

This item was submitted to Loughborough's Institutional Repository (<https://dspace.lboro.ac.uk/>) by the author and is made available under the following Creative Commons Licence conditions.



For the full text of this licence, please go to:
<http://creativecommons.org/licenses/by-nc-nd/2.5/>

Automated System Design Optimisation

by

Dovile Astapenko

A Doctoral Thesis

submitted in partial fulfilment of the requirements for the award of
Degree of Doctor of Philosophy of Loughborough University

©by Dovile Astapenko 2010

ABSTRACT

The focus of this thesis is to develop a generic approach for solving reliability design optimisation problems which could be applicable to a diverse range of real engineering systems. The basic problem in optimal reliability design of a system is to explore the means of improving the system reliability within the bounds of available resources. Improving the reliability reduces the likelihood of system failure. The consequences of system failure can vary from minor inconvenience and cost to significant economic loss and personal injury. However any improvements made to the system are subject to the availability of resources, which are very often limited.

The objective of the design optimisation problem analysed in this thesis is to minimise system unavailability (or unreliability if an unrepairable system is analysed) through the manipulation and assessment of all possible design alterations available, which are subject to constraints on resources and/or system performance requirements. This thesis describes a genetic algorithm-based technique developed to solve the optimisation problem. Since an explicit mathematical form can not be formulated to evaluate the objective function, the system unavailability (unreliability) is assessed using the fault tree method. Central to the optimisation algorithm are newly developed fault tree modification patterns (FTMPs). They are employed here to construct one fault tree representing all possible designs investigated, from the initial system design specified along with the design choices. This is then altered to represent the individual designs in question during the optimisation process. Failure probabilities for specified design cases are quantified by employing Binary Decision Diagrams (BDDs).

A computer programme has been developed to automate the application of the optimisation approach to standard engineering safety systems. Its practicality is demonstrated through the consideration of two systems of increasing complexity; first a High Integrity Protection System (HIPS) followed by a Fire Water Deluge System (FWDS). The technique is then further-developed and applied to solve problems of multi-phased mission systems. Two systems are considered; first an unmanned aerial vehicle (UAV) and secondly a military vessel. The final part of this thesis focuses on continuing the development process by adapting the method to solve design optimisation problems for multiple multi-phased mission systems. Its application is demonstrated by considering an advanced UAV system involving multiple multi-phased flight missions.

The applications discussed prove that the technique progressively developed in this thesis enables design optimisation problems to be solved for systems with different levels of complexity. A key contribution of this thesis is the development of a novel generic optimisation technique, embedding newly developed FTMPs, which is capable of optimising the reliability design for potentially any engineering system. Another key and novel contribution of this work is the capability to analyse and provide optimal design solutions for multiple multi-phase mission systems.

Keywords: optimisation, system design, multi-phased mission system, reliability, genetic algorithm, fault tree, binary decision diagram

ACKNOWLEDGMENTS

First and foremost, I wish to express my gratefulness to Dr Lisa Jackson for all of her support throughout the research. I wish to thank her for her invaluable time, supervision and advice with her exceptional insights into Risk and Reliability, for her kind moral support and for nurturing my skills as a researcher.

I also owe my gratitude to Prof. John Andrews for the opportunity to carry out the research presented here, for his time and for his constructive guidance throughout.

Thanks are also paid to the lectures in the Faculty of Fundamental Sciences at the Kaunas University of Technology for sharing their knowledge and helping me to develop lifelong foundations for my future career. I am particularly grateful to Dr Gediminas Dosinas and Dr Vytautas Janilionis for encouraging me to pursue my aspirations to do further research and broaden my horizons.

I wish to extend my warmest thanks to my fellow researchers at Loughborough University for their friendship and for the warm lifelong memories.

I am grateful for my sister Laura, her husband Darius and their daughter Ugnė Kaupai for believing in me and being there for our parents.

Specially to my husband, Pratap, I wish to express my gratitude for his love, dedication and confidence in me, and for reading the last drafts of this thesis.

I am particularly grateful to and humbled by my parents Larisa and Piotras Astapenko who brought me into this world, raised me and love me. Their sacrifices allowed me to pursue my aspiration to walk new paths. To them I dedicate this thesis.

ABBREVIATIONS

AFFF	Aqueous Film-Forming Foam
AR ² TS	Advances in Risk and Reliability Technology Symposium
BDD	Binary Decision Diagram
CPU	Central processing Unit
CW	Circulating Water
ESD	Emergency Shutdown
ESDV	Emergency Shutdown Valve
ESREL	The European Safety and Reliability Conference
FWDS	Firewater Deluge System
FTMP	Fault Tree Modification Pattern
GA	Genetic Algorithms
GSDOA	General System Design Optimisation Algorithm
GSDOP	General System Design Optimisation Programme
HIPS	High Integrity Protection System
IP	integer programming
ite	If-Then-Else
LP	Linear Programming
MDT	Maintenance Down Time
MFGP	Main Fire and Gas Panel
MG	Motor Generator
MILP	Mixed Integer Linear Programming
MINLP	Mixed Integer Nonlinear Programming
MOGA	Multi Objective Genetic Algorithm
MPMSDOA	Multiple Phased Missions System Design Optimisation Algorithm
MPMSDOP	Multiple Phased Missions System Design Optimisation Programme
NLP	Nonlinear Programming
NPGA	Niched Pareto Genetic Algorithm
NSGA	Non-Dominated Sorting Genetic Algorithm
PAES	Pareto-Achieved Evolution Strategy
PMSDOA	Phased Mission System Design Optimisation Algorithm
PMPSDOP	Phased Mission System Design Optimisation Programme
PSO	Particle Swarm Optimisation
PT	Pressure Transmitter
RAM	Random Accessible Memmory
RDGA	Rank-Density Based Genetic Algorithm
RS	Random Search
RWGA	Random Weighted Genetic algorithm
SA	Simulated Annealing
SOGA	Single Objective Genetic Algorithm
SPEA	Strength Pareto Evolutionary Algorithm
TS	Tabu Search
UAV	Unmanned Aerial Vehicle
VEGA	Vector Evaluated Genetic Algorithm
VFR	Voltage Frequency Regulator

NOMENCLATURE

$A(t)$	availability
$C(i)$	offspring population in generation i
C_I	initial design cost
C_S	storage cost
$Cost_D$	total system design cost
$Cost_M$	cost assigned for system maintenance
$Cost_{Sys}$	total system cost
c_{p_i}	cost of a single preventive maintenance activity for component i
c_{r_i}	cost of a single repair for component i
c_{t_i}	cost of a single maintenance test for component i
$cost_{d_i}$	design cost of a component i
$cost_{t_i}$	cost of maintenance testing for component i .
$cost_{p_i}$	preventive maintenance cost of the component i .
$cost_{c_i}$	corrective maintenance cost of the component i .
$C(P_1^*, P_2^*)$	function (measure) C
d_i	minimum value of the sum of the absolute differences in the values of objective functions between the i -th solution and any other solution in the P^*
$d_i(\mathbf{x}, B)$	magnitude of violation of a given constraint i for solution \mathbf{x}
\bar{d}	mean value of the distances d_i
$E[x_i]$	probability that event i has occurred
F^i, F_i	objective function value of the individual i
F_{all}	the best unpenalised value of the objective function yet found
F_{feas}	the best feasible value of the objective function yet found
$F_{fitness}^i$	fitness value of the individual i
F_j	the logical expression for the top event of the fault tree to occur in phase j
F_p^i	penalty value for an infeasible individual i
$F(x)$	objective function
$F(t)$	unreliability function
f_i	objective function value of the individual i
$f1, f2$	Boolean functions
$f(x)$	function of a variable x , structure function
G_i	gate number
GD	General Distance metric
$g(x)$	function of a variable x (inequality constraint)
H_T	number of hours of manual work required to test the component
$h(x)$	function of a variable x (equality constraint)
$ICH(i)$	value of the i th digit in a binary string
m	total number of components representing the system design case
mk	maximum number of redundant components required for successful operation, i.e. $mk \leq mn$

mn	maximum possible number of redundant components
mt	tmaximum number of possible different component types
nb	number of bits in a binary string
nc	total number of constraints set for the problem
N	number of chromosomes in a population
N_C	total number of minimal cuts set
N_S	Total number of spare stored
N_θ	number of different maintenance test intervals
N_{c_i}	number of system components which are tested at the same time interval
NFT_i	near-feasibility threshold that corresponds to a given constraint i
P_T	true Pareto optimal set of the problem.
\mathbf{Ph}_j	mission failure in phase j
$P(C_i)$	probability that a minimal cut set i exists
$P(i)$	parent populations in the generation i
$P(PFC_i)$	probability of phase failure combinations for phase i
P^*	non-dominated set of solutions, Pareto-optimal set of solutions
p_i	cost of a single preventive maintenance for the component i
Q_{sys}	system unavailability
$Q(t)$	unavailability function
$Q_{AV}(t)$	average unavailability
q_i	unavailability value of component i .
$R(t)$	Reliability function
r_j	a non-negative random number
S	Spacing metric
T_j	test time for each system component.
t	time
U_T	total number of time units per examined time period,
v_i	volume parameter calculated for component i
$W_i,$	number of expected failures of component i .
w_i	weight of component i
w_i	a constant weight for function $f_i(\mathbf{x})$
\mathbf{X}	vector of the decision variables
$x_k(t_i, t_j)$	variable indicating component k failure in time interval (t_i, t_j)
β_i	Weibull distribution parameter
η_i	Weibull distribution parameter
θ	time between inspections/maintenance activities
κ_i	user-specified severity parameter
λ	constant conditional failure rate
τ	mean time to repair
ν	repair rate

CONTENT

- 1. INTRODUCTION.....1**
 - 1.1. Introduction to Reliability and Risk Analysis.....1
 - 1.2. System Reliability Modeling.....2
 - 1.3. System Design and reliability.....3
 - 1.4. The Optimal Reliability Design Problem.....4
 - 1.5. Objectives of the Research.....4

- 2. FAULT TREE AND BINARY DECISION DIAGRAM ANALYSIS.....7**
 - 2.1. Introduction.....7
 - 2.2. Fault Tree Analysis.....8
 - 2.2.1. Fault Tree Construction.....8
 - 2.2.2. Qualitative Analysis.....10
 - 2.2.3. Minimal Cut Set Algorithms.....11
 - 2.2.4. Quantitative Analysis.....13
 - 2.2.4.1. Parameters of Component Performance.....13
 - 2.2.4.2. Maintenance Policies.....14
 - 2.2.5. Assessment of System Performance.....15
 - 2.3. Binary Decision Diagrams.....16
 - 2.3.1. BDD Architecture.....16
 - 2.3.2. BDD Construction.....17
 - 2.3.3. Reduction.....20
 - 2.3.4. Quantitative BDD Analysis.....21
 - 2.4. Summary.....23

- 3. TECHNIQUES FOR DESIGN OPTIMISATION IN RELIABILITY ENGINEERING.....25**
 - 3.1. Introduction.....25
 - 3.2. Linear Programming.....26
 - 3.3. Nonlinear Programming.....27
 - 3.4. Discrete Optimisation.....28
 - 3.5. Meta-heuristic Methods.....30

3.5.1.	Simulated Annealing.....	31
3.5.2.	Tabu Search.....	32
3.5.3.	Particle Swarm Optimization.....	33
3.5.4.	Genetic Algorithms.....	34
3.5.4.1.	Advantages and Disadvantages of a GA.....	37
3.5.4.2.	Fundamentals of the GA.....	37
3.6.	Summary.....	44
4.	GENERAL SYSTEM DESIGN OPTIMISATION ALGORITHM.....	46
4.1.	Introduction.....	46
4.2.	Fault Tree Modification Methodology.....	47
4.2.1.	Design Alteration Options.....	48
4.2.2.	Fault Tree Modification Patterns.....	49
4.2.2.1.	Overview.....	49
4.2.2.2.	Parallel Redundant Elements (Pattern 1).....	51
4.2.2.3.	<i>k-out-of-n</i> Redundancy (Pattern 2).....	53
4.2.2.4.	A Different Component Type Selection (Pattern 3).....	55
4.2.2.5.	Selection Option of a Component Type for New Redundant Components (<i>Pattern4</i> and <i>Pattern5</i>).....	56
4.2.2.6.	Fault Tree Alteration at Gate Level.....	59
4.2.3.	Quantitative Fault Tree Analysis.....	61
4.3.	Quantitative Fault Tree Analysis	
4.4.	Optimisation Algorithm.....	61
4.4.1.	Mathematical Problem Concept.....	61
4.4.2.	Evaluation of Design Requirements.....	63
4.4.3.	Genetic Algorithm Characterisation.....	66
4.4.3.1.	Chromosome Encoding.....	66
4.4.3.2.	Population Manipulation.....	68
4.4.3.3.	Replacement.....	69
4.4.4.	Optimisation Algorithm Structure.....	70
4.4.4.1.	Preparative Part.....	70
4.4.4.2.	Optimisation Part.....	71
4.4.5.	Computer Implementation of the GSDOA - GSDOP.....	73
4.4.5.1.	Initialisation Stage.....	73
4.4.5.2.	Construction of the Fault Tree for all Possible Design Alternatives.....	74

4.4.5.3. Optimisation Part	76
4.5. Summary	80
5. HIGH INTEGRITY PROTECTION SYSTEM DESIGN OPTIMISATION USING <i>GSDOA</i>	82
5.1. Introduction	82
5.2. High Integrity Protection System Description	82
5.3. Options for HIPS Design Alterations	85
5.4. HIPS Design Optimisation Problem	87
5.4.1. Data Initialisation	87
5.4.2. Fault Tree Construction	92
5.4.3. Chromosome Structure	96
5.5. Analysis of <i>GSDOP</i> Performance Solving the HIPS Design Optimisation Problem	98
5.5.1. Selection of the Values of the GA Parameters	98
5.5.2. Testing	104
5.6. Summary	108
6. FIREWATER DELUGE SYSTEM DESIGN OPTIMISATION USING IMPROVED <i>GSDOA</i>	110
6.1. Introduction	110
6.2. Description of FireWater Deluge System	110
6.2.1. Performance Principles	110
6.2.2. Firewater Deluge System Failure	113
6.3. Design Alteration Options	118
6.4. Data Arrangement for the Optimisation	120
6.5. Modification of the <i>GSDOA</i>	124
6.5.1. Penalty Function Method and New Replacement Procedure	125
6.5.2. Fitness Scaling	128
6.5.3. Structure of the Improved SOGA	131
6.6. Analysis of the Improved <i>GSDOP</i> Performance Solving the FWDS Design Optimisation Problem	132
6.6.1. Analysis of GA Parameters Influence on Algorithm Performance	132
6.6.2. Testing	135
6.7. Summary	138

7. MULTI-PHASED MISSION SYSTEM DESIGN OPTIMISATION	141
7.1. Introduction.....	141
7.2. Methods for Reliability Analysis of Phased Mission Systems.....	142
7.2.1. Overview.....	142
7.2.2. Non-Repairable Phased Missions.....	143
7.3. Phased Mission System Design Optimisation Algorithm (<i>PMSDOA</i>).....	150
7.3.1. Introduction.....	150
7.3.2. Fault Trees for all Possible Design Alternatives.....	152
7.3.3. Evaluation of Phased Mission System Failure Probability.....	153
7.3.4. Mathematical Representation of the Problem (Overall Mission Constraints).....	154
7.3.5. Development of Phased Mission System Design Optimisation Programme (<i>PMSDOP</i>).....	156
7.3.5.1. Data Processing.....	157
7.3.5.2. Preparation for Quantitative Analysis.....	158
7.3.5.3. Optimisation Algorithm.....	158
7.4. UAV Design Optimisation Using the Phased Mission Design Optimisation Algorithm.....	159
7.4.1. Introduction to a Phased Mission of an UAV.....	159
7.4.2. UAV Design Alternatives.....	163
7.4.3. Analysis of Optimisation Results.....	164
7.4.4. Summary of the Analysis.....	167
7.5. Military Vessel Design Optimisation Using <i>PMSDOP</i> (Case 1).....	167
7.5.1. Initial Military Vessel Design.....	168
7.5.2. Military Vessel Design Alternatives.....	172
7.5.3. Analysis of Optimisation Results.....	174
7.5.4. Summary of the Analysis.....	176
7.6. System Design Limitations for Individual Phases.....	177
7.7. Military Vessel Design Optimisation Problem with Constraints Added at Each Phase (Case 2).....	180
7.7.1. Analysis of Optimisation Results.....	182
7.7.2. Summary of the Analysis.....	182
7.8. Summary.....	183
8. DESIGN OPTIMISATION OF MULTI-PHASED MISSION SYSTEM CONSIDERING MULTIPLE MISSIONS	185

8.1.	Introduction.....	185
8.2.	Multi-Objective Optimisation.....	185
8.3.	Multi-Objective Optimisation Techniques.....	188
8.3.1.	Handling of Constraints.....	190
8.3.2.	Performance Metrics.....	190
8.3.3.	Proposed Multi-Objective Optimisation Technique.....	192
8.4.	Multiple Phased Missions System Design Optimisation Algorithm (<i>MPMSDOA</i>).....	195
8.4.1.	Mathematical Representation of the Problem.....	195
8.4.2.	Algorithm Particulars.....	196
8.5.	UAV Design Optimisation Using the <i>MPMSDOA</i>	198
8.5.1.	Problem Overview.....	198
8.5.2.	Analysis of Optimisation Results.....	200
8.5.3.	Summary of the Optimisation Analysis.....	203
8.6.	Summary.....	203
9.	CONCLUSIONS AND FUTURE WORK.....	205
9.1.	Summary.....	205
9.2.	Conclusions.....	208
9.3.	Future Work.....	209
	BIBLIOGRAPHY.....	211
	PUBLICATIONS.....	218
	APPENDIX 1.....	219
	APPENDIX 2.....	228
	APPENDIX 3.....	234
	APPENDIX 4.....	249

1. INTRODUCTION

1.1. INTRODUCTION TO RELIABILITY AND RISK ANALYSIS

Risk and reliability analysis of systems and their components is an integral part of modern equipment design. Furthermore every engineering project, contract and piece of equipment requires this discipline by law. Reliability and risk analysis has a potentially wide range of application areas. The developed methods have been adopted for safety cases in the nuclear, chemical and offshore industries. They are applied to assess the safety and reduce the hazards of systems in defence, marine and automotive industry areas. Risk and reliability techniques also find applications in production and maintenance studies during the design phase of new plants to improve their availability and profitability.

Risk and reliability is a relatively new field. Its conception has been developed primarily due to the complexity, sophistication, and automation inherent in modern technology. Reliability engineering emerged in the late 1940s and early 1950s when the problems of maintenance, repair, and field failures became severe for military equipment used. Much of the early work was confined to the analysis of performance aspects of systems in transportation and communication. In the early 1970s methods were developed for identifying hazards and for quantifying the consequences of failures. Over recent years major accidents such as Flixborough, Seveso, Piper Alpha and the Clapham rail incident increased the concern about risk associated with operating large plants. It prompted the focus in the use of the risk and reliability methods in the field of hazard assessment. Thus, engineers are working to maximise the benefits of modern processing technology ensuring its availability while reducing the safety risks to acceptable levels.

Reliability has two connotations. One is probabilistic in nature; the other is deterministic. The most widely accepted definition of reliability is the ability of an item to operate under specified operating conditions for an assigned period of time. Considering the probabilistic approach the ability of an item can be designated through a probability. Thus, reliability theory is concerned with predicting the probability of survival of a component or system performing its prescribed function during a given lifetime.

Risk can be formally defined as the potential of loss or injury resulting from exposure to a hazard. Quantitative risk analysis involves estimation of the degree of loss or the probability

that a component or system will fail to perform its function which results in a hazard occurring. Thus, reliability and risk are related to one another.

1.2. SYSTEM RELIABILITY MODELING

The main concern in reliability engineering is to identify potential failures of systems and prevent these failures from occurring. A system comprises a number of subsystems and components which are interconnected. It is important to model the reliability of individual items as well as the relationship between various items to determine the reliability of the system. One of the most important aspects of reliability analysis is the assessment of system reliability through the analysis of its constituent elements.

Various probabilistic methods are employed in system reliability modelling. Available methods can be broadly classified into inductive and deductive techniques. Using inductive techniques failure modes are identified at the component level first. Next, the effect of each component on the overall system is established. In the deductive techniques, the analysis starts with the identification of the potential system failure mode and works down through the system to identify possible causes of the hazard. In this research project two methodologies have been used for system reliability analysis: fault tree analysis (FTA) and binary decision diagram (BDD) method.

The FTA is a deductive technique. An undesirable event, called the top event, is postulated and the possible means for this event to occur are systematically deduced using a logic diagram, a fault tree. The deduction process is performed so that the fault tree embodies all component failures that contribute to the occurrence of the top event. It is also possible to include environmental conditions, human errors as well as specific component failures during the system operation.

A FTA may be qualitative, quantitative, or both. The analysis may provide a listing of the possible combinations of environmental factors, human errors and component failures that may result in the critical system condition. The probability that the critical event will occur during a specified time interval can also be determined.

Over recent years attention has been given to the development and use of the BDD method in system reliability analysis. A BDD is a directed acyclic graph, where all path through the BDD start at the root vertex and terminate in one of two states – a 1-state (system failure), or a 0-state (system success). BDDs provide an alternative approach to fault trees to represent

the failure logic of a system. The main benefit is that the method improves the efficiency and accuracy of the fault tree analysis procedure.

1.3. SYSTEM DESIGN AND RELIABILITY

During the design phase reliability engineering can have the greatest effect for enhancing the system's safety and reliability. The typical design is an iterative process and several trial systems are analysed in sequence before an acceptable design is obtained. The process begins with the identification of a need and the definition of a specification for the system. The conceptual design stage is the best time to incorporate reliability and also maintainability considerations. The second step of the process is to define a preliminary design of the system. The third step is to carry out a detailed design and analysis for all subsystems. System testing and evaluation follows next. However, it may not be the last step in the design process, because during testing and evaluation it may be revealed that the system performance criteria has not been met, or the reliability level may not be satisfactory, or that constraints are not satisfied. In fact, re-examination may be necessary at any step of the design process.

One of the goals of reliability engineering is to build high reliability into the system through careful design and analysis within the limits of constraints imposed on resources, which can be economic and physical. Some of the means through which system reliability can be enhanced are:

- reducing the system complexity;
- increasing the reliability of constituent components in the system;
- use of structural redundancy;
- putting in practice repair/preventive maintenance;
- decreasing the downtime of the system;

Implementation of the listed means requires resource consumption. Therefore during the design process the balance between reliability and resource consumption is essential. As modern systems are becoming more and more complicated it is highly unlikely that the trade-offs within the available resources, such that the optimal system reliability is achieved, can be made manually. For this reason an optimisation algorithm integrated within the design process is greatly needed.

1.4. THE OPTIMAL RELIABILITY DESIGN PROBLEM

System reliability is important in any system design. An optimal reliability design is such where the reliability of the system has been enhanced through all possible means available with minimum cost and under other constraints imposed on the development of a system.

A reliability design problem can be formulated as an optimisation problem where the objective may be the maximisation of system reliability/availability, minimisation of system unreliability/unavailability, minimisation of downtime or the number of failures, or minimisation of the overall cost associated with the system. In the literature, reliability optimisation problems are classified according to the types of their decision variables as redundancy allocation problems, reliability allocation problems, reliability-redundancy allocation problems and component assignment problems. The type of reliability optimisation problem determines the nature of decision variables, objective function and constraints considered. Thus design parameters, i.e. decision variables, may include the number of redundancies, component reliability value or arrangement of unknown components. The constraints may include budget restrictions, reliability requirements or design considerations such as volume and weight. One more of these criteria may be included in the objective function, while the others may be treated as constraints. Optimal system reliability design involves identifying objective functions as well as decision variables and constraints.

A wide range of mathematical optimisation methods exist, such as linear programming, nonlinear programming and evolutionary algorithms. However, the features offered by some of the methods make them inappropriate for real world system design optimisation problems. The majority of engineering system optimisation problems involve objective functions and constraints that are too complicated to manipulate with standard approaches (for example, linear programming optimization techniques). A genetic algorithm (GA), one of the most popular evolutionary algorithms, has been chosen for finding the optimal solutions for the reliability design optimisation problems considered in this research. The algorithm has the capability to solve complex and large scale optimisation problems with any kind of non-linear objective functions and constraints defined in discrete, continuous or mixed search spaces.

1.5. OBJECTIVES OF THE RESEARCH

One part of the reliability design optimisation process is the evaluation of design proposals, i.e. the objective function. The second part is the generation of new, and hopefully, better

designs. One of the challenges when solving an optimal reliability design problem is computing the objective function. Obtaining a closed-form mathematical expression for the objective function may be particularly difficult. In such cases the optimisation methods employed should not depend on the form of the objective function, which limits the number of optimisation techniques capable to solve reliability design optimisation problems.

The majority of the methods used for reliability design optimisation problems consider simple or well-structured systems. However, real world systems are usually very complex and analysing its simplified structure in order to apply such methods may compromise the accuracy of the results. Other techniques combine additional means for performance evaluation of design proposals, such as fault tree analysis, with an optimisation technique. However these techniques were developed for specific examples and have not been applied for a general case system.

This thesis is concerned with the development of a generic optimisation approach that can solve complex engineering system design optimisation problems and can be applicable to a diverse range of systems. The new approach developed will combine system reliability modelling techniques for the introduction and evaluation of design proposals and the optimisation technique for analysis and generation of solutions.

The first type of systems to be considered is general engineering systems. The second type of systems analysed is phased mission systems. A large number of systems which can employ different technologies such as electronic, nuclear and chemical can be analysed as phased mission systems. However, there is no demonstrated evidence in the literature for research that focuses on such phased mission system design optimisation problems. The third group of systems considered are systems designed for multiple phased missions. For these systems equipment can be designed for varying conditions (military tanks) or for varying operations (warfare).

The objectives of the research are to:

1. Review existing risk and reliability assessment methods, which can be used to assess the performance of different system designs. Identify optimisation techniques used to solve reliability design optimisation problems and provide an appropriate critical review with regards to their application to solve such problems. Following the review the main risk and reliability assessment methods and the optimisation technique which would be employed in the developed approach will be identified.

2. Develop a problem-independent genetic algorithm based optimisation technique and its computer code. The technique will be used to construct an optimal system design case with the aim of minimising its unavailability and at the same time ensuring optimal usage of available resources. The developed code will automate the optimisation process.
3. Demonstrate a systematic approach to system design by applying the technique developed in (2) to a number of safety systems. The application examples will demonstrate scalability and the potential of the technique to be applicable to a range of different systems. Critical appraisal of the application results will highlight the algorithm deficiencies that may result in modifications of the initial approach.
4. Develop a general genetic algorithm based optimisation technique and its computer code for multi-phased missions analysis. The optimisation technique will define the optimal phased mission system design with the aim to minimise the overall mission failure probability and to spend available resources for the best use.
5. Utilise the technique derived in (4) to demonstrate its potential applicability to different systems and scalability to problems with a diverse degree of complexity.
6. Develop and apply the general genetic algorithm based optimisation approach and its code to solve the design optimisation problems of phased mission systems involved in multiple missions. The approach will be used to construct an optimal design for the phased mission systems considering their performance and optimal usage of available resources throughout multiple missions.

2. FAULT TREE AND BINARY DECISION DIAGRAM ANALYSIS

2.1. INTRODUCTION

FTA is one of the engineering approaches used for systems safety and operational analysis [1]. It is a deductive methodology that specifies a system failure mode through logic statements of possible causes of system failure. FTA provides a schematic description of the possible combinations of system conditions that could lead to its failure. It can be both a design tool that identifies probable accidents in a system design and a diagnostic tool for prediction of hazardous causes of system failure. FTA involves two major steps: construction of the fault tree and its evaluation [2]. In the current research project FTA is used to represent system designs through the analysis of its causes of failure.

Fault trees are not, however, an ideal form for mathematical analysis. Quantitative fault tree analysis for complex systems can be computationally very extensive. As an alternative, BDDs, which represent the logic of system failure, are easier to manipulate than the fault tree for an exact quantitative assessment. BDDs were first introduced by Lee [3] for the representation of switching circuits. Later Akers [4] derived the basic BDD methodology aiming to define a diagram for a digital function which determines the output values of the function by examining the values of the inputs. Bryant [5] extended the BDD application to Boolean functions with further restrictions on the ordering of variables in the diagrams. The method for reliability analysis based on the BDD was introduced by Rauzy [6]. The proposed method improved the efficiency of analysis for the introduced industrial systems. In the current research project the BDD technique is used for the quantitative analysis of fault trees representing system designs.

In this chapter FTA is discussed in *Section 2.2*. Fault tree construction is described in *Section 2.2.1*. Qualitative and quantitative FTA is detailed in *Sections 2.2.2* and *2.2.3* respectively. *Section 2.3* discusses the BDD methodology. First, the BDD architecture is described in *Section 2.3.1*. BDD construction and reduction procedures are detailed in *Sections 2.3.2* and *2.3.3* respectively. Finally, qualitative and quantitative BDD analysis is discussed in *Sections 2.3.4* and *2.3.5* respectively.

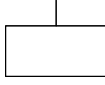


2.2. FAULT TREE ANALYSIS

2.2.1. Fault Tree Construction

Construction of a system fault tree is based on the knowledge of its design and operation [2]. It is a process of developing a tree of logical relationships among possible events that result in a specific system failure that is called a *top event*. The term *event* defines a dynamic change of state of a system element. In addition such factors as human actions or environmental characteristics can also be identified as events. Logical relationships among events are defined using *gates*.




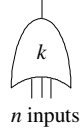
Thus a fault tree is a hierarchical diagram consisting of events and gates. The symbols representing different types of fault tree events are shown in *Table 2.1*. The rectangle defines the top event or an intermediate event that is the output of a logic gate. The circle represents a basic failure of a system element and is an input to a logic gate. The house event is an event that is expected either to occur or does not occur, i.e. it can be in a *TRUE* or *FALSE* state.

Table 2.1. Event Symbols

Event Symbol	Meaning of Symbol
	Top event/ intermediate event
	Basic event
	House event

The fundamental logic gates used in fault tree construction include the *AND* and the *OR* gate. An output event of the *AND* gate occurs if and only if all input events are present simultaneously. The *OR* gate describes the existence of an output event when at least one of input events exist. Other frequently used gates are the *NOT* gate and the *k/n* vote. The *NOT* gate describes the logic of occurrence of an output event when none of the input events occur. Finally, the *k/n* vote gate provides an output event if at least *k* out of *n* inputs occur. The *k/n* vote gate can be transformed to a branch of a fault tree using *OR* and *AND* gates. The symbols for the gates are presented in *Table 2.2*.

Table 2.2. Gate Symbols

Gate Symbol	Gate Name	Causal Relation
	AND	Output event results if all input event occur simultaneously
	OR	Output event results if one or more input events occur
	NOT	Output event result if the input event does not occur
	k/n vote	Output event results if at least k out of n input event occur.

A house event is a special type of event employed for specific use within a fault tree analysis. It can be turned on or off to specify the conditions present under a specific scenario. When a house event is turned on (TRUE state) the event is presumed to have occurred and the probability of that event is set to 1. When a house event is turned off (FALSE state) it is presumed that the event has not occurred, and the probability is set to 0. If a house event is turned on the gate that the house event inputs to is calculated normally. By turning a house event off, the gate that the house event inputs to can be removed from the tree.

In current research house events are utilised to make parts of a fault tree functional or non-functional when considering different system design options. For example, consider a water deluge system that can fail if either its pump fails or the nozzle is blocked. The system can have a pump fitted that can be chosen out of two samples – type 1 or type 2. Both scenarios can be represented in one fault tree using two house events as shown in *Figure 2.1a*. By turning one house event off the resulting fault tree will represent the failure causes of the deluge system with the specified pump sample. *Figure 2.1 b*) shows the case when house event PT1 has been turned off (a Type 1 pump has not been fitted) and house event PT2 set to TRUE (Type 2 is fitted).

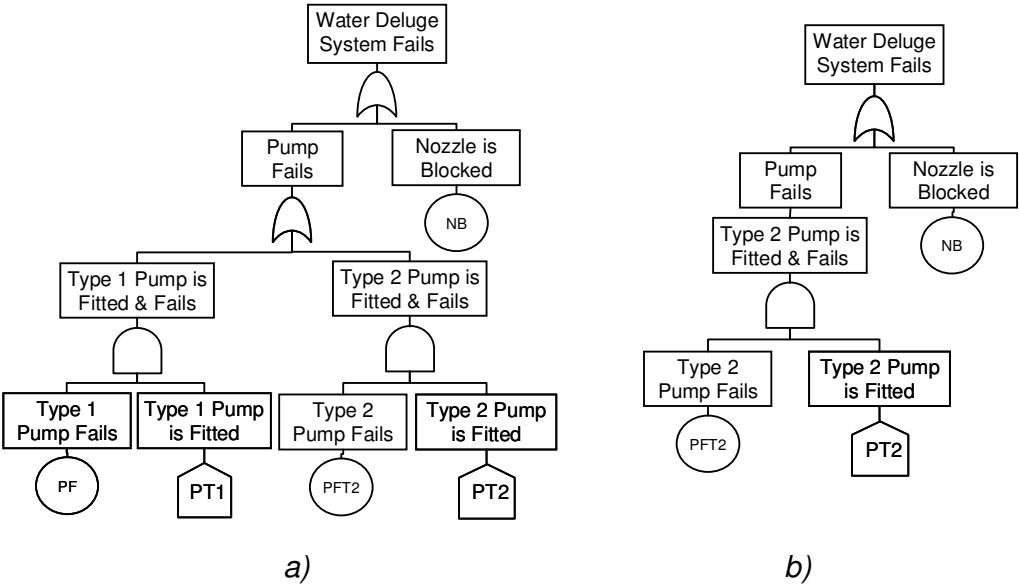


Figure 2.1. Fault Trees with House Event(s)

A constructed fault tree provides a convenient format of the possible causes of a specific system failure. Evaluation of the fault tree can be performed qualitatively and quantitatively [2].

2.2.2. Qualitative Analysis

Qualitative fault tree analysis involves the identification of the causes of a specific system failure which can occur due to failure of an individual system element or a combination of failures of system elements. A set of basic events whose existence cause the top event to occur is called a *cut set*. For example, a fault tree contains four basic events A, B, C and D. Failure of components A and B or C and D can lead to the top event as shown in Figure 2.2. As a result seven cut sets can be identified for this case: {A, B}, {C, D}, {A, B, C}, {A, B, D}, {A, C, D}, {B, C, D}, {A, B, C, D}.

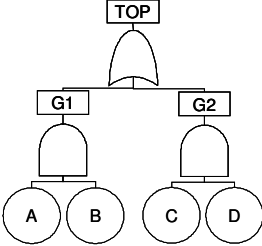


Figure 2.2. Example Fault Tree

For larger systems the number of cut sets can dramatically increase. Therefore in order to simplify the analysis consideration is usually given only to those cut sets that include the smallest combination of basic events. From the given example it would be cut sets {A, B} and {C, D}. The cut set {A, B, C} would not be considered. The cut set includes event C and its state is irrelevant to system failure since the system fails due to the occurrence of A and B. A cut set that can not be reduced and still ensures the existence of the top event is called a *minimal cut set*.

The aim of qualitative analysis is to produce a list of minimal cut sets for a given top event. Since each minimal cut set consists of a combination of component failures relevant to the given top event, the list of minimal cut sets is unique to this top event. A number of basic events in a minimal cut set defines its order. A one-component minimal cut set is called a first-order minimal cut set, a two-component set defines second-order minimal cut set and *etc.* In general, lower order minimal cut sets identify major contributors to system failure.

The top event can be represented in terms of a Boolean equation where fault tree events are logic variables and the final result is a sum of products (s-o-p). This transformation provides the list of all minimal cut sets relevant to the top event. One way of obtaining the s-o-p is using the top-down approach. However in some cases the resulting s-o-p may not be minimal and so the minimal cut sets cannot be obtained directly. Boolean reduction rules [7] need to be applied to allow the extraction of the minimal cut sets.

2.2.3. Minimal Cut Set Algorithms

The top-down approach mentioned earlier for obtaining minimal cut sets is based on logic Boolean operations together with substitution, expansion and reduction methods. Here each AND gate implements the logical AND function or intersection and is represented by the product (.). The OR gate is the logical OR function or union and is denoted by a sum sign (+). Correspondingly the NOT gate is the logical NOT function or inverter which is denoted by a horizontal bar over the variable to be inverted. Input events of each gate are variables of the corresponding logic function.

The top-down approach is started by deriving a logic expression for the top event. The listed Boolean events in the expression are then expanded by substituting in the logic expressions appearing one level lower in the tree. To simplify the expansion process Boolean reduction rules can be applied where necessary. The process is continued until the expression remaining

has only basic events. If the resulting s-o-p expression is not minimal then Boolean Laws of Algebra are applied. The obtained products are the minimal cut sets for the fault tree.

As an example consider the fault tree in *Figure 2.3*.

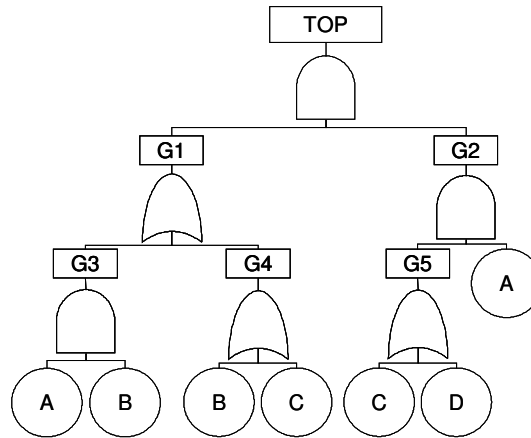


Figure 2.3. Example Fault Tree

The logical function for the top event *TOP* comprises of two Boolean events:

$$TOP = G1.G2$$

As *G1* and *G2* are gates they can be expanded into the following logic expressions:

$$G1 = G3 + G4, G2 = G5.A$$

Similarly expressions are substituted for *G3* and *G4* that results in the full expansion of gate *G1*:

$$G1 = A.B + B + C.$$

The obtained expression is simplified and the number of terms in the equation is reduced. The Absorption Law is used:

$$G1 = B + C$$

After the second Boolean event of the top logic function (*G2*) is fully expanded and Boolean reductions rules are applied the *TOP* equation is reduced:

$$TOP = A.B.D + A.C$$

This is the minimal form of the s-o-p. Each term is a minimal cut set. Thus for the given example fault tree two minimal cut sets exist. One cut set is of order two, i.e. contains two elements, {A, C} and the other one is of order three {A, B, D}.

Although the presented algorithm is not complex, the task to obtain the minimal cut sets for larger systems can become computationally intensive. There are a number of alternative techniques introduced to obtain minimal cut sets in a more efficient manner. For example, computation time and the memory required for minimal cut set generation can be reduced by employing BDDs [6], [8] since they provide a faster means of analysing fault trees. The BDDs will be discussed in detail in *Section 2.3*.

2.2.4. Quantitative Analysis

The aim of the quantitative analysis of the fault tree is to quantify a number of parameters in order to assess the system performance. It is used to calculate the probability and frequency of occurrence for the top event. It may also provide importance measures which indicate the contribution of specific basic events and their groups to the top event.

To quantify performance measures for a system, it is necessary to have the corresponding information for its components. The means for quantification of component performance are discussed in the following *Sub-section 2.2.4.1*

2.2.4.1. Parameters of Component Performance

A number of characteristics can be employed to describe component and system performance. The most useful ones include a measure of time to first failure, i.e. reliability, and a measure of expected up-time, i.e. availability.

For components that can be repaired, and so for which failure can be tolerated, a relevant measure of performance is availability. *Availability*, $A(t)$, is defined as the probability that a component is functioning at a given point in time. It is the fraction of the total time that a component is able to perform the required function. The complement of availability is *unavailability*, $Q(t)$ (*Equation 2.1*). It is the probability that a component is failed at time t .

$$Q(t) = 1 - A(t) \quad (2.1)$$

Reliability, $R(t)$, is a relevant measure for components where failure cannot be tolerated, and so the successful operation of the component over a stated period of time is an important performance measure. It is the probability that a component will operate without failure for a

stated period of time under specified conditions. The probability that a component fails to work continuously over a stated time interval under specified conditions is known as its unreliability, $F(t)$, where:

$$F(t) = 1 - R(t) \quad (2.2)$$

2.2.4.2. Maintenance Policies

Unavailability of a component is also influenced by the way it is maintained. There are three basic types of maintenance repair policies [7]:

1. No repair.
2. Unscheduled maintenance; repair is initiated when failure is revealed.
3. Scheduled maintenance; repair is initiated when failure is discovered.

If a component is unrepairable, for example an aircraft part whilst in flight, then its unavailability is equal to its unreliability. If it is considered that a components failure rate is constant then the unavailability is:

$$Q(t) = F(t) = 1 - e^{-\lambda t}, \quad (2.3)$$

where λ is a constant conditional failure rate, a measure of the rate at which failures occur.

If a component undergoes unscheduled maintenance its repair is carried out when a failure is revealed. Since the failure is immediately known the time to fix the failure includes repair time only and no detection time is included. For constant failure and repair rates it can be shown that the unavailability of the component is given by:

$$Q(t) = \frac{\lambda}{\lambda + \nu} \left(1 - e^{-(\lambda + \nu)t} \right) \quad (2.4)$$

Here ν is a repair rate which is assumed to be constant.

The third type, i.e. scheduled maintenance is common for systems which are not continuously operating, for example, protection systems. In this case a component failure will be revealed if a system is maintained or it is in operation. The time that a component is in the failed state will include the time it takes to identify the occurrence of a failure and the time needed to repair the component. If again it is considered that the failure rate is constant, θ is the time

between inspections and it is assumed that between inspections the component is effectively non-repairable then the average component unavailability is:

$$Q_{AV} = \frac{1}{\theta} \int_0^{\theta} 1 - e^{-\lambda t} dt = \frac{1}{\theta} \left[t + \frac{e^{-\lambda t}}{\lambda} \right]_0^{\theta} = 1 - \frac{1}{\lambda \theta} (1 - e^{-\lambda \theta}) \quad (2.5)$$

The average unavailability for components that undergo scheduled maintenance can be approximated as follows:

$$Q_{AV} = \lambda \left(\frac{\theta}{2} + \tau \right), \quad (2.6)$$

where τ is the mean time to repair.

2.2.5. Assessment of System Performance

The fault tree is drawn for a particular failure mode of the system. Therefore the probability of occurrence of the top event is the probability of that failure. The general method which gives the exact probability of the top event existence is based on minimal cut sets and the Inclusion-exclusion principle. Thus the top event probability (system failure probability) is given by:

$$Q_{sys}(t) = P \left(\bigcup_{i=1}^{N_c} C_i \right) \quad (2.7)$$

where $P(C_i)$ is the probability that a minimal cut set i exists, N_c is the total number of minimal cuts set in the fault tree that can not be smaller than one, i.e. at least one minimal cut set has to exist.

Expanding Equation 2.7 gives the Inclusion-exclusion formula:

$$Q_{sys}(t) = \sum_{i=1}^{N_c} P(C_i) - \sum_{i=2}^{N_c} \sum_{j=1}^{i-1} P(C_i \cap C_j) + \dots + (-1)^{N_c-1} P(C_1 \cap C_2 \cap \dots \cap C_{N_c}) \quad (2.8)$$

For example, consider the example fault tree presented in *Figure 2.3*. The fault tree has two minimal cut set $C_1 = \{A, C\}$ and $C_2 = \{A, B, D\}$. The expression for the calculation of the top event probability when using the inclusion-exclusion principle is obtained as follows:

$$\begin{aligned} Q_{sys}(t) &= P(C_1) + P(C_2) - P(C_1) \cdot P(C_2) = P(A.C) + P(A.B.D) - P(A.C.A.B.D) = \\ &= P(A) \cdot P(C) + P(A) \cdot P(B) \cdot P(D) - P(A) \cdot P(C) \cdot P(B) \cdot P(D) \end{aligned}$$

The increasing fault tree complexity results in more and more intensive computation in order to evaluate the top event probability when using the inclusion-exclusion principle. For exceedingly large problems a solution for the intensive computation problem could be the use of approximations. Upper and Lower Bounds for system unavailability and the Minimal Cut Set Upper Bound are the most commonly used approximation methods.

2.3. BINARY DECISION DIAGRAMS

2.3.1. BDD Architecture

With the BDD method the fault tree is first transformed into a BDD which encodes Shannon's decomposition [9] and represents the Boolean equation for the top event. It allows the minimal cut sets to be obtained directly and the exact failure probability to be determined in an efficient way.

A BDD is composed of a root vertex, non-terminal (internal) vertices and terminal vertices which are connected by branches. Sometimes vertices are also called nodes and branches are named edges. Terminal vertices end with the value 1 or 0 that corresponds to the system state, while non-terminal vertices represent the corresponding basic events of the fault tree. Every vertex has two branches with the assigned value 1 or 0. The branch with value 1 represents failure of a basic event or vertex occurrence and the 0 branch indicates functioning of the basic event or vertex non-occurrence. All the left branches leaving a vertex are the 1 branches, where the right branches are assigned the value 0. *Figure 2.4* represents an example of a BDD.

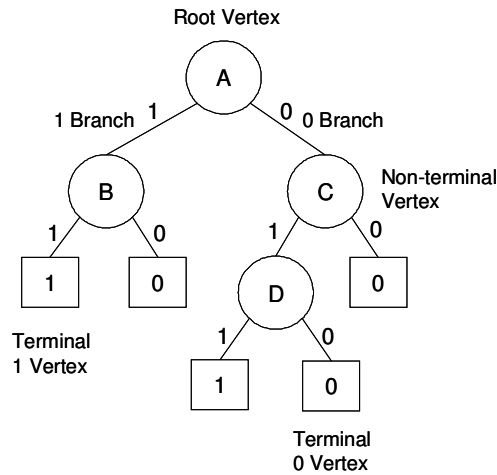


Figure 2.4. BDD Example

Fault tree cut sets can be directly found from its BDD. Every path through the diagram starts at the root vertex and proceeds to a terminal vertex. All the paths terminating in a 1 state yields the cut sets of the fault tree. A minimal cut set is formed by the vertices that lie on 1 branches on the way to a terminal 1 vertex. For example, there are 2 paths that terminate in a 1 state in the BDD presented in *Figure 2.4*: $\{ A, B \}$ and $\{ \bar{A}, C, D \}$. Since vertex A lies on the 0 branch in the second path it is not included in the minimal cut set. Thus the given BDD has two minimal cut sets: $\{ A, B \}$ and $\{ C, D \}$.

2.3.2. BDD Construction

Two methods are commonly used to convert a fault tree to the appropriate BDD. The first method is based on the top event logic function, while the second is derived using **If-Then-Else (ite)** technique. The logic function method requires a considerable amount of simplification by applying Boolean reduction laws after each function evaluation. These problems can be alleviated when using the second method.

The **ite** structure for the BDD construction was first introduced by Rauzy [6]. The method derives from Shannon's formula which is applied at each gate of the fault tree. To illustrate the formula consider a Boolean structure function for the top event $f(\mathbf{X})$. Pivoting about any variable x_i , the Shannon formula can be written as

$$f(\mathbf{X}) = x_i f_1 + \bar{x}_i f_2 \quad (2.9)$$

where f_1 and f_2 are Boolean functions with $x_i = 1$ and 0 respectively. The **ite** structure that corresponds to *Equation 2.9* is **ite**(x_i, f_1, f_2). It means **if** the Boolean variable x_i fails **then**

consider logic function $f1$ **else** consider logic function $f2$. Since the 1 branch in the BDD represents failure, $f1$ lies on the 1 branch and $f2$ appears on the 0 branch. The diagrammatic representation of the **ite** structure is in *Figure 2.5*.

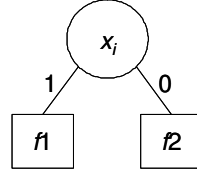


Figure 2.5. Representation of $\text{ite}(x_i, f1, f2)$

Consider

$$J = \text{ite}(x, f1, f2) \text{ and } H = \text{ite}(y, g1, g2)$$

The following operation procedures can be defined for the **ite** structures:

$$\begin{aligned} J \langle \text{op} \rangle H &= \text{ite}(x, f1 \langle \text{op} \rangle H, f2 \langle \text{op} \rangle H) \text{ if } x < y \\ J \langle \text{op} \rangle H &= \text{ite}(x, f1 \langle \text{op} \rangle g1, f2 \langle \text{op} \rangle g2) \text{ if } x = y \end{aligned} \quad (2.10)$$

Here $\langle \text{op} \rangle$ represents the Boolean operations such as AND (\cdot) and OR ($+$). The introduced procedures can be simplified using the following identities:

$$\begin{aligned} 1 \cdot \text{ite}(x, f1, f2) &= \text{ite}(x, f1, f2), \\ 0 \cdot \text{ite}(x, f1, f2) &= 0, \\ 1 + \text{ite}(x, f1, f2) &= 1, \\ 0 + \text{ite}(x, f1, f2) &= \text{ite}(x, f1, f2). \end{aligned} \quad (2.11)$$

Construction of the BDD from a fault tree is implemented according to the following conversion procedure:

Basic events are given an ordering, such as $x < y$ or $A < B$ (example from *Figure 2.4*). Usually a top-down ordering procedure is employed. According to the scheme the basic events placed higher up the tree are listed first and are “less than” those basic events appearing further down the tree. There are other ordering schemes such as top-down left-right, top-down left-right repeated, depth-first or priority depth first. The chosen ordering scheme may influence the size of the resulting BDD.

Assign each basic event x_i in the fault tree the **ite** structure $\text{ite}(x_i, f1, f2)$.

Consider each gate in the fault tree in a bottom-up approach.

Derive the **ite** structure for the top event and simplify the expression.

To draw the BDD successively break down each **ite** structure in the top event into its 1 and 0 branches.

As an example of the BDD construction method the fault tree in *Figure 2.6* is considered.

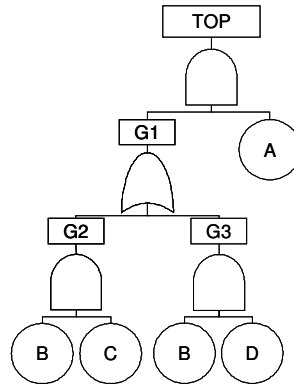


Figure 2.6. Example Fault Tree

To order the basic events a top-down, left-right ordering is used resulting in $A < B < C < D$. Each event is assigned an **ite** structure. Working from the bottom of the tree to the top operation procedures (2.10 and 2.11) are applied:

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

$$\begin{aligned} G3 &= \mathbf{ite}(B,1,0) \cdot \mathbf{ite}(B,1,0) \\ &= \mathbf{ite}(B, \mathbf{ite}(D,1,0), 0) \end{aligned}$$

$$\begin{aligned} G1 &= G2 + G3 \\ &= \mathbf{ite}(B, \mathbf{ite}(C,1,0), 0) + \mathbf{ite}(B, \mathbf{ite}(D,1,0), 0) \\ &= \mathbf{ite}(B, \mathbf{ite}(C,1,0) + \mathbf{ite}(D,1,0), 0) \\ &= \mathbf{ite}(B, \mathbf{ite}(C,1, \mathbf{ite}(D,1,0))), 0) \end{aligned}$$

Finally the top event is expressed:

$$\begin{aligned} \text{TOP} &= A.G1 \\ &= \mathbf{ite}(A,1,0) \cdot \mathbf{ite}(B, \mathbf{ite}(C,1, \mathbf{ite}(D,1,0))), 0) \\ &= \mathbf{ite}(A, \mathbf{ite}(B, \mathbf{ite}(C,1, \mathbf{ite}(D,1,0))), 0), 0) \end{aligned}$$

Once the **ite** structure for the top event is given the BDD is constructed by successively dividing each **ite** structure into the corresponding 1 and 0 branches. According to the given ordering event A is considered first and it becomes the root vertex. The structure $\mathbf{ite}(B, \mathbf{ite}(C, 1, \mathbf{ite}(D, 1, 0)), 0)$ lies below its left branch and its right branch is the terminal 0 vertex. According to the ordering next variable B follows, which is encoded in the vertex beneath the 1 branch of the A vertex. Thus the structure $\mathbf{ite}(B, \mathbf{ite}(C, 1, \mathbf{ite}(D, 1, 0)), 0)$ is analysed. Then $\mathbf{ite}(C, 1, \mathbf{ite}(D, 1, 0))$ will lie below the left branch of B and the right branch will terminate in the terminal 0 vertex. Following event C is considered and its branches are determined by breaking down the structure $\mathbf{ite}(C, 1, \mathbf{ite}(D, 1, 0))$. Its left branch terminates in the terminal vertex 1 and the right branch terminates in the last non-terminal vertex representing event D. The D vertex is finally broken down into terminal vertices 1 and 0. When all basic events have been considered and all branches end with terminal vertices the construction process is terminated. The resulting BDD is presented in *Figure 2.7*.

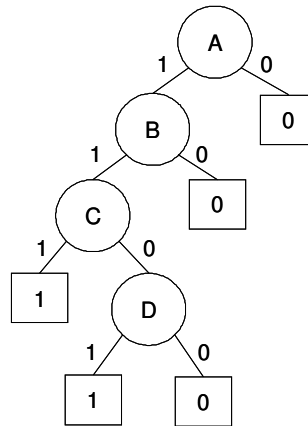


Figure 2.7. BDD for the Fault Tree in Figure 2.6

2.3.3. Reduction

The size of the fault tree influences the size of the BDD. However different basic event orderings applied to the same fault tree will result in different sized BDDs. Therefore a poor ordering can result in an inefficient BDD. One way of producing a more efficient diagram can be the application of reduction procedures for repeated nodes.

Two reduction operations can be performed in order to remove the irrelevant repeated events. For example, if a node X has two equivalent nodes lying below its left and right branches, then node X can be deleted and all of its incoming edges need to be directed to a node attached to its left branch. *Figure 2.8* illustrates the procedure. Another reduction operation is

applicable when two nodes are equivalent and their incoming edges are directed to different nodes where one of them lies beneath another as it is shown in *Figure 2.9 a)*. In this case one of the repeated nodes is deleted and its incoming edge is directed to the remaining node. The diagram resulting from this reduction is shown in *Figure 2.9 b)*.

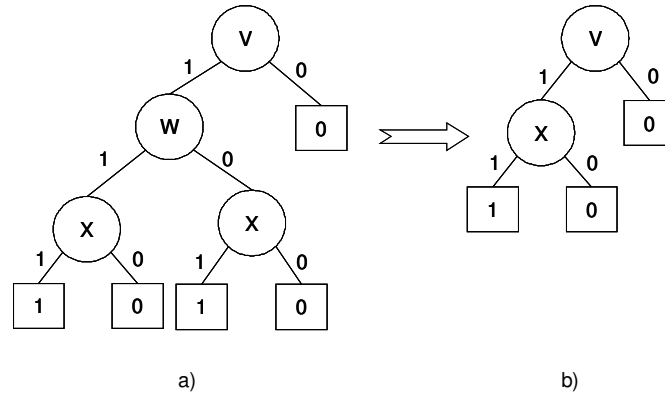


Figure 2.8. BDD Reduction Operation 1

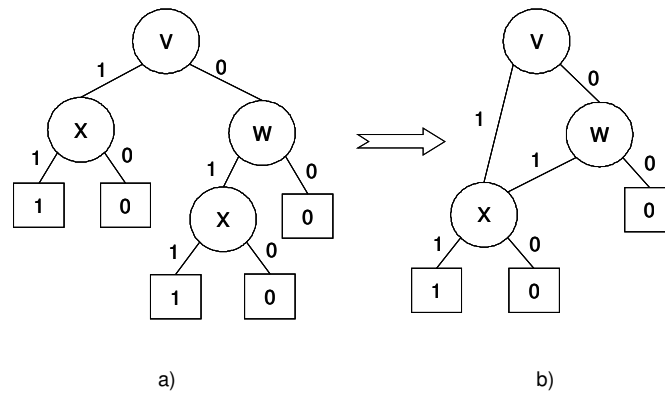


Figure 2.9. BDD Reduction Operation 2

2.3.4. Quantitative BDD Analysis

On the contrary to the fault tree approach, the BDD method used for the exact solution of the top event avoids the need to use approximations. The probability of the top event is obtained directly from the diagram which makes this method computationally efficient. If employing the BDD method system performance measures, such as the earlier mentioned system failure probability (the top event probability), unconditional system failure intensity or importance measures, can be found [10]. In this section the BDD methodology to evaluate the system failure probability is discussed.

As discussed, the BDD is constructed employing the **ite** structure which is derived from Shannon's formula. Given a structure function $f(\mathbf{X})$ for the top event, the probability is obtained by taking the expectation of each term of Equation 2.9:

$$E[f(X)] = q_i \cdot E[f1] + (1 - q_i) \cdot E[f2] \quad (2.12)$$

where $q_i = E[x_i]$ is the probability that event i has occurred.

Iteratively expanding the terms in Equation 2.12, i.e. calculating expectations for each node, results in a sum of disjoint products. In the resulting expression each product corresponds to a particular path through the BDD to a terminal 1 vertex. Therefore the probability of occurrence of the top event, Q_{sys} , is obtained by calculating the sum of the probabilities of the disjoint paths through the BDD. The disjoint paths are found by traversing all paths from the root vertex to terminal 1 vertices and including all events which lie on the 1 and 0 branches for each of the basic events.

Consider the BDD presented in Figure 2.10. In order to find the top probability, Equation 2.12 is applied to each node in the BDD in a bottom-up approach. Thus the nodes which have both terminal vertices are considered first:

$$F6 = \mathbf{ite}(D,1,0) \Rightarrow E[F6] = q_D \cdot 1 + (1 - q_D) \cdot 0 = q_D,$$

$$F3 = \mathbf{ite}(E,1,0) \Rightarrow E[F3] = q_E \cdot 1 + (1 - q_E) \cdot 0 = q_E,$$

Then node $F5$ is considered. Its **ite** structure is:

$$F5 = \mathbf{ite}(C, F6, 0).$$

Applying Equation 2.12 and substituting in the probability of node $F6$, the probability of node $F5$ is found:

$$E[F5] = q_C \cdot E[F6] + (1 - q_C) \cdot 0 = q_C \cdot q_D.$$

In the same manner the probabilities for nodes $F4$ and $F2$ are evaluated:

$$F4 = \mathbf{ite}(E,1, F5) \Rightarrow E[F4] = q_E \cdot 1 + (1 - q_E) \cdot q_C \cdot q_D = q_E + (1 - q_E) \cdot q_C \cdot q_D.$$

$$\begin{aligned} F2 = \mathbf{ite}(B, F4, F3) \Rightarrow E[F2] &= q_B \cdot (q_E + (1 - q_E) \cdot q_C \cdot q_D) + (1 - q_B) \cdot q_E \\ &= q_B \cdot (1 - q_E) \cdot q_C \cdot q_D + q_E. \end{aligned}$$

Finally the probability of occurrence of the top event, i.e. the sum of the probabilities of the disjoint paths through the BDD is found:

$$\begin{aligned} F1 &= \text{ite}(B, F4, F2); \quad E[F1] = q_A \cdot (q_E + (1 - q_E) \cdot q_C \cdot q_D) + (1 - q_A) \cdot (q_B \cdot (1 - q_E) \cdot q_C \cdot q_D + q_E) \\ &= q_A \cdot (1 - q_E) \cdot q_C \cdot q_D + (1 - q_A) \cdot q_B \cdot (1 - q_E) \cdot q_C \cdot q_D + q_E. \end{aligned}$$

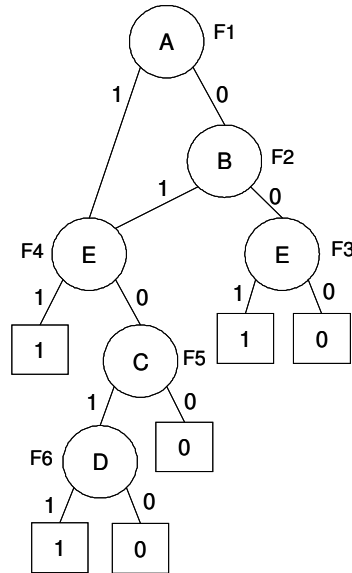


Figure 2.10. Example BDD

2.4. SUMMARY

The operation of a system can be considered from two standpoints: enumeration of various ways for system success, or enumeration of various ways leading to system failure. A primary goal of system reliability and safety analysis is to identify the causal relationships between events resulting in system failure and to find ways to reduce their numbers and impact by system redesigns and upgrades. In this chapter two methods have been introduced which are widely used in the field of reliability engineering.

The fault tree provides the diagrammatic failure logic of the system. Hence it can be used to identify problematic areas in the system design. Fault trees can be used for both qualitative analysis (to find system minimal cut sets) and quantitative system analysis (to evaluate system performance parameters), for example, system unavailability. However fault tree analysis for complex systems can be computationally very intensive. In order to reduce calculations approximations may be introduced, which however leads to the loss of accuracy. BDDs were introduced as a more efficient method for quantitative fault tree analysis.

The BDD method converts a fault tree to the BDD which encodes a Boolean equation for the top event. By using the BDD both the qualitative and the quantitative analysis can be achieved. The BDD method enables the exact system unavailability to be determined without explicitly using minimal cut sets which makes the method computationally very efficient. However the size of the BDD depends on the given ordering of the basic events used to build the BDD. Badly chosen ordering may result in a computationally-intensive BDD structure which reduces the efficiency of the analysis.

Both the fault tree analysis and the BDD technique are employed in this research. Since a fault tree provides a diagrammatic description of the failure logic it is used as a design tool for system design representation. BDD methods are computationally efficient mechanisms for quantitative fault tree analysis, therefore the analysed system unavailability value for an individual design is found using the BDD methodology. The application of these techniques is discussed in *Chapters 4 and 7*.

3. TECHNIQUES FOR DESIGN OPTIMISATION IN RELIABILITY ENGINEERING

3.1. INTRODUCTION

Optimisation Theory is widely used to solve engineering problems where the emphasis is on the maximisation or minimisation of a certain goal. From an engineering point of view one part of optimisation is the evaluation of design proposals. The second part is the generation of new improved designs. The goal of optimisation is to find the optimal solution given the properties of the system being designed and the behaviour of the system model [11].

A general mathematical form of the optimisation problem can be written as follows:

$$\max(\min)f(\mathbf{X}), \quad (3.1)$$

subject to inequality and equality constraints:

$$g_i(\mathbf{X}) \geq 0, i = 1, \dots, I; \quad (3.2)$$

$$h_j(\mathbf{X}) = 0, j = 1, \dots, J. \quad (3.3)$$

The vector \mathbf{X} ($\mathbf{X} = (x_1, x_2, \dots, x_n)$) is referred to as vector of design variables. The objective function, $f(\mathbf{X})$, given by *Equation 3.1* measures the quality of the solution. The objective function as well as the constraint functions defined by *Equations 3.2* and *3.3* may be linear or nonlinear functions of the design variables \mathbf{X} . These functions may be explicit or implicit in \mathbf{X} and may be evaluated using analytical or numerical techniques [12].

The optimisation of a system design is a classical optimisation problem in the area of system reliability engineering [13]. In general, the objective of such problems is to optimise a function-of-merit of the system design (reliability, cost, mean time to failure, etc) subject to known constraints on resources (cost, weight, volume, etc) and/or system performance requirements (reliability, availability, mean time to failure, etc.) [14]. A design is considered to be optimal if all the possible means available have been explored to enhance the reliability of the system under certain objectives, operational requirements and allocated resources.

The diversity of system structures, resource constraints and options for reliability improvement lead to the construction and application of several optimisation techniques [13]. The most common optimisation techniques which have been applied in the optimisation of system reliability, and have had some success in solving particular optimisation problems, are these: linear programming, nonlinear programming, discrete optimisation, dynamic programming, modern heuristics (metaheuristics) and multi-objective optimisation techniques. In some cases to find an optimal solution a group of optimisation techniques can be employed. Conversely, it is almost unrealistic to solve all reliability optimisation problems using one method.

In this chapter the basic concepts behind the listed optimisation techniques are considered. Examples of the applications of the techniques in reliability optimisation are also provided. Moreover, where appropriate a critical review, with regards to their application to reliability optimisation problems, is expressed. *Section 3.2* covers linear programming. Nonlinear programming methods are discussed in *Section 3.3*. In *Section 3.4* concepts and application examples of discrete optimisation methods are provided. *Section 3.5* discussed metaheuristics. The genetic algorithm method, an optimisation technique employed in this research, is discussed in more detail in *Section 3.5.4*.

3.2. LINEAR PROGRAMMING

Linear programming (LP) is the fundamental mathematical optimisation method. A number of optimisation problems can be expressed as linear programming problems. For example, nonlinear problems can be solved through a series of linear programming problems.

LP deals with problems where $f(\mathbf{X})$ is a linear function of n variables and the constraints are also linear. There are two main classes of algorithms to solve LP problems. The first class contains simplex-type algorithms, the second is the class of interior-point methods [15]. The simplex methods move from one extreme point on the boundary (vertex) of the feasible region to another along the edges of the boundary iteratively. This involves identifying the constraints (lines) on which the solution will lie [16]. An interior-point algorithm, in contrast to the classical Simplex algorithm, searches for a feasible solution point through the interior of the feasible region [17].

LP in reliability optimization is used for solving an optimisation problem with a linear form of non-negative variables subject to a system of linear inequalities or a nonlinear optimisation

problem having been transformed into a linear form. Kolesar [18] considered an optimal assignment of redundant components in systems which are subject to random failure. In general, the objective of the problems solved was to maximise the system reliability through assignment of redundant components subject to constraints on the total weight, cost, and so forth of the system. The constraint functions were of the linear form. The problems analysed included systems subject to a single type of failure and systems subject to both the possibility of premature operation and the possibility of failure to operate on command. In each case the problem was expressed as a linear programme.

Hsieh in [19] investigated reliability problems subject to multiple separable linear constraints of series-parallel redundant systems, where each subsystem had multiple component choices. A simple linear programming approach was proposed that approximates the integer nonlinear programming problem. The numerical results presented demonstrated the efficiency of the proposed approach, however it could not be guaranteed that the approach derived the global optimum. The main limitation of the approach was with regards to the requirements defined for constraints; they had to be linear and separable.

Most of the reliability optimisation problems have a nonlinear objective function and/or nonlinear constraints. Therefore the greatest disadvantage of the LP technique with regards to its application in reliability optimisation problems is the supposed linearity of the objective and constraint functions. Another disadvantage of LP techniques is that they are very time-consuming when solving large scale optimisation problems.

3.3. NONLINEAR PROGRAMMING

Nonlinear programming (NLP) involves problems where either the objective function, the constraints, or both are nonlinear. There are three main approaches for solving a NLP problem. The first approach involves methods with an iterative feasible direction search. This approach is useful for problems involving linear constraints. Methods from the second approach are based on Lagrange multipliers and can be easily implemented when the system involves single equality constraints. Using the third approach the solution of a constraint optimisation problem is obtained by solving a sequence of unconstrained optimisation problems, whose objective functions are penalised for violating the constraints. In this case solutions to the unconstrained problems approach an optimal solution of the original constrained problem [13].

Most of the reliability optimisation problems are discrete, mixed integer and nonlinear optimisation problems. Such problems can be solved by using NLP methods if appropriate rounding off procedures for integer variables are utilised. Everett III in [20] used the Lagrange multiplier method to optimise the redundancy of an m -stage system, each stage of which consisted of a number n_i of parallel (redundant) components. The formulated objective was to choose the stage redundancies (n_i 's) in such a manner as to minimise the cost of achieving some stated system reliability (or alternately, to maximise the system reliability subject to constrained total cost). Hwang *et al.* [21] proposed an augmented Lagrangian method and a reduced gradient method for system reliability optimisation. Li and Haimes in [22] proposed a 3-level decomposition approach for the optimal allocation of available resources to subsystems in order to maximise the reliability of a large system with a general network structure. The developed methodology greatly reduces the complexity of the large problem by solving several smaller-dimensional sub-problems iteratively. The sub-problems can be solved by any existing nonlinear programming method.

Both the generalised reduced gradient and Lagrangian methods are promising in solving reliability optimisation problems [23]. The Lagrangian method is not limited to differentiable functions and can be applied in situations involving the maximisation of any type of function over any set of strategies, discrete or continuous, numerical or non-numerical, with constraints that can be represented as bounds on real valued functions over the same strategy set [20]. However most of these techniques are problem-orientated. Therefore since they are designed for solving certain problems, it is difficult to adopt them for solving other problems [24]. Moreover, it is not always possible to solve larger scale problems using these approaches.

3.4. DISCRETE OPTIMISATION

Discrete optimisation problems involve discrete (integer) decision variables. It includes integer programming (IP), mixed integer linear programming (MILP) and mixed integer nonlinear programming (MINLP) problems [25]. In IP problems decision variables are scalar and integer. In MILP optimisation problems the linear objective function and linear constraints are analysed, and integer as well as continuous decision variables are involved. MINLP optimisation problems are similar to NP problems, however MINLP involves integer and continuous decision variables.

The majority of reliability optimisation problems are nonlinear integer programming problems. Since their solutions need to be integer they are more difficult to solve than general nonlinear programming problems. Discrete optimization approaches, such as dynamic programming, branch-and-bound techniques and integer programming methods are the most widely used exact optimisation approaches for solving such problems.

Yalaoui *et al.* [26] proposed a new dynamic programming method for the reliability redundancy allocation problem for series-parallel systems, where components and their reliability belong to a finite set. The solved problem is decomposed into as many sub-problems as subsystems. The global problem consists of determining the reliability target of the subsystems. In the obtained method sub-problems and the global problem are solved by a dynamic programming technique and result in convergence towards an optimal solution. Ng and Sancho used a hybrid ‘dynamic programming/depth-first search’ algorithm to solve redundancy allocation problems for series-parallel systems [27]. It computes a global optimal solution to the optimisation problem and provides an alternative to the traditional lagrangian approaches which often fail to identify an optimal solution of integer optimisation problems.

The most efficient branch-and-bound method for redundancy allocation problems at present is the method developed by Nakagawa *et al.* [28]. Sung and Cho in their paper [29] considered a reliability optimization problem for a series system with multiple-choice constraints incorporated for each subsystem to maximize the system reliability subject to the system budget. In the approach some solution properties were characterized, such as lower and upper bounds of the system reliability, to reduce the solution space in advance. A branch-and-bound solution algorithm was then derived based on the reduced solution space to search for the optimal solution. The authors stated that the proposed algorithm can be applied to various practical-size field systems. Ha and Kuo in [30] presented an efficient branch-and-bound approach to solve the redundancy allocation problem with the objective of system reliability optimisation. The main advantage of the proposed method is flexibility. It does not rely on any assumptions of linearity, a single constraint or separability, which make the method adaptable to various applications. Here a problem is considered to be separable if the system reliability is equal to the sum of reliability of the subsystems and the total amount of resources is equal to the sum of the consumption of resources at each subsystem. The authors demonstrated that the method is superior to the existing exact algorithms for redundancy allocation problems in terms of computation time.

Misra introduced an efficient technique for a variety of reliability optimization problems, which involve integer programming formulation [31]. The algorithm is based on functional evaluations and a limited search close to the boundary of resources. This procedure has the following advantages over the other existing techniques: it requires only functional evaluations and it does not require the conversion of the original decision variables into binary variables, there are no assumptions on the separability, differentiability and convexity/concavity of the objective functions and/or constraints.

The advantage of the discrete methods is that they give an exact optimal solution. Integer programming methods also yield integer solutions. However the transformation of nonlinear objective functions and constraints into linear forms so that integer programming can be employed can be a difficult task. Moreover the various integer programming techniques do not guarantee to find optimal solutions in a reasonable time [13]. In general, computational complexity of the discrete methods is very high. Branch-and-bound methods do not exploit separability to reduce the computation. Moreover the effectiveness of the methodology depends on the problem specifics. Most branch-and-bound algorithms are confined to linear constraints with an objective function that need not be linear. The implementation of DP is limited by the number of constraints and the system structures it can be applied to. It is not applicable to non-separable objective or constraint functions which arise in reliability optimization problems where complex structures are considered. It also has dimensional difficulties. For a system which has more than two constraints, the computational complexity of dynamic programming increases exponentially. Thus development of a good discrete method for reliability optimisation problems remains a challenge.

3.5. META-HEURISTIC METHODS

The major focus of recent work in reliability optimisation is in the development of modern heuristic algorithms [32]. Often these algorithm are referred to as metaheuristics or general heuristics [33]. As described in [34] “a heuristic is a technique which seeks good (i.e. near-optimal) solutions at a reasonable computational cost without being able to guarantee either feasibility or optimality, or even in many cases to state how close to optimality a particular solution is”. These methods facilitated solution of optimisation problems that were previously difficult or impossible to solve. The most popular of these tools are genetic algorithms (GA), random search (RS), simulated annealing (SA), tabu search (TS) and particle swarm optimisation (PSO). They consist of general search principles organized in a general search strategy.

Heuristics compute approximate or locally optimal solutions of an optimisation problem. Despite this fact they have some advantages over exact methods. They are more flexible than exact algorithms and are usually used to solve more complicated and larger sized problems.

3.5.1. Simulated Annealing

The idea of the SA originates from thermodynamics and metallurgy which was originally proposed as a simulation of the cooling of materials in a heat bath, the process known as annealing. It is an approach to search for the global optimal solution that attempts to avoid entrapment in poor local optima by allowing an occasional uphill move to inferior solutions. The method involves probabilistic transitions among the solutions of the problem. During the iteration process a random solution x is drawn in the neighbourhood of the current solution x_n . If the objective function value of the solution is not worse than the one of the current solution ($f(x) \leq f(x_n)$ in the minimisation case), x becomes the next current solution. Otherwise, either x becomes the current solution with probability $p(n)$ or x_n remains the current solution with the complementary probability $1-p(n)$. Typically, $p(n)$ decreases with time and with the size of deterioration of the objective function [33].

To obtain the optimal schedule of testing and maintenance of safety equipment in nuclear power plants Cebin proposed an optimisation method based on the SA algorithm [35]. The algorithm for minimisation of risk by finding the optimal test placement times was used to evaluate several examples. One of them was the high – pressure injection system that consists of seven valves and three pumps, which provides water to two injection paths [36]. The results have shown that it is possible to reduce risk by employing the developed algorithm. The most important result of the method possibly is the prevention of schedules of equipment outages, which result in high risk.

Recently, Kim *et al.* in [37] applied the SA to search for the optimal solution of reliability-redundancy allocation problems. The objective of the problem solved was to maximize the system reliability subject to three nonlinear resource constraints. Three types of systems were analysed: the series system, the series-parallel system and the complex (bridge) system. It was assumed that the system had identical components in the subsystem and one failure mode. The results of the conducted numerical experiments suggest that the best solution for the SA algorithm are better than most of the solutions considered in the comparative analyses.

SA is effective when a problem is highly complex without having any special structure. Although the SA gives satisfactory solutions for complicated combinatorial optimisation

problems, it has a major disadvantage. The SA involves a lot of computation effort with a large number of function evaluations and tests for solution feasibility.

3.5.2. Tabu Search

Tabu search (TS) can be described as an alternative to SA and a form of neighbourhood search. However the neighbourhoods in TS are assumed to be symmetric, i.e. x_1 is a neighbour of x_0 if and only if x_0 is a neighbour of x_1 [34]. The main idea of this method is to explore and analyse various regions of the search space. Using this method in the searching process at any stage memory (information about solutions visited up to that stage) rather than probability plays the important role [15]. The main parameters of the TS algorithm are the history record H (definition and usage), determination of the candidate neighbourhood set and the evaluation function $f(H, x)$. These parameters are usually changed and fitted to the problem that is to be solved. As with SA techniques the TS technique can be improved by combining it with other methods. Reference [38] details such combined algorithms.

Kulturel-Konak *et al.* in [39] have used the TS to solve redundancy optimisation problems. The series system of s independent k -out-of- n :G subsystems have been analysed. The algorithm was applied for two types of problems. The first problem maximised the system reliability given overall restrictions on the system cost and weight. Problem two minimised the system cost given overall restrictions on the maximum system weight and the minimum system reliability. It was also assumed that system weight and cost were linear combinations of component weight and cost. Moreover the TS was designed with the use of a penalty function which allowed search in the infeasible region. The application of the algorithm demonstrated encouraging results. When compared to GAs, the algorithm resulted in a superior performance in terms of best solutions found and reduced variability and greater efficiency.

Ouzineb *et al.* in [40] also developed an efficient TS based algorithm to solve redundancy allocation problems. The algorithm was applied to determine the minimal system cost configuration under availability constraints for multi-state series-parallel systems. The system analysed could have a range of performance levels from perfect functioning to complete failure. The elements of the system were characterized by their cost, performance and availability which belonged to a finite set. The algorithm proceeded by dividing the search space into a set of disjoint subsets, and then by applying TS to each subset. Comparison of

numerical results for the test problems from previous research showed that the proposed TS out-performed GA solutions, in terms of both the solution quality and the execution time.

TS is very useful for solving large complex optimisation problems that are very difficult to solve by exact methods. However it is rather difficult to define effective memory structures and memory-based strategies which are problem dependant. Thus development of an effective TS method requires thorough understanding of the problem and some numerical experimentation.

3.5.3. Particle Swarm Optimisation

Particle swarm optimisation (PSO) is a stochastic global optimization technique inspired by social behaviour of bird flocking or fish schooling. It was first introduced by Kennedy and Eberhart in [41]. PSO is initialised with a population of random solutions within a feasible range, called particles or individuals. In the algorithm during the learning procedure each individual particle keeps track of its coordinates in the search hyperspace which are associated with two factors: the best solution ever found (*personal best*) and the overall best fitness value and its location (*global best*). During the optimisation process at each time step, the velocity each particle moves toward its *personal best* and *global best* is changed. The velocity is dynamically adjusted by a random term, with separate random numbers being generated for acceleration toward *personal best* and *global best*.

Coelho [42] presented an efficient PSO algorithm to solve the reliability–redundancy optimisation problem. Two examples of reliability–redundancy design problems were considered: a complex bridge system and a specific system, and an overspeed protection system for a gas turbine. The latter was formulated as a mixed-integer nonlinear programming problem. Simulation results demonstrated that the proposed PSO performed well for the two examples of mixed-integer programming in reliability–redundancy applications. The solutions obtained by the PSO were better than the previously best-known solutions available in the literature.

The algorithm has very few parameters. It is very simple and easy to implement. Moreover it has a very efficient global search procedure. However, the main disadvantage of the algorithm is its poor local search ability. It has slow convergence in the refined search stage and prematurity. The PSO may fail to find the required optima in cases when the problem to be solved is too complicated and complex [43].

3.5.4. Genetic Algorithms

The GA was developed by J. Holland and his associates at the University of Michigan in the 1960s and 1970s. In 1975 *Adaptation in Natural and Artificial Systems* - the primary monograph about GAs by J. Holland was published [44]. GAs belong to the group of evolutionary algorithms simulating the natural evolutionary process of living beings and they are perhaps the most widely known type of these algorithms. GAs differ from conventional optimisation techniques in a number of fundamental ways. They work with a coding of the solution set, not solutions themselves. They also deal with populations of solutions rather than with single solutions. GAs use fitness function and probabilistic transition rules in the search process.

GAs are stochastic global search methods based on the mechanics of natural genetic variation and natural selection. Thus terminology used in GAs is analogous to biological systems. For example, *strings* that are used in the optimisation algorithm are analogues to *chromosomes* in biological systems. *Genes* form *chromosomes* and are located at particular *locus* (positions) on the chromosome. Analogically in a GA *variables* correspond to genes and a total package of strings forms *a structure*. Describing biological genetics the term *alleles* is used, which means that genes can have some values, thus in a GA alleles are *the possible values of variables*. To describe the collection of chromosomes that form the structure of the organism biologists use the term *genotype* and *phenotype* as a physical expression of the structure. In terms of the GA a genotype is *a coded string* and phenotype represents *the decoded set of parameters* [25], [45]. The GA and corresponding optimisation terms are summarised in Table 3.1.

Table 3.1. Explanation of GA Terms

<i>Genetic Algorithm Term</i>	<i>Optimisation Term</i>
Chromosome	Solution (string, individual)
Genes (bits)	Part of solution, a member of solution vector
Locus	Position of gene
Alleles	Values of gene
Phenotype	Decoded solution
Genotype	Encoded solution

The GA is a meta-heuristic method and it does not guarantee to find the global solution, but it has been theoretically and empirically proven that the method provides accurate and reliable optimisation results. Recently, GAs have received considerable attention and have been proven to be a powerful tool for solving a large number of complex optimisation problems. The algorithms are applied in such areas of reliability engineering as 1) redundancy allocation

and structure optimisation, 2) optimal network design, 3) maintenance and surveillance optimisation [24].

Coit D.W. and Smith A.E. used a GA to optimise series-parallel system design configurations when there are multiple component choices available for each of several k -out-of- n : G subsystems [46]. The problem considered was to select the optimal combination of parts and levels of redundancy to minimise system cost subject to reliability and weight constraints, or alternatively, to maximise reliability subject to cost and weight constraints. Tavakkoli-Moghaddam *et al.* [47] also solved the series-parallel systems reliability optimisation problem using a GA. The objective was to select the best redundancy strategy, component, and redundancy level for each subsystem in order to maximise the system reliability under system-level constraints.

Yun and Kim in [48] considered redundancy allocation problems in series systems. The authors adopted a GA approach to solve a problem in which redundancy can be available at all levels in the system. The objective of the problem presented was to maximise system reliability given the constraints of available resources such as cost, weight and volume. The results obtained from the illustrative example showed that considering modular redundancy could be better than using only component redundancy.

Hsieh *et al.* presented GAs for reliability design problems where both the component reliabilities and redundancy allocations were considered. The systems analysed included series, series-parallel and complex (bridge) systems [49]. The objective of the problems solved was to maximise the system reliability, while maintaining feasibility with respect to nonlinear constraints. The constraints considered included cost, weight and constraints on the products of volume and weight. The authors reported that the solutions of the numerical examples performed were better than previously best-known solutions.

Some other authors employed GAs to analyse redundancy optimisation problems for multi-state systems [50], [51]. Levitin *et al.* in [50] considered multi-state systems that have a range of performance levels and represented a general redundancy optimisation problem for such systems. In solving the optimisation problem three system component characteristics were used: nominal performance level, cost and availability. The objective of the optimisation was to minimise total cost subject to the required reliability or availability. Levitin [52] also discussed a redundancy optimisation problem for a multi-state system of 2 subsystems. The objective of the work was to choose elements from a list of available equipment in order to

optimise system design subject to availability constraints. The optimisation problem was formulated as an investment cost minimisation problem which was solved using a GA.

A GA based optimisation approach has also been applied to optimise surveillance and maintenance of components in order to improve system reliability [53], [54], [55], [56]. Munõz *et al.* in [53] presented a new approach aimed at the global and constrained optimisation of surveillance and maintenance of components based on risk and cost criteria. Lapa *et al.* in [54, 55] proposed a method for preventive maintenance scheduling optimisation of standby systems where a GA was employed as an optimisation technique. The goal of the approach was to improve the average availability of the system when optimising the preventive maintenance strategy. The proposed method was applied to a nuclear system. Marseguerra and Zio in [57] examined the approach of the optimal maintenance and repair strategies of an industrial plant considering some reliability and economic constraints. The GA was employed to search for an optimum combination of plant safety and economic performance that was evaluated for each possible maintenance and repair strategy.

There are many other GA applications in engineering reliability optimisation problems. For instance, Monga and Zuo in [58] introduced a reliability based design model for a series-parallel system with deteriorating components in order to optimise the life cycle cost of the system. To model the economic effects of the system life cycle acquisition costs, preventive maintenance costs, minimal repair costs and system's salvage value at the time of disposal were incorporated into the model. The objective of the problem was formulated as the minimisation of system cost subject to both active and non-active constraints. Dengiz *et al.* [59] developed a GA approach with specialised encoding, initialisation and local search operators to optimise the design of communication network topologies. The objective of the analysed problem was to minimise network cost given a minimal reliability requirement.

Ren and Dugan in [60] adopted a GA in a fault tree method to determine the optimal design configuration of a reliable system. The presented methodology could be employed to analyse optimal system design from two different design-viewpoints: to maximise system reliability given cost, weight and/or physical size constraints and to minimise system cost subject to reliability constraints. Andrews and Bartlett [61] also combined GA and fault tree approaches to optimise the design of a Firewater Deluge System. The objective of the problem was to minimise the system unavailability subject to cost and spurious system shutdown constraints.

3.5.4.1. Advantages and Disadvantages of a GA

GAs are one of the most widely used metaheuristics. They have become popular techniques to solve various optimisation problems. GAs can be used to solve complex discrete optimisation problems with any kind of non-linear objective functions and constraints defined in discrete, continuous or mixed search spaces. They do not require much mathematical information about the optimization problem [62]. Only a few assumptions on the objective as well as the constraint functions are involved. GAs use only the objective function itself to measure the fitness score of each solution. Therefore they can be very effective when the objective function is not available in a closed mathematical form.

In order to find a global optimum trade-off between the exploration and exploitation needs to be found. GAs combine elements of direct and stochastic search which can make a remarkable balance between exploration and exploitation of the search space [45], [63]. Therefore they can be very effective at performing global search and obtaining global optima [53], [62]. GA parameters, such as population size, maximum generation, crossover and mutation rates affect a balance between exploitation and exploration in the search space. However it takes much time to tune the unknown parameters [64].

GAs produce a variety of good quality solutions simultaneously, which is important in the decision-making process. They are also successful in locating potentially optimal regions. However they provide heuristic solutions since they are not designed for precisely locating the optimal solution. Moreover, they involve a lot of computational effort [13]. In order to reduce the effects of GAs drawbacks, algorithms can be hybridized with other domain-dependent heuristics when solving specific optimisation problems.

3.5.4.2. Fundamentals of the GA

A GA has been chosen as an optimisation technique for this research project. The choice of the GA can be attributed to a number of factors. GAs use a fitness function itself and do not require derivative or other auxiliary quantities. They can be very effective when the objective function is not available in a closed form. This factor is very important since the objective function represents the probability of system failure and is evaluated using FTA in the optimisation algorithm proposed in this research. Moreover, GAs are not problem orientated and can be easily implemented and adapted to solve different reliability optimisation problems for a range of systems considering both constrained and unconstrained optimisation cases. This trait allows GAs to be applied to a range of safety systems.

General Structure of a Genetic Algorithm

GA - differently from conventional search techniques - starts with an initial set of random solutions, i.e. chromosomes, called the population. The chromosomes evolve through iterations, called generations. At each iteration, the chromosomes are evaluated using some measure of fitness. New chromosomes of an auxiliary population, i.e. offspring chromosomes, are formed by merging two chromosomes from the current generation using a *crossover* operator and/or modifying a chromosome using a *mutation* operator. A new generation of chromosomes is formed by selecting some of the parents and offspring on the basis of their fitness values and rejecting others so as to keep the population size constant. After several generations, the algorithm converges to the best chromosome, which represents the optimum or suboptimal solution of the problem.

Let $P(i)$ and $C(i)$ be parent and offspring populations in the current generation i . The general form of a GA can be described through the following steps [62]:

Step 1: Make initial population $P(i)$, $i = 0$;

Step 2: Evaluate each chromosome in population $P(i)$, $i = 0$;

Step 3: Choose parents from population $P(i)$ to yield $C(i)$;

Step 4: Evaluate each chromosome in population $C(i)$;

Step 5: Select $P(i + 1)$ from $P(i)$ and $C(i)$

Step 6: If the maximum number of generations is reached, stop and return the best chromosome; if not, go to Step 3.

Initialisation

A GA starts with an initial population of say N encoded representations of solutions, i.e. chromosomes. Decision variables of each solution are coded using a coding technique; Holland suggested using binary (0 and 1) coding, but other coding techniques can be implemented, for example, integer or real-valued coding [34], [65]. The binary coding of decision variables has been used in the GA for this research. Using this technique every variable is converted into a binary string of corresponding length (number of digits). The number of bits (denoted with nb) required to code each variable is calculated using the following formula:

$$2^{nb-1} < (b-a) \leq 2^{nb} - 1, \quad (3.4)$$

where $[a, b]$ is the range of the decision variable.

A nb -digit binary string has 2^{nb} possible 0-1 combinations, i.e. it can represent 2^{nb} different discrete values. There are some rules that give equivalence between integer and binary numbers, for example, a discrete variable having V allowable discrete values can be transformed to a binary string using the formula:

$$j = \sum_{i=1}^n ICH(i)2^{i-1} + 1. \quad (3.5)$$

Here $ICH(i)$ is the value of the i th digit in the binary string, and n is the smallest integer satisfying $2^n > V$ [66].

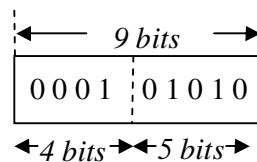
These binary strings, one for each decision variable, are concatenated to form one chromosome that represents one solution. As an example, suppose variables $x_1 \in [1, 15]$ and $x_2 \in [1, 18]$ are used in the problem. To calculate the number of bits in a chromosome *Formula (3.4)* is used where:

$$(15 - 1) = 14 \\ 2^2 < 14 < 2^4 - 1 \rightarrow n_1 = 4$$

$$(18 - 1) = 17 \\ 2^4 < 17 < 2^5 - 1 \rightarrow n_2 = 5$$

$$n = n_1 + n_2 = 9$$

The length of the chromosome is calculated as 9 bits which can be represented as follows:



The corresponding values of variables x_1 and x_2 are calculated using *Formula (3.5)*:

Binary Number	Decimal number
$x_1 = 0001$	9
$x_2 = 01010$	11

N chromosomes form the population of solutions with which the GA operates. The values of the initial solutions are usually generated randomly but several other methods can be used

[66]. For example, only solutions satisfying constraints can enter the initial population. The number of solutions in the initial population may vary. There are no strict rules as to the required population size for a problem, but it is known that this influences the algorithm performance. Some theoretical work has been done on this. Goldberg [45] showed that the optimal size for the population of binary strings depends on the length of the strings and the dependence is exponential, i.e. the optimal size of the population grows exponentially with the length of the string [34].

Reproduction

Once the initial population is determined *reproduction* is performed. Sometimes this process is also called *selection*. Both these terms can be used as it is the process which determines the number of times a particular individual is chosen for reproduction and, therefore, the number of offspring that an individual will produce. During this process selection and reproduction are performed. Selection is made according to the fitness values of each string (or individual). The fitness function is used to evaluate the fitness values of the strings. This function transforms the objective function value into a measure of relative fitness. A general form of the fitness function is

$$F(x) = g(f(x)), \quad (3.6)$$

where f is the objective function, g is the transformation function that changes the value of the objective function f to a non-negative number and F is the resulting relative fitness. In applications of the GA several forms of transformation can be used, such as linear transformation or power law scaling [67]. A commonly used transformation is that of proportional fitness assignment, which is sometimes interpreted as a probability of the string to be selected for the next generation. The transformation can be written as

$$F_i = \frac{f_i}{\sum_{i=1}^N f_i}, \quad (3.7)$$

where N is the population size, f_i is the objective function value of the individual i and F_i is the fitness value.

The rules that determine how individuals are selected for reproduction depend on the selection method. The most popular and easiest are roulette wheel selection methods. The methods are based on the link between the fitness value of each string in the population and the size of the particular segment of the determined interval which is usually interpreted as a

roulette wheel slot. The interval is determined as the sum of the individuals' (strings') selection probabilities or the sum of fitness values over all the individuals in the population. The current range $[0, sum]$ is divided into segments and the size of each segment corresponds to the fitness value of the associated individual. To select a string, a random number is generated from the interval $[0, sum]$ and an individual, that is associated with the segment spanning the random number, is selected.

There are two types of roulette wheel selection method. Both are described as stochastic sampling, but one type of method uses simple replacement of individuals and the other partial replacement. Employing stochastic sampling with replacement uses the same segment size and selection probability throughout the selection phase. Any individual string from the population is selected as described in the procedure previously outlined. In stochastic sampling with partial replacement each time an individual is selected for reproduction the segment associated with it is resized, i.e. the size of the segment is reduced by 1. This resizing process is continued until the segment size becomes equal to zero.

Crossover

Crossover is the main genetic operator, sometimes called recombination. The operator enables the creation of new strings at each generation. The strings that are created by crossover differ from that which are generated during initialisation. During initialisation new strings are generated randomly and new ones that are obtained during crossover have some identical parts of strings from which they are created. Thus, the crossover operates on two strings at a time, selected during the reproduction stage, (s_v, s_w) . The next step that follows depends on the form of crossover that is used. For example, using a single-point recombination operator an integer position, k , is selected uniformly at random from the range $[1, l-1]$. Here l is the length of the strings in the population. Strings s_x and s_y are crossed over at that k th position and the resulting two offspring are created. If genes of the string s_v are (v_1, v_2, \dots, v_l) and (w_1, w_2, \dots, w_l) are genes of the string s_w ($v_i, w_j \in \{0, 1\}$) then the new offspring are: $(v_1, \dots, v_i, w_{i+1}, \dots, w_l)$ and $(w_1, \dots, w_i, v_{i+1}, \dots, v_l)$ [69]. For example, consider two parent binary strings:

$$P_1 = 00|111011$$

$$P_2 = 11|010010.$$

Let the randomly elected crossover point be $i = 2$ and the two offspring would be:

$$O_1 = 00|010010.$$

$$O_2 = 11|111011$$

It is the simplest form of recombination. There are a number of more complicated variations of this GA operator, such as multipoint crossover or uniform crossover. For multipoint crossover, n non duplicate crossover positions are chosen, i.e. crossover points are chosen randomly and sorted into ascending order. Then, parts of the strings between successive crossover points are swapped between the two parents and two new strings are produced. Between parents the sections of the strings between the first and the second, the third and the fourth, i.e. between j and $j+1$ crossover points, are changed. Here j is an odd number of a sorted sequence of crossover points and it is less than the length of the string, i.e. $j < l$. For instance, let the crossover points be 2, 4 and 6. Then multipoint crossover produces two new offspring:

$$P_1 = 00|11|10|11$$

$$P_2 = 11|01|00|10,$$

$$O_1 = 00|01|10|10$$

$$O_2 = 11|11|00|11.$$

The crossover rate, p_c , used in the GAs controls the expected number of chromosomes that undergo the crossover operation. p_c is equal to the ratio between the number of offspring produced in each generation and the total number of chromosomes N in the population. Thus in total $p_c \times N$ chromosomes will undergo the crossover at each generation. If a crossover rate is high more of the solution space can be explored and settling for a false optimum can be avoided. However if the rate is too high computation time might be wasted in exploring unpromising regions of the solution space [62].

Mutation

Mutation is one more genetic operator used in the process of creating offspring chromosomes. It is utilised to recover good genetic material which can be lost during selection or crossover operations [67] and therefore prevents convergence to local optima. For mutation operation, a single chromosome is selected and at some mutation point chosen (at random) the element of the chromosome is modified. If the string is coded in binary then at mutation point the bit changes its state: $0 \rightarrow 1$ or $1 \rightarrow 0$. For example, mutating the third bit in string $P_1 = 00\bar{1}11011$ leads to the new offspring $O_1 = 00\bar{0}11011$.

The mutation operator when one element of the chromosome is changed at a time is called uniform mutation. More variations of the mutation operator are used in the GA. Multiple uniform mutation is described as uniform mutation of n randomly selected elements of the string. This number n is also selected at random and is from interval $[1, 2, \dots, l]$, where l is the length of the string. Employing Gaussian mutation all elements of the chromosome are changed. The rule that is used to change them can be written as:

$$s_k^O = s_k^P + f_k, \quad (3.8)$$

where s_k^O is the k th element of the mutated offspring, s_k^P is the k th element of the parent chromosome and f_k is a random number drawn from a Gaussian distribution [68].

In general, mutation in the algorithm is randomly applied with a probability, i.e. mutation rate, p_m , to the population of new offspring that were created during the crossover operation. Colin in [34] surveyed some different opinions regarding the application of mutation operator in GAs. For example, it was suggested that either crossover or mutation should be applied at each iteration. Another suggestion was to use crossover at the beginning iterations and as chromosomes begin to converge to start using just mutation. Mutation rate usually varies from 0.001 to 0.01 [67].

Replacement

Replacement operator, sometimes called reinsertion, is employed to form a population which becomes a parent population for the next generation of offspring chromosomes. Once the population is formed the sequence of genetic operators is repeated resulting in the new set of offspring solutions. Usually, the number of chromosomes in a parent population is kept fixed throughout the generation process.

A number of strategies exist on how to form a new generation of parent solutions. The basic approach as given in [45] is to utilise the n th generation of offspring solutions as a parent population for the $(n+1)$ th generation of solutions. In this case all chromosomes in the current parent population are replaced with the current offspring chromosomes.

Another replacement strategy is to substitute only certain chromosomes of the current parent population with the offspring solutions. For example, the least fit members of the parent population can be replaced with their offspring [67]. Similarly, the fittest two out of four parents involved in the crossover operation and the fittest two offspring individuals are selected for the new population. The replacement of the parent solutions can also be random.

In this case two children replace two individuals randomly chosen from the entire population [69].

In the steady-state GA [70] individuals from both current parent and offspring populations are first combined into one population. Consider N is the defined size of a population and N_o is the size of an offspring population. Then each individual in the combined population is ranked based on the fitness value. The N_o worst individuals in the ranking are removed and the best N individuals remain in the new parent population for the next generation.

3.6. SUMMARY

The primary goal of the optimisation of a system design in reliability engineering is to find the best way to increase system reliability. A number of accepted principles can be implemented to improve system reliability. However, at the same time, the steps taken will normally consume resources. Thus it is essential to find a balance between system reliability and resources consumption.

Numerical methods for optimisation have been developed extensively having a range of applications in diverse fields. In this chapter the approaches most commonly used in reliability engineering were introduced followed by examples of application. The review suggests that the use of each method has its practical advantages and disadvantages. Some methods are more efficient and accurate than the others. However, none of the methods have proven to be sufficiently superior to the others and the choice of the algorithm needs to be made according to the problem solved.

Most of the reliability optimisation problems are discrete, mixed integer and nonlinear optimisation problems. Due to their complexity, practical application of various approaches can be limited. To summarise:

Application of the linear programming techniques requires the objective and constraint functions to be linear. A problem formulated as nonlinear programming problem can be transformed into a linear problem. However, such transformation results in an increased number of variables and constraints to be treated therefore becomes more difficult in the sense of the computation time and memory space needed.

Most of the nonlinear and discrete techniques are problem orientated. Therefore, since they are designed for solving specific problems, it is difficult to adopt them for solving a wide

range of problems. Moreover, only a few nonlinear algorithms have proven effective when applied to large-scale problems.

The discrete methods require much computational effort to determine an exact optimal solution. Computational complexity of the methods is very high. In addition, the various integer programming techniques do not guarantee that optimal solutions can be obtained in a reasonable time.

Meta-heuristics, which can be used to solve complex discrete optimisation problems, also exhibit some drawbacks. SA involves a lot of computation effort with a large number of function evaluations and tests for solution feasibility. In TB memory structures and memory-based strategies are problem dependant. PSO may fail to find an optimum if the problem to be solved is too complicated and complex. Using GA requires tuning the unknown parameters.

Owing to numerous reports of its successful application, GAs have attracted more attention recently than other heuristic methods in reliability optimization problems. GA has also been chosen as an optimisation technique for this research. The choice was made considering development of the algorithm with its application to solve design optimisation problems for a range of systems, including safety and phased mission systems. GAs are not problem orientated and can be easily adopted to solve different reliability optimisation problems for a range of systems. Furthermore, the objective function of the optimisation problems solved cannot be defined in a closed mathematical form. GAs use an objective function itself and do not require its derivative or other auxiliary quantities. Only values of an objective function are required to represent the search space, which means that the algorithms can incorporate other methods for the evolution of objective function values. The objective of the problem solved was to define the system design that contributes to the minimisation of system failure within the context of pre-defined design constraints and resources. GAs can solve complex and large scale optimisation problems with any kind of non-linear objective functions and constraints defined in discrete, continues or mixed search spaces. The GAs are capable to perform global search and determine global optima. These properties suggest the GAs have a strong potential to determine solutions required in engineering system reliability problems for the work presented here.

4. GENERAL SYSTEM DESIGN OPTIMISATION ALGORITHM

4.1. INTRODUCTION

Safety systems usually installed in safety-critical control systems, such as nuclear power plants, oil platforms or chemical processes, prevent the occurrence of catastrophic consequences caused by a system failure. They have a specific functioning principle, i.e. such systems work on demand. A high likelihood of functioning on demand for a safety system can be ensured by altering its design. For this purpose redundancy techniques can be introduced or certain components may be replaced with ones with better reliability characteristics [71]. However, design alterations and therefore the level of system reliability improvement are usually subject to a number of limiting factors, such as cost or weight. Thus, the problem is to construct a system design that would improve its availability within the constraints imposed on its design.

In this chapter a General System Design Optimisation Algorithm (GSDOA) is introduced that determines an optimal design configuration for a safety system. The objective of the approach is to identify an optimal system design that contributes to the improvement of system availability within the context of pre-defined design constraints and resources. The introduced GSDOA is designed to be applicable to a range of safety systems. The approach combines an optimisation technique with both qualitative and quantitative system analysis methods, such as fault tree analysis (FTA) and binary decision diagrams (BDDs).

Fault trees provide a schematic description of the possible combinations of system conditions involving system components that could lead to system failure [72]. Therefore FTA has been employed as a means to represent system design cases and evaluate their failure probabilities by analysing the failure logic of the system. Fault tree modification patterns (FTMPs) have been developed to provide standardised elements to build a single fault tree representing all possible design cases under consideration which is then modified to identify specific design configurations. A BDD is a directed acyclic graph representing a Boolean Function. The quantitative analysis of fault trees can be performed by transformation into BDDs [6]. The BDD based approach is considered to be a computationally more efficient method. Thus in the design optimisation algorithm the BDD method has been implemented to quantify system failure.

The objective function of the problem solved cannot be defined in the closed mathematical form (see *Section 4.3.1*) and therefore the Genetic Algorithm (GA), a meta-heuristic optimisation technique, has been chosen as the optimisation technique to perform the optimisation part of the approach. GAs are stochastic global search methods which are based on the mechanics of natural genetic variation and natural selection [45]. The principles of the GA allow easy implementation and adaptation of the algorithm according to a solved problem. GAs use a fitness function itself and do not require derivative or other auxiliary quantities. Moreover, only values of an objective function are required to represent the search space which means that the algorithms can incorporate other methods for evolution of objective function values. Finally, GAs easily handle constrained optimisation problems [24].

The chapter is organised as follows. *Section 4.2* details rules on how to transform the fault tree and incorporate causes of failure for all given design alternatives of the analysed system in a single fault tree. The quantification process of the resulting fault tree is also discussed. In *Section 4.3*, the developed algorithm and the programme code are detailed. At the end of the chapter a summary of the developments is provided.

4.2. FAULT TREE MODIFICATION METHODOLOGY

A fault tree provides a schematic description of the possible causes of a specific system failure mode. Each event of the fault tree defines a dynamic change of state of a system element. Thus, if a system design is altered and new components are introduced the resulting fault tree for the new design system will also include new events representing failures of the new system components.

When considering a number of different design cases it is time-consuming to construct and then analyse a fault tree for each individual design. The problem can be resolved by using a fault tree representing all possible design alterations. This idea was first suggested by Andrews and Pattison [73]. A fault tree representing all possible design variations includes house events. The house events are employed to switch on or off different branches of the fault tree to model the causes of system failure for each design alternative. Thus the use of the house events overcomes the need to construct an individual fault tree for every possible design alternative.

In this research the problem of constructing a fault tree comprising all possible designs is extended. General rules which define changes in a fault tree according to the design options

specified for the optimisation problem have been developed. These rules are not adapted to one particular system and they have the potential to be employed to represent all considered possible design variations in one fault tree for any analysed system.

4.2.1. Design Alteration Options

A number of alterations regarding the structure and operation of a safety system influence its performance capabilities. System operation can be influenced by the time taken to maintain the system and how often maintenance actions are taken. Design alterations that can affect system availability are redundancies introduced at component or subsystem levels and also different component-type selections.

By introducing parallel redundancy, system components are duplicated. As a result, the failure of the system occurs if and only if all redundant components fail. It is also possible to introduce so called k -out-of- n redundancy. Here n defines the number of redundant components and k is the number of working components that are needed for successful system operation. In this case a system will be subject to failure if $n-k+1 \leq n$ components fail. If k is equal to 1 then it is equivalent to the simple redundancy case. Both redundancy types can be implemented at component or sub-system level. Considering system design optimisation the problem is to determine the redundancy level necessary subject to the available resources.

It is also possible to improve system availability by replacing a component with another component selected from a group of possible alternatives. Each possible candidate can have different characteristics such as failure rate, cost, weight or time taken for maintenance. The problem in making a decision about candidate suitability appears when the choice between different characteristics of the components needs to be made. For example, a choice needs to be made between a more reliable component which is expensive and a less reliable component which is considerably cheaper.

When considering a system design, an initial design will be specified. Along with this will be the options for alterations. These options are defined as structural design variables for quantification purposes since they define changes being introduced to the initial system design. In this research three parameters associated with structural design changes are utilised, defined as n , k and t and referred to as design variables.

n corresponds to redundancy allocation. For example, consider the possibility to install up to 4 pumps instead of one. The maximum value of n for the structural design variable associated

with the replacement of the pump would be equal to 4. Hence the design consideration is the number of redundant components, n , which can vary from 1 to 4.

As it was mentioned earlier, another option to change the system is to use k -out-of- n redundancy. If k as a design variable parameter is not defined its default value is equal to 1. Otherwise, it is assumed that it can be equal to any whole number in the interval $(1, n)$. It provides the choices of different k -out-of- n redundancy requirements.

The final design change considered in this study is where a different component type selection is implemented. The parameter t is used to identify a possible component type from a number of possible options. It is also possible to introduce a selection option of a component type for new redundant components.

4.2.2. Fault Tree Modification Patterns

4.2.2.1. Overview

The fault tree comprising all design alternatives is constructed using modification rules which are implemented in the fault tree of the initial system design. The initial system design fault tree, as well as the resulting one comprising all design alternatives, are considered to be coherent fault trees and to have two types of gates, AND and OR.

The rules for building a fault tree with all design alternatives implemented are defined as FTMPs. A FTMP defines a fault tree part representing all possible design variations introduced for the replacement of one chosen component. When solving a system design optimisation problem, a number of system components can be chosen to be replaced and therefore several FTMPs will be applied.

The fault tree part introduced with the FTMP incorporates groups of new basic events linked together by house event(s). Groups of house events corresponding to different FTMPs are independent from each other. Having general numbering rules for gates, basic events and house events introduced in the resulting fault tree structures enables the implementation of FTMPs in the fault tree of any system. Specific numbering rules exist for each FTMP.

During the analysis phase the fault tree that has been built using FTMPs is modified to represent particular designs. Values of house events introduced with the FTMP are set to alter the corresponding fault tree part by switching certain branches on and off so that only one possible design alternative is modelled. This is repeated for each FTMP implemented.

Each modification pattern can represent either one or two or three structural design variables and is defined by the following three parameters. mn is the maximum possible number of redundant components, i.e. the largest value of a design variable n ($n=1,2,\dots,mn$). mt is the maximum number of possible different component types or the largest value of a structural design variable t . Parameter mk defines a redundancy type and corresponds to design variable k . There are five FTMPs defining all possible component replacement cases. Each replacement is possible at both component and subsystem level. A FTMP is identified according to the values of three parameters mn , mt and mk :

Pattern 1: $mn > 1, mt = 1, mk = 1$; (parallel redundancy);

Pattern 2: $mn > 1, mt = 1, mk = mn$; (k -out-of- n redundancy);

Pattern 3: $mn = 1, mt > 1, mk = 1$; (type change);

Pattern 4: $mn > 1, mt > 1, mk = 1$; (parallel redundancy and type change);

Pattern 5: $mn > 1, mt > 1, mk = mn$; (k -out-of- n redundancy and type change);

Here the value of the parameter mk is either equal to mn , representing a k -out-of- n redundancy of components where $k \leq n$, or it is equal to 1 and represents a parallel redundancy.

Application of each pattern is discussed in more detail in the following sections (Sections 4.2.2.1 –4.2.2.5). The fault tree for the simplified fire protection system (Figure 4.1) is used as an example. The pump has been chosen as a replaceable component. Therefore, the corresponding basic event ‘Pump Fails’ is replaced with a new fault tree structure for every pattern introduced. When making fault tree alterations at event level one common rule applies to all patterns, that is an OR gate replaces the chosen basic event. The gate is numbered as $G_{max}+1$, where G_{max} defines the maximum gate number in the initial fault tree. Note that in all discussions the numbers in the gate symbols in the fault trees are used as gate reference numbers.

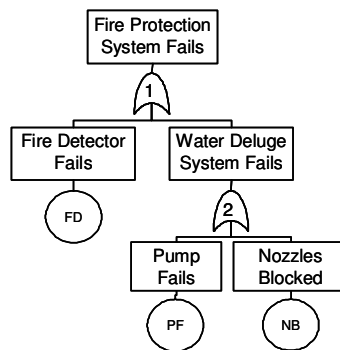


Figure 4.1. Fire Protection System Fault Tree

4.2.2.2. Parallel Redundant Elements (*Pattern 1*)

Pattern 1 defines the fault tree structure comprising possible design alternatives representing different numbers of redundant elements. For example, a chosen component can be replaced with up to mn redundant elements. In the resulting new part of the fault tree mn house events will be incorporated. Every house event is coupled with the corresponding fault tree structure providing the possible combinations of the causes of failure for i redundant components leading to the system failure. In other words, each house event indicates a design case where i ($i=1, 2, \dots, mn$) redundant elements are fitted in the system. It is also acceptable that the basic event chosen to be replaced remains incorporated in the new fault tree structure. It specifies failure of the component which can be referred to as component number 1 in the group of redundant components. The flowchart of the algorithm for implementation of *Pattern 1* is shown in *Figure 4.2*.

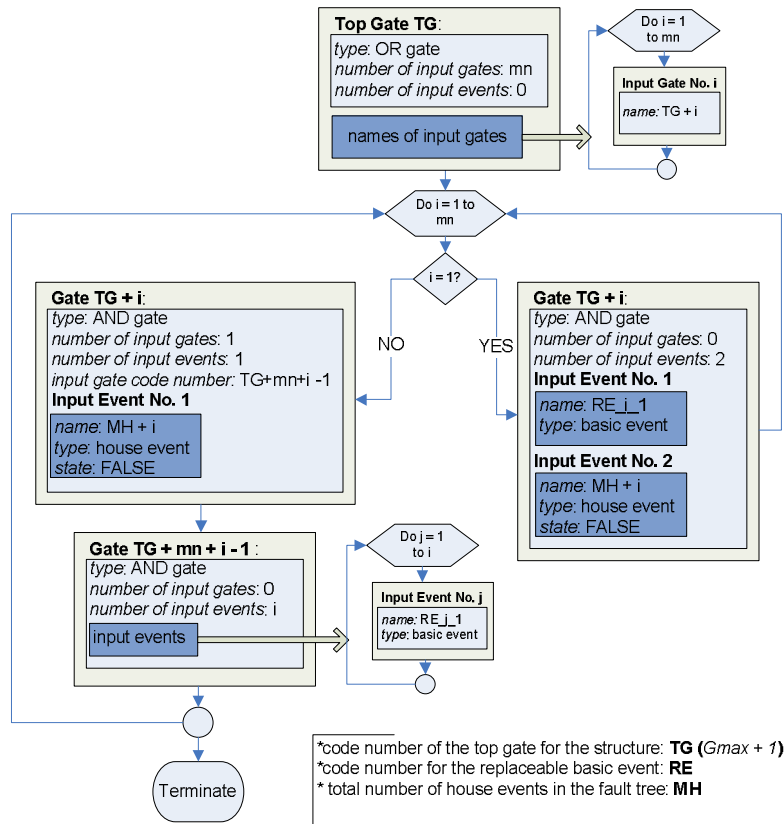


Figure 4.2. Flowchart for the algorithm of *Pattern 1*

The links between the house events introduced in the resulting new structure of the fault tree are explained through the analysis of the application of *Pattern1* for the simplified fire

protection system. In this illustration it is considered that the existing pump can be replaced with either 2 or 3 redundant pumps. The option to keep only one pump is also possible. Thus in order to construct the fault tree for the system with 3 design alternatives *Pattern 1* is employed where $mn = 3$. The obtained fault tree is shown in *Figure 4.3*.

As seen in *Figure 4.3*, a new OR gate replaces the chosen basic event. The original basic event remains in the fault tree. Thus, employing *Pattern1* $mn-1$ new basic events are introduced. ‘*Pump 2 fails*’ and ‘*Pump 3 fails*’ are the two new events for the case analysed. Correspondingly three house events are also introduced. The state of each house event is to be defined so that only one possible design alternative is represented. The following rule is used for the *Pattern1*:

if $iHE = TRUE; (i = 1,2,\dots,mn)$

then $jHE = FALSE; (j = 1,2,\dots,i-1,i+1,\dots,mn).$

Here iHE and jHE defines the i th and j th house events respectively. For instance, if in the given example the house event 2PF is set to TRUE then the house events 1PF and 3PF are in the FALSE state, which means the resulting fault tree defines the causes of failure for a fire protection system which has two redundant pumps fitted.

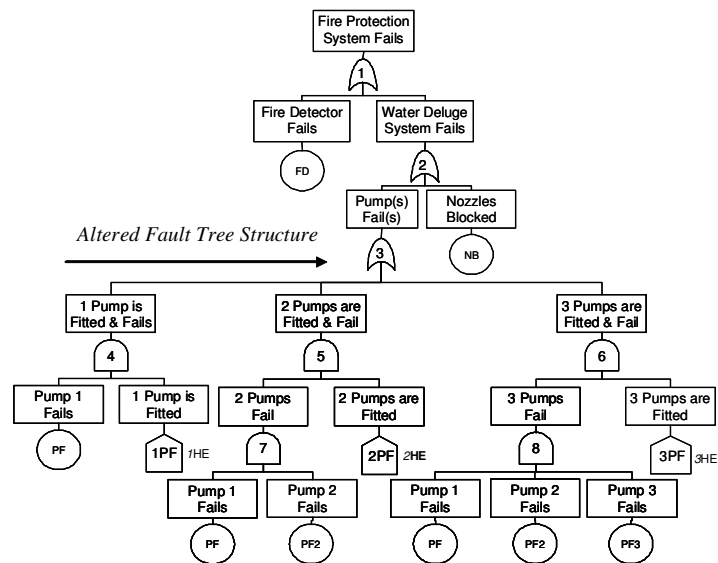


Figure 4.3. *Pattern 1* Application Example for Fire Protection System

4.2.2.3. *k-out-of-n* Redundancy (Pattern 2)

Pattern 2 is similar to *Pattern 1* in the way that it is employed to alter the initial fault tree in replacing one system element with a number of redundant elements. However, in this case *k-out-of-n* redundancy is used where k and n are both design variables, i.e. not defined *a priori*. It means the resulting fault tree represents all design alternatives for all possible combinations of n and k values. The flowchart of the algorithm for *Pattern 2* is shown in *Figure 4.4*.

In this case two groups of linked house events are introduced. Assuming that up to mn redundant components are introduced, each house event from the first group indicates a design case where i ($i = 1, 2, \dots, mn$) elements are fitted in the system. The second group of house events is used to define the minimum number of possible failures of the redundant elements causing the system failure (j), i.e. $j = mn - mk + 1, mk = 1, 2, \dots, mn$. The new fault tree structure also has $mn - 1$ new basic events plus the initial replaceable basic event.

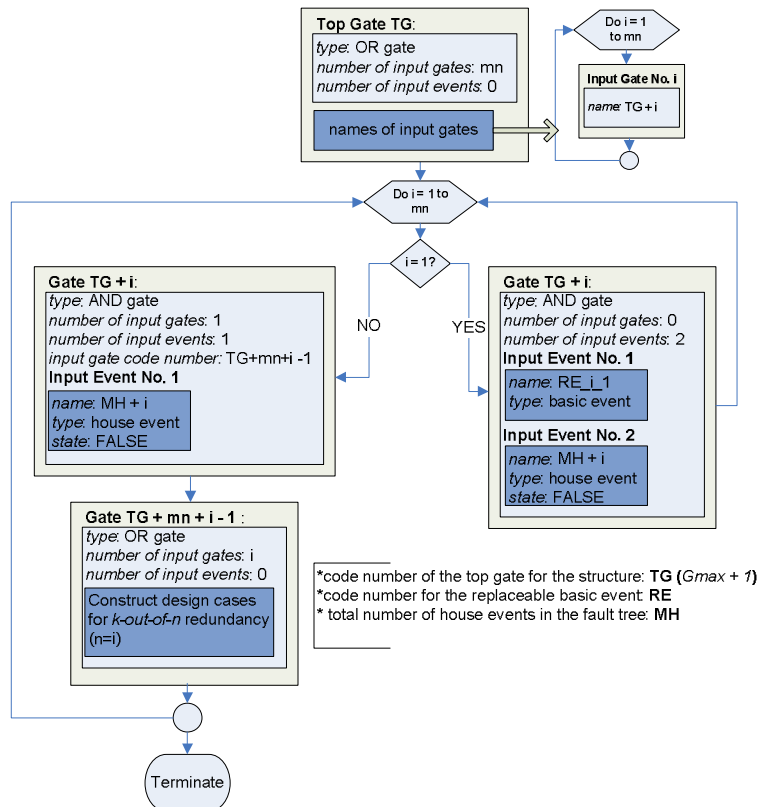


Figure 4.4. Flowchart for the algorithm of *Pattern 2*

To illustrate the case, the Fire Protection System example is considered where up to two redundant pumps can be introduced and *k-out-of-2* redundancy can be implemented. The fault

tree representing the causes of failure for all possible design alternatives is given in *Figure 4.5*. The house event *IPF (1HE)* set to TRUE defines the case when only one pump is used. Accordingly the house event *2PF (2HE)* in a state TRUE is associated with branches of the fault tree modelling the causes of the system failure when two pumps are fitted. The house event *PR (2_1HE)* from the second group is introduced to represent the case when $j = 2$, i.e. when *1-out-of-2* or parallel redundancy is used. Accordingly, the house event *k2R (2_2HE)* identifies the design case where 2 pumps are required to function for successful operation of the system, i.e. where $j = 1$ and failure of either pump causes the Fire Protection System failure.

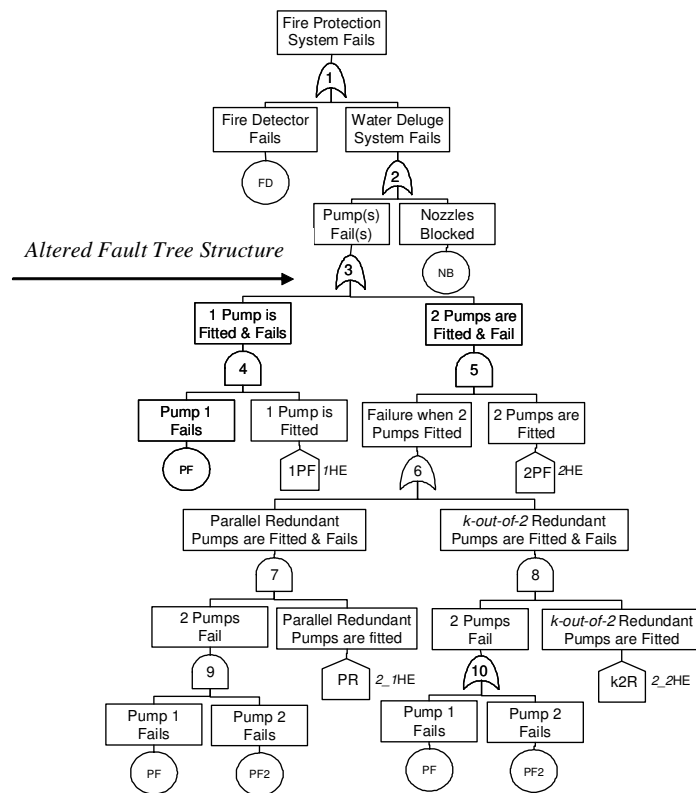


Figure 4.5. Pattern 2 Application Example for Fire Protection System

The rule defining links between house events in the first group is the same as the one used for the *Pattern1*. The rule defining the states of the house events in the second group is as follows:

if $i_kHE = \text{TRUE}, (i_k = 1, 2, \dots, i)$

then $i_lHE = \text{FALSE}, (i_l = 1, 2, \dots, i_k-1, i_k+1, \dots, i)$ and

$$j_IHE = \text{FALSE}, (j_l = 1, 2, \dots, j; j = 1, 2, \dots, i-1, i+1, \dots, mn)$$

Here i_kHE and i_lHE define the corresponding k th and l th house events from the second group which are linked with the i th house event from the first group. Similarly, j_lHE is the l th house event from the second group which is linked with the j th house event (set to state FALSE) from the first group of house events. According to these rules only two house events can be in a TRUE state while the rest of them are set to FALSE.

4.2.2.4. A Different Component Type Selection (Pattern 3)

Pattern3 is used to construct a fault tree structure where a different component type selection is implemented. A variable mt associated with this pattern identifies the number of possible component types. Each of the mt incorporated house events is coupled with a basic event. The basic event indicates failure of the component with specific characteristics indicating the type of the component. There are also $mt-1$ new basic events in the fault tree, since the basic event chosen to be replaced is incorporated in the new fault tree structure and defines failure for the component of type 1. The flowchart of the algorithm for implementing the choice for a different type component is shown in *Figure 4.6*.

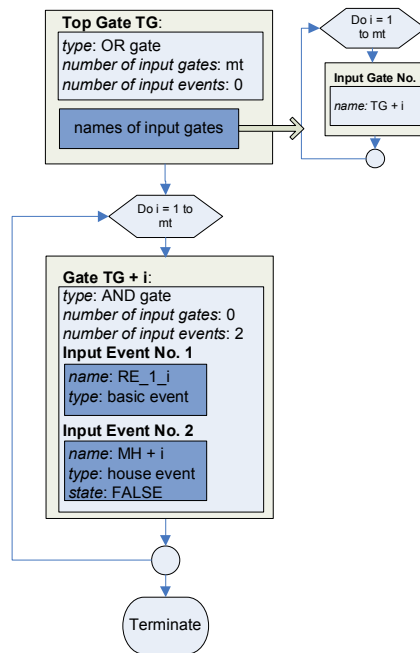


Figure 4.6. Flowchart for the algorithm of *Pattern 3*

The pattern with $mt=3$ applied for the replacement of the pump in the example Fire Protection System introduces the fault tree where failure causes for three design cases are incorporated (Figure 4.7).

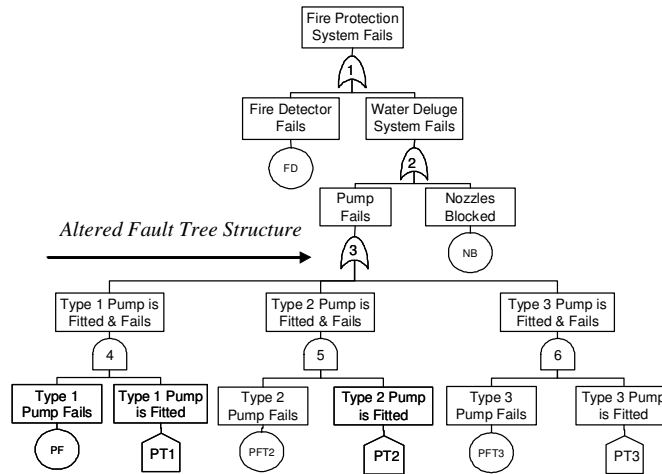


Figure 4.7. Pattern3 Application Example for Fire Protection System

The rule for defining the links between the introduced house events is identical to the one used for *Pattern1*. If one of mt house events is set to TRUE the rest of them have a state FALSE. For example, if the house event $PT3$ is set to TRUE then the rest of the two house events $PT1$ and $PT2$ are defined as FALSE. For this case the resulting fault tree represents the Fire Protection System where a type 3 pump is installed.

4.2.2.5. Selection Option of a Component Type for New Redundant Components

(Pattern4 and Pattern5)

When considering design changes it is possible to introduce redundant components where a selection option for a component type also exists. Since there are two redundancy types two possible FTMPs exist that can be used for the fault tree structure alterations regarding both redundancies and component type selections. *Pattern4* is employed when parallel redundant components and choices for each component type are considered. *Pattern5* is used to alter the fault tree in the matter of k -out-of- n component redundancy added together with the possibility to choose each component type.

Pattern4 combines two FTMPs, *Pattern1* and *Pattern3*, where mn represents the maximum possible number of parallel redundant components introduced and mt identifies the maximum number of possible different component types for each redundant component. There are two groups of house events introduced. The first group comprises mn and the second group has mt house events. The i th ($i=1,2,\dots, mn$) house event from the first group is associated with the

possible failure causes for the system design case when i redundant components are used. The failure event of each introduced component is then associated with the second group of house events resulting in t sets comprising one basic event coupled with one house event. Here each basic event indicates the failure of a specific type of component. The latter linking of house events with basic events is analogous to the replacement of each earlier introduced basic event when using *Pattern3*. The resulting fault tree structure also comprises $(mt \times mn) - 1$ new components. As in the previous cases the basic event chosen to be replaced represents the failure for component number 1 with type 1. The example of the application of *Pattern4* is shown in *Figure 4.8*.

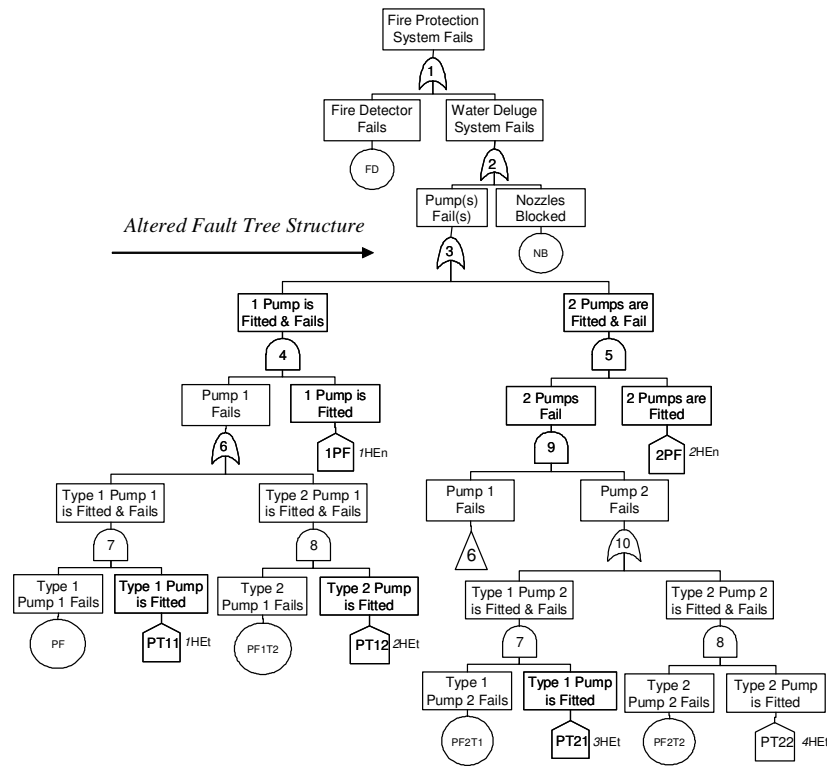


Figure 4.8. *Pattern4* Application Example for Fire Protection System

For the case analysed up to two redundant pumps can be installed and the possibility to choose one of two pump types is implemented. Thus there are two house events associated with the possible number of pumps to be installed, $1PF$ ($1HE_n$) and $2PF$ ($2HE_n$). Four house events $PT11$ ($1HE_t$), $PT12$ ($2HE_t$), $PT21$ ($3HE_t$) and $PT22$ ($4HE_t$) are also introduced which are associated with the choice of a type for any pump installed. Three new basic events are incorporated in the fault tree, which are ‘Type 2 Pump1 Fails’, ‘Type 1 Pump 2 Fails’ and ‘Type 2 Pump 2 Fails’.

The two groups of house events are independent. It means rules for assigning either TRUE or FALSE values for house events within the group apply for each group individually. For the general case a particular design is defined by assigning values for the house events as follows:

if $iHE_n = \text{TRUE}; (i = 1, 2, \dots, mn)$

then $jHE_n = \text{FALSE}; (j = 1, 2, \dots, i-1, i+1, \dots, mn).$

if $iHE_t = \text{TRUE}; (i = 1, 2, \dots, mt)$

then $(i+mt*k)HE_t = \text{TRUE}, (k = 1, 2, \dots, mn-1)$

and $jHE_t = \text{FALSE}; (j = 1, 2, \dots, mt*mn, j \neq i+mt*k, k = 1, 2, \dots, mn-1).$

Here iHE_n and jHE_n define the i th and j th house events from the first group of house events. Accordingly iHE_t and jHE_t are used to define the i th and j th house events associated with the choice of component type.

Note that once one of the sets of the house events implementing the choice of component type are assigned their values the house events in the remaining sets will have the same values assigned. It means the redundant components will be of the same type for any level of redundancy. For example, for the Fire Protection system the house events are assigned values as follows; $2PF = \text{TRUE}$ ($2HE_n$) and $PT11 = \text{TRUE}$ ($1HE_t$) therefore $1PF = \text{FALSE}$ ($1HE_n$) and $PT21 = \text{TRUE}$ ($3HE_t$), $PT12 = \text{FALSE}$ ($2HE_t$), $PT22 = \text{FALSE}$ ($4HE_t$). As seen in *Figure 4.8*, the resulting fault tree represents a system where two redundant pumps of type 1 are installed.

Pattern5 as well as *Pattern4* is also equivalent to a combination of two FTMPs. The resulting fault tree structure determined by *Pattern5* could be defined by firstly applying *Pattern2* and then again altering the obtained fault tree a number of times using *Pattern3*. However the use of only *Pattern5* is simpler and straightforward.

The implementation of *Pattern5* is very similar to the one of *Pattern4*. Indeed, the only existing difference between the two is that the fault tree structure defined by *Pattern5* corresponds to a system with k -out-of- n redundancy. Thus the new structure defined by the pattern also has two groups of house events. The first group comprises events to model k -out-of- n redundancy cases and the second corresponds to the choice of a type for a redundant component. These two groups of house events are implemented as being independent. Therefore the rules used for assigning values for house events from the first group are identical to the ones used for *Pattern2* and the rules employed for *Pattern3* are used for house

events from the second group. An example of the application of the pattern is given in Figure 4.9 where up to two pumps can be installed into the Fire Protection System and each of them can be either type1 or type 2.

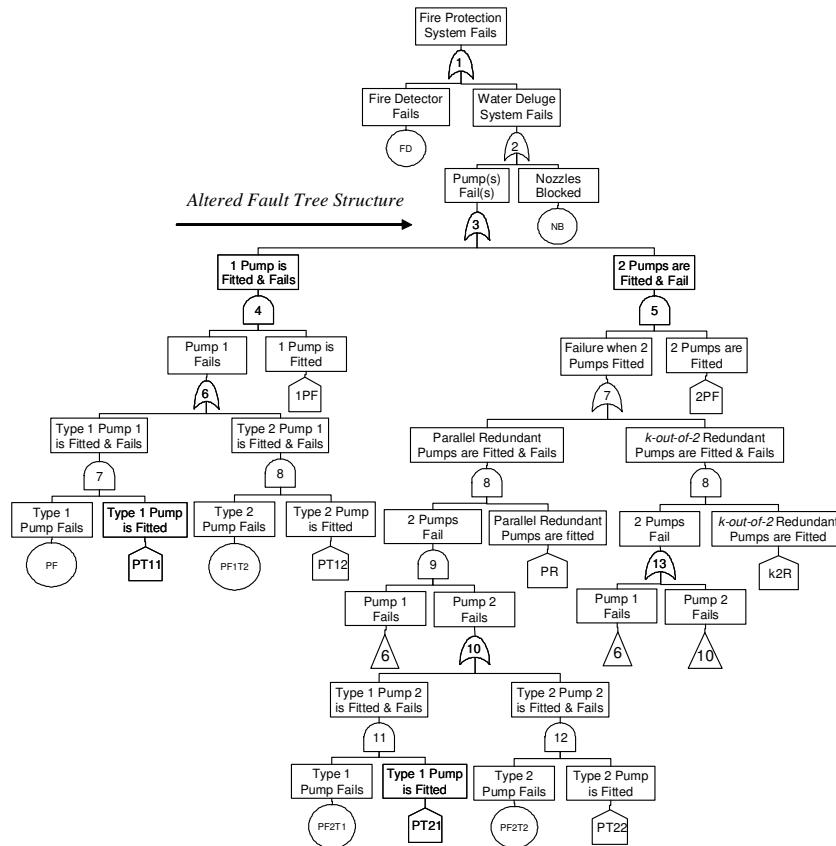


Figure 4.9. Pattern5 Application Example for Fire Protection System

4.2.2.6. Fault Tree Alteration at Gate Level

The question is posed: when can it be useful to apply FTMPs at gate level? Assume a situation when a pumping device is considered to be replaced with n redundant pumping devices where each of them is comprised of a pump and a switch. The pumping device fails if either the pump or the switch fails. It means the failure of the pumping device is described with an *OR* gate and two input basic events. Application of FTMPs for the pump and switch replacements individually would be complicated. However, if the failure of a pumping device is considered as a simple event then the fault tree comprising all design alternatives with different numbers of redundant pumping devices can be built directly applying *Pattern1* at the *OR* gate level.

As an example, the Fire Protection system is shown in *Figure 4.10*. In this case a redundant Water Deluge subsystem is introduced. *Pattern1* is applied for AND gate 2. The initial subsystem comprises of one pump and a nozzle. Thus after introducing a redundant Water Deluge subsystem two new basic events appear in the fault tree. They are input events of the output event ‘Water Deluge subsystem 2 fails’ and this fault tree part is considered as a new intermediate event. The structure of the new part is a reproduction of the initial fault tree structure for failure of the Water Deluge subsystem. Considering the replaceable fault tree structure as a simple event, only two house events are needed to be incorporated in the new fault tree. When one of them is set to TRUE and another one is set to FALSE the resulting fault tree represents either the system with one Water Deluge subsystem or the system with two parallel redundant Water Deluge subsystems.

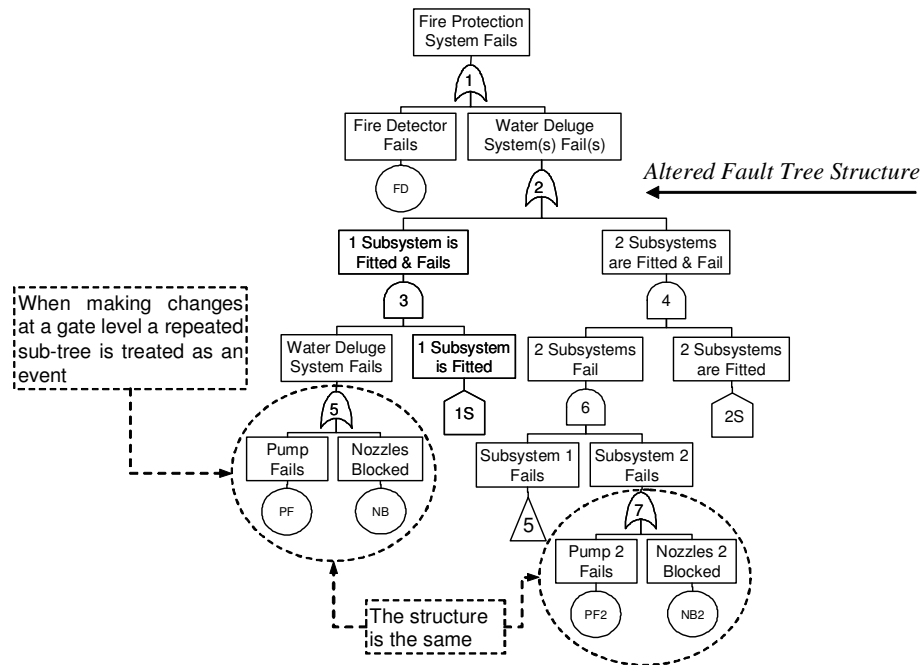


Figure 4.10. Example of Pattern 1 Application at Gate Level for Fire Protection System

The remaining four FTMPs can be employed at gate level in the same manner as the discussed *Pattern1* when a changeable part of a fault tree is treated as a simple event. In this case the number of introduced house events will be equal to the number of house events when FTMPs are applied at an event level. The same rules defining links between house events will be valid. The number of new simple events, i.e. fault tree parts having an identical structure, will be the same as the number of new basic events when applying FTMPs at event level.

4.3. QUANTITATIVE FAULT TREE ANALYSIS

When performing the quantitative system analysis for different design cases each design needs to be analysed individually. For this purpose the fault tree representing causes of failure for all possible design cases is utilised. House events in the fault tree are assigned the appropriate Boolean states to specify the fault tree structure for a particular design case. Since the value of the house event is specific to the design in question it must be specified prior to each individual design quantification.

To set values for house events each fault tree part constructed according to a specific FTMP is analysed individually. The house events are assigned either TRUE or FALSE values according to the rules specific to the FTMP. As a result the fault tree structure represents causes of failure of one design alternative defined by the FTMP.

The fault tree with house events assigned with specific values undergoes a trimming operation before its quantification process. Branches with house events set to a FALSE state are cut off resulting in a fault tree of an individual design case for the analysed system. Trimming allows the size of the fault tree to be greatly reduced which results in much faster calculations. It is especially useful when a large number of design variables is used and / or their range of possible values is large or when alterations are made at sub-system level.

After the trimming operation has been performed the resulting fault tree is converted to its BDD. Since the obtained fault tree has a simple structure a standard BDD approach is employed to calculate the top event probability, i.e. the probability of the failure of the specific system design. As it is known, conversion of the fault tree to the BDD format enables the exact system unavailability to be found in a computationally more efficient way. The BDD methodology for quantitative fault tree analysis was discussed in detail in *Chapter2*

4.4. OPTIMISATION ALGORITHM

4.4.1. Mathematical Problem Concept

The objective of the process of safety system design construction and / or design alteration is to determine an acceptable system design. The criteria for evaluation of the designs acceptability can consist of system requirements based on its reliability, cost, weight, physical size, etc. The most important feature of a safety system is that it works when the demand arises. Therefore, in the developed approach system unavailability, i.e. the probability of

system failure on demand, has been chosen to provide a measure of system performance for different design alternatives. A small value of the probability of system failure identifies a highly reliable system design. Other design requirements are considered within the context of pre-defined constraints.

The system design optimisation problem is analysed as a general single objective constrained optimisation problem. The merit function of the problem is the probability of system failure that is to be minimised:

$$\min Q(\mathbf{X})_{\text{sys}} \quad (4.1)$$

Here $Q(\mathbf{X})$ is a function defining system failure probability and \mathbf{X} is a m -dimensional vector $\mathbf{X} = \{x_1, x_2, \dots, x_m\}$ where its element x_i is the failure probability value of a basic event i , i.e. system component i . The vector dimension m is equal to the number of basic events, i.e. system components subject to failure, in the fault tree for a specific design case. Thus if the number of system components varies for different design cases the content of \mathbf{X} is adjusted to the changes. It follows that the optimisation objective to minimise system unavailability is equivalent to the objective to find the vector \mathbf{X} that corresponds to the minimum system unavailability. The given objective function does not have an explicit form. As such, the Fault Tree Modification Methodology and FTA discussed in Sections 4.2 and 4.3 are used to quantify the system unavailability of each potential design.

When developing a system design, considerations could also be given to other factors that influence the design criteria, additional to the used failure probability. Typically considered design requirements include system cost, physical size, power consumption, *etc.* In the approach developed for a general application it was decided to implement the possibility to set limitations to system cost, weight and volume (physical size). To use the resources efficiently it may be useful to have minimum and maximum limitation values. If only maximum limit values are needed then the minimum values become equal to zero. Thus the general system design optimisation problem is subject to the following constraints:

$$\begin{aligned} Cost_{\min} &< Cost_{\text{sys}} < Cost_{\max} , \\ Weight_{\min} &< Weight_{\text{sys}} < Weight_{\max} , \\ Volume_{\min} &< Volume_{\text{sys}} < Volume_{\max} , \end{aligned} \quad (4.2)$$

where $Cost_{\min}$ predefines a minimum possible cost for a system design, accordingly $Cost_{\max}$ is the maximum possible cost for the design being constructed and $Cost_{\text{sys}}$ is the actual cost for

the system design in question. Variables for weight and volume constraints have respective definitions.

Since a safety system works on demand, the time taken to maintain the system influences its availability. Therefore the possibility to define limits for minimum (MDT_{\min}) and / or maximum (MDT_{\max}) maintenance down time is also implemented in the algorithm:

$$MDT_{\min} < MDT_{\text{sys}} < MDT_{\max} \quad (4.3)$$

4.4.2. Evaluation of Design Requirements

The total cost of a system differs for each individual system design. Every time a possible system design is chosen its cost needs to be recalculated. In the proposed approach two main types of system cost are considered: design cost and maintenance cost. Design cost can include all costs associated with a system design, i.e. cost of a component, storage cost *etc.* The maintenance cost of the system can be divided into three categories: maintenance test cost, corrective cost and maintenance preventive cost. The generalised formula for system cost evaluation introduced in the optimisation approach is as follows:

$$Cost_{\text{sys}} = Cost_D + Cost_M, \quad (4.4)$$

where $Cost_D$ defines the total system design cost and $Cost_M$ is associated with the cost assigned for system maintenance.

The total system design cost for a specific design case is found by summing the design cost of each component as follows:

$$Cost_D = \sum_{i=1}^m cost_d_i. \quad (4.5)$$

Here $cost_d_i$ is the design cost of a component i , m is the total number of components representing the system design case analysed.

The total system maintenance cost per examined time period for a specific design alternative is evaluated as the sum of the system test cost, preventive maintenance cost and corrective maintenance cost:

$$Cost_M = \sum_{i=1}^m cost_t_i + \sum_{i=1}^m cost_p_i + \sum_{i=1}^m cost_c_i . \quad (4.6)$$

Here $cost_t_i$ is the cost of maintenance testing for component i , $cost_p_i$ is the preventive maintenance cost of the component and $cost_c_i$ is the corrective maintenance cost.

The cost of either testing or maintenance for each component per examined time period is determined by multiplying the number of tests and/or amount of maintenance carried out during the examined period by the cost of a single testing or maintenance. Both maintenance testing and preventive maintenance are performed at regular defined intervals. Corrective maintenance is performed when demand arises, i.e. when system failure occurs. Thus to find the cost of maintenance testing for a component i the following formula is employed:

$$cost_t_i = \frac{U_T}{\theta_{T_i}} c_t_i . \quad (4.7)$$

Here U_T is the total number of time units per examined time period, θ_{T_i} corresponds to the interval between two tests for component i and c_t_i is the cost of a single maintenance test. The time units defining U_T , and θ_{T_i} are the same.

Analogically, the preventive maintenance cost for component i can be evaluated:

$$cost_p_i = \frac{U_T}{\theta_{P_i}} c_p_i , \quad (4.8)$$

where θ_{P_i} is the time interval between two preventive maintenance activities and c_p_i represents the cost of a single preventive maintenance for the component. θ_{P_i} has the same time units as θ_{T_i} .

To find the cost of corrective maintenance for component i an expected number of failures occurring during the examined period needs to be identified. If the number of expected failures, W_i , is known the following formula can be used:

$$cost_c_i = W_i c_r_i \quad (4.9)$$

Here c_r_i represents the cost of a single repairs for component i .

If the number of expected failures is not given it can be evaluated using an appropriate formula. For component i which has constant conditional failure intensity λ_i the number of expected failures over the analysed period can be found using the following formula:

$$W_i(0, U_T) = \lambda_i(1 - q_i)U_T. \quad (4.10)$$

Here q_i is a failure probability of the component and U_T is the total number of time units in the examined time period.

If the component failure rate is defined by the Weibull distribution the expected number of failures per examined time period is:

$$W_i(0, U_T) = \frac{U_T}{\theta_{p_i}} \int_0^{\theta_{p_i}} \frac{\beta_i}{\eta_i} \left(\frac{t}{\eta_i} \right)^{\beta_i - 1} (1 - q_i) dt, \quad (4.11)$$

where θ_{p_i} is the preventive maintenance interval of the component i presented using the time units as for $U_T \cdot \beta_i$ and η_i are Weibull distribution parameters, t denotes time and q_i is the failure probability of component i .

System weight and volume for different design cases may also differ. New components introduced to the system will alter the overall system characteristics values. Thus if each component weight is given the total system weight can be evaluated as their sum:

$$Weight_{sys} = \sum_{i=1}^m w_i, \quad (4.12)$$

where w_i denotes the weight of component i and m is the total number of components fitted for the design case analysed.

The equivalent formula is used to evaluate the overall system volume where v_i is the volume parameter calculated for component i :

$$Volume_{sys} = \sum_{i=1}^m v_i \quad (4.13)$$

The second type of limitation that can be introduced is system maintenance down time. The following formula (*Formula 4.14*) is employed to find the total system maintenance down time:

$$MDT_{sys} = \sum_{i=1}^{N_{\theta}} \frac{U_T}{\theta_i} \left(\sum_{j=1}^{N_{c_i}} T_j \right) \quad (4.14)$$

where N_{θ} is the number of different maintenance test intervals, θ_i corresponds to the test interval, U_T is the total number of time units per examined time period, N_{c_i} defines the number of system components which are tested at the same time interval and T_j is the test time for each system component.

4.4.3. Genetic Algorithm Characterisation

A single objective GA has been chosen as the optimisation technique to solve the system unavailability minimisation problem. The implemented GA is summarised by the flowchart in *Figure 4.11*. Each stage of the algorithm is discussed in detail in the following sub-sections.

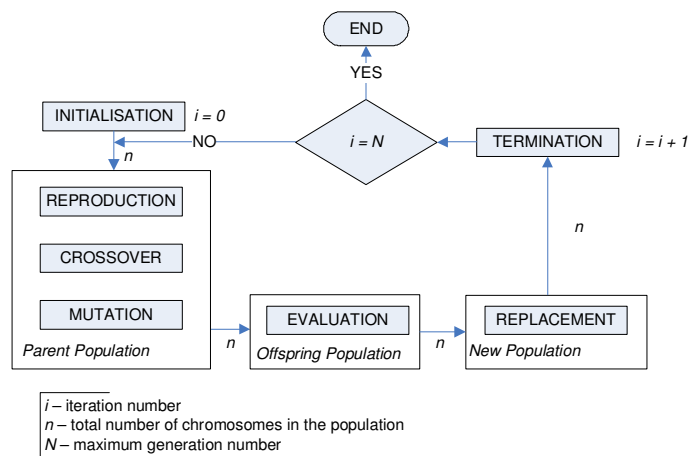


Figure 4.11. Structure of the Implemented SOGA

4.4.3.1. Chromosome Encoding

If applying a GA, a problem specific representation is needed to describe each chromosome of the population. In a general case, a chromosome encodes decision variables of the optimisation problem. In the proposed case, in *Equation 4.1*, failure probability values of system components compose the vector of the decision variables \mathbf{X} . When using a fault tree representing all possible system design alternatives and FTA to evaluate the objective function of the problem, the dimension of vector \mathbf{X} varies for different system design cases. If

coding these variables as genes the resulting chromosome structure will vary throughout the optimisation process. It will result in a rather complicated implementation of the GA, which would not necessary guarantee an efficient optimisation process.

The problem of the variable length chromosomes has been avoided by using chromosomes encoding parameters of FTMPs instead. As mentioned earlier, during the optimisation process structural design variables are assigned various specific values resulting in the variable dimension of vector \mathbf{X} . Since the set of components which are not considered to be replaced is the same for every design case, only sets of components introduced after the implementation of structural design changes are needed for identification of a specific design configuration case. Therefore the search for the optimal design which minimises system failure probability can be governed by manipulating only the values of structural design variables. Since the design variables are represented by FTMPs which define their maximum values, the structure of a chromosome is defined based on parameters of FTMPs. The number of FTMPs remains the same throughout the whole optimisation process which means that a fixed length chromosome is utilised for the problem analysed.

Binary numbers are used for the encoding of each variable. In defining the structure of a chromosome, first parameters of FTMPs (mn , mt and mk) are allocated a particular number of bits required to code their values. The bits associated with one parameter represent a gene. To evaluate the size of the gene *Formula 3.4* is used. Since the introduced values for parameters of FTMPs define the maximum range limit of an associated design variable(s), a parameter with the introduced value equal to 1 is omitted and no bits are allocated to it in the chromosome. Next, a required number of bits are allocated to code the maximum possible values of parameters which are not associated with the fault tree modifications, such as maintenance test intervals. The number of genes in the chromosome is increased by the amount which is equal to the number of different maintenance intervals. Such allocation of bits ensures that the size of a chromosome is sufficient to store any values of design variables and its size remains constant throughout the optimisation process. Thus the algorithm creates an individual fixed size chromosome for a specific problem analysed.

As an example, three FTMPs are considered. Patterns are given in the following order: *Pattern1* ($mn = 2$, $mt = 1$, $mk = 1$), *Pattern3* ($mn = 1$, $mt = 3$, $mk = 1$) and *Pattern5* ($mn = 4$, $mt = 3$, $mk = 4$). In order to encode the possible levels of parallel redundancy introduced with *Pattern1* in a binary format two bits are required. Thus the first two bits are allocated in the chromosome for the parameter of *Pattern1*. In this case $mt = 1$ which means that no changes

are made regarding the choice of a different type for the component. As such, no bits are allocated in the chromosome. Next, two bits in the chromosome are allocated to code the parameter $mt = 3$ for *Pattern3*. Finally, a number of bits required to code parameters of *Pattern5* are allocated. To code number 4 in binary digits three bits are required. Therefore the first three digits are allocated for parameter mn . A two bit length gene is allocated to code the value of the variable mt and the next three digits are assigned to parameter mk . The final structure of the chromosome is shown in *Figure 4.12*.

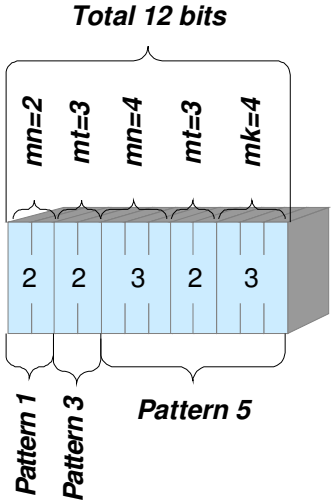


Figure 4.12. The Chromosome Structure

4.4.3.2. Population Manipulation

In the proposed algorithm an initial population is generated as follows:

Step 1: Each gene in a chromosome is assigned a random binary number.

Step 2: The chromosome is decoded, i.e. a binary number assigned to each gene is converted into a decimal number, to assign values of parameters for the design variables. At the same time it is checked to see if the obtained phenotype of each gene, i.e. the decoded value of each gene, does not exceed the predefined maximum value. If the generated value is bigger than the maximum possible value of the particular parameter then the gene is assigned a new binary number and its validation is checked again. The process is repeated until all the generated parameter values are valid.

Step 3: A system design fault tree is reconfigured according to the obtained parameter values of the design variables.

Step 4: System resources and system failure probability are evaluated for the design generated.

Step 6: The generated chromosome enters the initial population.

Step 7: If the number of chromosomes in the population is equal to the predefined value N the process is terminated. Otherwise the process is repeated from *Step 1*.

The initial population is considered as the first generation parent population and is used to generate a new generation offspring population. Each generation offspring population is created using three main GA operators. The reproduction operator is implemented employing a biased roulette wheel. The operator is used to select $N/2$ couples of parent strings entering into a mating pool. Strings of each couple are crossed over employing a one-point crossover operator. During the crossover process, a bit-by-bit mutation is also carried out. All the mentioned GA operators were discussed in detail in *Chapter 3*.

Each time a pair of parent chromosomes are crossed over and mutation is implemented two new strings are created. Before adding the new chromosomes into an offspring population each gene is decoded to check the generated values for the corresponding design parameters. In some cases the same number of bits allocated for one parameter value can represent a larger decimal number than the given parameter value. For example, in order to code *Pattern1* parameter $mm=4$ three bits need to be allocated in the chromosome. However decimal numbers 5, 6 and 7 can also be coded using three bits. To avoid situations when a chromosome with unfeasible genes enters into a population, a chromosome repair is carried out. If a gene value is outside of the range a new binary number is generated for the corresponding gene and its validation is checked again.

4.4.3.3. Replacement

The replacement procedure is implemented to preserve both elite and feasible chromosomes. After an offspring population is generated each pair of offspring chromosomes is compared with their parent chromosomes and a new parent population for the next generation is formed. The following outcomes of the comparison are possible:

1. If both offspring are infeasible parent chromosomes enter in the new parent population.
2. If one of the offspring chromosomes is feasible the fitness value that it yields, i.e. the measure of its fitness based on the objective function value, is compared with the fitness value of each parent chromosome. In this case two out of three chromosomes which have the

smallest fitness values are stored in the new parent population. Both the chromosome with the largest fitness values and the infeasible offspring chromosome are discarded.

3. If offspring chromosomes are feasible then both of them enter the new parent population.

4.4.4. Optimisation Algorithm Structure

This section provides an overview of the GSDOA. This algorithm has two major parts. The first part is considered as a preparative part for the optimisation process. At this stage all proposed design alternatives are introduced for the alteration of the initial system design using appropriate FTMPs. The second part of the algorithm, i.e. optimisation part, comprises the optimisation technique and the quantitative system failure analysis. The SOGA, chosen as the optimisation technique, governs the generation of system design alternatives converging to an optimal design case. FTA and BDD analysis are employed to quantify failure probabilities of the generated design cases and the obtained results are passed back to the generation process of new design alternatives. The final result of the optimisation process is a system design case with the minimal failure probability. The detailed structure of the GSDOA is shown in *Figure 4.13* and is further discussed in this section.

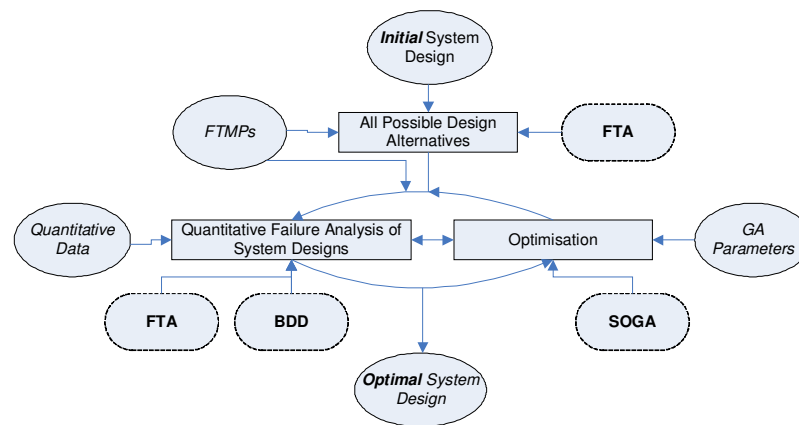


Figure 4.13. Structure of GSDOA

4.4.4.1. Preparative Part

To perform the design optimisation analysis for a chosen system the following data is required: a fault tree structure for the initial system design, a list of chosen design variables represented as FTMPs and parameters for the GA. Parameters used for the GA performance

and defined by the user are population size N , which should be an even number (the requirement of the population to have even number of chromosomes is discussed in Section 4.3.5.3), crossover rate, mutation rate and maximum number of generations performed before the optimisation process is terminated.

The fault tree for the initial system design and the list of the appropriate FTMPs with the associated replaceable fault tree events are employed to construct a fault tree incorporating all possible design alternatives. The construction of the fault tree occurs automatically within the derived computer program. Structural fault tree changes defined by a particular FTMP are completed for one chosen fault tree event at a time. The order in which FTMPs are implemented corresponds to the sequence they are provided in the initial data. The sequence of FTMPs is important and it is discussed in Section 5.4.2 considering a specific optimisation example. The constructed fault tree representing causes of failure for all design alternatives is then used in the following optimisation part of the algorithm.

4.4.4.2. Optimisation Part

The data required for this stage includes the fault tree representing all possible design alternatives created in the first part of the algorithm and the same list of FTMPs implemented. Moreover, for the quantitative analysis of system designs, a number of characteristics of system components are required, such as design and failure-related data.

Design characteristics include components volume, weight and cost. This set of data is arbitrary and varies from case to case. Three types of components can be introduced for the analysis according to their failure characteristics. If a component is repairable then its failure rate, mean time to repair and scheduled time interval for performing maintenance activities have to be provided. If component failure times are distributed under the Weibull distribution the distribution parameters β and η and also a value of scheduled maintenance interval need to be given. Since a time interval between scheduled maintenance activities can be considered as a design variable it can be omitted but it then needs to be defined in the provided list of design variables. Finally, if introducing unrepairable components their failure probability values should be provided.

Having the required data the optimisation process can be started. It starts with the generation of the initial feasible population of N chromosomes which was discussed in detail in Section 4.3.3.2. In the population each chromosome represents a specific design configuration and has a failure probability value of the design in question assigned to it. In order to find the failure

probability of a specific system design the fault tree representing all possible design alternatives is utilised. The chosen chromosome is decoded and the design variables are assigned specific values. Each house event representing the assigned value of the associated design variable is set to TRUE. The rest of the house events are set to either TRUE or FALSE according to the existing rules specific to each FTMP. Next, trimming of the fault tree follows where fault tree branches with the house events set to FALSE are eliminated. At this stage the fault tree structure is considered to be defined representing a particular design choice. The fault tree is converted to its BDD which is then used to evaluate the failure probability of the system design in question. The chromosome is assigned its fitness value.

Each chromosome also has assigned associated system characteristics for the system design it represents. To evaluate the characteristics *Equations 4.4 -4.14* are employed.

The initial population of chromosomes with their fitness values becomes a parent population and three genetic operators are used to generate a new offspring population. The offspring chromosomes provide a new set of system design cases. The fault tree for each of them is constructed as explained earlier and resulting system design characteristics and failure probability values are assigned to the corresponding chromosomes. At this stage the replacement procedure takes place and a new population comprising a number of chromosomes from either population is formed. The latter population is now considered a parent population for the next generation of offspring chromosomes. The optimisation process is terminated after a pre-set number of generations. The steps of the procedure are presented in *Figure 4.14*.

The algorithm provides a set of results. After a defined maximum number of generations, the feasible chromosome with minimal fitness value is presented. The genes of the chromosome are decoded to get values for the design variables representing the optimal system design case, while the chromosome fitness value gives the failure probability of the system. For the purposes of analysis the algorithm also produces a set of the minimum feasible and average fitness values for each generation.

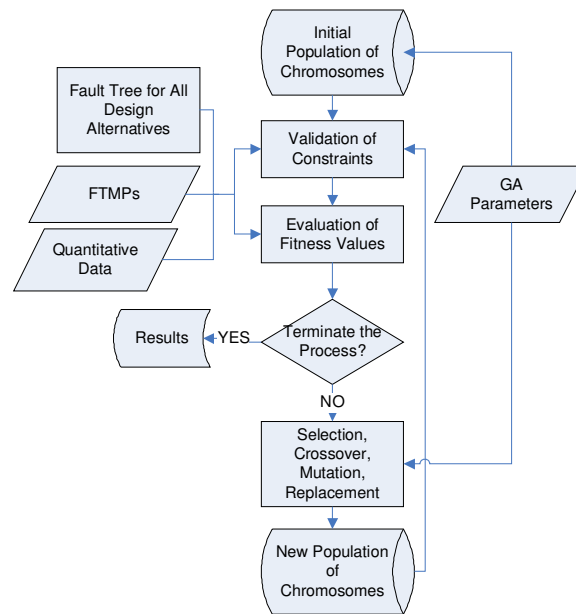


Figure 4.14. Optimisation Process Flow Chart

4.4.5. Computer Implementation of the GSDOA - GSDOP

The software built for a general system design optimisation algorithm is called GSDOP (General System Design Optimisation Programme). The code for the developed GSDOP was written using Microsoft Visual C++ 2003. It includes the code for the fault tree modification methodology and the code for optimisation process. The developed code for the GA implemented is based on the original Pascal Simple Genetic Algorithm code presented by Goldberg [45] which was translated and altered to adapt it to the problem analysed. The code for the computation method to convert a fault tree to a BDD structure and perform its quantitative analysis was previously developed at Loughborough University [74]. The source code was incorporated in the code for GSDOP. The routines constituting the developed programme and the subroutines they contain are discussed in detail.

4.4.5.1. Initialisation Stage

The routine *Initial_Data* is the initialisation routine called to read initial data stored in data files created by the user. The data is entered into the files manually. The files are named to define what set of data the file provides. Each name is composed of two parts. The first part can consist of any combination of letters and the second part is to be chosen from the given list in order to represent the content of the file. The list of possible file names, i.e. their default parts, is given in *Table 4.1*.

Table 4.1. Data Files

<i>Default Name</i>	<i>Information File Contains</i>
<i>_fts.txt</i>	A fault tree structure in a text format
<i>_var.txt</i>	Data of FTMPs employed
<i>_bse.aqd</i>	Data of basic events
<i>_cost_cst.txt</i>	Data of system cost constraints*
<i>_mdt_cst.txt</i>	Data of system maintenance down time constraints*
<i>_volume_cst.txt</i>	Data of system volume constraints*
<i>_weight_cst.txt</i>	Data of system weight constraints*
<i>_theta.txt</i>	Parameters of inspection intervals*
<i>_gav.txt</i>	Genetic algorithm variables

The sign “*” identifies optional data files. The optional files store data for evaluation of constraints. If any type of constraint is not considered for the analysed system the user does not create the corresponding file(s). For instance, if there are no limitations applied for system volume, then the file *_volume_cst.txt* is not provided.

The names of the files for the given initial data are kept in the main data file called “*data_files.txt*”. Examples of data files together with the requirements for their contents are provided in detail in *Appendix 1*.

The following sequence of subroutines is used to read specific data from the given files: *GA_Parameters_Data*, *Constraints_Data*, *Fault_Tree_Data*, *Variables_Data*, *Inspect_Interval_Data* and *Basic_Events_Data*. The subroutine *GA_Parameters_Data* reads parameters governing the GA, i.e. population size, crossover rate, mutation rate and maximum number of generations. The data is provided in the file *_gav.txt*. To read the available constraints data the subroutine *Constraints_Data* is employed. The subroutine also checks which constraints files are given. If any of the files are not provided validation of the corresponding constraints is not checked during the optimisation process. The subroutine *Fault_Tree_Data* reads the fault tree structure presented in a text format and saves it in a specific data structure form where one record represents one fault tree gate. The subroutine *Variables_Data* is employed to process the data of FTMPs. If the file *_theta.txt* is not provided it means either times between maintenance inspections for repairable components are given or all components are unrepairable. In this case the subroutine *Basic_Event_Data* is used to read failure characteristics of system components, which is stored in a file *_bse.aqd*. This file also contains data of components which will occur in the fault tree after implementation of FTMPs. If the file *_theta.txt* is provided it means there are *i* maintenance intervals considered as design variables and the information obtained from file *_bse.aqd* is not sufficient to perform quantitative analysis. The file *_theta.txt* stores the list of all basic events

of the fault tree where each of them is assigned a number ($1, 2, \dots, i$) indicating which maintenance interval is associated with a system component represented by the basic event. To read the data the subroutine *Inspect_Interval_Data* is used. Once the values of inspection intervals are generated the stored data is used to assign these values to corresponding basic events. Having the complete failure data the quantitative analysis can be performed.

4.4.5.2. Construction of the Fault Tree for all Possible Design Alternatives

Following the initialisation step the programme constructs a fault tree representing all possible design cases. The routine *Fault_Tree_Construction* is illustrated step by step in *Figure 4.15*. The routine is organised using a loop where one FTMP is implemented at each iteration. In the data file every FTMP is given with a code number of the associated fault tree element and a letter identifying a fault tree modification level. If the modification is made at event level then the subroutine *Change_Event_Gate* is employed to transform the corresponding basic event into an *OR* gate coded as $max_gate + 1$. Here max_gate is the maximum code number of a gate in the fault tree before changes are made. The subroutine *FT_Modification_Event* follows next. It implements a particular FTMP according to the values of its parameters given. However if the basic event has not been found the employed subroutine returns the value *FALSE* and no changes are made to the fault tree.

```

repeat for all  $i$  ( $i = 1, 2, \dots, total\_numbe\_of\_FTMPs$ )
If change made at event level = TRUE
    If Change_Event_Gate = TRUE
        FT_Modification_Event
    Else
        If Change_Gate_Gate = TRUE
            Copy_Part_Fault_Tree
            FT_Modification_Gate

```

Figure 4.15. Algorithm for Construction of a Fault Tree for Possible Design Cases.

If the modification has been set to be made at a gate level only the code number for the corresponding gate is changed to $max_gate + 1$ using the subroutine *Change_Gate_Gate*. If the identified gate was found the subroutine *Copy_Fault_Tree_Part* is implemented first. It creates a copy of a fault tree structure which will be incorporated as a principal event when altering the initial fault tree structure. The details of the utilization of the copied part of the fault tree were discussed in *Section 4.2.2.5*. A particular FTMP at the gate level is implemented using the subroutine *FT_Modification_Gate*.

The systematic coding of names of gates, basic and house events within new branches of the fault tree occurs automatically. The same rules apply in both fault tree modification cases, i.e. when changes are made at basic event and gate levels. Codes used for gates and house events consist of digits. As it was just mentioned earlier, the top gate of every new part of the fault tree is named as $max_gate + 1$ and the rest of the gates have numbers of $max_gate + 1 + i$, where $i = 1, 2, \dots, tn_gate$ (tn_gate is the total number of gates in the new part of the fault tree). The actual value for m is defined according to the FTMP employed and values of its parameters. A similar rule applies for names of house events. They have names $max_house + i$, where max_house is the name of the house event with the maximum number in a fault tree before the alteration is made, $i = 1, 2, \dots, tn_house$, where tn_house is the total number of house events in the part of the fault tree being incorporated. To identify values for max_gate and max_house two subroutines *Max_Gate_Number* and *Max_House_Number* are employed respectively.

The names of basic events in the incorporated new part of the fault tree are defined using particular rules. Each name is formed to represent the number and the type for the associated system component. The name has three parts. The first part is a code which is the same as the name of the initial event being replaced. The second part identifies the number of the component in a group of redundant components. If no redundant components have been introduced then it is coded as 1. The third number defines the type of the associated component. All these three parts of the name are separated using a sign “_”. For example, a basic event with code 5 is being replaced using **Pattern4** ($mn = 2, mt = 3, mk = 1$). It means failures caused by either one or two components are presented and it is possible to choose one of three different types of those components. Thus 6 (2×3) new basic events are added to the analysed fault tree with the following names: $5_1_1, 5_1_2, 5_1_3, 5_2_1, 5_2_2, 5_2_3$. The chosen component No 2 type 3 would be represented with the basic event coded 5_2_3 . The code 5_1_1 is assigned to the basic event chosen to be replaced.

Note, the user should also use the same codes for the basic events when entering the required initial data for system components, such as failure rates or cost *etc.*

4.4.5.3. Optimisation Part

The optimisation process is implemented in the routine *Genetic*. To manipulate populations of chromosomes two arrays of variables *pop_old* and *pop_new* are employed in the routine. One array represents a parent population and another is employed to store an offspring population data. Each element of an array represents a chromosome. The number of elements in each of

them is equal to the size of the population. An element of the array is a variable of a structure type named *record*. A structure is a collection of variables of different data types. The variables in the structure *record* also called elements or members represent chromosome-related data, such as, a binary structure of the chromosome, the fitness value or the cost of the system design the chromosome represents. The introduced structure type is shown in *Figure 4.16*.

```
struct record
{
    bool *chrom;
    gene *genes;
    int *phenotype;
    double objective;
    double fitness;
    int parent1;
    int parent2;
    double cost_d;
    double cost_t;
    double cost_c;
    double cost_p;
    double volume;
    double weight;
    double downtime;
};
```

Figure 4.16. Members of a Data Structure *record*.

The variable *chrom* is used to store a set of binary numbers, i.e. a chromosome. The variable *genes* is an array of strings of binary numbers of different where each string represents a gene. Values of decoded genes are stored using the variable *phenotype*. The objective function value corresponding to the chromosome is assigned using the variable *objective*. When an offspring chromosome is generated the order number of its parent chromosomes in the parent population are stored using variables *parent1* and *parent2*. To store system characteristics for a system design identified with the current chromosome variables *cost_d* (design cost), *cost_t* (maintenance testing cost), *cost_p* (preventive maintenance cost), *cost_c* (corrective maintenance cost), *volume*, *weight* and *downtime* are employed.

Form_Chromosome is the first subroutine employed which defines the number of genes in the chromosome and how many bits comprise each of the genes. The data of the FTMPs and maintenance intervals introduced as variables is used. Once the structure for the chromosomes is defined the initial population of chromosomes is generated using subroutine *Init_Population*. The structure of the subroutine is given in *Figure 4.17*. In the current subroutine one chromosome is generated at each iteration employing the subroutine

Init_Pop_Chromosome. The results obtained are stored as members *chrom*, *gene* and *phenotype* of a certain element of an array *old_pop*.

The values of maintenance intervals (if they were not defined initially) and structural design variables are obtained from the array *phenotype* and the fault tree representing all possible designs is modified accordingly. The process is managed using the subroutine *Fault_Tree_Design*.

Next, validation of the introduced constraints is checked. If the subroutine *Check_Constraints_Values* returns a result FALSE, the corresponding element of an array named *valid* is assigned the value 0. When the subroutine *Check_Constraints_Values* returns the result TRUE it means all introduced constraints are satisfied and the corresponding element of array *valid* is assigned the value 1. The subroutine also assigns values to the variables *cost_d*, *cost_t*, *cost_c*, *cost_p*, *volume*, *weight* and *downtime* of the considered element of the array *old_pop*. If certain constraints are not considered their corresponding variables are assigned the value 0.

```

Repeat until  $i \leq N$ 
  valid = 0;
  while valid = 0
    Init_Pop_Chromosome
    Fault_Tree_Design
    If Check_Constraints_Values = TRUE
      valid = 1;
    Else
      valid = 0;
  If a file _theta.txt exists
    Assign_Intervals
    Update_Failure_Data
  Quantification
   $i = i + 1$ .

```

Figure 4.17. Structure of the Subroutine *Init_Population*.

If maintenance intervals are considered as variables their generated values need to be assigned to the associated components, i.e. basic events, and failure characteristics for the components need to be updated. Therefore two additional subroutines are employed such as, *Assign_Intervals* and *Update_Failure_Data*. This step is omitted if the maintenance intervals are defined or are not considered in the analysis.

The following step is performed to obtain a failure probability value for the generated design alternative. The source code used for this purpose is implemented as an independent external

process. It returns a failure probability value of the system design considered which is assigned to the variable *objective*.

One iteration of the loop is completed. Since this was an initial chromosome generation its parents are not identified and members *parent1* and *parent2* for the considered element are assigned values of zero. The subroutines are repeated in the described order until a population of *N* chromosomes is generated.

The initial population is considered as a parent population for the next generation of an offspring population. Prior to the generation of any offspring population the details of its parent population are stored. The subroutine *Statistics* is employed to find the average of the objective function values and identify the minimal feasible value of the objective function as well as the chromosome that yields this minimal value. To report the results the subroutine *Report_Population* is used. Every time the subroutine is called it updates three result files “*Average_Objective*” “*Best_Objective*” and “*Genetic_Results*”. As the names suggest, files “*Average_Objective*” and “*Best_Objective*” store corresponding objective function values obtained from the parent population for each generation. The decoded chromosomes, i.e. genes’ phenotypes corresponding to the minimal feasible objective function value are stored for each parent population in the file “*Genetic_Results*”.

The process of generating new populations is organised using a main loop of the routine *Genetic*. A number of iterations in the loop is equal to the maximum number of generations which is defined by the user. In order to generate a new offspring population the subroutine *Generation* is employed at each iteration. The subroutine has a loop implemented. In this case, during each iteration, at first two parent chromosomes are selected. The subroutine *Reproduction* is employed where the biased roulette wheel method is implemented. The subroutine returns the order number of the selected chromosome in the population. These numbers are then used in the subroutine *Crossover* which performs the crossover operation. While performing the crossover operation both offspring chromosomes undergo a bit by bit mutation governed by the subroutine *Mutation*. Next the resulting two chromosomes are checked if each gene is feasible. If there is at least one infeasible gene it is regenerated using the subroutine *Repair*. The output of the subroutine is assigned to variables *chrom*, *gene* and *phenotype* and one iteration is completed. Thus in the subroutine *Generation* during a single iteration the two offspring chromosomes have been generated. Since every time two parent chromosomes are selected and two offspring are generated it is required that the total number

of chromosomes in a population would be an even number. The output of the subroutine *Generation* is a new population of chromosomes stored using the variable *new_pop*.

Following this step the subroutine *Fault_Tree_Design* is employed for each chromosome in the population. It transforms the fault tree with all possible design alternatives to the one that represents one particular design case defined by the values of the parameters of the design variables coded in the chromosome. Having the fault tree structure specified, appropriate characteristics associated with basic events of the fault tree are used to check if the generated design satisfies the predefined constraints. The subroutine *Check_Constraints_Values* is employed and an array *valid_new* is introduced. Each element of the array corresponds to one design case and is assigned the value 0 if any constraint is violated and 1 if none of constraints are violated. Next, subroutines *Assign_Intervals* and *Update_Failure_Data* are called if intervals between maintenance routines need to be defined. Otherwise the programme of quantitative analysis is called out straight after the constraints-checking procedure. The listed subroutines are applied for every chromosome in the population in the order introduced.

The new generation parent population is formed from both the current parent and offspring populations. The comparison between offspring chromosomes and their parent chromosomes is implemented in the subroutine *Replacement* as described in Section 4.3.3.3. The resulting population of chromosomes is considered as a new parent population which is used to produce new offspring chromosomes. Thus the results files “*Average_Objective*”, “*Best_Objective*” and “*Genetic_Results*” are updated and a single iteration of the main loop is completed. Once the number of iterations completed reaches the predefined number the optimisation process is terminated.

4.5. SUMMARY

A number of approaches seen in the literature have been developed by which an optimal design can be obtained by combining the fault tree analysis method and the optimisation technique. However they were developed for a specific example and have not been applied for a general case system. The current research detailed so far has created a mechanism to solve a design optimisation problem for any safety system. It is based on the following developments, created as part of this research:

- newly defined FTMPs and utilised to represent possible design alternatives in one fault tree for any system analysed;
- a newly developed optimisation algorithm combining the fault tree analysis and the GA without restricted application to one particular system.
- a newly developed programme code to automate the design optimisation process.

A specific structure of a system fault tree is used in the developed optimisation algorithm. It represents all given design alternatives. The newly developed FTMPs enable the required fault tree for any system analysed to be built. Using the FTMPs establishes the possibility to analyse design options for any safety system. Thus the application of the algorithm is not restricted to one particular system.

The algorithm discussed in this chapter, GSDOA, was developed to solve non-specific safety system design optimisation problems. The objective of the optimisation process is to define a particular set of system components that would constitute an optimal system design. As a result, the algorithm determines the case where the system failure probability is minimised and the utilisation of available resources is optimised.

The programme code, GSDOP, was developed to implement GSDOA. It has three major groups of subroutines. The first group is associated with the initialisation part of the programme where the initial data provided is processed. The second group represents subroutines employed to implement the required FTMPs and construct the fault tree representing causes of failure for all possible design cases. The last group of the programme subroutines are employed for the optimisation process. The code allows the user to introduce an initial design, specify the possible design changes and limitations on resources and will then automatically generate a possible solution.

The algorithm introduced and the code developed have the potential to be applied to a range of safety system. To analyse the performance of the code a number of systems needs to be examined. The systems chosen to be analysed are the High Integrity Protection System (HIPS) and Firewater Deluge System (FWDS). The results of the application examples are discussed in the following chapters, *Chapters 5 and 6*.

5. HIGH INTEGRITY PROTECTION SYSTEM DESIGN OPTIMISATION USING GSDOA

5.1. INTRODUCTION

The purpose of this chapter is to present the application of the developed GSDOA to solve a specific design optimisation problem. It provides guidelines of the GSDOA application process and also demonstrates its problem solving capability. A High Integrity Protection System (HIPS) has been chosen as an application example.

In hazardous industrial environments safety systems are designed to protect individuals and the workforce by lowering the risk of the occurrence of a system failure. The analysed HIPS is a safety system fitted on offshore platforms which aims to prevent a high-pressure surge passing through the system. In an offshore platform the high pressure can come from a production well and therefore protection is required for the equipment located downstream on the processing platform. Thus, the HIPS protects the processing equipment whose pressure rating could be exceeded.

In *Section 5.2* the structure of the HIPS and its fault tree are introduced. Design options are presented in *Section 5.3*. *Section 5.4* considers the HIPS design optimisation problem and includes detailed discussion about data preparation for the analysis. Implementation of FTMPs and construction of the fault tree for all possible design cases of HIPS is also discussed in this section. In *Section 5.5* performance analysis of the GSDOP is provided followed by the discussion of the results of the application, along with the generic implications of the analysis.

The research about the GSDOA and its application example to the HIPS was presented at 18th AR²TS and published in the proceedings (D. Astapenko and L.M. Bartlett, System Failure Minimisation Using Automated Design Optimisation. Proceedings of the 18th AR²TS (Advances in Risk & Reliability Technology Symposium), April 2009, Loughborough University, UK, p.347-359).

5.2. HIGH INTEGRITY PROTECTION SYSTEM DESCRIPTION

The basic structure of the HIPS is shown in *Figure 5.1*. The system has two levels of protection, which are the Emergency Shutdown (ESD) (Sub-system 1) and High – Integrity

protection subsystems (Sub-system 2). The ESD subsystem is the first level of protection that includes Wing, Master and ESD valves (ESDV) and also a pressure transmitter (PT). When the pipeline pressure exceeds the permitted value then all the valves in the ESD subsystem are closed. The pressure transmitter is used to monitor pressure in the pipeline. The second fitted subsystem works in a similar way to the first subsystem, but is independent in operation from the ESD subsystem and provides an additional level of protection. It has a fitted HIPS valve and a pressure transmitter.

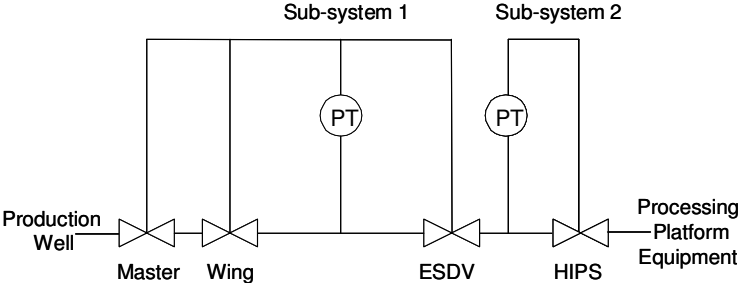


Figure 5.1. High Integrity Protection System

The causes of failure to protect the processing equipment from the high pressure surge passing through are analysed using FTA. The top event for the fault tree is defined as “Safety system fails to protect”. It will occur if all the valves along the pipeline fail to close or in another words both safety subsystems fail. Thus the input events related by AND gate logic are “Wing valve fails to protect”, “Master valve fails to protect”, “ESD valve fails to protect” and ”HIPS valve fails to protect”, as shown in Figure 5.2. Note that a number inside the gate symbol in Figure 5.2 and numbers in the rest of the fault tree figures in the chapter are used to number gates. Gate numbering is utilised throughout the analysis to help trace the development of the system fault tree and the implementation of FTMPs.

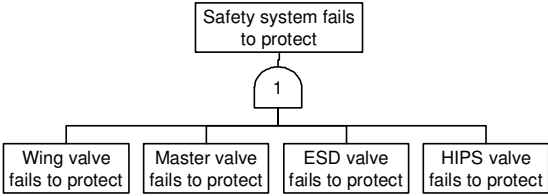


Figure 5.2. Top Event of System Fault Tree

All the system valves are of the “air to open” type. They are controlled by computer logic in the following way: if pressure in the pipeline is lower than the permitted value then the

solenoid of the valve is energised, the pneumatic line is pressurised and the valve is retained in the open state by the associated actuator. If the pressure increases above the acceptable value, the circuit of the output channel to the solenoid is opened. The pressure in the pipeline is monitored by the pressure transmitter, which sends a signal to the computer and the computer causes the circuit to open. Two relay contacts are available to break the circuit. Once the circuit is broken the solenoid is de-energised and a vent valve is activated. This results in a drop of pressure to the actuator that causes the valve to close.

First consider the wing valve. The fault tree logic representing its causes of failure is shown in *Figure 5.3*. The wing valve fails to close if either the wing valve itself fails or if the pneumatic line to the actuator of the valve remains pressurised. Failure of the valve occurs due to failure of either its valve part or its solenoid part which are represented with basic events “WV” and “SVW” respectively. The pneumatic line remains energised due to two reasons. The first one is the computer failure to send a trip signal. It occurs if either the computer logic fails or the pressure transmitter fails and it results in no trip signal to the computer. Failure of the computer logic is the basic event “PLC1” and failure of the pressure transmitter corresponds to the basic event “PT1”. The second reason why the solenoid valve stays energised is that both relay contacts stay closed. This situation in the fault tree is described with two basic events “R1/1” and “R1/2” connected with an AND gate.

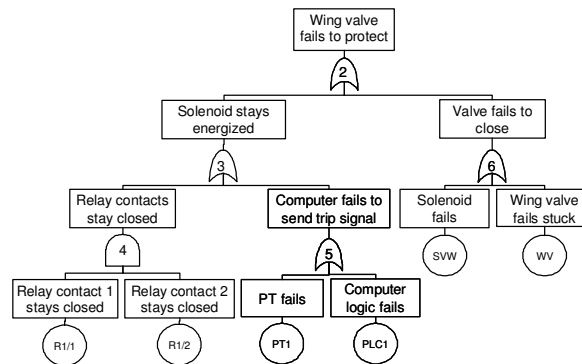


Figure 5.3. Fault Tree for Failure of Wing Valve

The causes of failure of the master and ESD valves are analogous to the ones for the wing valve. Therefore the structures of the fault trees for the valves remain the same. The only changes that are introduced to the fault tree for each valve correspond to the failure of a valve itself and its solenoid valve. In the fault tree for the master valve the basic event “MV” is used instead of the basic event “WV” which identifies the failure of the master valve itself. The failure of the solenoid valve for the master valve is defined with the basic event “SVM”

instead of event “SVW”. Accordingly basic events “ESD” and “SVE” are introduced in the fault tree for the ESD valve failure. The passage of fault logic from detection of hazardous gas levels to de-energising the solenoids of the wing, master and ESD valves is common for all three valves and the event “Solenoid stays energised” is identical to that described for the wing valve. Therefore the branching structure of the fault tree for this event, with the same gate numbers and the same basic events, is replicated in the fault tree parts for failure of the master and ESD valves.

The HIP subsystem has a pressure transmitter and a computer independent from the ESD subsystem. It means the two subsystems do not have any common components even though the logic of component failures leading to the subsystem failure is the same. Therefore the fault tree representing causes of failure of the valve in the HIPS subsystem has the same structure where the basic events are different from the ones in the fault trees for the valves of the ESD subsystem. The HIPS subsystem fault tree is shown in *Figure 5.4*.

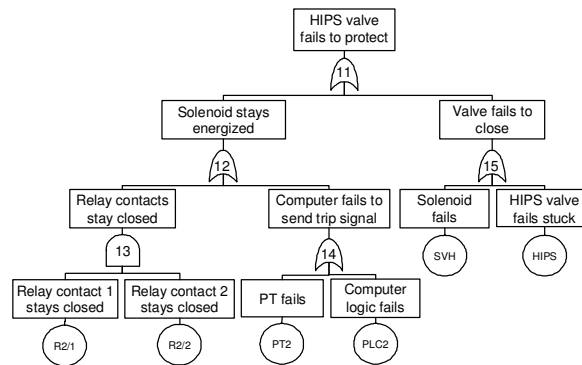


Figure 5.4. Fault Tree for Failure of HIPS Valve

5.3. OPTIONS FOR HIPS DESIGN ALTERATIONS

Having the initial design structure of the HIPS introduced there are a large number of options for it to be altered with respect to the minimisation of the system failure probability. Design alteration options considered in this case are listed in *Table 5.1*. The possibility to reduce the HIPS failure probability through the choice of performing maintenance at appropriate intervals is also considered. Time intervals between each maintenance testing performed for ESD (θ_1) and HIPS (θ_2) subsystems can vary from 1 week to 2 years. The measurement unit used for maintenance test intervals is one week.

Table 5.1. Structural Design Variables

<i>Associated Component</i>	<i>Description of Design Alteration</i>	<i>Design Variable</i>	<i>Possible Values of Parameters</i>
ESD Valve	Number of ESD valves fitted	n_1	1, 2
	Valve type	t_1	type 1, type 2
HIPS Valve	Number of HIPS valves fitted	n_2	1,2
	Valve type	t_2	type 1, type 2
Pressure Transmitter 1	Number of pressure transmitters fitted in subsystem 1	n_3	1 - 4
	Minimum number of pressure transmitters required to function for subsystem 1	k_3	1 - n_3
	Pressure transmitter type	t_3	type 1, type 2
Pressure Transmitter 2	Number of pressure transmitters fitted in subsystem 2	n_4	1 - 4
	Minimum number of pressure transmitters required to function for subsystem 2	k_4	1 - n_4
	Pressure transmitter type	t_4	type 1, type 2

When analysing the possible HIPS design alterations considerations are given to the design characteristics of cost and maintenance down time. The following requirements are introduced: 1) total system cost (design cost) must be less than 1000 units; 2) maintenance down time must be less than 130 hours per year.

For this example, each considered system component is repairable and can fail in the dormant failure mode. Dormant failure is the inability of the component to carry out its desired task on demand. The failure rate and mean repair time for each component option is given in *Table 5.2*. A new HIPS design structure needs to meet the defined cost and maintenance requirements therefore a design cost and a test time for each potential component is also given in the table.

Table 5.2. Component Data

<i>Component</i>	<i>Failure Rate, h</i>	<i>Mean Repair Time, h</i>	<i>Cost</i>	<i>Test Time, h</i>
Wing valve	1.14×10^{-5}	36	100	12
Master valve	1.14×10^{-5}	36	100	12
HIPS valve type 1	5.44×10^{-6}	36	250	15
HIPS valve type 2	1×10^{-5}	36	200	10
ESDV valve type 1	5.44×10^{-6}	36	250	15
ESDV valve type 2	1×10^{-5}	36	200	10
Solenoid valve	5×10^{-6}	36	20	5
Relay Contacts	0.23×10^{-6}	36	1	2
Pressure transmitter type 1	1.5×10^{-6}	36	20	1
Pressure transmitter type 2	7×10^{-6}	36	10	2
Computer logic	1×10^{-5}	36	20	1

5.4. HIPS DESIGN OPTIMISATION PROBLEM

The given list of structural design variables together with the possible choices of different time intervals between maintenance activities provide 42,831,360 different potential HIPS design and maintenance alternatives. To analyse each design case individually in order to find the one which minimises the failure probability is impractical. The application of GSDOP in this case could be very effective.

5.4.1. Data Initialisation

The initialisation stage of the optimisation process involves preparation of data in the format satisfying specific GSDOP requirements. First of all, code numbers are introduced for gates and basic events in the fault tree for the initial HIPS design. The gates are numbered starting with a top gate which has a number 1 (*Figure 5.2*). Next, gates representing the failure logic of the wing valve are coded as shown in *Figure 5.3*. Gates representing the failure logic of the master and ESD valves are coded in the same order and so are the gates representing the HIPS valve where the last gate has the code 15 (*Figure 5.3*). Numbers used to code basic events initially given in the fault tree are listed in *Table 5.3*.

Table 5.3. Number Codes for Basic Events

<i>Component Failure</i>	<i>Basic Event</i>	<i>Code</i>
Wing valve fails	WV	1
Solenoid valve for the wing valve fails	SVW	2
First relay contact in the ESD subsystem fails	R1/1	3
Second relay contact in the ESD subsystem fails	R1/2	4
Pressure transmitter in the ESD subsystem fails	PT1	5
Computer logic in the ESD subsystem fails	PLC 1	6
Master valve fails	MV	7
Solenoid valve for the master valve fails	SVM	8
ESD valve fails	ESD	9
Solenoid valve for the ESD valve fails	SVE	10
HIPS valve fails	HIPS	11
Solenoid valve for the HIPS valve fails	SVH	12
Pressure transmitter in the HIPS subsystem fails	PT2	13
Computer logic in the HIPS subsystem fails	PLC 2	14
First relay contact in the HIPS subsystem fails	R2/1	15
Second relay contact in the HIPS subsystem fails	R2/2	16

The given codes for gates and events are used when defining the fault tree structure in a text format. The fault tree for the HIPS in the text form is stored in the data file “*hips_fts.txt*”. The content of the data file is shown in *Figure 5.5*. Here the first number in each row identifies a gate code followed by the gate type. Next a number of input gates and events are given and first input gates and then input events are listed. An input gate is identified as “G:” and “E:” is

used to identify an input event. As a reminder, the rules of transformation of a graphical form of a fault tree into a text form are explained in detail in *Appendix 1*.

1	AND	3	0	G:2	G:7	G:12
2	OR	2	0	G:3	G:6	
3	OR	2	0	G:4	G:5	
4	AND	0	2	E:3	E:4	
5	OR	0	2	E:5	E:6	
6	OR	0	2	E:1	E:2	
7	OR	2	0	G:3	G:8	
8	OR	0	2	E:7	E:8	
9	OR	2	0	G:3	G:10	
10	OR	0	2	E:9	E:10	
11	OR	2	0	G:12	G:15	
12	OR	2	0	G:13	G:14	
13	AND	0	2	E:16	E:17	
14	OR	0	2	E:13	E:14	
15	OR	0	2	E:11	E:12	

Figure 5.5. Data File *hips_fts.txt*

The next step is to prepare the file named *hips_var.txt* which stores the given list of FTMPs. The content of the file is presented in *Figure 5.6*. The data file was prepared based on the list of the design variables. In total six FTMPs are used to implement the given design variables. In the file each row corresponds to one FTMP. The first letter (E or G) in the row denotes if the FTMP is implemented at event or gate level. Next the code number of event/gate being altered follows. The last three numbers are parameter values of the FTMP.

E	5	4	2	4
E	13	4	2	4
E	9	1	2	1
G	10	2	1	1
E	11	1	2	1
G	15	2	1	1

Figure 5.6. Data File *hips_var.txt*

The FTMP given in the file *hips_var.txt* are implemented within the programme automatically. Causes of failure corresponding to design cases with increased numbers of pressure transmitters are implemented at event level for both subsystems. Events with codes 5 (pressure transmitter in the ESD subsystem fails) and 13 (Pressure transmitter in the HIPS subsystem fails) are replaced using *Pattern5* where $mn=4$, $mt=2$ and $mk=4$. Changes to the fault tree representing the introduced possible design alternatives regarding the ESD valve are implemented using two FTMPs as follows. The ESD valve section contains the solenoid and the valve. The choice to use either a type 1 or type 2 valve refers only to the valve part and the solenoid part remains the same. However, when introducing a redundant valve, both the

solenoid and the valve parts are considered. This means that in order to implement the fault tree changes corresponding to the possible choices of redundancy, both basic events “ESD” and “SVE” need to be replaced. Therefore, first basic event “ESD” (code number 9) is replaced using *Pattern3* where $mt=2$. Following, gate number 10 is replaced using *Pattern1* where $mn=2$ which implements causes of failure for design options with either one or two redundant ESD valves. If the FTMPs are implemented in an opposite order, i.e. first *Pattern1* is implemented, then *Pattern3* would have to be implemented in each part of the fault tree where the branching structure with the top gate 10 was replicated since it includes the basic event “ESD”. The design variables for the HIPS valve are implemented in the same manner, first basic event “HIPS” and then gate 15 are altered.

Components in the HIPS are repairable and to find their unavailability values *Equation 2.6*. is employed which uses data of the failure rate, mean repair time and maintenance test interval. In the case analysed values for maintenance intervals are not specified therefore only failure rates and mean repair times of components are provided in the file for failure characteristics. Following requirements (see *Section A.3* in *Appendix 1*), digit 1 is used as maintenance interval value. The fragment of the file is presented in *Figure 5.7*.

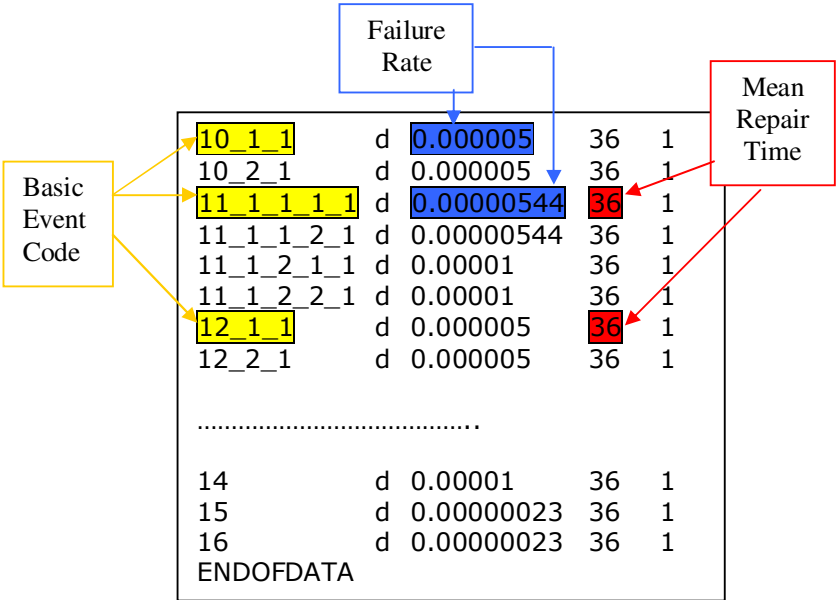


Figure 5.7. The Fragment of the File hips_bse.aqd

The data required for assigning values of maintenance intervals once they are generated to appropriate components is given in the data file *hips_theta.txt*. It stores the list of sets comprising of a basic even code and an identification number of associated maintenance

interval. General information about maintenance test intervals, such as the lower and upper bounds of their possible values, is also stored in the file. The fragment of the file is presented in Figure 5.8. As a reminder, both basic events for existing components and basic events for potential components are required to be listed in the latter data files. The codes for the basic events of new potential components are defined using the rules introduced in Section 4.3.5.2.

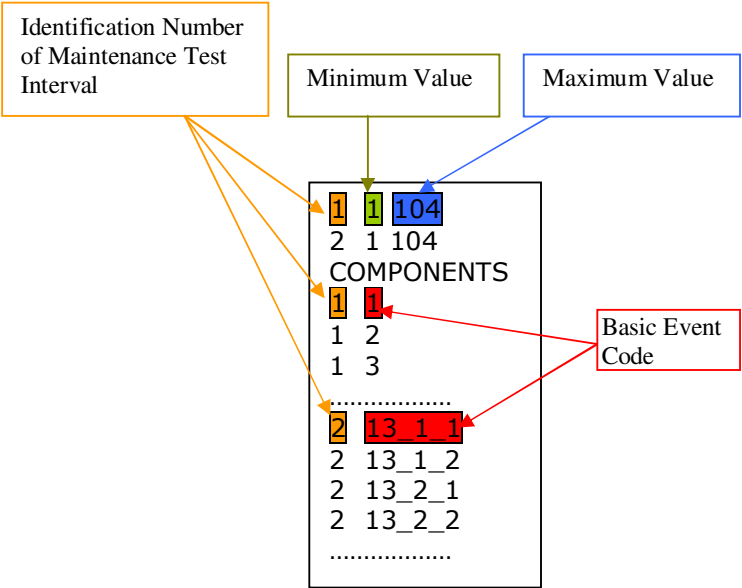


Figure 5.8. The Fragment of the File *hips_theta.txt*

Two types of constraints are considered in the HIPS optimisation problem: system cost and maintenance down time over a year. Data required for implementation of the problem constraints is stored in two files: *hips_cost_cst.txt* and *hips_mdt_cst.txt*. Figure 5.9 shows a fragment of the data file for system design cost. The first line stores lower and upper limits of the system cost where the lower bound is set to 0 and the maximum cost limit is 1000 units. Since only design cost is considered for the HIPS the total number of time units per examined period (one year) is equal to 0. After the keyword “COMPONENTS” all fault tree basic events with associated design costs are listed.

The data for the evaluation of the maintenance down time is stored in the file *hips_mdt_cst.txt* (A fragment of the file is shown in Figure 5.10). The file stores lower and upper bounds of possible values of maintenance down time which are listed after the keyword “MDT”. The key word “UNIT” identifies a total number of time units (used for determination of time intervals between maintenance activities) in the examined time period. The total number of time units is needed to evaluate the number of times the system is maintained during the examined period. Since maintenance test intervals are measured using weeks, the examined

period (a year) is divided into 52 weeks. The file also provides numerical values of test time and maintenance test interval for each component. Since in the case analysed test intervals for system components are considered as design variables and are generated during the optimisation process the test interval is set to 1 for each component in the data file. Note that limits on the system down time are given in hour units as test times for system components are also provided in hours.

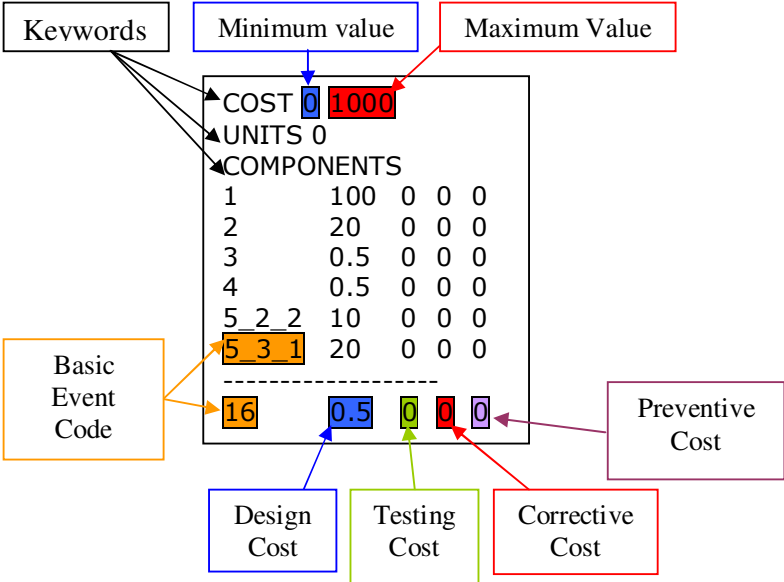


Figure 5.9. The Fragment of the File `hips_cost_cst.txt`

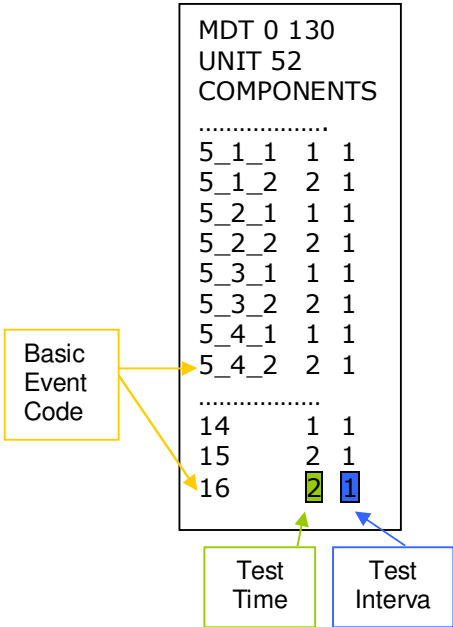


Figure 5.10. The Fragment of the File `hips_mdt_cst.txt`

5.4.2. Fault Tree Construction

The fault tree structure representing the causes of failure for the introduced design alternatives of the HIPS occurs automatically within the programme code. It is obtained using the given list of six FTMPs. The order the FTMPs are implemented is considered to be the most easy and straightforward. These general guidelines were followed when choosing the order:

- FTMPs can be listed according to the code numbers of events/gates they are implemented at. Increasing/decreasing order of code numbers can be used.
- FTMPs implemented at event level can be listed first followed by those being implemented at gate levels.
- If the situation occurs when failure of the component considered is defined using a fault tree structure rather than a basic event and type and redundancy options are introduced two FTMPs should be used. First, the FTMP (*Pattern 3*) for type options is implemented at event level which is an input event of this fault tree structure. Next, the FTMP (*Pattern 1, Pattern 2*) for redundancy options is implemented at the gate level which is the top gate of the structure.

Pattern5 associated with event *E5* (PT1) is the first FTMP to be implemented. It is used to add a new fault tree section which represents the causes of failure to monitor pressure in the pipeline in the ESD subsystem where up to four redundant pressure transmitters can be fitted. The new fault tree logic is shown in *Figure 5.11*.

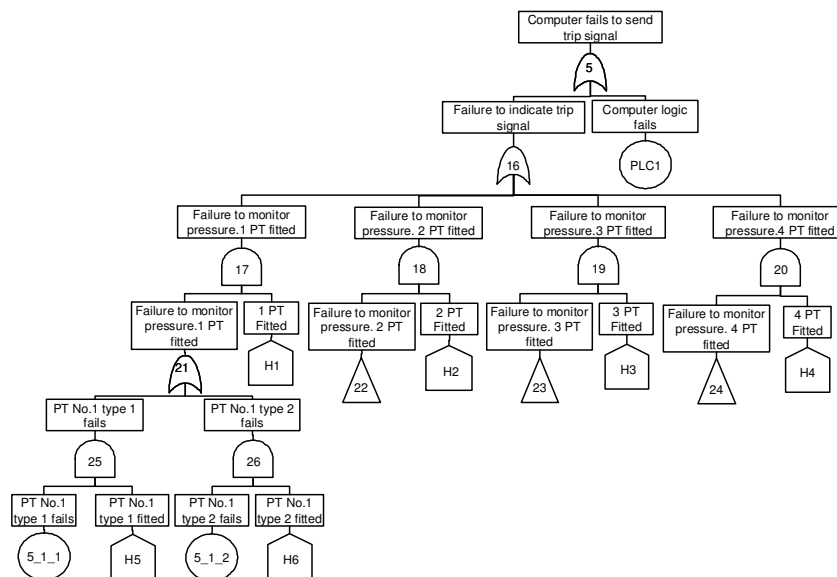


Figure 5.11. Pressure Transmitters Fail to Indicate High Pressure

The top gate of the incorporated part has a code 16 ($max_gate + 1 = 15 + 1$). Analysing from the top of the structure the first level of gates represents design cases with different numbers of pressure transmitters fitted. Here *Gate17* represents the failure to indicate a trip when 1 pressure transmitter is fitted, *Gate18* – failure to indicate a trip when 2 pressure transmitters are fitted, *Gate19* – failure to indicate a trip when 3 pressure transmitters are fitted and *te20* corresponds to failure to indicate a trip when 4 pressure transmitters are fitted.

A failure scenario is defined according to the values of the following house events: *H1* (1 pressure transmitter is fitted into the subsystem 1), *H2* (2 pressure transmitters are fitted), *H3* (3 pressure transmitters are fitted) and *H4* (4 pressure transmitters are fitted). Only one house event from the group can be assigned a value of TRUE while the rest of them are set to FALSE. In the next level of gates the logic of the occurring failures for the different numbers of pressure transmitters fitted is broken down. For example, consider the event of failure to monitor pressure when two pressure transmitters are fitted (*Gate22*) shown in *Figure 5.12*. House event *H2* would be assigned the value of TRUE in this case. If the system is designed to monitor pressure using either of the transmitters then failure of both of them will result in the failure to monitor pressure in the pipeline. Conversely, if two pressure transmitters must register an increase in pressure, failure of either will result in failure to register the increase in pressure. To indicate which design case is analysed one of the house events *H7* or *H8* is assigned the value of TRUE. In both cases the failure of a pressure transmitter is an intermediate event. Each of the events terminate with two pairs of basic events coupled with house events through an AND gate logic. The structure models an alternative type case for a component, i.e. pressure transmitter. This structural relationship is common throughout the analysed fault tree structure to model the inclusion of two different transmitter types.

The failure to monitor pressure when either three or four pressure transmitters are fitted is modelled in a similar manner. Note that three possible combinations of two pressure transmitters failing will be among all possible causes of failure to monitor pressure if three pressure transmitters are to be used. Accordingly, there will be five possible combinations of two pressure transmitters failing and three combinations of three transmitters failing if four pressure transmitters are in the system.

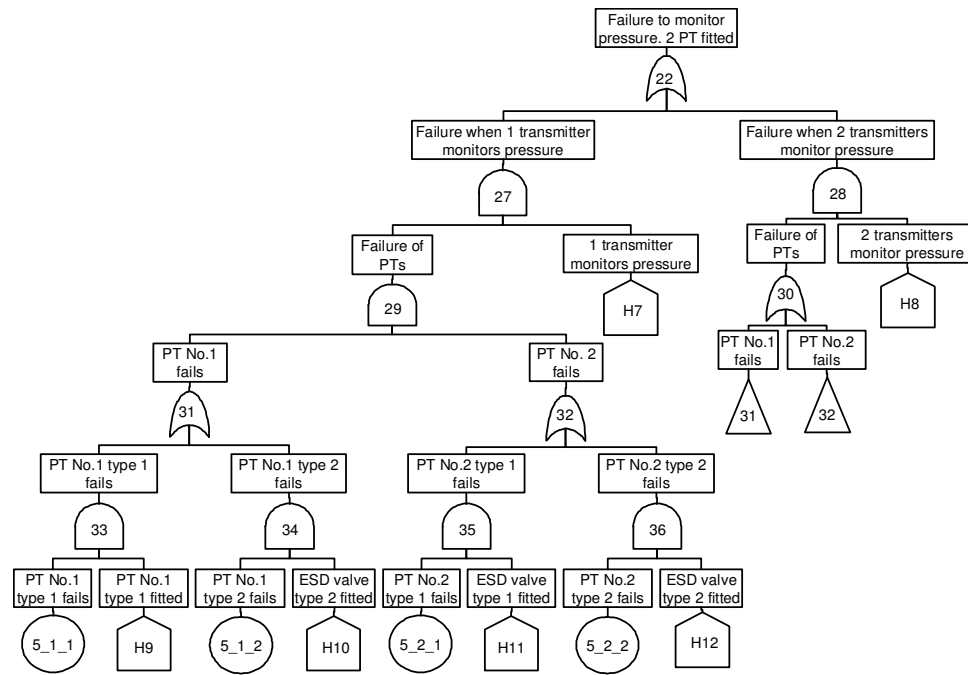


Figure 5.12. Two Pressure Transmitters fail to Indicate High Pressure

The implementation of *Pattern5* to replace event *E13* (pressure transmitter in the HIPS subsystem fails) results in the incorporation of the fault tree part which represents the causes of failure to monitor pressure in the HIPS subsystem. The structure of this part of the fault tree is identical to the one that represents the failure of pressure transmitters in the ESD subsystem.

As explained in *Section 5.4.1* two FTMPs are used to implement design alternatives introduced for the ESD valve. First basic event ESD (code number 9) is replaced using *Pattern3* where $mt=2$. The resulting fault tree structure (*Figure 5.13*) represents causes of failure of the ESD solenoid valve for two different types of the valve part. As seen in *Figure 5.13* the solenoid part remains the same. Two house events *H67* and *H68* are introduced to model this valve type as a design alternative. Event code 9 is renamed into code *9_1_1* (ESD valve No. 1 type 1 fails) and the new introduced event is coded as *9_1_2* which represents failure of ESD valve part No. 1 type 2.

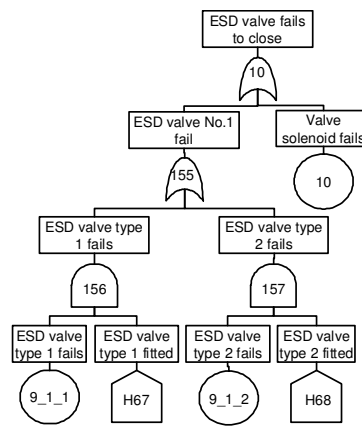


Figure 5.13. Fault Tree Structure for Alternatives of Different Type of ESD Valve

Next, *Pattern1* ($mn=2$ and $mk=1$) is implemented to model the structural changes in the fault tree regarding the possible introduction of a redundant solenoid valve. Failure of the solenoid valve is represented using an intermediate event in the fault tree. Therefore the FTMP is implemented at gate level (Gate No. 10) to replicate the logic relationship between failures of the basic valve components leading to the failure of the valve (*Figure 5.14*). The resulting fault tree structure has two additional house events *H69* and *H70* to model the redundancy level. Since one of the input events of gate 10 represents the earlier introduced alternative of different types of the valve part of the ESD solenoid valve (*Figure 5.13*) this structural relationship is mimicked in the fault tree part modelling the inclusion of the redundant valve. Therefore other new house events coded *H137* and *H138* used to model the type of the second introduced redundant valve mimic the house events *H67* and *H68* associated with the first redundant valve. The codes of house events *H137* and *H138* are derived as follows: $max_house + i$, where i is the code of a house event being replicated, i.e. $137=70 + 67$ and $138=70 + 68$. The codes of the existing basic events are changed following the rules introduced in *Section 4.3.5.2* as follows: $9_1_1_1_1$, $9_1_2_1_1$ and 10_1_1 . They represent input events of the intermediate event ‘Solenoid ESD valve No.1 fails’. Input basic events of the intermediate event ‘Solenoid ESD valve No.2 fails’ are coded as $9_1_1_2_1$, $9_1_2_2_1$ and 10_2_1 .

The design alternatives for the HIPS valve are the same as the ones analysed for the ESD valve. Therefore the same FTMPs are used in the same order as the ones for ESD valve. First *Pattern3* is implemented at event level (code 11), then design alternatives regarding redundant HIPS valves are implemented using *Pattern1* at gate level (gate code 15).

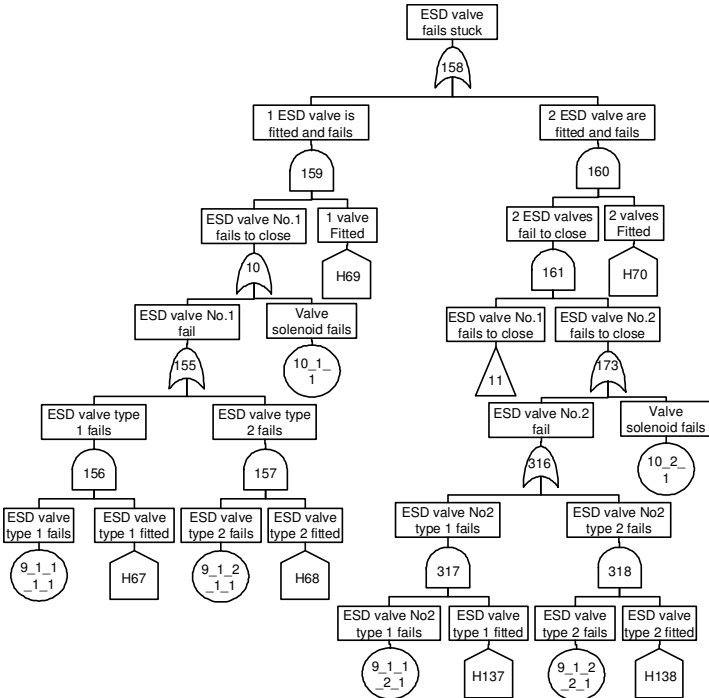


Figure 5.14. Fault Tree Structure for Design Alternatives of ESD Valve

5.4.3. Chromosome Structure

The structure of chromosomes utilized during the optimisation process is specified automatically by the developed code for each problem analysed. The structure of chromosomes for the HIPS optimisation problem is based on the data stored in the files *hips_var.txt* and *hips_theta.txt*. The file *hips_var.txt* provides information about FTMPs corresponding to the given structural design variables and the file *hips_theta.txt* stores the list of different time intervals between maintenance testing and ranges of their possible values. *Figure 5.15* shows the complete structure of the chromosome. Here the links between data stored in data files *hips_var.txt* and *hips_theta.txt* and chromosome structure are demonstrated.

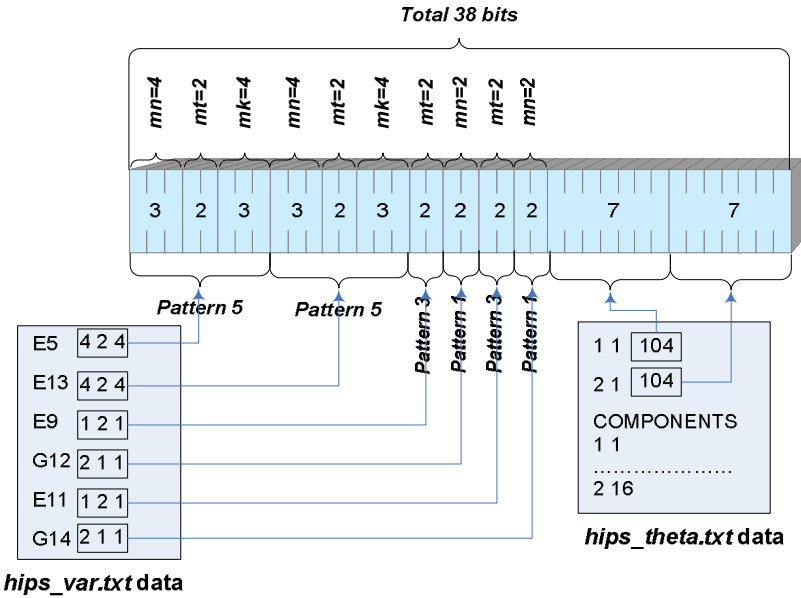


Figure 5.15. The Chromosome Structure

As a reminder, parameters of the given FTMPs are coded in the first part of a chromosome. Numbers of bits required to code each parameter are found employing *Formula 3.4*. A parameter with its value equal to 1 identifies that the corresponding structural changes are not made and therefore no bits are allocated for the parameter. Thus to code parameters of the first FTMP, *Pattern5*, implemented at event E5 (PT1), eight bits are allocated. Here three bits are used to code parameter mn which is equal to 4, $mt=2$ is coded using two bits and three bits are utilised to code the value of parameter mk which is equal to 4. Eight bits are allocated in the same order for the FTMP applied for the basic event E13 (PT2). Following this, four bits are allocated to code parameters of FTMPs representing design options for the ESD valve. The first two bits code the value of parameter mt for *Pattern3* ($mt=2$) and other two bits are allocated for parameter $mn=2$ for *Pattern1*. In the same way four bits are allocated for the remaining two FTMPs related to design alternatives of the HIPS valve.

After allocating 24 bits for the coding of the structural design variables, i.e. their corresponding FTMPs, the rest of the bits of the chromosomes are allocated for the two maintenance test intervals. A time interval between maintenance for both subsystem 1 and subsystem 2 can be from 1 to 104 weeks. This means that in order to code each value in binary numbers seven bits are required. Thus the chromosomes used in the HIPS optimisation problem have eight genes and thirty eight bits in total.

5.5. ANALYSIS OF GSDOP PERFORMANCE SOLVING THE HIPS DESIGN OPTIMISATION PROBLEM

5.5.1. Selection of the Values of the GA Parameters

GAs are guided search methods and values of the GA parameters such as population size, crossover rate and mutation rate influence the convergence of the optimisation process. Moreover the best values for the GA parameters, i.e. the ones assuring good convergence to the optimal solution, are case dependent. Setting values for GA parameters presents a challenge in terms of achieving good performance of the algorithm as there are no precise instructions with regards to what the values should be, although some guidance is available based on empirical studies [75]. Some common settings for mutation rate and crossover rate are presented in [76]. The right size of population is also important. If the population is too small it can cause the loss of genetic diversity. On the other hand if a population is too large over abundance of genetic diversity can appear [68]. Therefore an option to change the values of the GA parameters has been implemented in the GSDOP and the user can define these in the data file *hips_gav.txt*.

When solving the HIPS problem the choice of GA parameter values was based on studies focused on determining good parameter values for genetic operators [77] and a trial-and-error approach. In total forty eight sets of different combinations of values of the GA parameters were used. The values of the parameters are listed in *Table 5.4*. Due to the stochastic nature of the GA ten runs were performed for each set of parameters. Each time the process was terminated after 100 generations.

Table 5.4. The List of Values of GA Parameters

<i>Value No.</i>	<i>1st Value</i>	<i>2nd Value</i>	<i>3rd Value</i>	<i>4th Value</i>
<i>Parameter</i>				
Population Size	10	20	30	50
Mutation Rate	0.001	0.002	0.005	0.1
Crossover Rate	0.75	0.8	0.95	-

The numerical quantities used for comparison of the optimisation results are derived as averages of the best feasible unavailability values from 10 runs for each combination of GA parameters. Thus each set of the GA parameter values is assigned its corresponding minimal unavailability value. The average of the best unavailability values is also evaluated for each value of the GA parameter considered. The results are presented in table format. *Tables 5.5 - 5.9* are used to analyse the effect of different population sizes on the optimisation results

using different combinations of mutation and crossover rates. Tables 5.10-5.14 show how combinations of different crossover rates and population sizes influence the optimisation process when specific mutation rates are chosen. Accordingly, Tables 5.15-5.19 store the results of the optimisation process when each different crossover rate value is combined with all possible combinations of mutation rates and population sizes.

Results in Table 5.5 demonstrate that an average minimal unavailability value obtained after 100 generations decreases if larger sized populations are used. The minimum unavailability value is obtained when the population size is equal to 50 (3.44E-07). Therefore it can be stated that the convergence rate of the algorithm can be improved by increasing the population size. However computation intensity increases with larger population sizes and this needs to be taken into account.

Table 5.5. Minimal Average System Unavailability Values for Different Population Sizes

	Population Size			
	10	20	30	50
Mean of Minimal System Unavailability Values	4.44E-07	3.87E-07	3.69E-07	3.44E-07

As expected, if the population size is small, i.e. 10 chromosomes (Table 5.6), increasing mutation rate induces smaller system unavailability values to be found over the limited number of generations. When the population of chromosomes is small it takes longer to analyse the search space. Therefore the high mutation rate which creates diversity in the population enables explorations of diverse regions of the search space. Two different effects of crossover rate values can be noticed in this case. When the mutation rate range from 0.001 to 0.005 is used and the crossover rate increases, so does the average of the minimal system unavailability values. However combinations of higher crossover rates and the mutation rate equal to 0.01 results in smaller unavailability values to be found. Thus the minimal average unavailability value found using a population of 10 chromosomes is equal to 3.48E-07.

Table 5.6. Minimal Unavailability Values when Population Size is Equal to 10

Population size = 10				
		Crossover Rate		
		0.75	0.8	0.95
Mutation Rate	0.001	4.977E-07	5.330E-07	5.348E-07
	0.002	4.765E-07	4.665E-07	5.297E-07
	0.005	3.953E-07	3.978E-07	4.098E-07
	0.01	3.844E-07	3.510E-07	3.480E-07

The results obtained for a population size of 20 chromosomes are shown in *Table 5.7*. Here the performance of the optimisation algorithm was also improved when the mutation rate was increased. The influence of the crossover rate values on the potential of the programme in finding the optimal solution varies for every mutation rate. However the tendency that a high crossover rate corresponds to a smaller optimal unavailability value remains.

Table 5.7. Minimal Unavailability Values when Population Size is Equal to 20

Population size = 20				
		Crossover Rate		
		0.75	0.8	0.95
Mutation Rate	0.001	4.355E-07	3.653E-07	5.343E-07
	0.002	4.212E-07	3.954E-07	3.832E-07
	0.005	3.603E-07	3.837E-07	3.551E-07
	0.01	3.419E-07	3.192E-07	3.526E-07

For larger populations, i.e. 30 and 50 chromosome populations, the relationship between different combinations of mutation and crossover rates and the optimal values was even less consistent as shown in *Tables 5.8* and *5.9*. In both cases the increase of mutation rate up to 0.005 is beneficial for the optimisation performance. However the mutation rate equal to 0.01 introduces diversity in the population which has a negative effect on the convergence of the process. The negative effect of a diverse population is noticeable when the crossover rate is equal to 0.95 and the population consists of 30 chromosomes. Combinations of the latter mutation rate with crossover rates equal to 0.8 and 0.95 also result in higher optimal values for 50 chromosome populations. In both cases the algorithm performs better if combinations of mutation and crossover rate are used such that one rate is low and the other has a high value then using combinations where both parameter rates are either low or high. The smallest unavailability value is obtained using the highest mutation rate (0.01) and the lowest crossover rate (0.75).

Table 5.8. Minimal Unavailability Values when Population Size is Equal to 30

Population size = 30				
		Crossover Rate		
		0.75	0.8	0.95
Mutation Rate	0.001	3.948E-07	4.439E-07	3.613E-07
	0.002	3.845E-07	4.315E-07	3.598E-07
	0.005	3.551E-07	3.365E-07	3.403E-07
	0.01	3.251E-07	3.338E-07	3.523E-07

Table 5.9. Minimal Unavailability Values when Population Size is Equal to 50

Population size = 50				
		Crossover Rate		
		0.75	0.8	0.95
Mutation Rate	0.001	3.633E-07	4.036E-07	3.494E-07
	0.002	3.420E-07	3.607E-07	3.483E-07
	0.005	3.220E-07	3.167E-07	3.083E-07
	0.01	3.150E-07	3.435E-07	3.498E-07

When considering only the mutation rate its value has the same influence on algorithm performance as the population size. Smaller unavailability values of feasible chromosomes are found when using larger mutation rate values (*Table 5.10*). The minimal mean of the averaged minimal unavailability values is equal to 3.434E-07 and corresponds to the mutation rate equal to 0.01.

Table 5.10. Minimal Average System Unavailability Values for Different Mutation Rates

	Mutation Rate			
	0.001	0.002	0.005	0.01
Mean of Minimal System Unavailability Values	4.347E-07	4.091E-07	3.567E-07	3.434E-07

The minimal average system unavailability values obtained using combinations of the GA parameter values where the mutation rate is set to 0.001 are listed in *Table 5.11*. The results show that using small population sizes, such as 10 or 20 chromosomes, the approach performs better if a low crossover rate is used. However if the population size is increased better results are obtained using the crossover rate equal to 0.95. The results imply that a high reproduction rate decreases the convergence rate of the algorithm if the diversity in small populations is limited. On the other hand, when populations of chromosomes are larger new strings introduced more quickly into the population improves the optimisation process.

Table 5.11. Minimal Unavailability Values when Mutation Rate is Equal to 0.001

Mutation Rate = 0.001				
		Crossover Rate		
		0.75	0.8	0.95
Population Size	10	4.977E-07	5.330E-07	5.348E-07
	20	4.355E-07	3.653E-07	5.343E-07
	30	3.948E-07	4.439E-07	3.613E-07
	50	3.633E-07	4.036E-07	3.494E-07

When the mutation rate equal to 0.002 is used it is difficult to identify a clear tendency of influence of other parameter values on the optimisation results (*Table 5.12*). For example, the algorithm finds smaller near-optimal feasible solutions when the population size is either 10

or 20 if using the crossover rate equal to 0.8 in comparison with the other crossover rates. However, if the size of a population is equal to 30 the smallest near optimal solution is obtained using the crossover rate equal to 0.95. If the population size is 50 chromosomes then the smallest average unavailability value is obtained using the crossover rate equal to 0.75.

Table 5.12. Minimal Unavailability Values when Mutation Rate is Equal to 0.002

Mutation Rate = 0.002				
		Crossover Rate		
		0.75	0.8	0.95
Population Size	10	4.765E-07	4.665E-07	5.297E-07
	20	4.212E-07	3.954E-07	3.832E-07
	30	3.845E-07	4.315E-07	3.698E-07
	50	3.420E-07	3.607E-07	3.483E-07

Opposite to the previous two cases, parameter combinations where mutation rate is equal to 0.005 (Table 5.13) result in the decreasing minimal near optimal solutions when the population size increases for each crossover rate value. Here the influence of the crossover rate values to the optimisation process depends on the size of populations used. If a population size is equal to 10 chromosomes then higher crossover rates lead to the loss of chromosomes with good genetic information. On the contrary, the quicker introduction of new chromosome when using the high crossover rate, i.e. 0.95, induces diversity in the large populations and therefore the minimal near optimal solution is found when a population of 50 chromosomes is used.

Table 5.13. Minimal Unavailability Values when Mutation Rate is Equal to 0.005

Mutation Rate = 0.005				
		Crossover Rate		
		0.75	0.8	0.95
Population Size	10	3.953E-07	3.978E-07	4.098E-07
	20	3.603E-07	3.837E-07	3.551E-07
	30	3.551E-07	3.365E-07	3.403E-07
	50	3.220E-07	3.167E-07	3.083E-07

Table 5.14 presents the optimisation results obtained using the mutation rate equal to 0.01 and all possible combinations of population sizes and crossover rates. In this situation, if a population comprises of 10 chromosomes then smaller unavailability values are obtained using higher a crossover rate. Conversely, minimal average unavailability value increases if a crossover rate increases when the population size is equal to 50 chromosomes. It follows that for the high mutation rate value of either the population size or crossover rate used should be small in order to maintain diversity in the population and preserve good genetic information at the same time. The influence of different parameter values on the optimisation results is more

pronounced for the population size than the crossover rate. Therefore it can also be noted that the population size has more influence on the optimisation process than the crossover rate when the mutation rate used is equal to 0.01.

Table 5.14. Minimal Unavailability Values when Mutation Rate is Equal to 0.01

Mutation Rate = 0.01				
		Crossover Rate		
		0.75	0.8	0.95
Population Size	10	3.844E-07	3.510E-07	3.480E-07
	20	3.419E-07	3.192E-07	3.526E-07
	30	3.251E-07	3.338E-07	3.523E-07
	50	3.195E-07	3.435E-07	3.498E-07

Increasing the values of the crossover rate has an opposite effect on the optimisation results than previously discussed increasing population size or mutation rate. *Table 5.15* shows that the larger mean value of minimal unavailability values corresponds to the higher crossover rate. Thus on average the smallest minimal unavailability values for the HIPS system are found using the crossover rate equal to 0.75.

Table 5.15. Minimal Average System Unavailability Values for Different Crossover Rates

	Crossover Rate		
	0.75	0.8	0.95
Mean of Minimal System Unavailability Values	3.824E-07	3.864E-07	3.892E-07

Using the GA parameter combinations with crossover rate set to 0.75 variations in both population sizes and mutation rates have the same effect on the optimisation process. If their values increase the minimal obtained unavailability value decreases. The results are presented in *Table 5.16*.

Table 5.16. Minimal Unavailability Values when Crossover Rate is Equal to 0.75

Crossover Rate = 0.75					
		Mutation Rate			
		0.001	0.002	0.005	0.01
Population Size	10	4.977E-07	4.765E-07	3.953E-07	3.844E-07
	20	4.355E-07	4.212E-07	3.603E-07	3.419E-07
	30	3.948E-07	3.845E-07	3.551E-07	3.251E-07
	50	3.633E-07	3.420E-07	3.220E-07	3.195E-07

It is difficult to identify a consistent pattern of influence of mutation rates and population sizes to the optimisation results when the crossover rate used is equal to 0.8 as shown in *Table 5.17*. Combinations with population size equal either to 20 or 50 chromosomes result in

smaller optimal unavailability values. The mutation rate equal to 0.01 dominates among other mutation rates in the contribution towards better optimisation results.

Table 5.17. Minimal Unavailability Values when Crossover Rate is Equal to 0.8

Crossover Rate = 0.8					
		Mutation Rate			
		0.001	0.002	0.005	0.01
Population Size	10	5.330E-07	4.665E-07	3.978E-07	3.510E-07
	20	3.653E-07	3.954E-07	3.837E-07	3.192E-07
	30	4.439E-07	4.315E-07	3.365E-07	3.338E-07
	50	4.036E-07	3.607E-07	3.167E-07	3.435E-07

When the crossover rate is high and equal to 0.95 the algorithm performs better if larger size populations are used (Table 5.18). Moreover, parameter combinations where population size is small result in smaller optimal solutions when the mutation rate increases. However for larger population sizes such a consistent pattern does not exist. In this case the overall minimal unavailability value is obtained using the population of 50 chromosomes and the mutation rate equal to 0.005.

Table 5.18. Minimal Unavailability Values when Crossover Rate is Equal to 0.95

Crossover Rate = 0.95					
		Mutation Rate			
		0.001	0.002	0.005	0.01
Population Size	10	5.348E-07	5.297E-07	4.098E-07	3.480E-07
	20	5.343E-07	3.832E-07	3.551E-07	3.526E-07
	30	3.613E-07	3.698E-07	3.403E-07	3.523E-07
	50	3.494E-07	3.483E-07	3.083E-07	3.498E-07

From the presented results it follows that the algorithm produces better solutions if a larger population size is used. The same principle is valid for the mutation rate. Even though the smallest average system unavailability value is obtained using the crossover rate equal to 0.95 the mean system unavailability values increases if the crossover rate is high (Table 5.15). Therefore, a further analysis of the optimisation of HIPS design is performed employing the following set of the GA parameter values: the population size equal to 50, the mutation rate equal to 0.01 and crossover rate set to 0.75. The analysis is discussed in Section 5.5.2.

5.5.2. Testing

To perform further testing the specified set of GA parameter values is used and 10 more runs are carried out. The number of generations in a single run of the programme is left the same, i.e. 100 generations.

The average fitness value in each generation is used as one of the indicating factors when considering convergence of the optimisation process. As an example the results obtained per generation for runs 3, 5, 7 and 10 are presented in *Figure 5.16*. The results demonstrate that an average population fitness value converges towards the optimal solution. Due to the random nature of the GA it is common that unfit genes are introduced in the population which results in the fluctuation in average population fitness. However the results are scattered in a relatively broad range and the convergence to a particular design can not be identified. This occurs due to the inability of the algorithm to ensure a structured random search, and instead the search tends to degenerate to unstructured enumerative technique. It was anticipated that this situation would change and populations would be dominated by highly-fit chromosomes which are similar to the best overall chromosome once a number of generations performed before termination of the process would be increased. However, after performing a number of runs when the termination condition for the optimisation process was set to 300 generations, the desired convergence was not achieved. *Figure 5.17* presents the results for three experiment runs.

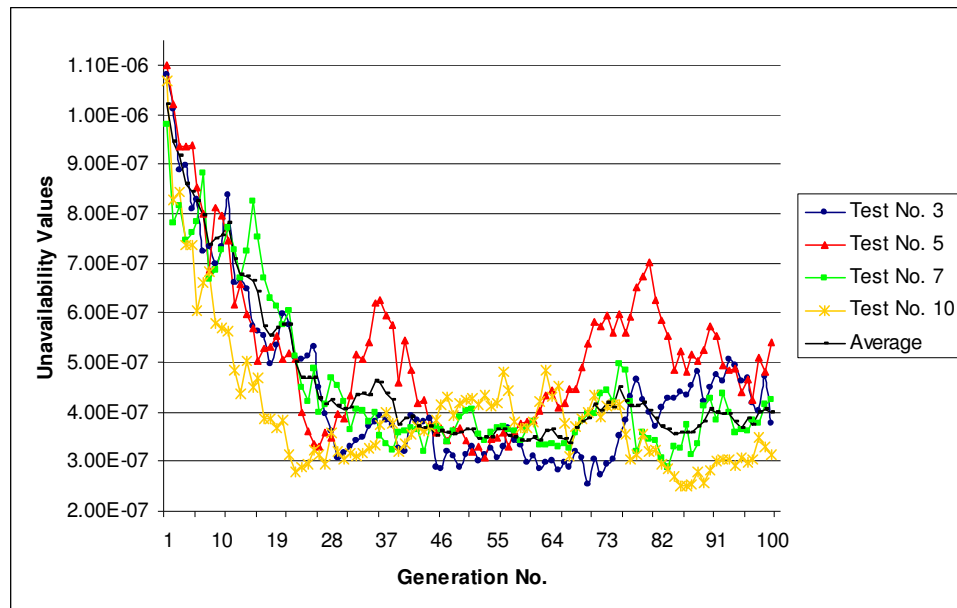


Figure 5.16. Average Unavailability Value in Each Generation. Total 100 generations

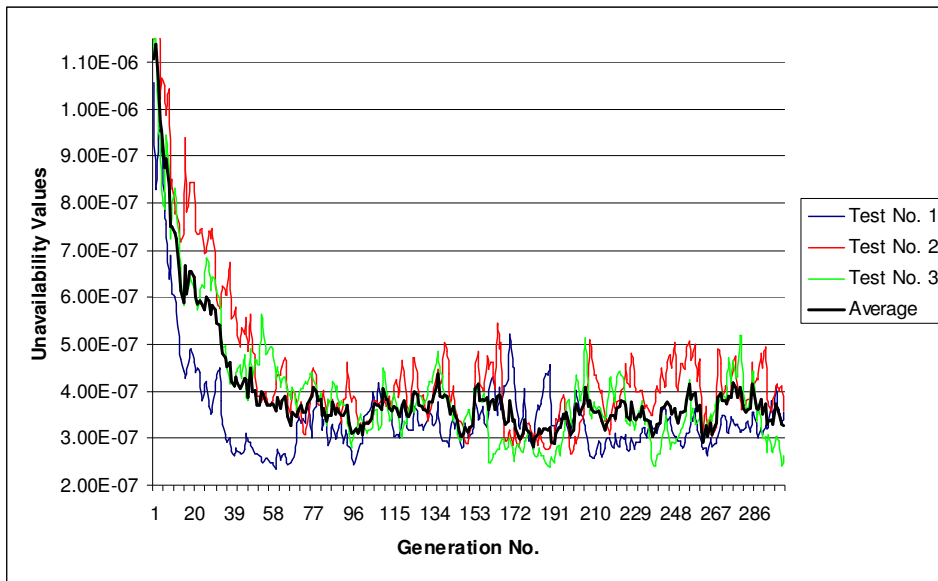


Figure 5.17. Average Unavailability Value in Each Generation. Total 300 generations

The obtained results suggest that using the current optimisation algorithm the rate at which fit chromosomes produce fit offspring and therefore the convergence rate is slow. The cause of the current situation could be the approach used to introduce new chromosomes into a new parent population. For constrained problems, the optimum solution lies on the boundary of at least one of the constraints. The chosen approach preserves genetic information of feasible solutions however it restricts the search within the infeasible region (even though it preserves good genetic information) therefore limiting the search space.

Even though the convergence of average fitness values to one particular value representing the optimal design case has not been achieved the algorithm shows the capability to find fit chromosomes and therefore near optimal design cases in the search space. *Figure 5.18* demonstrate the best feasible unavailability values obtained in each generation in 4 out of 10 runs. It shows a steady convergence of the best fitness values per generation to a particular minimum value.

The best optimisation solution over 100 generations was obtained during run no 10. The best chromosome, i.e. the chromosome that provides the smallest unavailability value arose in generation 94 which corresponds to a system unavailability value equal to 3.13E-07. This chromosome comprises the following values of genes which are represented in decimal format: 2 2 2 2 2 2 2 1 2 2 46 34. Each of them is associated with a particular design variable. Equivalentents between genes and values of design variables are listed in *Table 5.19*.

The total cost of the generated system design is 982 units and its maintenance down time is equal to 125.21 hours.

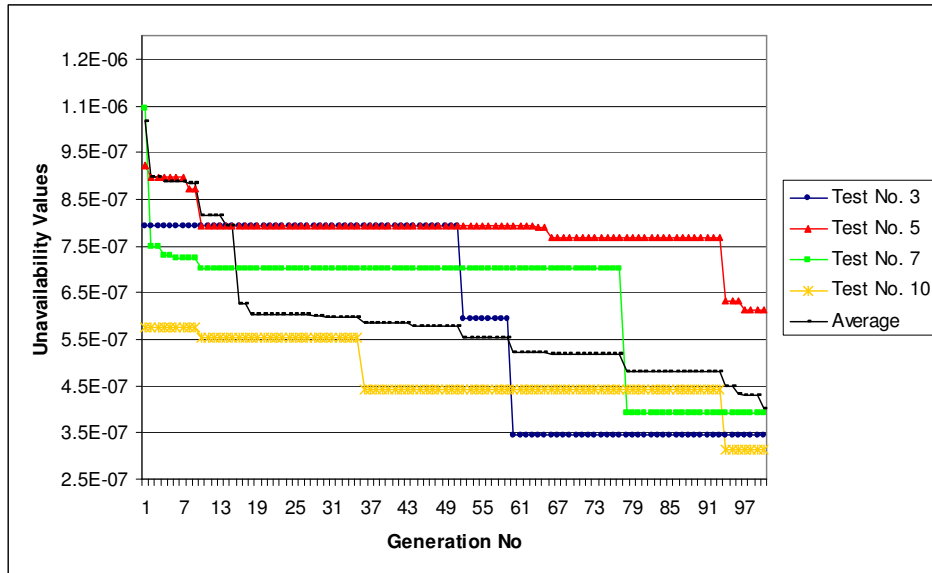


Figure 5.18. Best Fitness Value per Each Generation

Table 5.19. Design Variables Values

<i>Design Variable Values (GSDOP)</i>	<i>Design Variable Values (overall optimal)</i>	<i>Design Variable</i>	<i>Description</i>
2	2	n_3	Number of pressure transmitters for ESD subsystem
2	2	t_3	Type of pressure transmitter in ESD subsystem
2	2	k_3	Number of pressure transmitters required to trip in ESD subsystem
2	2	n_4	Number of pressure transmitters for HIPS subsystem
2	2	t_4	Type of pressure transmitter in HIPS subsystem
2	2	k_4	Number of pressure transmitters required to trip in HIPS subsystem
2	2	n_1	Number of ESD valves
1	1	t_1	Type of a ESD valve
2	2	n_2	Number of HIPS valves
2	2	t_2	Type of a HIPS valve
46	44	θ_1	Maintenance test interval for ESD subsystem (in weeks)
34	33	θ_2	Maintenance test interval for HIPS subsystem (in weeks)

For comparison, the unavailability of the initial design system is equal to $1.568\text{E-}06$, the cost is 862 units and maintenance down time is 57.2 hours when components of the ESD subsystem are tested every 71 weeks and HIPS subsystem undergoes maintenance testing every 102 weeks. The unavailability value of the overall optimal feasible system design (found using an exhaustive search) is equal to $3.05\text{E-}07$, total cost is equal to 982 units and maintenance down time equates to 129.75 hours a year. The design variable values corresponding to the overall optimal system design are presented in *Table 5.19*. Thus, using the developed GSDOP a HIPS design was identified with a significantly smaller unavailability value and design cost compared to the initial system design. The GSDOP also identified structural design variable values corresponding to the overall optimal feasible system design. Values of the optimal maintenance intervals were very similar to the ones found during the exhaustive search.

5.6. SUMMARY

In this chapter a systematic approach to system design optimisation has been demonstrated with the successful application of the developed GSDOP to solve a specific HIPS design optimisation problem. This example is the first step towards the validation of the potential of the algorithm for application to a number of safety systems. It demonstrates that the algorithm is:

- applicable to solve a chosen design optimisation problem;
- scalable according to the size of the system and the optimisation problem analysed.

In the chapter it has been demonstrated what data needs to be provided by the user and how it is utilised in the automated design optimisation process for the specific system problem. It has also been illustrated how to provide the set of FTMPs according to the chosen design options and how the programme builds the corresponding fault tree representing all possible design cases. The chromosome structure defined within the programme for the HIPS problem has also been discussed.

An analysis of the performance of the programme has demonstrated how different GA parameter values influence the optimisation results. The size of populations had the most pronounced effect on the optimisation process. Larger chromosome populations lead to better algorithm performance. In the mean time the balance between high and low values of mutation and crossover rates had to be determined in order to achieve faster convergence

towards the optimal solution. It has been noticed that a high mutation rate and a low crossover rate lead to smaller optimal solutions over a fixed number of generations.

The programme has an option that allows the user to set values for GA parameters, such as the population size, the crossover rate, the mutation rate and the maximum number of generations to be performed. Therefore the user can try different combinations of GA parameter values for obtaining an optimal solution or use the default options to find a near-optimal if not an optimal problem solution. The default population size is 50 chromosomes, the crossover rate is equal to 0.95 and the mutation rate is equal to 0.01.

The application of the GSDOP exhibits the potential of the algorithm to solve a system design optimisation problem and find the near-optimal solution. However the optimisation results have revealed that the implemented GA lacks the ability to converge to a population dominated by the fittest designs. The solution to this problem could be a number of improvements which would encourage the preservation of elite chromosomes when creating new generation populations. A new penalisation method, an approach to form a parent population and a scaling procedure are the amendments to be considered to increase the effectiveness of the GSDOA and form the future research discussed in the following chapters of this thesis.

6. FIREWATER DELUGE SYSTEM DESIGN OPTIMISATION USING IMPROVED GSDOA

6.1. INTRODUCTION

The first application of the GSDOA was demonstrated when solving the HIPS design optimisation problem discussed in *Chapter 5*. The HIPS is a relatively simple safety system and therefore when solving its optimisation problem all specifics of the developed algorithm could not be demonstrated. Moreover, the potential of the algorithm to solve the design optimisation problem for different safety systems should be realised with more than one application example. The Firewater Deluge System (FWDS) has been chosen to demonstrate both the applicability of the algorithm to a range of safety systems and its ability to analyse more complicated problems. Here the complexity of the optimisation of the FWDS is determined by the increased size of the fault tree for the initial system design and the larger number of design variables defining the size and complexity of the search space for the optimisation problem solved.

This chapter is organised as follows. First, the FWDS and its optimisation problem are introduced. System performance principles are explained along with the causes of failure and the resulting system fault tree in *Section 6.2*. The optimisation problem is formulated considering the design variables and available resources specified in *Section 6.3*. Initial data arrangements including a detailed explanation of FTMPs used to construct the fault tree for all possible design variations are discussed in *Sections 6.4*. Since the application of the GSDOP to the HIPS revealed some shortcomings of the developed optimisation methodology, improvements were made to the GA and they are discussed in *Section 6.5*. The last section of the chapter, *Section 6.6*, discusses the performance analysis of the improved GSDOP in the context of the design optimisation problem of the FWDS.

6.2. DESCRIPTION OF FIREWATER DELUGE SYSTEM

6.2.1. Performance Principles

FWDS (*Figure 6.1*) is a safety system that supplies, on demand, controlled pressurised water and foam to a particular area that is protected by a deluge system on an offshore platform. It can be used to mitigate the consequences of jet and pool fires and to reduce overpressure of an

explosion. There are three major system parts: the Deluge system, the Water Supply and Distribution System and the Aqueous Film-Forming Foam (AFFF) Supply and Distribution System.

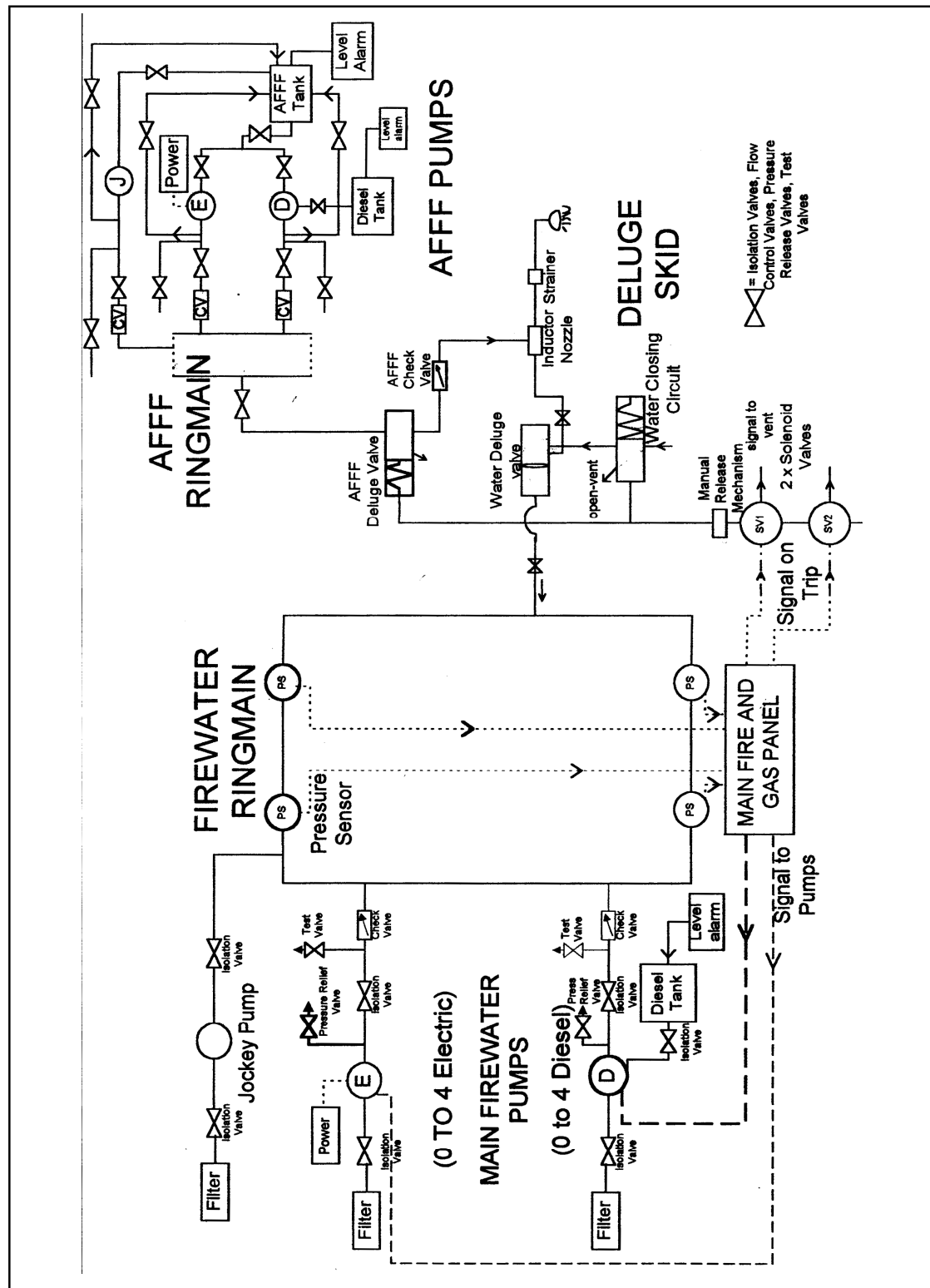


Figure 6.1. The Firewater Deluge System

The Deluge system is a fabricated steel framework, called a skid, where the deluge valve set together with all associated equipment is mounted on. The main distribution line, water closing circuit and control air circuit complete the deluge valve set. The system works in the following way. When a signal from the Main Fire and Gas Panel (MFGP) is received air pressure is released from the control air circuit by de-energising and opening solenoid valves. Air may also be released from the control air circuit manually, i.e. by opening the system local manual release valve on the skid. The air pressure drop results in the opening of the valmatic release valve and activation of the water closing circuit. Water starts running to drain followed by the pressure fall on the deluge valve diaphragm. When the pressure on the diaphragm falls to a particular set level, the firewater main pressure overcomes the load, controlled by the diaphragm, and water starts to flow into the distribution pipe through the nozzles and onto the hazard.

The deluge valve set is also fitted with an AFFF supply line. When the air pressure drops in the control air circuit of the deluge system, the AFFF and valmatic release valves open simultaneously. This allows the induction of foam concentrate from the AFFF line via the foam proportioner while the water flows through the foam inductor in the main distribution line. The distribution network is then supplied with the water and foam (approximately 3%) solution through the nozzles and onto the hazard.

The AFFF system is activated when either the air pressure drops in the control air circuit or when any firewater pump starts to supply water at the pressure level that meets the design requirements. In order to keep the AFFF system at approximately the same pressure level as the firewater system an air driven jockey pump is run continuously. The analysed FWDS has two AFFF pumps fitted. One pump is motor driven and supplied from the platform power plant. Another one is diesel driven, which is supplied from a diesel tank sized for a 24 hour supply.

The aqueous film-forming foam concentrate is stored in a tank and is distributed through the ringmain network when demand arises. The tank has to be filled to a certain level; otherwise a low level alarm fitted in the tank sends an alarm signal to the Central Control Room.

The deluge system is also connected to a pressurised ringmain network. The pressure is maintained by a jockey pump that draws water from the sea. If the ringmain pressure falls it is detected by the pressure transducer and a signal is sent to the MFGP. As a result the MFGP activates the firewater pumps to supply water at a sufficient pressure level that meets the deluge requirements. Water into the pumps is taken directly from the sea. Each firewater

pump can be activated in two ways: automatically and manually. A pump can be activated manually locally or at the fire control panel.

In the current system design two firewater pumps are fitted and they are powered in the same way as the ones for AFFF. One pump is powered from the main electric plant and the other is powered from the diesel engines. Note that diesel is supplied to the firewater diesel driven pump independently from that of the AFFF pump.

6.2.2. FireWater Deluge System Failure

For quantitative system failure analysis components in the FWDS are divided into two categories: “wear - out” or “non wear-out”. The unavailability, q , for components of “non wear-out” type is determined using *Formula 6.1*:

$$q = \lambda \left(\tau + \frac{\theta}{2} \right), \quad (6.1)$$

here λ is dormant failure rate, τ is dormant mean time to repair and θ is a maintenance test interval. Failure of components of “wear-out” type is expressed employing the Weibull probability distribution. Thus component unavailability is found using *Formula 6.2*:

$$q = \int_0^{\theta} \left(1 - e^{-\left(\frac{t}{\eta}\right)^{\beta}} \right) dt, \quad (6.2)$$

here η and β are the scale and shape parameters of the Weibull distribution respectively and t refers to time. Preventive maintenance is only carried out on components of wear-out type.

The initial system design is represented by the fault tree with the top event defined as the “Firewater Deluge System Fails to Activate on Demand”. The top event occurs as a result of any of the following failures: failure to activate both firewater and AFFF pumps mechanisms, i.e. failure of the distribution network, failure to supply sufficient amount of water to the ringmain, failure to supply sufficient foam to the ringmain or failure of the Deluge system. The event of failure to activate mechanisms of firewater and AFFF pumps is defined as “Failure to initiate pumps mechanisms” as shown in *Figure 6.2*. It occurs as a result of the coincident failure of both automatic and manual activation of the pumps. Failure of the push button on the MFGP (basic event “PBF”) or the operator’s failure to push the button (basic event “OE”) will cause failure of manual activation. Automatic mechanism activation fails if

either the firewater pump selector unit fails (basic event “FSU”) or the pressure transmitter on the firewater ringmain that senses low pressure fails (basic event “PT”).

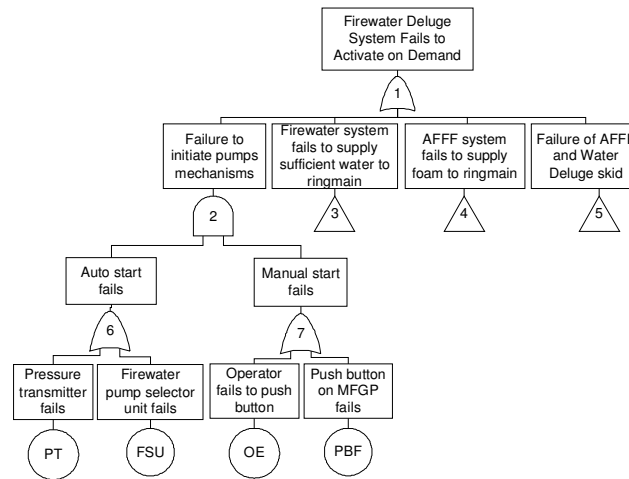


Figure 6.2. Causes of WFDS failure to activate on demand

Basic event failure data for the FWDS is provided in *Appendix 2*. Events considered in the distribution network failure process and their corresponding fault tree basic events are listed in *Table A.2.1*. Failure and repair data of each component used for quantitative analysis and information regarding the systems’ available resources are provided in *Table A.2.2*. The latter includes the number of hours of manual work required to test the component (H_T), the number of spares stored (N_S), storage cost per component (C_S) and component initial cost, i.e. design cost (C_I). Events due to human error require only specification of the probability of their occurrence. Exception applies to the failure event of the manual push button (PBF) which is a component failure however its probability of occurrence is specified directly.

If the firewater pump mechanisms or lines fail the FWDS cannot supply sufficient water to the ringmain. The fault tree structure for the failure logic is presented in *Figure 6.3*.

Considering that the system initial design has one electrically driven and one diesel driven pump, failure will occur if both pumps fail to supply water. Both pumps have similar failure scenarios. The failure occurs if the energy supply goes out of order or either the pump itself fails or components on the pump line fail. The causes of failure for the diesel driven pump and its line leading to the FWDS failure are shown in *Figure 6.3*. Accordingly, logic relations among the electrically powered pump and its line failure events are shown in *Figure 6.4*. All failure events considered leading to firewater supply failure are specified in *Table A.2.3*. The associated component data for the events is given in *Table A.2.4*.

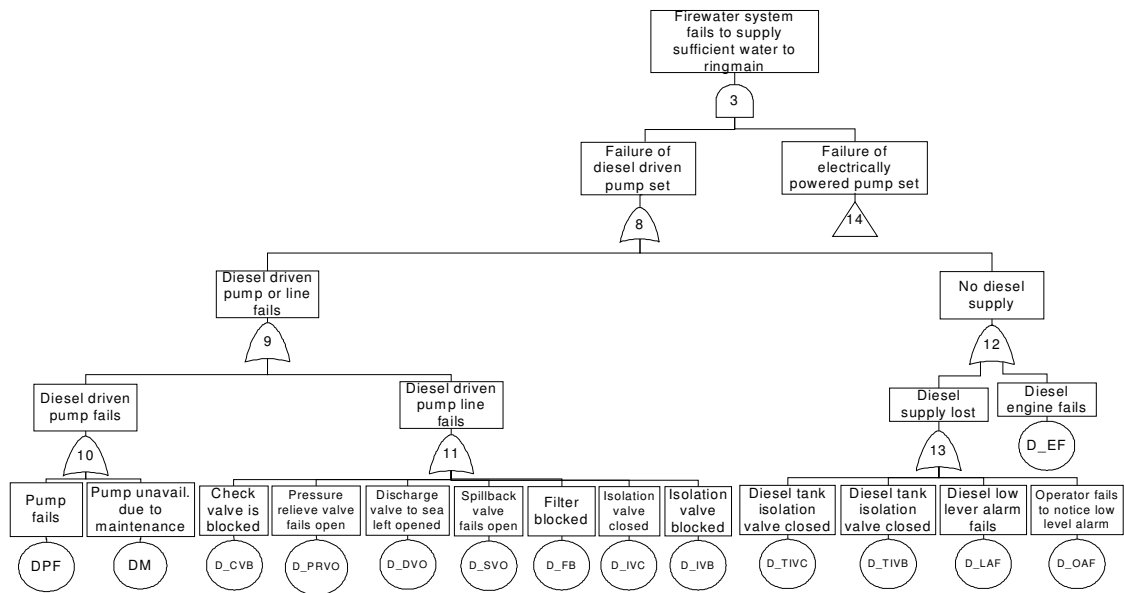


Figure 6.3. Causes of failure of firewater diesel driven pump

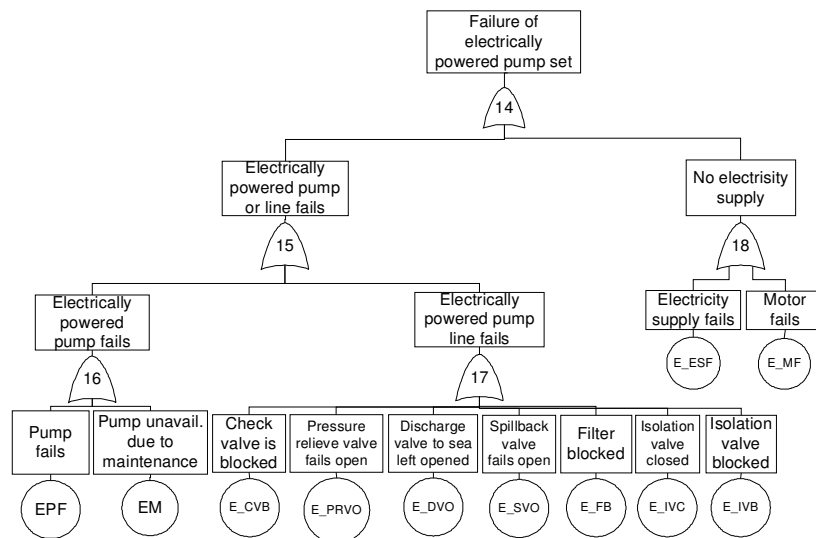


Figure 6.4. Causes of failure of firewater electrically powered pump

AFFF supply fails as the result of failures of either the AFFF pump mechanisms or pump lines or isolation of the AFFF tank. Since the system design considered has one diesel driven and one electrically powered AFFF pump the fault tree structure of the AFFF supply failure is similar to that of the firewater supply system. The fault tree is shown in Figures 6.5 and 6.6. All failure events leading to the AFFF supply system failure are listed in Table A.2.5. Associated data for these events is given in Table A.2.6 in Appendix 2.

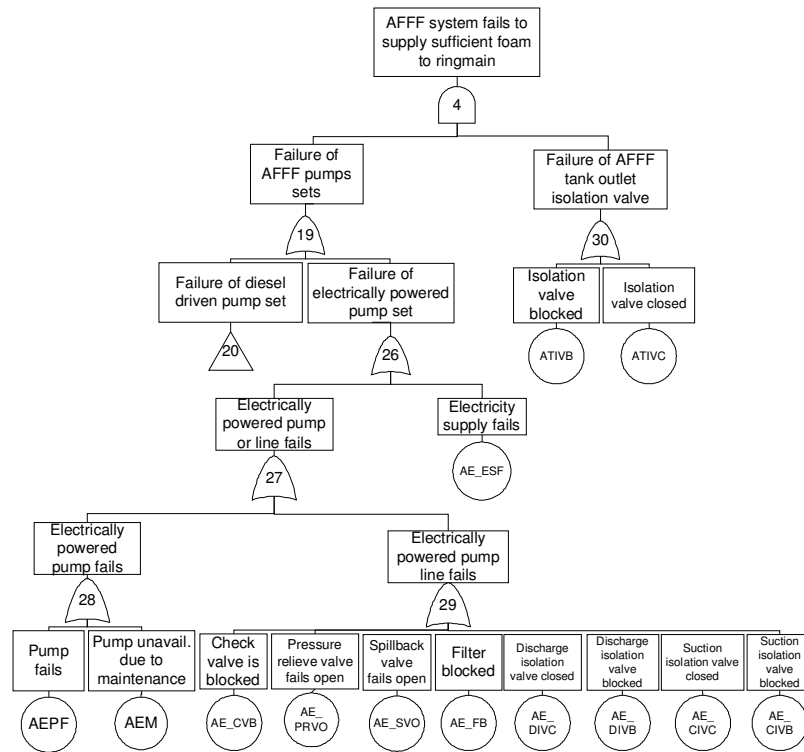


Figure 6.5. Causes of failure of AFFF electrically powered pump

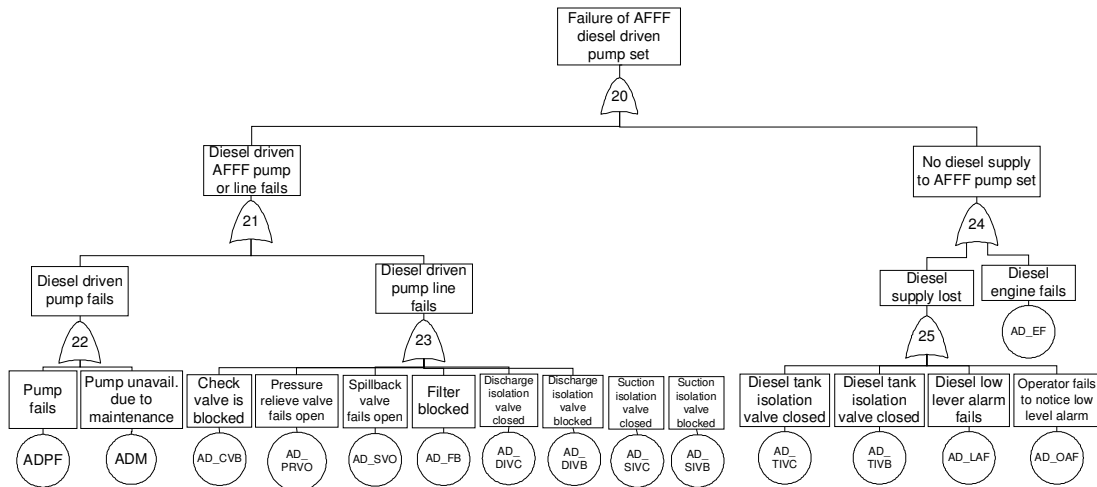


Figure 6.6. Causes of failure of AFFF diesel driven pump

The last set of causal relationships resulting in FWDS failure to activate on demand is associated with failure of the Deluge system. Its failure occurs if either the AFFF or Water deluge skid fails. The causes of failure for AFFF deluge skid are depicted in *Figure 6.7*. The corresponding fault tree of the Water Deluge skid is presented in *Figure 6.8*.

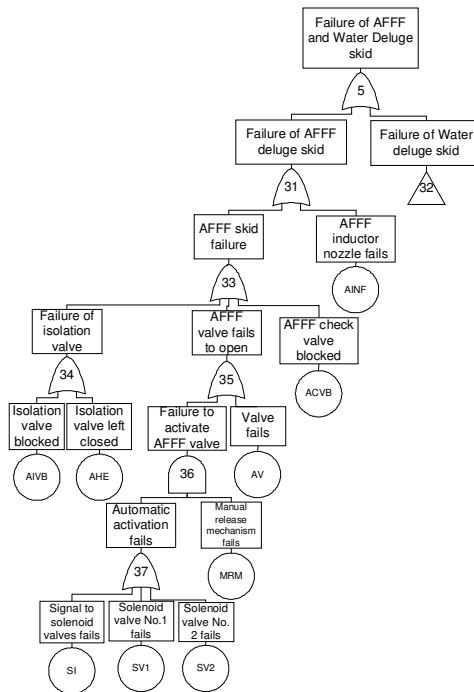


Figure 6.7. Causes of failure of AFFF deluge skid

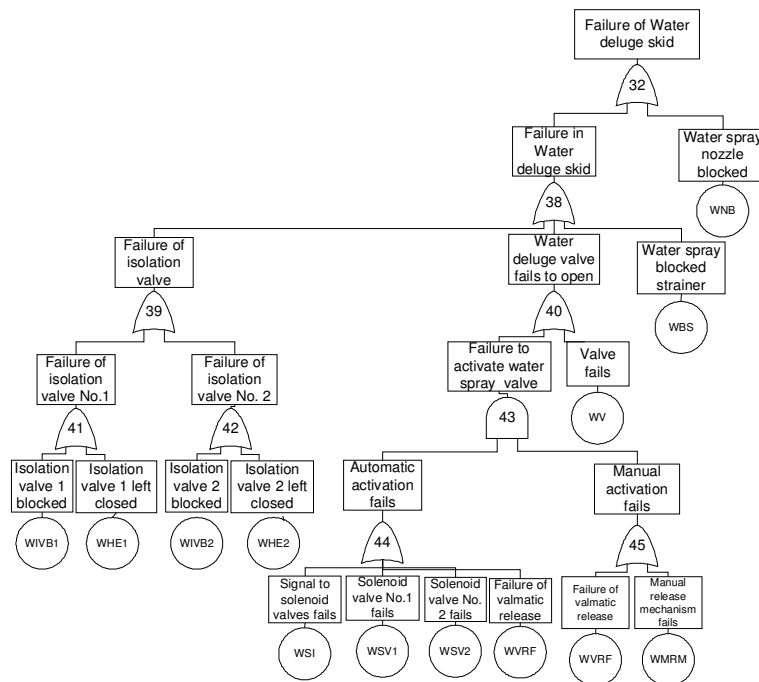


Figure 6.8. Causes of failure of Water deluge skid

There are two main events leading to the failure of the AFFF Deluge skid, such as failure of the AFFF skid itself or blockage of the inductor nozzle (basic event “AINF”). The logic of failure of the AFFF skid itself is broken down into three events. First event, “Isolation valve fails” occurs if the AFFF isolation valve is blocked (basic event “AIVB”) or it is left closed

(basic event “AHE”). The rest of the events are “AFFF valve fails to open” and “AFFF check valve is blocked” (basic event “ACVB”). The AFFF deluge valve fails to open if it fails itself (basic event “AV”) or both mechanic (basic event “MRM”) and automatic activation fail.

Accordingly, the failure of the Water Deluge skid occurs if either the water spray nozzle becomes blocked (basic event “WNB”) or the skid fails itself. Failure of the skid occurs if the strainer becomes blocked (basic event “WBS”) or either the deluge valve or one of isolation valves fails. An isolation valve fails if it is blocked or left closed. Developing further the event “Water deluge valve fails to open” requires consideration that restricts activation of the valve or failure of the valve itself (basic event “WV”). The valve is not activated if the signal to the solenoids is not sent, both fitted solenoid valves remain energised or the valmatic relieve valve fails.

Events considered in the Deluge system failure process are listed in *Table A.2.7* and the failure and repair data, initial cost and test time for each component are specified in *Table A.2.8*.

6.3. DESIGN ALTERATION OPTIONS

The introduced options for the possible FWDS design alternatives lead to a rather complicated optimisation problem. First of all, the list of structural design variables together with the choices of different maintenance intervals defines a very large set of possible candidate solutions. Secondly, the choice of the appropriate FTMPs and the order they are implemented are not straight forward. In order to construct fault tree structures incorporating certain design alternatives some of the FTMPs need to be implemented at gates which do not exist in the fault tree of the initial design and are created during the fault tree modification process. Finally, the resulting fault tree representing all possible design cases increases in size dramatically. It therefore increases the demand on computational resources.

The list of structural design variables to be used for possible FWDS alterations is provided in *Table 6.1*. The choice of values of time intervals for three different maintenance performance tasks is also considered. The maintenance test interval for the firewater and AFFF pump sets (θ_P) can vary from 1 to 28 days. Maintenance of the ringmain (θ_R) can be performed at time intervals ranging from 1 to 24 weeks. The Deluge skid can undergo maintenance at time intervals (θ_D) from 3 and up to 18 months.

Table 6.1. Structural Design Variables

<i>Associated System Component</i>	<i>Description of Design Alteration</i>	<i>Design Variable Notation</i>	<i>Possible Values</i>	<i>Initial Design</i>
Firewater diesel driven pump	Number of firewater diesel driven pumps	n_{FD}	1, 2, 3, 4	1
	Pump capacity	$t1_{FD}$	100%, 50%, 33 1/3%	100%
	Type of firewater pump (for the 50% and 33 1/3% capacity pumps only)	$t2_{FD}$	type1/ type2	type1
Firewater electrically powered pump	Number of firewater electrically powered pumps	n_{FE}	1, 2, 3, 4	1
	Pump capacity	$t1_{FE}$	100%, 50%, 33 1/3%	100%
	Type of firewater pump (for the 50% and 33 1/3% capacity pumps only)	$t2_{FE}$	type1/ type2	type1
AFFF diesel driven pump	Number of AFFF diesel driven pumps	n_{AD}	1, 2	1
	Pump capacity	t_{AD}	100%, 50%	100%
AFFF electrically powered pump	Number of AFFF electrically powered pumps	n_{AE}	1, 2	1
	Pump capacity	t_{AE}	100%, 50%	100%
Pressure transmitter on the ringmain	Number of pressure transmitters	n_{PT}	1, 2, 3, 4	1
	Minimum number of pressure transmitters required to function	k_{PT}	1, 2, 3, 4	1
	Pressure transmitter type	t_{PT}	type1/ type2/ type 3	type1
Inductor nozzle	Inductor nozzle material type	t_{IN}	old/ new	old
AFFF deluge valve	AFFF deluge valve type	t_{AD}	type1/ type2/ type3	type1
Deluge nozzle	Deluge nozzle material type	t_{DN}	old/ new	old
Water deluge valve	Water deluge valve type	t_{WD}	type1/ type2/ type 3	type1
Valmatic relief valve	Valmatic relief valve material type	t_{VR}	old/ new	old

Implementation of the listed structural design variables results in the fault tree structure incorporating failure events of new components. The data of the new components to be used for the quantitative failure analysis and evaluation of other characteristics of the introduced system designs is provided in *Table A.2.9* in *Appendix 2*. Note that the notation of the basic events used in the table differs from the one required for the GSDOP. Providing data for the GSDOP the new components will be coded following the introduced rules as applied to the HIPS (*Chapter 5*).

The following requirements are also introduced regarding limitations imposed on new system design cases: 1) total system design cost cannot exceed 81000 units; 2) maintenance down time must be less than 30 days per year.

As discussed in *Chapter 4*, specific formulas are implemented in the GSDOA to evaluate system design characteristics such as cost and maintenance down time. *Equation 4.4* which is

used for evaluation of system cost requires cost data for each component. For the analysis of FWDS these values are not provided and need to be evaluated. Design cost for each component is evaluated by multiplying the given number of spares stored (N_S) by the storage cost (C_S) and adding it up to the given initial cost, i.e.:

$$\text{cost}_{d_i} = C_{iS} \cdot N_{iS} + C_{iI}, \quad (6.1)$$

where i identifies a component. Before the analysis a number of parameters also need to be specified in order to use *Equation 4.14* to evaluate the system maintenance down time. One of them is the total number of time units per examined time period, U_T . In the case analysed it is equal to 365 days since a day is the smallest time unit used for defining maintenance intervals.

6.4. DATA ARRANGEMENT FOR THE OPTIMISATION

The data preparation for the FWDS optimisation problem is organised according to the general requirements regarding the application of the GSDOP. The data arrangement in the files such as *fwds_fts.txt*, *fwds_bse.aqd*, *fwds_cost_cst.txt*, *fwds_mdt_cst.txt*, *fwds_theta.txt* and *fwds_gav.txt* is very similar to the one used for the HIPS system. Therefore to avoid repetitiveness in this section the main focus is only given to the data preparation for construction of the fault tree representing all possible design cases.

As it is known fault tree gates and events are identified using coded numbers in all data files. Gates of the fault tree of the initial FWDS design are coded as shown in *Figures 6.2 -6.8*. Code numbers for basic events are listed in *Table 6.2*.

Table 6.2. Basic Event Codes

<i>Event</i>	<i>Code</i>	<i>Event</i>	<i>Code</i>	<i>Event</i>	<i>Code</i>	<i>Event</i>	<i>Code</i>
PT	1	E_CVB	21	AD_TIVC	41	AHE	61
FSU	2	E_PRVO	22	AD_TIVB	42	AV	62
OE	3	E_DVO	23	AD_LAF	43	MRM	63
PBF	4	E_SVO	24	AD_OAF	44	SI	64
DPF	5	E_FB	25	AE_ESF	45	SV1	65
DM	6	E_IVB	26	AEPF	46	SV2	66
D_CVB	7	E_IVC	27	AEM	47	WNB	67
D_PRVO	8	E_ESF	28	AE_CVB	48	WBS	68
D_DVO	9	E_MF	29	AE_PRVO	49	WIVB1	69
D_SVO	10	APFD	30	AE_SVO	50	WHE1	70
D_FB	11	ADM	31	AE_FB	51	WIVB2	71
D_IVB	12	AD_CVB	32	AE_DIVB	52	WHE2	72
D_IVC	13	AD_PRVO	33	AE_DIVC	53	WV	73
D_EF	14	AD_SVO	34	AE_SIVB	54	WVRF	74

<i>Event</i>	<i>Code</i>	<i>Event</i>	<i>Code</i>	<i>Event</i>	<i>Code</i>	<i>Event</i>	<i>Code</i>
D_TIVC	15	AD_FB	35	AE_SIVC	55	WSI	75
D_TIVB	16	AD_DIVB	36	ATIVB	56	WSV1	76
D_LAF	17	AD_DIVC	37	ATIVC	57	WSV2	77
D_OAF	18	AD_SIVB	38	AINF	58	WMRM	78
EPF	19	AD_SIVC	39	ACVB	59		
EM	20	AD_EF	40	AIVB	60		

The FTMPs for the construction of the FWDS fault tree representing all design cases to be considered are defined according to the given list of structural design variables. There are ten FWDS components chosen to be replaced with new components or component sets and in order to construct the fault tree corresponding to the relating changes eighteen FTMPs are employed. These are given in data file *fwds_var.txt* (Figure 6.9). As a reminder, each row in the data file is associated with one FTMP. The first letter identifies if replacement is performed at gate or at event level which is followed by a gate/event code. The three numbers represent parameter values of the corresponding FTMP.

```
G 10 1 3 1
G 59 1 2 1
G 69 1 2 1
G 16 1 3 1
G 223 1 2 1
G 239 1 2 1
G 9 4 1 1
G 15 4 1 1
G 22 1 2 1
G 21 2 1 1
G 28 1 2 1
G 27 2 1 1
E 1 4 3 1
E 58 1 2 1
E 62 1 3 1
E 67 1 2 1
E 73 1 3 1
E 74 1 2 1
```

Figure 6.9. Data File *fwds_var.txt*

In this case the order the FTMPs are implemented is very important. For instance, to implement the design alteration regarding the firewater diesel driven pump four FTMPs are required and they need to be applied in a specific order. Moreover, two out of the four FTMPs are implemented in the fault tree branches created using the first FTMP. (The step by step implementation of the patterns will be discussed later in the next paragraph). It means code numbers of the events in the new fault tree part need to be known. It is relatively easy to follow the numbering of new gates and events when implementing the first FTMPs. However after a number of alterations it might be very complicated to identify the codes manually.

Therefore in this case the FTMPs which should be implemented in the added new fault tree structures are listed at the top of the list.

The first three FTMPs from the list (*Figure 6.9*) are used to make alterations to the fault tree to represent possible changes to the capacity and type of the firewater diesel driven pump. First the fault tree is altered to introduce the choice of different capacity diesel driven pump, i.e. 100%, 50% and 33 1/3% capacity. Since there are two basic events related to the failure of the pump *Pattern3* ($mn=1, mt=3, mk=1$) is implemented at Gate 10. The resulting fault tree structure contains failure events of the 50% and 33 1/3% capacity pumps. Gates 59 and 69 represent failure of 50% and 33 1/3% capacity pumps respectively. Thus to implement a further choice of a different type and either 50% or 33 1/3% capacity pump *Pattern3* ($mn=1, mt=2, mk=1$) is implemented twice at the latter gates.

The same design alternatives are introduced with regards to the replacement of the firewater electrically-powered pump. Thus, two FTMPs need to be implemented within the newly added fault tree structure after the introduction of different capacity pumps. Therefore instead of continuing to implement the redundancy strategy of the diesel driven pump, first FTMPs are introduced to represent the causes of failure when different type and capacity electrically-powered firewater pump(s) are used. *Pattern3* ($mn=1, mt=3, mk=1$) is implemented at gate number 16 which results in the new fault tree structure representing possible choice of three capacity pumps. Following *Pattern3* ($mn=1, mt=2, mk=1$) is applied twice at gates 223 and 239. It leads to the fault tree structure where either failure of a type1 or type2 pump can be considered for a 50% or 33 1/3% capacity pump installed.

Fault tree alterations for the implementation of different numbers of redundant firewater pumps are completed using two *Pattern1* FTMPs. First *Pattern1* ($mn=4, mt=1, mk=1$) is used to incorporate a fault tree part which represents different redundancy levels for the diesel-driven pump. When introducing a redundant pump it is installed together with a new pump line. Either failure of the pump itself or failures of components on the pump line will result in the failure of the redundant supply of water to the ringmain. Therefore *Pattern1* is applied at gate level, gate 9. Accordingly, the second *Pattern1* ($mn=4, mt=1, mk=1$) is applied at gate 15 to incorporate a fault tree structure representing the causes of failure for different numbers of redundant electrically powered pumps and their lines. Since the FTMPs associated with different pump capacities and types were applied first and followed by the introduced redundancy alternatives, the redundancy strategy is applied to each pump with different capacity and type.

Next four FTMPs are used to implement the design alterations regarding AFFF pumps. First a diesel driven pump is considered. In this case both the number of pumps is being increased and the choice of different capacity pumps is introduced. If the causes of pump failure were associated only with the pump itself then one FTMP could be used. However, each redundant pump is installed with a pump line and failures of the components on the line need to be taken into consideration. Therefore first *Pattern3* ($mn=1, mt=2, mk=1$) is applied at gate level, gate 22, to represent the causes of failure for different capacity diesel driven AFFF pumps. Following this, *Pattern1* ($mn=2, mt=1, mk=1$) is implemented leading to the introduction of design cases with redundant sets of AFFF pumps and their lines to supply foam into the ringmain. Accordingly *Pattern3* and *Pattern1* are implemented at gates 28 and 27 respectively to represent the design changes regarding the AFFF electrically powered pump.

The rest of the FTMPs can be implemented in any order. In the presented cases *Pattern5* ($mn=4, mt=3, mk=4$) is implemented first at event level. Basic event No. 1 “Pressure transmitter fails” is replaced with a fault tree structure representing the possible design cases when up to four pressure transmitters can be installed. The number of transmitters that must work can vary from one to four and either of different types of transmitters can be installed. Next, *Pattern3* is implemented at the following event levels, event 58 ($mn=1, mt=2, mk=1$), event 62 ($mn=1, mt=3, mk=1$), event 67 ($mn=1, mt=2, mk=1$), event 73 ($mn=1, mt=3, mk=1$) and event 74 ($mn=1, mt=2, mk=1$) with regards to the choices of different material type or valve type for the AFFF inductor nozzle, AFFF deluge valve, waterspray deluge nozzle, water deluge valve and valmatic relief valve respectively.

After the implementation of all FTMPs the number of gates in the FWDS fault tree increases from 45 to 317, the number of basic events increases from 78 to 242 and 131 house events are included.

The structure of chromosomes to be employed is problem-dependant and is defined by the programme according to the FTMPs introduced and maintenance time intervals considered as design variables. There are twenty genes in the chromosome coding the parameters of the chosen FTMPs as shown in *Figure 6.10*. The first six genes, each one comprising of two bits, correspond to *Pattern3* which was implemented six times. In all those genes parameter mt is coded with its value equal to either 2 or 3. The seventh and eighth genes code parameters $mn=4$ of *Pattern1* which is applied twice. The next four genes each of the size of 2 bits code parameters $mt=2$ (*Pattern3*), $mn=2$ (*Pattern1*), and $mt=2$ (*Pattern3*), $mn=2$ (*Pattern1*) which represent design alterations of the AFFF diesel-driven and AFFF electrically-powered pumps

respectively. Three genes of sizes 3, 2 and 3 bits represent the parameters $mn=4$, $mt=3$ and $mk=4$ of *Pattern5*. The remaining five genes are allocated to code the parameters mt for the last five FTMPs implemented.

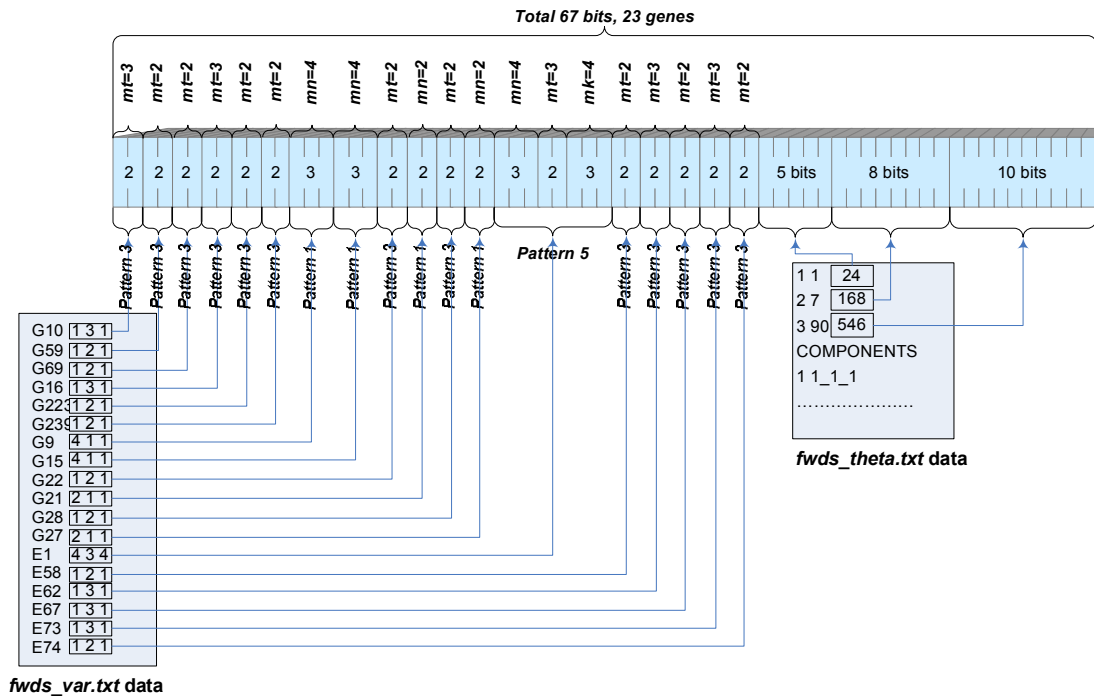


Figure 6.10. Chromosome Structure

The last three genes in the chromosome are utilized to code values of three time intervals between maintenance activities. Time units for different time intervals are unified and a day has been used as a time unit. Thus to code the maintenance test interval θ_P , a five bit gene is used. Maintenance of the ringmain (θ_R) can be performed at time intervals ranging from 1 to 24 weeks or from 7 to 168 days. Therefore eight bits are allocated to code the possible values. Finally, a gene of 10 bits is allocated to code the maximum value of the time interval, which is 546 days (18 months) between any maintenance performance for the Deluge skid (θ_D).

6.5. MODIFICATION OF THE GSDOA

Application of the GSDOA for the optimisation of the HIPS design discussed in *Chapter 5* has revealed the issue of the algorithms ability to converge to a population dominated by the fittest designs. It was envisaged that when analysing larger scale systems or solving optimisation problems with a large search space the non-convergence problem would be even more pronounced. In order to improve the performance of the algorithm a number of modifications have been implemented in the algorithm. A penalty function approach for the

handling of constraint violations and a new replacement procedure were introduced (Section 6.5.1). Furthermore, scaling of the fitness function has been implemented (Section 6.5.2).

6.5.1. Penalty Function Method and New Replacement Procedure

In the initial GSDOA a death penalty strategy was implemented. Using this approach an offspring chromosome which violated any constraint was not entered into the new parent population and one of its parent chromosomes was entered instead. Such a strategy restricts the search region by allowing the selection of just those population members which do not violate any constraint. Although evolutionary strategies normally employ death penalties their use may eliminate good genetic material by eliminating those population members which are close to being feasible. Therefore an approach of applying some penalties to solutions that violate one or more constraints has been introduced.

Applying a penalty method a constrained optimisation problem is transformed into an unconstrained one by associating a penalty, i.e. by adding (or subtracting) a certain value to (or from) the objective function, for any constraint violation. The basic approach is to define a fitness value for an individual i by extending the domain of the objective function $F^i(\mathbf{X})$ as follows:

$$F_{fitness}^i(\mathbf{X}) = F^i(\mathbf{X}) \pm F_p^i \quad (6.3)$$

where F_p^i represents either a penalty for an infeasible individual i , or a cost for repairing such individual, i.e. the cost for making it feasible [78]. If a minimisation problem is considered the value of an objective function is increased by the magnitude of a penalty.

A number of penalty function categories exist. Coello mentions the following ones [79]: static, dynamic, annealing, adaptive, co-evolutionary and the previously employed death penalty. The choice of a new penalty method to be implemented has been made according to properties of the algorithm, i.e. considering prospective application of the approach to a variety of safety system design optimisation problems. Therefore the chosen penalty method had to be problem-independent as much as possible and only constraint-specific. For this reason the following form of a general penalty function was chosen which was introduced by Coit *et. al.* [80]:

$$F_p(\mathbf{x}) = (F_{all} - F_{feas}) \sum_{i=1}^{nc} \left(\frac{d_i(\mathbf{x}, B)}{NFT_i} \right)^{\kappa_i}. \quad (6.4)$$

Here F_{all} is the best unpenalised value of the objective function yet found, F_{feas} is the best feasible value of the objective function yet found, NFT_i denotes the near-feasibility threshold that corresponds to a given constraint i , $d_i(\mathbf{x}, B)$ is a magnitude of violation of a given constraint i for solution \mathbf{x} , κ_i is the user-specified severity parameter and nc is the total number of constraints set for the problem.

The general form of the penalty function (Formula 6.4) has been specified and modified for the developed GSDOA. Four pairs of constraints defining maximum and minimum limits of considered factors are implemented in the algorithm. Thus the maximum possible value of variable nc is equal to eight. The severity parameter is set to 2 for every constraint resulting in a penalty of the square of the Euclidean distance from the infeasible solution to the feasible region over all constraints. In the implemented algorithm the dynamic form of the near feasible threshold has been used, as it allows the penalty value to be adjusted according to the search history. The near-feasibility threshold for each constraint is defined as follows:

$$NFT_i = \frac{NFT_{oi}}{1 + 0.1 \cdot g}. \quad (6.5)$$

Here NFT_{oi} represents the actual value of a constraint i and g denotes the generation number.

The general form of the penalty function adjusted to the implemented optimisation approach can be defined for the case when all constraints are violated as follows:

$$F_p(\mathbf{x}) = (F_{all} - F_{feas}) \cdot \left[\left(\frac{\Delta Cost_{max}}{NFT_{Cost_{max}}} \right)^2 + \left(\frac{\Delta Cost_{min}}{NFT_{Cost_{min}}} \right)^2 + \left(\frac{\Delta MDT_{max}}{NFT_{MDT_{max}}} \right)^2 + \left(\frac{\Delta MDT_{min}}{NFT_{MDT_{min}}} \right)^2 + \left(\frac{\Delta Volume_{max}}{NFT_{Volume_{max}}} \right)^2 + \left(\frac{\Delta Volume_{min}}{NFT_{Volume_{min}}} \right)^2 + \left(\frac{\Delta Weight_{max}}{NFT_{Weight_{max}}} \right)^2 + \left(\frac{\Delta Weight_{min}}{NFT_{Weight_{min}}} \right)^2 \right], \quad (6.6)$$

where Δ is the magnitude of violation of the constraint, i.e. the difference between the actual constraint value of a generated system design and the defined constraint value. For example:

$$\Delta Cost_{\max} = Cost_{system} - Cost_{\max} \cdot \quad (6.7)$$

Thus each individual in a population is evaluated using the formula:

$$F_{fitness}(\mathbf{x}) = F(\mathbf{x}) + F_p(\mathbf{x}),$$

$$F_{fitness}(\mathbf{x}) = F(\mathbf{x}) + (F_{all} - F_{feas}) \left[\left(\frac{\Delta Cost_{\max}}{NFT_{Cost \max}} \right)^2 + \left(\frac{\Delta Cost_{\min}}{NFT_{Cost \min}} \right)^2 + \left(\frac{\Delta MDT_{\max}}{NFT_{MDT \max}} \right)^2 + \left(\frac{\Delta MDT_{\min}}{NFT_{MDT \min}} \right)^2 + \left(\frac{\Delta Volume_{\max}}{NFT_{Volume \max}} \right)^2 + \left(\frac{\Delta Volume_{\min}}{NFT_{Volume \min}} \right)^2 + \left(\frac{\Delta Weight_{\max}}{NFT_{Weight \max}} \right)^2 + \left(\frac{\Delta Weight_{\min}}{NFT_{Weight \min}} \right)^2 \right] \quad (6.8)$$

If certain constraints are not violated the corresponding summands are eliminated from Equation 6.8 automatically within the developed programme.

To implement the penalisation methodology in the optimisation programme two new routines were added: *Best* and *Penalty_Adaptive*. Routine *Best* is used to identify the best unpenalised value of the objective function and the best feasible value of the objective function in a population yet found. The second routine *Adaptive_Penalty* estimates any constraint violations and returns the value of the penalty magnitude F_p .

With the new fitness penalisation methodology introduced the previously used replacement approach also needs to be modified. Several replacement methods used for construction of the parent population for the generation of new chromosomes are possible depending on the type of GA being used. An accurately chosen replacement methodology can be beneficial to improve the algorithm. It can prevent insufficient diversity in the population which can lead the algorithm converging too quickly towards a weak solution.

The new replacement strategy was implemented by employing an algorithm described by Chambers [68]. The idea of this algorithm is to replace a parent population with an offspring population. If the best parent chromosome is fitter than the best offspring chromosome then it replaces the worst offspring chromosome. This is performed every time a new offspring population is generated.

The influence of the alterations made on the algorithm performance has been analysed. The initial GSDOP and the one with the newly implemented penalisation and replacement methodologies were applied to solve the FWDS design optimisation problem. Two sets of GA

parameter values were chosen to compare the influence of different GA parameter values on the performance of the algorithms. The first set used includes a population size equal to 10 chromosomes, crossover rate equal to 0.75 and mutation rate equal to 0.001. From the earlier analysis of the HIPS system (*Chapter 5*) it is known that the algorithm performs better if the population size and mutation rate are increased. Therefore the next set of results chosen includes a population of 50 chromosomes with crossover rate and mutation rate equal to 0.95 and 0.01 respectively. Five runs were performed for each set of parameters. A minimal system failure probability value, i.e. best feasible fitness value, after 100 generations was considered as the problem solution. The obtained results are presented in *Table 6.3*.

Table 6.3. Results of the Application of the Initial and the Improved GSDOP

	<i>Crossover Rate=0.75, Mutation Rate=0.001, Population Size=10,</i>		<i>Crossover Rate=0.95, Mutation Rate=0.01, Population Size=50</i>	
	<i>Original SOGA</i>	<i>Improved SOGA</i>	<i>Original SOGA</i>	<i>Improved SOGA</i>
Run No.1	0.1701	0.1483	0.1071	0.0928
Run No.2	0.1153	0.1036	0.1059	0.0938
Run No.3	0.1643	0.1044	0.1064	0.0965
Run No.4	0.1646	0.1284	0.1082	0.0936
Run No.5	0.1676	0.1172	0.1082	0.0952
Average	0.1564	0.1204	0.1072	0.0944

The GSDOP with new the penalisation and replacement routines introduced achieves the best fitness values which on average are smaller than the ones achieved using the original GSDOP. Results are improved by 23% and 12% using the first and the second sets of GA parameters correspondingly. Moreover, the difference in the best fitness values achieved for two sets of GA parameters decreases from 31% to 22% after implementing the alterations to the initially introduced algorithm. It demonstrates the lower sensitivity to variation in GA parameter values and increased robustness of the algorithm.

6.5.2. Fitness Scaling

Research shows that linear fitness scaling improves the performance of GA algorithms and it is especially valuable when small populations are utilised. Scaling of the fitness function has been introduced to avoid two problems that can occur during the optimisation process when employing GAs. The first one represents the situation when a few extra-ordinary individuals appear among other less-fit individuals in the population at early generations. This could lead to a situation in which the chromosomes with extraordinary fitness values take over a significant portion of the population in just a few generations, leading the process to premature convergence. The other problem may occur in later generations when the average

fitness value of the whole population is close to the fitness value of the best individual in the population. If this situation is left alone, average members will contribute a similar number of copies in the future generations and survival of the fittest, which is necessary for improvement becomes a random walk among the less-fit solutions.

The chosen scaling method introduced in [45] defines a linear relationship between an initial fitness value and fitness value after the scaling. It calculates the scaled fitness score for a given chromosome i , $F_{scaled}(\mathbf{x})$, using the chromosome fitness value, $F(\mathbf{x})_{fitness}$, equal to its objective function value if the chromosome is feasible or penalised objective function value (Equation 6.8) if the chromosome is not feasible as follows:

$$F(\mathbf{x})_{scaled} = a \cdot F(\mathbf{x})_{fitness} + b \quad (6.9)$$

Here parameters a and b are linear function coefficients and are problem-independent. These parameters depend on a population and are re-evaluated in each generation.

In the implemented method the linear function coefficients are selected so that the average fitness before scaling and the average scaled fitness values are equal ($f_{avg\ scaled} = f_{avg\ initial}$). The second condition which parameters need to satisfy can be described as follows:

$$f_{max\ scaled} = c_{mult} f_{avg\ initial} \quad (6.10)$$

where c_{mult} defines the number of expected chromosome copies desired for the best member in the population. Goldberg in [45] suggests that the c_{mult} value would be from 1.2 to 2 for populations where the number of chromosomes is from 50 to 100. Thus, the value for variable c_{mult} has been fixed to 1.2 in the scaling procedure implemented in the modified GSDOP.

In some cases negative fitness values can be introduced if using the linear scaling procedure. To avoid such situations equality between average initial fitness and average scaled fitness is maintained and the minimum fitness value condition is introduced. Thus the parameters a and b need to be defined such that the following non-negative test condition would be satisfied:

$$f_{min\ initial} < \frac{(c_{mult} f_{average\ initial} - f_{max\ initial})}{c_{mult} - 1} \quad (6.11)$$

The minimum initial fitness is then mapped to a scaled fitness equal to 0, i.e. $f_{min\ scaled} = 0$.

The described scaling procedure was implemented in the GSDOP using three new subroutines: *Prescale*, *Scale* and *Scalepop*. The subroutine *Scalepop* is the main subroutine that governs the scaling process and also includes the other two. The subroutine *Prescale* is employed to calculate the linear function coefficients a and b . The subroutine *Scale* is used to scale each chromosomes fitness value at a time.

For comparison the results of simulations performed are given in *Table 6.4*.

Table 6.4. Results of the Application of the Initial and the Improved GSDOP

	<i>Crossover Rate=0.75, Mutation Rate=0.001, Population Size=10,</i>		<i>Crossover Rate=0.95, Mutation Rate=0.01, Population Size=50</i>	
	<i>Improved SOGA</i>	<i>Improved (scaling implemented) SOGA</i>	<i>Improved SOGA</i>	<i>Improved (scaling implemented) SOGA</i>
Run No.1	0.1483	0.0945	0.0928	0.0928
Run No.2	0.1036	0.1515	0.0938	0.0925
Run No.3	0.1044	0.1025	0.0965	0.0924
Run No.4	0.1284	0.1144	0.0936	0.0924
Run No.5	0.1172	0.0937	0.0952	0.0932
Average	0.1204	0.1113	0.0944	0.0926

The results show the effectiveness of the scaling procedure in producing fitter chromosomes over a fixed number of generations. The results obtained also demonstrate lower sensitivity to different values of GA operators. Moreover the application of the scaling reduces the fluctuation within average population fitness values, leading to lower average fitness values and steady convergence towards the optimal solution as shown in *Figure 6.11*.

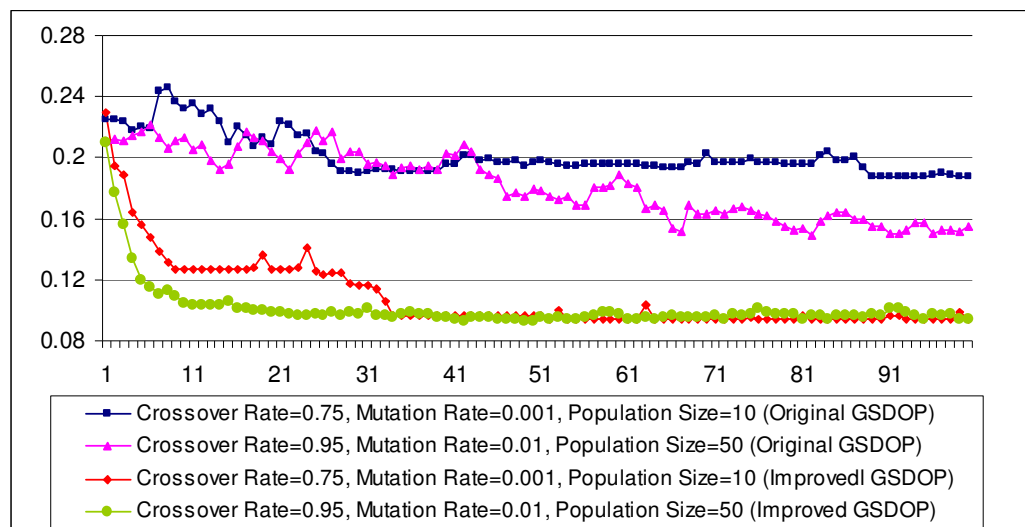


Figure 6.11. Average Fitness Values per Generation

6.5.3. Structure of the Improved SOGA

After the implementation of the introduced alterations the structure of the original SOGA utilised in the GSDOA has changed. The new SOGA (Figure 6.12) now includes two new procedures: penalisation and scaling. Additionally, alterations have also been made to the replacement procedure.

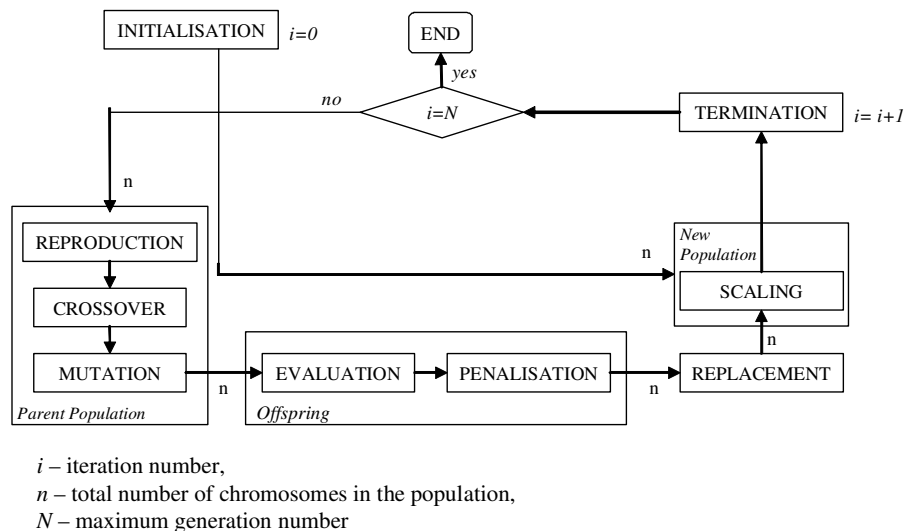


Figure 6.12. Structure of the improved SOGA

Thus the initial population in the improved SOGA is generated through the following routine:

Step 1: Assign a random binary number for each gene in a chromosome.

Step 2: Check if the obtained phenotype of each gene does not exceed the predefined maximum value. If the generated value of any gene is bigger than the maximum possible value of the corresponding parameter then generate a new binary number and checked the validation again. Repeat until all the generated parameter values are valid.

Step 3: Use the obtained parameters of the design variables to construct a corresponding design case.

Step 4: Evaluate the objective function value for the design generated.

Step 5: Evaluated system resources for the design generated. If they do not exceed the predefined limits *Step 6* follows. Otherwise penalise the objective function value by the corresponding magnitude and proceed to *Step 6*.

Step 6: The chromosome enters the initial population.

Step 7: If the number of chromosomes in the population is equal to the predefined value N the process is terminated. Otherwise the process is repeated from *Step 1*.

Penalisation and scaling are incorporated in the cycle for generation of offspring populations. In the new GA a new population of chromosomes is produced by utilising three genetic operators as in the originally used GA algorithm. Once the new chromosomes are generated the objective function value for each of them is evaluated followed by the inspecting whether or not the predefined constraints are violated. If at least one constraint is violated an objective function value corresponding to that chromosome is penalised. Next, the new replacement procedure follows. The resulting population comprises of offspring chromosomes and one or more parent chromosomes if the best parent chromosome is fitter than the best offspring chromosome. Fitness values of the chromosomes in the newly formed population undergo the scaling procedure. At this stage one generation cycle is completed. The detailed process is repeated until the number of performed generations is equal to a predefined maximum number of generations.

6.6. ANALYSIS OF THE IMPROVED GSDOP PERFORMANCE SOLVING THE FWDS DESIGN OPTIMISATION PROBLEM

6.6.1. Analysis of GA Parameters Influence on Algorithm Performance

Different sets of GA parameter values have been used to solve the FWDS optimisation problem and to analyse performance of the improved GSDOP with new approaches implemented for penalisation, replacement and scaling. The GA parameter values used were the same as the ones for HIPS optimisation problem which had been chosen following guidelines provided in literature. However, on the basis of the performance results obtained when solving the HIPS optimisation problem (*Chapter 4*), the size equal to 20 chromosomes and the mutation rate equal to 0.002 have been removed from the analysis. Thus the twenty seven different combinations of values of the GA parameters have been employed in total. The values of the parameters are listed in *Table 6.5*. As in the previous application example due to the stochastic nature of the GA ten runs have been performed for one set of parameters and the average of the best feasible fitness values, i.e. minimal feasible values, was calculated per each generation. Each time the process was terminated after 100 generations. All obtained analysis results can be found in *Appendix 2*, while this section provides their summary.

Table 6.5. The List of Values of GA Parameters

<i>Value No.</i>	<i>1st Value</i>	<i>2nd Value</i>	<i>3rd Value</i>
<i>Parameter</i>			
Population Size	10	30	50
Mutation Rate	0.001	0.005	0.01
Crossover Rate	0.75	0.8	0.95

First the influence of the population size is considered. The average of the minimal feasible fitness values obtained for all combinations of crossover and mutation rate values is evaluated for each population size. The results are presented in *Table 6.6*. As expected, the larger population size introduces more diversity in the population and therefore improves the capability of the algorithm to find a near-optimal solution. If the results are compared with the ones of the HIPS problem the difference between average minimal unavailability values is much smaller for the FWDS problem. It suggests that either the search space for the problem is rather small or the improved algorithm is less susceptible to changes of the population size.

Table 6.6. Minimal Average System Unavailability Values for Different Population Sizes

	Population Size		
	<i>10</i>	<i>30</i>	<i>50</i>
Mean of Minimal System Unavailability Values	0.1001	0.0954	0.0940

The averages of minimal feasible fitness values obtained for each combination of crossover and mutation rates over 10 runs when the population size has been fixed to 10, 30 and 50 chromosomes are given in *Tables A.2.10-A.2.12*. The results suggest that the rate at which new chromosomes are introduced is less influential on the optimisation performance than the retention of diversity in populations.

Table 6.7 shows that the average of feasible minimal fitness values evaluated for all combinations of different crossover rate and population size decreases if the mutation rate is increased. Analysing the influence of the mutation rate (*Tables A.2.13-A.2.15*) it has been noticed that for the small mutation rate the algorithm performs better if an average population size is used. Apparently, combinations of small mutation rate and large populations promote the production of sub-optimal solutions. The increasing population size and the mutation rate encourage the convergence of the algorithm to an optimal solution. However the same rule does not apply to the crossover rate. If the mutation rate is lower and equal to 0.05 a trade-off between the mutation rate and crossover rate needs to be found. Overall the best feasible fitness values for the same combinations of crossover rate and populations size decrease with

increasing mutation rate. It shows the importance of the diversity in the population. Nevertheless its level needs to be chosen according to the number of chromosomes in the populations.

Table 6.7. Minimal Average System Unavailability Values for Different Mutation Rates

	Mutation Rate		
	<i>0.001</i>	<i>0.005</i>	<i>0.01</i>
Mean of Minimal System Unavailability Values	0.0990	0.0958	0.0947

The average of the best feasible fitness values for all combinations of mutation rates and population sizes was also evaluated for each value of crossover rate. The results are presented in *Table 6.8*. It shows that a high crossover rate indicating a quick introduction of new strings into the population helps to improve the performance of the optimisation process.

Table 6.8. Minimal Average System Unavailability Values for Different Crossover Rates

	Crossover Rate		
	<i>0.75</i>	<i>0.8</i>	<i>0.95</i>
Mean of Minimal System Unavailability Values	0.1003	0.0947	0.0945

Results of the analysis performed to identify how combinations of different mutation rates and populations sizes influence the FWDS optimisation process for each crossover rate value are presented in *Tables A.2.16-2.18*. Here there is a tendency for the fitness value to decrease for increasing values of both analysed parameters, i.e. mutation rate and population size. Even though this tendency appears for every crossover rate, the increasing crossover rate does not lead to smaller best fitness values. These results justify the earlier made conclusion that the choice of values for the mutation rate and population size has a stronger influence on the optimisation process than the crossover rate value for solving the FWDS problem.

Studying the performance results of algorithm after 100 generations, a relationship can be identified between the average fitness value of the population and the best feasible solution found. The results obtained using populations of 50 chromosomes are shown in *Figure 6.13*. Here the best feasible fitness value is larger then the average population fitness value for mutation rates equal to 0.001 and 0.005. When the mutation rate is increased up to 0.01 the best feasible fitness value obtained is smaller than the average fitness value. This relationship applies to all values of crossover rate.



Figure 6.13. Differences Between Average and the Best Feasible Fitness Values

The optimisation results show that the GA is capable of finding good feasible solutions, but converges to an infeasible solution if the mutation rate is lower. If the mutation rate equal to 0.01 is used the algorithm converges to a feasible solution and the quality of the solution improves. It reinforces the hypothesis that increasing diversity increases the possibility of obtaining good solutions among the populations.

The algorithm demonstrates the ability to solve the FWDS design optimisation problem and to find a near-optimal solution. The analysis performed when using different values of GA parameters and the influence of the parameters on the optimisation process suggests that the algorithm performs better if a larger population size and higher mutation rate are used. Results obtained for the HIPS (discussed in *Section 5.5.1*) also indicate that by using a larger population size and a higher mutation rate the performance of the algorithm improves.

Thus to perform further testing on FWDS optimisation problem the mutation rate equal to 0.01 and population comprising of 50 chromosomes were chosen. Based on the results in *Table 6.8* the crossover rate equal to 0.95 was included in the set of chosen GA parameters. The analysis performed is discussed in the following section.

6.6.2. Testing

The set of GA parameters (i.e. crossover rate, mutation rate and population size equal to 0.95, 0.01 and 50 respectively) identified in the preceding subsection are considered further here applying them and by performing ten more runs. The number of generation-iterations in a single run of the programme has been left the same, i.e. 100 generations. At the second stage of the analysis the number of generation-iterations performed has been increased up to 200. Five more runs have been performed to check the level of the improvement in terms of

algorithm convergence. Average fitness value of the population and the best feasible fitness value yet found are two quantities evaluated for each generation and represented graphically.

Figure 6.14 presents the average population objective function values for each generation from the four chosen runs. The average objective function value rapidly decreases during the first 30 generations. In later generations both the convergence rate and the fluctuation in average fitness values decreases significantly. The results demonstrate steady convergence of average population fitness and dominance of highly-fit chromosomes in populations with later generations.

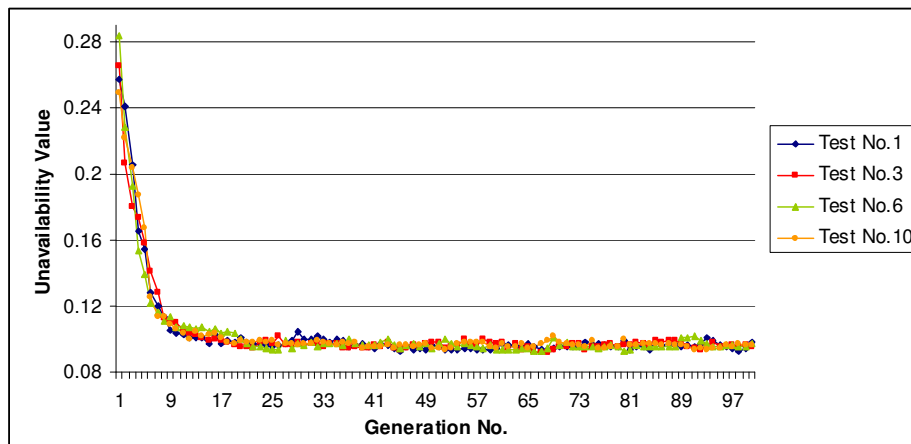


Figure 6.14. Average Unavailability Value in Each Generation. Total 100 generations

Figure 6.15 presents the best feasible solutions found over 100 generations of four chosen simulation runs. It confirms the capability of the algorithm to find good feasible solutions. The standard deviation of the best solutions found after 100 generations is equal to 0.00034. It suggests that the algorithm exhibits low sensitivity to the random number seed and therefore is robust.

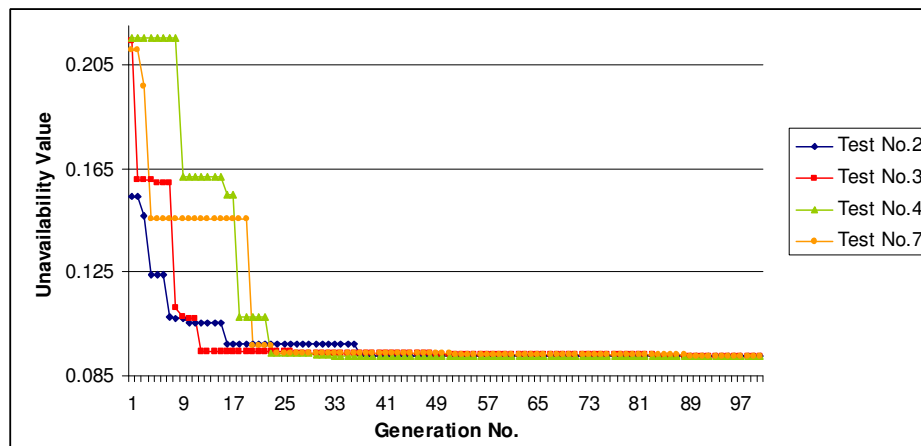


Figure 6.15. Best Feasible Fitness Value per Each Generation

Results obtained from 200 generation runs showed improvement in the best fitness values. For instance, the mean of the best solutions after 100 generations was equal to 0.09278 and after 200 generations it was equal to 0.09261. The standard deviation of the best solutions found also decreased to 0.00031. It suggests that increasing the number of generations encourages further exploration of the boundary region between feasibility and unfeasibility, leading to better solutions. However one should take into account that increasing the number of generations also increase computational burden. Therefore it needs to be assessed if the level of improvement in the optimal solutions is worth the computation resources required in order to find it.

The best system design arose during one of the runs in generation 161. The best system design includes 2 firewater diesel-driven and electrically-powered pumps. All four pumps have 33 1/3% capacity and are identified as type 2 pumps. It also includes 2 AFFF diesel-driven and 2 AFFF electrically-powered 100 % capacity pumps. The optimal design has 2 pressure transmitters of type 3 where at least 1 pressure transmitter is required to function. A new inductor nozzle and new valmatic relief valve together with an old deluge nozzle are in the found optimal design. Finally, both AFFF deluge and water deluge valves of type 3 are included in the new FWDS design. The maintenance test interval for the firewater and AFFF pump was assigned to 10 days. Time intervals between maintenance of the ringmain and the Deluge skid were equal to 12 days (~2 weeks) and 90 days (~3 months) correspondingly. The newly designed FWDS with the new maintenance scheduling implemented has an unavailability of 0.092348. Its cost is equal to 80873 units and maintenance downtime is 29.6 days a year.

Table 6.9 lists the parameters of the best designs achieved throughout the optimisation analysis. Results show that the optimal number of firewater and AFFF pumps is equal to two in all best design cases. 33 1/3% capacity firewater pumps appear more often than 50% capacity ones and 100% capacity firewater pumps have not been included in any optimal design solution found. However, considering the capacity of AFFF pumps, 100% capacity dominates in the best designs generated. According to the results, there is no need to increase the number of pressure transmitters on the ringmain if it is replaced with a pressure transmitter of type 3 in order to reduce the system failure associated with the activation of the automatic FWDS mechanism. The new inductor nozzle dominates in the optimal FWDS designs. In all listed cases the new valmatic relief valve and the original deluge nozzle are introduced. Both AFFF deluge and water deluge valves of type 3 are included in the new FWDS designs. The optimal maintenance test interval for the firewater and AFFF pump sets varies from 7 days to up to 21. Ringmain maintenance can be performed at 12-day intervals.

Finally for all optimal designs found the optimal time interval between maintenance of the deluge skid is 90 days (3 months).

Table 6.9. Best Generated FWDS Designs

<i>Associated System Component</i>	<i>Design Variable Notation</i>	<i>Design 1</i>	<i>Design 2</i>	<i>Design 3</i>	<i>Design 4</i>
Firewater diesel driven pump	n_{FD}	2	2	2	2
	$t1_{FD}$	3	3	2	3
	$t2_{FD}$	2	2	1	1
Firewater electrically powered pump	n_{FE}	2	2	2	2
	$t1_{FE}$	3	3	3	2
	$t2_{FE}$	2	1	2	1
AFFF diesel driven pump	n_{AD}	2	2	2	2
	t_{AD}	1	1	2	2
AFFF electrically powered pump	n_{AE}	2	2	2	2
	t_{AE}	1	1	1	1
Pressure transmitter on the ringmain	n_{PT}	2	1	1	1
	k_{PT}	1	1	1	1
	t_{PT}	3	3	3	2
Inductor nozzle	t_{IN}	2	2	2	1
AFFF deluge valve	t_{AD}	3	3	3	3
Deluge nozzle	t_{DN}	1	1	1	2
Water deluge valve	t_{WD}	3	3	3	3
Valmatic relief valve	t_{VR}	2	2	2	2
Maintenance test intervals	θ_P	10	7	16	21
	θ_R	12	12	12	13
	θ_D	90	90	90	90
Characteristics	$Cost$	80873	80373	80873	80873
	MDT	29.6	29.7	29	26.8
	Q	0.092348	0.092349	0.092353	0.092478

6.7. SUMMARY

The successful application of the GSDOA to solve the FWDS optimisation problem proves that the algorithm:

- has potential to be applicable to different safety systems;
- is efficient in finding good near optimal solutions;
- is indeed scalable and can deal with large scale complicated problems.

In this chapter the application of the GSDOA and the implementation of GSDOP to solve the FWDS design optimisation problem have been demonstrated. In general, to be able to solve a design optimisation problem using the algorithm it is imperative to form the list of FTMPs corresponding to the structural design variables of the optimisation problem in question. The list of FTMPs for the FWDS and their order in the formed list has been discussed and

reasoned. The problem-specific chromosome structure encoding the design variables and therefore the FTMPs used has also been detailed in this chapter. Thus the FWDS example supports the potential applicability of the algorithm for different safety systems.

The current FWDS design optimisation problem has been solved using the improved GSDOA where new penalisation, replacement and scaling procedures were implemented in the GA. The modifications demonstrate that the optimisation process has been improved. The modified approach has an enhanced ability to converge on the fittest design. After a number of generations the fluctuation of the average fitness values decreases, showing that the domination of the designs similar to the best feasible overall design case increases in the populations. Results from the optimisation analysis performed for FWDS and previously for HIPS also suggest that the algorithm exhibits some sensitivity to the GA parameter variations. Therefore, it is suggested to use larger sizes of chromosome populations and a higher mutation rate. The crossover rate (if its value is reasonably high) has less influence on the optimisation process.

Solving the more complicated optimisation problem has introduced an increase in computational intensity. For comparison, it takes 558 seconds to run a simulation of one hundred generations with a 50 chromosome population for the FWDS. A simulation with the same parameters takes 220 seconds to run for the HIPS. The computations were carried out on a Dual Core 1.60 GHz processor with 1.00GB of RAM on a 32-bit operating system. Therefore in order to extend the capability of the algorithm to solve a wider range of problems the future work should also focus on improving the performance of the algorithm and minimisation of CPU time. It is noteworthy that this can be a key requirement for real-time optimisation of engineering systems.

The application of the GSDOP to solve the FWDS problem has demonstrated that the algorithm produces good feasible solutions for the large scale and more complicated problem. Despite the fact that it could not guarantee to find the global optimal solution the algorithm provided near-optimal solutions using minimal computational resources. To find the global optimal solution for such problem using the exhaustive search method significantly more computational resources and time would be required. There are 1,911,029,760 combinations of the design variables and therefore the design cases to be analysed. Given that it takes 0.1 seconds to generate a specific system design case and evaluate its failure probability (based on the results presented for computation time discussed in the paragraph above) it would take

approximately 53084 hours or 6 years to find the global solution using the exhaustive search approach.

The further development of the generalised optimisation methodology focuses on solving the design optimisation problems of multi-phased mission systems.

7. MULTI-PHASED MISSION SYSTEM DESIGN OPTIMISATION

7.1. INTRODUCTION

A phased mission system represents a system where its performance objectives are divided into consecutive, non-overlapping phases. During each phase the system performs a certain task or a number of tasks that have to be completed at the same time. The mission is considered to be completed if every task in each phase is completed successfully. Failure to complete any phase successfully causes the whole mission failure.

A large number of systems which can employ different technologies such as mechanical, electronic, nuclear and chemical devices appear in industry and can be considered as phased mission systems. The relevance and significance of optimising phased mission system performance and the appropriate use of limited resources is therefore evident. In spite of its importance, however, there is limited demonstrated evidence in the literature for research that focuses on such phased mission optimisation problems. Susova & Petrov [81] proposed a model for aircraft maintenance system optimisation. The model is based on a Markov homogeneous process and is employed to ensure aircraft safety and minimise operation costs.

The research in this project has been focused on the development of an approach to construct an optimal phased mission system design with the aim of minimising its failure within the context of pre-defined design constraints and resources. The developed approach is based on GSDOA which is used for the optimisation analysis of safety system designs (*Chapters 4 and 6*). Since any general system - including a safety system - can be considered as a system performing a mission consisting of a single phase, GSDOA was altered so that a multi-phased mission system analysis can be performed.

In this chapter an overview of methodologies used for reliability analysis of phased mission system is given (*Section 7.2*). The developed approach and programme code are detailed in *Sections 7.3 and 7.6* respectively. An unmanned aerial vehicle (UAV) and military vessel have been selected to demonstrate the methods application. The application examples are discussed in *Sections 7.4, 7.5 and 7.7*.

7.2. METHODS FOR RELIABILITY ANALYSIS OF PHASED MISSION SYSTEMS

7.2.1. Overview

When analysing multi-phased mission systems it is necessary to acknowledge that a number of system components may stay inactive during some phases, and that they can be employed to complete a certain task later in the mission. For example, a braking system of a plane is not in use whilst the plane is in the take-off or cruise phase, but it is activated when the landing phase starts. The configuration of a system can also vary throughout each phase in the mission. Therefore a situation may occur when a component fails at some point during the mission and its failure is revealed later or its condition becomes critical just for one particular phase. As such, the performance of the system in a certain phase is influenced by results of its performance in previous phases.

The dependencies between phases as well as dependencies between failure behaviours for the same components across different phases make reliability analysis of phased mission systems complicated. The approaches used to evaluate the reliability of phased mission systems can be classified into three major groups. The first group of methods, which are called combinatorial methods, are based on FTA, block diagrams and BDD analysis. These methods are usually applied to non-repairable systems, since neither fault trees nor block diagrams can easily reflect the repair processes. However, Vaurio [82] suggested a fault tree based approach to analyse phased mission systems with repairable components.

The second group of methods use Markov chain based approaches. These methods can be adapted to analyse more complicated system behaviour compared with combinatorial methods. Markov chain methods can be employed to analyse systems with random phase durations and/or repairable components. For example, Alam and Ubaid [83] suggested the Markov approach for quantitative reliability evaluation of systems with both deterministic and stochastic mission-phase change times. Systems, where the failure rates of components change from phase to phase, can also be analysed using this approach. A non-homogeneous Markov model was considered by Smotherman and Zemoudeh in [84]. The method is applicable for systems where mission-phase change times can be state dependent, mission phases are of random duration and repair and failure rates are globally time dependant. However, since the number of system states grows exponentially with the number of components in the system and that initial conditions are needed to be defined for every phase, it restricts the applicability of Markov methods for larger systems.

Simulation methods can be used for very complex systems when neither combinatorial nor Markov methods can be employed. Despite the fact that these methods have least restrictions and both repairable and non-repairable systems can be analysed with different dependencies between components, they are the most expensive approaches in terms of computation.

7.2.2. Non-Repairable Phased Missions

This part of the thesis is dedicated to the design optimisation of non-repairable phased mission systems. Reliability analysis of non-repairable systems can be performed using combinatorial, Markov chain or even simulation approaches. However in order to avoid state space explosion problems arising in the Markov methods, fault tree and BDD based approaches are usually employed. This section reviews combinatorial methods developed for the reliability analysis of phased mission systems.

The fundamental idea of how to employ fault tree analysis for phased missions was introduced by Esary and Ziehms in [85]. Their approach is to combine fault trees for failure causes of each single phase into one system fault tree and to analyse the system as a single-phase mission. Their method provides an exact unreliability value for a phased mission system where all components are non-repairable and have s -independent failure characteristics.

The method suggested by Esary and Ziehms comprises of four major steps. At first each phase fault tree undergoes a simplification stage. The simplification is actually the cancellation of a number of minimal cut-sets. If the phase i list of minimal cut-sets contains a minimal cut-set that appears in any minimal cut-set from later phases then the minimal cut-set is eliminated from the phase i list. For example, phase 1 has three minimal cut-sets: {A,B}, {A,C} and {A,D}. Phase 2 has the following minimal cut-sets: {A,B} and {E,F}. The cut set {A, B} can be cancelled from phase 1 since it also appears in phase 2 and it means that if the system fails in phase 1 it will still be in the same state in phase 2. As a result just the two remaining cut-sets in phase 1 are used for further analysis.

The second step of the method involves basic event transformations. At this stage every basic event in phase i is replaced by a series of basic events where each of them represents the corresponding component failure in every phase up to and including phase i . For example, basic event A in phase 3 would be replaced by components $A_1 + A_2 + A_3$. It means that if component A is failed in phase 3, it could have failed during phase 1 or phase 2 or phase 3,

since the component is non-repairable. Using fault tree analysis, basic event A would be replaced by an “OR” gate with corresponding input events A_1 , A_2 and A_3 .

At the third step the individual transformed phases are joined in a series to form a single system. The modified fault tree for each phase becomes an input event of the “OR” gate that leads to the fault tree top event “Mission Failure”. A multi-phase mission system can then be analysed as a single-phase mission system. Minimal cut sets can be identified from the new logic model and the usual quantitative evaluation technique can then be used to obtain the system unreliability value.

Somani and Trivedi introduced a new technique for phased mission system analysis based on Boolean algebraic methods [86]. Specifically, the logic expressions are employed to represent phase failure combinations (PFC) for each phase. Phase failure combinations for any phase i consists of those combinations which represent failure modes in phase i (E_i) but are not failure combinations in any of the subsequent phases ($E_{i+1}, E_{i+2}, \dots, E_n$):

$$PFC_i = E_i \cap \overline{(E_{i+1} \cup \dots \cup E_n)} \quad (7.1)$$

In the approach they also introduced the notation to denote failure of a component in each phase equivalent to the one used by Esary and Ziehms in [85]. According to the notation event A_i indicates failure of component A during an interval from the start of the mission to the end of the analysed phase i . Using the new notation a Boolean expression specifying failure combinations for subsystems presented in *Figure 7.1* and used, for example, for system phase i , can be expressed as follows:

$$\begin{aligned} E_{iX} &= A_i + B_i + C_i, \\ E_{iY} &= A_i + B_i C_i, \\ E_{iZ} &= A_i B_i C_i. \end{aligned} \quad (7.2)$$

Algebraic rules were introduced to simplify the logic expressions for $PFCs$, where i and j are two phases and $i < j$:

$$\begin{aligned} A_i A_j &\rightarrow A_j & \overline{A_i} + \overline{A_j} &\rightarrow \overline{A_j} \\ \overline{A_i} \overline{A_j} &\rightarrow \overline{A_i} & A_i + A_j &\rightarrow A_i \\ \overline{A_i} A_j &\rightarrow 0 & A_i + \overline{A_j} &\rightarrow 1 \end{aligned} \quad (7.3)$$

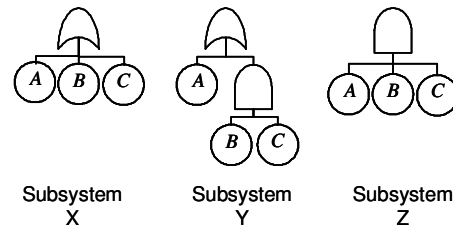


Figure 7.1. Fault trees of phases X, Y and Z

Thus employing the earlier introduced Boolean rules system unreliability can be found using the following equation:

$$Q_{SYS} = P(E_n) + \sum_{i=1}^{n-1} P(PFC_i) \quad (7.4)$$

where $P(E_n)$ is the probability of failure of the last phase n and $P(PFC_i)$ is the probability of phase failure combinations for phase i . The probability of phase failure combinations are computed for each phase i ($i = 1, 2, \dots, n-1$). The logical expressions are derived using the phase fault trees and their probabilities are found using a failure distribution function for each component.

Ma and Trivedi in [87] introduced a method to analyse and solve phased mission systems by combining a phase algebra approach with a cancellation methodology. This method is based on the approach represented by Somani and Trivedi in [86]. The new improvements made the method more computationally efficient compared the original approach. The authors introduced the minimal cut-sets cancellation rule, which was originally implemented by Esary and Ziehms. The second improvement of the algorithm refers to the introduced formula of the sum of disjoint phase products, which is a phased-extension of the formula for the sum of disjoint products. Thus *Equation 7.4* for the unreliability of a phased mission system can be rewritten as:

$$Q_{SYS} = \sum_{i=1}^n P(DPC_i) \quad (7.5)$$

where n denotes the total number of phases in the mission and DPC_i is a disjoint phase constituent for phase i . If PE_i represents a set of generally non-disjoint minimal cut-sets then the phase constituent can be defined as $PC_i = \overline{PE_1} \overline{PE_2} \dots \overline{PE_{i-1}} PE_i$, where $1 < i \leq n$. A PC_i is defined as DPC_i if the phase products in PC_i are mutually disjoint.

La Band and Andrews in [88] proposed a new method for phased mission system analysis which was also based on the fault tree approach. By employing the new method, the unreliability of each phase can be determined in addition to the whole mission unreliability. Actually, the whole mission unreliability is the sum of unreliability values of all phases in the mission:

$$Q_{MISS} = \sum_{i=1}^n Q_i, \quad (7.6)$$

where Q_i denotes unreliability of phase i and the total number of phases in the mission is equal to n .

In the proposed methodology for *Equation 7.6* to be valid, fault trees for each phase need to be modified following certain rules. The first rule is equivalent to the basic event transformation rule introduced by Esary and Ziehms in [85]. According to that rule every basic event of a fault tree needs to be represented as an “OR” gate with i basic events for any phase i . Each basic event represents the component failure event in every phase up to and including phase i . As an example, the failure of component A in phase 2 is presented in *Figure 7.2*. Here, A_1 is the failure of component A in phase 1, and A_2 represents the failure of the component in phase 2.

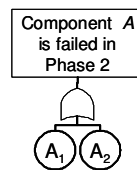


Figure 7.2. Representation of event A as an “OR” gate

The second rule specifies combinations of the causes of success of previous phases with the causes of failure for the phase being analysed. In other words, if a system fails in phase i , it means it could not have failed during any previous phase j ($j = 1, 2, \dots, i-1$). Therefore, system failure in phase i is represented by an “AND” gate that incorporates the success of previous phases j (using “NOT” logic) and the failure for phase i . *Figure 7.3* represents the failure of a system in phase 2. Phase 2 failure is shown as a combination of success in phase 1 and failure in phase 2. In phase 2 components A , B and C are replaced using “OR” gates to indicate that the components could have failed during phase 1 or phase 2, as shown previously

in Figure 7.2. The overall mission unreliability is then equal to the sum of unreliability for all phases (as stated in Equation 7.6).

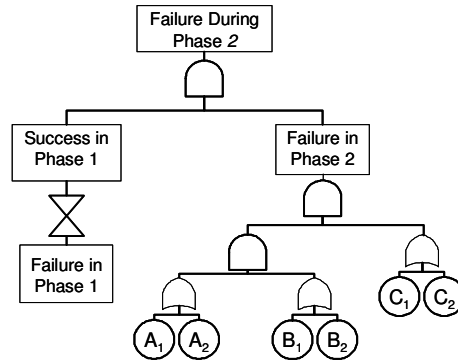


Figure 7.3. An example of system failure during Phase 2.

The representation of the mission fault trees necessitate a different approach for quantification analysis of the mission. In the general case, minimal cut-sets could be determined from a phase fault tree and the inclusion-exclusion formula would be applied in order to find each phase reliability. Since any phase after the first phase incorporates causes of success of previous phases and they are represented using NOT logic, the phase fault tree will be non-coherent. Therefore in this case prime implicants identify the sets of basic events that cause system failure in any phase, except the first one to occur. The authors introduced a fault tree modularization technique in order to enable prime implicants to be found more efficiently. When applying the inclusion-exclusion formula for the established prime implicants the authors introduced a new algebra over the phase to manipulate the derived logical expressions. The summarised algebraic laws are presented (Equation 7.7). The following notation is used to represent the presented laws. A_i denotes the failure of component A in phase i and $\overline{A_i}$ represents the functioning of component A throughout phase i . Phase i appears some time before phase j , i.e. $i < j$. Therefore the new notation A_{ij} was introduced that indicates the failure of a component A at some time from the start of phase i to the end of phase j .

$$\begin{array}{ll}
 A_i \cdot A_i = A_i & \overline{A_i} \cdot A_{ij} = A_{i+1,j} \\
 A_i \cdot A_j = 0 & A_i + A_{i+1} + \dots + A_j = A_{ij} \\
 A_i \cdot A_{ij} = A_i & \overline{A_i A_{i+1} \dots A_j} = \overline{A_{ij}} \\
 \overline{A_i} \cdot A_i = 0 & A_i B_i \rightarrow A_i B_{ij} \\
 & A_i B_j
 \end{array}
 \tag{7.7}$$

In the algorithm [88] a fault tree conversion to its corresponding BDD was implemented after the required alterations to the mission or each phase fault tree were made, i.e. just before the quantification process is started. It improved the efficiency of the analysis but on the other hand, problems associated with constructing large BDDs appeared. For example, it requires a global variable ordering scheme to construct these BDDs. Prescott *et. al.* [89] presented a novel BDD based approach for phased mission analysis to overcome the global variable ordering scheme requirement. In the proposed approach at the first step the fault trees for mission phases are converted to BDDs. Since each BDD is converted by employing its own variable ordering scheme, the size of the BDD is minimised.

The second step of the algorithm concerns to the assignment of the time intervals over which each of the system component contributes to the phase failure for each constructed BDD. Each component is assigned an indicator variable:

$$x_k(t_i, t_j) = \begin{cases} 1, & \text{if component } k \text{ fails from time } t_i \text{ to time } t_j, \\ 0, & \text{otherwise.} \end{cases} \quad (7.8)$$

Here k identifies a component, t_i is a start time of the interval and t_j is the end of the time interval. Since failure of a component can contribute to the failure of the phase at any time from the start of the mission to the end of the phase, t_i denotes the start of the mission and t_j marks the end of the phase.

If considering the success state of a component, i.e. when component k does not fail from the start of the mission until a certain time t_i , it can also be expressed using an indicator variable:

$$\overline{x_k}(t_0, t_i) = x_k(t_i, \infty) \quad (7.9)$$

From *Equation 7.9* it follows that if component k has not failed by time t_i then it must fail some time later.

The authors use the same approach as described in [88] to represent mission failure during any phase. The mission fails in phase i , if it did not fail in any of the previous $i - 1$ phases and failure occurs in the phase i . If \mathbf{F}_j represents the logical expression for the top event to occur in phase j and \mathbf{Ph}_j denotes mission failure in phase j then the following equations are valid:

$$\begin{aligned}
\mathbf{Ph}_1 &= \mathbf{F}_1, \\
\mathbf{Ph}_2 &= \overline{\mathbf{F}_1} \mathbf{F}_2, \\
&\dots\dots\dots \\
\mathbf{Ph}_j &= \overline{\mathbf{F}_1} \overline{\mathbf{F}_2} \dots \overline{\mathbf{F}_{j-1}} \mathbf{F}_j
\end{aligned} \tag{7.10}$$

Using this approach *Equation 7.6* is valid which can be rewritten using the new notation:

$$Q_{MISS} = \sum_{i=1}^n P(\mathbf{Ph}_i), \tag{7.11}$$

where n is the total number of phases in the mission.

The next algorithm step involves building the logical expressions for mission failure in each phase following the rule identified with *Equations 7.10* and using the appropriate BDDs. When connecting a number of BDDs to construct a logical expression for mission failure in phase i , some BDDs might contain identical variables. However these variables are treated as being independent since time intervals were associated with each of them in order to take into account the existing dependencies among them.

The quantification process involves the analysis of BDDs representing mission failure in each phase, i.e. BDDs constructed for each Ph_i ($i=1, 2, \dots, n$). In order to find the logical expression of a possible outcome represented by the BDD, every possible path from a BDD root node to a terminal 1 node has to be identified. In the proposed approach a simplification process of each path takes place at the same time that the path is traversed. It involves manipulating the time intervals associated with those variables that occur more than once along the path. The following rules apply:

$$\begin{aligned}
x_k(t_i, t_j) &= 0, \text{ if } t_i > t_j \\
x_k(t_{i1}, t_{j1}) x_k(t_{i2}, t_{j2}) &= x_k(\max(t_{i1}, t_{i2}), \min(t_{j1}, t_{j2}))
\end{aligned} \tag{7.12}$$

When the path logic is simplified, the probability values for each path variable are calculated and then multiplied together to give the path probability. The probability of each variable needs to be determined according to *Equation 7.13* where t_0 denotes the starting time of the mission and Q_k is the cumulative failure distribution function for component k .

$$P[x_k(t_i, t_j) = 1] = \begin{cases} Q_k(t_j - t_0) - Q_k(t_i - t_0), & \text{if } t_j \neq \infty \\ 1 - Q_k(t_i - t_0), & \text{if } t_j = \infty \end{cases} \tag{7.13}$$

Finally, the probabilities of all paths starting from the corresponding BDD root node and terminating at terminal 1 nodes are summed which gives each phase a failure probability value, i.e. $P(\mathbf{Ph}_i)$. The overall mission failure probability is found using Equation 7.11.

To summarise, the method introduced by Prescott *et. al.* converts each phase fault tree to a BDD at the beginning of the analysis. Later all dependencies between components in different phases are incorporated in each BDD. It means that a global ordering scheme of variables is not required when constructing BDDs. The potential to minimise the size of the BDDs to be quantified also exists. Overall, it makes the algorithm computationally more efficient than the previously mentioned algorithms. Owing to these properties this algorithm was chosen to perform the quantification analysis in the phased mission design optimisation algorithm. The algorithm is discussed in the following section (Section 7.3).

7.3. PHASED MISSION SYSTEM DESIGN OPTIMISATION ALGORITHM (PMSDOA)

7.3.1. Introduction

The phased mission system design optimisation algorithm (PMSDOA) has been developed to solve multi-phased mission system design optimisation problems. On the basis that the GSDOA is applicable to single phased mission systems it has been used as a foundation to construct the new algorithm. The main techniques and performance principles implemented in the GSDOA have been adopted in the PMSDOA. Amendments introduced in the new approach were required due to specific characteristics and principles of a phased-mission system operation.

The PMSDOA has the same conceptual structure as that of the GSDOA. It contains two main parts as shown in Figure 7.4. In the first part all the possible system designs are introduced. The fault tree modification patterns chosen according to a given list of structural design variables are implemented in the fault tree for each phase. The second part comprises the quantitative system analysis and the optimisation technique. The quantitative system analysis is implemented to evaluate the mission failure probability for different design cases. The resulting fault trees from the first part of the algorithm are utilised. Different system designs are generated using the initially applied optimisation technique – the single objective GA (SOGA).

Some changes have been introduced in both parts of the algorithm. First of all considering a multi-phased mission system a number of fault trees need to be analysed at the same time. Thus, the methodology used to represent all possible design alternatives needs to be modified. Quantification of system failure is performed employing the new methodology designed for multi-phased mission systems.

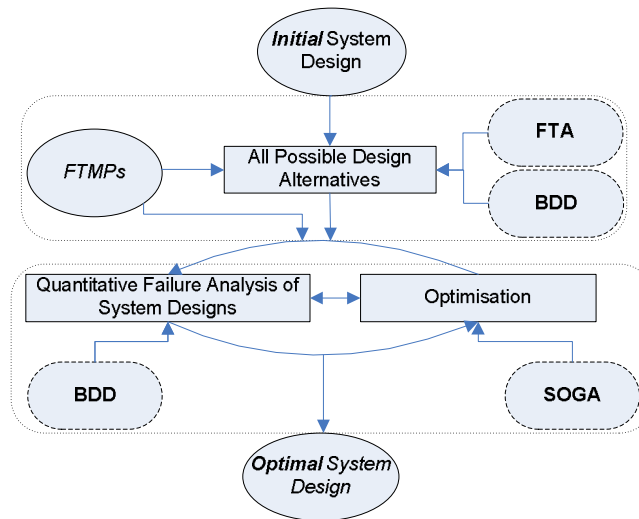


Figure 7.4. Structure of Phased Mission System Design Optimisation Algorithm

With the same performance principles as the GSDOA the new algorithm has the potential to solve optimisation problems of different phased mission systems, i.e. it is not restricted in applicability to a particular system. However, it is designed for non-repairable phased mission systems. A number of assumptions are considered. System components are assumed to be working at the mission start. If a component failure occurs it therefore remains present in the system. It is also assumed that the length of each phase is determined and known.

The PMSDOA has been built gradually extending its capability to solve a wider range of problems. At the first stage a methodology for a simple optimisation problem was developed. The objective of the analysed problem was to minimise the overall mission failure probability. Next, constraints for system design regarding the overall mission were introduced. Finally, the approach was extended by introducing constraints for system design characteristics at different phases of the mission. In this section algorithm development is discussed in detail emphasising the alterations and improvements made to the algorithm. Application examples are also presented.

7.3.2. Fault Trees for all Possible Design Alternatives

One of the characteristics of the GSDOA is the utilisation of system fault trees to represent the initial system design as well as all possible design alternatives introduced. The adaptation of the methodology for multi-phased mission systems is a complex process; amendments to the methodology have been introduced due to the increased number of fault trees being analysed at the same time.

It is known that a given set of structural design variables with their parameter values defines the possible system design alternatives. Even though a phased mission system performs different tasks throughout a mission and has an individual fault tree for each phase its design is unchangeable from the beginning to the end of the mission. Therefore the given set of structural design variables applies to each phase of the mission and the same design alternatives should be represented in all of its fault trees. However it is also known that a number of system parts may be inactive during certain phases which results in the use of different fault trees for different phases. This means that in some cases particular design alternatives cannot be represented in every phase fault tree. As such, individual subsets of structural design variables need to be assigned for each phase. Each subset contains only those variables which correspond to failure events stated in the phase fault tree.

In the programme the subsets of FTMPs representing the corresponding structural design variables are formed when constructing the fault trees, incorporating the failure causes of all possible system design alternatives as follows. One phase fault tree is considered at a time starting with the first phase. The initial design fault tree is traversed to locate any events corresponding to design variables from the given list. When such an event is found the required alterations are made, i.e. the event is replaced by an associated sub-tree structure defined using a certain FTMP. After the fault tree is checked for all events from the given list and the required modifications are made, the fault tree for the second phase is traversed and searched for the events identified to be replaced. The task is completed when the required alterations are made for the last phase fault tree.

As a reminder, a sub-tree of each FTMP includes a group of house events linked with each other. These links identify which house events are set to TRUE and which are set to FALSE. The rules defining the links were introduced when discussing GSDOA. When analysing phased mission systems one phase fault tree is altered at a time and the FTMPs which are only relevant to this fault tree are used. Thus, house events appearing in the fault tree are not linked to house events in the rest of fault trees. For example, house events of the first phase

fault tree are linked with house events used only in this fault tree and do not have any links with house events in fault trees of the second or any other phase. This means that there is no need to introduce additional rules defining the links of house events between different fault trees. Moreover the existing house event rules can be applied by analysing each fault tree of the mission individually. However, a group of house events corresponding to the same design variable can appear in a number of fault trees. In this case values of house events within the group need to be the same in each phase fault tree in order to maintain the consistent system design throughout each phase. For this purpose a data record is kept for each implemented FTMP. It includes the list of fault trees where it has been implemented and a set of house event values once they are specified. Using such records house events within the groups of house events representing the same design alternative are assigned the same values throughout the mission fault trees.

House events belonging to a particular phase fault tree are identified according to the house event numbering rules. House events in a fault tree for phase i are numbered from $h_{i-1} + 1$ to $h_{i-1} + h_i$. Here h_{i-1} is the total number of house events in phase $i - 1$ and h_i is the total number of house events in a fault tree for phase i . For the first phase h_{i-1} is equal to 0, thus house events would be numbered $1, \dots, h_1$. House events in the second phase would have numbers $h_1 + 1, h_1 + 2, \dots, h_1 + h_2$. House events in the rest of the fault trees would be numbered accordingly. Being able to differentiate house events for each phase fault tree enables groups of house events that are linked together to be identified easily. The numbering rules for house events within a linked group remain the same as the ones in GSDOA.

7.3.3. Evaluation of Phased Mission System Failure Probability

It is known that FTA is not a computationally efficient methodology to quantify system failure. In the GSDOA the BDD methodology has already been used instead. Therefore the BDD based approach introduced by Prescott *et. al.* [89] and discussed in Section 7.2 has been chosen to evaluate the mission failure probabilities.

The employed BDD method can only be applicable once a system design is specified, i.e. once a fault tree representing the specific design is constructed. In the single phase algorithm a trimming operation was introduced before the fault tree is converted into its BDD. Although fault tree trimming and conversion slowed down the optimisation process, it allowed the size of the tree being analysed to be greatly reduced. The general system optimisation problems solved using this methodology were relatively small. When solving a phased mission system problem the number of fault trees analysed increases considerably and the old methodology

becomes computationally inefficient. Therefore an improved approach has been introduced. In the new approach the conversion of mission fault trees to their corresponding BDDs is completed only once, after fault trees that account for all possible design alternatives are constructed. The resulting BDDs are then utilised for specification of system design cases.

The implementation of the new approach is based on properties of both house events and BDDs. Using BDD analysis the probability of occurrence of the top event of the corresponding fault tree is equal to the probability of the sum of the disjoint paths from the root node to each terminal 1 vertex. The probability of any disjoint path is obtained by multiplying probabilities for the variables along the path. Therefore if a house event set to FALSE is assigned the failure probability value equal to 0 each path containing such a house event will have a failure probability value equal to 0. Accordingly a house event set to TRUE needs to be assigned a value equal to 1. As a result BDD paths with house events set to FALSE states will be eliminated and the remaining paths will represent a specific system design case.

By employing this methodology the need to trim fault trees for each generated design can be overcome. Moreover, the conversion of each phase fault tree to its BDD - as is required for the evaluation of the mission failure probability - can also be omitted. Owing to the generally increased number of fault trees in phased mission system analysis, this methodology becomes especially beneficial and computationally more efficient than that used in GSDOA.

7.3.4. Mathematical Representation of the Problem (Overall Mission Constraints)

The initial phased mission system design optimisation problem is introduced as the minimisation of the failure probability for the overall mission by altering the system design structure. Mathematically the problem is expressed as follows:

$$\min Q(\mathbf{X})_{mission} \quad (7.14)$$

Here $Q(\mathbf{X})_{mission}$ is the mission failure probability. \mathbf{X} (n -dimensional vector of independent variables) is the result of the union of vectors of the failure probability values of system components that ensure successful system performance and their individual or combined failures cause mission failure:

$$\mathbf{X} = \bigcup_{j=1}^m \mathbf{X}_j . \quad (7.15)$$

Here, m is the number of phases in the mission. Each vector \mathbf{X}_j represents the failure probability values of the system components whose failures contribute towards the system failure during phase j ($j = 1, 2, \dots, m$). In other words, \mathbf{X} is a vector of failure characteristics of components of a system with a specific design.

As in the GSDOA case the objective function cannot be expressed in an explicit form and the methodology based on fault tree and BDD analysis is employed to evaluate the value of the function as discussed in Section 7.3.3. Using the BDD based approach introduced by Prescott *et. al.* [89] for quantitative system analysis the mission failure probability is expressed as a sum of phase failure probabilities (Equation 7.6). Therefore the optimisation problem can be defined as follows:

$$\min Q(\mathbf{X}_i)_{mission} = \sum_{j=1}^m Q_j(\mathbf{X}_{ij}), \quad (7.16)$$

where $Q_j(\mathbf{X}_{ij})$ is the probability of failure during phase j for the system with design i .

The requirements for a new design are typically associated with its reliability, cost, weight and system size. In the proposed optimisation algorithm reliability (unreliability) of a phased mission system is the optimisation objective. The remaining factors are considered as optimisation problem constraints. Thus the possibility to set a limitation on the system cost, weight and volume has been implemented in the algorithm. Three inequalities are introduced regarding design limitations:

$$\begin{aligned} Cost_{\min} &< Cost_{mission} < Cost_{\max} , \\ Weight_{\min} &< Weight_{mission} < Weight_{\max} , \\ Volume_{\min} &< Volume_{mission} < Volume_{\max} . \end{aligned} \quad (7.17)$$

Here *min* means a defined minimal resource value, accordingly *max* is a maximum defined value and *mission* identifies an existing value of a design characteristic for a specific system design case. To use the resources efficiently it may be useful to have minimum and maximum constraints. If only maximum limit values are needed then the minimum constraint values become equal to zero.

Amounts of resources utilised for a specific design alternative are calculated using the methodology employed in GSDOA. As such the total system design cost is found by summing the design cost of each component, i.e. employing *Equation 4.4*. Since in this case only unreparable systems are considered the maintenance cost is eliminated from the formula resulting in the following expression:

$$Cost_{mission} = Cost_D = \sum_{i=1}^{ncm} cost_d_i \quad (7.18)$$

where $cost_d_i$ is the design cost of a component i and ncm is the total number of different components fitted in the system and subject to failure. Accordingly system weight and volume are evaluated using *Equations 4.12* and *4.13* correspondingly.

When solving a system design optimisation problem it is not an obligation to use all three constraints. Only one or two constraints can be introduced for a particular phased mission design optimisation problem. It is not a requirement that only limitations to system cost, weight or volume are analysed. Other limitations can also be used if values of the introduced limited attributes for the overall system can be evaluated using a formula equivalent to *Formula 7.18*.

7.3.5. Development of Phased Mission System Design Optimisation Programme (PMSDOP)

The programme code for PMSDOA has been developed using the basis of the original GSDOP code and implementing the required amendments. Throughout the development process some of the existing routines have been modified and a number of new routines have been introduced which are discussed in this section. However, the structure of the code has been preserved as the one of GSDOP presented in *Section 4.3.5*.

As a reminder, phased mission system design optimisation programme (MPSDOP) has three groups of routines. The first group of routines is utilised to read the required data from the data files. Routines in the second group control the construction of the mission fault trees representing all possible system design alternatives and their conversion to BDDs. Finally, the third group comprises of one main routine and a number of subroutines implemented for the optimisation process. To avoid repetition only new routines and subroutines developed with regards to the analysis of phased mission systems are introduced in this section.

7.3.5.1. Data Processing

For the performance of the optimisation analysis the required problem data is presented in two groups of files. The first group combines files storing the data of GA parameter values, design variables and system component data such as cost, weight and/or volume together with the imposed limitation on each characteristic. The files storing the names of mission phases also belong to this group. The second group comprises pairs of files storing data for individual phases of the mission. The data files needed for the analysis and links among all data files are shown in *Figure 7.5*.

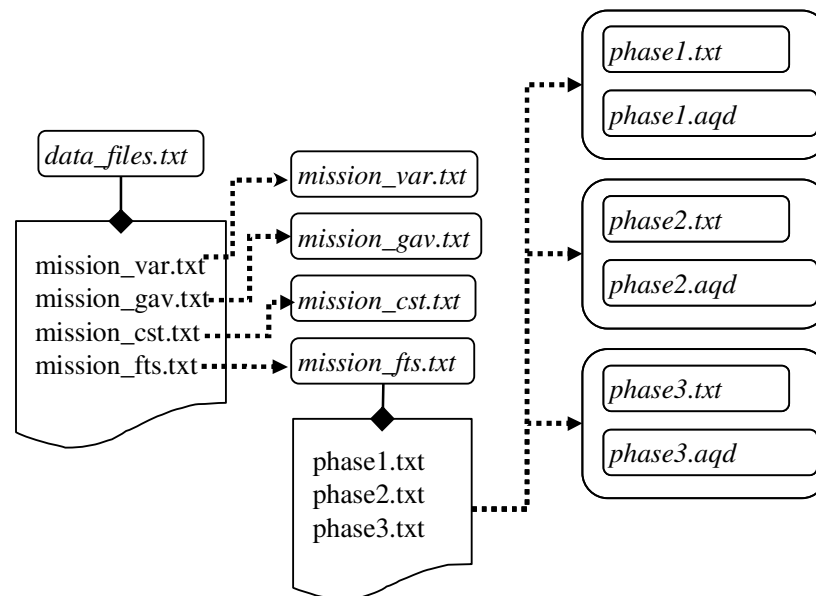


Figure 7.5. Example of Data Files

As in the GSDOP the initial data file named “*data_files.txt*” is utilised in the PMSDOP to store names of the first group of data files. The files listed are named using the same principle as the one in GSDOA. Since only unreparable systems are analysed the data and therefore files associated with system maintenance are not considered. Another change introduced is that the file *_fts.txt* stores the names of data files where fault tree structures for each mission phase are stored.

Data specific to each mission phase is stored in two files for each phase. The first file stores the fault tree structure of the phase of the initial system design. The second file of type *.aqd* and named as the first file stores the failure characteristics of the system components considering that component failure occurs during this phase.

Eight routines are used to read all the data from the provided files. New routines *File_Names* and *Data_Phases* were specially introduced for a phased mission optimisation problem. The routine *File_Names* is used to read the names of files where mission fault tree structures are stored. The routine *Data_Phases* is employed to read each phase initial fault tree structure from a data file. A subset of design variables for each phase is formed using the routine *Data_Variables*. The routines *Data_Files*, *Data_Constraints* and *Basic_Event_Data* have been adapted for phased mission analysis from the GSDOA.

7.3.5.2. Preparation for Quantitative Analysis

The preparation for quantitative analysis stage involves the construction of a fault tree for every mission phase which represents the causes of the system failure for each possible system design during that phase. The process is performed employing the routine *Phase_Fault_Tree_Construction*. It is organised using a loop where one phase fault tree is modified at a time using a corresponding subset of FTMPs.

A computational method to convert the failure logic represented by each fault tree to a BDD structure was implemented using a programme developed at Loughborough University by Remenyte-Prescott [74]. A routine *Files_Bdd* is used to provide the required data for this process. The conversion of the fault trees into BDDs completes the preparation stage for the optimisation process.

7.3.5.3. Optimisation Algorithm

The optimisation algorithm is implemented in the routine *Genetic* which was adapted from GSDOP. The optimisation of a phased mission design starts with the generation of an initial population of chromosomes providing values for utilised design variables. The information obtained is then passed to a new routine called *Mission_Fault_Tree_Designs* which is employed to assign the appropriate Boolean states of the house events in each phase BDD. In this routine the rules for assigning values to house events are implemented which ensures that the values of house events associated with the FTMP representing a particular design variable are the same throughout all mission BDDs where the FTMP has been implemented.

With given component failure characteristics, BDDs incorporating all possible design alternatives (constructed for each mission phase) and values of house events specified, the mission failure probability for the generated designs can be evaluated. The programme code used for the quantification analysis has been developed previously at Loughborough University by Remenyte-Prescott and is based on the methodology introduced by Prescott *et. al.*

al. [89]. The executable file of the code has been incorporated in the algorithm source code as an independent process and as a result provides the mission failure probability and failure probability values for each phase.

Two other new subroutines developed are associated with limitations imposed on the system design characteristics. The subroutine *Mission_Resources_Values* evaluates system cost, weight and volume (if these characteristics are considered) and returns the value 0 if at least one constraint is violated. Otherwise it returns the value 1. The subroutine *Mission_Penalty_Adaptive* replaces the subroutine *Penalty_Adaptive* previously used in the improved GSDOP (Chapter 6). However, the methodology for penalising fitness values of infeasible chromosomes remains the same.

The rest of the subroutines and routines employed in the optimisation process are directly adopted from the GSDOP. Thus no changes have been made in the subroutines developed to perform the generation of new populations, crossover and mutation operators and scaling of chromosomes fitness values.

7.4. UAV DESIGN OPTIMISATION USING THE PHASED MISSION DESIGN OPTIMISATION ALGORITHM

7.4.1. Introduction to a Phased Mission of an UAV

The algorithm developed for phased mission system design optimisation problems has been applied to optimise the design of an unmanned aerial vehicle (UAV) for a six-phase mission. For the problem analysed the emphasis is only given to the minimisation of the mission failure probability and design limitations are not considered. Thus, the UAV design optimisation problem is analysed as a single objective unconstrained optimisation problem. The purpose of the analysis is to validate the capability of the developed optimisation algorithm to solve a phased mission optimisation problem.

The UAV is remotely controlled and can perform specific tasks for the duration of its defined mission. UAVs have been used in a variety of forms and for a variety of missions. They can be employed to perform military, civil or research missions. In military applications UAVs are commonly used for missions which would otherwise present a high risk as manned missions. Civil use of UAVs may include aerial photography and observation of traffic patterns. UAVs can also serve as upper atmospheric weather stations.

A generalised mission for a simplified architecture UAV is analysed. It consists of six phases occurring in the following order: *take-off*, *climb*, *en-route* in controlled airspace, *en-route* in uncontrolled airspace, *descent* and *landing*. The UAV mission is considered successful if the aircraft completes all phases successfully.

The first phase of the mission, i.e. the *take-off* phase, is considered to be unsuccessful and mission failure occurs if the landing gear cannot be retracted or either the braking system, propulsion system or avionics system fails. If the *take-off* phase is aborted for any reason or communication errors occur these will also lead to phase failure. Note that a communication mistake can occur in any phase and cause mission failure, and so it appears in each phase fault tree. The detailed fault tree for the *take-off* phase is shown in *Figure 7.6*.

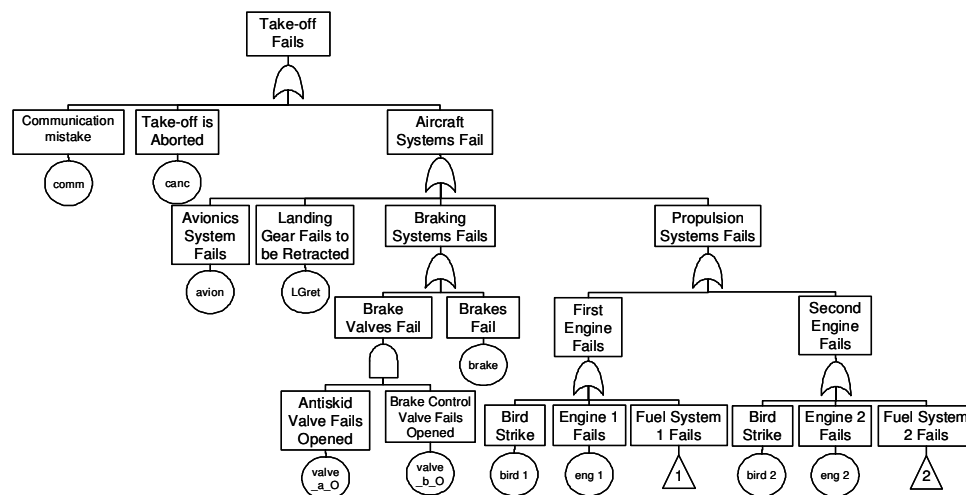


Figure 7.6. Fault Tree for the Take-off Phase.

Following take-off a UAV starts climbing to a particular altitude. During this phase failure in the propulsion or the avionics systems will cause mission failure. Other causes of failure for the phase include flight surface failures or the occurrence of a severe storm. The phase fault tree is presented in *Figure 7.7*. Failure logic for both fuel systems identified with transfer symbols 1 and 2 are developed further in *Figure 7.10*.

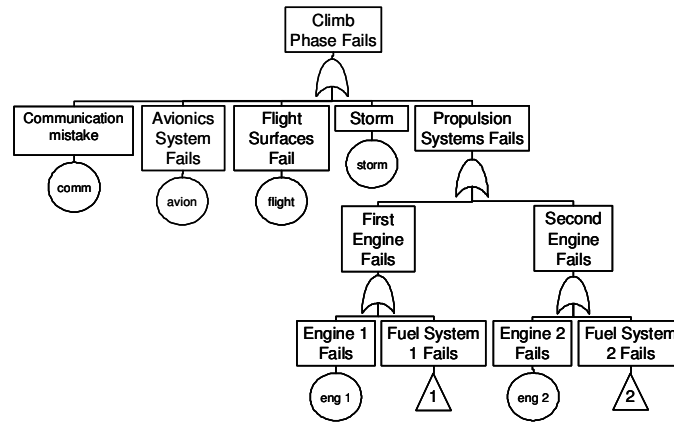


Figure 7.7. Fault Tree for Climb Phase

When the required altitude is reached the *en-route* phase starts. The phase is divided into two phases. At first a UAV flies through controlled airspace after which uncontrolled airspace follows. When the UAV enters controlled airspace the navigation system together with the propulsion and avionics systems are in use. Failure in any of these systems as well as storm occurrence or an air collision will result in mission failure. The same cause of failure appear when the UAV flies through uncontrolled air space. However, when the UAV is in controlled airspace an air collision with another aircraft may occur due to an air traffic control failure. The sense and avoidance system has to fail for an air collision to occur if the UAV is in uncontrolled airspace. As an example, a fault tree of the *en-route* phase in uncontrolled airspace is shown in Figure 7.8. The fault tree for the *en-route* phase in control airspace is similar where the basic event *avoid* is replaced with a basic event *atc* which identifies failure of air traffic control. Fault trees for failures of fuel systems are presented in Figure 7.10.

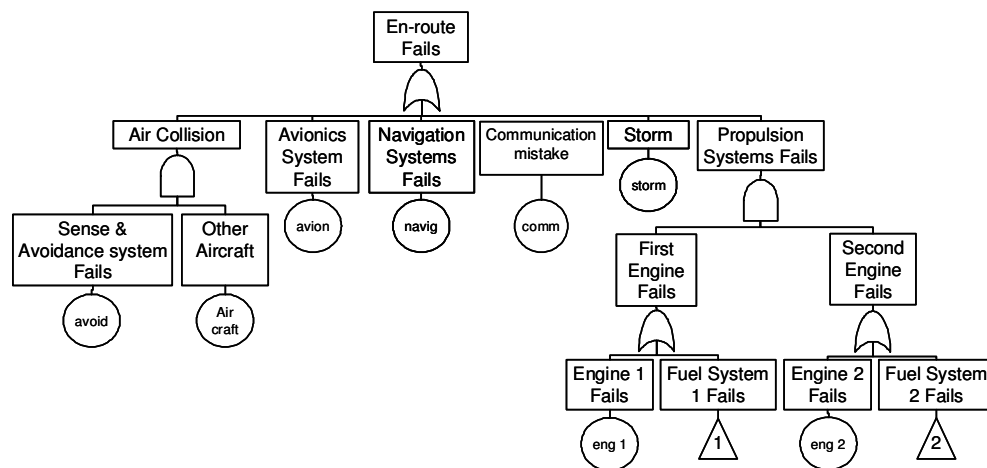


Figure 7.8. Fault Tree for the En-route Phase

The *descent* phase follows the *en-route* phases. During this phase a UAV changes altitude before starting to land. UAV failure causes in this case are identical to the ones that appear when the vehicle is changing its altitude after the *take-off* phase. Therefore the fault tree for the *descent* phase is identical to the fault tree of the *climb* phase shown in *Figure 7.7*.

The last phase of the UAV mission is the *landing* phase. As shown in the phase fault tree in *Figure 7.9* the braking system, propulsion system and avionics system are in use during this phase. Failure of any of these systems causes the mission failure. The phase and at the same time the whole mission can also fail if the landing gear cannot be extended or the flight surfaces fail. If the mission has to be finished in a certain time then abortion of the *landing* phase for any reason will also result in phase failure.

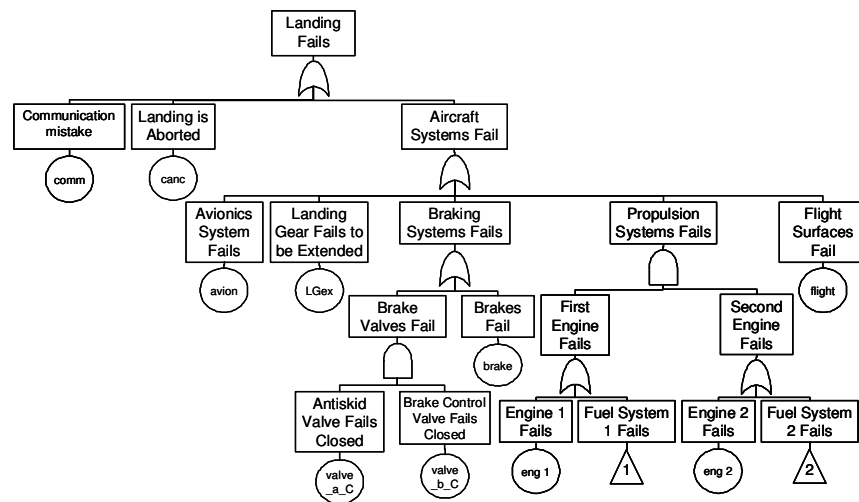


Figure 7.9. Fault Tree for the Landing Phase

The fuel systems are one of several UAV subsystems which operate throughout the mission. Failure of any of them can cause mission failure during any phase. Failure causes for both fuel systems are shown in *Figure 7.10*.

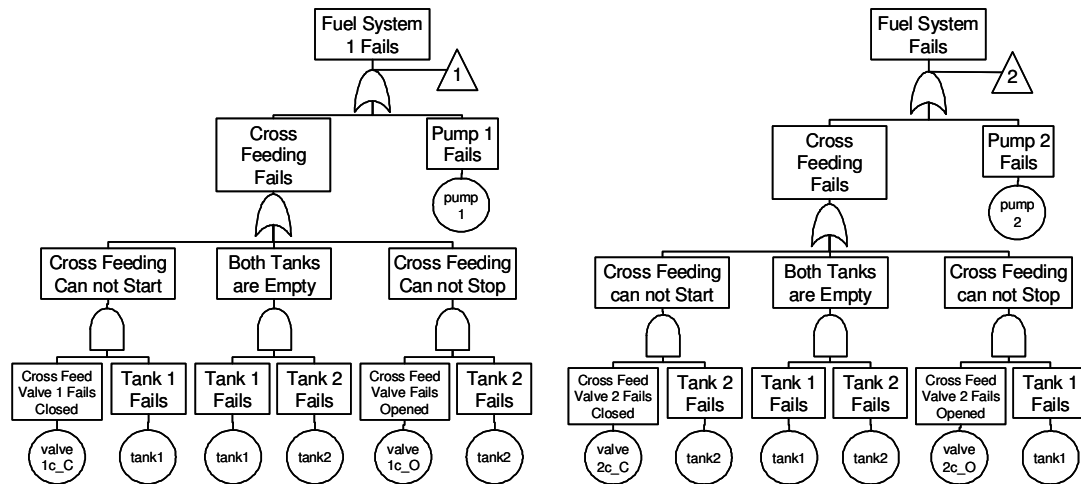


Figure 7.10. Fault Trees of Fuel Subsystems

7.4.2. UAV Design Alternatives

The objective of the introduced UAV design optimisation problem is to minimise the overall UAV mission failure probability by altering the initial vehicle design and constructing an optimal design case. All structural design variables and their parameter values considered for the optimisation problem are presented in *Table 7.1*.

Table 7.1. Design variables

<i>Associated System Component</i>	<i>Description of Design Alteration</i>	<i>Possible Values of Parameters of Design Variables</i>
Landing gear	Type of a landing gear	type1, type 2
Antiskid valve	Type of an antiskid valve	type1, type 2, type 3
Brakes	Number of brake sets	3, 2, 1
	Type of a brake set	type1, type 2
	Minimal number of failed brakes that cause failure to brake	3, 2, 1
Navigation system	Number navigation subsystems	2, 1
	Type of a navigation subsystem	type1, type 2
	Minimal number of failed navigation subsystems causing navigation failure	2, 1
Sense and avoidance system	Number of sense and avoidance subsystems	2, 1
	Type of a sense and avoidance subsystem	type1, type 2

Basic event failure probability data used to perform quantitative analysis of system failure is presented in *Tables A.3.1* and *A.3.2* in *Appendix 3*. *Table A.3.1* provides failure probability values of system components that are in the initial system design together with the probability values of occurrence of the external factors leading to system failure. The failure data for new components which appear in the potential design cases is provided in *Table A.3.2*. It is assumed that failure characteristics of each component do not change through each phase, i.e. their failure probability values in each phase are the same.

As it is known, the programme constructs a chromosome structure corresponding to the problem analysed by utilising values of the parameters of FTMPs. *Figure 7.11* presents the binary form of the chromosome structure. Here the first two bits refer to the first design variable, i.e. the type of the landing gear. The second two bits are used to represent the type of an antiskid valve. The third group of genes are used for the parameters of the design variable referring to the brake sets while the fourth group is used to code parameters of the design variable associated with the navigation system. The last two couples of two bits are used to define the number and type of a sense and avoidance system(s).

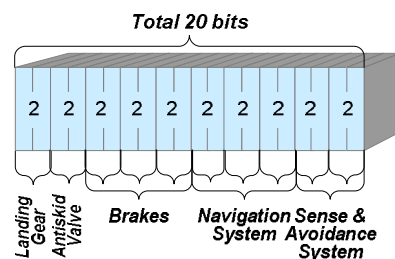


Figure 7.11. Structure of Chromosome

7.4.3. Analysis of Optimisation Results

Two aspects have been considered throughout the analysis of the application of PMSDOP. One of them is the ability of the algorithm to find the optimal (near-optimal) solution. For this purpose the global minimum fitness value has been identified employing the exhaustive search method and evaluating all possible design alternatives. The smallest UAV mission failure probability value that has been found is equal to 0.111118 while the initial design UAV failure probability is 0.135871. The combination of values of design variables corresponding to the minimal mission failure probability is presented in *Table 7.2*.

Table 7.2. Values of design variables for the optimal UAV design

<i>Changeable Component</i>	<i>Description of Modifications / Design Variable</i>	<i>Design Variable Value (New Design)</i>	<i>Initial Design</i>
Landing gear	Type of a landing gear	type 2	Type 1
Antiskid valve	Type of an antiskid valve	type 2	Type 1
Brakes	Number of brake sets	3	1
	Type of a brake set	type 2	Type 1
	Minimal number of failed brakes that cause failure to brake	3	1
Navigation system	Number of navigation subsystems	2	1
	Type of a navigation subsystem	type1	type1
	Minimal number of failed navigation subsystems causing navigation failure	2	1
Sense and avoidance system	Number of avoidance subsystems	2	1
	Type of an sense and avoidance subsystem	type1	type1

Another aspect of the analysis considered is the sensitivity of the optimisation process to variations in the values of the GA parameters. The optimisation process was performed employing a set of different combinations of GA parameter values. Different values of population size, crossover rate and mutation rate were employed while the number of generations was initially chosen equal to 100. Three population sizes were analysed: 50, 30 and 10 chromosomes. Mutation rates were chosen equal to 0.001, 0.005 and 0.01 and crossover rate values were equal to 0.75, 0.8 and 0.95. These values are based upon the earlier experiments from application examples of GSDOA.

For each set of GA parameter values the best fitness value was the main quantity considered in each generation. Due to the stochastic nature of GAs the simulations were run five times for each combination of GA parameter values. Following this the averages were derived for the best fitness values at each generation. The standard deviation was also evaluated for each set of five best fitness values taken from the last generations. Finally, an average number of generations needed for the global minimal fitness value to be found was derived for each set of five runs. All results are provided in *Appendix 3*.

In total one hundred and thirty five runs were carried out to investigate all GA parameters. The main findings are summarised in this section. When the size of chromosome populations was small, i.e. equal to 10, the results showed that the trade-off between the rate at which new strings are introduced and the rate of diversity created in the population needs to be found in order to increase the convergence rate of the algorithm. For instance, if the crossover rate is high then fewer generations are required to approach a near minimal solution when smaller mutation rate values are used. Conversely, using the low crossover rate and higher mutation rates a near optimal solution is found with less generations. The results are presented in

Figures A.3.1-A.3.2 in Appendix 3. The lowest fitness value after performing 100 generations was obtained using the mutation rate equal to 0.01 for all crossover rate values. The global minimum solution was obtained in at least one out of the five runs for all combinations of GA parameters. The average number of generations required to be performed in order to find the global minimum for each combination of parameters is presented in Table 7.3. Results show that on average the smallest number of generations was performed when the mutation rate was equal to 0.05 for any value of crossover rate.

Table 7.3. Number of Generations Performed to Find the Minimal Fitness Value

Crossover Rate	<i>0.75</i>			<i>0.8</i>			<i>0.95</i>		
Mutation Rate	<i>0.001</i>	<i>0.005</i>	<i>0.01</i>	<i>0.001</i>	<i>0.005</i>	<i>0.01</i>	<i>0.001</i>	<i>0.005</i>	<i>0.01</i>
Number of Generations	81	71	74	82	56	58	88	25	54

Means of best fitness values when 30 chromosome populations were used are presented in Figures A.3.5-A.3.7 in Appendix 3. In this case the fitness values approached a near optimal and/or optimal value in less than 20 generations for all combinations of crossover and mutation rates. During all five runs the global minimum was found by employing mutation rates 0.005 and 0.01 in combination with every crossover rate value introduced. Average numbers of generations performed to find the optimal solution (Table 7.4) tended to decrease with the increasing mutation and crossover rates. They are also smaller than the ones obtained when using 10 chromosome populations.

Table 7.4. Number of Generations Performed to Find the Minimal Fitness Value

Crossover Rate	<i>0.75</i>			<i>0.8</i>			<i>0.95</i>		
Mutation Rate	<i>0.001</i>	<i>0.005</i>	<i>0.01</i>	<i>0.001</i>	<i>0.005</i>	<i>0.01</i>	<i>0.001</i>	<i>0.005</i>	<i>0.01</i>
Number of Generations	45	27	20	35	27	19	17	23	36

For populations of 50 chromosomes differences between convergence rates are small and less significant for different combinations of crossover and mutation rates than the ones obtained when using populations of a smaller size as shown in Figures A.3.9-A.3.11 in Appendix 3. After increasing the population size, the global minimum fitness values were found in 100 generations for all five runs using each combination of the mutation and crossover rates. It was noticed that the algorithm requires more generations to be performed in order to find the optimal solution if the mutation rate is increased as shown in Table 7.5. It suggests that the high mutation rate tends to convert the GA into a random search procedure when the population size is large.

Table 7.5. Number of Generations Performed to Find the Minimal Fitness Value

Crossover Rate	<i>0.75</i>			<i>0.8</i>			<i>0.95</i>		
Mutation Rate	<i>0.001</i>	<i>0.005</i>	<i>0.01</i>	<i>0.001</i>	<i>0.005</i>	<i>0.01</i>	<i>0.001</i>	<i>0.005</i>	<i>0.01</i>
Number of Generations	23	33	23	17	23	25	17	30	33

7.4.4. Summary of the Analysis

The analysis carried out to identify GA parameter values that result in good algorithm performance revealed three sets of parameter values. The first set of parameters includes the population size of 30 chromosomes, the crossover rate equal to 0.95 and the mutation rate equal to 0.001. The second and the third sets have the mutation rate of 0.001, population size 50 and crossover rates 0.8 and 0.95 respectively. In this case the quality of optimisation algorithm in terms of performance has been interpreted according to the smallest number of generations required to find the global minimum and the smallest standard deviation of the best fitness values after 100 generations from five runs.

The computations were carried out on a Dual Core 1.60 GHz processor with 1.00GB of RAM on a 32-bit operating system. The average CPU time to evaluate an objective function value was equal to 0.143 seconds and the average time to generate a single population was equal to 6.43 seconds. For this analysis the mutation rate equal to 0.01 was used along with the population size of 50 chromosome and the crossover rate equal to 0.95.

The application example of the UAV design optimisation problem has shown the capability of the developed algorithm to solve a design optimisation problem of an unconstrained multi-phased mission system. During the optimisation process the global optimum of the problem has been found. It is the set of values of structural design variables corresponding to the minimum mission failure probability selected from the introduced design alternatives. The analysis results also suggest that combinations of a large population size and a high crossover rate tend to reduce the number of generations required to be performed in order to find the optimal solution. Furthermore, the optimisation process is more sensitive to the size of population rather than the values of GA operators.

7.5. MILITARY VESSEL DESIGN OPTIMISATION USING *PMSDOP* (CASE 1)

The military vessel performing a mission “Harbour/ Sea Training” is the second example of the application of *PMSDOP*. The military vessel design optimisation problem has been

chosen to demonstrate the capability of the developed algorithm to solve larger scale and more complex problems. The objective of the optimisation is to choose a vessel design that would successfully complete a specified mission with minimal failure probability and optimal usage of available resources. Therefore given limitations to vessel cost and weight have been considered throughout the optimisation process. More complexity to the analysed problem has been added by considering that the failure probability values of vessel components are not the same in each phase.

7.5.1. Initial Military Vessel Design

The simplified vessel structure comprises of six independent systems: propulsion & power system, electrical distribution system, cooling water system, hydraulic system, hydroplanes & steering system and rudder control system. In this example the systems are analysed as non repairable systems. The first propulsion & power system comprises two major subsystems: propulsion power provision and primary propulsion. The fault trees for the subsystems are shown in *Figures 7.12* and *7.13* respectively. Failure of either subsystem causes power disruption in the vessel. All failure causes, i.e. basic events of the fault trees for the propulsion & power system are listed in *Table A.3.6* in *Appendix 3* together with their failure probability values.

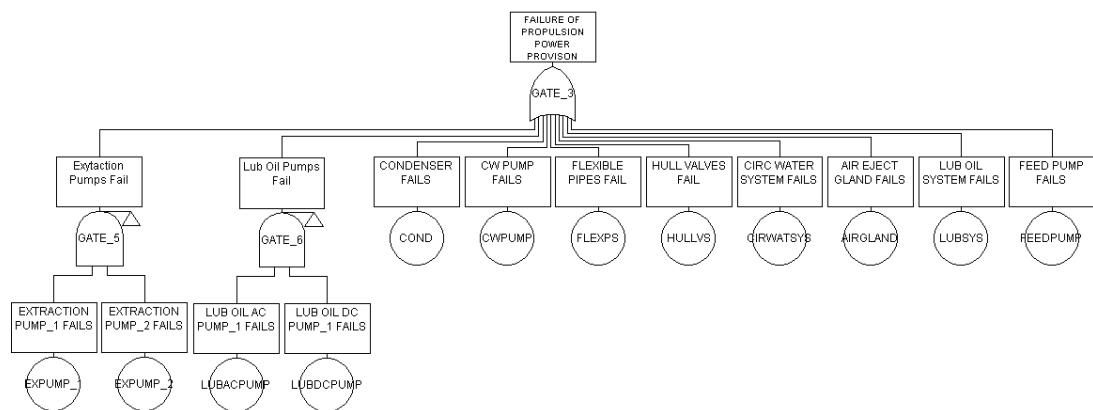


Figure 7.12. Propulsion Power Provision Fault Tree

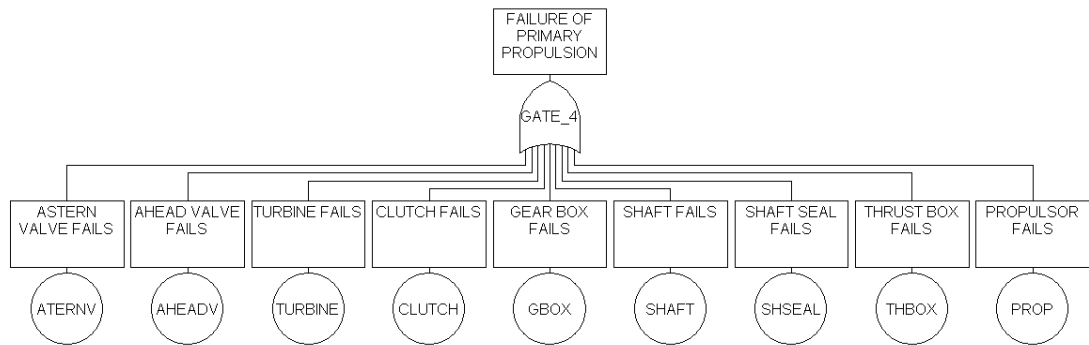


Figure 7.13. Primary Propulsion Fault Tree

The electrical supply system includes alternating current generation, alternating current supply, direct current generation and direct current supply subsystems. Alternating current subsystems are constructed of two redundant units. For example, alternating current can be supplied by two redundant switch boards and the alternating current generation subsystem has two redundant turbine generator units. Both the direct current generation and direct current supply subsystems are constructed to have single units, i.e. no component redundancies are implemented in the subsystems. The fault tree of the system is presented in *Figures 7.14* and *7.15*. *Table A.3.7 (Appendix 3)* provides the list of basic events in the fault trees and their failure probability value.

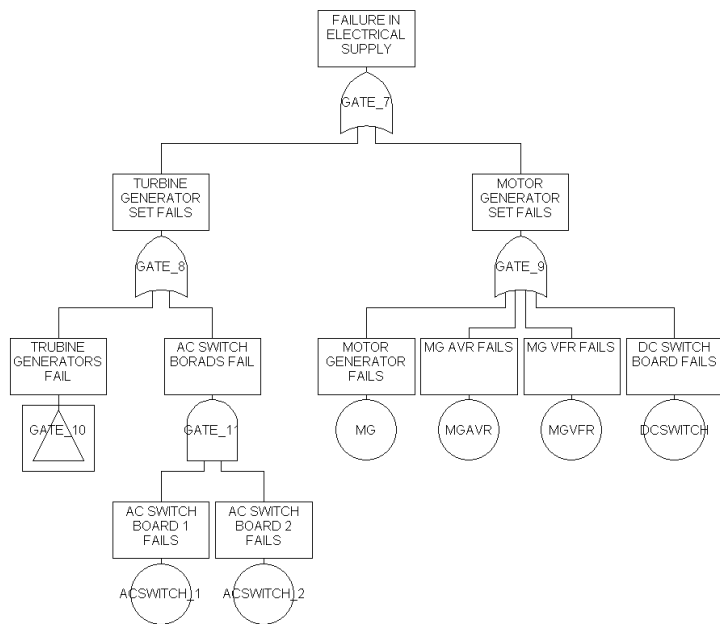


Figure 7.14. Electrical Supply System Fault Tree

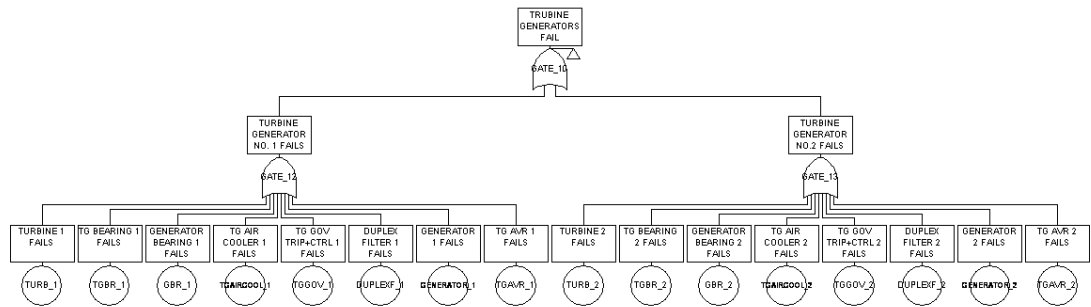


Figure 7.15. Fault Tree for Turbine Generators

The fresh water cooling system has two redundant paths. Each path includes an inlet and an outlet hull valves, two flexible coupling units, a sea water service unit, a pump and a heat exchanger. Therefore the system fails if failures in both paths appear. A fault tree in Figure 7.16 shows all possible system failure causes. Failure data of system components is provided in Table A.3.8 in Appendix 3.

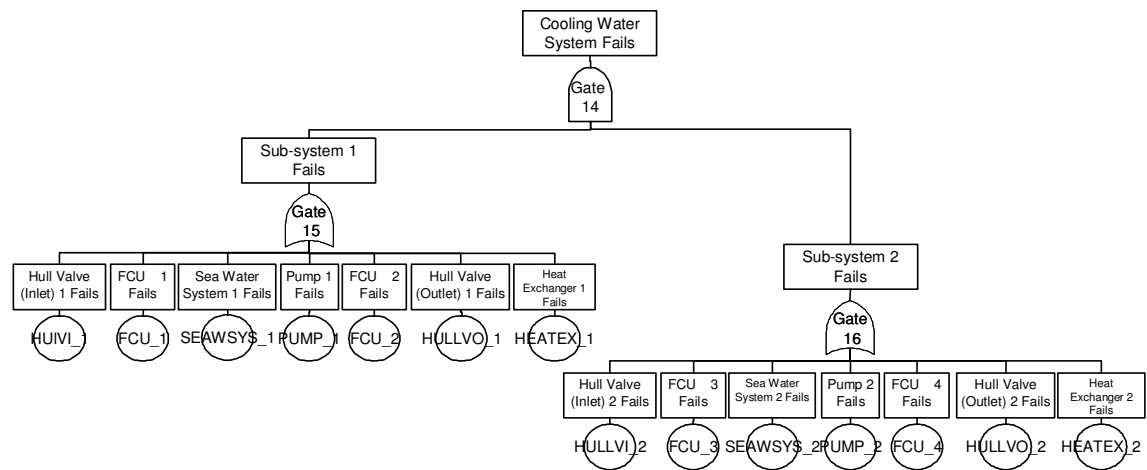


Figure 7.16. Fresh Water Cooling System Fault Tree

The hydraulics system comprises of the following subsystems: external hydraulics, aft hydraulics and main hydraulics subsystem. Failure modes of the external and main hydraulics subsystems are identical. The subsystems fail if either plant or hydraulics unit fails. Aft hydraulics subsystem has two different redundant plants. One of them uses alternating current and the other uses direct current. Causes of failure for the hydraulics subsystems are shown in the fault tree in Figure 7.17. System component failure data is presented in Table A.3.9 in Appendix 3.

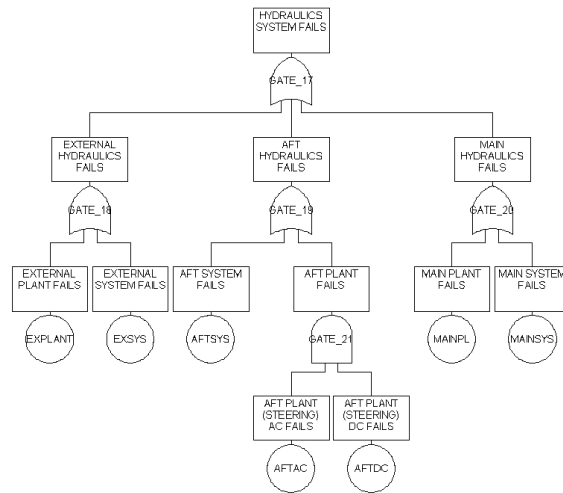


Figure 7.17. Hydraulics System Fault Tree

The hydroplanes control system comprises of aft hydroplane and forward hydroplane control units. Each subsystem has an individual control surface, a ram servo unit and an order transmission box. Hydraulic tilting and air tilting cylinders together with emergency air control are additionally installed in the aft hydroplane control subsystem. The forward hydroplane subsystem also has a tilting cylinder. The fault tree for the hydroplanes control system is given in *Figure 7.18* and component failure data is provided in *Table A.3.10* in *Appendix 3*.

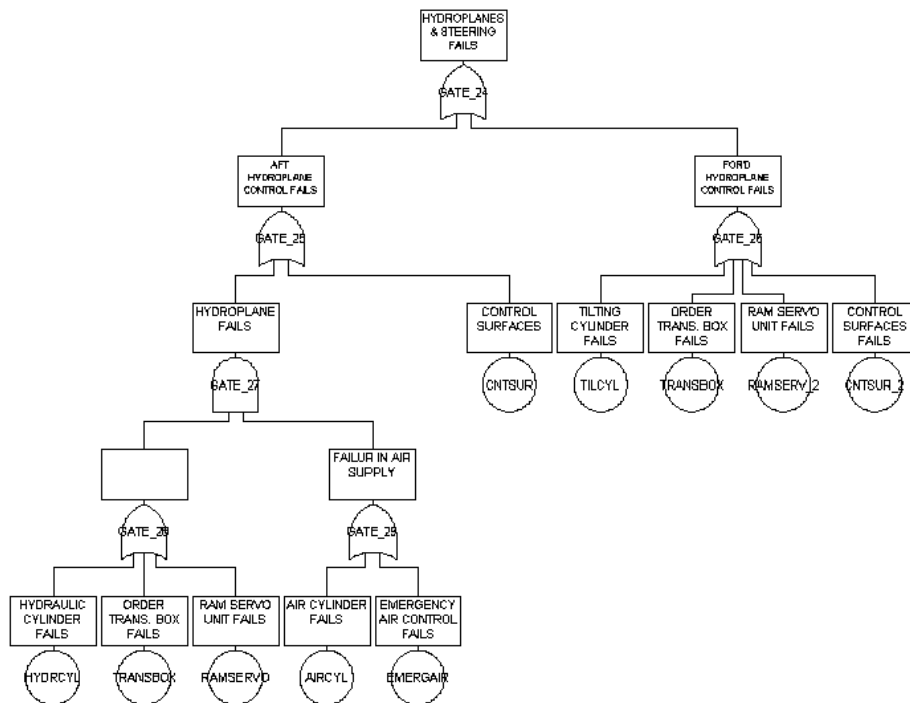


Figure 7.18. Hydroplanes Control System Fault Tree

Finally, the rudder control system failure may be caused by failure of control surfaces, rudder ram failure, ram servo unit failure or rate control failure. *Table A.3.11 in Appendix 3* provides the system components failure data.

The analysed mission of the military vessel is divided into phases according to tasks needed to be completed. The mission comprises four phases carried out in the following order: harbour shore support, transit shallow water, receive broadcast and again harbour shore support. During the mission the number of earlier described systems being in use varies. For example, during the *Harbour* phase the vessel does not perform any task and therefore the electrical supply system alone is required to operate. Therefore the phase fault tree corresponds to the electrical supply system fault tree which was presented in *Figures 7.14 and 7.15*. When the vessel transits to deep waters or in order to perform the task of the *Broadcast* phase all six systems are in operation. Failure of the mission during those phases will occur if any of these systems fail. Therefore the fault trees for the latter phases comprise the fault trees given for the different ship systems as shown in *Figure 7.19*.

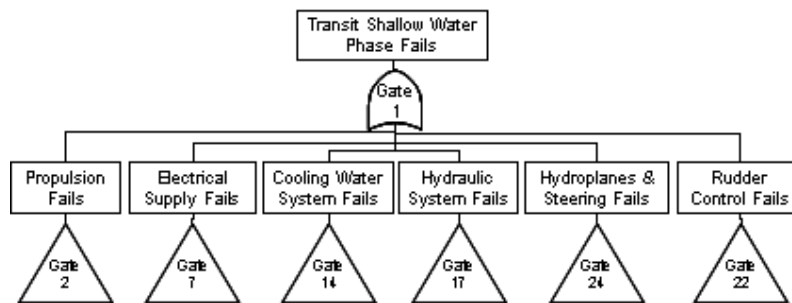


Figure 7.19. Transit Shallow Water Phase Fault Tree

7.5.2. Military Vessel Design Alternatives

Six components have been chosen to be replaced if necessary in order to develop new design alternatives which would reduce the failure probability of the mission. As in the previous optimisation examples, the possible modifications to the vessel systems designs are characterised with the list of structural design variables. The list is provided in *Table 7.6*.

Table 7.6. Design Variables and their Values

<i>Component</i>	<i>Design Variable Description</i>	<i>Design Variable Value</i>
Circulating Water (CW) Pump	Number of CW Pumps	3, 2, 1
	Number of CW Pumps Required to trip	3, 2, 1
	Type of a CW pump	Type 1, Type 2
Feed Pump	Number of Feed Pumps	4, 3, 2, 1
	Number of Feed Pumps required to Trip	4, 3, 2, 1
	Type of a Feed Pump	Type1, Type 2
Ahead Valve	Number of Ahead Valves	3, 2, 1
	Type of an Ahead Valve	Type 2, Type 1
Motor Generator (MG) Voltage and Frequency Regulator (VFR)	Number of MG VFRs	2, 1
	Type of a MG VFR	Type 1, Type 2
External Hydraulic Plant	Type of an External Hydraulic Plant	Type 1, Type 2
Main Hydraulic Plant	Type of a Main Hydraulic Plant	Type 1, Type 2

Owing to the potential introduction of new components and their arrangements new basic events are incorporated in the fault trees of the mission phases. As a reminder, if design alterations result in incorporation of components that differ from those originally used their corresponding basic events have also different failure probability values. In the case analysed there are six such components. The list of basic event failure probability values corresponding to the new components is provided in *Table A.3.12* in *Appendix 3*.

Development of potential vessel design alternatives involves the considerations of two design characteristics, such as cost and weight of the military vessel. A new design is required not to exceed the initial design cost and weight which are 400 units and 21000 units respectively. Cost and weight values for the individual vessel components are estimated. Units of measurements are non-dimensional for both component characteristics. The cost and weight of each component are provided in *Table A.3.13* in *Appendix 3*.

As there are 12 design variables specified for the military vessel design optimisation problem, the chromosome structure created for the optimisation process comprises 12 genes. The first three genes are used for design variables associated with the CW pump. The second group of three genes relates to the feed pump. The third group of genes is employed to code the number and type of ahead valves in the vessel. The next two genes refer to the VFR of the motor generator. Finally the last two genes are used to identify the types of external and main hydraulic plants respectively. The complete structure of the chromosome is presented in *Figure 7.20*.

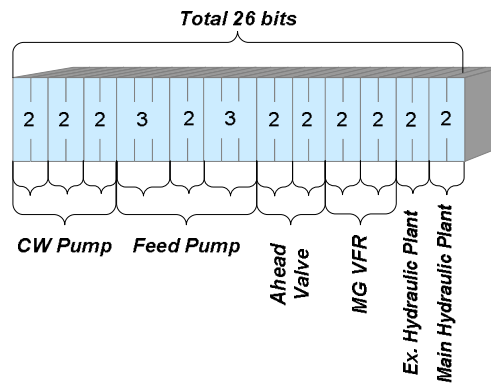


Figure 7.20. Structure of Chromosome

7.5.3. Analysis of Optimisation Results

Solving the military vessel design optimisation problem the same aspects of the algorithm performance have been considered as the ones in the UAV optimisation case. These are the ability of the algorithm to find the global optimal solution and the influence of different values of GA parameters to the optimisation process. Thus first the minimum failure probability value and the corresponding optimal set of values of design variables have been found by performing an exhaustive search. The smallest vessel mission failure probability value was equal to 0.0945415 considering the limitations set on the overall vessel cost and weight. The failure probability of the initial design vessel is 0.1231664. The list of the values of design variables corresponding to the optimal design is presented in *Table 7.7*. The cost of the optimal design vessel is 399.9 units and its weight is equal to 20376 units.

Table 7.7. Design Variables and their Values

<i>Component</i>	<i>Design Variable Description</i>	<i>Design Variable Value for the Optimal Design</i>	<i>Design Variable Value for the Initial Design</i>
CW Pump	Number of CW Pumps	2	1
	Number of CW Pumps Required to Operate	1	1
	Type of a CW pump	Type 2	Type 1
Feed Pump	Number of Feed Pumps	2	1
	Number of Feed Pumps required to Operate	1	1
	Type of a Feed Pump	Type1	Type1
Ahead Valve	Number of Ahead Valves	3	1
	Type of an Ahead Valve	Type 2	Type 1
MG VFR	Number of MG VFRs	2	1
	Type of a MG VFR	Type 1	Type 1
External Hydraulic Plant	Type of an External Hydraulic Plant	Type 2	Type 1
Main Hydraulic Plant	Type of a Main Hydraulic Plant	Type 1	Type 1

The same scheme to analyse the performance of the algorithm is used as the one for UAV problem. The algorithm efficiency is studied for the sets of GA parameters where population size is equal to 50, 30 or 10 chromosomes, mutation rate is equal to 0.001, 0.005 or 0.01 and crossover rate value is equal to 0.75, 0.8 or 0.95. As done previously, simulations are run five times for each combination of GA parameter values and the best feasible chromosome fitness value yet found is stored at each generation. The means of the best feasible fitness values over five runs for each generation are used to compare algorithm performance. The standard deviation and the best mean feasible chromosome fitness value obtained in 100 generations for each set of five best fitness values are also evaluated. All detailed results can be found in *Appendix 3*.

The means of best feasible chromosome fitness values yet found at each generation for every combination of crossover and mutation rates when population size is set to 10 chromosomes are shown in *Figures A.3.12-A.3.14* in *Appendix 3*. The mean values are noticeably bigger for the combination of both small mutation and crossover rates. It means that the results tend to converge to a local minimum when the diversity of a population is small. Increasing the crossover rate eliminates the problem of population diversity when using a small mutation rate. Utilising populations of 10 chromosomes the GA with both higher mutation and crossover rates has produced design options with on average smaller failure probabilities. The results are given in *Table 7.8*.

Table 7.8. Mean Values of the Lowest Failure Probability after 100 Generations

<i>Mutation Rate</i>	<i>0.001</i>	<i>0.005</i>	<i>0.01</i>
<i>Crossover rate</i>			
0.75	0.098335	0.094922	0.094934
0.8	0.095628	0.095027	0.095212
0.95	0.095406	0.095189	0.094765

When the population size has been increased up to 30 chromosomes the optimisation results obtained using different mutation rates for the same crossover rate have assimilated. The biggest improvement in optimisation results appear on average in 25 generations. From the results in *Table 7.9* it can be seen that when using different mutation rates the minimum mean failure probabilities are less diverse in their values when the crossover rate is 0.75 than using the crossover rate equal to 0.95. The tendency also appears for the minimum mean values to decrease when at least one value of the genetic operator increases. The global minimum was

found at least in one run out of five for every mutation rate when the crossover rate was equal to 0.8 and using mutation rates of 0.001 and 0.01 when the crossover rate was 0.95.

Table 7.9. Mean Values of the Lowest Failure Probability after 100 Generations

<i>Mutation Rate</i> <i>Crossover rate</i>	<i>0.001</i>	<i>0.005</i>	<i>0.01</i>
0.75	0.094736	0.094778	0.094695
0.8	0.094768	0.094668	0.094728
0.95	0.094676	0.094828	0.094562

When the population size is increased up to 50 chromosomes the biggest improvements in optimisation results appear on average in 20 generations. Therefore when increasing the population size the chances to find an optimal solution in fewer generations are also increased. In this case the mean fitness values obtained are smaller for combinations of a mutation rate of 0.01 with every crossover rate as seen in *Table 7.10*. Standard deviations of the minimum values of five runs were also the smallest when combinations of GA parameters included a mutation rate equal to 0.01. These results are presented in *Figure A.3.23* in *Appendix 3*. The global minimum was found at least once for every combination of crossover and mutation rates when using a population of 50 chromosomes.

Table 7.10. Mean Values of the Lowest Failure Probability after 100 Generations

<i>Mutation Rate</i> <i>Crossover rate</i>	<i>0.001</i>	<i>0.005</i>	<i>0.01</i>
0.75	0.094755	0.09461	0.094588
0.8	0.094736	0.094732	0.094571
0.95	0.094632	0.094591	0.094571

7.5.4. Summary of the Analysis

Since GA parameters as the factors influencing the performance of the algorithm tend to be less problem-dependent there is scope to find sets of their values aimed at improving the performance of the GA in more general applications. The performed analysis has shown that on average the best performance of the algorithm is achieved when using a population of 50 chromosomes, crossover rate equal to 0.95 and mutation rate equal to 0.01. In this case the global optimum solution was identified. It was also noticed that in solving the larger-scale

constrained problem, better optimisation results were obtained using the higher mutation rate (0.01) rather than using the lower one as used when solving the UAV optimisation problem. It suggests that a large population size and high crossover and mutation rates are promising in achieving good performance of the algorithm for larger constrained optimisation problems.

For the analysis of the timescales of the optimisation process the population of 50 chromosomes with the mutation rate equal to 0.01 and the crossover rate equal 0.95 were used. As previously the computations were carried out on a Dual Core 1.60 GHz processor with 1.00GB of RAM on a 32-bit operating system. The average CPU time to evaluate the military vessel mission failure probability value was equal to 0.164 second and the average time to generate a single population was equal to 9.454 seconds.

The military vessel design optimisation example indicates the ability of the algorithm to solve a large scale and complex optimisation problem for a phased mission system. Moreover it provides evidence that the algorithm is not a system-specific optimisation tool and has the potential to analyse a range of optimisation problems for different phased mission systems.

The developed approach combining FTA and the GA is a novel optimisation tool. Its novelty pertains to its application to solve design optimisation problems of phased mission systems and the potential to analyse a range of different optimisation problems. Systems having components with changing failure probabilities across phases and constraints defined for systems overall cost, weight and/or volume can be analysed.

7.6. SYSTEM DESIGN LIMITATIONS FOR INDIVIDUAL PHASES

At the initial development stage of PMSDOA the optimisation objective was to minimise the overall mission failure probability subject to the overall system cost and/or weight and/or its volume. In some cases a number of system parts can be inactive during some phases and they are not considered when performing both qualitative and quantitative phase failure analysis. Only active subsystems are analysed. It follows that it would be beneficial to introduce design constraints only for active subsystems in every phase.

Introduction of constraints for each phase individually enables the impact of design alterations to both subsystems in use and the overall system to be analysed. Defined constraint limits and design requirements set for different phases control the choice of design alterations for specific subsystems and therefore a more precise analysis of the overall system design characteristics can be performed. The capability to analyse either a single subsystem or a

group of subsystems depends on the system being analysed. For example, if analysing the earlier introduced military vessel mission, it is possible to define design requirements for only one specific subsystem, as the electrical distribution subsystem is active during the first and the last phases.

The developed PMSDOA has been enhanced with the possibility to use additional constraints set at each phase of the mission. Two groups of constraints were introduced. The first group refers to system characteristics such as cost, weight and volume. The characteristics evaluated for a particular phase refer only to the subsystems active in that phase. In order to quantify the corresponding properties, characteristics of components in each subsystem are added up. For example, consider system cost in phase j . System cost, i.e. cost of subsystems which are active, is evaluated as follows:

$$Cost_{mission}^j = \sum_{k=1}^{ncm_i^j} cost_k \quad (7.19)$$

Here ncm_i^j is the total number of different components subject to failure included in subsystems which are in use in phase j , $cost_k$ is the cost for a component k which has an associated basic event in the phase fault tree. The subsystems analysed represent a system with a particular design i .

If any costs referring to the phase cannot be associated with any of the basic events then these costs are included as an additional fixed cost in the total phase cost:

$$Cost_{mission}^j = \sum_{k=1}^{ncm_i^j} cost_k + cost_{additional}^j \quad (7.20)$$

If a design of any active subsystem during the phase analysed is changed then the phase cost will also change. Such a methodology allows the cost of certain subsystems to be separated from the overall system cost and analysed individually. For example, consider again a harbour phase of the military vessel where only an electrical distribution subsystem is active. In this case the phase cost is equal to the cost of the subsystem. Thus alterations to vessel design can be identified which do not overcome the predefined cost limits of the electrical subsystem.

The system weight and volume for a particular phase can be estimated using the same formulas with corresponding component characteristics.

Since the optimal usage of available resources is considered both minimum and maximum constraints are introduced for the design characteristics for each phase. The constraints for resources can be defined as follows:

$$\begin{aligned} Cost_{\min}^j &< Cost_{mission}^j < Cost_{\max}^j, \\ Weight_{\min}^j &< Weight_{mission}^j < Weight_{\max}^j, \\ Volume_{\min}^j &< Volume_{mission}^j < Volume_{\max}^j. \end{aligned} \quad (7.21)$$

where j identifies a phase number, min means a defined required minimal limit of a system attribute, accordingly max is a maximum defined limit and $mission$ identifies the system attribute for a particular design when performing a task in phase j .

In some cases design alterations can result in the reduction of system failure probability during certain phases but at the same time they can increase the system failure probability for the other phases. The second group of constraints are placed on phase failure probability values for different phases. The following set of equations (7.22) is used:

$$\begin{aligned} Q_1(\mathbf{X}_1) &\leq Q_1(\mathbf{X}_1)_{\max} \\ Q_2(\mathbf{X}_2) &\leq Q_2(\mathbf{X}_2)_{\max} \\ &\dots\dots\dots \\ Q_m(\mathbf{X}_m) &\leq Q_m(\mathbf{X}_m)_{\max} \end{aligned} \quad (7.22)$$

Here $Q_i(\mathbf{X}_i)$ identifies the mission failure probability in phase i , $Q_i(\mathbf{X}_i)_{\max}$ is the maximum allowed system failure probability value in phase i and m defines the number of phases in the analysed mission. The number m varies for different systems. Implementing these constraints allows component combinations to be identified that minimise the failure probability of the whole mission without exceeding limits set for system failure probability values during each phase. It also ensures that the improvement achieved for the whole mission is not a result of a significant improvement in one phase and significant decline of reliability in another.

The amendments introduced to the algorithm extend its possibilities of applications. Problems analysed can be specified more accurately resulting in more precise optimisation results.

The improved PMSDOA was applied to solve both the UAV and military vessel design optimisation problems. In the application examples constraints for failure probabilities and/or design characteristics for each phase were introduced. The application examples of PMSDOP to solve the UAV design optimisation problems were published in the proceedings of ISSC

conference [90] and International Journal of Performability Engineering [91]. The application examples of the improved PMSDOA to solve the military vessel design problems were published in the proceedings of the conference ESREL-2008 [92] and the International Journal of Reliability and Safety [93]. The military vessel design optimisation problem with constraints defined for the electrical distribution system and the overall vessel is discussed in detail in the following section (*Section 7.7*).

7.7. MILITARY VESSEL DESIGN OPTIMISATION PROBLEM WITH CONSTRAINTS ADDED AT EACH PHASE (CASE 2)

In this section the application of the improved PMSDOP to solve the earlier analysed military vessel design optimisation problem (*Section 7.5*) is discussed. The problem has been extended and constraints for system cost and weight of active subsystems for each phase together with system failure probabilities during every phase have been introduced. The aim of the application is to analyse the capability of the algorithm to solve a larger scale and more complex optimisation problem with an increased number of constraints.

Owing to the fact that only the electrical subsystem is active during the harbour phase its own design characteristics can be analysed together with the overall ship characteristics. The limits introduced for to system failure probabilities for each phase allows the overall system failure probability to be minimised and at the same time a failure probability of the electrical distribution to be maintained below the predefined limit. Similarly the introduced cost and weight constraints at each phase are the limits for cost and weight of the electrical distribution system and the whole vessel. Thus in the application example three sets of constraints are considered:

$$\begin{aligned}
 Q_{new_design}^j &\leq Q_{max}^j \\
 Cost_{new_design}^j &\leq Cost_{max}^j \\
 Weight_{new_design}^j &\leq Weight_{max}^j
 \end{aligned}
 \tag{7.23}$$

Here j is phase number and $j = \{1, 2, 3, 4\}$; $Q_{new_design}^j$ denotes the failure probability of a new design vessel during phase j while Q_{max}^j is a predefined limit of a probability that the vessel fails during the same phase. The remaining variables identify the cost and weight for a new design and the predefined limits in the corresponding phases.

All limits for constraints were evaluated. Probabilities of mission failure in each phase of the initial military vessel design were employed as guidance limits of the failure probability constraints used for the optimisation problem. Limits for cost and weight constraints were also estimated according to the initial design vessel cost and weight in each phase. The limits introduced are listed in *Table 7.11*.

Table 7.11. Limits of Constraints

<i>Phase</i>	<i>Failure Probability</i>	<i>Cost</i>	<i>Weight</i>
Harbour	0.0058	120	1300
Transit	0.055	400	21000
Broadcast	0.058	400	21000
Harbour	0.0053	120	1300

In order to analyse the influence of different GA parameter values on the performance of the algorithm the same values of crossover and mutation rates were used as the ones in the previous application examples. However the previous examples of applications of both GSDOA and PMSDOA have revealed that populations of 10 chromosomes produce the worst results. Furthermore owing to the fact that this is a more complicated problem a larger population size has been introduced and populations of 30 and 70 chromosomes were utilised. The termination condition, i.e. the number of generations performed, has also been changed and reduced to 75 generations. This allows the reduction of the time required to find an optimal solution. However it still needs to be investigated to ensure that by performing just 75 generations a global minimum can be found.

As in the previous cases five simulations were run for each combination of different values of GA parameters. The average and best fitness values were the two main quantities considered in each generation. These derived values were then used for further analysis. In addition, the average number of generations, needed to be performed in order to find the global minimal fitness value, was also derived for different combinations of crossover and mutation rates.

An exhaustive search has also been performed for this problem. The global feasible minimum value of mission failure probability was the same as that for the first ship optimisation problem, i.e. 0.094542. The list of the values of design variables defining the optimal ship design is presented in *Table 7.7*. However, if the cost limits for the second and third phases were just a few units higher, for example 402 units, the set of design variable values representing the optimal system design would have been different from that for the first ship optimisation problem.

7.7.1. Analysis of Optimisation Results

The average fitness values for each generation obtained when using sets of different GA parameter values are presented in *Figures A.3.24-A.3.27* in *Appendix 3*. They show the best feasible fitness values for each generation for the same sets of algorithm parameters.

To summarise, the average fitness values are larger when populations of 70 chromosomes are used. A higher mutation rate also increases the average fitness values for all three crossover rate cases. On the other hand both the size of a population and mutation rate have contrasting effects on the best fitness values. The best fitness values converge to a minimum value faster when a population of 70 chromosomes is used. It is also obvious that a higher mutation rate influences the results in the same way if the low crossover rate is used. However it cannot be stated that by using higher mutation rates fewer generations are performed before the minimum fitness values has been found when using crossover rates equal to 0.8 and 0.95. The average numbers of generations required to obtain the optimal fitness values are presented in *Table 7.12*. It can be seen that the algorithm finds the optimal solutions in fewer generations if the crossover rate is increased in combination with a large population size and high mutation rate.

Table 7.12. Number of Generations Performed to find the Minimal Fitness Value

Mutation Rate \ Crossover Rate	Population Size =70			Population Size = 30		
	0.001	0.005	0.01	0.001	0.005	0.01
0.75	75	75	67	75	75	75
0.8	38	75	32	75	75	75
0.95	38	71	32	75	75	75

7.7.2. Summary of the Analysis

The military vessel design with the lowest failure probability which satisfies constraints defined for the electrical distribution system and the overall vessel has been identified using few sets of different values of GA parameters. The best performance of the algorithm has been achieved using a population of 70 chromosomes, mutation rate equal 0.01 in combination with two crossover rate values: 0.8 and 0.95. The observation was made after taking into account the number of generations required to be performed before the optimal solution had been found.

The successful application example demonstrates the potential of the algorithm to solve large scale and strongly constrained phased mission system design optimisation problems. When analysing the influence of GA parameters on the performance of the algorithm it was noticed that the same rules can be applied for choosing crossover and mutation rate values for both the highly constrained and the less constraint optimisation problems. The algorithm performs better if both the mutation and crossover rates are high. Moreover by increasing the number of constraints considered, the population sizes should also be increased in order to find an optimal or a near-optimal solution when it is envisaged that a relatively small number of generations will be performed.

7.8. SUMMARY

The research detailed in this chapter has focused on the development of a basic methodology to solve multi-phased mission system design optimisation problems which has not been systematically treated in the literature. The following goals have been achieved:

- The methodology implemented to solve safety system design optimisation problems has been successfully adapted to multi-phased mission system design optimisation problems.
- The programme code, PMSDOP, for the new PMSDOA has been developed to automate the design optimisation problem for a chosen phased mission system.
- The PMSDOA and PMSDOP maintain the essential property of the original approach, i.e. potential application to a broad range of multi-phased mission systems.
- The new PMSDOA has been applied to solve different optimisation problems catering for:
 - 1) the overall mission failure minimisation considering mission constraint limitations;
 - 2) the overall mission failure minimisation considering phase failure and explicit phase constraints limitations.

Three application examples of the developed PMSDOA have been analysed. The solved UAV design optimisation problem was relatively simple. No constraints were considered in the analysis. The first military vessel design optimisation problem was more complex and larger

scale than the UAV problem and had limitations introduced for the overall vessel cost and weight. In the second vessel design problem restrictions were introduced on the cost and weight of active subsystems and mission failure probability during each mission phase.

The optimal designs were found corresponding to the global minimum failure probabilities of the missions for all three cases. As expected the optimisation process converged much slower and more generations were required to find the global minimum value when larger scale and more complex problems were solved, i.e. ship optimisation problems.

A study of the influence of different GA parameter values on the optimisation process was also carried out. Results showed that the algorithm performed better when larger population sizes and both higher crossover and mutation rates were used. It was also noticed that the algorithm performance is more sensitive to population size than the chosen variations of crossover and mutation rates.

Consideration is needed for the computational intensity and processing time of the PMSDOP. Differences in computational intensity and processing time between the two problems were evident. It took on average 220 seconds to run 100 generations using 50 chromosomes populations for the UAV problem. While solving the military vessel design optimisation problem the average run time was 560 seconds using the same values of GA parameters. In addition, it requires more generations to be performed in order to find a global solution. In solving the current problems the run time is not an issue as the analysis does not need to be performed in real-time. However, improvements made to the quantitative system analysis methodology as well as advancements of the implemented GA could significantly reduce the processing time and improve the efficiency of the programme. This could improve the potential of the algorithm to solve real time and more complex mission design optimisation problems.

8. DESIGN OPTIMISATION OF MULTI-PHASED MISSION SYSTEM CONSIDERING MULTIPLE MISSIONS

8.1. INTRODUCTION

Most real world systems are designed to perform a number of missions. When improving a phased mission system design it is desirable that it would also improve the likelihood of success of its different missions. In some cases a system design that guarantees a low failure probability for one mission may have no influence on system performance in another mission or may increase its failure probability in the worst case. Thus, optimisation of phased mission system design should involve an analysis of all the missions the system is designed for or at least those deemed the most critical.

A problem of optimisation of a system design involving a number of missions can be analysed as a multi objective optimisation problem. In this case the failure probabilities of each mission being minimised are the objectives of the problem. A system design that provides a trade-off across the minimal failure probabilities for each mission is the solution of the problem. A number of constraints can also be included in the problem being solved.

Building upon the developed single objective PMSDOA, a multi objective optimisation technique has been developed. The new algorithm has been named as the multiple phased missions system design optimisation algorithm (MPMSDOA).

In this chapter a brief introduction is given about multi-objective optimisation (*Section 8.2*). Evolutionary algorithms used for multi-objective optimisation and the detailed explanation of the adapted multi-objective genetic algorithm follow in *Section 8.3*. In *Section 8.4* the methodology developed to solve multi objective phased mission design optimisation problems is introduced. The last section (*Section 8.5*) of the chapter addresses the application example of the developed approach and discusses the optimisation results obtained.

8.2. MULTI-OBJECTIVE OPTIMISATION

A multi-objective optimisation problem has more than one objective which are to be minimised or maximised. Without loss of generality (a component to be maximised can be

converted into a minimisation objective by multiplying by negative one), consider the problem of simultaneously minimising the k components $f_i, i = 1, \dots, k$, of a vector function f :

$$f(\mathbf{x}) = (f_1(\mathbf{x}), \dots, f_k(\mathbf{x})). \quad (8.1)$$

Here $\mathbf{x} = (x_1, \dots, x_n)$ is an n -dimensional decision variable vector in the decision variable space \mathbf{X} [94]. The decision variable space is generally restricted by a number of constraints. Thus the multi-objective optimisation problem can be stated as:

$$\min f(\mathbf{x}) = \min(f_1(\mathbf{x}), \dots, f_k(\mathbf{x})) \quad (8.2)$$

subject to the following constraints:

$$\begin{aligned} g_i(\mathbf{x}) &\geq 0, i = 1, \dots, I; \\ h_j(\mathbf{x}) &= 0, j = 1, \dots, J. \end{aligned} \quad (8.3)$$

The solution to the problem is a vector $\mathbf{x}^* = (x^*_1, \dots, x^*_n)$ which minimises a given set of objective functions.

Single-objective optimisation algorithms have one objective to be optimised and so they provide one unique solution for it. A general multi-objective optimisation problem presents a set of solutions and each of them represents a trade-off in the objective space. For example, consider a situation when system reliability needs to be improved at as low a cost as possible. One way to improve it is by introducing redundant components. The new components will increase the cost of the system. The more redundant components are introduced the more expensive the system becomes. If the system reliability needs to be improved and its cost minimised the two objectives will be conflicting. A system with minimum cost will have a low reliability rate. Conversely, the system with high reliability will cost more. The problem will not have one unique solution. Instead there will be solutions where a trade-off between reliability and cost exists. Thus a resulting outcome to a general multi-objective optimisation problem is a set of optimal solutions, each of which satisfies the objectives and is not dominated by any other solution [95].

Due to the existence of conflicting objectives the concept of domination is used in multi-objective optimisation algorithms. If considering a minimisation problem, dominance of one solution over another can be defined as follows:

Definition 8.1. A solution $\mathbf{x} = (x_1, \dots, x_n)$ is said to dominate another solution $\mathbf{y} = (y_1, \dots, y_n)$ ($\mathbf{x} \prec \mathbf{y}$), if both conditions are true [94]:

The solution \mathbf{x} is no worse than the solution \mathbf{y} in all objectives, i.e. $\forall i \in \{1, \dots, n\}$,
 $f_i(\mathbf{x}) \leq f_i(\mathbf{y})$.

The solution \mathbf{x} is strictly better than the solution \mathbf{y} for at least one objective function, i.e.
 $\exists i \in \{1, \dots, n\}: f_i(\mathbf{x}) < f_i(\mathbf{y})$.

If any of the above conditions is violated, the solution \mathbf{x} does not dominate the solution \mathbf{y} .

Definition 8.1 defines the relationship between any two solutions. Three types of relationships can be identified between solutions on the basis of dominance [94]: a) solution \mathbf{x} dominates solution \mathbf{y} , b) solution \mathbf{x} is dominated by solution \mathbf{y} , c) solutions \mathbf{x} and \mathbf{y} do not dominate each other.

For a given finite set of solutions there exist groups of solutions of different non-domination levels [45]. One of the groups represents solutions which are non-dominated with respect to each other.

Definition 8.2. Among a set of solutions \mathbf{X} , the non-dominated set of solutions P^* are those that are not dominated by any member of set \mathbf{X} .

Definition 8.3. A solution $\mathbf{x} = (x_1, \dots, x_n)$ is set to be Pareto-optimal if and only if there is no other solution $\mathbf{y} = (y_1, \dots, y_n)$ in \mathbf{X} for which $f_i(\mathbf{y})$ dominates $f_i(\mathbf{x})$.

The non-dominated set of Pareto-optimal solutions is called the Pareto-optimal set. Any two Pareto-optimal solutions do not dominate each other. This set also has the property of dominating all other solutions which do not belong to this set. In other words, any solution in the Pareto-optimal set will dominate any other solution from outside the set. For a given Pareto-optimal set the corresponding objective function values in the objective space are called the Pareto front. It follows that the search space of objectives can be divided into two non-overlapping regions. One region is optimal and another is non-optimal. Thus the solution of the multi-objective optimisation problem is a set of solutions from the Pareto optimal set.

There are two goals in multi-objective optimisation. The first goal is to identify a set of solutions as close as possible to the Pareto-optimal set. The convergence of the solutions to

the true optimal solutions corresponds to convergence of the single objective optimisation algorithm to an optimal unique solution. The second goal is entirely specific to a multi-objective optimisation problem. Solutions in the best-known Pareto set should be as diverse as possible. Only a diverse set can provide a good set of trade-off solutions amongst the objectives.

The discussed multiple Pareto optimal solutions do not exist for all multi-objective optimisation cases. The optimal Pareto set contains more than one solution only if the objectives of the problem are conflicting to each other. If the objectives are not conflicting to each other, the cardinality of the Pareto-optimal set is one. It means that any objective function has the same optimal solution [94].

8.3. MULTI-OBJECTIVE OPTIMISATION TECHNIQUES

A number of classical search and optimisation techniques exist to deal with multi-objective optimisation problems. Coello *et al.* [96] grouped them into three categories: enumerative, deterministic and stochastic. The classical methods combine the objectives into a single, parameterized objective function by analogy to decision making before search. The parameters of this function are systematically varied by the optimizer. Each time an optimisation run is performed one particular Pareto-optimal solution is achieved. Thus the method needs to be applied several times with different parameter settings in order to find different solutions and achieve the approximate Pareto-optimal set. The weighted sum method, ϵ -constrained method, weighted metric methods and goal programming method are a few representatives of the classical techniques. Debs [94] mentioned potential difficulties which may accompany these techniques. For example, most algorithms need to be applied many times in order to find the approximation to the Pareto-optimal set. Some of the techniques may require additional knowledge about the problem being solved. Sometimes they may be sensitive to the shape of the Pareto front. Moreover classical methods are rather inefficient for problems having discrete search spaces. Due to their drawbacks these methods are not widely used in practice to solve multi-objective optimisation problems.

Recently, GAs have become established as an alternative to classical methods to explore the Pareto-optimal front in multi-objective optimisation problems. These algorithms have proven themselves as a general, robust and powerful search mechanism. They have the ability to find multiple Pareto-optimal solutions in a single run, they work without derivatives, converge speedily to Pareto-optimal solutions and handle combinatorial optimization problems. GAs

are also less susceptible to the shape or discontinuity of a Pareto front and have the potential to converge to Pareto-optimal solutions with a high degree of accuracy [97].

The first multi-objective GA was proposed by Schaffer in 1984 and was called vector evaluated GA (VEGA). The next important contribution towards the development of a GA based optimisation technique was made by Goldberg [45]. He introduced the new non-dominated sorting procedure. Since then many researches have developed different versions of multi-objective optimisation algorithms. Several survey papers have been published on evolutionary multi-objective optimisation where authors have attempted to summarise studies in the field from different perspectives and introduced comparative analysis of different algorithms [98], [99], [100]. Generally speaking, multi-objective GAs can be categorised on the bases of elitism, diversification approaches or different fitness assignment strategies [95].

Fonseca and Fleming [99] compared the fitness assignment procedures and distinguished plain aggregating approaches, population-based non-Pareto approaches, and Pareto-based approaches. The first group of methods combine the objectives into one higher scalar function that is used for fitness calculation. The objectives may be combined using either addition, multiplication or any other combination of arithmetic operations [98]. There are, however a number of problems applying such approaches. For example, some accurate scalar information on the range of the objectives needs to be provided in order to avoid dominates of one objective over the others. On the other hand, if the required information is available this is not only the simplest approach, but also one of the most efficient [98]. Popular aggregation methods are the Weighted-sum Approach, Target Vector Optimization, and the Method of Goal Attainment.

As the title implies, Population-based non-Pareto approaches are based on population policies or special handling of the objectives and generate multiple non-dominated solutions. However actual definition of Pareto optimality is not directly implemented in these algorithms [99]. Some of the most popular approaches that belong to this category are the earlier mentioned VEGA, Lexicographic Ordering and Weighted Min-max Approach.

The idea of using Pareto-based fitness assignment was first proposed by Goldberg. All approaches of this type use Pareto dominance in order to determine the reproduction probability of each individual [101]. Examples of such approaches are Multi Objective Genetic Algorithm (MOGA), Non-dominated Sorting Genetic Algorithm (NSGA) proposed by Srinivas and Deb [102] and Niche Pareto Genetic Algorithm (NPGA).

Other well known algorithms that have also been widely studied [95] are: Strength Pareto Evolutionary Algorithm (SPEA), Improved SPEA (SPEA2), Pareto-Archived Evolution Strategy (PAES), Rank-Density Based Genetic Algorithm (RDGA) and Random Weighted Genetic algorithm (RWGA).

8.3.1. Handling of Constraints

Solutions of some real-world problems may be constrained by a number of restrictions imposed on the decision variable. A number of different constraint handling strategies exist for a single objective GA [96], [103]. One of the strategies is to create such genetic operators that always produce feasible solutions. A repair process can also be implemented where infeasible solutions are transformed in to being feasible. Another methodology employs a penalty function to increase the fitness of infeasible solutions (when a minimisation problem is considered). There is also a methodology called death penalty. When using this methodology all infeasible solutions are discarded from the analysis process.

The first three penalty strategies can be directly applied in a multi-objective GA [95]. However implementation of the penalty method is not straight forward in those multi-objective GAs where fitness assignment is based on the non-dominance rank of a solution. Otherwise if fitness assignment is based on objective function values, as in plain aggregating approaches, penalty function strategies can also be implemented directly.

Several methods have been developed specifically to solve constraint multi-objective problems. For example, Jimenez et al. [104] proposed a niched selection strategy to deal with infeasibility while maintaining diversity and dominance of the population. Deb et al. [105] proposed a constrained tournament method where the constraint-domination concept and binary tournament selection method based on it are implemented.

8.3.2. Performance Metrics

There are multiple optimisation goals in the multi-objective optimisation, unlike the single optimisation goal for a single objective optimisation. The first goal requires a search towards the Pareto-optimal set and to discover solutions as close to the Pareto optimal solutions as possible. The second goal requires a search along the Pareto-optimal front in order to find solutions as diverse as possible in the obtained non-dominated set. Since the two goals of the optimisation are distinctive and somewhat conflicting, no single metric can be used to assess the performance of the algorithm in an absolute way. Therefore at least two performance metrics should be used [94].

A number of different performance measures have been suggested. Fonseca and Fleming [106] developed statistical methods. Zitzler and Thiele in [101] used dominated area in the objective function space. Visualization of a Pareto set of solutions can also be used to demonstrate the performance of the algorithm [107]. The quantitative metrics (performance metrics) give a quantitative performance measurement and also are simple to formulate.

The performance metrics can be categorized in three groups: metrics that measure the convergence to the Pareto-optimal front explicitly, metrics that are used to measure diversity among the obtained solutions explicitly and the ones which measure both goals of multi-objective optimisation [94]. Three metrics are given here which were chosen as potential metrics to evaluate the performance of the developed algorithm.

Generational distance [94] or simply called the convergence metric [105] measures how far the given solutions of the obtained Pareto-optimal set are on the average from the true Pareto-optimal front. For each solution obtained, a minimum Euclidian distance between the solution and the solutions on the Pareto-optimal front is computed [107]:

$$GD = \frac{1}{|P^*|} \sum_{\mathbf{x} \in P^*} \min \{ \|\mathbf{x} - \mathbf{x}_T\|; \mathbf{x}_T \in P_T \}, \quad (8.4)$$

where $|P^*|$ is the cardinality of an obtained Pareto-optimal set P^* , $\|\cdot\|$ is Euclidian distance metric and P_T is the true Pareto-optimal set of the problem. A small value of this metric indicates good convergence towards the Pareto-optimal front.

Let P_1^* and P_2^* be two obtained Pareto sets. The function C maps the ordered pair (P_1^*, P_2^*) into the interval $[0, 1]$:

$$C(P_1^*, P_2^*) = \frac{|\{ \mathbf{x}'' \in P_2^*; \exists \mathbf{x}' \in P_1^* : \mathbf{x}' \prec \mathbf{x}'' \}|}{|P_2^*|}. \quad (8.5)$$

The value $C(P_1^*, P_2^*) = 1$ signifies that all solutions in P_2^* are dominated by or equal to solutions in P_1^* . The opposite, value $C(P_1^*, P_2^*) = 0$ means that no solution in P_2^* is covered by any of the solutions in P_1^* . The measure C , which is also sometimes called coverage metric, can be extended for the performance measure of an algorithm if a true Pareto set is

known [97]. Thus the metric $C(P^*, P_T)$ will determine what proportion of the solutions obtained in the Pareto-optimal set are dominated by members of the true Pareto-optimal set.

The spacing metric evaluates diversity among non-dominated solutions. A number of definitions of this metric have been proposed [97], [107]. For the algorithm analysis the following metric was employed [105]:

$$S = \sqrt{\frac{1}{|P^*|} \sum_{i=1}^{|P^*|} (d_i - \bar{d})^2}, \quad (8.6)$$

where d_i is the minimum value of the sum of the absolute differences between the value of the objective function of the i -th solution and the values of the objective function of any other solution in the P^* , i.e.:

$$d_i = \min_{j \in P^* \wedge j \neq i} \sum_{m=1}^k |f_m^i - f_m^j|. \quad (8.7)$$

In Equation 8.6 \bar{d} is the mean value of the distances d_i is given by:

$$\bar{d} = \frac{\sum_{i=1}^{|P^*|} d_i}{|P^*|} \quad (8.8)$$

This spacing metric measures the standard deviations of different d_i values. When the obtained solutions are near uniformly spaced, the distance measure will have a small value [105].

8.3.3. Proposed Multi-Objective Optimisation Technique

The purpose of this part of the research was to expand the application of the originated system design optimisation tool and develop a methodology for optimising a design of the system performing a number of multi-phased missions. The developed optimisation approach needed to be altered and a multi-objective GA (MOGA) has been used instead of a single-objective one. The RWGA which transforms a multi-objective problem into a single-objective one can be easily adapted as an optimisation technique in the earlier developed PMSDOA. Moreover, the complexity of the algorithm is smaller than other multi-objective evolutionary algorithms while the algorithm can be very efficient [98]. Therefore the RWGA has been chosen as the

core to the new developed GA. This section provides a detailed explanation of the developed MOGA.

Murata and Ishibuchi [108] proposed a weighted-based algorithm where the following weighted sum approach is used in order to combine multiple objective functions into a scalar fitness function (it is assumed that all the objective functions should be minimised):

$$f(\mathbf{x}) = w_1 f_1(\mathbf{x}) + \dots + w_i f_i(\mathbf{x}) + \dots + w_k f_k(\mathbf{x}), \quad (8.9)$$

where \mathbf{x} is an n -dimensional decision variable vector (a string if using GA terminology), $f(\mathbf{x})$ is a combined fitness function, $f_i(\mathbf{x})$ is the i -th objective function, w_i is a constant weight for $f_i(\mathbf{x})$, and k is the number of objective functions.

If a weight vector $\mathbf{w} = (w_1, \dots, w_n)$ is constant for each generation, the search direction in the genetic algorithm becomes fixed. In the proposed approach random weights are introduced to search for Pareto optimal solutions. A normalised weight vector \mathbf{w} is randomly generated for each solution during the selection phase at each generation. A random real number is assigned to each weight using the following equation:

$$w_i = \frac{r_j}{\sum_{j=1}^n r_j}, \quad (8.10)$$

where r_j is a non-negative random number and $\sum_{i=1}^n w_i = 1$. This approach utilizes multiple search directions in a single run without using additional parameters [95].

The elite preserve strategy was also implemented in the original algorithm. After a new population is generated a random portion of it is replaced with an equal number of solutions chosen from the external population. The external population is stored and updated at every generation. However in the algorithm developed for the optimisation of multiple phased missions a different strategy was used for maintaining non-dominated solutions.

The implemented elitist strategy to preserve non-dominated solutions and induce convergence of the population towards Pareto optimality was based on that proposed by Marseguerra *et al.* [109]. During the optimisation search, non-dominated solutions are stored and updated using an external archive. The maximum size of the archive is set to $2N$, where N is the number of

strings in the population. Non-dominated solutions of a current population are compared with those stored in the archive after each generation using the concept of dominance described in *Definition 8.1*. The following outcomes are possible:

1. If a new solution dominates an existing member of the archive, the dominated solution is replaced.
2. If a new solution is dominated by any member of the archive, no changes are made to the current archive.
3. If the new solution and the rest of the archive members do not dominate each other either of the following rules apply:
 - 3.1. If the archive is not full, the new solution is added into the archive,
 - 3.2. If the archive is full, the new individual replaces the member of the archive which is the closet to the new solution in the solution space. The Euclidean distance is used to measure the distance between two solutions.

The concept of elitism is implemented by introducing members of the external population of non-dominated solutions into the selection procedure. The mating pool of parent strings ready to undergo the selection process is combined with members of the current population where N_{elite} random strings are removed and replaced with the N_{elite} randomly selected members of the archive. Typically N_{elite} is not more than $N/4$.

The procedure of the proposed algorithm can be summarised as follows:

Step 1: Random population of N chromosomes is generated.

Step 2: Fitness values are assigned to each generated solution $\mathbf{x} \in P$:

1. A random number r_i in $[0,1]$ is generated for each objective $i, i = 1, \dots, k$.
2. The random weight for each objective is generated using *Formula 8.10*.
3. Fitness of the solution is calculated using the given *Formula 8.9*.

Step 3: The selection probabilities are specified for each string of the population as follows:

$$P_i(x) = \frac{f_i(x) - f^{\min}}{\sum_{x \in P} (f_i(x) - f^{\min})}, \text{ where } f^{\min} = \min\{f(x) | x \in P\} \quad (8.11)$$

Step 4: Pairs of parents are selected using probability values from Step 3. Single point crossover is performed to produce N offspring chromosomes. While performing crossover mutation is carried out with a predefined mutation rate.

Step 5: The external population of non-dominated solutions is updated.

Step 6: N_{elite} random strings are removed from the current population and replaced with the N_{elite} randomly selected members of the external elite population.

Step 7: If the maximum number of generations has not been reached the algorithm is started from Step 2. Otherwise the external archive of non-dominated solutions is the final set of Pareto-optimal solutions.

Both tasks required in the multi objective algorithm are presented in the proposed GA. It converges to the Pareto-optimal solutions as the application analysis discussed later in this chapter will show. The diversity in the non-dominated solutions is maintained in two ways: 1) a random weight vector is used to evaluate each solution stimulating to search for different solutions in the Pareto-optimal region, and 2) a proportion of the population is replaced with the solutions from the external set [94].

8.4. MULTIPLE PHASED MISSIONS SYSTEM DESIGN OPTIMISATION ALGORITHM (MPMSDOA)

In real world problems it is common that one system is designed to perform a number of different missions. It is therefore possible that the system reliability changes with every mission owing to the different tasks performed. If the system structural design and therefore its reliability has been improved on the basis of one mission analysis the resulting outcome does not provide information on how these changes influence system performance in the rest of the missions. Moreover the changes may result in an undesirable decrease of system reliability in some of the missions. If all missions to be performed are analysed every time an alteration is introduced in the system design this problem can be overcome.

8.4.1. Mathematical Representation of the Problem

Following the considerations above it was decided to alter the PMSDOA and to adapt it to analyse systems designed for multiple multi-phased missions. In the earlier developed PMSDOA (*Chapter 7*) the problem was formulated as the minimisation of the overall mission failure probability. The multiple missions problem can also be solved as a minimisation

MOGA. Thus, at first fault trees representing all possible system design alternatives are created for each considered mission. As in the PMSDOA FTA and BDD analysis are employed and the fault tree structure alterations are made using rules of FTMPs. The generation of system design alternatives and the assessment of their unreliability is governed by the MOGA.

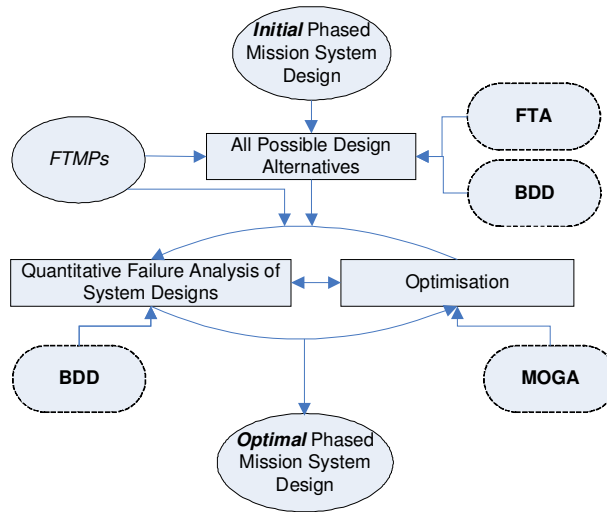


Figure 8.1. Structure of the MPMSDOA

The data required for the optimisation process includes a number of different data groups. An initial system design is given in the form of fault trees constructed for each phase of every mission analysed. To construct fault trees representing all possible design alternatives a list of components chosen to be replaced and associated design variables with their parameter values are used. Failure and design characteristics of all potential system components which are used for qualitative system design analysis also need to be provided. The exception to this applies only to design characteristics for components since they do not need to be given if no restrictions to the system design are considered. On the other hand, if system design restrictions are considered they should also be provided. Finally, the user can also define the GA parameter values as in the earlier versions of the optimisation algorithm.

It is considered that the analysed system performs one mission at a time. As such the missions can be treated as being independent and can be analysed individually. Therefore the optimisation approach is developed so that fault trees representing all possible system design alternatives are constructed for each mission at a time. Thus, FTMPs corresponding to design variables found in fault trees for phases of the first mission are implemented first. Next, the

corresponding FTMPs are employed to alter the fault trees of the second mission. The process is continued until the last mission fault trees are modified. Moreover, the methodology for implementation of FTMPs in one mission fault trees can be directly adapted from the PMSDOA. The resulting fault trees are then converted to their BDDs and are used for quantitative system failure analysis in this form.

The optimisation part of the MPMSDOA implements the MOGA which is based on RWGA and was discussed in detail in *Section 8.4*. The main principle of the algorithm is to combine a number of objective functions into one scalar function. The number of objectives is not fixed in the algorithm, allowing the analysis to be performed for various numbers of missions. It therefore ensures the potential of the algorithm to analyse different design optimisation cases. The principles for the evaluation of the objective functions (in this case more than one objective function is assigned for each chromosome) remain the same.

The process to find values of objective functions for each chromosome, i.e. to evaluate a particular design system failure during each mission, is organised as follows. Once a generated chromosome is decoded and the phenotypes of its genes are obtained sets of house events introduced in the fault trees (BDDs) are defined accordingly. Since the missions are considered to be independent the quantitative analysis for each mission is also performed for each mission at a time. Thus first house events of the first mission are set to either 0 or 1. The same procedure is then performed for the second and the following missions. The house event numbering rules and the rules for the assignment of their values according to the generated values of the design variables are the same as the ones in the PMSDOA. The next stage involves evaluating the failure probability for the mission. Having values of house events defined, each mission can undergo the quantification process. As one mission is analysed at a time, the same algorithm as the one implemented in the PMSDOA is used.

8.5. UAV DESIGN OPTIMISATION USING THE *MPMSDOA*

8.5.1. Problem Overview

The UAV design optimisation problem presented earlier when discussing the PMSDOA has also been chosen as an application example of the MPMSDOA and multiple phased missions system design optimisation programme (MPMSDOP).

The UAV considered in the analysis is designed to perform two missions. One mission that needs to be performed involves flying over controlled airspace. It consists of the following

phases: *take-off*, *climb*, *en-route* (in controlled airspace), *descent* and *landing*. The second mission requires operation of the UAV over uncontrolled airspace. The phases constituting the mission are the following: *take-off*, *climb*, *en-route* (in uncontrolled airspace), *descent* and *landing*. Fault trees for each phase are the same as the ones given in *Section 7.4*. No constraints for any mission were defined.

For the UAV design optimisation with respect to its failure minimisation during both missions replacements of seven components have been considered. The complete list of structural design variables and their values is given in *Table 8.1*.

Table 8.1. Design variables

Component	Description of Modifications / Design Variable	Possible Values of Design Variables
Landing gear	Type of a landing gear	type1, type 2
Antiskid valve	Type of an antiskid valve	type1, type 2, type 3
Brakes	Number of brake sets	3, 2, 1
	Type of a brake set	type1, type 2
	Minimal number of failed brakes that cause failure to brake	3, 2, 1
Engine 1	Type of engine 1	type1, type 2, type 3
Engine 2	Type of engine 2	type1, type 2, type 3
Navigation system	Number navigation subsystems	2, 1
	Type of a navigation subsystem	type1, type 2
	Minimal number of failed navigation subsystems causing navigation failure	2, 1
Sense and avoidance system	Number of sense and avoidance subsystems	2, 1
	Type of a sense and avoidance subsystem	type1, type 2

While developing the algorithm it has been considered that the reliability characteristics of the components could vary for both different phases and missions. Certain existing external factors can influence these changes. For example, if a mission is performed only under extreme weather conditions, some UAV parts can have higher failure probabilities. On the other hand, the same parts operating under normal weather conditions would have lower failure probabilities. It was decided to introduce such a situation in the UAV analysis. The estimated failure probabilities for the majority of components are the same throughout both missions. However potential new components have different failure characteristic for mission

1 and mission 2. The data for quantitative analysis is provided in *Appendix 4*. Failure probabilities for the components in the initial design are given in *Table A.4.1*. *Table A.4.2* provides data of failure characteristics for the possible new components. Using the data and considering the initial UAV design failure probabilities of the first and the second missions are equal to 0.14276 and 0.16792 respectively.

8.5.2. Analysis of Optimisation Results

The introduced UAV design optimisation problem has been used to analyse the performance of the developed MPMSDOA. As in the previous optimisation cases, the optimisation process was performed by employing different combinations of GA parameter values. Different values of population size, crossover rate and mutation rate were employed. Three population sizes were analysed: 30, 50 and 70 chromosomes. Three mutation rates were used: 0.001, 0.005 and 0.01. Crossover rate values were equal to 0.75, 0.8 and 0.95. The algorithm was set to terminate after 100 generations for any combination of GA parameter values.

As mentioned earlier in the chapter, two goals are needed to be achieved in any multi-objective optimisation. The obtained results need to converge to the Pareto-optimal set and diversity in the solutions of the obtained Pareto-optimal set needs to be maintained. To evaluate if each of the goals is obtained three performance metrics were used in the optimisation analysis of the UAV design. For each combination of GA parameter values the convergence metric, function C and spacing metric were obtained. The obtained results are presented graphically and can be found in *Appendix 4*. It is known that GAs have a stochastic nature, therefore five independent runs were made for each combination of GA parameter values and the average values were evaluated for all three metrics. These values were considered in the comparative analysis of the algorithm performance.

The true Pareto-optimal set is required for the evaluation of all three performance metrics. The set contains 120 members for the problem solved which were identified after performing an exhaustive search. The results of the search are presented in *Figure 8.2*. It can be seen that the true Pareto-optimal set is not distributed uniformly and its elements arrange in ten clusters. Therefore it is expected that the obtained non-dominated sets of solutions will not be spread uniformly. Some of the obtained optimal solutions, i.e. sets of the optimal values of design variables, are presented in *Table 8.2*.

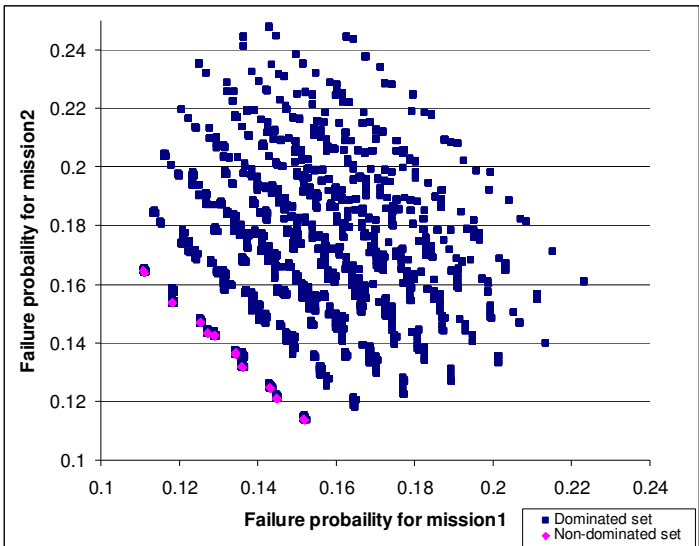


Figure 8.2. True Pareto-optimal Set

Table 8.2. Values of design variables for the optimal UAV design

Component	Description of Modifications / Design Variable	Design Variable Value (New Designs)			
		type 1	type 1	type 2	
Landing gear	Type of a landing gear	type 1	type 1	type 2	
Antiskid valve	Type of an antiskid valve	type 3	type 2	type 1	
Brakes	Number of brake sets	3	3	1	
	Type of a brake set	type 1	type 1	type 2	
	Minimal number of failed brakes that cause failure to brake	3	2	1	
Engine 1	Type of engine 1	type 1	type 3	type 3	
Engine 2	Type of engine 2	type 2	type 1	type 2	
Navigation system	Number navigation subsystems	2	2	1	
	Type of a navigation subsystem	type 1	type 1	type 2	
	Minimal number of failed navigation subsystems causing navigation failure	2	2	1	
Sense and avoidance system	Number of avoidance subsystems	2	2	1	
	Type of an sense and avoidance subsystem	type 2	type 1	type 2	
		$Q^1_{mission}$	0.12710	0.11808	0.14318
		$Q^2_{mission}$	0.14356	0.15400	0.12440

Figures A.4.1-A.4.4 in *Appendix 4* present the results of metrics obtained using different combinations of mutation rates and population sizes with the crossover rate equal to 0.75. The average distance between the solutions of the obtained Pareto set and the solutions of the true Pareto set decrease significantly in 20 generations for all combinations of GA parameters used. The convergence of the results is clear; it can be stated that the mutation rate has an apparent influence on the optimisation process. The metric values decrease if the mutation rate is increased for all population sizes. If the metric values obtained using the same mutation rate and different population sizes are compared it can be seen that the smallest metric values are associated with a population size of 70 chromosomes for all mutation rates. Values of Function C tend to decrease when the mutation rate and the population size increase. The spacing metric gained higher values when populations of 70 chromosomes were used. Since the true Pareto-optimal front is not spread uniformly and forms a number of clusters it was expected that the obtained sets of non-dominated solutions will have even more clusters of solutions. However, the obtained values of the spacing metric are rather small which shows a good spread of the non-dominated solutions.

Figures A.4.5 – A.4.8 in *Appendix 4* provide evaluation results of the algorithm performance when GA parameter combinations comprised different mutation rates and the population sizes and the crossover rate equal to 0.8. In these runs the convergence to the Pareto-optimal set is slower using combinations of mutation rates equal to 0.001 and 0.005 and population size of 30 chromosomes. The non-dominated sets obtained when using a population of 50 chromosomes with mutation rate equal to 0.01 and population of 70 chromosomes with a mutation rate of 0.005 are the closest to the true Pareto-optimal set. In the final non-dominated sets obtained the proportion of solutions, which are weakly dominated by solutions of the true Pareto-optimal set, is smallest for all GA parameter sets with population size equal to 70 chromosomes. However, the increased value of the crossover rate resulted in the increased number of dominated solutions in the final non-dominated sets. The increased value of crossover rate also influenced the maintained diversity in the optimal sets obtained. The spacing values have increased for all combinations of mutation rate equal to 0.01 and different population sizes.

Results of the generational distance obtained using combinations of GA parameter sets with the crossover rate equal to 0.95 are given in *Figures A.4.9 - A.4.12* in *Appendix 4*. The consistent pattern of decreasing convergence metric values with increasing mutation rates noticed earlier does not exist for this case. The smallest generational distances for populations of 30 and 50 chromosomes were achieved using a mutation rate equal to 0.005. However, the

overall smallest average distance between the obtained set and the true Pareto-optimal set was obtained using a mutation rate equal to 0.01 and a population size of 70 chromosomes. The results of coverage metric (Function C) imply that the algorithm performs better using population size of 70. This fact can also be justified by analysing the evaluation results of the maintained population diversity. The spacing metric takes smaller values for every mutation rate when using populations of 70 chromosomes than employing any other size of the population.

8.5.3. Summary of the Optimisation Analysis

The given analysis suggests that differences in values of the mutation rate and the population size have less influence on algorithm performance when a crossover rate equal to 0.95 is used. However the algorithm produced better results overall when larger population sizes were used. The exact influence of the mutation rate on the algorithm performance has not been identified. The Pareto-optimal set obtained maintained diverse solutions and was the closest to the true Pareto-optimal set when the population of 70 chromosomes, crossover and mutation rates equal to 0.95 and 0.01 respectively were employed.

8.6. SUMMARY

During a system design process the objective is to design a cost effective and reliable system. Considering systems performing phased missions, it is often assumed that they are designed to perform one specific task. However, most real world systems perform a number of different missions. Thus the challenge arises how to optimise the phased mission system design which would be suitable for a number of operation tasks and would be reliable within the context of pre-defined design constraints and resources. In this chapter such problems were considered and the main interest was focused on the following:

- development of the basic design optimisation algorithm, MPMSDOA, to optimise the design structure of systems which are constructed to perform a number of different phased-missions;
- development of the code, MPMSDOP, to automate the design optimisation process;
- development of the MPMSDOA and MPMSDOP to be problem-independent, i.e. potentially any system performing any number of missions could be analysed;
- demonstrate the application of the algorithm to solve a chosen system design optimisation problem.

The developed algorithm defines the system design from the given list of possible design alternatives such that a trade-off between failure probability values of different missions is achieved. If failure data of system components given for different missions are conflicting or design alterations have a different impact on the system reliability during different missions a set of optimal design solutions is provided. Otherwise the algorithm will produce one optimal design case.

The MPMSDOA has been applied to optimise a UAV designed to perform two different missions. Some components had failure characteristics which were different for each mission and were conflicting. The performance of the algorithm was analysed using the number of different combinations of GA parameter values. As in the previous application examples, the algorithm tended to find good solutions quicker if higher mutation and crossover rates were used and large chromosome populations were utilised. The obtained results have demonstrated the capability of the algorithm to solve the optimisation problem and find a subset of the true Pareto-optimal set.

The major drawback of the algorithm that has been noticed is the increasing performance time when larger systems or systems designed to perform more missions are analysed. Attention should be given to the improvement of the phased mission quantification technique in this case. The multi-objective GA implemented in the algorithm is one of the simplest examples of MOGA. Therefore the use of a more sophisticated MOGA could also improve the optimisation process, for example, less generations could be required to find the Pareto-optimal solutions. The improved algorithm could be applied to a larger range of systems. More complex systems and a larger number of missions could also be analysed.

9. CONCLUSIONS AND FUTURE WORK

9.1. SUMMARY

Following an extensive critical review on the available design reliability optimisation techniques, the development of a generic optimisation approach applicable to a diverse range of real engineering systems was deemed necessary. The majority of the methods used for reliability design optimisation problems consider simple or well-structured systems. Moreover, usually only one or two means through which system reliability can be enhanced are considered. For example, a number of methods take into account the possible choices of a redundancy strategy or increasing the reliability of constituent components in the system, or a combination of both strategies. In some cases system performance is optimised through maintenance scheduling. However, real world systems are usually very complex and therefore the list of methods applicable to such problems is rather short. It is possible to analyse a simplified structure of the real system, however, in this case there is a compromise in the accuracy of the results. Additionally, it is worth considering a set of different means through which system availability/reliability can be enhanced such as redundancy allocation, reliability allocation, repair/ preventive maintenance scheduling and/or decreasing the downtime of the system.

The development of a General System Design Optimisation Algorithm (GSDOA) was completed in order to create a technique which identifies the optimal system design with the optimal usage of available resources, such that the best performance possible is obtained. GSDOA has the potential to be applicable to a diverse range of real engineering systems. The benefit of using the general algorithm is the ability to identify an optimal or near optimal design case for any system considered rather than developing the system dependant approach. In the approach availability/reliability has been chosen to indicate the performance of a system. Improvement of system availability/reliability has been considered through the optimal allocation of redundancies and/or component reliabilities which may be subject to constraints and maintenance scheduling, if a repairable system is analysed. For quantitative comparison of different design options unreliability/unavailability measures have been utilised. Limitations for system cost, weight, volume and maintenance down time have been considered as possible constraints for the available resources.

The developed approach uses a combination of FTA and BDD analysis to introduce and evaluate the design proposals. Since FTA provides a schematic visual representation of the possible combinations of system conditions that could lead to its failure it is employed to represent all potential design configurations through their failure causes. The first major development was to devise patterns to enable transformation of an initial design fault tree into one for all possible designs so the optimisation approach could be applied. Moreover the fault tree modification patterns (FTMPs) were developed to represent any design alteration in the fault tree for any system under consideration. House events which are incorporated when applying the FTMPs developed enable the construction of a single fault tree representing causes of the system failure mode for all possible design alternatives. By setting certain house events to TRUE while the rest of them are set to FALSE the corresponding branches are switched on and off which results in the fault tree for a specific design alternative. Possible design alternatives for a considered system case are determined using the FTMPs which are chosen according to the list of design parameters (component selection and/or redundancy type and levels) given.

The conventional techniques for the quantitative analysis of fault trees can be computationally intensive and require the use of approximations, which inevitably leads to a loss of accuracy. The BDD technique is used as an alternative approach for performing the required analysis. Converting the fault tree, representing all possible design alternatives, into its BDD rather than converting individual design fault trees to their BDDs enhances the computational efficiency of quantitative failure analysis for any problem solved.

In the approach developed new possible optimal design solutions are generated and analysed using a simple GA. The GA provides a means of optimisation which is capable of coping with all requirements of the design problems considered. First of all the design parameters, i.e. decision variables used, are discrete numbers. The objective function of the optimisation problem formulated is not in a closed mathematical form and FTA and BDD analysis are used to evaluate its values. System design alterations are subject to a number of constraints which need to be considered during the analysis. Finally, the approach is not one system orientated which means the chosen technique needs to be easily adaptable for solving design optimisation problems for different systems.

A computer programme has been developed to automate the application of the GSDOA to solve system design optimisation problems. The approach has been assessed through the consideration of two systems of increasing complexity; namely the High Integrity Protection

System (HIPS) and the Fire Water Deluge System (FWDS) relating to an offshore platform. Through applying the approach the applicability to different system problems and scalability of the technique has been addressed. The application of the initial GA tested on the HIPS has highlighted potential areas of improvement. This resulted in the development of an approach with the improved GA which was applied to the FWDS. The improved GA has fitness scaling, an adaptive penalty methodology and a new population replacement strategy implemented. This in turn leads to the applicability of the method to complex engineering systems.

The development of the approach applicable to a range of different systems made it possible to adapt it to analyse multi-phased mission systems. Phased mission systems are important in various applications, e.g. military operations; however there is no demonstrated evidence in the literature for research about the design optimisation problems of such systems. The alterations made to the algorithm were with regards to the evaluation of system performance during each phase of the mission. In the Phased Mission System Design Optimisation Algorithm (PMSDOA) the initially-developed methodology of FTMPs has been amended to introduce house events and therefore design alternatives in the fault tree for each phase. Each phase fault tree represents only those possible design alternatives whose failure contributes to the failure of the phase in question. The GA implemented has been amended to include limitations on available resources and system performance characteristics throughout the whole missions as well as each individual phase. The initially developed computer code has also been amended to implement the corresponding alterations of the algorithm.

Three application examples of the new optimisation approach have been investigated. First a relatively simple unmanned aerial vehicle (UAV) design optimisation problem was solved. Next military vessel design optimisation problems of increasing complexity followed. The optimal designs were found corresponding to the global minimum failure probabilities of the missions within the limits of defined available resources (if such were considered) for all three cases. The application examples proved the scalability of the algorithm and its applicability to different systems.

The use of the optimisation approach was further developed adapting it to solve design optimisation problems for multi-phased mission systems considering more than one mission. It is common that one system is designed to perform a number of different missions, e.g. a military vessel. Therefore design optimisation problems of such systems are relevant, even though such problems have not been considered before. In this case the amendments to the

algorithm were made with regards to the analysis of system performance and design alternatives in different missions. In the Multiple Phased Mission System Design Optimisation Algorithm (MPMSDOA) the system unreliability for each mission analysed is evaluated individually using the methodology developed in PMSDOA. The initially used single objective GA has been replaced with the multi objective GA since a trade-off of system performance in different missions was required. The programme code was developed which automates the reliability design optimisation process for multiple multi-phased missions. All new developments were successfully implemented in the UAV design optimisation problem.

It is known that GAs require tuning of their parameters to achieve good performance results. The influence of different values of GA parameters on the performance of the developed algorithm has been analysed for each instance of the application example. The tendency has been noticed that the developed algorithm performs better, i.e. finds an optimal solution in fewer generations, if larger size populations, comprising of 50 or 75 chromosomes, are used. In some cases, there is a trade-off between mutation rate and crossover rate. The general tendency, however, is that a higher mutation rate and a higher crossover rate improve convergence of the algorithm. Thus a population of 50 chromosomes, a mutation rate equal to 0.01 and a crossover rate equal to 0.95 are the recommended default values to be used for solving optimisation problems.

9.2. CONCLUSIONS

The developed FTMPs enables all proposed design options to be modelled in a single fault tree by transforming an initial design fault tree into one for all possible designs. The FTMPs are developed to be applicable for any system under consideration.

The application of FTA and BDD analysis for the evaluation of the objective function, i.e. system unavailability/unreliability means that the developed algorithm can be applied to solve optimisation problems for complex engineering systems. Moreover, this methodology enables the development of a general optimisation approach adaptable to any system since the individual FT and BDD is constructed and analysed for the system under consideration.

The application examples proved the scalability of the developed optimisation approach. The algorithm is capable of solving reliability optimisation problems for industrial systems of different degrees of complexity.

The significant novelty of the developed optimisation methodology is its capability to analyse phased mission systems and provide their optimal design solutions. The algorithm is also applicable to solve design optimisation problems of the phased mission systems which are involved in more than one mission.

The employment of the MOGA which does not depend on the form of the objective functions and their evaluation methodology assures that the algorithm developed for multiple multi-phased missions is not the one-problem oriented approach. Furthermore, it provides a number of good solutions which are critical in the system design process.

The objectives of the research have been met resulting in the developed automated general GA based optimisation approaches for the construction of an optimal system design case with minimal unavailability/unreliability and optimal utilisation of available resources. The approaches are applicable to general systems, multi-phased mission systems and multiple multi-phased missions systems.

9.3. FUTURE WORK

Future work would involve application of the developed automated algorithms to a variety of different engineering systems. It is foreseen that analysis of more complicated problems will introduce additional computational intensity in terms of evaluating the objective function, which may incur a processing time issue. The issue of scalability may pose the most difficulty with an upper limit being needed on the number and complexity of each phase (due to processing consideration) when solving problems of phased mission systems. Therefore, future work should focus on improving the performance of the algorithm to minimise CPU time and improve the convergence rate.

The GA has proven itself to be a very flexible and useful technique for the optimisation of system designs. Hybrid optimisation techniques combining GA with heuristic algorithms, simulation annealing methods, ant colony optimisation or neural networks could be very promising optimisation techniques in this field. They have the ability to retain the advantages of GAs in robustness and feasibility but significantly improve their computational efficiency and searching ability in finding the global optimum. This could also enable analysis of larger/more complex systems. Another way to improve performance of the optimisation process is to handle constraints as additional objectives by transforming a single objective constrained optimisation problem into a multi-objective one. Multi-objective optimisation

techniques seem one of the most promising in solving constrained optimisation problems. In this case a MOGA or a hybrid MOGA could be implemented.

FTA and BDD analysis are very useful techniques for the assessment of system failure characteristics. However the results of the application examples have highlighted the need to improve the efficiency of the methodology for objective function evaluation. Incorporating the use of alternative analysis techniques, for example, the ones used in real time analysis could provide a solution for the efficiency problem. Furthermore, incorporating the use of alternative methods, such as Markov methods, could expand the applicability of the optimisation approaches to a more diverse range of systems, such as where inter-dependency between components exists.

There are a number of other aspects of the design optimisation approaches developed that could be improved. For example, it would be useful if the user could define the possible minimum value of a design variable instead of using the default value which is equal to 1. Moreover, the order FTMPs are implemented could be chosen automatically within the programme ensuring the optimal size and complexity of the resulting fault tree. The data provision could be made more user-friendly. It is also envisaged from the methods that there is scope to easily alter the range of constraints considered.

BIBLIOGRAPHY

1. Crosetti, P.A. Fault tree analysis with probability evaluation. *IEEE Transactions on Nuclear Science* 1971; 18(1): 465-471.
2. Barlow, R. E., Fussell, J. B., Singpurwalla, N.D. *et al. Reliability and fault tree analysis: theoretical and applied aspects of system reliability and safety assessment*. Philadelphia: Society for Industrial and Applied Mathematics; 1975.
3. Lee, C.Y. Representation of switching circuits by binary-decision programs. *The Bell System Technical Journal* 1959; 38: 985 – 999.
4. Akers, S.B. Binary Decision Diagrams. *IEEE Transactions on Computers* 1978; C-27(6): 509-516.
5. Bryant, R. E. Graph-based algorithms for Boolean function manipulation. *IEEE Transactions on Computers* 1986; C-35(8): 677-691.
6. Rauzy, A. New algorithms for fault trees analysis. *Reliability Engineering & System Safety* 1993; 40(3): 203-211.
7. Andrews, J.D., Moss T. R. *Reliability and Risk Assessment*. 2nd edn. London: Professional Engineering Pub; 2002.
8. Sinnamon, R. M., Andrews, J.D. New approaches to evaluating fault trees. *Reliability Engineering & System Safety* 1997; 58(2): 89-96, 1997.
9. Schneeweiss, W.G. *Boolean functions: with engineering applications and computer programs*. London: Springer-Verlag; 1988.
10. Sinnamon, R. M., Andrews, J.D. Improved efficiency in qualitative fault tree analysis. *Quality and Reliability Engineering International* 1997; 13(5): 293-298.
11. Murty, K.G. *Linear and combinatorial programming*. London: Wiley; 1976.
12. Vanderplaats, G. N. *Numerical optimization techniques for engineering design with applications*. London: McGraw-Hill; 1984.
13. Kuo, W., Prasad, V.R., Tillman F.A. *et al. Optimal reliability design: fundamentals and applications*. Cambridge: Cambridge University Press; 2000.
14. Ramirez-Marquez, J. New approaches for reliability design in multistate systems. *Handbook of Performability Engineering* 2008; 465-476.
15. Pardalos, P.M., Resende, M.G.C. *Handbook of applied optimization*. Oxford: University Press; 2002.
16. Joshi, M.C. *Optimization: theory and practice*. Harrow: Alpha Science International; 2004.
17. Dantzig, G. B., Thapa, M. N. *Linear programming 2. Theory and extensions*. London: Springer; 2003.

18. Kolesar, P.J. Linear programming and the reliability of multicomponent systems. *Naval Research Logistics Quarterly* 1967; 14(3): 317-327.
19. Hsieh, Y. A linear approximation for redundant reliability problems with multiple component choices. *Computers & Industrial Engineering* 2003; 44(1): 91-103.
20. Everett III, H. Generalized Lagrange multiplier method for solving problems of optimum allocation of resources. *Operations Research* 1963; 11(3): 399-417.
21. Hwang, C., Tillman, F. A. and Kuo, W. Reliability optimization by generalized Lagrangian-function and reduced-gradient methods. *IEEE Transactions on Reliability* 1979; 28(28): 316-319.
22. Li, D., Haimes, Y. Y. A decomposition method for optimization of large-system reliability. *IEEE Transactions on Reliability* 1992; 41(2): 183-188.
23. Tillman, F. A. *Optimization of systems reliability*. New York: Dekker; 1980.
24. Levitin, G. Genetic algorithms in reliability engineering. *Reliability Engineering & System Safety* 2006; 91(9): 975-976.
25. Diwekar, U. *Introduction to applied optimization*. Kluwer: Academic Publishers; 2003.
26. Yalaoui, A., Chatelet, E. and Chengbin C. A new dynamic programming method for reliability & redundancy allocation in a parallel-series system. *IEEE Transactions on Reliability* 2005; 54(2): 254-261.
27. Ng, K., Sancho, N. G. F. A hybrid dynamic programming/depth-first search algorithm, with an application to redundancy allocation. *IIE Transactions* 2001; 33 (12): 1047-1058.
28. Nakagawa, Y., Nakashima K. and Hattri, Y. Optimal reliability allocation by branch-and-bound technique. *IEEE Transactions on Reliability* 1978; 27(1): 31-38.
29. Sung, C.S., Cho, Y. K. Reliability optimization of a series system with multiple-choice and budget constraints. *European Journal of Operational Research* 2000; 127(1): 159-171.
30. Ha, C., Kuo, W. Reliability redundancy allocation: an improved realization for nonconvex nonlinear programming problems. *European Journal of Operational Research* 2006; 171(1): 24-38.
31. Misra, K. B. An algorithm to solve integer programming problems: an efficient tool for reliability design. *Microelectronics and Reliability* 1991; 31(2-3): 285-294.
32. Kuo, W., Prasad, V.R. An annotated overview of system-reliability optimization. *IEEE Transactions on Reliability* 2000; 49(2): 176-187.
33. Pirlot, M. General local search methods. *European Journal of Operational Research* 1996; 92(3): 493-511.
34. Reeves, C. R. *Modern heuristic techniques for combinatorial problems*. Oxford: Blackwell Scientific; 1995.
35. Cepin M. Optimization of safety equipment outages improves safety. *Reliability Engineering & System Safety* 2002; 77(1): 71-80.

36. Harunuzzaman, M., Aldemir, T. Optimization of standby safety system maintenance schedules in nuclear power plants. *Nuclear Technology* 1996; 113(3): 354-367.
37. Kim, H., Bae, C. and Park, D. Reliability-redundancy optimization using simulated annealing algorithms. *Journal of Quality in Maintenance Engineering* 2006; 12(4): 354-363.
38. Meta-Heuristics International Conference, *Meta-heuristics : theory & applications*. Boston: Kluwer Academic; 1996.
39. Kulturel-Konak, S., Smith, A.E. and Coit, D.W. Efficiently solving the redundancy allocation problem using tabu search. *IEE Transactions* 2003; 35(6): 515-526.
40. Ouzineb, M., Nourelfath, M., Gendreau, M. Tabu search for the redundancy allocation problem of homogenous series-parallel multi-state systems. *Reliability Engineering and System Safety*; 93(8): 1257-1272.
41. Eberhart, R., Kennedy, J. A new optimizer using particle swarm theory. *Micro Machine and Human Science, 1995. MHS '95, Proceedings of the Sixth International Symposium on Micro Machine and Human Science*; 1995.
42. Coelho, L. D. S. An efficient particle swarm approach for mixed-integer programming in reliability-redundancy optimization applications. *Reliability Engineering & System Safety* 2009; 94(4): 830-837.
43. Shi, Y., Eberhart, R. C. Empirical study of particle swarm optimization, *CEC 99. Proceedings of the 1999 Congress on Evolutionary Computation*; 3:1945-1950.
44. Holland, J. *Adaptation in natural and artificial systems*. Ann Arbor: University of Michigan Press; 1975.
45. Goldberg, D. E. *Genetic algorithms in search, optimization, and machine learning*. Reading: Addison-Wesley; 1989.
46. Coit, D.W., Smith, A.E. Reliability optimization of series-parallel systems using a genetic algorithm. *IEEE Transactions on Reliability* 1996; 45(2): 254-260.
47. Tavakkoli-Moghaddam, R., Safari, J., Sassani, F. Reliability optimization of series-parallel systems with a choice of redundancy strategies using a genetic algorithm. *Reliability Engineering & System Safety* 2008; 93(4): 550-556.
48. Yun, W. Y., Kim, J. W. Multi-level redundancy optimization in series systems. *Computers & Industrial Engineering* 2004; 46(2): 337-346.
49. Hsieh, Y., Chen, T., Bricker, D. L. Genetic algorithms for reliability design problems. *Microelectronics & Reliability* 1998; 38(10): 1599-1605.
50. Levitin, G., Lisnianski, A., Ben-Haim, B. *et al.* Redundancy optimization for series-parallel multi-state systems. *IEEE Transactions on Reliability* 1998; 47(2): 165-172.
51. Levitin, G. Multistate series-parallel system expansion-scheduling subject to availability constraints. *IEEE Transactions on Reliability* 2000; 49(1): 71-79.

52. Levitin, G. Redundancy optimization for multi-state system with fixed resource-requirements and unreliable sources. *IEEE Transactions on Reliability* 2001; 50(1): 52-59.
53. Munõz, A., Martorell, S., Serradell, V. Genetic algorithms in optimizing surveillance and maintenance of components. *Reliability Engineering & System Safety* 1997; 57(2): 107-120.
54. Lapa, C. M. F., Pereira, C. M. N. A., Mol, A. C. D. A. Maximization of a nuclear system availability through maintenance scheduling optimization using a genetic algorithm. *Nuclear Engineering and Design* 2000; 196(2): 219-231.
55. Lapa, C. M. F., Pereira, C. M. N. A., Frutuoso de Melo, P.F. Surveillance test policy optimization through genetic algorithms using non-periodic intervention frequencies and considering seasonal constraints. *Reliability Engineering & System Safety* 2003; 81(1): 103-109.
56. Lapa, C. M. F., Pereira, C. M. N. A., de Barros, M. P. A model for preventive maintenance planning by genetic algorithms based in cost and reliability. *Reliability Engineering & System Safety* 2006; 91(2): 233-240.
57. Marseguerra, M., Zio, E. Optimizing maintenance and repair policies via a combination of genetic algorithms and Monte Carlo simulation. *Reliability Engineering & System Safety* 2000; 68(1): 69-83.
58. Monga, A., Zuo, M.J. Optimal design of series-parallel systems considering maintenance and salvage value. *Computers & Industrial Engineering* 2001; 40(4): 323-337.
59. Dengiz, B., Altiparmak, F., Smith, A.E. Local search genetic algorithm for optimal design of reliable networks. *IEEE Transactions on Evolutionary Computation* 1997; 1(3): 179-188.
60. Ren, Y., Dugan, B.J. Design of reliable systems using static and dynamic fault trees. *IEEE Transactions on Reliability* 1998; 47(3), 234-244.
61. Andrews, J. D., Bartlett, L. M. Genetic algorithm optimization of a Firewater Deluge system, *Quality and Reliability Engineering International*. 19(1): 39-52, 2003.
62. Gen, M. *Genetic algorithms and engineering design*. New York: Wiley; 1997.
63. Beasley, D., Bull, D. R., Martin, R. R. An overview of genetic algorithms: part 1, fundamentals. *University Computing* 1993; 15(2): 58-69.
64. Gen, M., Yun, Y. Soft computing approach for reliability optimization: state-of-the-art survey. *Reliability Engineering & System Safety* 2006; 91(9), 1008-1026.
65. Vinod, G., Kushwaha, H. S., Verma, A. K. *et al.* Optimisation of ISI interval using genetic algorithms for risk informed in-service inspection. *Reliability Engineering & System Safety* 2004; 86(3): 307-316.
66. Arora, J. S. *Introduction to optimum design*. 2nd edn. Elsevier Academic Press; 2004.
67. Zalzal, A. M. S., Fleming, P. J. *Genetic algorithms in engineering systems*. London: Institution of Electrical Engineers; 1997.

68. Chambers, L. *The practical handbook of genetic algorithms*. 2nd edn. Boca Raton: Chapman & Hall/CRC; 2001.
69. Busacca, P.G., Marseguerra, M., Zio, E. Multiobjective optimization by genetic algorithms: application to safety systems. *Reliability Engineering & System Safety* 2001; 72(1): 59-74.
70. Marseguerra, M., Zio, E., Martorell, S. Basics of genetic algorithms optimization for RAMS applications. *Reliability Engineering & System Safety* 2006; 91(9): 977-991.
71. Martorell, S., Sánchez, A., Carlos, S. *et al.* Alternatives and challenges in optimizing industrial safety using genetic algorithms. *Reliability Engineering & System Safety* 2004; 86(1): 25-38.
72. R. E. Barlow, H. E. Lambert., *Reliability and fault tree analysis: theoretical and applied aspects of system reliability and safety assessment*. Philadelphia: Society for Industrial and Applied Mathematics; 1975.
73. Andrews, J. D., Pattison, R. L. Optimal safety system performance. *Proceedings of Annual Reliability and Maintainability Symposium* 1997; 76-83.
74. Remenyte-PreScott, R., *System failure modelling using binary decision diagrams, Thesis (Ph.D.)*. Loughborough University. Department of Aeronautical and Automotive Engineering; 2007.
75. *Proceedings of the Third International Conference on Genetic Algorithms, George Mason University, June 4-7, 1989*. San Mateo, Calif.: Morgan Kaufmann Publishers; 1989.
76. Bäck, T. *Evolutionary algorithms in theory and practice: evolution strategies, evolutionary programming, genetic algorithms*. Oxford: New York; 1996.
77. Pham, D., Karaboga, D. Genetic algorithms with variable mutation rates: application to fuzzy logic controller design. *Proceedings of the Institution of Mechanical Engineers, Part I: Journal of Systems and Control Engineering* 1997; 211(2): 157-167.
78. Coello Coello, C.A. *A survey of constraint handling techniques used with evolutionary algorithms*. Rep. No. Technical Report Lania-RI-99-04. Veracruz, Mexico: Laboratorio Nacional de Informática Avanzada; 1999.
79. Coello Coello, C. A. Theoretical and numerical constraint-handling techniques used with evolutionary algorithms: a survey of the state of the art. *Computer Methods in Applied Mechanics and Engineering* 2002; 191(11-12): 1245-1287.
80. Coit, D.W., Smith, A.E., Tate, D. M. Adaptive penalty methods for genetic optimization of constrained combinatorial problems. *INFORMS Journal of Computing* 1996; 8(2): 173-182.
81. Susova, G. M., Petrov, A.N. Markov model-based reliability and safety evaluation for aircraft maintenance-system optimization. *Proceedings of Annual Reliability and Maintainability Symposium* 1997; 1997. 29-36.
82. Vaurio, J. K. Fault tree analysis of phased mission systems with repairable and non-repairable components. *Reliability Engineering & System Safety* 2001; 74(2): 169-180.

83. Alam, M., Al-Saggaf, U. Quantitative reliability evaluation of repairable phased-mission systems using Markov approach. *IEEE Transactions on Reliability* 1986; 35(5): 498-503.
84. Smotherman, M., Zemoudeh, K. A non-homogeneous Markov model for phased-mission reliability analysis. *IEEE Transactions on Reliability* 1989; 38(5): 585-590.
85. Esary, J.D., Ziehms, H. Reliability of phased missions. *Proceedings of Conference on Reliability and Fault-Tree Analysis, 1974*. Philadelphia: Society for Industrial and Applied Mathematics; 1975, 213-236.
86. Somani, A. K., Trivedi, K. S. Phased-mission system analysis using Boolean algebraic methods. *Proceedings of the 1994 ACM SIGMETRICS Conference on Measurement and Modelling of Computer Systems*. Nashville, United States; 1994, 98-107.
87. Ma, Y., Trivedi, K.S. An algorithm for reliability analysis of phased-mission systems. *Reliability Engineering & System Safety* 1999; 66(2): 157-170.
88. La Band, R., Andrews, J.D. Phased mission modelling using fault tree analysis. *Proceedings of the Institution of Mechanical Engineers, Part E: Journal of Process Mechanical Engineering* 2004; 218(2): 83-91.
89. Prescott, D. R., Remenyte-Prescott, R., Reed, S. *et al.* A reliability analysis method using binary decision diagrams in phased mission planning. *Proceedings of the Institution of Mechanical Engineers, Part O: Journal of Risk and Reliability* 2009; 223(2), 133-143.
90. Astapenko, D., Bartlett, L.M. Phased mission failure minimisation using optimal system design. *Proceedings of the 26th International System Safety Conference, ISSC, 25-29 Aug 2008*. Vancouver, Canada.
91. Astapenko, D., Bartlett, L.M. Phased mission system design optimisation using genetic algorithms. *International Journal of Performability Engineering* 2009; 5(4), 313-324.
92. Astapenko, D., Bartlett, L.M. System design optimisation involving phased missions. *Proceedings of the European Safety and Reliability Conference, ESREL-2008, 22-25 Sept 2008*. Valencia, Spain: CRC Press/Balkema; 2008, 2021-2027.
93. Astapenko, D., Bartlett, L.M. System design optimisation involving phased missions. *International Journal of Reliability and Safety (IJRS)*, 2009; 3(4): 331-344. DOI: 10.1504/IJRS.2009.028580.
94. Deb, K. *Multi-objective optimization using evolutionary algorithms*. Chichester: Wiley; 2001.
95. Konak, A.; Coit, D. W.; Smith, A. E. Multi-objective optimization using genetic algorithms: a tutorial. *Reliability Engineering & System Safety* 2006; 91(9): 992-1007.
96. Coello Coello, C.A.; Van Veldhuizen, D.A.; Lamont, G.B. *Evolutionary algorithms for solving multi-objective problems*. 2nd edn. New York; 2002.
97. Suman, B. Study of self-stopping PDMOSA and performance measure in multiobjective optimization. *Computers & Chemical Engineering* 2005; 29(5): 1131-1147.
98. Coello, C. A. An updated survey of GA-based multiobjective optimization techniques. *ACM Computing Surveys* 2000; 32(2): 109-143.

99. Fonseca, C.M.; Fleming, P.J. An overview of evolutionary algorithms in multiobjective optimization. *Evolutionary Computation* 1995; 3: 1-16.
100. Van Veldhuizen, D., Lamont, G. *Multiobjective evolutionary algorithm research: a history and analysis*. Air Force Inst. Technol., Dayton, OH, Tech. Rep. TR-98-03, 1998.
101. Zitzler, E., Thiele, L. Multiobjective evolutionary algorithms: a comparative case study and the strength Pareto approach. *IEEE Transactions on Evolutionary Computation* 1999; 3(4): 257-271.
102. Srinivas, N.; Deb, K. Multiobjective optimization using nondominated sorting in genetic algorithms. *Evolutionary Computation* 1994; 2(3): 221-248.
103. Fonseca, C. M., Fleming, P. J. Multiobjective optimization and multiple constraint handling with evolutionary algorithms I: a unified formulation. *IEEE Transactions on Systems, Man and Cybernetics Part A: Systems and Humans* 1998; 28(1): 26-37.
104. Jimenez, F.; Gomez-Skarmeta, A.F.; Sanchez, G. *et al.* An evolutionary algorithm for constrained multi-objective optimization. *CEC '02. Proceedings of the 2002 Congress on Evolutionary Computation* 2002; 1133-1138.
105. Deb, K.; Pratap, A.; Agarwal, S. *et al.* A fast and elitist multiobjective genetic algorithm: NSGA-II. *IEEE Transactions on Evolutionary Computation* 2002; 6(2): 182-197.
106. Fonseca, C.M.; Fleming, P.J. On the performance assessment and comparison of stochastic multiobjective optimizers. *Proceedings of the 4th International Conference on Parallel Problem Solving from Nature*, 1996.584-593.
107. Zitzler, E.; Deb, K.; Thiele, L., Comparison of multiobjective evolutionary algorithms: empirical results. *Evolutionary Computation* 2000; 8:173-195.
108. Murata, T., Ishibuchi, H. MOGA: multi-objective genetic algorithms. *IEEE International Conference on Evolutionary Computation* 1995; 289-294.
109. Marseguerra, M.; Zio, E.; Martorell, S. Basics of genetic algorithms optimization for RAMS applications. *Reliability Engineering & System Safety* 2006; 91(9): 977-991.

PUBLICATIONS

Conference papers

1. Astapenko, D., Bartlett, L.M. Phased mission failure minimisation using optimal system design. *Proceedings of the 26th International System Safety Conference, ISSC, 25-29 Aug 2008*. Vancouver, Canada.
2. Astapenko, D., Bartlett, L.M. System design optimisation involving phased missions. *Proceedings of the European Safety and Reliability Conference, ESREL-2008, 22-25 Sept 2008*. Valencia, Spain: CRC Press/Balkema; 2008, 2021-2027.
3. Astapenko D., Bartlett, L.M. System failure minimisation using automated design optimisation. *Proceedings of the 18th AR²TS (Advances in Risk & Reliability Technology Symposium), April 2009*. Loughborough University, UK, 347-359.

Journal papers

1. Astapenko, D., Bartlett, L.M. Phased mission system design optimisation using genetic algorithms. *International Journal of Performability Engineering* 2009; 5(4), 313-324.
2. Astapenko, D., Bartlett, L.M. System design optimisation involving phased missions. *International Journal of Reliability and Safety (IJRS)*, 2009; 3(4): 331-344. DOI: 10.1504/IJRS.2009.028580.

APPENDIX 1

The correct and sufficient data is critical for finding a solution for the optimisation problem to be solved. This section provides guidelines for data presentation in the required format. Each subsection is denoted to a set of data stored in a single file. It provides a detailed explanation of the file content based on a specific example. It also specifies the main rules to be followed to assure the data is presented in a correct format.

A.1. FILE – *fts.txt*

The file *_fts.txt* stores the fault tree structure of the initial system design in the text format. It is required that names (codes) of fault tree gates and basic events contain only digits. Therefore code-names should be introduced to basic events and/or gates whose names do not meet this requirement. It is also important that each basic event and gate code-name is unique, i.e. two or more fault tree components cannot have the same code-name.

Consider the fault tree example in *Figure A.1*. In the given example, gate codes correspond to the given gate numbers. The basic event codes to be used are provided in *Table A.1*.

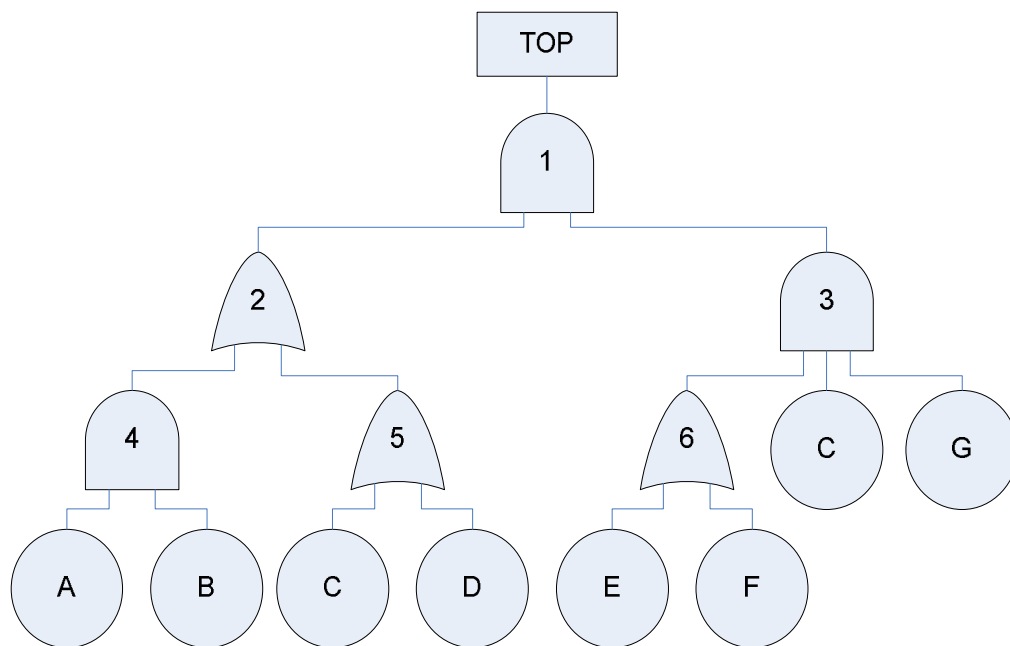


Figure A.1 Example Fault Tree

Table A.1 Number Codes for Basic Events

<i>Basic Event</i>	<i>Code</i>
A	1
B	2
C	3
D	4
E	5
F	6
G	7

Each row in the file contains data of a single gate and it starts with the gate code. Note that gates can be listed in any order in the file. The only requirement is that the top gate data is written in the first row of the file.

After a gate code the gate type follows identified with a key word 'OR' or 'AND'. The next two numbers identify the total number of input gates and events that the considered gate has. It is required that the first figure denotes the total number of input gates. Finally, the code-names of input gates and input events are listed. Each gate code-name comprises an identification symbol 'G:' and its code. Accordingly an input event code-name comprises of the symbol "E:" and its code.

Consider the example fault tree in *Figure A.1*. The fault tree has six gates, thus six data rows will be needed in order to present its structure in a text format. The top gate is an AND gate and has two input gates coded as 2 and 3. Thus the first row of the fault tree data file would be:

1 AND 2 0 G:2 G:3

Gate 2 is an OR gate and also has two input gates. In the data file it would be written as follows:

2 OR 2 0 G:4 G:5

Gate 3 is an AND gate. It has one input gate and two inputs events. In this case the data in the text format would be provided as follows:

3 AND 1 2 G:6 E:3 E:7

The rest of the gates have two input basic events each. For example, the data row for gate 4 would be:

4 AND 0 2 E:1 E:2

To summarise, there are three main requirements for the presentation of the fault tree structure. Names of basic events and gates contain only digits. The top gate data has to be written in the first row of the file. First the total number of input gates is to be written followed by the total number of input events. Accordingly foremost input gates are listed and then the list of input events is given.

A.2. FILE *_var.txt*

The file stores the data required to implement changes in the system design regarding the structural design variables introduced for the problem solved. In the file each row represents a fault tree event (gate or basic event) chosen to be replaced and parameters of a FTMP, associated with a particular design variable or a set of variables, which defines the fault tree structure to be incorporated instead. Each row is started with the fault tree event identification. For this reason, either symbol 'G' (gate) or 'E' (basic event) is used followed by the event code. Next values are written for the FTMP parameters. The values for the parameters are written in the following order. First mn value is given, then mt is defined and finally an mk value is written (see Section 4.2.2).

Consider the previous example fault tree in Figure A.1. Two FTMPs are introduced, *Pattern 1* ($mn = 2$, $mt = 1$, $mk = 1$) and *Pattern 4* ($mn = 3$, $mt = 2$, $mk = 1$). The events chosen to be replaced are gate 6 and basic event 4 respectively. The data file for the chosen modifications would consist of two rows:

G 6 2 1 1

E 4 3 2 1

A.3. FILE *_bse.aqd*

Basic event failure data is stored in the file with the ending *_bse.aqd*. Every row in the data file corresponds to a single basic event which is coded with a unique code-name. As mentioned earlier (Section A.1) it is required to use only digits for codes of basic events. It is also essential to provide new basic events, which occur after implementation of modification in the fault tree, and their correct code-names following the rules introduced in Section 4.3.5.2.

The programme is developed to utilise three different types of component failure data. Depending on a component failure type, a number of numerical parameters used to find the basic event probability will vary. An identification letter written next to the basic event code is used to determine the model to be used. Letter 'd' identifies a dormant failure probability. In this case failure rate, mean time to repair and maintenance test interval values are provided for the basic event. Letter *w*, which identifies that component failure times are distributed under the Weibull distribution, is followed by values of the distribution parameters β and η , and a maintenance test interval value. Note that in both cases, in order to find the component unavailability value the maintenance test interval value is required. However, if maintenance test intervals are design variables, then a digit 1 is to be written as a maintenance test interval value. Finally, in the data file letter *f* is to be used if a component unavailability value is given which will be written after the letter.

The last row containing a key word 'ENDOFDATA' identifies the end of list of basic events.

Consider the fault tree example given in *Figure A.1*. All basic events and their failure data are presented in *Figure A.2*.

1	d	0.000007	0.000004	30
2	d	0.000014	0.000004	30
3	d	0.000021	0.000004	30
4_1_1	w 2	14035		40
4_2_1	w 2	14035		40
4_3_1	w 2	14035		40
4_1_2	w 1	12075		55
4_2_2	w 1	12075		55
4_3_2	w 1	12075		55
5_1_1	w 4	17050		30
5_2_1	w 4	17050		30
6_1_1	f	0.00001		
6_2_1	f	0.00001		
7	f	0.0000005		
ENDOFDATA				

Figure A.2 File *_bse.aqd*

In the file fault tree basic events are named using their code-names given in *Table A.1*. Basic events 1, 2 and 3 are given failure rate, mean time to repair and maintenance test interval values. Failure rates for basic events 4 and 5 have a Weibull distribution and for events 6 and 7 actual failure probabilities are defined. Assuming that the FTMP from *Section A.2* are being implemented the list of basic events increases as new basic events are introduced.

A.4. FILE *_cost_cst.txt*

If the system design improvement is restricted by its cost, then a data file with the ending *cost_cst.txt* is required in order to solve the optimisation problem. The file is used to store values for design cost limits as well as each component cost.

In the first row of the file following the key word ‘COST’ minimum and maximum design cost limits are to be written, whereby the first number represents the minimum value. For evaluation of maintenance costs the total number of maintenance interval time units covering an examined time period (one year) is provided after the keyword ‘UNITS’. If maintenance costs are not considered for a problem solved, value 0 should be written.

After the keyword “COMPONENTS” component cost data is provided. Each subsequent row corresponds to a single basic event. As in the file with the ending *_bse.aqd*, a row starts with the unique event code. Then component costs are listed. First component design cost is to be written. Next, maintenance costs are listed in the following order: a single maintenance test cost, component maintenance and repair costs. If any of the mentioned costs is not considered a zero value has to be written.

As an example, the fault tree in *Figure A.1* is considered that is modified using FTMPs as described in *Section A.2*. The optimal design cost cannot exceed 20 units. *Figure A.3* demonstrates the data file contents where maintenance costs are considered for only basic events 1, 2 and 3.

A.5. FILE *_mdt_cst.txt*

The data required to implement systems down time constraints is stored in the file *_mdt_cst.txt*. Minimum and maximum range limits for possible values of maintenance down time are listed after the keyword ‘MDT’ in the first row. The second row of the data file starts with the key word ‘UNIT’ which identifies that the numerical value followed after is a total

number of time units (used for maintenance time intervals) in the examined time period (U_T as used in *Formula 4.14*). The third row with the key word 'COMPONENTS' indicates that the consequent rows of the data file contain basic events data. In each row a basic event code is followed by test time and time interval values between maintenance activities. These are used to calculate maintenance down time for each component.

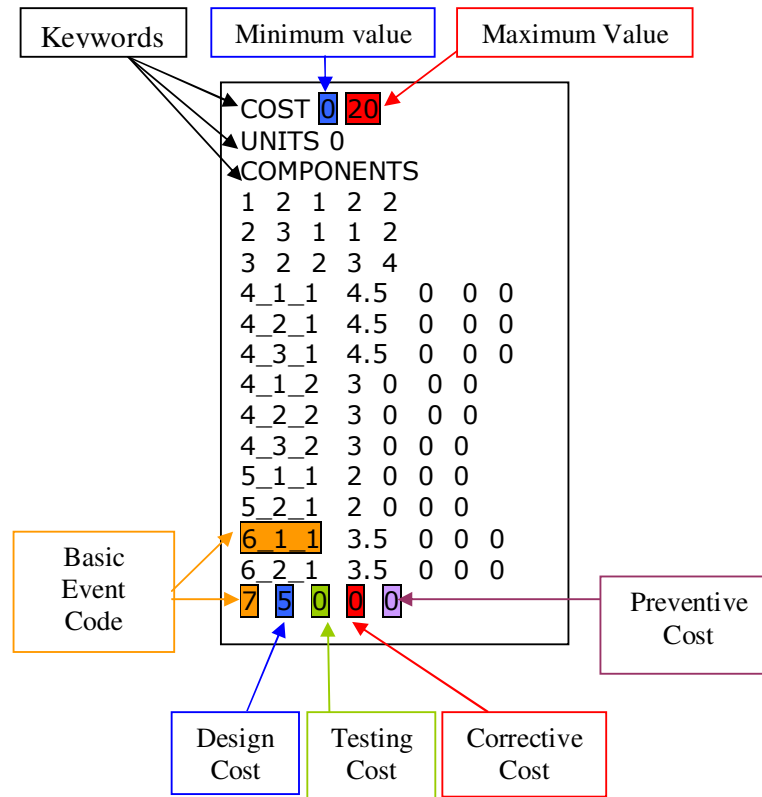


Figure A.3 File *_cost_cst.txt*

If maintenance intervals are not known and defined as design variables, the maintenance interval value for each basic event is set to 1 in the data file.

Figure A.4 represents the data file content prepared for the same fault tree example as in the previous section (*Section A.4*). Maintenance intervals are evaluated using days as time units. It means that the total number of time units in one year is equal to 365. Each component test time is measured in hours therefore the same time units, i.e. hours, are to be used for maintenance down time assessment. Maintenance down time limits are set to 10 and 15 hours a year.

MDT 10 15				
UNIT 365				
COMPONENTS				
1	3	30		
2	1	30		
3	2	30		
4_1_1	1	40		
4_2_1	1	40		
4_3_1	1	40		
4_1_2	1.5	55		
4_2_2	1.5	55		
4_3_2	1.5	55		
5_1_1	0.5	30		
5_2_1	0.5	30		
6_1_1	1.2	25		
6_2_1	1.2	25		
7	1	30		

Figure A.4 File *_mdt_cst.txt*

A.6. FILES *_volume_cst.txt* AND *_weight_cst.txt*

The structures of data files *_volume_cst.txt* and *_weight_cst.txt*, which are used to store data for calculation of volume and weight constraints, are identical. The first row has minimum and maximum limit values of system volume/ weight identified with the key word 'VOLUME'/'WEIGHT'. In either file the second row has the key word 'COMPONENTS'. The consequent rows start with a basic event code followed by the corresponding component volume/ weight.

A.7. FILE *_theta.txt*

If time intervals between maintenance activities are considered as design variables an additional data file with the ending *_theta.txt* is required to be produced. The data in the file is divided into two parts. Different system components can undergo maintenance at different time intervals, therefore the range limits for time intervals between anticipated different maintenance activities are provided first. It is implemented by storing each interval identification number and possible minimum and maximum limit values.

The second part of the file is separated from the first one by the key word “COMPONENTS”. It is used to store maintenance interval data of system components. Each row starts with a maintenance interval identification number as introduced in the first part of the file. After the interval identification number a basic event code follows which represents a system component that has been assigned to undergo maintenance at these particular intervals. It is common, that a number of components undergo maintenance at the same time, therefore a maintenance interval identification number needs to be listed as many times as there are components maintained at these time intervals. In the same way, the remaining specified maintenance intervals and associated basic events are listed. Note that this file does not provide numerical values of maintenance intervals, but it defines links between each basic event and its associated maintenance interval.

To illustrate the file content an example introduced in *Section A.1* and analysed through out the appendix is analysed. Consider the system components 1, 2, 3 and 4 (as well as components 4_1_1, 4_1_2, etc.) undergo maintenance at the same time and it can be performed within the range of 3 to 5 months. The remaining components 5 (including 5_1_1 and 5_2_1), 6 (including 6_1_1 and 6_2_1) and 7 can be maintained as often as 6 months but at least once a year. In consistence with the earlier given *_mdt_cst.txt* file example, range limits of maintenance intervals should be provided in the same time units, i.e. days. For example, 3 month will be equivalent to 90 days and 5 month will be equal to 150 days. The data file is shown in *Figure A.5*.

A.8. FILE *_gav.txt*

GA parameters such as population size, crossover rate, mutation rate and number of generations have to be defined by the user in the data file *_gav.txt*. The example of the file content is shown in *Figure A.6*. The keywords in each file line are indispensable because they are used to identify a value of a certain parameter used by the GA.

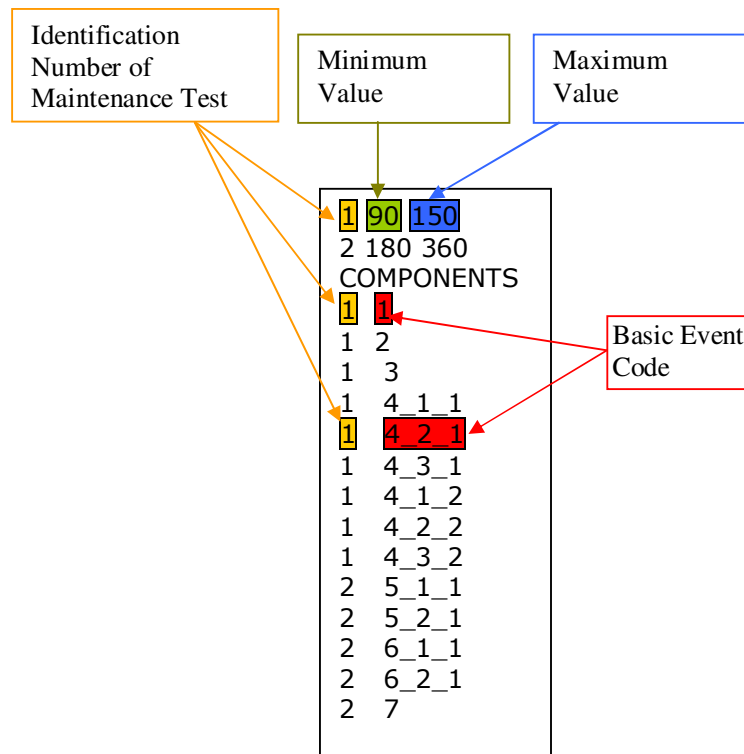


Figure A.5 File_theta.txt

```

population 10
crossover 0.75
mutation 0.002
generations 100
    
```

Figure A.6. File_gav.txt

APPENDIX 2

A.2.1. DATA FOR QUANTITATIVE FWDS FAILURE ANALYSIS

Table A.2.1. Distribution Network Failure Events

<i>Notation</i>	<i>Event Description</i>
FSU	Failure of pump selector unit to initiate start of the standby pump in sequence, on detection of failure of duty pump/pump to restore ringmain pressure.
OE	Designated duty pump/pump inadvertently left in a mode other than auto start at the end of the test.
PBF	Manual push button in the control room failing to initiate pump start when pressed.
PT (type 1)	Failure of ringmain low pressure sensor to indicate low ringmain pressure.

Table A.2.2. Distribution Network Failure Basic Events

<i>Notation</i>	λ	τ	H_T	N_S	C_S	C_I	Q
FSU	8e-6	2.4e-5	1	1	200	2000	
PT	7e-6	4e-6	1	2	100	500	
OE							0.01
PBF							0.01

Table A.2.3. Firewater System Failure Basic Events

<i>Notation</i>	<i>Event Description</i>
E_FB/D_FB	The pump, which includes seawater filter, is blocked by debris (electrical and diesel driven).
E_IVB/D_IVB	The firewater pump line isolation valve being blocked (electrical and diesel driven).
E_IVC/D_IVC	The firewater pump line isolation valve closed (electrical and diesel driven).
E_PRVO/D_PRVO	Pressure relief valve on header from pump to ringmain fails open (electrical and diesel driven).
E_SVO/D_SVO	The flow control valve fails to open on demand. (It is used to dump excess flow from pumps to ringmain) (electrical and diesel driven).
E_CVB/D_CVB	Check valve on header between the pump and ringmain blocked (electrical and diesel driven).
D_DVO/E_DVO	Line discharge valve to sea fails open (electrical and diesel driven).
EM/DM	Maintenance is being carried out on an firewater pump (electrical and diesel driven).
D_TIVB	Diesel tank isolation valve blocked.
D_TIVC	Diesel tank isolation valve closed.
D_LAF	Diesel tank level switch fails to signal low level to control room.
D_OAF	Operator fails to notice firewater diesel tank low level alarm.
D_EF	Firewater diesel engine fails.
DPF	Failure of firewater diesel pump.
E_ESF	Failure of electric supply to the electric driven pumps.
E_ESF	Failure of electric supply to the electric driven pumps.
E_MF	Global motor failure.
EPF	Failure of firewater electric pump.

Table A.2.4. Firewater System Failure Basic Events

<i>Notation</i>	λ	τ	H_T	N_S	C_S	C_I	β	η	Q
E_FB/D_FB	2.8e-5	1.2e-	2	4	150	100			
E_IVB/D_IVB	1.8e-5	1.8e-5	2	2	300	400			
E_PRVO/D_PRVO	1.2e-5	1.8e-5	2	2	300	500			
E_SVO/D_SVO	1.8e-5	2.4e-5	2	3	300	800			
E_CVB/D_CVB	1.8e-5	1.8e-5	2	2	300	400			
E_IVC/D_IVC									0.01
D_DVO/E_DVO									0.01
EM/DM									0.04
D_TIVC									0.01
D_OAF									0.01
D_EF									0.00128
D_TIVB	3e-5	8e-6	2	2	300	400			
D_LAF	3e-5	6e-6	2	2	200	200			
DPF			2	1	1000	2900	2	14035	
EPF			2	1	1000	3000	2	16667	
E_ESF	5e-5	2e-6	2			1000			
E_MF									4.5e-05

Table A.2.5. Basic Events of AFFF Supply System Failure

<i>Notation</i>	<i>Event Description</i>
AE_FB/AD_FB	The pump, which includes filter, is blocked by debris (electrical and diesel driven).
AE_SIVB/AD_SIVB	The AFFF pump line suction isolation valve being blocked (electrical and diesel driven).
AE_SIVC/AD_SIVC	The AFFF pump line suction isolation valve closed (electrical and diesel driven).
AE_PRVO/AD_PRVO	Pressure relief valve on header from pump to ringmain fails open (electrical and diesel driven).
AE_SVO/AD_SVO	The flow control valve fails open on demand. (It is used to dump excess flow from pumps to ringmain) (electrical and diesel driven).
AE_CVB/AD_CVB	Check valve on header between the pump and ringmain blocked (electrical and diesel driven).
AE_DIVB/AD_DIVB	The AFFF pump line discharge isolation valve being blocked (electrical and diesel driven).
AE_DIVC/AD_DIVC	The AFFF pump line discharge isolation valve closed (electrical and diesel driven).
AE/DM	Maintenance is being carried out on an AFFF pump (electrical and diesel driven).
AD_ATIVB	AFFF diesel tank isolation valve blocked.
AD_ATIVC	AFFF diesel tank isolation valve closed.
AD_LAF	Diesel tank level switch fails to signal low level to control room.
AD_OAF	Operator fails to notice AFFF diesel tank low level alarm.
AD_EF	Diesel global engine failure.
ADPF	Failure of AFFF diesel pump.
AE_ESF	Failure of electric supply to the electric driven AFFF pumps.
ATIVB	AFFF tank isolation valve blocked.
ATIVC	AFFF tank isolation valve closed.
AEPF	Failure of AFFF electric pump.

Table A.2.6. Data of AFFF System Failure Basic Events

<i>Notation</i>	λ	τ	H_T	N_S	C_S	C_I	β	η	Q
AE_FB/AD_FB	2.8e-5	1.2e-5	2	4	150	100			
AE_SIVB/AD_SIVB	1.8e-5	1.8e-5	2	2	300	400			
AE_PRVO/AD_PRVO	1.2e-5	1.8e-5	2	2	300	500			
AE_SVO/AD_SVO	1.8e-5	2.4e-5	2	3	300	800			
AE_CVB/AD_CVB	2.5e-5	1.8e-5	2	2	300	500			
AE_DIVB/AD_DIVB	1.8e-5	1.8e-5	2	2	300	400			
AE_SIVC/AD_SIVC									0.01
AE_DIVC/AD_DIVC									0.01
AE/DM									0.04
AD_ATIVC									0.01
AD_OAF									0.01
AD_EF									0.00128
AD_ATIVB	3e-5	8e-6	2	2	300	400			
AD_LAF	3e-5	6e-6	2	2	200	200			
ADPF			2	1	800	1450	2	14035	
AEPF			2	1	800	1500	2	16667	
AE_ESF	5e-5	2e-6	2			1000			
ATIVB	3e-5	8e-6	2	2	300	400			
ATIVC									0.01

Table A.2.7. Deluge System Failure Basic Events

<i>Notation</i>	<i>Event Description</i>
SI/WSI	Failure of MFGP to select and send a close signal to the solenoid valve correctly in AFFF and water Deluge Skid accordingly.
MRM/WMRM	Manual release mechanism fails to dump instrument air.
SV1/SV2/WSV1/WSV2	Solenoid activated valve fails to dump instrument air on receipt of the signal from the MFGP in AFFF and water Deluge Skid accordingly (there are 2 solenoid valves in each deluge skid).
WBS	Strainer, located downstream of the water deluge valve, blocked.
WNB	Deluge nozzle on the waterspray system blocked.
WIVB1/WIVB2	Locked open butterfly valve blocked (one upstream and one downstream of the water deluge valve).
WHE1/WHE2	Operator leaves the normally locked open butterfly valve in the shut position (one upstream and one downstream of the water deluge valve).
WV	Water deluge valve fails to open.
WVRF (old type)	Valmatic relief valve sticks closed on activation.
AHE	AFFF isolation valve left closed.
AIVB	Normally locked open butterfly valve on the AFFF distribution line blocked (only one isolation valve on AFFF line).
AINF	The foam supply into the firewater distribution line is blocked by the inductor nozzle.
ACVB	The check valve in the AFFF injection line is blocked.
AV (type 1)	AFFF deluge valve fails to open on demand.

Table A.2.8. Data of Basic Events for Deluge System Failure

<i>Notation</i>	λ	τ	H_T	N_S	C_S	C_I	Q
SI/WSI	2e-7	6e-6					
MRM/WMRM	1e-5	1.2e-5	2	1	300	600	
SV1/WSV1	3e-6	1.2e-5	2	2	300	400	

<i>Notation</i>	λ	τ	H_T	N_S	C_S	C_I	Q
SV2/ WSV2	2e-5	1.2e-5	2	2	300	250	
WBS	2.8e-5	1.2e-5	2	4	75	100	
WNB(old type)	3e-5	1.2e-5	2	3	300	1000	
WIVB1/WIVB2	1.8e-6	1.8e-6	2	2	300	400	
WHE1/WHE2							0.01
WV (type 1)	4e-5	1.8e-5	2	2	200	400	
WVRF (old type)	5e-6	1.2e-5	2	1	300	600	
AHE							0.01
AIVB	1.8e-5	1.8e-5	2	2	300	400	
AINF (old type)	3e-5	1.2e-5	2	3	300	1000	
ACVB	2.5e-5	1.8e-5	2	2	300	600	
AV (type 1)	4e-5	1.8e-5	2	2	150	300	

Table A.2.9. Data of New System Components

<i>Associated Design Variable</i>	<i>Basic Event Notation</i>	λ	τ	β	η	H_T	N_S	C_S	C_I
t_{PT}	PT (type 2)	1.4e-5	4e-6			1	2	100	200
t_{PT}	PT (type 3)	2.1e-5	4e-6			1	2	100	100
t_{1FE}, t_{2FE}	EPF (50% capacity) (type 1)			1.5	22857	2	2	900	1800
t_{1FE}, t_{2FE}	EPF (50% capacity) (type 2)			1.5	26667	2	2	900	2000
t_{1FE}, t_{2FE}	EPF (33 1/3% capacity) (type 1)			1.5	32000	2	2	800	1200
t_{1FE}, t_{2FE}	EPF (33 1/3% capacity) (type 2)			1.5	40000	2	2	800	1400
t_{1FD}, t_{2FD}	DPF (50% capacity) (type 1)			1.5	20000	2	2	900	1500
t_{1FD}, t_{2FD}	DPF (50% capacity) (type 2)			1.5	22857	2	2	900	1800
t_{1FD}, t_{2FD}	DPF (33 1/3% capacity) (type 1)			1.5	28571	2	2	800	1000
t_{1FD}, t_{2FD}	DPF (33 1/3% capacity) (type 2)			1.5	33333	2	2	800	1100
t_{AE}	AEPF (50% capacity)			1.5	22857	2	2	600	900
t_{AD}	APFD (50% capacity)			1.5	20000	2	2	600	750
t_{WD}	WV (type 2)	3.5e-5	1.8e-5			2	2	200	500
t_{WD}	WV (type 3)	2.8e-5	1.8e-5			2	2	200	600
t_{AD}	AV (type 2)	3.5e-5	1.8e-5			2	2	150	400
t_{AD}	AV (type 3)	2.8e-5	1.8e-5			2	2	150	500
t_{DN}	WNB(new type)	5e-6	1.2e-5			2	3	300	3000
t_{IN}	AINF (new type)	5e-6	1.2e-5			2	3	300	3000
t_{VR}	WVRF (new type)	2e-6	1.2e-5			2	1	300	900

A.2.2. ANALYSIS OF THE IMPROVED *GSDOP* PERFORMANCE SOLVING THE FWDS DESIGN OPTIMISATION PROBLEM

Table A.2.10. Minimal Unavailability Values when Population Size is Equal to 10

Population size = 10				
		Crossover Rate		
		<i>0.75</i>	<i>0.8</i>	<i>0.95</i>
Mutation Rate	<i>0.001</i>	0.1113	0.0977	0.0983
	<i>0.005</i>	0.1088	0.0960	0.0953
	<i>0.01</i>	0.1048	0.0947	0.0942

Table A.2.11. Minimal Unavailability Values when Population Size is Equal to 30

Population size = 30				
		Crossover Rate		
		<i>0.75</i>	<i>0.8</i>	<i>0.95</i>
Mutation Rate	<i>0.001</i>	0.1076	0.0950	0.0948
	<i>0.005</i>	0.0937	0.0939	0.0936
	<i>0.01</i>	0.0934	0.0934	0.0934

Table A.2.12. Minimal Unavailability Values when Population Size is Equal to 50

Population size = 50				
		Crossover Rate		
		<i>0.75</i>	<i>0.8</i>	<i>0.95</i>
Mutation Rate	<i>0.001</i>	0.0958	0.0956	0.0949
	<i>0.005</i>	0.0935	0.0938	0.0933
	<i>0.01</i>	0.0933	0.0928	0.0926

Table A.2.13. Minimal Unavailability Values when Mutation Rate is Equal to 0.001

Mutation Rate = 0.001				
		Crossover Rate		
		<i>0.75</i>	<i>0.8</i>	<i>0.95</i>
Population Size	<i>10</i>	0.1113	0.0977	0.0983
	<i>30</i>	0.1076	0.0950	0.0948
	<i>50</i>	0.0958	0.0956	0.0949

Table A.2.14. Minimal Unavailability Values when Mutation Rate is Equal to 0.005

Mutation Rate = 0.005				
		Crossover Rate		
		<i>0.75</i>	<i>0.8</i>	<i>0.95</i>
Population Size	<i>10</i>	0.1088	0.0960	0.0953
	<i>30</i>	0.0937	0.0939	0.0936
	<i>50</i>	0.0935	0.0938	0.0933

Table A.2.15. Minimal Unavailability Values when Mutation Rate is Equal to 0.01

Mutation Rate = 0.01				
		Crossover Rate		
		<i>0.75</i>	<i>0.8</i>	<i>0.95</i>
Population Size	<i>10</i>	0.1048	0.0947	0.0942
	<i>30</i>	0.0934	0.0934	0.0934
	<i>50</i>	0.0933	0.0928	0.0926

Table A.2.16. Minimal Unavailability Values when Crossover Rate is Equal to 0.75

Crossover Rate = 0.75				
		Mutation Rate		
		<i>0.001</i>	<i>0.005</i>	<i>0.01</i>
Population Size	<i>10</i>	0.1113	0.1088	0.1048
	<i>30</i>	0.1076	0.0937	0.0934
	<i>50</i>	0.0958	0.0935	0.0933

Table A.2.17. Minimal Unavailability Values when Crossover Rate is Equal to 0.8

Crossover Rate = 0.8				
		Mutation Rate		
		<i>0.001</i>	<i>0.005</i>	<i>0.01</i>
Population Size	<i>10</i>	0.0977	0.0960	0.0947
	<i>30</i>	0.0950	0.0939	0.0934
	<i>50</i>	0.0956	0.0938	0.0928

Table A.2.18. Minimal Unavailability Values when Crossover Rate is Equal to 0.95

Crossover Rate = 0.95				
		Mutation Rate		
		<i>0.001</i>	<i>0.005</i>	<i>0.01</i>
Population Size	<i>10</i>	0.0983	0.0953	0.0942
	<i>30</i>	0.0948	0.0936	0.0934
	<i>50</i>	0.0949	0.0933	0.0926

APPENDIX 3

A.3.1. UAV DESIGN OPTIMISATION USING THE PHASED MISSION DESIGN OPTIMISATION ALGORITHM

Table A.3.1. Basic Event Failure Probability Data

<i>Basic Event Name in Fault Trees</i>	<i>Description</i>	<i>Failure Probability</i>
LGret	Landing gear can not be extended	0.02
LGex	Landing gear can not be retracted	0.025
Avion	Avionics system fails	0.01
Valve_b_O	Brake control valve fails opened	0.05
Valve_b_C	Brake control valve fails closed	0.05
Valve_a_O	Antiskid valve fails opened	0.04
Valve_a_C	Antiskid valve fails closed	0.05
Brake	Brakes fail	0.03
Eng 1	Engine 1 fails	0.01
Eng 2	Engine 2 fails	0.01
Valve1c_O	Cross feed valve 1 fails opened	0.037
Valve1c_C	Cross feed valve 1 fails closed	0.06
Valve2c_O	Cross feed valve 2 fails opened	0.037
Valve2c_C	Cross feed valve 2 fails closed	0.04
Tank1	Tank 1 fails	0.01
Tank2	Tank 2 fails	0.01
Pump1	Pump 1 fails	0.03
Pump2	Pump 2 fails	0.03
Navig	Navigation system fails	0.01
Avoid	Sense and avoidance system fails	0.01
Flight	Flight control surfaces fail	0.01
Canc	Phase is aborted	0.035
Bird1	Bird strike on engine 1	0.06
Bird2	Bird strike on engine 2	0.06
Atc	Air traffic control failure	0.01
Aircraft	Other aircraft	0.02
Storm	Storm	0.03
Comm	Communication mistake	0.01

Table A.3.2. Additional Basic Events Data

<i>Description</i>	<i>Basic Event Code in the Data Files</i>	<i>Failure Probability</i>
Type 1 landing gear can not be extended	1_1_1	0.02
Type 2 landing gear can not be extended	1_1_2	0.01
Type 1 antiskid valve fails opened	6_1_1	0.04
Type 2 antiskid valve fails opened	6_1_2	0.035

<i>Description</i>	<i>Basic Event Code in the Data Files</i>	<i>Failure Probability</i>
Type 3 antiskid valve fails opened	6_1_3	0.045
Type 1 brakes fail	8_1_1	0.03
Type 2 brakes fail	8_1_2	0.02
Type 1 navigation system fails	19_1_1	0.01
Type 2 navigation system fails	19_1_2	0.015
Type 1 sense and avoidance system fails	20_1_1	0.01
Type 2 sense and avoidance system fails	20_1_2	0.0125

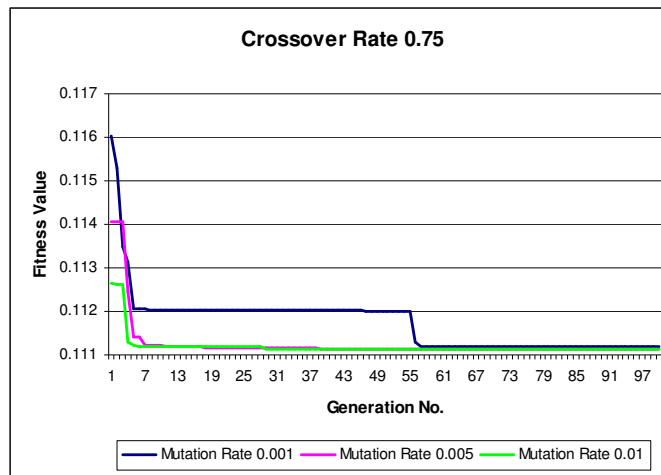


Figure A.3.1. Means of Best Fitness Values from Five Runs when Crossover Rate = 0.75, Population Size 10 Chromosomes

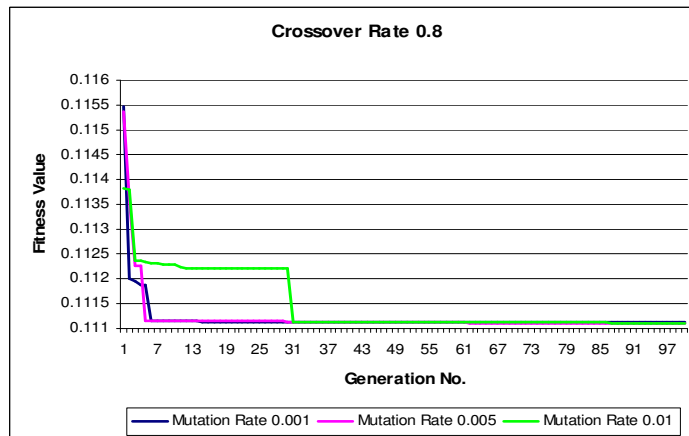


Figure A.3.2. Means of Best Fitness Values from Five Runs when Crossover Rate = 0.8, Population Size 10 Chromosomes

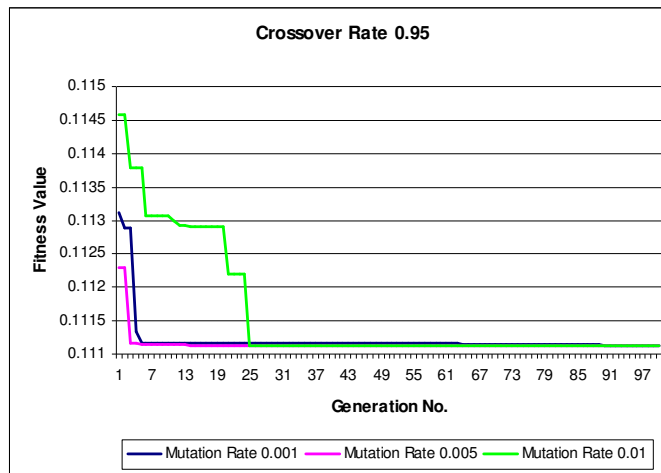


Figure A.3.3. Means of Best Fitness Values from Five Runs when Crossover Rate = 0.95, Population Size 10 Chromosomes

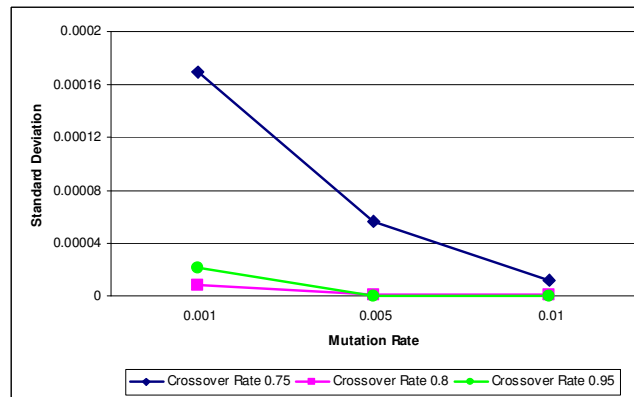


Figure A.3.4. Standard Deviation of the Best Fitness Values from Five Runs, Population Size 10 Chromosomes

Crossover rate \ Mutation Rate	Mutation Rate		
	0.001	0.005	0.01
0.75	0.111193998	0.111143857	0.111124663
0.8	0.111123294	0.11111812	0.11111812
0.95	0.111127812	0.111117676	0.111117676

Table A.3.3. Average Best Optimisation Results after 100 Generations for Five Runs, Population Size 10 Chromosomes

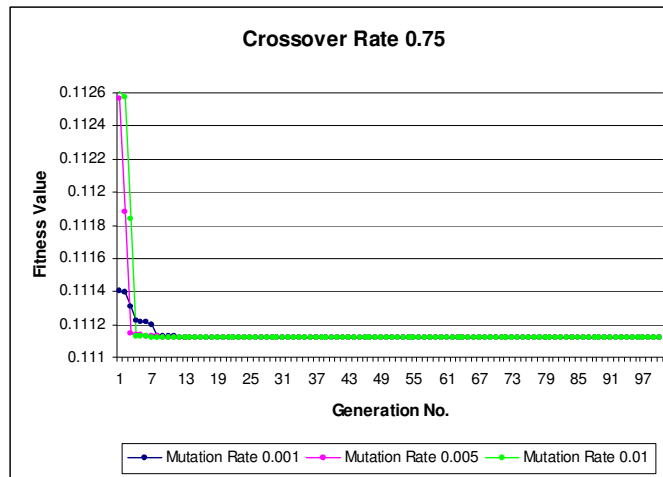


Figure A.3.5. Means of Best Fitness Values from Five Runs when Crossover Rate = 0.75, Population Size 30 Chromosomes

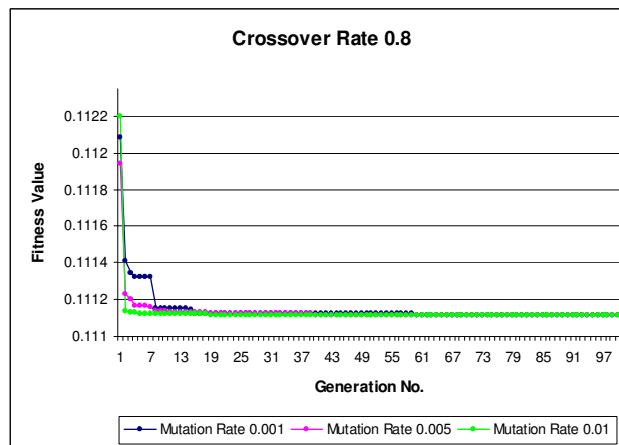


Figure A.3.6. Means of Best Fitness Values from Five Runs when Crossover Rate = 0.8, Population Size 30 Chromosomes

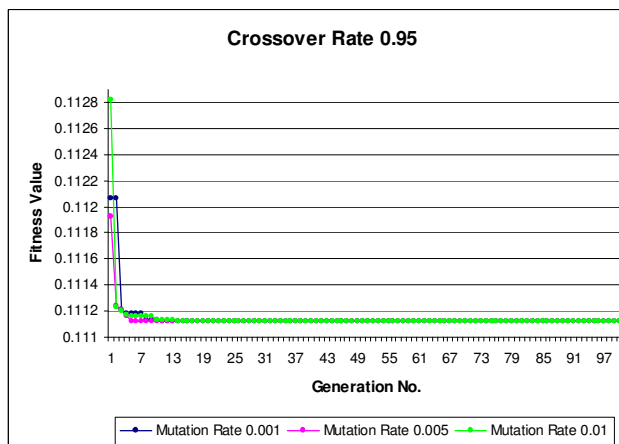


Figure A.3.7. Means of Best Fitness Values from Five Runs when Crossover Rate = 0.95, Population Size 30 Chromosomes

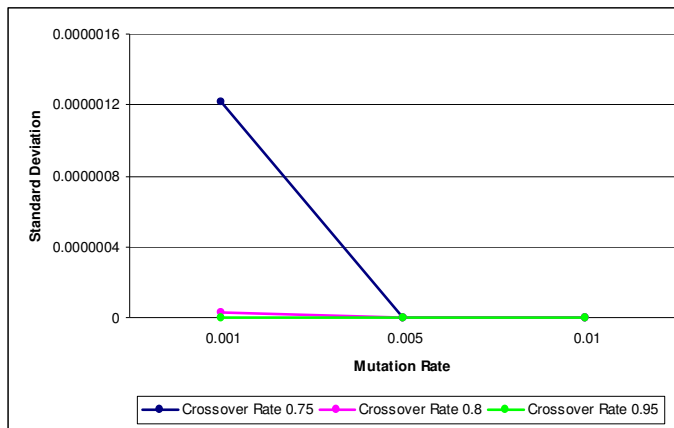


Figure A.3.8. Standard Deviation of the Best Fitness Values from Five Runs when Population Size is 30 Chromosomes

Table A.3.4. Average Values for Best Optimisation Results after 100 Generations, Population Size 30 Chromosomes

Mutation Rate \ Crossover rate	0.001	0.005	0.01
0.75	0.111118565	0.111117676	0.111117676
0.8	0.111117689	0.111117676	0.111117676
0.95	0.111117676	0.111117676	0.111117676

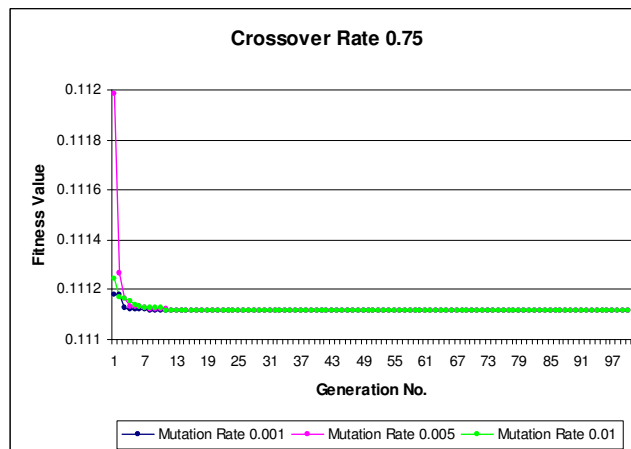


Figure A.3.9. Means of Best Fitness Values from Five Runs when Crossover Rate = 0.75, Population Size 50 Chromosomes

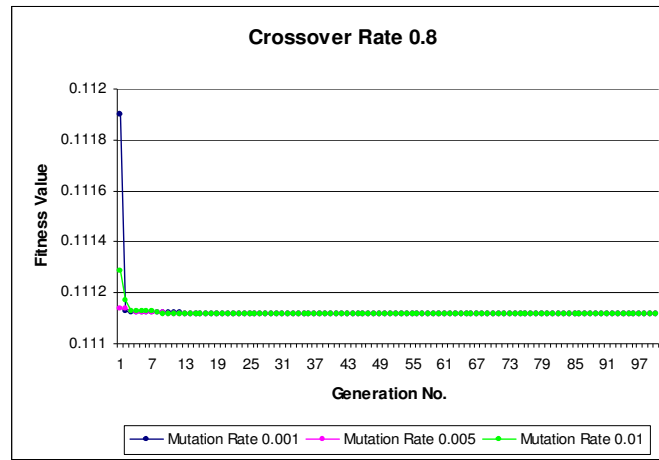


Figure A.3.10. Means of Best Fitness Values from Five Runs when Crossover Rate = 0.8, Population Size 50 Chromosomes

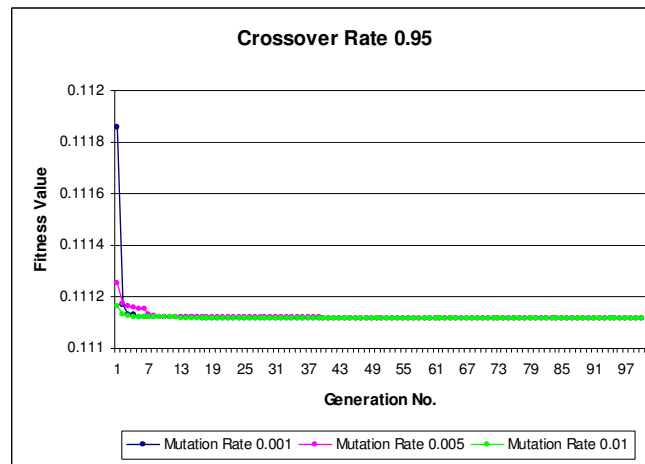


Figure A.3.11. Means of Best Fitness Values from Five Runs when Crossover Rate = 0.95, Population Size 50 Chromosomes

Table A.3.5. Average Values for Best Optimisation Results after 100 Generations, Population Size 50 Chromosomes

Mutation Rate \ Crossover rate	0.001	0.005	0.01
0.75	0.111117676	0.111117676	0.111117676
0.8	0.111117676	0.111117676	0.111117676
0.95	0.111117676	0.111117676	0.111117676

A.3.2. MILITARY VESSEL DESIGN OPTIMISATION USING *PMSDOP* (CASE 1)

Table A.3.6. Data of Power & Propulsion System Components

<i>Failure Events</i>	<i>Failure Probabilities</i>
Nuclear steam raising plant fails	1.7278×10^{-4}
Condenser fails	2.5654×10^{-4}
Circulating water pump fails	8.7222×10^{-4}
Circulating water hull valve fails	1.9368×10^{-4}
Circulating water system fails	1.9368×10^{-4}
Air ejection gland fails	2.5654×10^{-4}
Lubricating oil system fails	2.3277×10^{-4}
Main lubricating oil filter fails	3.1033×10^{-4}
Main lubricating oil cooler and thermostat fails	2.3278×10^{-4}
Feed pump fails	3.1035×10^{-4}
Extraction pump 1, 2 fails	4.6399×10^{-4}
Main lubricating oil pump (alternating current) fails	3.8733×10^{-4}
Main lubricating oil pump (direct current) fails	5.8093×10^{-4}
Ahead valve fails	2.6165×10^{-3}
Astern throttle valve fails	2.6165×10^{-3}
Turbine fails	1.3093×10^{-3}
Main gear box fails	4.4457×10^{-3}
Clutches fail	4.4457×10^{-3}
Shaft fails	2.5654×10^{-4}
Shaft seal fails	2.5654×10^{-4}
Thrust block fails	2.5654×10^{-4}
Propulsor fails	1.7611×10^{-4}

Table A.3.7. Data of Electrical Supply System Components

<i>Failure Events</i>	<i>Failure Probabilities</i>
Motor generator (MG) Fails	1.8141×10^{-3}
MG alternating current automatic voltage regulator fails	6.6208×10^{-4}
MG direct current voltage and frequency regulator fails	2.9791×10^{-3}
Direct current switchboard fails	3.3112×10^{-4}
Alternating current switch board 1, 2 fails	2.8407×10^{-4}
Turbine 1, 2 Fails	2.6193×10^{-4}
Turbo generator (TG) bearing 1, 2 fails	2.0075×10^{-4}
Generator bearing 1, 2 fails	2.0075×10^{-4}
TG air cooler 1, 2 fails	2.0075×10^{-4}
TG governor trips & control system 1, 2 fails	2.6196×10^{-4}
Duplex filter 1, 2 fails	2.6196×10^{-4}
Generator 1, 2 fails	2.3919×10^{-3}
TG automatic voltage regulator 1, 2 fails	9.952×10^{-5}

Table A.3.8. Data of Fresh Water Cooling System Components

<i>Failure Events</i>	<i>Failure Probabilities</i>
Hull valve (inlet) 1, 2 fails	1.7611×10^{-4}
Flexible coupling unit 1, 2, 3, 4 fails	1.7611×10^{-4}

<i>Failure Events</i>	<i>Failure Probabilities</i>
Sea water services system 1, 2 fails	1.7611×10^{-4}
Sea water service pump 1, 2 fails	5.2825×10^{-4}
Hull valve (outlet) 1, 2 fails	1.7611×10^{-4}
Heat exchanger 1, 2 fails	1.7611×10^{-4}

Table A.3.9. Data of Hydraulics System Components

<i>Failure Events</i>	<i>Failure Probabilities</i>
External hydraulic plant fails	1.0861×10^{-3}
External hydraulic system fails	2.7164×10^{-4}
Aft system fails	2.7164×10^{-4}
Aft plant (steering) (alternating current) fails	2.7131×10^{-3}
Aft plant (steering) (direct current) fails	9.4637×10^{-3}
Main hydraulic plant fails	1.6287×10^{-3}
Main hydraulic system fails	2.7164×10^{-4}

Table A.3.10. Data of Hydroplanes Control System Components

<i>Explanation</i>	<i>Failure Probabilities</i>
Aft hydroplane control surfaces fail	2.3277×10^{-4}
Hydraulic tilting cylinder fails	2.3277×10^{-4}
Aft hydroplane ram servo unit fails	6.9770×10^{-4}
Aft hydroplane order transmission box fails	2.3283×10^{-4}
Air tilting cylinder fails	2.3277×10^{-4}
Air in emergency control fails	6.9770×10^{-4}
For'd hydroplane control surfaces fail	2.3277×10^{-4}
Tilting cylinder fails	2.3277×10^{-4}
For'd hydroplane ram servo unit fails	6.9770×10^{-4}
For'd hydroplane order transmission box fails	2.3283×10^{-4}

Table A.3.11. Data of the Rudder Control System Components

<i>Failure Events</i>	<i>Failure Probabilities</i>
Control surfaces fail	4.6549×10^{-4}
Rudder ram fails	2.3277×10^{-4}
Ram servo unit fails	1.3963×10^{-3}
Rate control fails	2.3283×10^{-4}

Table A.3.12. Data of New Vessel Components

<i>Components</i>	<i>Failure Probabilities</i>
CW pump type 2	5.2825×10^{-4}
Feed pump type 2	1.2546×10^{-4}
Ahead valve type 2	3.6328×10^{-3}
MG VFR type 2	1.9654×10^{-3}
External hydraulic plant type 2	5.6120×10^{-4}
Main hydraulic plant type 2	2.7131×10^{-3}

Table A.3.13. Data of Components Cost and Weight

<i>Component</i>	<i>Cost</i>	<i>Weight</i>	<i>Component</i>	<i>Cost</i>	<i>Weight</i>
NRSP	10	20	TG Air Cooler 1, 2	7	200
Condenser	6	700	TG Governor Trips & Control System 1, 2	6	60
CW Pump Type 1	1	76	Duplex Filter 1, 2	3	10
CW Pump Type 2	0.7	80	Generator 1, 2	4	120
CW Hull Valve	1	10	TG AVR 1, 2	3	15
Circ Water System	9	50	Hull Valve (Inlet) 1, 2	2	10
Air Ejection Gland	6	10	Flexible Coupling Unit 1, 2, 3, 4	3	70
Lub Oil System	7	30	Sea Water Services System 1, 2	6	30
Main Lub Oil Filter	5	3	Sea Water Service Pump 1, 2	3	200
Main Lub Oil Cooler and Thermostat	7	20	Hull Valve (Outlet) 1, 2	3	70
Feed Pump Type 1	4	6	Heat Exchanger 1,	4	8
Feed Pump Type 2	5	5	External Hydraulic Plant Type 1	7	1500
Extraction Pump 1, 2	3	5	External Hydraulic Plant Type 2	7	1700
Main Lub Oil Pump AC	4	4	External Hydraulic System	8	200
Main Lub Oil Pump DC	2	4.5	Aft System	9	150
Ahead Valve Type 1	2	6.5	Aft Plant (Steering) AC	4	100
Ahead Valve Type 2	1.5	5	Aft Plant (Steering) DC	4	100
Astern Throttle Valve	2	6.5	Main Hydraulic Plant Type 1	8	1000
Turbine	9	2300	Main Hydraulic Plant Type 2	7	900
Main Gear Box	7	2600	Main Hydraulic System	10	150
Clutches	7	2600	Aft Hydroplane Control Surfaces	4	60
Shaft	5	30	Hydraulic Tilting Cylinder	3.5	15
Shaft Seal	6	10	Aft Hydroplane Ram Servo Unit	5	50
Thrust Block	5	50	Aft Hydroplane Order Transmission Box	3	310
Propulsor	10	5000	Air Tilting Cylinder	2.5	12
MG	5	150	Air in Emergency Control	4	30
MG AC AVR	3	5	For'd Hydroplane Control Surfaces	4	60
MG DC VRF Type 1	1	4	Tilting Cylinder	3	15
MG DC VRF Type 2	1.5	5	For'd Hydroplane Ram Servo Unit	5	50
DC Switchboard	5.5	10	For'd Hydroplane Order Transmission Box	3	310
AC Switch board 1, 2	6	15	Control Surfaces	5	60
Turbine 1, 2	8	150	Rudder Ram	6	230
TG bearing 1, 2	7	30	Ram Servo Unit	4.5	50
Generator Bearing 1, 2	5	20	Rate Control	6	30

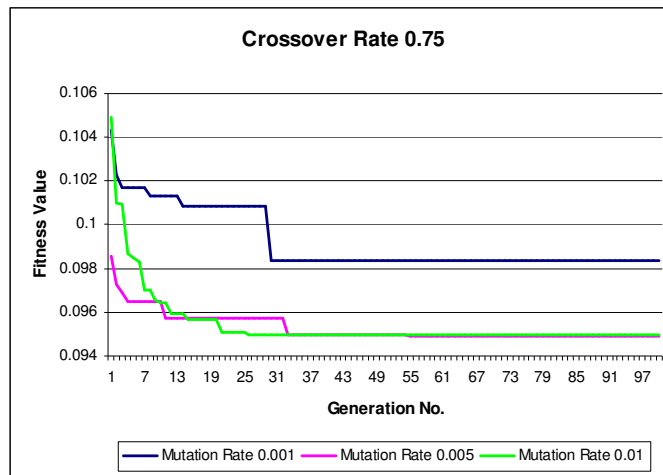


Figure A.3.12. Means of Best Fitness Values from Five Runs when Crossover Rate = 0.75, Population Size 10 Chromosomes

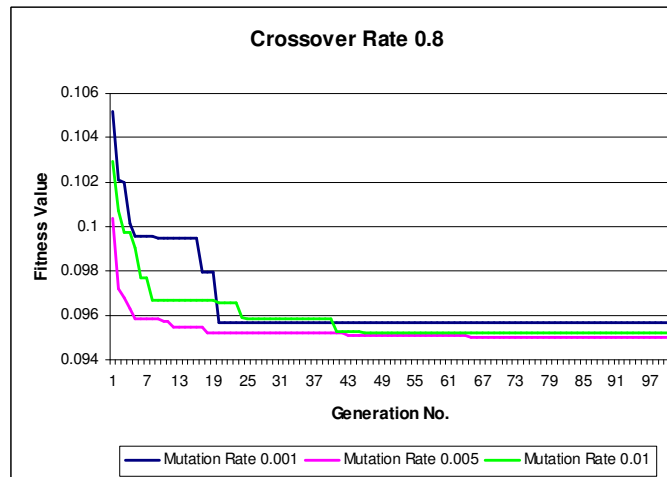


Figure A.3.13. Means of Best Fitness Values from Five Runs when Crossover Rate = 0.8, Population Size 10 Chromosomes

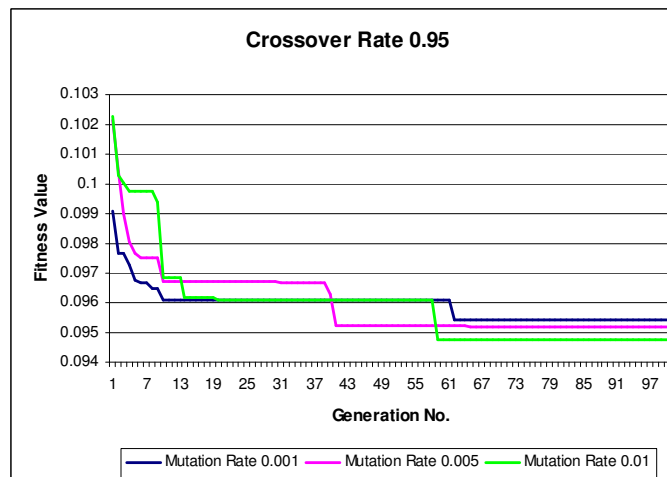


Figure A.3.14. Means of Best Fitness Values from Five Runs when Crossover Rate = 0.95, Population Size 10 Chromosomes

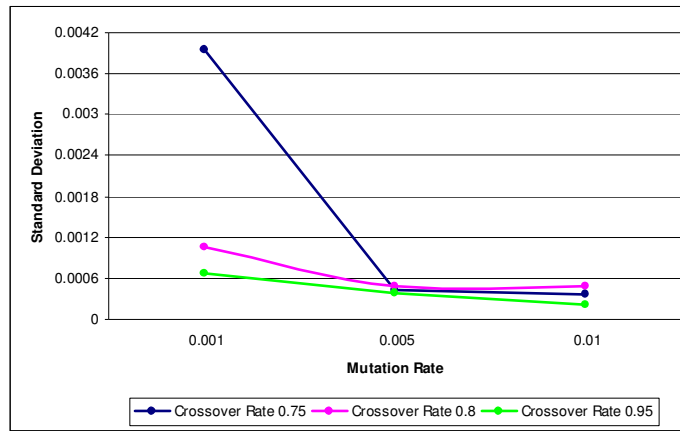


Figure A.3.15. Standard Deviation of the Best Fitness Values from Five Runs, Population Size 10 Chromosomes

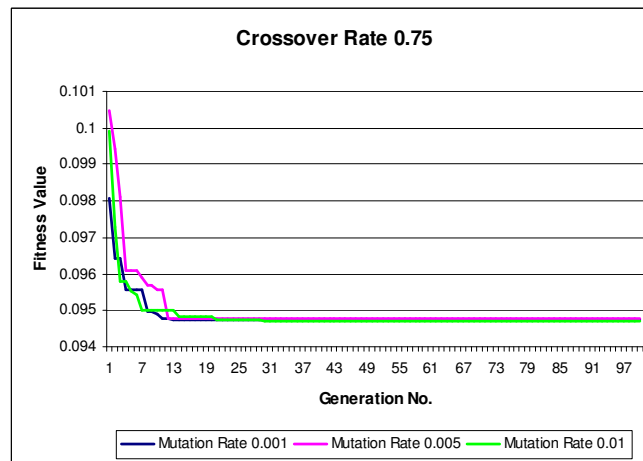


Figure A.3.16. Means of Best Fitness Values from Five Runs when Crossover Rate = 0.75, Population Size 30 Chromosomes

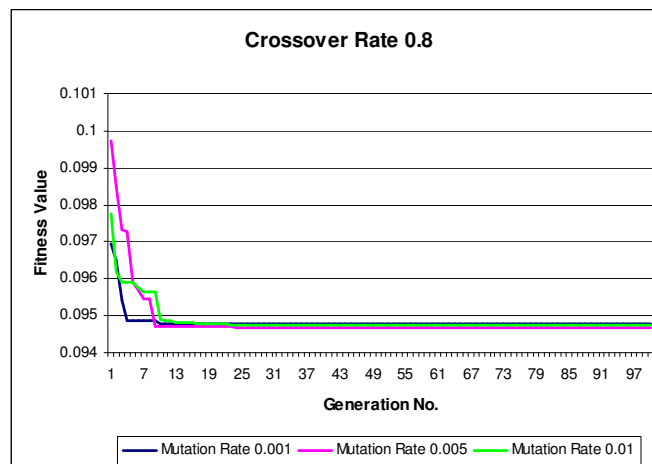


Figure A.3.17. Means of Best Fitness Values from Five Runs when Crossover Rate = 0.8, Population Size 30 Chromosomes

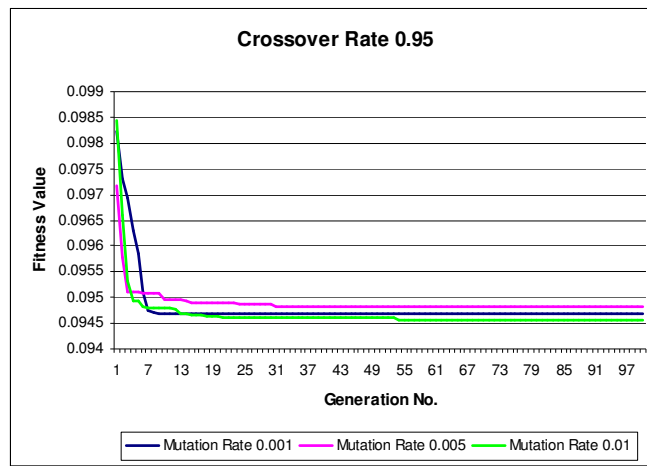


Figure A.3.18. Means of Best Fitness Values from Five Runs when Crossover Rate = 0.95, Population Size 30 Chromosomes

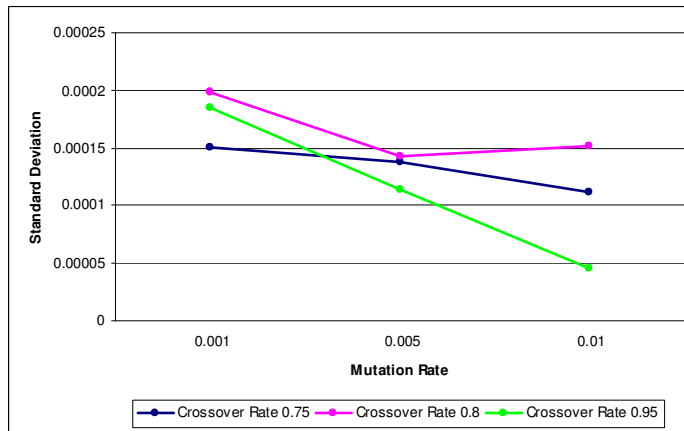


Figure A.3.19. Standard Deviation of the Best Fitness Values from Five Runs, Population Size 30 Chromosomes

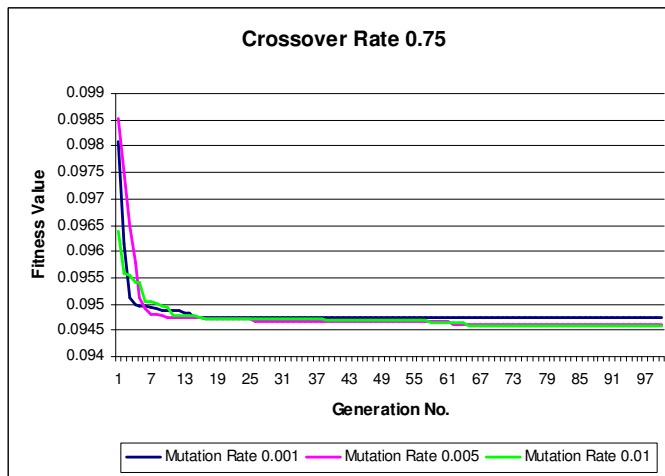


Figure A.3.20. Means of Best Fitness Values from Five Runs when Crossover Rate = 0.75, Population Size 50 Chromosomes

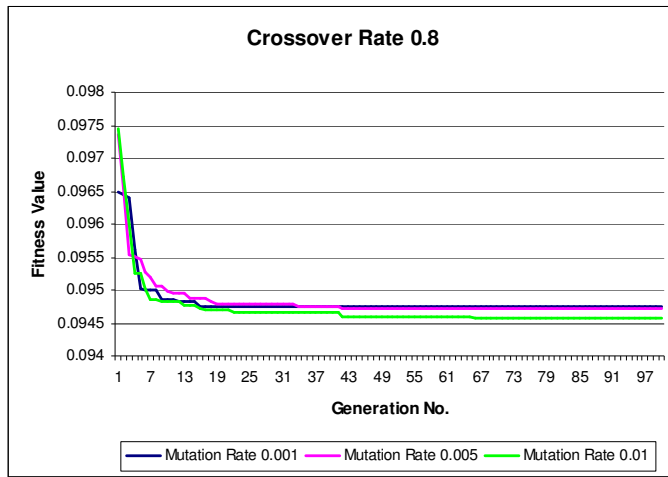


Figure A.3.21. Means of Best Fitness Values from Five Runs when Crossover Rate = 0.8, Population Size 50 Chromosomes

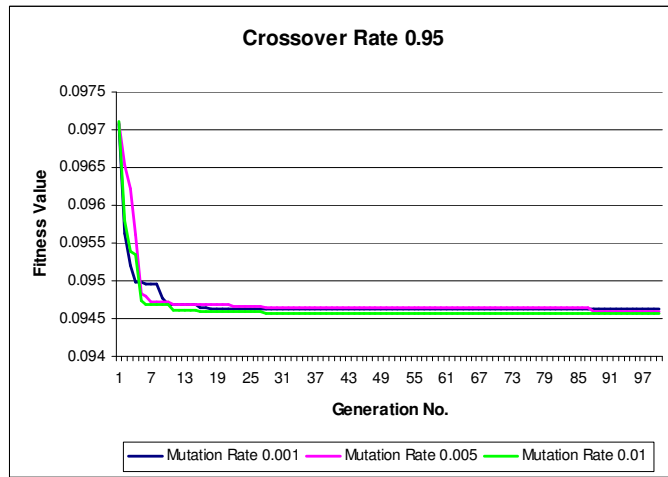


Figure A.3.22. Means of Best Fitness Values from Five Runs when Crossover Rate = 0.95, Population Size 50 Chromosomes

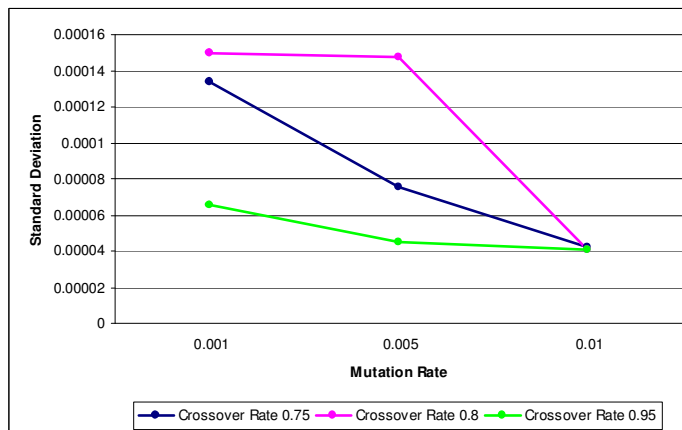


Figure A.3.23. Standard Deviation of the Best Fitness Values from Five Runs, Population Size 50 Chromosomes

A.3.3. MILITARY VESSEL DESIGN OPTIMISATION PROBLEM WITH CONSTRAINTS ADDED AT EACH PHASE

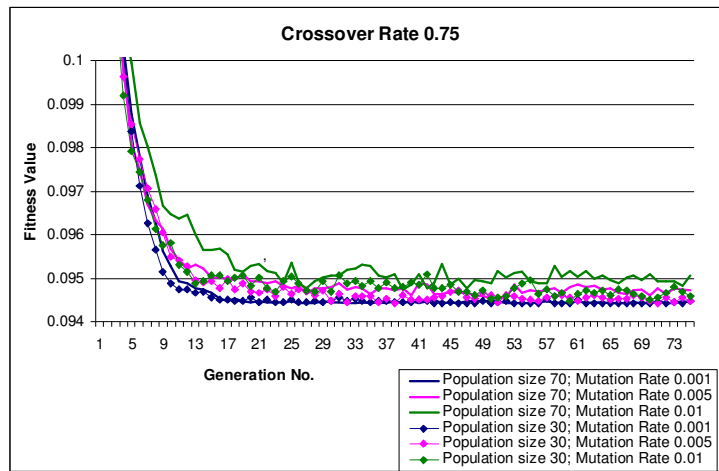


Figure A.3.24. Average Fitness Values for Generations when Crossover Rate = 0.75

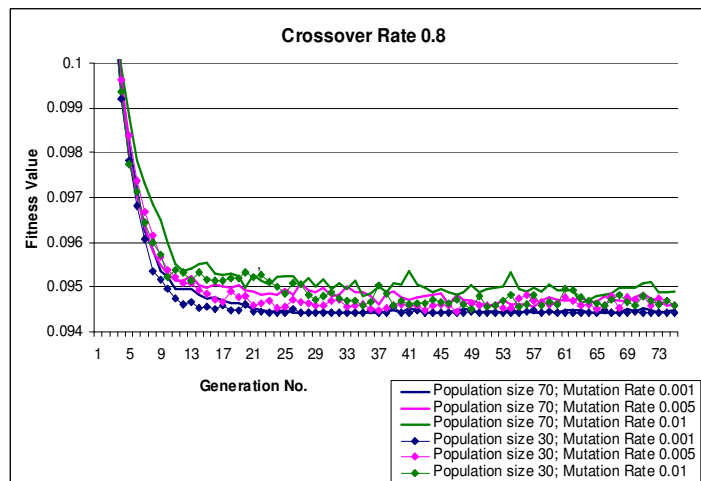


Figure A.3.25. Average Fitness Values for Generations when Crossover Rate = 0.8

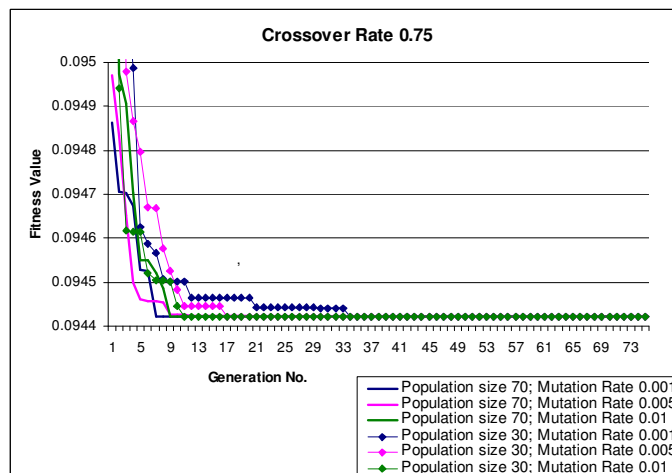


Figure A.3.26. Best Fitness Values for Generations when Crossover Rate = 0.75

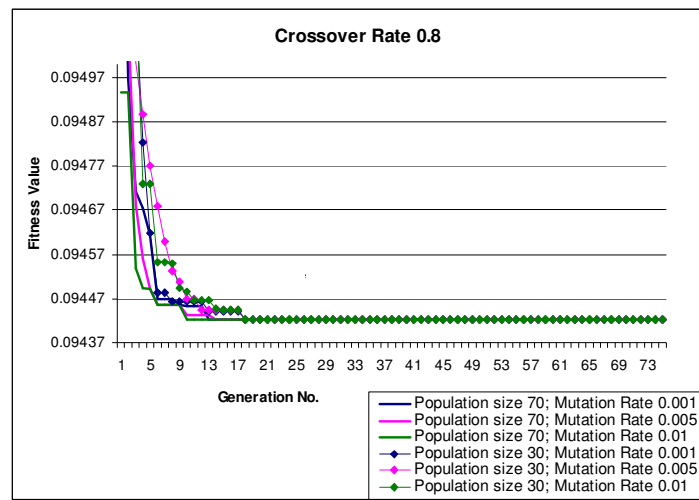


Figure A.3.27. Best Fitness Values for Generations when Crossover Rate = 0.8

APPENDIX 4

UAV DESIGN OPTIMISATION USING THE *MPMSDOA*

Table A.4.1. Basic Event Data

Basic Event Name in Fault Trees	Description	Failure Probability
LGret	Landing gear can not be extended	0. 002
LGex	Landing gear can not be retracted	0. 0025
Avion	Avionics system fails	0.001
Valve_b_O	Brake control valve fails opened	0.005
Valve_b_C	Brake control valve fails closed	0.005
Valve_a_O	Antiskid valve fails opened	0.004
Valve_a_C	Antiskid valve fails closed	0.005
LGret	Landing gear can not be extended	0.002
LGex	Landing gear can not be retracted	0.0025
Avion	Avionics system fails	0.001
Valve_b_O	Brake control valve fails opened	0.005
Valve_b_C	Brake control valve fails closed	0.005
Valve_a_O	Antiskid valve fails opened	0.004
Valve_a_C	Antiskid valve fails closed	0.005
Brake	Brakes fail	0.003
Eng 1	Engine 1 fails	0.004
Eng 2	Engine 2 fails	0.006
Valve1c_O	Cross feed valve 1 fails opened	0.0037
Valve1c_C	Cross feed valve 1 fails closed	0.006
Valve2c_O	Cross feed valve 2 fails opened	0.0037
Valve2c_C	Cross feed valve 2 fails closed	0.004
Tank1	Tank 1 fails	0.001
Tank2	Tank 2 fails	0.001
Pump1	Pump 1 fails	0.003
Pump2	Pump 2 fails	0.003
Navig	Navigation system fails	0.001
Avoid	Sense and avoidance system fails	0.001
Flight	Flight control surfaces fail	0.001
Canc	Phase is aborted	0.0035
Bird1	Bird strike on engine 1	0.006
Bird2	Bird strike on engine 2	0.006
Atc	Air traffic control failure	0.001
Aircraft	Other aircraft	0.002
Storm	Storm	0.003
Comm	Communication mistake	0.001

Table A.4.2. Additional Basic Event Data

Description	Failure Probability during Mission 1	Failure Probability during Mission 2
Type 1 landing gear can not be extended	0.002	0.008
Type 2 landing gear can not be extended	0.006	0.003
Type 1 antiskid valve fails opened	0.004	0.004
Type 2 antiskid valve fails opened	0.003	0.006
Type 3 antiskid valve fails opened	0.008	0.002
Type 1 brakes fail	0.003	0.001
Type 2 brakes fail	0.001	0.004
Type 1 engine 1 fails	0.004	0.005
Type 2 engine 1 fails	0.006	0.003
Type 3 engine 1 fails	0.009	0.009
Type 1 engine 2 fails	0.006	0.002
Type 2 engine 2 fails	0.0035	0.005
Type 3 engine 2 fails	0.0015	0.008
Type 1 navigation system fails	0.001	0.008
Type 2 navigation system fails	0.005	0.0025
Type 1 sense and avoidance system fails	0.001	0.007
Type 2 sense and avoidance system fails	0.004	0.0025

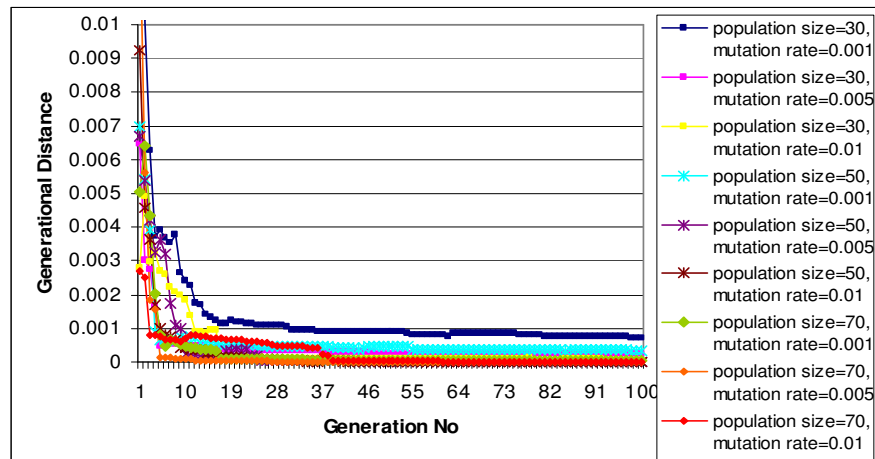


Figure A.4.1. Generational Metric Values when Crossover Rate = 0.75

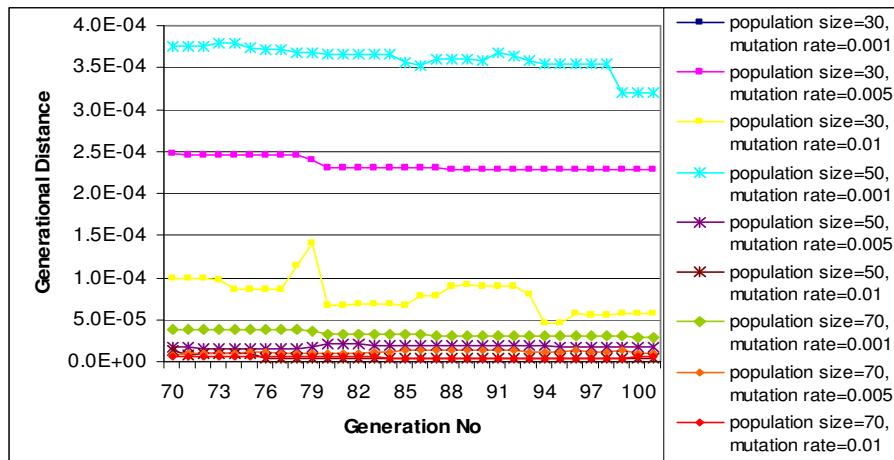


Figure A.4.2. Generational Metric Values for the Last 30 Generations when Crossover Rate = 0.75

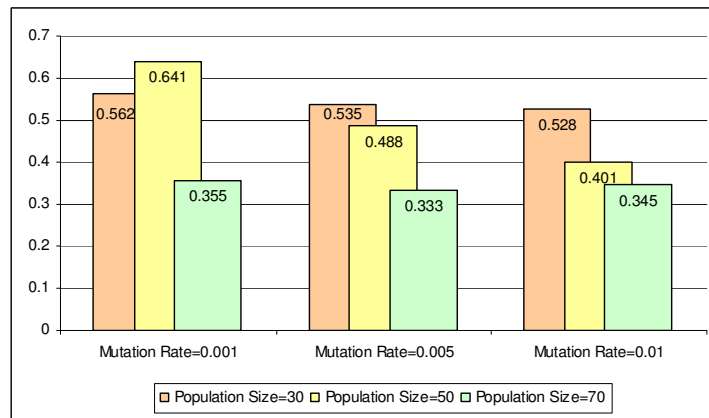


Figure A.4.3. Function C Values when Crossover Rate = 0.75



Figure A.4.4. Spacing Metric Values when Crossover Rate = 0.75

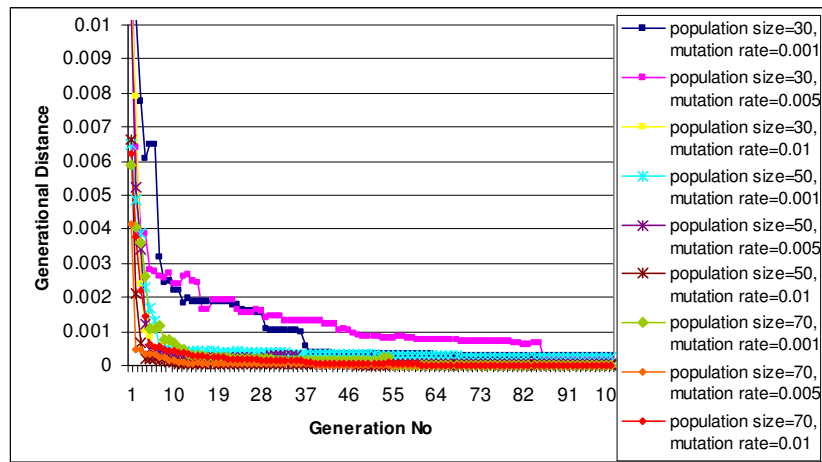


Figure A.4.5. Generational Metric Values when Crossover Rate = 0.8

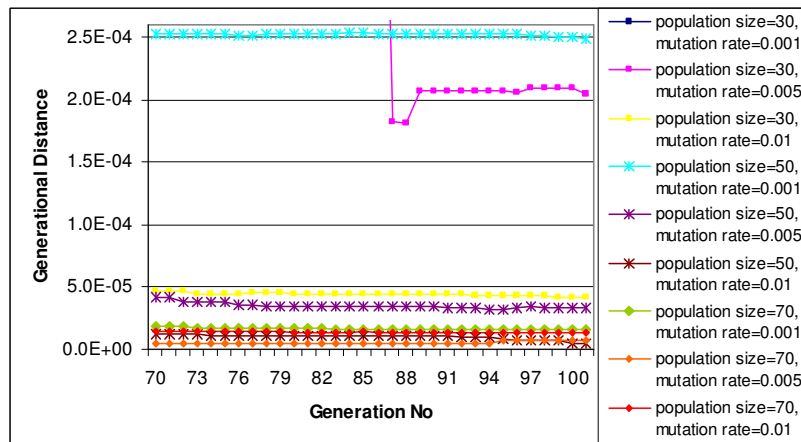


Figure A.4.6. Generational Metric Values for the Last 30 Generations when Crossover Rate = 0.8

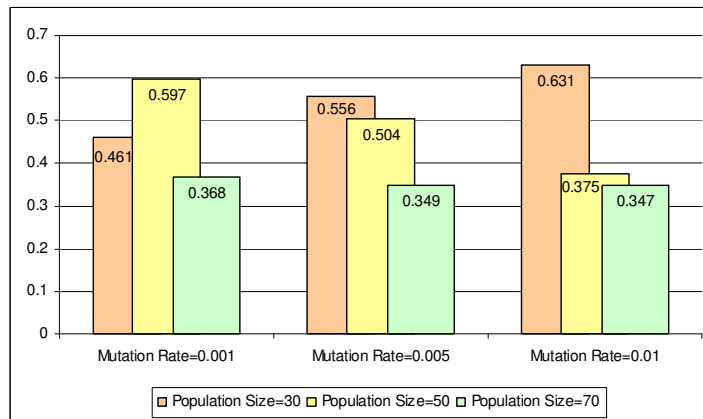


Figure A.4.7. Function C Values when Crossover Rate = 0.8



Figure A.4.8. Spacing Metric Values when Crossover Rate = 0.8

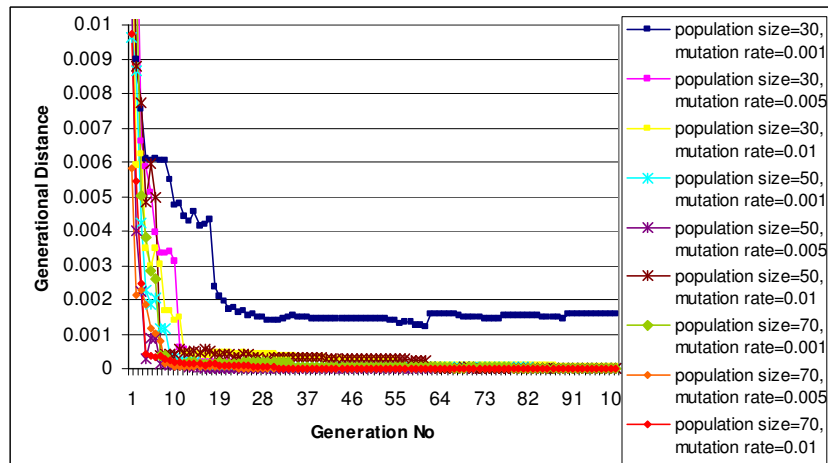


Figure A.4.9. Generational Metric Values when Crossover Rate = 0.95

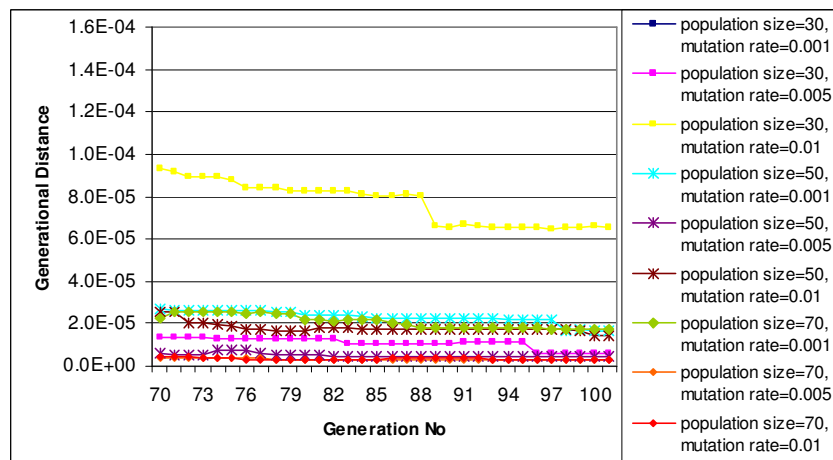


Figure A.4.10. Generational Metric Values for the Last 30 Generations when Crossover Rate = 0.8

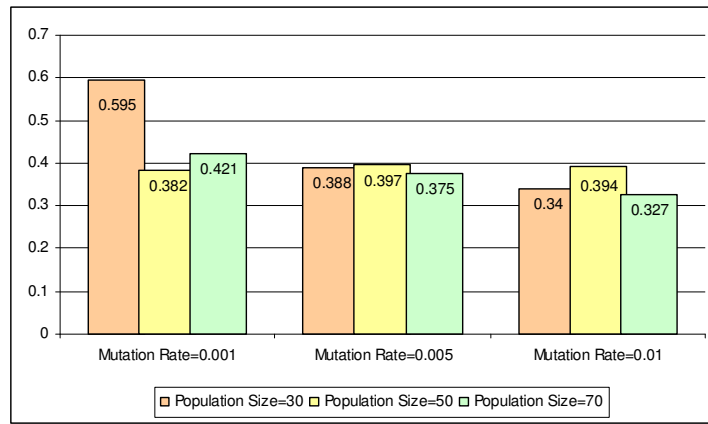


Figure A.4.11. Function C Values when Crossover Rate = 0.95

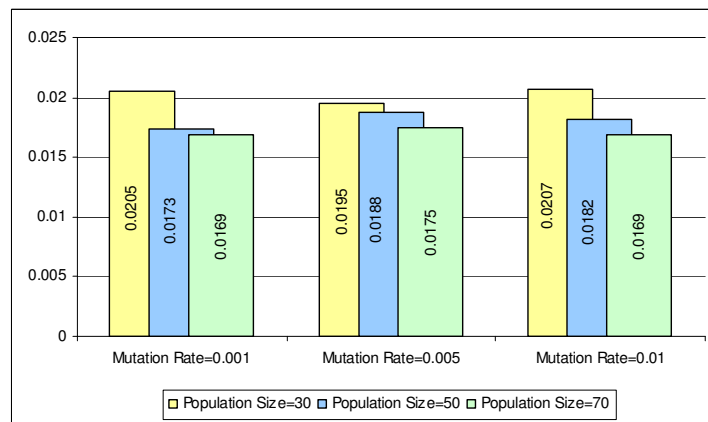


Figure A.4.12. Spacing Metric Values when Crossover Rate = 0.95