tg9	
312	lg4	and	2	0	0	b3tg10, b3tg11	
313	lg5	and	2	0	0	b3tg10, b4tg9	
314	lg6	and	2	0	0	b3tg11, b4tg9	
315	lg7	and	2	0	0	b3tg22, b3tg32	
316	lg8	and	2	0	0	b3tg22, b3tg34	
317	lg9	and	2	0	0	b3tg22, b4tg22	
318	lg10	and	2	0	0	b3tg32, b3tg34	
319	lg11	and	2	0	0	b3tg32, b4tg22	
320	lg12	and	2	0	0	b3tg34, b4tg22	
321	lg13	and	2	0	0	b3tg9, b3tg10	
322	lg14	and	2	0	0	b3tg9, b3tg11	
323	lg15	and	2	0	0	b3tg10, b3tg11	
324	lg16	and	2	0	0	b3tg22, b3tg32	
325	lg17	and	2	0	0	b3tg22, b3tg34	
326	lg18	and	2	0	0	b3tg32, b3tg34	
327	lg19	and	3	0	0	b3tg9, b3tg10, b3tg11	
328	lg20	and	3	0	0	b3tg9, b3tg10, b4tg9	
329	lg21	and	3	0	0	b3tg9, b3tg11, b4tg9	

Gate No	Gate Name	Gate Type	Input Gate No	Input Event No	House Event No	Input Gate List	Input Event List
330	lg22	and	3	0	0	b3tg10, b3tg11, b4tg9	
331	lg23	and	3	0	0	b3tg22, b3tg32, b3tg34	
332	lg24	and	3	0	0	b3tg22, b3tg32, b4tg22	
333	lg25	and	3	0	0	b3tg22, b3tg34, b4tg22	
334	lg26	and	3	0	0	b3tg32, b3tg34, b4tg22	
335	lb1	or	3	0	0	lb2, lb3, lb4	
336	lb2	or	6	0	0	lb5, lb6, lb7, lb8, lb9, lb10	
337	lb3	or	8	0	0	lb11, lb12, lb13, lb14, lb15, lb16, lb17, lb18	
338	lb4	and	4	0	0	b3tg22, b3tg32, b3tg34, b4tg22	
339	lb5	and	4	0	0	b3tg9, b3tg10, b3tg22, b3tg32	
340	lb6	and	4	0	0	b3tg9, b3tg10, b3tg22, b3tg34	
341	lb7	and	4	0	0	b3tg9, b3tg10, b3tg22, b4tg22	
342	lb8	and	4	0	0	b3tg9, b3tg10, b3tg32, b3tg34	
343	lb9	and	4	0	0	b3tg9, b3tg10, b3tg32, b4tg22	
344	lb10	and	4	0	0	b3tg9, b3tg10, b3tg34, b4tg22	
345	lb11	and	4	0	0	b3tg9, b3tg22, b3tg32, b3tg34	
346	lb12	and	4	0	0	b3tg9, b3tg22, b3tg32, b4tg22	
347	lb13	and	4	0	0	b3tg9, b3tg22, b3tg34, b4tg22	
348	lb14	and	4	0	0	b3tg9, b3tg32, b3tg34, b4tg22	
349	lb15	and	4	0	0	b3tg10, b3tg22, b3tg32, b3tg34	
350	lb16	and	4	0	0	b3tg10, b3tg22, b3tg32, b4tg22	
351	lb17	and	4	0	0	b3tg10, b3tg22, b3tg34, b4tg22	
352	lb18	and	4	0	0	b3tg10, b3tg32, b3tg34, b4tg22	
353	lb19	or	3	0	0	lb20, lb21, lb22	
354	lb20	or	6	0	0	lb23, lb24, lb25, lb26, lb27, lb28	
355	lb21	and	2	0	0	lb29, lb30	
356	lb22	or	3	0	0	lb38, lb39, lb40	
357	lb23	and	5	0	0	b3tg9, b3tg10, b3tg11, b3tg22, b3tg32	
358	lb24	and	5	0	0	b3tg9, b3tg10, b3tg11, b3tg22, b3tg34	
359	lb25	and	5	0	0	b3tg9, b3tg10, b3tg11, b3tg22, b4tg22	
360	lb26	and	5	0	0	b3tg9, b3tg10, b3tg11,	

Gate No	Gate Name	Gate Type	Input Gate No	Input Event No	House Event No	Input Gate List	Input Event List
						b3tg32, b3tg34	
361	lb27	and	5	0	0	b3tg9, b3tg10, b3tg11, b3tg32, b4tg22	
362	lb28	and	5	0	0	b3tg9, b3tg10, b3tg11, b3tg34, b4tg22	
363	lb29	or	3	0	0	lb31, lb32, lb33	
364	lb30	or	4	0	0	lb34, lb35, lb36, lb37	
365	lb31	and	2	0	0	b3tg9, b3tg10	
366	lb32	and	2	0	0	b3tg9, b3tg11	
367	lb33	and	2	0	0	b3tg10, b3tg11	
368	lb34	and	3	0	0	b3tg22, b3tg32, b3tg34	
369	lb35	and	3	0	0	b3tg22, b3tg32, b4tg22	
370	lb36	and	3	0	0	b3tg22, b3tg34, b4tg22	
371	lb37	and	3	0	0	b3tg32, b3tg34, b4tg22	
372	lb38	and	5	0	0	b3tg9, b3tg22, b3tg32, b3tg34, b4tg22	
373	lb39	and	5	0	0	b3tg10, b3tg22, b3tg32, b3tg34, b4tg22	
374	lb40	and	5	0	0	b3tg11, b3tg22, b3tg32, b3tg34, b4tg22	
375	gate1	and	1	1	1	gate4	he2
376	gate2	and	1	1	1	gate5	he1
377	gate3	and	1	1	1	gate6	he0
378	gate4	or	2	0	0	gb226, gb227	
379	gate5	or	2	0	0	gb226, gb2218	
380	gate6	or	2	0	0	gb2218, gb2227	
381	gate7	and	1	1	1	gate11	he3
382	gate8	and	1	1	1	gate12	he2
383	gate9	and	1	1	1	gate13	he1
384	gate10	and	1	1	1	gate14	he0
385	gate11	or	3	0	0	gate15, gate16, gate17	
386	gate12	or	3	0	0	gate21, gate22, gate23	
387	gate13	or	3	0	0	gate24, gate25, gate26	
388	gate14	or	3	0	0	gate18, gate19, gate20	
389	gate15	and	2	0	0	gb226, gb227	
390	gate16	and	2	0	0	gb226, gb237	
391	gate17	and	2	0	0	gb227, gb237	
392	gate18	and	2	0	0	gb2218, gb2227	
393	gate19	and	2	0	0	gb2218, gb2317	
394	gate20	and	2	0	0	gb2227, gb2317	
395	gate21	and	2	0	0	gb226, gb227	
396	gate22	and	2	0	0	gb226, gb2218	
397	gate23	and	2	0	0	gb227, gb2218	
398	gate24	and	2	0	0	gb226, gb2218	
399	gate25	and	2	0	0	gb226 gb2227	
400	gate26	and	2	0	0	gb2218 gb2227	
401	gate27	and	1	1	1	gate32	he4
402	gate28	and	1	1	1	gate33	he3

Gate No	Gate Name	Gate Type	Input Gate No	Input Event No	House Event No	Input Gate List	Input Event List
403	gate29	and	1	1	1	gate34	he2
404	gate30	and	1	1	1	gate35	he1
405	gate31	and	1	1	1	gate36	he0
406	gate32	or	4	0	0	gate37, gate38, gate39, gate40	
407	gate33	or	4	0	0	gate37, gate41, gate42, gate43	
408	gate34	or	4	0	0	gate41, gate44, gate45, gate46	
409	gate35	or	4	0	0	gate45, gate47, gate48, gate49	
410	gate36	or	4	0	0	gate49, gate50, gate51, gate52	
411	gate37	and	3	0	0	gb226, gb227, gb237	
412	gate38	and	3	0	0	gb226, gb227, gb248	
413	gate39	and	3	0	0	gb226, gb237, gb248	
414	gate40	and	3	0	0	gb227, gb237, gb248	
415	gate41	and	3	0	0	gb226, gb227, gb2218	
416	gate42	and	3	0	0	gb226, gb237, gb2218	
417	gate43	and	3	0	0	gb227, gb237, gb2218	
418	gate44	and	3	0	0	gb226, gb227, gb2227	
419	gate45	and	3	0	0	gb226, gb2218, gb2227	
420	gate46	and	3	0	0	gb227, gb2218, gb2227	
421	gate47	and	3	0	0	gb226, gb2218, gb2317	
422	gate48	and	3	0	0	gb226, gb2227, gb2317	
423	gate49	and	3	0	0	gb2218, gb2227, gb2317	
424	gate50	and	3	0	0	gb2218, gb2227, gb2421	
425	gate51	and	3	0	0	gb2218, gb2317, gb2421	
426	gate52	and	3	0	0	gb2227, gb2317, gb2421	
427	gate53	and	1	1	1	gate57	he4
428	gate54	and	1	1	1	gate58	he3
429	gate55	and	1	1	1	gate59	he2
430	gate56	and	1	1	1	gate60	he1
431	gate57	or	5	0	0	gate65, gate66, gate67, gate68, gate69	
432	gate58	or	5	0	0	gate67, gate70, gate71, gate72, gate73	
433	gate59	or	5	0	0	gate71, gate74, gate75, gate76, gate77	
434	gate60	or	5	0	0	gate76, gate78, gate79, gate80, gate81	
435	gate61	and	1	1	1	gate64	he4
436	gate62	and	1	1	1	gate82	he3
437	gate63	and	1	1	1	gate83	he2
438	gate64	or	6	0	0	gate84, gate85, gate86, gate87, gate88, gate89	
439	gate65	and	4	0	0	gb226, gb227, gb237, gb248	
440	gate66	and	4	0	0	gb227, gb237, gb248, gb2218	
441	gate67	and	4	0	0	gb226, gb227, gb237,	

Gate No	Gate Name	Gate Type	Input Gate No	Input Event No	House Event No	Input Gate List	Input Event List
						gb2218	
442	gate68	and	4	0	0	gb226, gb237, gb248, gb2218	
443	gate69	and	4	0	0	gb226, gb227, gb248, gb2218	
444	gate70	and	4	0	0	gb226, gb227, gb237, gb2227	
445	gate71	and	4	0	0	gb226, gb227, gb2218, gb2227	
446	gate72	and	4	0	0	gb226, gb237, gb2218, gb2227	
447	gate73	and	4	0	0	gb227, gb237, gb2218, gb2227	
448	gate74	and	4	0	0	gb226, gb227, gb2218, gb2317	
449	gate75	and	4	0	0	gb226, gb227, gb2227, gb2317	
450	gate76	and	4	0	0	gb226, gb2218, gb2227, gb2317	
451	gate77	and	4	0	0	gb227, gb2218, gb2227, gb2317	
452	gate78	and	4	0	0	gb2218, gb2227, gb2317, gb2421	
453	gate79	and	4	0	0	gb226, gb2227, gb2317, gb2421	
454	gate80	and	4	0	0	gb226, gb2218, gb2227, gb2421	
455	gate81	and	4	0	0	gb226, gb2218, gb2317, gb2421	
456	gate82	or	6	0	0	gate86, gate90, gate91, gate92, gate93, gate94	
457	gate83	or	6	0	0	gate92, gate95, gate96, gate97, gate98, gate99	
458	gate84	and	5	0	0	gb226, gb227, gb237, gb248, gb2218	
459	gate85	and	5	0	0	gb226, gb227, gb237, gb248, gb2227	
460	gate86	and	5	0	0	gb226, gb227, gb237, gb2218, gb2227	
461	gate87	and	5	0	0	gb226, gb227, gb248, gb2218, gb2227	
462	gate88	and	5	0	0	gb227, gb237, gb248, gb2218, gb2227	
463	gate89	and	5	0	0	gb226, gb237, gb248, gb2218, gb2227	
464	gate90	and	5	0	0	gb226, gb227, gb248, gb2218, gb2317	
465	gate91	and	5	0	0	gb226, gb227, gb248, gb2227, gb2317	
466	gate92	and	5	0	0	gb226, gb227, gb2218, gb2227, gb2317	

Gate No	Gate Name	Gate Type	Input Gate No	Input Event No	House Event No	Input Gate List	Input Event List
467	gate93	and	5	0	0	gb226, gb248, gb2218, gb2227, gb2317	
468	gate94	and	5	0	0	gb227, gb237, gb2218, gb2227, gb2317	
469	gate95	and	5	0	0	gb226, gb227, gb2218, gb2227, gb2421	
470	gate96	and	5	0	0	gb226, gb227, gb2218, gb2317, gb2421	
471	gate97	and	5	0	0	gb226, gb227, gb2227, gb2317, gb2421	
472	gate98	and	5	0	0	gb226, gb2218, gb2227, gb2317, gb2421	
473	gate99	and	5	0	0	gb227, gb2218, gb2227, gb2317, gb2421	
474	gate100	and	1	1	1	gate102	he4
475	gate101	and	1	1	1	gate103	he3
476	gate102	or	7	0	0	gate104, gate105, gate106, gate107, gate108, gate109, gate110	
477	gate103	or	7	0	0	gate107, gate111, gate112, gate113, gate114, gate115, gate116	
478	gate104	and	6	0	0	gb226, gb227, gb237, gb248, gb2218, gb2227	
479	gate105	and	6	0	0	gb226, gb227, gb237, gb248, gb2218, gb2317	
480	gate106	and	6	0	0	gb226, gb227, gb237, gb248, gb2227, gb2317	
481	gate107	and	6	0	0	gb226, gb227, gb237, gb2218, gb2227, gb2317	
482	gate108	and	6	0	0	gb226, gb227, gb248, gb2218, gb2227, gb2317	
483	gate109	and	6	0	0	gb226, gb237, gb248, gb2218, gb2227, gb2317	
484	gate110	and	6	0	0	gb227, gb237, gb248, gb2218, gb2227, gb2317	
485	gate111	and	6	0	0	gb226, gb227, gb237, gb2218, gb2227, gb2421	
486	gate112	and	6	0	0	gb226, gb227, gb237, gb2218, gb2317, gb2421	
487	gate113	and	6	0	0	gb226, gb227, gb2317, gb2227, gb2317, gb2421	
488	gate114	and	6	0	0	gb226, gb227, gb2218, gb2227, gb2317, gb2421	
489	gate115	and	6	0	0	gb226, gb237, gb2218, gb2227, gb2317, gb2421	
490	gate116	and	6	0	0	gb227, gb237, gb2218, gb2227, gb2317, gb2421	
491	gate117	and	7	0	0	gb226, gb227, gb237,	

Gate No	Gate Name	Gate Type	Input Gate No	Input Event No	House Event No	Input Gate List	Input Event List
						gb248, gb2218, gb2227, gb2317	
492	gate118	and	7	0	0	gb226, gb227, gb237, gb248, gb2218, gb2227, gb2421	
493	gate119	and	7	0	0	gb226, gb227, gb237, gb248, gb2218, gb2317, gb2421	
494	gate120	and	7	0	0	gb226, gb227, gb237, gb248, gb2227, gb2317, gb2421	
495	gate121	and	7	0	0	gb226, gb227, gb237, gb2218, gb2227, gb2317, gb2421	
496	gate122	and	7	0	0	gb226, gb227, gb248, gb2218, gb2227, gb2317, gb2421	
497	gate123	and	7	0	0	gb226, gb237, gb248, gb2218, gb2227, gb2317, gb2421	
498	gate124	and	7	0	0	gb227, gb237, gb248, gb2218, gb2227, gb2317, gb2421	
499	gc	or	2	0	0	gc1, gc2	
500	gc1	or	4	0	0	gapb1, gapb2, gapb3, gapb4	
501	gc2	or	0	2	0		tivb, tivc
502	gapb1	and	1	1	1	ab1g1	ha1p
503	gapb2	and	1	1	1	ab2g1	ha2p
504	gapb3	and	1	1	1	ab3g1	ha3p
505	gapb4	and	1	1	1	ab4g1	ha4p
506	ab1g1	and	1	1	1	ab1g2	ha100
507	ab1g2	or	2	0	0	ab1g3, ab1g4	
508	ab1g3	and	1	1	1	ab1g5	hae1
509	ab1g4	and	1	1	1	ab1g8	hae0
510	ab1g5	or	2	0	0	ab1g6, age1l	
511	ab1g6	and	1	1	1	ab1g7	haeno1
512	ab1g7	or	0	2	0		apfe100, m
513	ab1g8	or	2	0	0	ab1g9, agd1	
514	ab1g9	and	1	1	1	ab1g10	hadno1
515	ab1g10	or	0	2	0		apfd100, m
516	ab2g1	or	2	0	0	ab2hg1, ab2fg1	
517	ab3g1	or	2	0	0	ab3hg1, ab3fg1	
518	ab4g1	or	2	0	0	ab4hg1, ab4fg1	
519	ab2hg1	and	1	1	1	ab2hg2	ha100
520	ab2hg2	or	3	0	0	ab2hg3, ab2hg4, ab2hg5	
521	ab2hg3	and	1	1	1	ab2hg6	hae2
522	ab2hg4	and	1	1	1	ab2hg9	hae1
523	ab2hg5	and	1	1	1	ab2hg10	hae0
524	ab2hg6	and	2	0	0	ab1g5, ab2hg7	
525	ab2hg7	or	2	0	0	ab2hg8, age2l	

Gate No	Gate Name	Gate Type	Input Gate No	Input Event No	House Event No	Input Gate List	Input Event List
526	ab2hg8	and	1	1	1	ab1g7	haeno2
527	ab2hg9	and	2	0	0	ab1g5, ab1g8	
528	ab2hg10	and	2	0	0	ab1g8, ab2hg11	
529	ab2hg11	or	2	0	0	ab2hg12, agd2	
530	ab2hg12	and	1	1	1	ab1g10	had_no2
531	ab2fg1	or	3	0	0	ab2fg3, ab2fg4, ab2fg5	
532	ab2fg3	and	1	1	1	ab2fg6	hae2
533	ab2fg4	and	1	1	1	ab2fg12	hae1
534	ab2fg5	and	1	1	1	ab2fg16	hae0
535	ab2fg6	or	2	0	0	ab2fg7, ab2fg8	
536	ab2fg7	or	2	0	0	zx1, age1l	
537	ab2fg8	or	2	0	0	zx14, age2l	
538	ab2fg12	or	2	0	0	ab2fg7, ab2fg13	
539	ab2fg13	or	2	0	0	zx7, agd1	
540	ab2fg16	or	2	0	0	ab2fg13, ab2fg17	
541	ab2fg17	or	2	0	0	zx13, agd2	
542	ab3hg1	and	1	1	1	ab3hg2	ha100
543	ab3hg2	or	2	0	0	ab3hg3, ab3hg4	
544	ab3hg3	and	1	1	1	ab3hg5	hae2
545	ab3hg4	and	1	1	1	ab3hg6	hae1
546	ab3hg5	and	3	0	0	ab1g5, ab2hg7, ab1g8	
547	ab3hg6	and	3	0	0	ab1g5, ab1g8, ab2hg11	
548	ab3fg1	and	1	1	1	ab3fg2	ha50
549	ab3fg2	or	2	0	0	ab3fg3, ab3fg4	
550	ab3fg3	and	1	1	1	ab3fg5	hae2
551	ab3fg4	and	1	1	1	ab3fg8	hae1
552	ab3fg5	or	2	0	0	ab3fg6, ab3fg7	
553	ab3fg6	and	2	0	0	ab2fg7, ab2fg8	
554	ab3fg7	and	2	0	0	ab2fg6, ab2fg13	
555	ab3fg8	or	2	0	0	ab3fg9, ab3fg10	
556	ab3fg9	and	2	0	0	ab2fg7, ab2fg16	
557	ab3fg10	and	2	0	0	ab2fg13, ab2fg17	
558	ab4hg1	and	1	1	1	ab4hg2	ha100
559	ab4hg2	and	1	1	1	ab4hg3	hae2
560	ab4hg3	and	4	0	0	ab1g5, ab2hg7, ab1g8, ab2hg11	
561	ab4fg1	and	1	1	1	ab4fg2	ha50
562	ab4fg2	and	1	1	1	ab4fg3	hae1
563	ab4fg3	or	2	0	0	ab4fg4, ab4fg5	
564	ab4fg4	or	2	0	0	ab3fg6, ab2fg16	
565	ab4fg5	or	2	0	0	ab2fg6, ab3fg10	
566	age1l	or	0	9	0		ef, cvb11, prvo11, svo11, fb11, tivb, tivc, ivc111, ivb011
567	age2l	or	0	9	0		ef, cvb11, prvo11, svo11, fb11, tivb, tivc, ivc112, ivb11
568	agd1	or	1	9	0	agdfsr1	ef, cvb12, prvo12, svo12, fb12, tivb, tivc, ivc122, ivb12
569	agd2	or	1	9	0	agdfsr1	ef, cvb12, prvo12,

Gate No	Gate Name	Gate Type	Input Gate No	Input Event No	House Event No	Input Gate List	Input Event List
							svo12, fb12, tivb, tivc, ivc121, ivb012
570	agdfsr1	or	1	1	0	agdfsr2	ef
571	agdfsr2	or	0	4	0		tivc, tivb, laf, oaf
572	zx1	and	1	1	1	zx2	haeno1
573	zx2	or	2	0	0	zx3, zx4	
574	zx3	and	1	1	1	zx5	ha501
575	zx4	and	1	1	1	zx6	ha502
576	zx5	or	0	2	0		apfe501, m
577	zx6	or	0	2	0		apfe502, m
578	zx7	and	1	1	1	zx8	hadno1
579	zx8	or	2	0	0	zx9, zx10	
580	zx9	and	1	1	1	zx11	ha501
581	zx10	and	1	1	1	zx12	ha502
582	zx11	or	0	2	0		apfd501, m
583	zx12	or	0	2	0		apfd502, m
584	zx13	and	1	1	1	zx8	hadno2
585	zx14	and	1	1	1	zx2	haeno2
586	gd	or	2	0	0	gd1, gd2	
587	gd1	or	2	0	0	ag1, ag2	
588	gd2	or	2	0	0	wg1, wg2	
589	ag1	or	2	1	0	ag3, ag4	cvb
590	ag2	or	2	0	0	ag13, ag14	
591	ag3	or	0	2	0		ivb, ahe4
592	ag4	or	2	0	0	ag5, ag6	
593	ag5	and	1	1	0	ag7	mrm
594	ag6	or	3	0	0	ag10, ag11, ag12	
595	ag7	or	0	3	0		si, sv1, sv2
596	ag10	and	0	2	1		hav1, av1
597	ag11	and	0	2	1		hav2, av2
598	ag12	and	0	2	1		hav3, av3
599	ag13	and	0	2	1		hold, nbo
600	ag14	and	0	2	1		hnew, nbn
601	wg1	or	2	1	0	wg3, wg4	wbs
602	wg2	or	2	0	0	wg18, wg19	
603	wg3	or	2	0	0	wg5, wg6	
604	wg4	or	2	0	0	wg7, wg8	
605	wg5	or	0	2	0		tivb, whe
606	wg6	or	0	2	0		tivb, whe
607	wg7	and	2	0	0	wg9, wg10	
608	wg8	or	3	0	0	wg15, wg16, wg17	
609	wg9	or	1	3	0	wg11	si, sv1, sv2
610	wg10	or	1	1	0	wg11	mrm
611	wg11	and	2	0	0	wg12, wg13	
612	wg12	and	0	2	1		hold, wvrfo
613	wg13	and	0	2	1		hnew, wvrfn
614	wg15	and	0	2	1		hwv1, wv1
615	wg16	and	0	2	1		hwv2, wv2
616	wg17	and	0	2	1		hwv3, wv3
617	wg18	and	0	2	1		hold, nbo

Gate No	Gate Name	Gate Type	Input Gate No	Input Event No	House Event No	Input Gate List	Input Event List
618	wg19	and	0	2	1		hnew, nbn

Table D.2 FDS Spurious Trip Fault Tree Structure

Gate No	Gate Name	Gate Type	Input Gate Number	Input Event Number	House Event Number	Input Gate List	Input Event List
1	top	or	3	0	0	g1, sg1, sg2	
2	g1	or	4	0	0	g2, g3, g4, g5	
3	g2	and	1	1	1	g6	h1s
4	g3	and	1	1	1	g11	h2s
5	g4	and	1	1	1	g18	h3s
6	g5	and	1	1	1	g29	h4s
7	g6	and	1	1	1	g7	h1ts
8	g7	or	3	0	0	g8, g9, g10	
9	g8	and	0	2	1		hpt1, s1pt1
10	g9	and	0	2	1		hpt2, s1pt2
11	g10	and	0	2	1		hpt3, s1pt3
12	g11	or	2	0	0	g12, g13	
13	g12	and	1	1	1	g12a	h1ts
14	g12a	or	2	0	0	g7, g14	
15	g13	and	1	1	1	g13a	h2ts
16	g13a	and	2	0	0	g7, g14	
17	g14	or	3	0	0	g15, g16, g17	
18	g15	and	0	2	1		hpt1, s2pt1
19	g16	and	0	2	1		hpt2, s2pt2
20	g17	and	0	2	1		hpt3, s2pt3
21	g18	or	3	0	0	g19a, g20a, g21a	
22	g19a	and	1	1	1	g19	h1ts
23	g19	or	3	0	0	g7, g14, g22	
24	g20a	and	1	1	1	g20	h2ts
25	g20	or	3	0	0	g26, g27, g28	
26	g21a	and	1	1	1	g21	h3ts
27	g21	and	3	0	0	g7, g14, g22	
28	g22	or	3	0	0	g23, g24, g25	
29	g23	and	0	2	1		hpt1, s3pt1
30	g24	and	0	2	1		hpt2, s3pt2
31	g25	and	0	2	1		hpt3, s3pt3
32	g26	and	2	0	0	g7, g14	
33	g27	and	2	0	0	g7, g22	
34	g28	and	2	0	0	g14, g22	
35	g29	or	4	0	0	g30a, g31a, g32a, g33a	
36	g30a	and	1	1	1	g30	h1ts
37	g30	or	4	0	0	g7, g14, g22, g34	
38	g31a	and	1	1	1	g31	h2ts
39	g31	or	6	0	0	g38, g39, g40, g41, g42, g43	
40	g32a	and	1	1	1	g32	h3ts
41	g32	or	4	0	0	g44, g45, g46, g47	

Gate No	Gate Name	Gate Type	Input Gate Number	Input Event Number	House Event Number	Input Gate List	Input Event List
42	g33a	and	1	1	1	g33	h4ts
43	g33	and	4	0	0	g7, g14, g22, g34	
44	g34	or	3	0	0	g35, g36, g37	
45	g35	and	0	2	1		hpt1, s4pt1
46	g36	and	0	2	1		hpt2, s4pt2
47	g37	and	0	2	1		hpt3, s4pt3
48	g38	and	2	0	0	g7, g14	
49	g39	and	2	0	0	g7, g22	
50	g40	and	2	0	0	g7, g34	
51	g41	and	2	0	0	g14, g22	
52	g42	and	2	0	0	g14, g34	
53	g43	and	2	0	0	g22, g34	
54	g44	and	3	0	0	g7, g14, g22	
55	g45	and	3	0	0	g7, g14, g34	
56	g46	and	3	0	0	g7, g22, g34	
57	g47	and	3	0	0	g14, g22, g34	
58	sg1	or	0	2	0		ssv1, ssv2
59	sg2	or	2	0	0		sg3, sg4
60	sg3	and	0	2	1		h_old, swov
61	sg4	and	0	2	1		h_new, swnv

APPENDIX E

Publications:

- J. Borisevic, L. M. Bartlett. *Safety System Optimization by Improved Strength Pareto Evolutionary Algorithm (SPEA2)*, proceedings of the 17th AR^2TS, pp. 38-49, 2007.

- J. Borisevic, L. M. Bartlett. *Genetic algorithm based multi-objective optimization of a Firewater Deluge System*, Risk, Reliability and Societal Safety, Aven & Vinnem, Taylor & Francis Group, London, ISBN 978-0-415-44786-7, pp. 107-114, 2007.

- J.Riauke, L. M. Bartlett, *An Offshore Safety System Optimization using a SPEA2 based approach*, Proc. IMechE Vol. 222 Part O: J. Risk and Reliability, JRR113, pp. 271-282, 2008.

- J. Riauke, L. M. Bartlett. *Safety System Design Optimization using Multi-objective Genetic Algorithm,* International Journal of Reliability and Safety (IJRS), Vol. 3, No. 4, pp. 397-412, 2009, DOI: 10.1504/IJRS.2009.028584.

- J. Riauke, L. M. Bartlett. *An Integrated Design optimization Approach for Systems with Dependencies*, proceedings of the 18th AR^2TS, pp. 455-466 , 2009.

- L. M. Bartlett, J. Riauke. *Multi-objective Offshore Safety System Design Optimization*, paper has been submitted for publication in International Journal of Performability Engineering (IJPE).