



Pilkington Library

Author/Filing Title PATTISON

Vol. No. Class Mark T


**Please note that fines are charged on ALL
overdue items.**

OPEN SHELF COPY

0402293428



Safety System Design Optimisation

 Loughborough University Pilkington Library
Date Dec 00
Class
Acc No. 040229342

M000 2657LB

ACKNOWLEDGMENTS

"In my distress I...cried unto my God." – Psalm 18:6

I would like to acknowledge all those who helped me so much with the completion of this thesis. I am indebted to my mum, dad and sister Ruth who, as always, were by my side and gave me support and encouragement at the most important moments. I thank my boyfriend, Dave and my friends (Kirsty, Sarah, Jim and many others) for being there to lean on and for always being great fun.

I would especially like to acknowledge the support of Dr. John Andrews who supervised me through my PhD. His help and advice was crucial, in addition to his constancy and friendship.



"The end of a matter is better than its beginning and patience is better than pride." – Ecclesiastes 7:8

"When you are a Bear of Very Little Brain, and you Think of things, you find sometimes that a Thing which seemed very Thingish inside you is quite different when it gets out into the open and has other people looking at it" – Winnie-the-Pooh

ABSTRACT

Key Words: Fault Trees, Binary Decision Diagrams, Optimisation, Genetic Algorithms, Safety Systems, Reliability.

This thesis investigates the efficiency of a design optimisation scheme that is appropriate for systems which require a high likelihood of functioning on demand. Traditional approaches to the design of safety critical systems follow the preliminary design, analysis, appraisal and redesign stages until what is regarded as an acceptable design is achieved. For safety systems whose failure could result in loss of life it is imperative that the best use of the available resources is made and a system which is optimal, not just adequate, is produced.

The object of the design optimisation problem is to minimise system unavailability through manipulation of the design variables, such that limitations placed on them by constraints are not violated.

Commonly, with mathematical optimisation problems there will be an explicit objective function which defines how the characteristic to be minimised is related to the variables. As regards the safety system problem, an explicit objective function cannot be formulated, and as such, system performance is assessed using the fault tree method. By the use of house events a single fault tree is constructed to represent the failure causes of each potential design to overcome the time consuming task of constructing a fault tree for each design investigated during the optimisation procedure. Once the fault tree has been constructed for the design in question it is converted to a BDD for analysis.

A genetic algorithm is first employed to perform the system optimisation, where the practicality of this approach is demonstrated initially through application to a High-Integrity Protection System (HIPS) and subsequently a more complex Firewater Deluge System (FDS).

An alternative optimisation scheme achieves the final design specification by solving a sequence of optimisation problems. Each of these problems are defined by assuming some form of the objective function and specifying a sub-region of the design space over which this function will be representative of the system unavailability.

The thesis concludes with attention to various optimisation techniques, which possess features able to address difficulties in the optimisation of safety critical systems. Specifically, consideration is given to the use of a statistically designed experiment and a logical search approach.

CONTENTS

1.	Introduction.	1
1.1	Introduction to Risk and Reliability Assessment.	1
1.2	Background.	1
1.3	Terminology.	3
1.4	Methods of Analysis.	4
1.5	System Design.	8
1.6	Safety Design Considerations.	9
1.7	The Design Optimisation Problem.	11
1.8	Objectives of the Project.	12
2.	Fault Tree Analysis.	14
2.1	Background.	14
2.2	General Description.	14
2.2.1	System Definition and Fault Tree Construction.	15
2.3	Qualitative Analysis.	17
2.3.1	Derivation of Minimal Cut Sets.	19
2.3.2	Modularisation.	24
2.4	Quantitative Analysis.	25
2.4.1	Component Failure Parameters.	26
2.4.1.1	Failure Rate Models.	32
2.4.2	System Failure Parameters.	35
2.4.2.1	Top Event Quantification.	35
2.4.2.2	Unconditional System Failure Intensity.	36
2.5	Faulttree+.	42
2.5.1	Basic Event Model Types.	43
2.5.2	Analysis in Faulttree+.	45
2.6	Binary Decision Diagrams.	47
2.6.1	Introduction to BDD's.	48
2.6.2	General Description.	49

2.6.3	Binary Decision Diagram Construction.	50
2.6.4	Minimising Procedure.	54
2.6.5	Ordering.	57
2.6.6	Top Event Quantification.	58
2.6.6.1	Top Event Probability.	58
2.6.6.2	Unconditional System Failure Intensity.	60
3.	Optimisation Techniques.	66
3.1	Introduction.	66
3.2	Linear Programming.	67
3.2.1	The Simplex Method.	68
3.2.2	Integer Programming.	68
3.2.3	Reliability and LP Techniques.	78
3.2.4	Further Research.	80
3.2.5	Summary.	81
3.2.6	Logical Search.	83
3.3	Gradient Techniques.	84
3.3.1	Basic Methods.	85
3.3.2	Advanced Methods.	87
3.3.3	Constrained Gradient Techniques.	91
3.3.4	Summary.	98
3.4	Direct Search Methods.	99
3.4.1	Unconstrained Direct Search Methods.	100
3.4.2	The Constrained Case.	105
3.4.3	Further Research.	108
3.4.4	Summary.	108
3.5	Random Search Methods.	111
3.5.1	Pure Random Search.	111
3.5.2	Controlled Random Search.	112
3.5.3	Partitioned Random Search.	116
3.5.4	The Combinatorial Heuristic Method.	118

4.	Genetic Algorithms.	121
4.1	Introduction to Evolutionary Computation.	121
4.2	Introduction to Genetic Algorithms.	122
4.3	The Simple Genetic Algorithm.	124
4.3.1	Representation.	125
4.3.2	Initialisation.	127
4.3.3	Fitness Evaluation.	127
4.3.4	Selection.	128
4.3.5	Genetic Operators.	129
4.3.6	Result Designation and Termination Criteria.	130
4.3.7	GA Parameters.	131
4.4	The Theoretical Foundation.	132
4.5	The Constrained Case.	133
4.6	Premature Convergence.	136
4.7	Modifications to the Genetic Algorithm.	137
4.7.1	Representation.	137
4.7.2	Fitness Evaluation.	138
4.7.3	Selection.	139
4.7.4	Recombination Operators.	142
4.7.5	Hybridisation.	144
5.	System Analysis.	147
5.1	Introduction to HIPS System.	147
5.1.1	Description of HIPS System.	147
5.2	Safety System Analysis.	150
5.2.1	Evaluate the System Unavailability.	151
5.2.1.1	Construction of the System Unavailability Fault Tree.	152
5.2.1.2	Data Input and Analysis.	164
5.2.1.3	Conversion of the Fault Tree to a BDD.	164
5.2.1.3.1	BADD.	165
5.2.1.3.2	BDD Structure to Represent System Unavailability.	169

5.2.1.4	Computational Method for BDD Quantification.	170
5.2.1.4.1	Input of the BDD File Structure.	171
5.2.1.4.2	Input Event Data.	171
5.2.1.4.3	Analysing the BDD Structure.	172
5.2.2	Cost and MDT Evaluation.	177
5.2.3	Evaluate the Frequency of Spurious Trip Occurrence.	179
5.2.3.1	Construction of the Spurious Trip Fault Tree.	179
5.2.3.2	Data Input.	182
5.2.3.3	Conversion of the Spurious Trip Fault Tree to a BDD.	182
5.2.3.4	A Computational Method for the Top Event Unconditional Failure Intensity of the BDD.	183
5.2.3.4.1	Input of the BDD File Structure and Event Data.	183
5.2.3.4.2	Analysing the BDD Structure.	184
5.2.4	Accuracy Comparison with Faulttree+	184
6.	Implementing the GA to Optimise the High Integrity Protection System.	189
6.1	Introduction.	189
6.2	Computer Implementation of the GA to Optimise the HIPS – GASSOP.	190
6.2.1	Input to GASSOP.	190
6.2.2	Coding and Initialisation.	192
6.2.3	Evaluating String Fitness.	194
6.2.3.1	Derivation of Penalty Formulae.	196
6.2.3.2	Fitness Information per Generation.	199
6.2.4	Selection.	199
6.2.4.1	Fitness Conversion Method.	199
6.2.4.2	Sampling.	202
6.2.5	Action of Genetic Operators.	203
6.2.6	Update Design Data.	206
6.2.7	Update Fitness Data.	208
6.2.8	Check for Termination.	208
6.2.9	Output from GASSOP.	208
6.2.10	Results from a Run of GASSOP.	209

6.3	GA Parameters.	212
6.3.1	Discussion of Quantitative Results.	213
6.4	Further Testing.	214
6.4.1	Discussion of Results.	217
7.	Modifications to the GA.	218
7.1	Introduction.	218
7.2	Utilisation of the MDT Resource.	219
7.2.1	MDT Modification Method 1.	220
7.2.2	MDT Modification Method 2.	221
7.2.3	MDT Modification Method 3.	224
7.2.4	Results of the MDT Modification Methods.	225
7.2.5	Discussion of the MDT Modification Results.	228
7.3	Derivation of a Modified Cost Penalty Formula.	230
7.3.1	Results of the GA Using the Modified Cost Penalty.	232
7.3.2	Discussion of the Modified Cost Penalty.	233
7.4	The Conversion Method.	234
7.4.1	Application of the Original Method.	234
7.4.2	A modified Conversion Method.	236
7.4.3	An Example of the Modified Conversion Process.	239
7.4.4	Results Comparing Both Conversion Methods.	241
7.4.5	Discussion of Conversion Method Results.	242
7.5	The Modified GA.	242
7.5.1	Discussion of Results from the Modified GA.	244
8.	Grid-Sampling Optimisation Technique.	246
8.1	Introduction.	246
8.2	The Optimisation Algorithm.	247
8.2.1	Formulation of the Objective Function.	250
8.2.2	Limiting the Scope of the Objective Functions.	254

8.2.3	Searching the Restricted Design Space.	256
8.2.4	Reducing the Restricted Design Space.	258
8.2.5	Application of the Grid-Sampling Method.	259
8.2.6	Results Using the Grid-Sampling Method.	263
8.2.7	Discussion of Results.	265
8.3	Other Formulations of the Objective Function.	266
8.3.1	The Pure Quadratic Objective Function.	266
8.3.1.1	Application Using the Pure Quadratic Objective Function.	268
8.3.1.2	Results Using the Pure Quadratic Objective Function.	270
8.3.2	The Cross Quadratic Objective Function.	271
8.3.2.1	Application Using the Quadratic Objective Function with Cross Terms.	273
8.3.2.2	Results Using the Quadratic Objective Function with Cross Terms.	277
8.3.3	Discussion of Results Using Other Forms of the Objective Function.	278
9.	Implementing the Optimisation Procedure to a Firewater Deluge System.	280
9.1	Introduction.	280
9.2	Description of the Firewater Deluge System.	280
9.2.1	The Deluge System.	282
9.2.2	The Firewater Supply and Distribution System.	286
9.2.3	The AFFF Supply and Distribution System.	290
9.2.4	Design Variables.	291
9.3	Safety System Analysis.	293
9.3.1	Evaluate the System Unavailability.	294
9.3.1.1	Construction of the System Unavailability Fault Tree.	294
9.3.1.2	Converting the System Unavailability Fault Tree to a BDD.	299
9.3.1.3	Computational Method for BDD Quantification.	301
9.3.1.3.1	Transferring the BDD.	303
9.3.1.3.2	Derivation of Component Unavailability.	304
9.4	Evaluate the Frequency of Spurious Trip Occurrences of the FDS.	313
9.4.1	Construction of the Spurious Trip Fault Tree.	313

9.4.2	Conversion of the Spurious Trip Fault Tree to a BDD.	314
9.4.3	Evaluate the Top Event Unconditional Failure Intensity of the Fault Tree	314
9.5	Life Cycle Costs.	314
9.5.1	Initial Cost.	315
9.5.2	Corrective Maintenance Cost.	318
9.5.3	Preventative Maintenance Cost.	321
9.5.4	Testing Cost.	322
9.5.5	Evaluate the Life Cycle Cost.	324
9.5.6	Implementing Life Cycle Cost Evaluation within the Computational Method.	325
9.6	Computer Implementation of the GA to optimise the FDS.	327
9.6.1	Memory.	328
9.6.2	Coding and Initialisation.	328
9.6.3	Evaluate String Fitness.	329
9.6.4	Derivation of Penalty Formulae.	331
9.6.5	Selection.	334
9.6.6	Genetic Operator Action plus Update Routines.	335
9.6.7	Output from GASSOPm.	337
9.6.8	Results.	337
9.6.8.1	Discussion of Results.	343
10.	A Logical Search Approach.	344
10.1	Introduction.	344
10.2	Applicable Features of the Optimisation Literature.	344
10.3	A Logical Search Approach.	346
10.3.1	The Logical Search Algorithm.	346
10.3.2	Detailed Results from a Run of <i>Search_logic</i>	349
10.3.2.1	Further Results.	352
10.3.3	Order of Variables in <i>Search_logic</i>	353
10.3.2.1	Alternative Ordering Number 1, <i>altord1</i>	354
10.3.2.2	Alternative Ordering Number 2, <i>altord2</i>	355
10.3.4	Discussion of Results.	356

11.	A Statistically Designed Experiment.	358
11.1	Introduction.	358
11.2	Orthogonal Arrays.	359
11.3	Steps in a Statistically Designed Experiment.	360
11.4	Application of the Statistically Designed Experiment to optimise the HIPS.	360
11.4.1	Aim and Response Variable.	361
11.4.2	Choice of Factors and Levels.	361
11.4.3	Matrix Experiment.	362
11.4.4	Data Analysis.	367
11.4.5	Selecting Optimum Factor Levels.	369
11.4.6	The Influence of the Parameter P.	370
11.4.7	Consideration of the Constraints.	371
11.4.8	Follow-up Experiment.	376
11.4.8.1	Design of a Matrix Experiment.	377
11.4.8.2	Data Analysis.	380
11.4.8.3	Interpretation of Results.	383
11.4.8.4	Optimum Factor Settings.	384
11.4.9	Discussion of Results.	385
11.5	Application of Local Optimisation Techniques.	386
11.6	Conclusion and Summary.	388
12.	Conclusions and Future Work.	389
12.1	Conclusions.	389
12.2	Future Work.	391

NOMENCLATURE

A	$m \times n$ coefficient matrix
A	Number of AFFF pumps fitted (FDS)
A_E	Number of electrically powered AFFF pumps fitted (FDS)
A_P	Percentage capacity of the AFFF pumps fitted (FDS)
$A(t)$	Probability of availability at time t
b	m -dimensional column vector
C	Material type for corrosion resistant and non-corrosion resistant components
C_i	Minimal cut set i
C_{HP}	Cost per hour of manual work to carry out preventative maintenance
C_{HR}	Cost per hour of manual work to repair failure (dormant or spurious)
C_{HT}	Cost per hour of manual work to test the component
C_I	Initial cost
C_R	Number of hours manual work required to repair the component
C_S	Storage costs per component
C_{SP}	Cost of spares each time preventative maintenance is carried out
C_{SR}	Cost of spares for each repair carried out (dormant or spurious)
c^T	n -dimensional row vector
D	AFFF deluge valve type
E	Number of ESD valves fitted (HIPS)
F	Number of firewater pumps fitted (FDS)
F_E	Number of electrically powered firewater pumps fitted (FDS)
F_P	Percentage capacity of the firewater pumps fitted (FDS)
F_T	Firewater pump type fitted (FDS)
$F(t)$	Cumulative distribution function of the failure distribution
$f(t)$	Probability density function for the failure distribution
F_{SYS}	Probability of spurious trip occurrence
F_{SYS}^j	Spurious trip frequency of design point j
F_{SYS}^P	Predicted spurious trip frequency
F_{SYS}^T	Temporary Spurious Trip Frequency
g	Gradient vector
$G(t)$	Cumulative distribution function of the repair distribution

$g(t)$	Probability density function for the repair distribution
$G_i(q)$	Criticality function for component i (the probability that the system is in the failed state with respect to component i)
H	Number of HIPS valves fitted (HIPS)
H	Hessian Matrix
H_T	Number of hours manual work required to test the component
H_P	Number of hours manual work required to carry out preventative maintenance
$h(t)$	Hazard rate function or conditional failure rate
K_1	Number of pressure transmitters required to trip subsystem 1 (HIPS)
K_2	Number of pressure transmitters required to trip subsystem 2 (HIPS)
$m(t)$	Conditional repair rate
n	Number of minimal cut sets (unless otherwise stated)
N_1	Number of pressure transmitters fitted, subsystem 1 (HIPS)
N_2	Number of pressure transmitters fitted, subsystem 2 (HIPS)
P	Pressure transmitter type (HIPS and FDS)
p	Roulette wheel percentage
$P(C_i)$	Probability of occurrence of minimal cut set i
$P_i(t)$	Probability of being in state i at time t
$po_{xi}^1(q)$	Probability of the path section from the 1 branch node of X_i to a terminal 1 node (Probpost 1 branch)
$po_{xi}^0(q)$	Probability of the path section from the 0 branch node of X_i to a terminal 1 node (Probpost 0 branch_
$pr_{xi}(q)$	Probability of the path section from the root node to node X_i (Probprev)
$Q(t)$	Probability of unavailability at time t
Q_{SRS}	Top event probability or unavailability
Q'_{SRS}	Penalised system unavailability
Q_{SRS}^j	Unavailability of design point j (i.e. x^j)
Q_{SRS}^P	Predicted unavailability of the top event
Q_{SRS}^T	Temporary unavailability of the top event
$q_i(t)$	Unavailability of event (or component) j
Q_{MCSUB}	The minimal cut set upper bound approximate for system unavailability
$R(t)$	Component (or system) reliability

$s(x^{(k)})$	Search direction in the space of the current design variables from the current estimated minimum point $x^{(k)}$
V	Valve type (HIPS)
v	Constant repair rate
$v(t)$	Unconditional repair intensity
$V(0, t)$	Expected number of repairs in 0 to t
W	Water deluge valve type (FDS)
$w(t)$	Unconditional failure intensity
$W(0, t)$	Expected number of failures in 0 to t
$w_c(t)$	Unconditional failure intensity of cut set C
$w_j(t)$	Unconditional failure intensity of event (or component) j
$w_{SYS}(t)$	System unconditional failure intensity
x_i^j	Value of variable i at design point j
\mathbf{x}^0	Initial design vector
\mathbf{x}^T	Temporary design point
\mathbf{x}^P	Predicted minimum design point
β	Shape parameter of the Weibull distribution
η	Scale parameter of the Weibull distribution
λ	Constant failure rate
λ_D	Dormant failure rate
λ_s	Spurious failure rate
$\lambda(t)$	Conditional failure intensity
μ	Mean time to repair (MTTR)
τ	Mean time to failure (MTTF)
τ_D	Dormant mean time to repair
τ_s	Spurious mean time to repair
θ_1	Inspection interval for subsystem 1 (HIPS)
θ_2	Inspection interval for Subsystem 2 (HIPS)
θ_P	Maintenance test interval for the firewater and AFFF pump system (FDS)
θ_P^M	Maintenance test interval for the firewater and AFFF pump system, modified (FDS)
θ_{PM}	Preventative maintenance on components of wear out type (FDS)
θ_R	Maintenance test interval for the ringmain (FDS)
θ_R^M	Maintenance test interval for the ringmain, modified (FDS)

ABBREVIATIONS

AFFF	Aqueous Film-Forming Foam
AGREE	The Advisory Group on Reliability of Electronic Equipment
ANOM	Analysis of Means
ANOVA	Analysis of Variance
APRS	Adaptive Partitioned Random Search
BADD	Binary Algorithm Decision Diagrams
BDD	Binary Decision Diagram
BIP	Binary Integer Programming
CDF	Cumulative Distribution Function
CM	Corrective Maintenance
CMCAPL	Corrective Maintenance Costs of the AFFF Pumps and Lines
CMCDS	Corrective Maintenance Cost of the Deluge Skid
CMCFPL	Corrective Maintenance Costs of the Firewater Pumps and Lines
CMCR	Corrective Maintenance Cost of the Ringmain
CRS	Controlled Random Search
ELRAFT	An Efficient Logic Reduction Analysis of Fault Trees
ESD	Emergency Shutdown
FATRAM	FAult Tree Reduction AlgorithM
FDS	Firewater Deluge System
FMEA	Failure Mode and Effects Analysis
GA	Genetic Algorithm
GASSOP	Genetic Algorithm Safety System Optimisation Procedure
GASSOP_m	Genetic Algorithm Safety System Optimisation Procedure modified
GPM	Gradient Projection Method
GRGM	Generalised Reduced Gradient Method
GUI	Graphics User Interface
HE	Human Error
HIPS	High-Integrity Protection System
ICAPL	Initial Cost of the AFFF Pumps and Lines
ICDS	Initial Cost of the Deluge Skid
ICFPL	Initial Cost of the Firewater Pumps and Lines
ICR	Initial Cost of the Ringmain

ILP	Integer Linear Programming
IP	Integer Programming
ite	If-then-else
LCC	Life Cycle Cost of the FDS
LOC	Local Optimisation Algorithm
LP	Linear Programming
MCLP	Multiple Criteria Linear Programming
MDT	Maintenance Down Time
MFGP	Main Fire and Gas Panel
MILSTDS	The Department of Defence Adopted Standards in 1966
MIP	Mixed Integer Programming
MTI	Maintenance Test Interval
MTTF	Mean Time To Failure
MTTR	Mean Time To Repair
PDF	Probability Density Function
PLC	Computer Logic
PMC	Cost due to Preventative Maintenance
PRS	Partitioned Random Search
SCMC	Cost Incurred by the FDS due to Corrective Maintenance
SDE	Statistically Designed Experiment
SIC	Initial Cost of the FDS
SGA	Simple Genetic Algorithm
SPMC	Cost Incurred by the FDS due to Preventative Maintenance
STC	Cost Incurred due to System Testing of the FDS Eh Year
SUMT	Sequential Unconstrained Optimisation
TCAPL	Cost of Testing the AFFF Pumps and Lines
TCDS	Cost of Testing the deluge Skid
TCFPL	Cost of Testing the Firewater Pumps and Lines
TCR	Cost of Testing the Ringmain
TMEC	Cost due to Total Maintenance Effort on the FDS

CHAPTER 1

INTRODUCTION

1.1 Introduction to Reliability and Risk Assessment

Throughout the history of modern engineering failures of systems have been observed in every field. The impact of system failures varies from minor inconvenience and costs to personal injury, significant economic loss and death. The 1986 explosion of the space shuttle Challenger was as a result of the failure of the rubber O-rings that were used to seal the four sections of the booster rocket. In the same year, history's worst nuclear power reactor accident occurred in Chernobyl, USSR, which resulted in the leakage of radioactivity into the atmosphere. With the ever-increasing complexity of modern technology and increased public awareness to the potential hazards, the field of risk and reliability analysis has come into greater prominence.

Risk and reliability analysis techniques attempt to study, characterise, measure and analyse the failure and repair of systems in order to improve their operational use by increasing their design life, eliminating or reducing the likelihood of failure and safety risks, and reducing downtime, thereby increasing available operating time.

Reliability is not only an important part in the engineering design process, but also a necessary function in life-cycle costing, cost-benefit analysis, repair and facility resourcing, the determination of inventories and spare part requirements and the establishment of preventative maintenance programs. ✓

1.2 Background

Attention was first given to reliability quantification within the aircraft industry after the First World War. Initially the number of flights completed successfully by one and multi-engine aircraft were compared and each occurrence of failure thoroughly investigated. Attempts at reliability improvement were, however, applied on an ad

hoc basis with little consideration given to the expression of reliability in quantitative terms. As information regarding system failures accumulated the concept of expressing reliability in terms of the failure rate for a particular type of aircraft or system emerged. Requirements were subsequently developed in terms of accident rates per hours of flying time and, thus, quantification became embedded in the design process. By 1960, for automatic landing systems, the design criteria for a fatal landing risk of less than one per 10^7 landings could be established.

The development of mathematical reliability models began during World War II in Germany with a group working on the V-1 missile project. They abandoned the assumption that regarding reliability a chain could not be stronger than its weakest link, since it failed to account for the random failure when applied to a series system. This led to reliability being considered as the product of components joined in series (that is, using probability laws) and the further realisation that in a series system the reliability of the individual components must be much higher than the system reliability for satisfactory system performance.

The Advisory Group on Reliability of Electronic Equipment (AGREE) was created by the Department of Defence in 1952 due to excessive failure rates observed in electronic systems, especially in military equipment. Studies conducted by AGREE showed that it cost the Armed Services \$2 per year to maintain every dollars worth of electronic equipment. As a result the group advised that reliability criteria, in terms of failure rate, life expectancy, design adequacy and success prediction were applied to mass-produced components.

Simultaneously, following a series of missile accidents, concern for safety grew and in 1966 the Department of Defence adopted standards (MILSTDS) and required safety studies be performed at all the product development stages. Subsequently the UK Ministry of Defence also introduced similar standards, thereby significantly improving the reliability and maintainability of military equipment and systems.

Increased importance was also attached to safety in the nuclear field. The first extensive risk assessment of an industrial facility concerned nuclear power plants and was published in 1975. Professor Rasmussen and some 50 engineers analysed a vast

spectrum of nuclear accidents and then assessed their potential consequence to the public. Following the Three Mile Island accident the number of risk assessment studies of nuclear power plants increased.

Reliability and risk analysis techniques now encompass availability and maintainability. They are being adopted across many industrial sectors to control and manage major industrial hazards, on the one hand, and to design consumer goods, on the other.

1.3 Terminology

This section introduces some of the terminology used in reliability assessment. A more detailed discussion of these concepts is given in chapter 2.

Reliability is defined to be

the probability that a component or system will perform a required function for a given period of time when used under stated operating conditions.

Reliability is, therefore, concerned with how long the product continues to function once it has become operational. As such, the unit of time must be identified and the system should be observed under normal performance.

Maintainability is defined to be

the probability that a failed component or system will be restored to a specified condition within a period of time when maintenance is performed in accordance with prescribed procedures.

This concept applies only to repairable systems. That is, maintainability characterises the ability of a system to resume the performance of its function after a failure.

Availability is defined as

the probability that a component or system is performing its required function at a given point in time when used under stated operating conditions.

Availability may be interpreted as the percentage of time a component or system is operating over a specified time interval. It differs from reliability in that availability is the probability that the component is currently in the working state even though it may have previously failed and been restored to its normal operating condition. System availability can, therefore, never be less than system reliability. Availability measures the combined affect of both the failure and the repair process and as such, is an important characteristic of the system.

A failure is *the termination of the ability of a component or system to perform a required function*. By extension, a failure is said to have occurred when the ability of a component or system to perform a required function is altered.

A fault is *the inability of a component or system to perform its required function*. A fault is always the result of a failure. However, it may not be direct failure of the component or system itself.

1.4 Methods of Analysis

Various probabilistic methods are employed in reliability and risk assessment. All available methods can be broadly classified into inductive and deductive techniques. In the inductive procedures, the analysis begins at the component level, identifies the failure modes of each component, and establishes the effect of each component failure on the overall system. In the deductive techniques, the analysis starts with consideration of the potential hazards and works down through the system to identify the system hardware failures or human errors, which could have caused these hazards. The following gives a brief description of those techniques extensively applied in safety analysis.

Fault Tree Analysis

Fault tree analysis is an example of a top-down, deductive approach. A fault tree is a graphical representation of the relationship between certain specific events and the ultimate undesired, or top, event. Boolean logic is used to combine the causal events in the tree. A qualitative analysis consists of identifying the various combinations of events that will cause the top event to occur. A quantitative evaluation can then be performed by assigning failure probabilities to the basic events and computing the probability of the top event. Computer programs are available to carry out analysis on a constructed fault tree. Fault tree analysis is discussed in detail in chapter 2.

Over recent years research has focused on developments to improve the accuracy and efficiency of fault tree analysis. This resulted in the development of the Binary Decision Diagram (BDD) method, which is discussed further in chapter 2.

Failure Mode and Effect Analysis

The Failure Modes and Effects Analysis (FMEA) is an inductive or 'bottom-up' approach. The purpose is to identify the different failures and modes of failure that can occur at the component, subsystem and system levels and to evaluate the consequences of these failures. There are four main steps in the performance of an FMEA:

- 1) Definition of the system, its function and components.
- 2) Identification of the component failure modes and their causes.
- 3) Study of the failure mode effects.
- 4) Conclusions and recommendations.

The FMEA is usually performed during the conceptual and initial design phases of the system in order to assure that all possible failure modes have been considered and that proper provisions have been made to eliminate all the potential failures.

Markov Analysis

The kinetic tree theory used in fault tree methods requires the statistical independence of the basic events. Markov analysis provides a means of analysing the reliability and availability of systems whose components violate these assumptions of independence, such as standby redundancy, common cause failures, secondary failures and multiple component states.

Markov analysis looks at the system as being in one of several states. One possible state, for example, is that in which one component has failed but the other components continue to operate. The state transition diagram identifies all the discrete states of the system and the possible transitions between those states. Figure 1.1 illustrates the most simple state transition diagram in which the failure and repair behaviour of a single component is portrayed. The component has two states only, the working state (W) and the failed state (F). It is a repairable component and hence, the component may move from the failed state as well as moving from the working to failed state. These possible transitions are represented by the transition lines and arrows in the transition diagram. States in the transition diagram that cannot be left once the system has entered the state are termed absorption states. Markov diagrams representing non-repairable systems will contain absorption states.

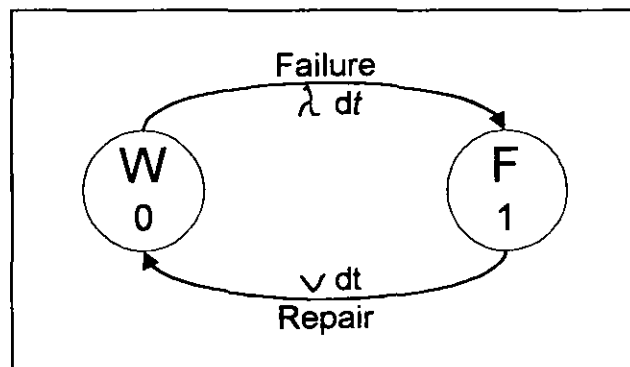


Figure 1.1 One Component State Transition Diagram

The state transition diagram should contain all possible states and transitions for the component or system. As a result, large systems are generally exceedingly complicated and difficult to construct. This is the major drawback of Markov methods.

The fundamental assumption in a Markov process is that the probability that a system will undergo transition from one state to another depends only on the current state of the system and not on any previous states the system may have experienced. That is, the system must be characterised by a lack of memory. In addition, the basic Markov approach considers only constant transition rates.

The state transition diagram in figure 1.1 may be translated into a set of linear equations, which represent the time-dependent behaviour of the state probabilities. These equations are

$$\frac{dP_0(t)}{dt} = -\lambda P_0(t) + \nu P_1(t) \quad (1.1)$$

$$\frac{dP_1(t)}{dt} = \lambda P_0(t) - \nu P_1(t) \quad (1.2)$$

where $P_i(t)$ is the probability of being in state i at time t , λ the component failure rate and ν the component repair rate.

In matrix form this is

$$\begin{aligned} [P_0(t) \ P_1(t)] &= [P_0(t) \ P_1(t)] \begin{bmatrix} -\lambda & \lambda \\ \nu & -\nu \end{bmatrix} \\ \mathbf{P} &= \mathbf{P}\mathbf{A} \end{aligned} \quad (1.3)$$

The square matrix \mathbf{A} , i.e. transition matrix, in equation (1.3) can be formulated directly from the transition diagram.

The first stage of the Markov analysis is to draw the state transition diagram for the respective system and hence, formulate the transition matrix. The transition matrix is then used to solve for the probabilities that the system is in the working and failed state. For a more detailed discussion of Markov Analysis the reader is referred to reference 5.

1.5 System Design

Failure of a safety system for a potentially hazardous industrial system or process may have severe consequences, possibly injuring members of the work force or public and occasionally resulting in loss of life. It is, therefore, imperative that such systems have a high likelihood of functioning on demand.

Typically the design of a safety system follows the traditional process of *preliminary design, analysis, appraisal* and *redesign*. If, following analysis, the initial design does not meet some pre-determined acceptability criteria for system unavailability, deficiencies in the design are removed and the analysis and appraisal stages repeated. Once the predicted system unavailability of the design reaches the acceptability criteria the design process stops and the system is adopted. For a system whose failure could result in fatality it could be accepted that a merely adequate level for system unavailability is not sufficient. The aim should be to produce the optimal performance attainable within the constraints imposed on resources.

It is highly unlikely that the design parameters can be manually selected such that optimal system performance can be achieved within the available resources. For this reason an optimisation algorithm integrated within the design process is required. The 'traditional' v's the 'optimal' approach is illustrated in figure 1.2

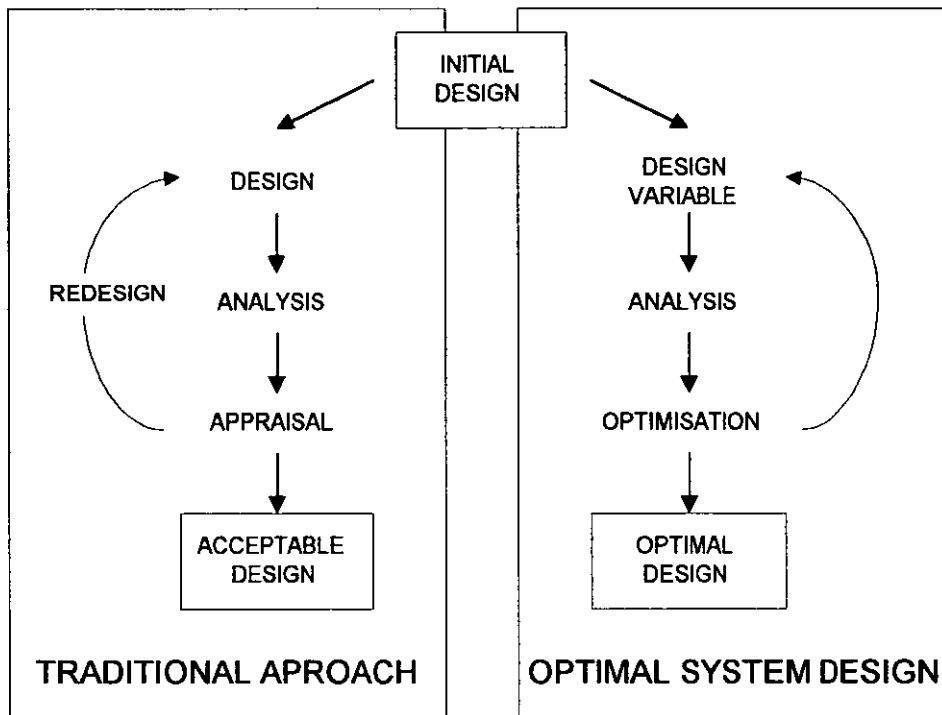


Figure 1.2 Traditional V's Optimal Design Process

1.6 Safety Design Considerations

Safety systems are designed to operate when certain conditions occur and to prevent their development into a hazardous situation. As such, there are certain features common to all safety systems. All safety systems have sensing devices which monitor for the occurrence of the triggering events. These sensors usually measure some process variable and transmit its current level to a controlling device. The controlling device determines whether the current situation is acceptable by comparing the input signal to a set point. When the sensed variable violates the set point the protective action is initiated. The protective action may either prevent a hazardous situation occurring or reduce its consequences.

The design engineer has a number of choices to make regarding the structure and operation of the safety system, which can influence reliability. These design options are described below.

Redundancy and Diversity Levels

The safety system must be designed to have a high likelihood of functioning on demand. Thus, single component failure should not be able to prevent the system from functioning. One means of achieving this is by incorporating redundancy or diversity into the system structure. Redundancy duplicates elements in the system while diversity involves the addition of a totally different means of achieving the same function. Both redundancy and diversity can be used at component level or sub-system level.

Increased levels of duplication in the form of fully redundant and fully diverse elements will also increase the number of spurious system trips. To counteract this, partial redundancy is commonly utilised where k of the n components fitted are sufficient for correct system functioning.

Component Selection

Each component selected for the design will be chosen from a group of possible alternatives. For every valve, relay, pressure sensor, etc., for which a selection is to be made there will be several choices, each with associated characteristics such as failure rate in each failure mode, cost and time taken for their scheduled maintenance. The design engineer has to decide how to trade-off these characteristics to give the most effective option for the overall system performance.

Maintenance Interval

Generally the time interval between preventative maintenance activities is assigned on an *ad hoc* basis. This is now changing with the more frequent application of reliability centred maintenance. Even so, the allocation of available maintenance effort is only considered after the system design has been finalised. Since component selection determines the time taken to maintain the system there are significant gains to be made by considering the maintenance frequency at the design stage.

Limitations on the Design Choices

There are many options open to the design engineer. To produce the most effective system these parameters need to be selected to give optimal system performance. The choice of design is not, however, unrestricted. Some of the possible design variations will not be feasible. Practical considerations of limits placed on resources will prevent a completely free choice of system design. Such considerations may include cost, limited maintenance effort, system weight, space limitations and other requirements of the system performance such as limits to the spurious trip occurrence rate.

1.7 The Design Optimisation Problem

There are many mathematical optimisation methods available. The application of methods such as linear programming, dynamic programming, non-linear programming and sequential unconstrained optimisation (SUMT) to reliability problems is explained by Tillman (Ref. 88). However, the features offered by the methods make them inappropriate for the industrial problems considered in this thesis.

Many of the optimisation methods require an explicit function (objective function), which defines how the characteristic to be minimised is related to the design variables. A variation in some of the design variables, such as redundancy levels, gives a discrete change in the structure of the system and prevents an objective function being deduced.

Design variables, which represent the levels of redundancy for a component or subsystem, are integer. If partial redundancy is incorporated through the use of voting systems, the number of successful channels to give the trip condition also needs to be chosen. This too is an integer variable for the design.

When a particular type of component is selected there will be a choice from several options which fulfil the same function. For example equipment supplied from different manufacturers. A variable corresponding to each potential piece of equipment can take the value 1 to indicate its selection and 0 to represent non-selection. This Boolean variable is again an integer over a restricted range. The optimisation scheme must be appropriate for integer variables.

Constraint forms, which are linear or non-linear functions of the design variables need to be incorporated in a general solution scheme. Constraints which determine aspects of the system performance (such as limitations on the expected number of spurious trips) which can only be evaluated by a full analysis of each potential design (implicit constraints) may also be specified.

1.8 Objectives of the Project

The objectives of the work programme were to:

- 1) Review existing reliability and risk assessment methods, which can be applied to industrial systems.
- 2) Review existing optimisation techniques and critically appraise their capabilities to industrial applications. Identify the features of each method which are appropriate for integer variables, can cope with implicit and explicit constraint forms of equality or inequality type, do not require linear forms of explicit constraints, do not require an explicit objective function and have an efficient analysis routine that is able to quickly evaluate a large number of potential designs.
- 3) Apply a Genetic Algorithm (GA) optimisation scheme to an industrial problem and investigate the effects of the GA parameters.
- 4) Critically appraise the results from the GA approach and modify the algorithm accordingly.

- 5) Test the GA approach further by application to a more complex industrial system. Give attention to the inclusion of wear out components.
- 6) Develop alternative algorithms, which can cope with the features of design optimisation identified in (2). Formulate the methods in a manner for efficient computer implementation.
- 7) Thoroughly investigate the application of the methods derived in (6) to an industrial problem. Critically appraise each algorithm and determine their most beneficial features as regards how efficiently and effectively they achieve the optimal design.

CHAPTER 2

FAULT TREE ANALYSIS

2.1 Background

System design is often based on traditional engineering analysis, experience and judgement. With rapid technology evolution and increasing system complexity, the additional use of more comprehensive analytic techniques in the development of safe, efficient, systems is appropriate.

A primary goal of reliability and safety analysis is to identify the causal relationships between events, which result in system failure, and to find ways of ameliorating their impact by system redesigns and upgrades. One of the most powerful and widely used analytic techniques in the field of reliability engineering is Fault Tree Analysis. Fault Tree Analysis provides a well-accepted means of improving or predicting the reliability and efficiency of complex systems.

Fault tree analysis was first conceived in 1961 by H. A. Watson of Bell Telephone Laboratories in connection with a US Air Force contract to study the Minuteman Missile Launch Control System. During the late 1960's and early 1970's the technique evolved and was applied to several studies carried out to obtain qualitative reliability information about relatively complex systems. Advances in the technique are covered in a comprehensive bibliography in fault tree analysis and its applications in reference 24.

2.2 General Description

The various types of system safety analysis procedures can be broadly classified into inductive and deductive techniques. In the inductive procedures analysis begins at the component level. Each relevant component failure is identified and its effect on overall system performance established. Thus, the inductive analysis works its way

up the system through higher assemblies until the complete system is analysed. Conversely, deductive techniques start the analysis with an enumeration of the potential hazards and work down through the system to identify the system hardware failures or human errors, which could have caused these.

Fault tree analysis is an example of a top-down, deductive technique, structured in terms of events rather than components. The perspective is on faults rather than reliability. A fault tree has a root, which represents a system failure mode. This undesired state is termed the top event. Once the top event is defined branches are extended to intermediate events directly responsible for its occurrence. Each level of the tree is, thus systematically deduced. Branches are terminated when basic events, which represent the lowest resolution of the tree, are encountered.

A single fault tree is not a model of all possible causes of system failure. Given a particular undesired state, a single fault tree reveals the possible combinations of component failures that may lead to this state. It may, therefore, be necessary to construct more than one fault tree during the assessment of any system to deal with further undesired states.

The fault tree acts as a visual tool. The fundamental concept of fault tree analysis is the translation of a physical system into a structured logic diagram. The tree is, thus, a graphical representation of the various parallel and sequential combinations of faults that lead to the occurrence of the top event. Following fault tree construction qualitative and quantitative analysis may be performed. Qualitative methods help in understanding the logical structure of the various failure modes of a system and their interrelationship. Quantitative methods, on the other hand, assign failure probabilities or unavailabilities to the basic events and predict the probability of the top event.

2.2.1 System Definition and Fault Tree Construction

Prior to Fault tree construction it is essential that the system definition is thoroughly addressed. Appropriate boundary conditions for the system's environment must be defined. An important boundary requirement is the top event. This determines the

comprehensiveness of the analysis. The limit of resolution to which the analysis will develop is determined by the choice of basic events and this determines the detail of the fault tree structure. In addition, system definition requires specification of the initial condition of each component and the particular time domain under consideration.

The fault tree logic diagram is constructed from two basic elements, 'gates' and 'events'. 'Gates' connect events according to their causal relations. A gate may have one or more input events but only one output event. The type of gate determines what combination of input events must occur for the output event to occur. Table 2.1 defines the gate symbols used in the fault trees developed during the work carried out in this thesis. Other gate types exist. For a comprehensive list see references 2 and 3. These additional gates only serve to reduce the size of the fault tree diagram and have to be expressed in terms of AND, OR and NOT logic prior to analysis. An equivalent minimal list of event symbols is shown in table 2.2.

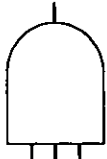
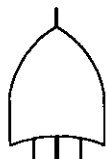
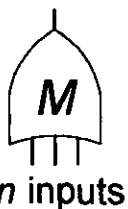
Gate Symbol	Gate Name	Casual Relations
	AND gate	Output event occurs if all input events occur simultaneously.
	OR gate	Output event occurs if at least one input event occurs.
	<i>m</i> -out-of- <i>n</i> gate (voting gate)	Output event occurs if <i>m</i> -out-of- <i>n</i> input events occur.

Table 2.1 Common Gate Symbols

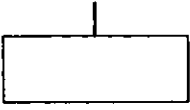
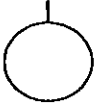
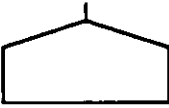
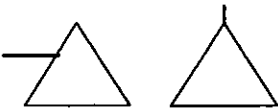
Event Symbol	Meaning of Symbol
 Rectangle	Intermediate event further developed by a gate
 Circle	Basic event
 House	House event. Either occurring or not occurring
 Triangles	Transfer Symbol

Table 2.2 Common Event Symbols

2.3 Qualitative Analysis

A fault tree represents the causal relations resulting in an undesired system state. Occurrence of the undesired state can result from many different combinations of events. Each unique combination is termed a system failure mode and may involve single or multiple component failures. Qualitative analysis consists of identification of these system failure modes.

System failure modes of a given fault tree are clearly defined in the concept of a *cut set*. A cut set is a collection of basic events, such that if they all occur the top event is guaranteed to occur.

When dealing with complex systems the number of possible failure modes is often very large. In order to restrict and simplify qualitative analysis, consideration is given only to the smallest combination of basic events, which cause the occurrence of

the top event of the fault tree. A *minimal cut set* is such that if any basic event is removed from the set the remaining events are no longer a cut set. A non-minimal cut set has redundant components.

A *path set* is the dual concept to the cut set. It is a collection of basic events such that if none of the events in the set occur, the top event is guaranteed not to occur. Similarly, a minimal path set is a path set such that if any basic event is removed from the set the remaining events collectively are no longer a path set.

A minimal cut set can be characterised by the number of basic events comprising it, which is termed its 'order'. In performing a qualitative evaluation more importance should be placed on first and second order minimal cut sets since it is expected that they are more likely to occur than cut sets having more events. For example, if a single event has a probability of occurrence in the order of 10^{-2} , then a double event cut set could be expected to have a probability of occurrence in the order of 10^{-4} . System performance can be dramatically improved if attention is focussed on eliminating the lower order cut sets.

Logical notation allows the top event to be represented in terms of the minimal cut sets,

$$T = C_1 + C_2 + \dots + C_n \quad (2.1)$$

where T represents the top event, C_i the minimal cut set i and n the number of minimal cut sets. The '+' symbol in the logic equation represents the OR operator.

The minimal cut sets are represented as an AND combination of basic events, i.e.

$$C_i = X_{i1} \cdot X_{i2} \dots X_{ij} \quad (2.2)$$

where X_{ij} is the basic event j in the i^{th} minimal cut set. The '.' symbol is used to represent logical AND in the equation.

2.3.1 Derivation of Minimal Cut Sets

Here, the discussion of qualitative assessment techniques is restricted to analysis of coherent fault trees, i.e. basic events represent failures and the logic diagram contains only AND and OR gates. A system which has failure modes involving both failure and success events is referred to as a non-coherent system. Non-coherent fault trees, therefore, contain AND, OR and NOT logic.

The first algorithms to determine minimal cut sets were based on Monte Carlo simulation techniques (Ref. 20). Later methods are deterministic. The basic idea behind deterministic methods is direct expansion of the top event in terms of the constituent basic events using Boolean algebra.

The fault tree OR gate logic represents the union of events. For example, an OR gate with two input events A and B and the output event Q can be represented by the equivalent Boolean expression $Q = A \cup B$ (or $Q = A + B$). The AND gate can be represented using the Boolean expression for intersection. Therefore, the Boolean equivalent of an AND gate with two inputs A and B would be $Q = A \cap B$ (or $Q = A.B$). The NOT gate is equivalent to the Boolean operation of complementation.

Methods to determine cut sets using the above expressions are classified into two groups, bottom-up and top-down. Both algorithms involve the expansion of Boolean expressions to obtain the Boolean logic expression for the top event and transform it into disjunctive normal (or sum of product) form. Each product then represents a minimal cut set. The substitution procedure in a bottom-up algorithm proceeds from the primary (or basic) events and works upward to the top event, while a top-down algorithm begins with the top event and works down.

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

1. Commutative laws:

$$A + B = B + A, \quad A . B = B . A$$

2. Associative laws:

$$(A + B) + C = A + (B + C), \quad (A \cdot B) \cdot C = A \cdot (B \cdot C)$$

3. Distributive laws:

$$A + (B \cdot C) = (A + B) \cdot (A + C), \quad A \cdot (B + C) = A \cdot B + A \cdot C$$

4. Identities:

$$A + 0 = A, \quad A + 1 = 1, \quad A \cdot 0 = 0, \quad A \cdot 1 = A$$

5. Idempotent laws:

$$A + A = A, \quad A \cdot A = A$$

6. Absorption law:

$$A + A \cdot B = A, \quad A \cdot (A + B) = A$$

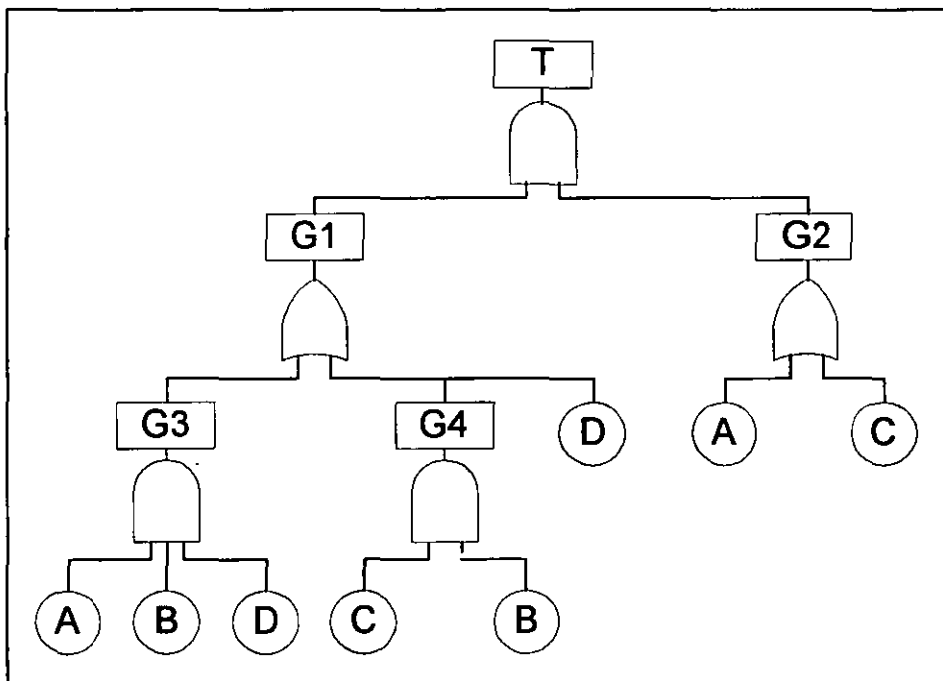


Figure 2.1 Example Fault Tree for Calculation of Minimal Cut Sets

To illustrate the top-down approach, consider the fault tree in figure 2.1. The top gate, T, is an AND gate with two gate inputs, G1 and G2. The first operation, therefore, substitutes T with G1.G2. Next, consider G1 which is an OR gate with two gate

inputs, G3 and G4, and one event input, D. The top event can now be expressed as $(G3 + G4 + D).G2$. This process is repeated, continuing down each branch until all references to gates are removed from the expression. Boolean laws enable simplification and reduction at each stage. The result is an expression for the top event defined solely in terms of basic events. Hence, application of the top-down algorithm gives

$$\begin{aligned} T &= G1 . G2 \\ &= (G3 + G4 + D) . G2 \\ &= (ABC + BC + D) . G2 \end{aligned}$$

By the absorption law $(ABC + BC = BC)$ this reduces to

$$T = (BC + D) . G2$$

Substitute for $G2 (= A + C)$

$$\begin{aligned} T &= (BC + D) . (A + C) \\ &= ABC + BCC + AD + CD \end{aligned}$$

as $C.C = C$

$$T = ABC + BC + AD + CD$$

Again, applying the absorption law $(BC + ABC = BC)$ gives

$$T = BC + AD + CD \tag{2.3}$$

The minimal cut sets $\{BC\}$, $\{AD\}$ and $\{CD\}$ can be extracted from the top event expression (2.3). Expression (2.3) represents the minimal cut sets of the fault tree in sum of products form.

It is clear from the example that the evaluation of fault trees by hand can be a formidable job. The need for a computerised technique to determine minimal cut sets

when considering larger, more complex trees becomes apparent. One of the earliest computer programs using the deterministic method is the PREP program developed by Vesely and Narum (Ref. 90). Fussel and Vesley (Ref. 32) developed an alternative top-down orientated algorithm called MOCUS. It is based on the fact that OR gates increase the number of cut sets whereas AND gates enlarge the size of the cut sets.

The top event label is inserted as the first element of a two-dimensional matrix. This matrix develops as lower level events are substituted. The result is ultimately a matrix consisting purely of primary events. The matrix formulation is constructed as follows:

1. Locate the top event in the first row and column of the matrix.
2. Continually scan the array and implement the following replacement:
 - a) Expand each OR gate vertically in terms of its gate inputs, thus increasing the number of cut sets. Duplicate all other elements in the initial row.
 - b) Expand each AND gate horizontally in terms of its gate inputs, thus enlarging the size of each cut set.Repeat the process above until all gates have been replaced and only primary events remain.
3. The final step involves reduction of the cut sets to their minimal form through application of Boolean laws.

Alternative Methods

Many alternative algorithms with unique features have been proposed to obtain minimal cut sets for the top event in a more efficient manner. The main goal of these algorithms is to obtain the minimal cut sets as quickly as possible using the smallest amount of core memory.

Semanders (Ref. 76) in the computer code ELRAFT (An Efficient Logic Reduction Analysis of Fault Trees) introduces the concept of prime number representation of

basic events for reduction of fault trees. This concept is useful in storing the cut sets and eliminating dependencies.

Semanders applies the conventional bottom-up procedure in conjunction with the unique feature that each basic event is assigned a prime number. A particular combination of basic events can, therefore, be expressed uniquely by a single number, which is equal to the product of prime numbers corresponding to the basic events. Consider for example, a fault tree with basic events A, B and C and cut sets {AB}, {ABC}. If events A, B and C are assigned the prime numbers 2, 3 and 5 respectively, cut set AB is represented by $2 \times 3 = 6$, cut set ABC by $2 \times 3 \times 5 = 30$. However, 6 is a factor of 30, thus 30 is dropped. Consequently, non-minimal sets are automatically eliminated and only those proving to be minimal are stored as a single number.

Wheeler et al (Ref. 91) introduce another unique numbering scheme using bit manipulation. This achieves greater efficiency in terms of computer effort and storage. A disadvantage of this approach may be the difficulty in organising basic events as binary bits for larger tree structures.

Fatram (Fault Tree Reduction Algorithm) introduced by Rasmuson and Marshall (Ref. 70) is a top-down algorithm similar to that of MOCUS. It differs in the order with which it considers each gate. Gates to be resolved are selected in such a manner that computer core requirements are minimised.

Initially all AND gates and OR gates with gate inputs are resolved and reduction techniques applied. OR gates with only basic events remain in the matrix. The next step deals with these remaining OR gates. The action taken depends on whether these basic event inputs are repeated. For a detailed description of the algorithm with worked examples the reader is referred to the paper.

An interesting addition to the FATRAM algorithm is a process termed 'weeding' or 'culling'. This process is based on the assumption that lower order cut sets largely determine system performance. Consequently, the effect exerted by cut sets above a certain order is insignificant. The weeding process ignores cut sets above a

predetermined 'ceiling' for cut set size, thus reducing ineffectual computer effort as cut sets are resolved.

A comparison between FATRAM and MOCUS carried out in the paper portrays impressive results. Twenty fault trees are compared in total with varying results. A fault tree with 25 gates and 36 basic events showed a particularly dramatic improvement. MOCUS determined 1184 cut sets in 7.36s, requiring 8288 words of core. FATRAM took 0.547s and used only 112 core words (Ref. 70).

Many algorithms, which aim to increase efficiency, incorporate innovative methods to handle repeated events, as inferred in the FATRAM approach above. As stated by Ziani, much effort is wasted creating then deleting non-minimal cut sets in conventional algorithms when repeated events are present (Ref. 50). In his paper Ziani introduces an algorithm with improvements based on reducing the number of set comparisons required to find minimal cut sets. The algorithm is based on the partitioning of cut sets into two families, those containing repeated events and others. The first step generates all cut sets by a conventional approach such as MOCUS. The cut sets are partitioned and the family containing repeated events considered next. Reduction is performed on this family alone, thus limiting the maximal number of comparisons. This reduced family of cut sets is then added to the 'other' group to create the full minimal set. In addition to its simplicity, a major advantage of this approach is its ability to combine with other algorithms, thus decreasing computer effort and enhancing performance.

2.3.3 Modularisation

The majority of algorithms for cut set derivation are based on Boolean reduction techniques. Even with modifications most of these methods get into computational difficulties with large trees. Large trees lead to an impractical number of Boolean indicators, which must be examined during Boolean reduction. Computational time is reduced when all combinations of Boolean indicators are being investigated.

'Weeding' can eliminate much unnecessary computer effort. However, order cut off procedures can leave out important minimal cut sets and hence, lead to considerable

errors. This is often exacerbated when the component failure data is markedly different. In addition, many algorithms are designed for special case scenarios and are not efficient for general fault tree analysis purposes.

An alternative approach, which has the potential to lead to more efficient computer programs and also identify subsystems of a fault tree which are intuitively meaningful, is modularisation. The theory behind this technique is that analysis algorithms can only guarantee efficiency if they exploit the simplest form of fault tree. It may be possible to decompose a large fault tree into several sub-trees. Then, even if the number of steps by the analysis grows exponentially, the analysis is applied to small trees.

A module of a fault tree is a set of at least two events which has only one output to the rest of the tree and no inputs from the rest of the tree. These modules can then be used to split the analysis problem into disjoint parts. Once a module has been analysed a reduced system can be defined in which the module is treated as a basic event, thus reducing the size of the remaining analysis problem. A more comprehensive review of modularisation can be found in reference 15.

Modularisation proves particularly effective in simplifying analysis on certain trees when used in conjunction with conventional methods. Problems arise, however, in creating effective techniques to find the modules of a general fault tree. In addition, certain trees do not contain modules and thus, cannot be simplified in this way.

2.4 Quantitative Assessment

This section develops the theory and techniques required to predict the reliability performance of a system. Computerised quantitative assessments are based on the analytic methodology called 'Kinetic Tree Theory' introduced by Vesely in 1970 (Ref. 89). Kinetic Tree Theory requires that all basic events in the tree structure are independent. It has the ability to model primary events, which have time-dependent failure probabilities and requires that all the minimal cut sets of the fault tree are known. Quantitative assessments yield values for the system reliability, the system

availability, the expected number of system failures over a given time period and the system failure rate, the inverse of which depicts the mean time to failure of the system. Additional probabilistic quantities, such as the mean time between repairs are simply obtained from the above characteristics.

The measures of importance of individual failure events and cut sets of a fault tree are another feature of quantitative fault tree analysis. Whilst the evaluation of the top event provides system reliability and availability information, probabilistic importance computation can generate a numerical ranking to assess weaknesses in the system.

2.4.1 Component Failure Parameters

An accurate description of a component's failure and repair cycle is central to the identification of system hazards, since these are caused by combinations of component faults. Thus, basic events related to system components with binary events, i.e. normal and failed, must first be quantified.

It is assumed that at any given time a component is either working or failed. A new component is automatically assumed to be in the working state and this is subject to change as time evolves. When the component fails it enters the failed state. A non-repairable component will remain in this state forever. A repairable component will return to the working state following repair and continue in this two-state cycle. Three common assumptions are made to ease mathematical analysis: transition from one state to the next is instantaneous, only one state transition can occur in a sufficiently small interval of time, a repaired component is considered to be as good as new.

The Failure Process

Four related probability functions: the reliability function, the cumulative distribution function, the probability density function and the hazard rate function can be used to

compute component reliabilities. Specifying any one of these will uniquely and completely characterise the failure process. Various summary measures of reliability can then be determined.

Let $R(t)$ denote component reliability, that is the probability that a component will function over some time period. Thus,

$$R(t) = P[T \geq t] \quad (2.4)$$

where t is the designated period of time for the items operation and T the time to failure of the item. Also, $R(t) \geq 0$, $R(0) = 1$ and $\lim_{t \rightarrow \infty} R(t) = 0$.

Let $F(t)$ denote the cumulative distribution function (CDF) of the failure distribution, that is the probability that a component fails at some time prior to t .

$$F(t) = P[T < t] \quad (2.5)$$

where $F(0) = 0$ and $\lim_{t \rightarrow \infty} F(t) = 1$.

The corresponding probability density function (PDF), $f(t)$, is thus

$$f(t) = \frac{dF(t)}{dt} \quad (2.6)$$

This function describes the shape of the failure distribution and has the two properties

$$f(t) \geq 0 \quad \text{and} \quad \int_0^{\infty} f(t) dt = 1 \quad (2.7)$$

The reliability function and the CDF represent areas under the curve defined by $f(t)$, giving rise to the following relations

$$F(t) = \int_0^t f(t') dt' \quad (2.8)$$

$$\text{and } R(t) = \int_t^{\infty} f(t') dt' \quad (2.9)$$

$$\text{where } R(t) = 1 - F(t) \quad (2.10)$$

The mean time to failure (MTTF) is defined by

$$\text{MTTF} = E(T) = \int_0^{\infty} t f(t) dt \quad (2.11)$$

The conditional failure rate or hazard rate function, $h(t)$, describes the instantaneous transition to the failed state. The hazard rate is a measure of the first and only failure of an item in the next instant of time, given that the item is presently operating.

$$\begin{aligned} h(t) dt &= \frac{P[\text{component fails between } [t, t + dt]]}{P[\text{no failures in } [0, t]]} \\ &= \frac{f(t) dt}{1 - F(t)} = \frac{f(t) dt}{R(t)} \end{aligned} \quad (2.12a)$$

Integrating both sides of equation (2.12a) gives:

$$\begin{aligned} \int_0^t h(t') dt' &= \int_0^t \frac{f(t') dt'}{1 - F(t')} = -\ln[1 - F(t)] \\ \therefore F(t) &= 1 - \exp\left(-\int_0^t h(t') dt'\right) \end{aligned} \quad (2.12b)$$

In practice $h(t)$ often exhibits a bathtub shape and is referred to as a bathtub curve (Ref.'s 28,41), as shown in figure 2.2.

The Repair Process

Repairable items undergo repair and are, thus returned to their normal state. Parameters representing the repair process are derived in the same manner as for the failure process. Time is now measured from the instant the component failed. Thus,

$$G(t) = [\text{a failed component is repaired in } [0, t]]$$

The corresponding density function is

$$g(t) = \frac{dG(t)}{dt} \quad (2.13)$$

and conditional repair intensity

$$m(t) = \frac{g(t)}{1 - G(t)} \quad (2.14)$$

Integrating gives

$$G(t) = 1 - \exp\left(-\int_0^t m(t') dt'\right) \quad (2.15)$$

then the mean time to repair (MTTR) is defined by

$$\text{MTTR} = \int_0^{\infty} t g(t) dt \quad (2.16)$$

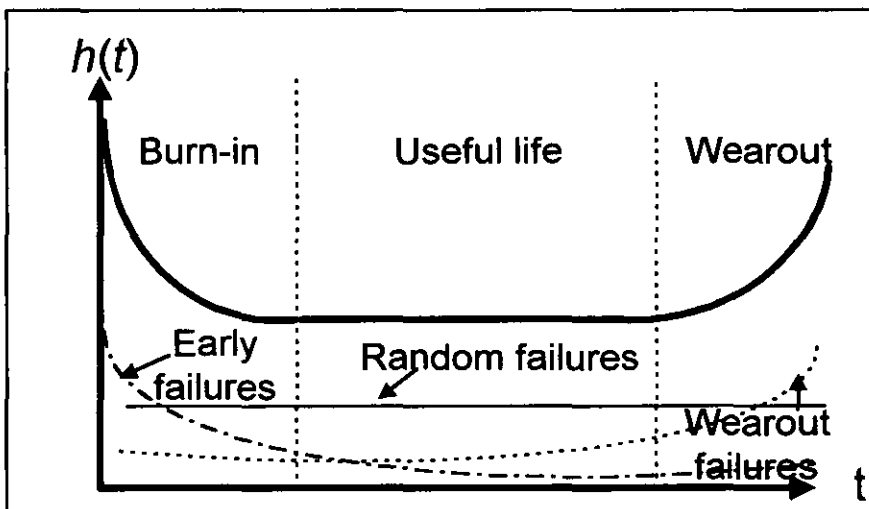


Figure 2.2 The Bathtub Curve

The Whole Process

The lifetime of a repairable component enters a continuous cycle consisting of repetitions of the failure-to-repair and the repair-to-failure process. For repairable components a relevant measure of performance is the availability as opposed to the reliability. More precisely, *availability*, $A(t)$, is defined as the probability that a component is performing its required function at a given point in time. Conversely, *unavailability*, $Q(t)$, is the probability that a component is in the failed state at time t , where

$$Q(t) = 1 - A(t) \quad (2.17)$$

The entire life cycle of a component needs to be modelled to calculate component availability and attention given to additional parameters required to achieve this.

The hazard rate is applicable only to non-repairable components as it relies on the component having worked continuously from zero to time t . To model the whole process two further parameters are introduced: the unconditional and the conditional failure intensity.

The *unconditional failure intensity*, $w(t)$, is the probability that a component fails per unit time at t given that it was working at $t = 0$.

The *conditional failure intensity*, $\lambda(t)$, is the failure rate based on those components which are working at time t . In contrast, $w(t)$ is based on the whole population.

Conditional probabilities can be used to relate λ and w

$$\begin{aligned} \lambda(t)dt &= P[\text{a component fails in } [t, t + dt) \text{ given that it was working at } t = 0] \\ &= \frac{P[\text{a component fails in } [t, t + dt)]}{P[\text{component is working at } t]} \\ &= \frac{w(t)dt}{1 - Q(t)} = \frac{w(t)dt}{A(t)} \end{aligned} \quad (2.18)$$

Repair parameters must also be considered. The *unconditional repair intensity*, $v(t)$, is the probability that a failed component is repaired per unit time at t , given that it worked at $t = 0$. The *conditional repair intensity*, $\mu(t)$, is the probability that a component is repaired per unit time at t given that it is failed at t and was working at time zero, i.e. $\mu(t)$ is based only on those components in the failed state at time t , whereas v is based on the whole population.

Using the density functions $f(t)$ and $g(t)$, integral equations can be developed whose solution yields the unconditional failure and repair intensities. These integrals are derived in Andrews and Moss (Ref. 5) and are

$$w(t) = f(t) + \int_0^t f(t-u)v(u)du \quad (2.19)$$

$$v(t) = \int_0^t g(t-u)w(u)du \quad (2.20)$$

By solving the simultaneous equations (2.19) and (2.20) the expected number of failures and repairs can be determined over any time period from the following equations

$$W(0, t) = \int_0^t w(u)du \quad (2.21)$$

$$V(0, t) = \int_0^t v(u)du \quad (2.22)$$

In turn, these expectations can be used to derive component unavailability, giving:

$$Q(t) = W(0, t) - V(0, t) \quad (2.23)$$

As described in Andrews and Moss (Ref. 5).

2.4.1.1 Failure Rate Models

Several probability models are useful in describing a failure process. The most common models are based upon the exponential, Weibull, normal and lognormal probability distributions. Those utilised in the analysis processes detailed later are described below.

The Exponential Reliability Function

A failure distribution that has a constant failure rate has failure times given by the exponential probability distribution. A great deal of reliability analysis is carried out using the assumption that components are operating during their 'useful life' period, as represented by the bathtub curve in figure 2.2. This 'useful life' period exhibits a reasonably constant rate of failure, characterised by random failures of the component. The exponential distribution is, therefore, one of the most important and widely used reliability distributions.

To develop the Constant failure rate (CFR) model for the failure process assume $h(t) = \lambda$, using equations (2.12b), (2.10) and (2.6) with $t \geq 0$

$$F(t) = 1 - e^{-\lambda t} \quad (2.24)$$

$$\text{and } R(t) = e^{-\lambda t} \quad (2.25)$$

$$\text{hence } f(t) = \lambda e^{-\lambda t} \quad (2.26)$$

Using equation (2.11) The mean of the distribution can be found yielding the mean time to failure (MTTF or μ)

$$\mu = \frac{1}{\lambda} \quad (2.27)$$

Similarly, for the repair process with constant repair rate ν and using equation (2.16) the mean time to repair (MTTR or τ) is

$$\tau = \frac{1}{\nu} \quad (2.28)$$

If the constant failure and repair rates apply then the solution of equations (2.19) and (2.20) can be performed using Laplace transforms to yield the unconditional failure and repair intensities

$$w(t) = \frac{\lambda \nu}{\lambda + \nu} - \frac{\lambda^2}{\lambda + \nu} \exp[-(\lambda + \nu)t] \quad (2.29)$$

$$v(t) = \frac{\lambda \nu}{\lambda + \nu} - \frac{\lambda \nu}{\lambda + \nu} \exp[-(\lambda + \nu)t] \quad (2.30)$$

Further substitution into equation (2.23) yields

$$Q(t) = \frac{\lambda}{\lambda + \nu} \{1 - \exp[-(\lambda + \nu)t]\} \quad (2.31)$$

If the component has been operable for a long period of time, i.e. $t \rightarrow \infty$, the stationary or steady-state unavailability can be used and is estimated by

$$Q = \frac{\lambda}{\lambda + \nu} = \frac{\tau}{\mu + \tau} \quad (2.32)$$

The assumption that the MTBF is much greater than the MTTR results in a very simple pessimistic approximation of component unavailability

$$Q = \lambda \tau \quad (2.33)$$

Standby or safety systems lie in the dormant state and function only when the demand requires. A dormant system failure may go undetected for a substantial period of time, the fault being noticed only when demand requires the system to function, or the

system is maintained or tested. Repair of such systems relies on the time between system inspection intervals as well as the MTTR, that is the detection time and the repair time. If a system is inspected every θ time units, the average unavailability is given as

$$Q_{AV} = \lambda \left(\tau + \frac{\theta}{2} \right) \quad (2.34)$$

The Weibull Distribution

Weibull is an exceedingly useful probability distribution in reliability. In contrast to the exponential distribution, Weibull is able to model increasing and decreasing failure rates. Thus, lending itself to the first (wear-in) and last (wear-out) phases of the bathtub curve in addition to the useful life period. It is characterised by a hazard rate function of the form

$$h(t) = \frac{\beta}{\eta} \left(\frac{t}{\eta} \right)^{\beta-1}, \quad \eta > 0, \beta > 0, t \geq 0 \quad (2.35)$$

Beta (β) is referred to as the shape parameter. Modifying the value of β has a dramatic effect on the PDF. Eta (η) is known as the scale parameter (or characteristic life) and influences both the mean and spread of the distribution.

The CDF and PDF using Weibull gives

$$F(t) = 1 - \exp \left[- \int_0^t \frac{\beta}{\eta} \left(\frac{t'}{\eta} \right)^{\beta-1} dt' \right] \quad (2.36)$$

$$f(t) = \frac{\beta}{\eta} \left(\frac{t}{\eta} \right)^{\beta-1} \exp - \left(\frac{t}{\eta} \right)^{\beta} \quad (2.37)$$

For $\beta < 1$, the hazard rate applies to the burn-in phase . When $\beta = 1$, the hazard rate is constant and the distribution is identical to the exponential with $\lambda = 1/\eta$. For $\beta > 1$, the hazard rate applies to the wear-out phase. For $\beta \geq 3$, the PDF tends toward a normal distribution, thus portraying symmetry. When $1 < \beta < 3$, the Weibull portrays its typically skewed distribution.

The disadvantage of the Weibull distribution is the increase in complexity when analysing the whole failure and repair process. Introducing time-dependent failure models creates unconditional transition equations, which are too complex to solve using Laplace transforms. The analyst usually has to resort to numerical methods to achieve component unavailability. This are is covered in more detail in chapter 9.

2.4.2 System Failure Parameters

2.4.2.1 Top Event Quantification

Once failure probabilities are assigned to each basic event and minimal cut sets have been obtained, quantification of the top event can be instigated. The top event probability or unavailability is denoted as $Q_s(t)$.

The top event exists when at least one minimal cut set exists. If the basic events are independent this gives rise to what is termed the inclusion-exclusion formula, where the top event probability, Q_{EXACT} , is defined as

$$Q_{EXACT} = \sum_{i=1}^n P(C_i) - \sum_{i=2}^n \sum_{j=1}^{i-1} P(C_i \cap C_j) + \dots + (-1)^{n-1} P(C_1 \cap C_2 \cap \dots \cap C_n) \quad (2.38)$$

where n is the number of minimal cut sets. In the general case

$$P(C_1 \cap C_2 \cap \dots \cap C_n) = P(C_1)P(C_2) \dots P(C_n)$$

To demonstrate the use of the inclusion-exclusion expansion the top event probability of the fault tree in figure 2.3 is derived.

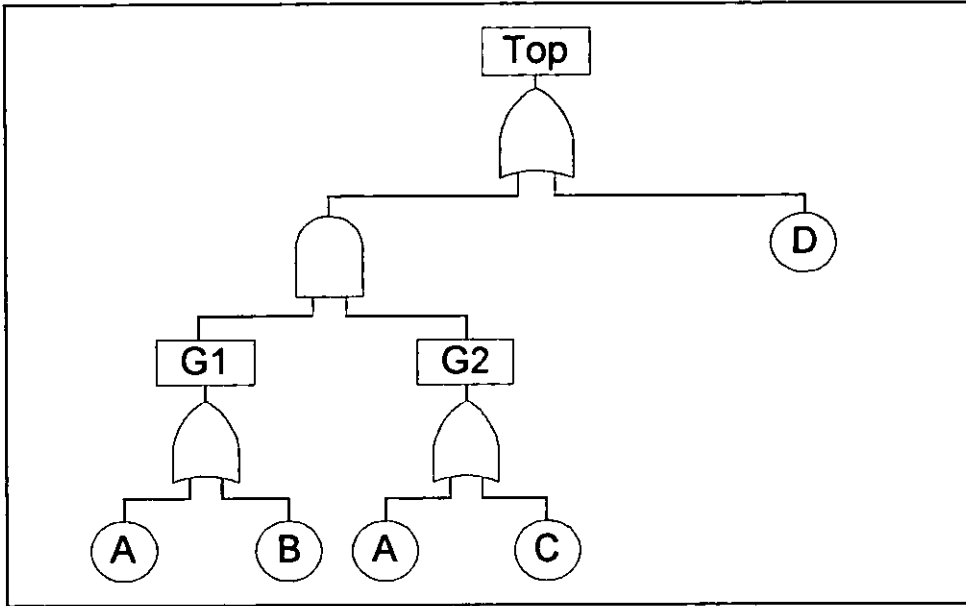


Figure 2.3. Example Fault Tree to Calculate Top Event Probability

The fault tree in question has minimal cut sets $\{A\}$, $\{BC\}$ and $\{D\}$. The top event can therefore be expressed as

$$T = A + BC + D$$

Each basic event is assigned a probability of 0.2 and the minimal cut sets denoted by C_1 , C_2 and C_3 respectively. The cut sets, therefore, have probabilities of failure 0.2, 0.04 and 0.2. Expansion of equation (2.38) gives

$$\begin{aligned} Q_s(t) &= [P(C_1) + P(C_2) + P(C_3)] - [P(C_1 \cap C_2) + P(C_1 \cap C_3) + P(C_2 \cap C_3)] \\ &\quad + [P(C_1 \cap C_2 \cap C_3)] \\ &= [0.2 + 0.04 + 0.2] - [0.008 + 0.04 + 0.008] + [0.0016] \\ &= [0.44] - [0.02] + [0.0016] \\ &= 0.4216 \end{aligned}$$

For a small system such as this the top event probability can be calculated exactly. Fault trees representing real life scenarios can have hundreds and thousands of minimal cut sets. As can be seen from this simple example, as the fault tree complexity increases calculations involving the inclusion-exclusion expansion soon become intolerable. For this reason approximation techniques are introduced to estimate lower and upper bounds for system unavailability.

Lower and Upper Bounds for System Unavailability

The inclusion-exclusion principle is based on alternately adding and subtracting each successive term. Truncating the expansion after a negative term results in a lower bound for system unavailability, conversely truncation after a positive term gives an upper bound. Thus

$$\sum_{i=1}^n P(C_i) - \sum_{i=2}^n \sum_{j=1}^{i-1} P(C_i \cap C_j) \leq Q_{EXACT} \leq \sum_{i=1}^n P(C_i) \quad (2.39)$$

i.e. Lower bound \leq Exact \leq Upper bound

As is shown in the above example, the contribution from each term becomes less significant as the expansion progresses. The rarer the basic event probabilities, the smaller the number of terms required to give an accurate upper or lower bound approximation. The simplest approximation, i.e. the single term upper bound, is the Rare Event approximation. It is so called as it is only accurate if the component failure events are rare.

The Minimal Cut Set Upper Bound

A more accurate upper bound approximate can be obtained using the minimal cut set upper bound. The top event occurs if at least one minimal cut set occurs. Conversely, if no minimal cut sets occur the system does not fail. Hence,

$$Q_s(t) = P(\text{at least one minimal cut set occurs}) \\ = 1 - P(\text{no minimal cut sets occur})$$

where

$$P(\text{no minimal cut sets occur}) \geq \prod_{i=1}^{N_c} P(\text{minimal cut set } i \text{ does not occur})$$

N_c being the number of minimal cut sets. The equation portrays equality when basic events are not common to more than one minimal cut set. Thus,

$$Q_s(t) \leq 1 - \prod_{i=1}^{N_c} [1 - P(C_i)] = Q_{\text{MCSUB}} \quad (2.40)$$

2.4.2.2 Unconditional System Failure Intensity

The calculation of minimal cut set probabilities has already been considered. The expected number of times the cut set occurs per unit time at t , $w_c(t)$, is given by

$$w_c(t) = \sum_{j=1}^N \left(w_j(t) \prod_{\substack{i \neq j \\ i=1}}^N q_i(t) \right) \quad (2.41)$$

where, N is the number of events in the cut set, $w_j(t)$ is the unconditional failure intensity of event j and $q_i(t)$ the unavailability of event i .

A further quantitative system measure is the expected number of top event occurrences over a given period of time. The frequency of the top event from zero time to time t is given in equation (2.21), where w is the unconditional failure intensity. The unconditional failure intensity represents the expected number of times the top event occurs at time t . Thus, $w_s(t)dt$ is the expected number of top event occurrences in $[t, t + dt)$.

A means to achieve $w_s(t)$ uses the criticality function for each component i , denoted by $G_i(q)$ (also known as Birnbaum's measure of importance). The criticality

function is defined as the probability that the system is in the critical state with respect to component i . Failure of component i , therefore, transforms the system from the working to the failed state.

A means to evaluate the criticality function requires determination of the probability that the system fails only if component i fails. This is the probability that the system fails with component i failed minus the probability that the system fails with component i working. This gives

$$G_i(q) = Q(1_i, q) - Q(0_i, q) \quad (2.42)$$

where

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

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

Alternatively

$$G_i(q) = \frac{\partial Q(q)}{\partial q_i} \quad (2.43)$$

where

$$\frac{\partial Q(q)}{\partial q_i} = \frac{Q(1_i, q) - Q(0_i, q)}{1 - 0}$$

Subsequently, w_s is evaluated using

$$w_{sys}(t) = \sum_i G_i(q) w_i(t) \quad (2.44)$$

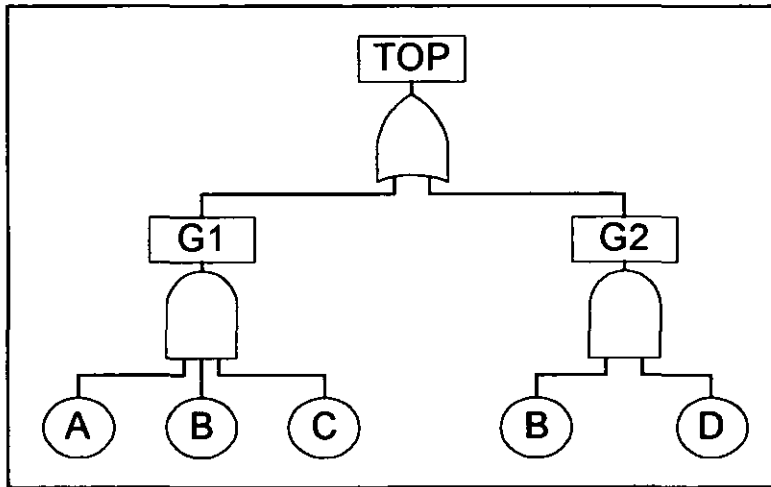


Figure 2.4 Example Fault Tree to Calculate the Unconditional Failure Intensity

As an example the unconditional failure intensity and expected number of system failures for the fault tree in figure 2.4 are derived. The minimal cut sets of the tree are {BD} and {ABC}, i.e. C_1 and C_2 respectively. Component failure data is given in table 2.3.

Component	$\lambda(\text{hours})$	$\tau(\text{hours})$
A	3×10^{-4}	12
B	9×10^{-5}	16
C	1.1×10^{-4}	10
D	4×10^{-4}	12

Table 2.3 Component Failure Data

Consideration is first given to component failure parameters. Using the steady-state unavailability approximation, $q = \lambda\tau$, component unavailabilities are

$$q_A = 3.6 \times 10^{-3}$$

$$q_B = 1.44 \times 10^{-3}$$

$$q_C = 1.1 \times 10^{-3}$$

$$q_D = 4.8 \times 10^{-3}$$

Using $w = \lambda(1 - q)$, the component unconditional failure intensities are

$$w_A = 2.9892 \times 10^{-4}$$

$$w_B = 9.8704 \times 10^{-5}$$

$$w_C = 1.0987 \times 10^{-4}$$

$$w_D = 3.9808 \times 10^{-3}$$

Table 2.4 to 2.7 specify each possible combination of component states where A, B C and D are the critical components respectively. A '*' verifies system failure as a result of the respective combination and the associated probability is stated in 'Prob'.

Component State			System State	
B	C	D	Crit	Prob
W	W	W		
W	W	F		
W	F	W		
W	F	F		
F	W	W		
F	W	F	*	6.91e-6
F	F	W	*	1.58e-6
F	F	F	*	1.6e-9

Table 2.4 Component A Critical

Component State			System State	
A	C	D	Crit	Prob
W	W	W		
W	W	F	*	4.8e-3
W	F	W		
W	F	F	*	5.28e-6
F	W	W		
F	W	F	*	1.73e-5
F	F	W	*	3.96e-6
F	F	F	*	1.9e-8

Table 2.5 Component B Critical

Component State			System State	
B	C	D	Crit	Prob
W	W	W		
W	W	F		
W	F	W		
W	F	F	*	6.91e-6
F	W	W		
F	W	F		
F	F	W	*	5.18e-6
F	F	F	*	1.21e-5

Table 2.6 Component C Critical

Component State			System State	
A	C	D	Crit	Prob
W	W	W		
W	W	F		
W	F	W	*	1.44e-3
W	F	F	*	1.58e-6
F	W	W		
F	W	F		
F	F	W	*	3.96e-6
F	F	F	*	5.7e-9

Table 2.7 Component D Critical

Using tables 2.4 to 2.7 $G_i(q)$ for each component is

$$G_A(q) = 85 \times 10^{-6}$$

$$G_B(q) = 4.92 \times 10^{-3}$$

$$G_C(q) = 1.21 \times 10^{-3}$$

$$G_D(q) = 1.44 \times 10^{-3}$$

Using equation (2.44) the unconditional failure intensity is

$$\begin{aligned} w_s(t) &= ((2.99 \times 10^{-4}) \times (8.5 \times 10^{-6})) + ((9.87 \times 10^{-5}) \times (4.92 \times 10^{-3})) \\ &\quad + ((1.1 \times 10^{-4}) \times (1.21 \times 10^{-3})) + ((3.98 \times 10^{-3}) \times (1.44 \times 10^{-3})) \\ &= 2.54 \times 10^{-9} + 4.86 \times 10^{-7} + 1.33 \times 10^{-9} + 5.73 \times 10^{-6} \\ &= 6.22 \times 10^{-6} \end{aligned}$$

The above calculations consider only the time interval $[t, t + dt)$. To find the expected number of system failures over say 10 years (i.e. 87600 hours), equation (2.21) gives

$$\begin{aligned} W(0,87600) &= \int_0^{87600} w_s(t) dt \\ &= 6.2207 \times 10^{-6} \times 87600 \\ &= 0.0545 \end{aligned}$$

2.5 Faulttree+

A fault tree analysis software package called FAULTTREE+ (Version 4.1) provides both graphical capabilities to construct the failure logic model and comprehensive analysis functions. The program is controlled via a Graphics User Interface (GUI), which enables functions and edit operations to be predominantly controlled using a mouse device.

Faulttree+ provides interactive facilities for constructing, editing and plotting fault trees. The program will allow the user to construct single large fault trees. A useful feature, however, is that the user may divide the tree into 'pages' by attaching 'page'

markers on selected gates within a single large tree. The tree is, therefore, more manageable, easier to view and less prone to human error.

2.5.1 Basic Event Model Types

Prior to analysis of the fault tree, data must be entered for each included primary event. Before data input the primary event type must be determined. The allowed event types are basic, conditional, undeveloped, dormant and house. In addition one of four alternative quantitative failure models for each event must be selected, where the various mathematical expressions that are available to calculate the component unavailability constitute the component's 'model type'. These models are discussed in detail below.

Fixed Unavailability and Unconditional Failure Intensity

The fixed model specifies a fixed, time-independent unavailability (q_i) for an event. A fixed unconditional failure intensity (w_i) may also be specified..

The system unavailability and total down time are dependent only on the component unavailability and so the component's unconditional failure intensity is not necessary to determine the parameter. The unconditional failure intensity will be required for basic events in the spurious system failure fault tree, where top event frequency of occurrence is required.

Constant Failure and Repair Rate

This model specifies fixed failure and repair rates. It is assumed that a failure event is instantaneously revealed and repair instigated immediately. The event unavailability and unconditional failure intensity are then calculated from the following expressions:

$$q(t) = \frac{\lambda}{\lambda + \nu} \{1 - \exp - (\lambda + \nu)t\} \quad (2.45)$$

$$w(t) = \lambda \{1 - q(t)\} \quad (2.46)$$

where $q(t)$ and $w(t)$ are the unavailability and unconditional failure intensity at time t , λ the constant failure rate and ν the constant repair rate. Non-repairable components are given a repair rate of zero.

Mean Time to Failure and Repair

The mean time to failure (MTTF), μ , and the mean time to repair (MTTR), τ , parameters are given and used to calculate the constant failure and repair rate using equations (2.26) and (2.27) respectively. The failure and repair rates are then used in equations (2.45) and (2.46), defined in the previous model, to evaluate $q(t)$ and $w(t)$ respectively.

Dormant Failure with Periodic Inspection

This model is appropriate for dormant components whose failure remains unrevealed until periodic inspection is performed. It produces an average unavailability and unconditional failure intensity from the constant failure rate (λ), the MTTR (τ) and inspection interval parameter (θ). The model takes an average value of the typical saw-toothed behaviour with period of θ , see figure 2.5.

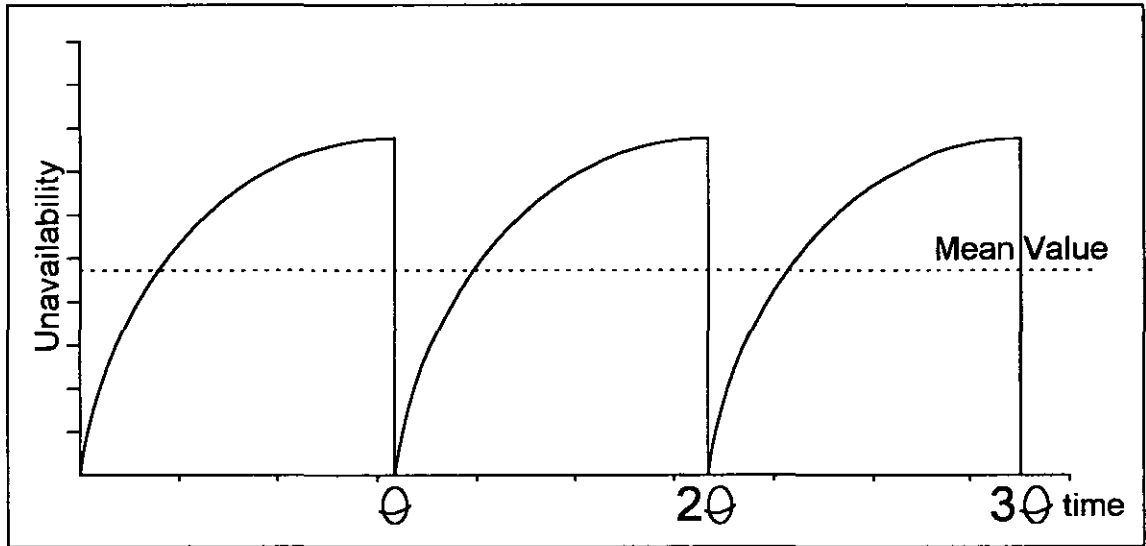


Figure 2.5 Dormant Failure with Periodic Inspection

The unavailability and unconditional failure intensity for each basic event are given by

$$q_{AV} = \lambda \left(\tau + \frac{\theta}{2} \right) \quad (2.47)$$

$$w = \lambda(1 - q_{AV}) \quad (2.48)$$

Note a more accurate equation for (2.47), when the mean time to repair is negligible, is

$$q_{AV} = 1 - \frac{1}{\lambda\theta} (1 - e^{-\lambda\theta}) \quad (2.49)$$

2.5.2 Analysis in Faulttree+

Having constructed the fault tree and entered all primary event input data format the 'analysis' can be performed. On selection of the analysis option two ascii files describing the tree structure and the quantitative data are automatically created. These files are given the same base name as the current graphics file but with

extensions '.ats' (tree structure) and '.aqd' (quantitative data). (The format of each ascii file is given in Appendix I).

To illustrate the analysis process using Faultree+ consider the fault tree in figure 2.6, where the model indicators and significant parameters are stated beneath each primary event (see Appendix II). The tree is named example.

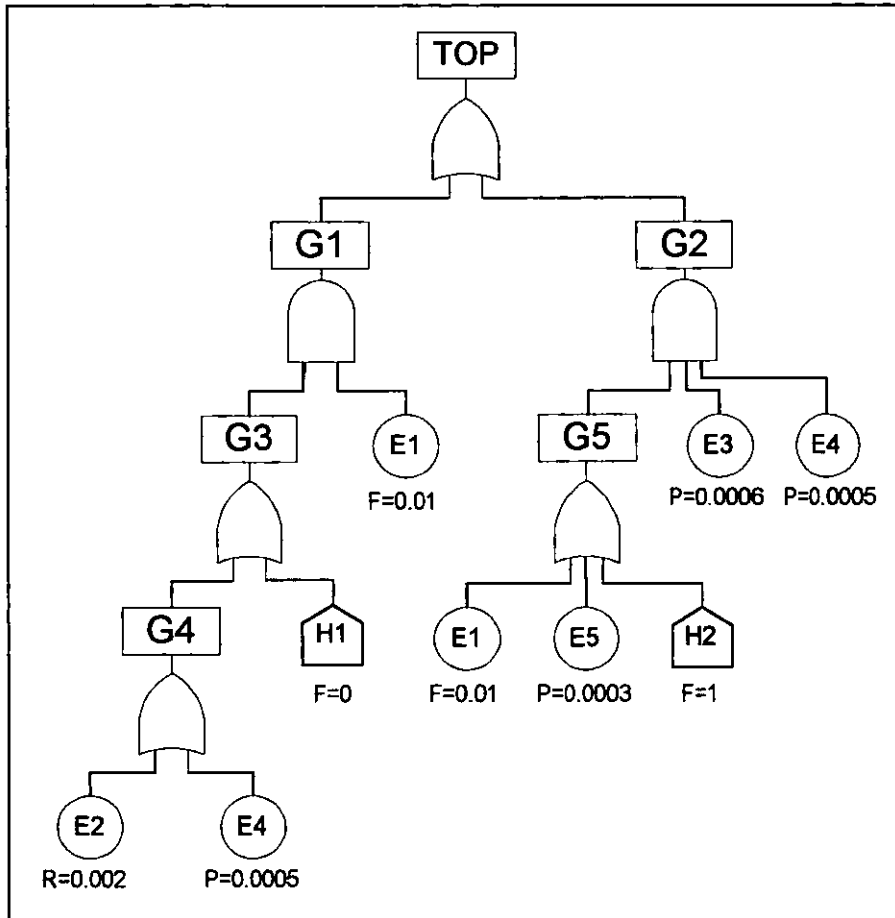


Figure 2.6 Fault tree 'Example'

On selection of the 'analysis' option the corresponding ascii files, 'example.ats' and 'example.aqd' are created. These are shown in figures 2.7 and 2.8 respectively. The first line in the ats file verifies that the top gate is an OR gate with 2 gate inputs: gate1 and gate2. All gates, which appear in the tree, are subsequently defined. As regards the aqd file, the first line describes the basic event 'event1'. The model code indicates a fixed model type. The second line states the value of the associated

parameters, i.e. the constant unavailability is 0.01 and the unconditional failure intensity 0.005 (both standard deviations are undefined). Note that Event3 is of dormant type, i.e. 'P' and hence the failure rate, inspection interval and mean time to repair parameters are specified.

top	OR	2	0gate1	gate2		
gate2	AND	1	2gate5	event3	event4	
gate1	AND	1	1gate3	event1		
gate3	OR	1	1gate4	house1		
gate4	AND	0	2event2	event4		
gate5	AND	0	3event1	event5	house2	

Figure 2.7 Fault Tree Structure File Called example.ats

Event1				F		
0.01	0.005	0	0			
Event3				P		
0.0006	8904	36	0	0		
House2				H		
1	0	0	0			
Event2				R		
0.002	0.03	0	0			
House2				H		
0	0	0	0			
Event4				P		
0.0005	5040	36	0	0		
Event5				R		
0.0003	0.004	0	0			

Figure 2.8 Quantification Data File Called example.aqd

Analysis determines an upper and lower bound to system unavailability, system unconditional failure intensity, system failure rate and expected failures, total down time and the number of minimal cut sets.

2.6 Binary Decision Diagrams

Fault trees provide a clear and logical visual tool, which is then analyzed, first qualitatively then quantitatively. Fault trees are not, however, an ideal form for

mathematical analysis. Computing the minimal cut sets of a fault tree and evaluating the associated probability are problems that increase exponentially with respect to the number of basic events involved. Since the 1960's a lot of effort has been devoted to both aspects of fault tree analysis. Available analysis tools, such as Faulttree+ (described in section 2.5), resort to the use of approximations. In addition, they assume that basic events have a small likelihood of occurrence and introduce problems such as estimation of truncation error. Analysis techniques have in effect reached 'saturation point'. Computerized methods are so well developed that further refinement is unlikely to significantly alleviate computer effort. Substantial improvement in fault tree analysis will, thus only result from a completely new approach.

2.6.1 Introduction to BDD's

Binary Decision Diagrams (BDD's) provide an alternative approach for manipulating Boolean functions that overcome the limitations of many conventional procedures. BDD's were formally defined in 1986 by Bryant (Ref. 14), who started from much more ancient ideas of C.Y.Lee (Ref. 49) and S.B.Akers (Ref. 3). The use of BDD's in the reliability analysis framework was initiated by O.Coudert and J.C.Madre (Ref. 19) and further developed by Rauzy (Ref. 71) to analyse fault trees.

The fault tree structure is first converted to a BDD, which represents the Boolean equation for the top event in a form much easier to manipulate than the fault tree. The BDD is then used to encode the minimal cut sets and carry out an exact quantitative assessment. Analysis of BDD's has been shown to be much faster than the quantification of the fault tree structure itself.

This section describes how to construct the BDD and obtain the minimal cut sets. In addition, factors affecting efficient construction and necessary minimisation procedures resulting in an optimal BDD are discussed.

2.6.2 General Description

A BDD is a rooted, directed acyclic graph. All paths through the BDD terminate in one of two states, either a 1 state, which corresponds to system failure, or a 0 state, which corresponds to system success. All the paths terminating in a 1 state give the minimal cut sets of the fault tree. A BDD is composed of terminal and non-terminal vertices, which are connected by branches. Terminal vertices have the value 0 or 1 and non-terminal vertices correspond to the basic events of the fault tree. Each vertex has a 0 branch which represents basic event non-occurrence (works) and a 1 branch which represents basic event occurrence (fails). All the right hand branches leaving each vertex are the 0 branches and all the left-hand branches are 1 branches. As an example BDD consider figure 2.9.

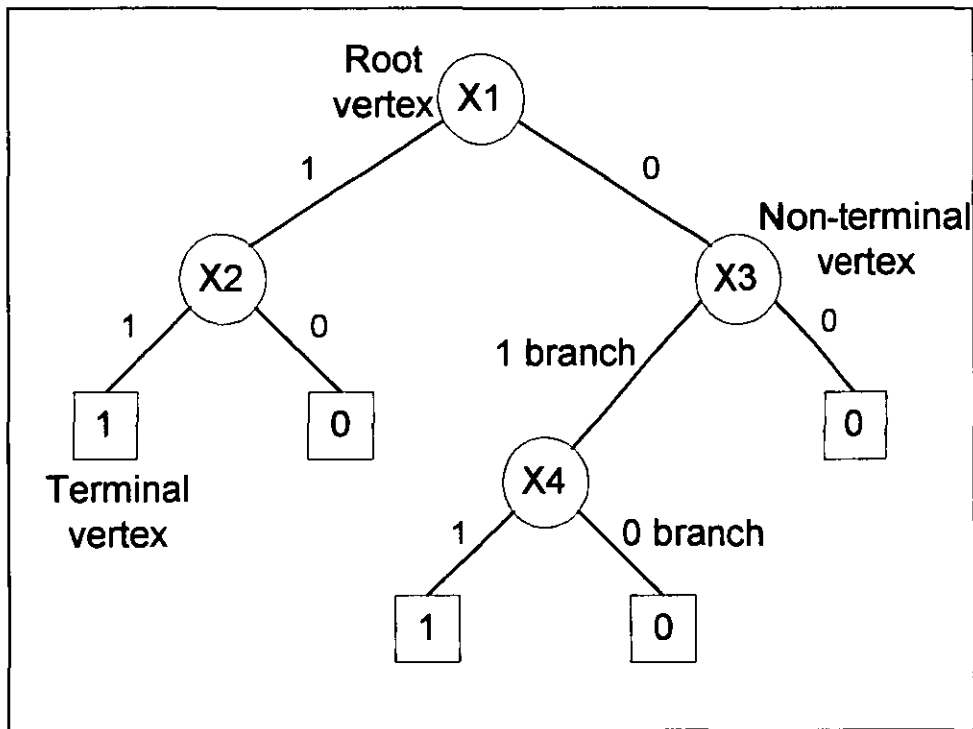


Figure 2.9 An Example BDD

Every path starts from the top basic event, called the root vertex, and proceeds down through the diagram to the terminal vertices. Only the vertices (or nodes) that lie on a 1 branch on the way to the terminal 1 vertex are included in a path, which represents the cut sets. For example a path, or cut set, of the BDD shown in figure 2.9 is

{X1.X2}. To reach the left most terminal 1 vertex a 1 branch is traveled from X1 and a 1 branch from X2. The other path set of the BDD in figure 2.9 is {X3.X4}. The path taken from the root vertex is a 0 branch and hence, X1 is not included in the latter cut set. The BDD represents the same logical function as its equivalent fault tree.

2.6.3 BDD Construction

The BDD method developed by Rauzy (Ref. 71) constructs the BDD from its equivalent fault tree using a bottom up procedure. The variables must first be assigned an ordering. Rauzy recommends the use of a top-down, left-right ordering, where the basic events which are placed higher up the tree are listed first and regarded as being 'less than' those further down the tree, such as $X1 < X2$. Basic events encountered for the first time on the same level are considered from left to right, i.e. those to the left are considered 'less than' those at their right. In addition **ite** connectives are assigned to each basic event, X_i , in the fault tree, i.e. **ite**(X_i , 1, 0). Each basic event X_i can either fail (1 branch) or work (0 branch). The following procedures are then used to compute the BDD.

Let $J = \text{ite}(x, F1, F2)$ and $H = \text{ite}(y, G1, G2)$ then

1. If $x < y$;

$$J\langle\text{op}\rangle H = \text{ite}(x, F1\langle\text{op}\rangle H, F2\langle\text{op}\rangle H)$$

2. If $x = y$;

$$J\langle\text{op}\rangle H = \text{ite}(x, F1\langle\text{op}\rangle G1, F2\langle\text{op}\rangle G2)$$

Where $\langle\text{op}\rangle$ denotes a Boolean operation of the logic gates in the fault tree. This is replaced by the dot or product symbol if the gate is an AND gate and the sum symbol if the gate is an OR gate. Where the fault tree includes vote gates the following procedure is also used.

3. If F is a k/n vote gate with inputs F_1, K, F_n , i.e. $F = at - least(k, F_1, K, F_n)$, then F is written implicitly as $(F_1 \cap at - least(k - 1, F_2, K, F_n)) \cup at - least(k, F_2, K, F_n)$, where $\cap \equiv$ AND and $\cup \equiv$ OR.

The following identities are used with the procedures above to simplify the *ite* structure at each stage.

If $\langle op \rangle$ is an OR gate;

$$1 + H = 1$$

$$0 + H = H$$

If $\langle op \rangle$ is an AND gate;

$$1 \cdot H = H$$

$$0 \cdot H = 0$$

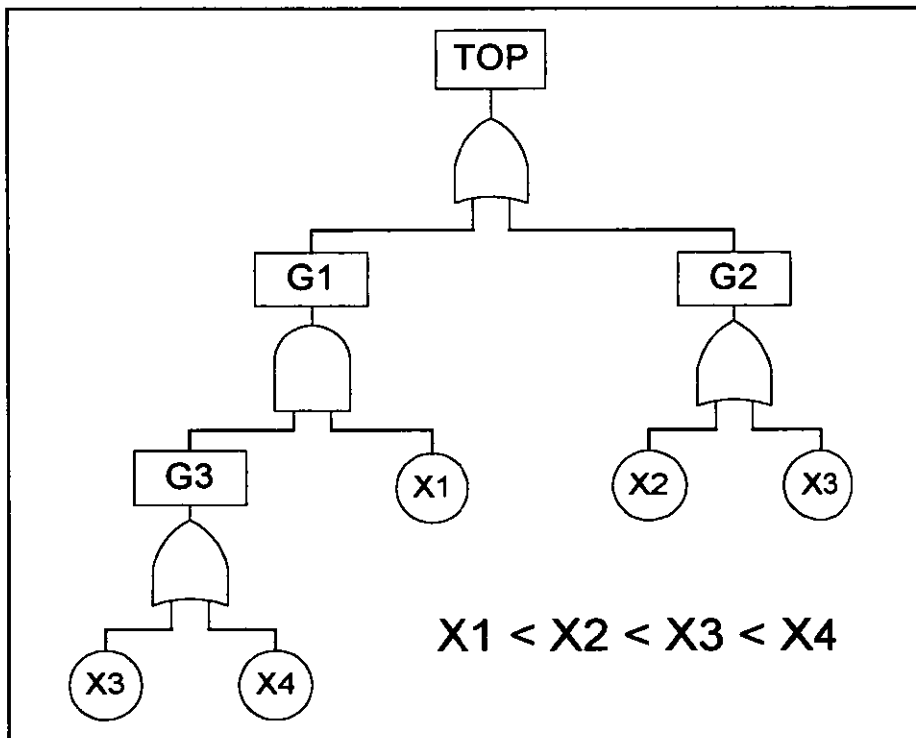


Figure 2.10 Example Fault Tree for ite Construction

The BDD construction procedure is demonstrated using the fault tree shown in figure 2.10 and obtains the *ite* structure for the top event in the following manner.

1. The variable ordering is $X1 < X2 < X3 < X4$;
2. Each basic event is assigned an **ite** structure;

$$X1 = \mathbf{ite}(X1, 1, 0)$$

$$X2 = \mathbf{ite}(X2, 1, 0)$$

$$X3 = \mathbf{ite}(X3, 1, 0)$$

$$X4 = \mathbf{ite}(X4, 1, 0)$$

3. Applying a bottom-up approach and using the relevant procedures and simplification identities gives

$$\begin{aligned} G3 &= \mathbf{ite}(X3, 1, 0) + \mathbf{ite}(X4, 1, 0) \\ &= \mathbf{ite}(X3, 1 + \mathbf{ite}(X4, 1, 0), 0 + \mathbf{ite}(X4, 1, 0)) \\ &= \mathbf{ite}(X3, 1, \mathbf{ite}(X4, 1, 0)) \end{aligned}$$

$$\begin{aligned} G1 &= G3.X1 \\ &= \mathbf{ite}(X3, 1, \mathbf{ite}(X4, 1, 0)).\mathbf{ite}(X1, 1, 0) \\ &= \mathbf{ite}(X1, 1.\mathbf{ite}(X3, 1, \mathbf{ite}(X4, 1, 0)), 0.\mathbf{ite}(X3, 1, \mathbf{ite}(X4, 1, 0))) \\ &= \mathbf{ite}(X1, \mathbf{ite}(X3, 1, \mathbf{ite}(X4, 1, 0)), 0) \end{aligned}$$

$$\begin{aligned} G2 &= X2 + X3 \\ &= \mathbf{ite}(X2, 1, 0) + \mathbf{ite}(X3, 1, 0) \\ &= \mathbf{ite}(X2, 1, \mathbf{ite}(X3, 1, 0)) \end{aligned}$$

$$\begin{aligned} Top &= G1 + G2 \\ &= \mathbf{ite}(X1, \mathbf{ite}(X3, 1, \mathbf{ite}(X4, 1, 0)), 0 + \mathbf{ite}(X2, 1, \mathbf{ite}(X3, 1, 0))) \\ &= \mathbf{ite}(X1, \mathbf{ite}(X3, 1, \mathbf{ite}(X4, 1, 0)) + \mathbf{ite}(X2, 1, \mathbf{ite}(X3, 1, 0)), 0 + \mathbf{ite}(X2, 1, \mathbf{ite}(X3, 1, 0))) \\ &= \mathbf{ite}(X1, \mathbf{ite}(X2, 1 + \mathbf{ite}(X3, 1, \mathbf{ite}(X4, 1, 0))), \mathbf{ite}(X3, 1, 0)) + \mathbf{ite}(X3, 1, \mathbf{ite}(X4, 1, 0)), \mathbf{ite}(X2, 1, \mathbf{ite}(X3, 1, 0))) \\ &= \mathbf{ite}(X1, \mathbf{ite}(X2, 1, \mathbf{ite}(X3, 1 + 1, 0 + \mathbf{ite}(X4, 1, 0))), \mathbf{ite}(X2, 1, \mathbf{ite}(X3, 1, 0))) \end{aligned}$$

Therefore,

$$\text{Top} = \text{ite}(X1, \text{ite}(X2, 1, \text{ite}(X3, 1, \text{ite}(X4, 1, 0), \text{ite}(X2, 1, \text{ite}(X3, 1, 0))))$$

The top event is stated in terms of an equation of nested *ite* structures. Working from left to right, each variable is successively broken down into its left and right branches. Thus, X1 is the root variable with X2 on both its right and left branch. Considering the left branch, X2 breaks down into 1 on its left branch and X3 on the right. In turn, X3 breaks down into 1 and X4, where X4 represents the terminal *ite* structure, i.e. *ite*(X4, 1, 0). Returning to the right branch of the root node, the nested *ite* structure *ite*(X2, 1, *ite*(X3, 1, 0) is broken down into its respective branches. The resulting BDD structure is depicted in figure 2.11.

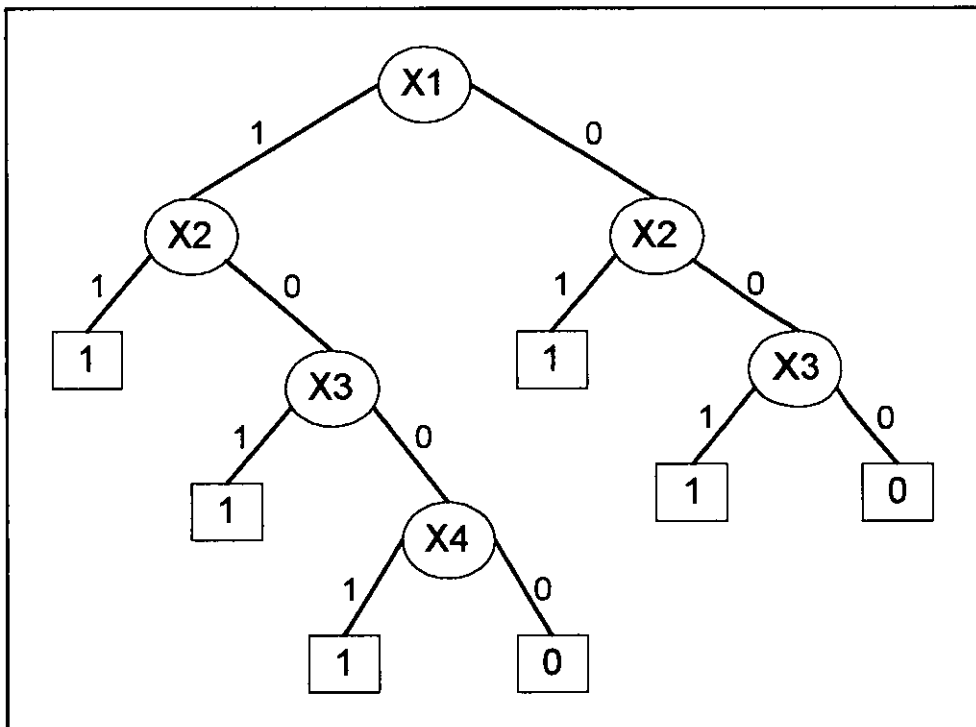


Figure 2.11 BDD for the Fault Tree Shown in Figure 2.10

The BDD structure can then be used to obtain the cut sets of the fault tree. Paths through the node are traced from the root node to a terminal 1 node. Only those basic events connecting to the following node via a 1 branch, indicating component failure, are included in the path. Thus, the paths through the BDD which correspond to the cut sets of the fault tree in the above example are

1. {X1.X2}
2. {X1.X3}
3. {X1.X4}
4. {X2}
5. {X3}

A BDD obtained using the *ite* procedure with an optimal basic event ordering scheme produces near minimal if not minimal cut sets. A final check may need to be made to eliminate those that are non-minimal. The resulting BDD in the above example is non-minimal. Cut sets 1 and 2 are redundant. This introduces the need for a minimisation procedure, which is discussed in the next section.

2.6.4 The Minimising Procedure

The BDD does not always give minimal cut sets, as shown in the *ite* construction corresponding to the fault tree in figure 2.11. In such cases, Boolean reduction can be used to establish the minimal set. The primary asset of the BDD technique is that computation time and storage requirements are reduced. If, however, non-minimal cut sets are produced, the need to reduce these using Boolean operations may destroy the gain in efficiency achieved by the BDD. To generate minimal combinations of cut sets alone a non-minimal BDD must undergo a minimising procedure. A minimisation algorithm due to Rauzy (Ref. 71) achieves its objective through introduction of a 'without' operator. Consider the simple BDD in figure 2.12.

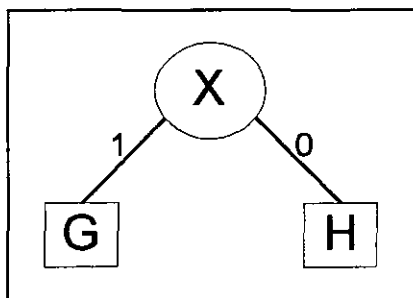


Figure 2.12 BDD for *ite*(X, F, G)

Assigning an ite connective to the top event gives $\text{Top} = \text{ite}(X, G, H)$. Let δ be a minimal solution of G , i.e. $\delta = \text{Sol}_{\min}(G)$ and let γ be a minimal solution of H , i.e. $\gamma = \text{Sol}_{\min}(H)$. Now, for $x \cap \delta$ to be a minimal solution of the top event, δ must not be a minimal solution of H . So, if σ is a complete set of the minimal solutions to the top event, i.e. $\sigma = \text{Sol}_{\min}(\text{Top})$, the implication is that

$$\sigma = [(\delta < \text{exp} > \gamma) \cap X] \cap [\gamma]$$

where $\delta < \text{exp} > \gamma$ is the set of δ excluding any which are in γ , i.e. the ‘without’ operator. Applying the without operator to each node in turn transforms the BDD such that it defines exactly the minimal cut sets of the fault tree.

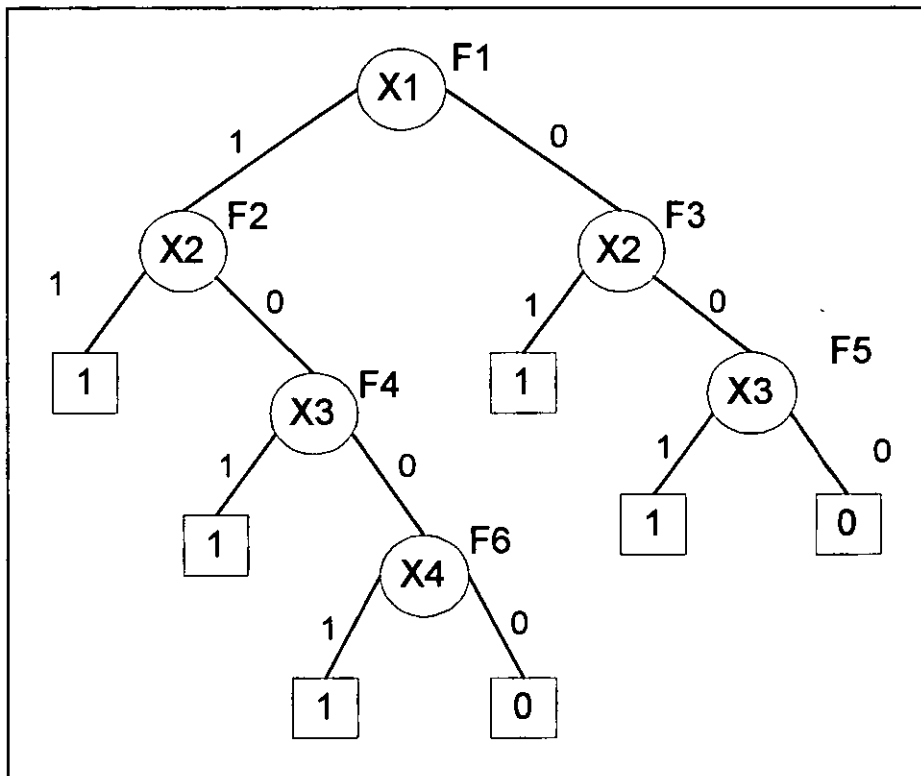


Figure 2.13 Example of a Non-minimal BDD

To illustrate the minimisation algorithm consider the non-minimal BDD in figure 2.13. Considering the nodes in a top-down fashion gives

$$F1 = \text{ite}(X1, F2, F3)$$

The minimal solutions of F2 are {X2} and {X3}. The minimal solutions of F3 are {X2}, {X3} and {X4}. Minimal Solution {X2} and {X3} are common to both F2 and F3 and hence must be removed from F2 by the without operator. This is achieved by eliminating nodes F2 and F4, thus making F6 the son of the 1 branch of F1, i.e. $F1 \rightarrow \text{ite}(X1, F6, F3)$.

Continuing Down the tree:

$$F3 = \text{ite}(X2, 1, F4)$$

The only solution on the 1 branch is X2. Therefore, no common solutions exist. Finally, both F4 and F5 represent the terminal *ite* structure and are thus minimal. The resulting minimal BDD is shown in figure 2.14. Tracking the paths through the BDD in its minimal form gives cut sets

1. {X2}
2. {X3}
3. {X1.X4}

which correspond to the minimal cut sets of the fault tree.

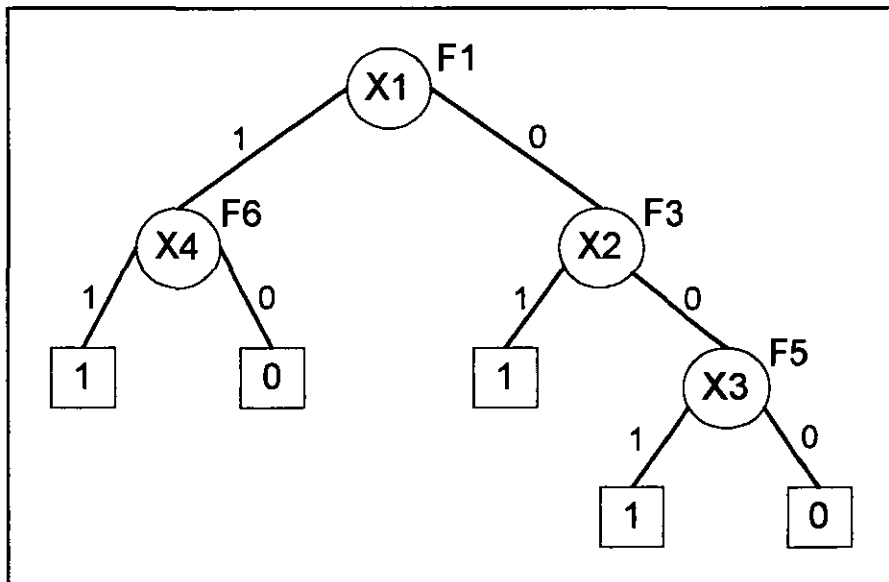


Figure 2.14 Minimal Form of the BDD Shown in Figure 2.13

2.6.5 Ordering

Efficient construction and the resulting size of the BDD are highly dependent on the ordering taken for the basic events. As stated by Bouissou (Ref. 13), there exists an extremely wide bracketing for BDD size. A BDD cannot exceed 2^{n-1} , i.e. the size of the corresponding binary tree, where n is the number of basic events. On the other hand, it cannot be lower than the number of variables the function really depends on. To illustrate this sensitivity to ordering, consider the fault tree in figure 2.10. The equivalent BDD constructed with an ordering of $X1 < X2 < X3 < X4$ is shown in figure 2.11. In contrast, the fault tree is also represented by the BDD in figure 2.15, constructed with a reverse ordering, i.e. $X4 < X3 < X2 < X1$.

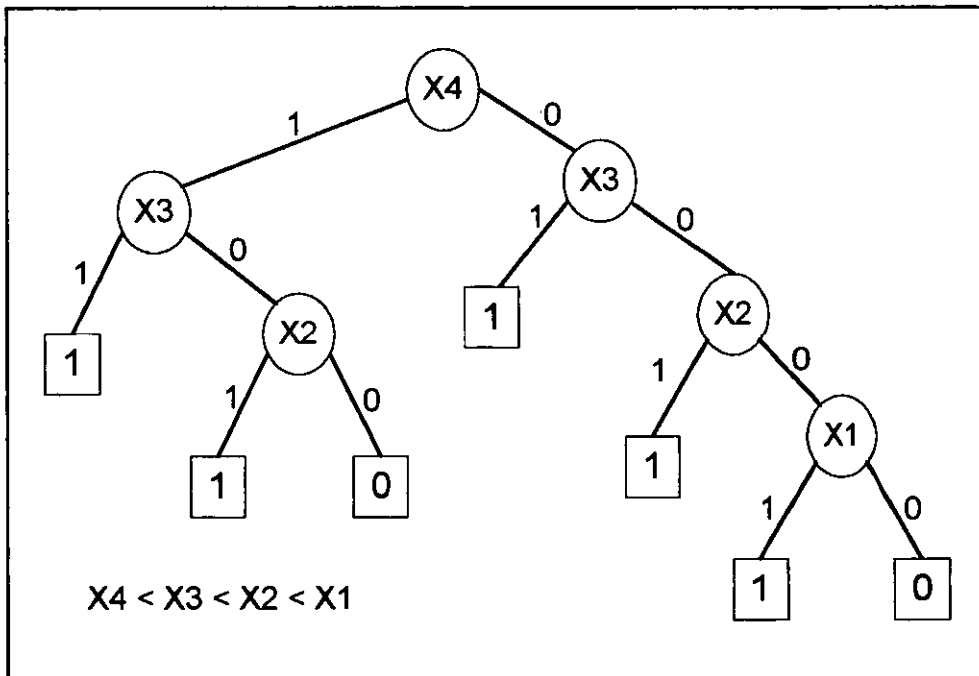


Figure 2.15 BDD Corresponding to Figure 2.10 with Ordering $X4 < X3 < X2 < X1$

As previously demonstrated the top-down, left-right ordering often results in a non-minimal BDD. The minimisation algorithm rectifies this. However, this increases computation time, thus counteracting the gains achieved through use of the BDD technique. An ordering scheme, which results directly in a minimal BDD, is beneficial.

In recent years much research has focused on a means to find an optimal ordering. Friedman and Supiwit (Ref. 31) describe an algorithm for finding an optimal ordering. It has, however, excessive running time. In addition, the algorithm requires that the Boolean function for the fault tree is determined. Deriving the Boolean function itself generally requires knowledge of the minimal cut sets, thus deeming the algorithm useless for practical applications. The general view is that the means to achieve a good ordering is to use heuristic methods (Ref.'s. 13,58).

In addition to the number of events and gates, the complexity of a fault tree depends on the number of repeated sub-trees and repeated primary events not belonging to the sub-trees (Ref. 18). As the number of repeated events increase, the number of non-minimal cut sets increase, thus the greater becomes the complexity of the analysis. On this basis, recent ordering techniques involve consideration of the number of repetitions of events, prior to placement. Sinammon (Ref. 79) investigates the effects of three different ordering schemes. Within each of these three schemes there is also the option to list repeated events first. For more detail the reader is referred to the paper. The results affirm the sensitivity of BDD construction to variable ordering. However, the effect due to the particular ordering implied dependency on the particular fault tree structure being analysed. In conclusion, there did not appear to be a general-rule based ordering scheme optimal for all fault trees. Further research concerning variable ordering is carried out by Bartlett (Ref. 8).

2.6.6 Top Event Quantification Using the BDD

2.6.6.1 Top Event Probability

Kinetic Tree Theory forms the basis for quantification of fault trees using minimal cut sets with component failure and repair distributions, as described in sections 2.3 and 2.4.

Top event quantification using a BDD avoids the need to use approximations and obtains an exact probability of the top event directly from the diagram. If the top

event its structure is $f(x) = \text{ite}(x_i, f1, f2)$, then the corresponding Boolean function is $f(x) = x_i f1 + \bar{x}_i f2$. When a Boolean function is expressed in this form, the probability of the top event is obtained by taking the expectation of each term

$$E[f(x)] = q_i E[f1] + (1 - q_i) E[f2] \quad (2.50)$$

where $q_i = E[x_i]$ the probability that event i has occurred. Each path to a terminal 1 vertex is, therefore, mutually exclusive or disjoint. The probability of occurrence of the top event is the sum of the probabilities of these disjoint paths (Ref. 78).

The probability of each disjoint path is a product of the probabilities of included basic events encountered on route from the top node to the terminal vertex, where the 0 branch infers the use of component success probabilities and the 1 branch component failure.

It is important to note that an unminimised BDD is used to trace the disjoint paths. The minimisation algorithm alters the top event structure function into a format, which no longer encodes Shannon's decomposition. This is discussed in greater detail in reference 79.

As an example, consider the non-minimal BDD structure in figure 2.11, previously obtained through application of the ite technique to the fault tree in figure 2.10. The failure data assigned to each event is shown in table 2.8.

Basic Event	Failure rate (λ_i)	Unavailability (q_i)	Unconditional Failure Intensity (w_i)
X1	2×10^{-5}	0.01	1.98×10^{-5}
X2	6×10^{-6}	0.03	5.82×10^{-6}
X3	1×10^{-6}	0.05	9.5×10^{-7}
X4	5×10^{-5}	0.02	4.9×10^{-5}

Table 2.8 Basic Event Data

The BDD has 5 terminal vertices representing system failure. The corresponding disjoint paths are

- 1) $X_1.X_2$
- 2) $X_1.\overline{X_2}.X_3$
- 3) $X_1.\overline{X_2}.\overline{X_3}.X_4$
- 4) $\overline{X_1}.X_2$
- 5) $\overline{X_1}.\overline{X_2}.X_3$

System unavailability (Q_{SYS}) is the sum of the product of each disjoint path

$$\begin{aligned}
 Q_{SYS} &= P(X_1.X_2 + X_1.\overline{X_2}.X_3 + X_1.\overline{X_2}.\overline{X_3}.X_4 + \overline{X_1}.X_2 + \overline{X_1}.\overline{X_2}.X_3) \\
 &= q_{X_1}q_{X_2} + q_{X_1}(1 - q_{X_2})q_{X_3} + q_{X_1}(1 - q_{X_2})(1 - q_{X_3})q_{X_4} \\
 &\quad + (1 - q_{X_1})q_{X_2} + (1 - q_{X_1})(1 - q_{X_2})q_{X_3} \\
 &= (0.01 \times 0.03) + (0.01 \times 0.97 \times 0.05) + (0.01 \times 0.97 \times 0.95 \times 0.02) \\
 &\quad + (0.99 \times 0.03) + (0.99 \times 0.97 \times 0.05) \\
 &= 7.87 \times 10^{-2}
 \end{aligned}$$

2.6.6.2 Unconditional System Failure Intensity

A further quantitative system measure is the expected number of top event occurrences over a given period of time, as described in section 2.4.5.2.

Direct implementation of equation (2.42) to find the criticality of each component involves substitution of $q_i = 0$ then $q_i = 1$ followed by evaluation of the system failure probabilities. This process requires $2n$ top event calculations, where n is the number of components in the system.

Deviation from this direct approach uses the BDD structure. Each component occurs at least once in the BDD. Figure 2.16 illustrates the occurrence of component X_i .

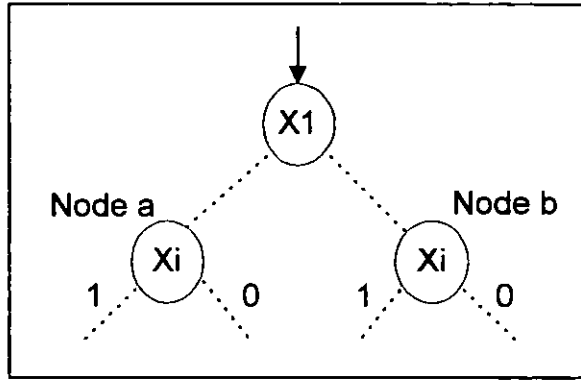


Figure 2.16 Consider Variable Xi

Determination of each term in the criticality function for component Xi can be achieved using

$$\begin{aligned}
 Q(1_i, q) &= \sum_n (pr_{Xi}(q) \cdot po_{Xi}^1(q)) + Z(q) \\
 Q(0_i, q) &= \sum_n (pr_{Xi}(q) \cdot po_{Xi}^0(q)) + Z(q)
 \end{aligned}
 \tag{2.51}$$

where

- $pr_{Xi}(q)$ - is the probability of the path section from the root node to node Xi (Probprev).
- $po_{Xi}^1(q)$ - is the probability of the path section from the 1 branch node of Xi to a terminal 1 node (Probpost 1 branch).
- $po_{Xi}^0(q)$ - is the probability of the path section from the 0 branch node of Xi to a terminal 1 node (Probpost 0 branch).
- $Z(q)$ - is the probability of the paths from the root node to the terminal 1 nodes which do not go through the node for variable Xi.
- n - all nodes for variable Xi in the BDD.

and hence:

$$G_i(q) = \sum_n pr_{Xi}(q) [po_{Xi}^1(q) - po_{Xi}^0(q)]
 \tag{2.52}$$

The BDD in figure 2.13 is used to demonstrate this approach. Consider first the evaluation of *Probpost1* ($po_{x_i}^1(q)$) and *Probpost0* ($po_{x_i}^0(q)$) for each node.

Probpost(F6)

$$\begin{aligned} F6 &= \text{ite}(X4, 1, 0) \\ po_{x_4}^1(q) &= \text{prob}(1) = 1 \\ po_{x_4}^0(q) &= \text{prob}(0) = 0 \\ Q_{F6} &= p(X4)p(1) + p(\overline{X4})p(0) = 0.02 \end{aligned}$$

Probpost(F5)

$$\begin{aligned} F5 &= \text{ite}(X3, 1, 0) \\ po_{x_3}^1(q) &= \text{prob}(1) = 1 \\ po_{x_3}^0(q) &= \text{prob}(0) = 0 \\ Q_{F5} &= p(X3)p(1) + p(\overline{X3})p(0) = 0.05 \end{aligned}$$

Probpost(F4)

$$\begin{aligned} F4 &= \text{ite}(X3, 1, F6) \\ po_{x_3}^1(q) &= \text{prob}(1) = 1 \\ po_{x_3}^0(q) &= \text{prob}(F6) = 0.02 \\ Q_{F4} &= p(X3)p(1) + p(\overline{X3})p(0.02) = 0.069 \end{aligned}$$

Probpost(F3)

$$\begin{aligned} F3 &= \text{ite}(X2, 1, F5) \\ po_{x_2}^1(q) &= \text{prob}(1) = 1 \\ po_{x_2}^0(q) &= \text{prob}(F5) = 0.05 \\ Q_{F3} &= p(X2)p(1) + p(\overline{X2})p(0.05) = 0.0785 \end{aligned}$$

Probpost(F2)

$$\begin{aligned} F2 &= \text{ite}(X2, 1, F4) \\ po_{x_2}^1(q) &= \text{prob}(1) = 1 \\ po_{x_2}^0(q) &= \text{prob}(F4) = 0.069 \\ Q_{F2} &= p(X2)p(1) + p(\overline{X2})p(0.069) = 0.09693 \end{aligned}$$

Probpost(F1)

$$F1 = \text{ite}(X1, F2, F3)$$

$$po_{X1}^1(q) = \text{prob}(F2) = 0.09693$$

$$po_{X1}^0(q) = \text{prob}(F3) = 0.0785$$

$$Q_{F1} = p(X1)p(0.09693) + p(\overline{X1})p(0.0785) = 0.07868$$

An agreement can be seen between the probability of the top event Q_{SYS} evaluated previously using the sum of the disjoint paths and the unavailability of the top node Q_{F1} .

Attention is now given to calculation of *Probprev* ($pr_{X_i}(q)$) for each node.

$$Probprev(F1) = 1$$

$$\begin{aligned} Probprev(F2) &= p(X1) \\ &= 0.01 \end{aligned}$$

$$\begin{aligned} Probprev(F3) &= p(\overline{X1}) \\ &= 0.99 \end{aligned}$$

$$\begin{aligned} Probprev(F4) &= p(\overline{X2})probprev(F2) \\ &= (0.97)(0.01) \\ &= 0.0097 \end{aligned}$$

$$\begin{aligned} Probprev(F5) &= p(\overline{X2})probprev(F3) \\ &= (0.97)(0.99) \\ &= 0.9603 \end{aligned}$$

$$\begin{aligned} Probprev(F6) &= p(\overline{X3})probprev(F4) \\ &= (0.95)(0.0097) \\ &= 0.009215 \end{aligned}$$

The node, corresponding basic event, probability of post 1 branch, probability of post 0 branch and probability of the previous path section are portrayed in the respective columns of table 2.9.

Node	Basic Event	<i>Probpost1</i>	<i>Probpost0</i>	<i>Provprev</i>
F1	X1	0.09693	0.0785	1
F2	X2	1	0.069	0.01
F3	X2	1	0.05	0.99
F4	X3	1	0.02	0.0097
F5	X3	1	0	0.9603
F6	X4	1	0	0.009215

Table 2.9 Probtable

Using the values in table 2.9 within equation (2.52), evaluation of the criticality function of each component is straightforward.

X1 is represented by F1 alone:

$$\begin{aligned}
 G_{X_1}(q) &= pr_{F_1}(q)[po_{F_1}^1(q) - po_{F_1}^0(q)] \\
 &= 1[0.09693 - 0.0785] \\
 &= 0.01843
 \end{aligned}$$

X2 is represented by both F2 and F3:

$$\begin{aligned}
 G_{X_2}(q) &= pr_{F_2}(q)[po_{F_2}^1(q) - po_{F_2}^0(q)] + pr_{F_3}(q)[po_{F_3}^1(q) - po_{F_3}^0(q)] \\
 &= 0.01[1 - 0.069] + 0.99[1 - 0.05] \\
 &= 0.9498
 \end{aligned}$$

X3 is represented by both F4 and F5:

$$\begin{aligned}
 G_{X_3}(q) &= pr_{F_4}(q)[po_{F_4}^1(q) - po_{F_4}^0(q)] + pr_{F_5}(q)[po_{F_5}^1(q) - po_{F_5}^0(q)] \\
 &= 0.0097[1 - 0.02] + 0.9603[1 - 0] \\
 &= 0.9611
 \end{aligned}$$

X4 is represented by F6 alone:

$$\begin{aligned}
 G_{X4}(q) &= pr_{F6}(q)[po_{F6}^1(q) - po_{F6}^0(q)] \\
 &= 0.009215[1 - 0] \\
 &= 0.009215
 \end{aligned}$$

The criticality function values of each component are used in equation (2.44) (each component's unconditional failure intensity (w_i) is stated in table 2.8) to give the steady state system unconditional failure intensity w_{SYS} :

$$\begin{aligned}
 w_{SYS} &= G_{X1}(q)w_{X1} + G_{X2}(q)w_{X2} + G_{X3}(q)w_{X3} + G_{X4}(q)w_{X4} \\
 &= (0.01843)(1.98 \times 10^{-5}) + (0.9498)(5.82 \times 10^{-6}) + (0.9611)(9.5 \times 10^{-7}) \\
 &\quad + (0.009215)(4.9 \times 10^{-5}) \\
 &= 7.2573 \times 10^{-6}
 \end{aligned}$$

System unconditional failure intensity considers only the time interval $[t, t + dt)$.

Using equation (2.21) the expected number of system failures over a specific time period can be obtained. Consider, for example, a 10 year operating period (i.e. 8760 hours) then the expected number of failures in this time is

$$\begin{aligned}
 W(0,8760) &= \int_0^{8760} 7.2573 \times 10^{-6} dt \\
 &= 0.0636
 \end{aligned}$$

CHAPTER 3

OPTIMISATION TECHNIQUES

3.1 Introduction

Optimisation theory is a means to identify the best candidate from a collection of possible alternatives without having to explicitly enumerate and evaluate all possibilities. Some criterion is established as a performance measure for the optimisation procedure. In an industrial process a common criterion used is minimum cost, where the product cost can depend on a large number of interrelated controlled parameters in the manufacturing process. More complex systems consider multiple criteria.

In practice it is very difficult to determine if the best candidate obtained is indeed a global minimum, i.e. the best point over the whole search space. In most circumstances it can only be said that the point obtained is a minimum within a local area of search.

Many types of optimisation techniques exist. The particular technique applied is dependent on the characteristics of the problem. Some factors of concern are 'a priori' knowledge of the search region itself and the associated objective function, the number and type of design variables and the number and type of constraints, if any, restricting the search region.

The purpose of this chapter is to consider the basic concepts behind some well-known optimization techniques, to analyse and evaluate aspects of their algorithm structure in conjunction with modifications and extensions in recent research. Where appropriate potential difficulties in application of the algorithm to complex optimisation problems are expressed. Conversely, potentially advantageous features are noted. The ultimate aim is to propose a method or methods appropriate for effective and efficient application to the safety system optimization problem under consideration.

The chapter opens with a section on *linear programming*. *Gradient based* optimisation techniques are then considered followed by a discussion of *direct and random search methods*.

3.2 Linear Programming

The term Linear Programming (LP) defines a particular class of optimisation techniques in which the objective function is a linear function of the design variables and the constraints of the system can be expressed as linear equations or inequalities. In recent years much research has been carried out in this area, proving it to be one of the most valued optimization tools.

In matrix-vector notation any LP can be transformed into the standard form

$$\begin{aligned} &\text{minimise} && \mathbf{c}^T \mathbf{x} \\ &\text{subject to} && \mathbf{Ax} \geq \mathbf{b} \\ &&& \mathbf{x} \geq 0 \end{aligned} \tag{3.1}$$

Here \mathbf{x} is an n -dimensional vector, \mathbf{c}^T is an n -dimensional row vector, \mathbf{A} is an $m \times n$ coefficient matrix, and \mathbf{b} is an m -dimensional column vector. The vector inequality $\mathbf{x} \geq 0$ means each component of \mathbf{x} is nonnegative.

The LP finds the best solution in the feasible region. In general, there are more variables than equations. Hence, the selection of the best solution to minimise the system is a non-trivial problem. From a geometric viewpoint, optimal solutions occur at corner (or extreme) points of the convex search space. The basic concept of the LP is to focus on the extrema to establish the most optimal feasible corner point. The system of m equations in n unknowns is reduced by elementary row operations to canonical form. The variables x_1, \dots, x_m , that appear in only one equation with unit coefficient are termed basic variables. The remaining $n - m$ variables are termed nonbasic variables. A basic solution is obtained from the canonical system by setting the nonbasic variables to zero. If the vector of the resulting solution is nonnegative it is termed a basic feasible solution. It can be shown that every corner point of the feasible region corresponds to a basic feasible solution of the constraint equation.

3.2.1 The Simplex Method

Having established that the optimal solutions lie at extrema restricts the potentially infinite set of points to analyse. The simplex method is an iterative algorithm, which takes advantage of this knowledge and solves the LP in a more efficient manner by examining only a fraction of the total number of basic feasible solutions. A more detailed description on solving a LP using the Simplex Method can be found in references 42, 48 and 83.

3.2.2 Integer Programming

Combinatorial optimisation involves assigning discrete numerical values, from some finite set, to a finite set of variables, \mathbf{x} , to minimise the function $f(\mathbf{x})$ whilst simultaneously satisfying a given set of constraints. The functions and constraints may be non-linear, discontinuous or implicit. Integer linear programming (ILP or IP) is a simplified subset of this general area in which the objective function and constraints are assumed to be linear. If n is the number of decision variables, a problem in which a subset of q ($1 \leq q \leq n$) variables are constrained to be integer is termed a mixed IP. If all n variables must be integer a pure IP is being considered. A further consideration in the reliability optimisation problems is the binary variables (Boolean or 0-1 variables). Boolean variables are restricted to just two values, generally 0 or 1. As regards the decision scenario, this determines situations such as, is a particular component fitted; yes or no? IP problems containing only binary variables are referred to as a *Binary Integer Programming* (BIP) problem or a *zero-one programming problem*. Without the integer restrictions we are dealing with the original LP problem.

Here is a typical example of a small constrained IP problem

$$\begin{aligned} \text{maximise} \quad & z = 6x_1 + 5x_2 \\ \text{subject to} \quad & x_1 + x_2 \leq 5 \\ & 13x_1 + 6x_2 \leq 49 \\ & x_1, x_2 \geq 0 \text{ and integer} \end{aligned} \tag{3.2}$$

IP problems seek the determination of the optimum point among all the discrete points included in the feasible solution space. The inference is that an IP problem is an LP problem with fewer solutions to consider, and should, therefore, be relatively easy to solve. Two major problems arise with this reasoning.

The reduced solution space, though finite, can still be exceedingly large. Consider, for example, binary variables. If n variables exist there corresponds 2^n potential solutions. In general, as the dimensionality of the problem increases linearly the equivalent solution grows exponentially, thus, requiring much computer effort.

Secondly, the key to the efficiency of the simplex method in solving LP problems is the guarantee that the optimal solutions will be feasible corner points of the solution space. Integer points within the region may not, and in fact in the majority of cases do not, lie on the region's boundary, let alone the corners. Removing these feasible solutions breaks down the crux of the simplex procedure.

In special cases the resulting optimal solution of an IP problem utilising the simplex algorithm is integer. This is, however, particularly unusual. A key limitation of the simplex method is the inherent assumption of divisibility.

Much research and many sophisticated optimisation techniques are dependent on the special features of the simplex algorithm. Despite the obvious flaws, in many ways it is advantageous to utilise the simplex method to solve IP problems, particularly considering the lack of globally successful methods specifically developed for problems involving integer variables.

One approach is to carry out *LP relaxation*. The IP is solved using the simplex algorithm without considering the restriction on the integer variables. In other words the IP is converted or 'relaxed' to its equivalent LP and the continuous optimum solution obtained. Regarding the IP model, expression (3.2), the corresponding relaxed LP is the same except that the variables x_1 and x_2 do not have an integer stipulation.

Elements in the continuous optimal vector solution corresponding to the integer restricted variables are then rounded to their closest integer value. Discussion of

the relationship between an IP problem and its relaxed LP problem is discussed in a paper by Joseph *et al* in 1998 (Ref. 47).

There are pitfalls associated with this approach. The rounded integer solution is not necessarily optimal. In addition, the integer solution may not be feasible. Any integer model having an original equality constraint can never yield a feasible solution through rounding. Binary (or 0-1 variables) emphasize further the inadequacy of rounding. Binary variables imply decisions based on their value, 0 or 1. It is nonsensical to deal with fractional values of such variables, and the use of rounding as an approximate is logically unacceptable. In general, rounding is reasonable when the values of the variables are so large only negligible changes are produced. When dealing with problems in which the integer variables assume small values, rounding and truncating may significantly distort the optimal solution. Binary variables are an extreme example of this. The implication is the need for more sophisticated methods to solve IP problems.

IP techniques are generally categorised into two broad types: (1) *Search methods* and (2) *Cutting methods*.

Theoretically, any IP problem can be solved by simply listing all possible feasible solutions, evaluating each point and choosing the best, i.e. *exhaustive enumeration*. As stated earlier, even when finite the solution space can be very large without having significantly high dimensionality. Search methods are techniques, which enumerate as small a portion of all candidate solutions as possible while automatically discarding the remaining points as non-promising. In other words, many points are enumerated implicitly. This is termed *implicit enumeration*. The *branch and bound* technique is a prime example of an implicit enumeration technique, specifically designed for an IP problem. The *zero-one method* is, in the main, considered as a special case of the branch and bound approach.

Cutting methods are motivated by the fact that the simplex solution to a linear program must occur at an extreme point. The idea is to solve the relaxed LP model and identify its continuous optimal point. Utilising this continuous optimum, specially developed secondary constraints are added to the relaxed LP. The secondary constraints are violated by the non-integer solution of the relaxed

LP, but not by any feasible integer point. These constraints modify the solution space and will iteratively force the optimum extreme point of the resulting modified LP model toward the desired integer restriction.

Cutting methods and the branch and bound technique are similar in that they both eliminate non-integer optimal solutions to the associated LP model, but leave all feasible integer solutions untouched. There are, however, fundamental differences between the two approaches. The branch and bound technique creates many different linear programming problems, whereas the successive constraints of cutting plane methods are added to the original LP. Consequently, the original feasible region of the branch and bound problem is divided into disconnected sub-regions. In contrast, cutting plane methods gradually reduce the original feasible region as extra constraints are added.

Branch and Bound Technique

Branch and bound is an iterative technique. The original search space is too large to solve directly. It is, therefore, divided into smaller and smaller sub-regions and handled separately. This procedure is achieved via a sequential process of *branching*, *bounding* and *fathoming*, which results in a decision tree. The first iteration, carried out on the relaxed LP of the original problem, establishes the point at which the tree is rooted. Subsequent iterations, via the branching process, produce further nodes emanating from the root node. The branch and bound algorithm will be explained by means of the two-dimensional example expressed in expression (3.2). (Two dimensions are used to simplify the explanation and enable a graphical representation).

Figure 3.1 shows the feasible integer solution grid and the continuous optimal solution, ignoring the integer restriction, to the above model. The grid points represent the solution to the IP.

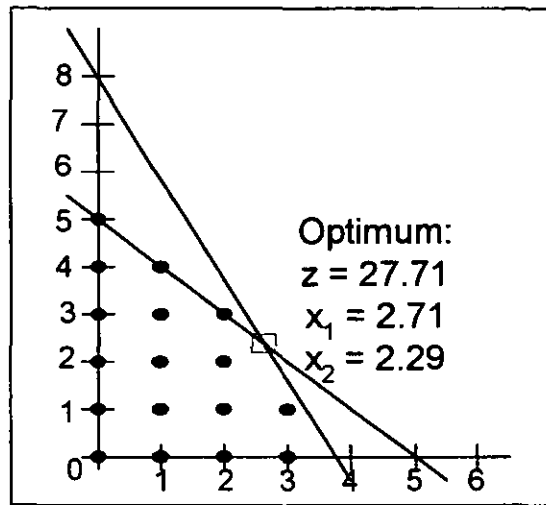


Figure 3.1 Integer Solution Grid for IP Model

The problem is to establish the maximum. A lower bound is, therefore initially set at $-\infty$, i.e. let $z_{(0)}^* = -\infty$. The first step is to solve the relaxed LP, i.e. the original model without the integer restriction, referred to as the LP0. The solution of the LP0, as depicted on Figure 1, is $x_1 = 2.71$, $x_2 = 2.29$ and $z = 27.71$. A check is carried out to ensure the solution is non-integer. An integer solution would imply an immediate optimal feasible solution and the algorithm terminates. Otherwise, the continuous optimum defines a bound, where the bound is a 'criteria of goodness' for the root node. Any feasible solution to the IP will not exceed this value.

The initial region must now be partitioned into two disjoint subsets, giving rise to the concept of branching. In effect branching signifies partitioning the current solution space into mutually exclusive subspaces without eliminating any feasible integer points. Branching is carried out about one variable only at a time. Any variable, whose current value in the optimum LP0 solution violates its integer requirement, can be chosen as the branching variable. Both x_1 or x_2 could be chosen here. Select $x_1 (= 2.71)$ arbitrarily, we can partition the original set of feasible solutions into one subset comprising all solutions $x_1 \leq 2$, and the other subset containing all solutions with $x_1 \geq 3$, see Figure 3.2. Note that no feasible integer points have been eliminated.

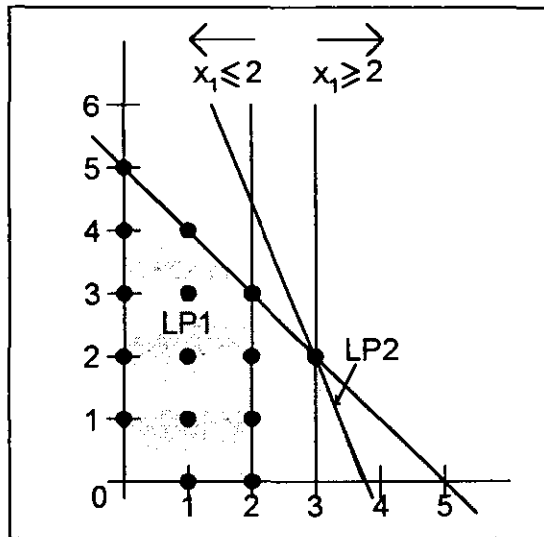


Figure 3.2 Partitioning the Feasible Space

As regards the decision tree the partitioned linear models create two new nodes emanating from the root node, LP1 and LP2, where the root node represents all feasible solutions, as shown in Figure 3.3.

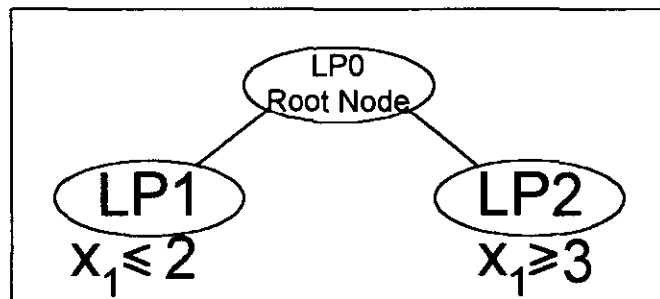


Figure 3.3 The Decision Tree

Consider each node (or sub-region) in turn. The relaxed LP on LP1 is solved, having the added restriction $x_1 \leq 2$. The solution is $x_1 = 2$, $x_2 = 3$, and $z = 27$. The optimal solution of the subspace of LP1 is in fact integer. There are no further branching variables to choose, neither can this feasible integer solution for the subset be improved upon. The node is said to be fathomed. The feasible integer solution is compared against the lower bound. If this is an improvement it becomes the incumbent. It is essential for the efficient performance of the branch and bound technique to establish a feasible integer lower bound relatively early in the iterative process. This is then used as a benchmark for the bounds of other

nodes, arising from the solution of their relaxed LP, and, hence, as an aid in the fathoming process.

If the solution to the maximum is restricted to an integer value an overall solution to the IP would have been found and the algorithm terminated as a result of the knowledge that the bound on the root node is less than 28. Without this added integer restriction, however, attention must be focused on any unexplored nodes. LP2 is, therefore, considered. The relaxed LP, having the added restriction $x_1 \geq 3$, i.e. LP2, is solved. The continuous optimal solution has a potential branching variable, $x_2 = 1.667$ ($x_1 = 3$, therefore feasible). Further branching is, however, unnecessary. Using accumulated knowledge this node can be deemed fathomed. The continuous optimal solution to the subset is $z = 26.33$ and, hence, is not comparable to the integer lower bound incumbent.

A node is deemed fathomed if the sub-problem yields a feasible integer solution to the IP problem or if the sub-problem cannot yield a better feasible than the best available lower bound of the IP problem. The sub-problem which has no feasible solution, either continuous or integer, is a special case of the latter. Having fathomed all nodes the branch and bound algorithm is terminated

In summary the branch and bound algorithm precedes as follows:

- *Initialization:* Set a lower bound, say $z_{(0)} = -\infty$, and $i = 0$, representing the initial iteration. Considering the original search space, ignore the integer restriction and solve the LP, i.e. LP0, using the simplex method. Express this solution as the bound for the root node. If this optimal solution vector is integer the root node is fathomed and the algorithm terminates. Else, the procedure continues.
- *Steps for each iteration, i ;*
 - (1) Select the most recently created, unfathomed node. (At $i = 0$ this will be the root node). Arbitrarily select a branching variable, x_j (where $j = 1, 2, \dots, n$), with optimum value x_j^* in the LP i solution. Create two branching nodes, LP($i + 1$) and LP($i + 2$), with additional constraints $x_j \leq \lfloor x_j^* \rfloor$ and $x_j \geq \lfloor x_j^* \rfloor + 1$ respectively (where $\lfloor x \rfloor$ is the integer part of x).

- (2) Apply the simplex method to the relaxed LP of each new sub-problem and obtain its bound. If the continuous optimal solution vector is integer, compare to the existing feasible lower bound and update the incumbent.
- (3) Apply the fathoming conditions to each sub-problem and discard those deemed to be fathomed. If both nodes, $LP(i + 1)$ and $LP(i + 2)$, are unfathomed proceed with the iterations along one branch, sequentially branching and fathoming until this branch is sufficiently partitioned such that fathomed nodes are reached. Return, then, to the other branch.

Termination of the branch and bound algorithm occurs when all nodes are deemed to be fathomed.

As regards the decision on the iteration sequence, the most recently created node is considered as this reduces computational effort regarding the solution of the relaxed LP. This selection procedure may vary, a possible choice being the node with the most promising continuous bound (Ref. 48). The algorithm can also vary in its method for choice of branching variable, as opposed to the arbitrary approach stated above (Ref. 83).

Zero-one Method

A special case of the branch and bound technique is the zero-one or additive method, introduced initially by Balas as a new approach in 1965, before its similarities to the branch and bound were recognized (Ref. 30,83). This method is carried out on a binary linear program (BIP). Although this seems very restrictive, any pure IP can be converted into a binary problem using conversion techniques (Ref.'s 30,48).

This method differs from the branch and bound in that it does not require the simplex method in its solution. The objective function is expressed solely in terms of positive coefficients. The binary variables are partitioned in various structured ways, some fixed at 1, others 0 and the rest classed as 'free'. Bounds are then assigned to each partition by adding coefficients of the objective function relating to the values of the binary variables for the particular partition, hence, the name the additive method.

Cutting Methods

A cutting plane for an IP problem is a new constraint that reduces the feasible region for the LP relaxation without eliminating any feasible integer solutions to the IP problem, as shown in Figure 3.4.

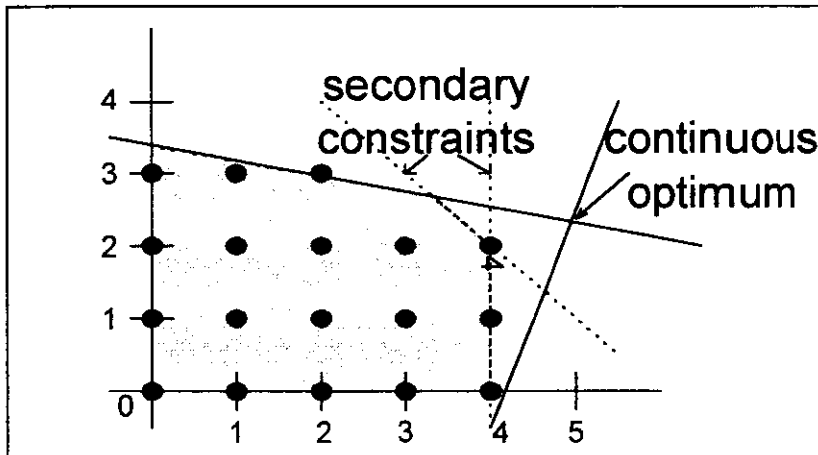


Figure 3.4 The Cutting Plane Technique

As in the branch and bound approach the corresponding relaxed LP is first solved for its optimal solution. If this solution is all integer the optimal integer solution has been found and the process terminates. If at least one variable is non-integer, the next step is to establish a secondary constraint to add to the relaxed LP model.

The fractional method to derive a cutting plane proceeds as follows. It should be noted, prior to solving the relaxed LP for the fractional cut method constraints should be manipulated to ensure all coefficients are integer and not left as fractions.

A source row is chosen from the simplex tableau, generated to solve the relaxed LP. The row selected must be one of the constraint equations corresponding to a non-integer solution. As a general rule the source row with the largest truncated value, having removed the integer part of the right hand-side constraint value, is chosen. This constraint equation is used to generate the *fractional cut*, which is then added to the simplex tableau. Derivation of the fractional cut varies according to whether a mixed or pure IP (MIP or IP) is being considered (Ref. 42). An optimal solution to the amended LP is then found using the dual

simplex method, which is equivalent to cutting of the solution space towards the optimal integer solution.

If application of the dual simplex method results in an integer solution the process ends. Else, a new fractional cut is generated from the resulting tableau and the algorithm proceeds until an integer solution is obtained.

Further discussion focuses on the choice of source row to derive the fractional cut (Ref. 42). Different cuts may be generated from the same simplex tableau. The aim is to obtain the strongest cut, i.e. that which cuts deepest into the search space, without loss of feasible integer solutions. The basic idea behind a cutting plane method remains the same. Methods differ in their derivation of the cutting plane (Ref. 61,83). The fractional cut mentioned above is a standard approach. An alternative approach can be applied to a pure BIP (MP).

Cutting plane methods have two major problem areas. Firstly, the introduction of round-off errors that may distort the original data. Secondly, no feasible integer solution exists until the optimal point is attained. Thus, if the process is terminated early no information is gathered.

Branch and Cut Method

A general conclusion is that the cutting technique alone cannot be used effectively to solve the general integer problem. Recent research has used aspects of the cutting technique within the context of the branch and bound algorithm, the aim being to move the solution of each sub-problem towards the desired integer solution. This is the basic concept behind a new approach termed the branch and cut technique (Ref. 42, 61 and 89).

Initially the branch and cut technique was introduced as an algorithm to solve pure BIP problems. Recently it has been applied to the mixed BIP (Ref. 89). Successful outcomes from this method have resulted for large problems in excess of thousands of variables. Typically, the approach depends heavily on the sparsity of the constraint coefficient matrix. In addition to various other characteristics of problems which have not yet been pinpointed.

Briefly stated the method uses the generation of cutting planes in conjunction with clever branch and bound techniques. These, however, are preceded by a technique referred to as automatic problem processing. Automatic problem processing involves inspection by an automated computer process of the initial IP model. This process reforms the original model to make the problem quicker to solve without eliminating any feasible solutions. As regards a BIP this is achieved through three techniques: fixing variables, eliminating redundant techniques and tightening constraints.

Branch and cut methods for IP solve a sequence of LP problems. Traditionally these LP relaxations have been solved using the simplex method. Mitchell in 1997 (Ref. 61) considers the use of the interior point method as an alternative to solve such problems. For a more detailed description of interior point methods see Reference 96. A simplex tableau is not available when using an interior point method. It is, therefore, necessary to modify the classical techniques for fixing variables and generating cutting planes from the optimal tableau.

3.2.3 Reliability and Linear Programming

In the area of system reliability much work has been done on the redundancy allocation problem (Ref. 25, 87). Typically, a system has N subsystems in series where components may be added in parallel or in series to each subsystem to optimise the system reliability or some other objective without violating the system constraints. Figure 3.5 shows network diagrams of such structures. The aim is to select the optimum number of redundant components for each subsystem. (It is assumed redundant units are operating in parallel (hot standby), subject to failure.

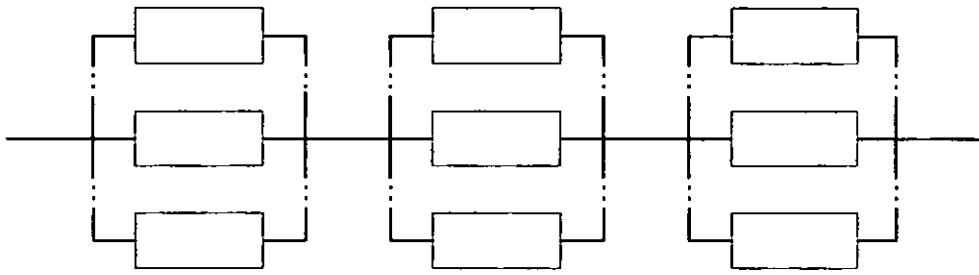


Figure 3.5 A Circuit Diagram

Reliability expressions can be developed for such network configurations.

Consider one of the simplest systems, consisting of a set of n components in series. If each component is independent in operation, each contributes to the system failure mode under consideration and the system is to operate for a fixed period of time, then the reliability of the system is the product of the individual component reliabilities.

$$R_{SYS} = \prod_{i=1}^n R_i \quad (3.3)$$

where R_{SYS} is the system reliability and R_i the reliability of components i, \dots, n .

Aggarwal in 1975 (Ref. 2) uses the simplest approach to establish the solution of a redundancy allocation problem. The algorithm consists of adding redundancy to the stage where unreliability is so far the highest. Previous such methods add an extra level of redundancy having ensured the design remains within the permissible region. The method described in this paper takes into account the constraints, analysing factors such as whether more than one element can be added to the stage and the design still be permitted.

Further research illustrates the formulation of reliability optimisation problems for a variety of series/parallel systems in this manner and solves them as IP problems (Ref. 88). Specifically the following problems are addressed: (1) Maximum reliability of a parallel redundant system subject to several non-linear constraints, (2) Minimum cost of a series redundant system subject to several non-linear constraints, while maintaining an acceptable level of reliability.

Specifically Ghare in 1975 (Ref. 33) applies a branch and bound method to a reliability optimisation problem. A later paper (Ref. 55) compares this to a couple

of approximation techniques (The algorithm in Ref. 2 being one of them). The branch and bound approach required greater computer effort in both time to run and programming, but gave a far higher guarantee of obtaining an optimal solution. Further exact techniques such as cutting plane and implicit search have been applied to similar redundancy problems.

3.2.4 Further Research

Recent research on traditional integer programming methods focuses on applying the techniques to a wider array of problem types, as opposed to the limited single criterion linear objective function and linear constraint model.

Shi in 1997 (Ref. 78) introduces a branch and bound approach to solve a binary integer linear program with multiple criteria and multi-constraint levels, using the framework of a multiple criteria LP (MCLP). The motivation being that many real-world problems should be classified as an IP problem under a multiple criteria environment. Integer design variables are converted to their binary equivalent. A modified form of the simplex method is then used to solve the relaxed multiple criteria model. This MCLPP can be formulated as

$$\begin{aligned}
 &\text{maximise} && \alpha C x \\
 &\text{subject to} && A x \leq D\beta \\
 &&& x \geq 0
 \end{aligned} \tag{3.4}$$

where, C , A and D are matrices representing the objective function and constraints respectively, α is a criteria parameter and β a constraint level parameter. (α and β are assumed unknown and explain the probability distribution for the sub-problem bounds described later).

The branch and bound procedure is then adopted. Once the branching is made by partitioning the original feasible set into smaller subsets, each sub-problem needs to be solved to obtain its bound. The upper bound of a sub-problem is defined as the expected objective value of its relaxation problem with a probability distribution over the parameters for multiple criteria and multiple constraint levels.

3.2.5 Summary

Disadvantages

In general exact integer techniques are dependent on the nature of the objective function and constraints. This is due to the fundamental requirement on sequential use of the simplex method as the algorithms progress. Ironically, the feature which makes these methods so effective. Its application requires the problem to be described in mathematical terms so that the functions and variables can be manipulated by standard rules. In addition, these functions are generally enforced to be linear. Applications of implicit enumeration, such as the additive algorithm, do not use the simplex method, but require a monotonic objective function that is separable.

The simplex algorithm is unable to cope with a black box type function, which does not have an explicit form to relate the decision variables over the search space. Similarly implicit constraint types and generally non-linear constraints are difficult to deal with. All features which are present in the safety system optimisation problem.

Application of the interior point method to solve the LP, fix variables and develop cuts has similar disadvantages. The interior point method does not involve the complexities of the simplex tableau, but an objective function and relatively strict mathematical formulation of the problem is still required.

Reliability expressions can be formulated for series and parallel network configurations. Methods to formulate such expressions are, however, dependent on the systems structure. As networks become more complex, methods to calculate the system failure probability become more comprehensive. As the complexity increases so to does the ability to obtain the network in the first place. The use of network configurations, both in terms of construction and evaluation, for the demand of the safety system under consideration is unrealistic and too simplistic.

For a fixed number of integer variables BIP problems are, in the main, easier to solve than problems with general integer variables. Conversion of general integer

variables to their binary equivalent very rapidly increases the dimensionality of the problem. One of the most important determinants of computation time and solvability of IP problems is the number of integer design variables. The set of variables in the safety system problem creates a difficult mixture for IP application. Generally, the small integer variable range and presence of binary variables eliminates the possibility of rounding. The ratio of binary variables does not, however, encourage the conversion to a pure BIP.

Advantages

Many features of traditional IP techniques and their recent developments restrict their implementation to the safety system optimisation problem. Certain characteristics and concepts do, however, show potential and may be modified to the design optimisation problem.

Developing expressions for the reliability of a subsystem from network configurations of redundant systems opens up a whole new range of possibilities as regards the application of IP techniques. The system viewed as a whole with binary decision variables, maintenance test times and so on, is too complex for such an approach. An idea, perhaps, may be to partition the system configuration and isolate those areas, which could be expressed as a series/parallel network. For example, the number of pressure transmitters and the number required to trip forms a standard parallel-voting network. An IP technique could be applied, using the reliability expression for the voting network, to this section. This contributes an avenue of investigation though many other factors must be considered, such as the dependency of the system reliability on the whole system, i.e. interactions between subsystems. The effect of altering redundancy in the pressure transmitter voting subsystem is dependent on the structure of other components within the system.

The Branch and bound algorithm follows a tree search procedure where at each step all possible solutions of the current problem are partitioned into two or more subsets. Each subset is assigned a bound and, depending on this value is either dismissed as worse than an existing solution or the subset is further partitioned. If it is a maximisation problem an upper bound to each subset is applied. Estimating

this upper bound, or evaluating it exactly is a complex issue in itself. The branch and bound is dependent on the accuracy of this estimate. In addition, it is never known how close the current largest function value obtained is to the global maximiser. A recently proposed method, partitioned random search (PRS) (Ref. 86) is a tree search type of algorithm, which attempts to overcome this complication of the branch and bound approach. This is discussed in a later Section of the chapter.

3.2.6 Logical Search

Having studied the structured traditional LP techniques a more applicable approach to the safety system design optimisation points to the use of search logic. Misra in 1991 (Ref. 60) proposed a simple and efficient technique for solving integer-programming problems that generally arise in system reliability design. It involves a systematic search near the boundary of constraints and involves function evaluations only. In addition the algorithm is designed to solve problems in which the decision variables are restricted to integer values. Motivation behind the approach considers the time and effort wasted analysing points well within the feasible space. It is logical to assume the optimal point will lie close to the boundary of the region since, in the main, a better design will result from exhaustive utilisation of the systems resources.

The basic idea is to set up an initial design vector x . The next step fixes x_1 as large as possible without breaking the constraints and the remaining $n - 1$ variables are set at their lower bound. Each variable is considered in turn and increased such that the constraints are not violated. The value at a design point is only calculated when the design falls within tolerable slacks, i.e. a specified distance from or about the boundary, specified for the constraints. This ensures that only those feasible points that are close to the boundary of the constraints are considered for function evaluations. The approach resembles a general technique, which alternately considers and fixes variables (Ref. 61). The decision of which variable to fix and at what value, rather than being based on the design vector of the optimal relaxed solution, is based on past function evaluations, inter-dependency of variables and the distance from the constraint

boundary. The reader is referred to Reference 18 for a more detailed description of the steps of the algorithm.

A disadvantage to this approach is the intensive demand on function evaluations. By virtue, however, little knowledge of objective and constraint functions is required, integrality of the decision variables is maintained and function evaluations are limited by the innovative inclusion of tolerable slacks about the constraints. The algorithm lends itself nicely to optimisation of the safety system design problem.

3.3 Gradient Techniques

Many optimisation methods employ gradient information. A traditional means to classify such methods is to refer to those that only use first derivatives as first order methods and to those, which use both first and second derivatives as second order methods. The objective of this section is to introduce initially the fundamental first and second order gradient approaches for unconstrained optimisation. These provide a framework from which many more methods are formed. The more sophisticated application of gradient methods to the constrained optimisation problem will then be considered.

It is assumed throughout this section that the function and its first and second derivatives exist and are continuous. For reasons of consistency the minimisation problem (as opposed to maximisation) is considered. Most gradient methods employ a similar iteration procedure

$$x^{(k+1)} = x^{(k)} + a^{(k)}s(x^{(k)}) \quad (3.5)$$

where, $x^{(k)}$ is the current estimated minimum point, $a^{(k)}$ the step length parameter, $s(x^{(k)})$ the search direction in the space of the design variables from the current point. A particular method is characterised by the manner in which the search direction and step length are determined at each iteration.

3.3.1 Basic Methods

The Method of Steepest Descent

The method of steepest descent, also known as Cauchy's method, is the fundamental first order method. Cauchy's method uses the Jacobian gradient, i.e. the first partial derivatives of $f(x)$ with respect to x denoted by $\nabla f(x)$, evaluated at some point $x^{(k)}$ to determine the search direction. The gradient of a function points in the direction of the greatest increase in the value of the function and is orthogonal to the contours of $f(x)$ that pass through $x^{(k)}$. The negative of the gradient is the direction of steepest descent. The search direction is, therefore, the negative unit vector of the Jacobian about the current point approximation

$$s(x^{(k)}) = -\frac{\nabla f(x^{(k)})}{\|\nabla f(x^{(k)})\|} \quad (3.6)$$

Using equation (3.6) the new point is

$$x^{(k+1)} = x^{(k)} - a^{(k)} \frac{\nabla f(x^{(k)})}{\|\nabla f(x^{(k)})\|} \quad (3.7)$$

The unit vector of the Jacobian defines only the direction of search. This in conjunction with the step length establishes the new point. Approaches to the method of steepest descent differ in their derivation of the step length. In the main, $a^{(k)}$ is determined so that $f(x^{(k+1)})$ is a minimum along the search direction using an appropriate line search. A less general approach selects a fixed, or adjustable, value for a . If adjustable, a is then reduced as the minimum is approached.

The steepest descent method is iterative due to the first direction of movement yielding only a restricted minimum, which is often not toward the actual minimum. Convergence, though guaranteed, is often not realistic, as the number of iterations required is intolerably large. This is particularly true close to the minimum. Steepest descent is most effective far away from the minimum where the step length is large and the number of iterations required to reach the vicinity of the minimum generally low.

Newton's Method

Cauchy's method is in effect a Taylor expansion about the objective function truncated linearly. Second order methods incorporate the third term of the Taylor series, thus establishing a quadratic approximation to the function value at the new point, $x + \Delta x$

$$f(x^{(k+1)}) \approx f(x^{(k)}) + g^T \Delta x^{(k)} + \frac{1}{2} (\Delta x^{(k)})^T \mathbf{H} \Delta x^{(k)} \quad (3.8)$$

where,

$$g^T = \nabla^T f(x^{(k)})$$
$$\mathbf{H} = \nabla^2 f(x^{(k)})$$

Second order methods make use of the Hessian matrix, i.e. the second partial derivatives of $f(x)$ with respect to x evaluated at $x^{(k)}$, denoted by \mathbf{H} . To approximate Δx , the Taylor expansion in equation (3.8) is partially differentiated with respect to each of the components of Δx and the resulting expression equated to zero to give

$$\Delta x^{(k)} = -\mathbf{H}^{-1} g \quad (3.9)$$

Direct use of this search direction in the standard formula over successive iterations from an initial point is known as Newton's method, that is

$$x^{(k+1)} = x^{(k)} - \mathbf{H}^{-1} g \quad (3.10)$$

At the minimum the Jacobian is equal to 0 and it is required that the inverse Hessian matrix is positive definite. Throughout the iterative process using the current approximate to the minimum this is not always the case. Hence, search directions diverging from the minimum may be derived. Regarding points close to the minimum, where the quadratic approximation of the function is most accurate, the Hessian matrix generally portrays positive definite characteristics. In consequence, Newton's method holds no guarantee that points located away from the minimum will converge. Both this and the mathematical effort required to calculate the second partial derivatives and invert the Hessian at each iteration complicate the application of Newton's method.

The modified Newton's method goes some way to alleviate divergence of points far from the minimum. The modification simply involves adding a line search, as in Cauchy's method, that is

$$x^{(k+1)} = x^{(k)} - a^{(k)}\mathbf{H}^{-1}g \quad (3.11)$$

where, $a^{(k)}$ is chosen such that $f(x^{(k+1)})$ tends to a minimum. This ensures that the value of the function at the new point is less than the value at the current point.

The conflicting positive attributes of first and second order methods naturally instigate the idea to precede second order methods by first order approaches, often steepest descent, away from the minimum. As the vicinity of the minimum is reached a second order method takes over.

3.3.2 Advanced Methods

Conjugate Directions

Conjugate directions use the history of previous iterations and aim to mimic the positive characteristics of Newton's method, i.e. accelerated search close to the minimum. They avoid, however, the mathematical effort and storage requirements associated with use of the Hessian matrix. Additionally, conjugate directions attempt to incorporate the robust characteristics of steepest descent away from the minimum.

Conjugate directions are based upon the model of a quadratic objective function. Functions not in this form are approximated to be so. The theoretical basis for this is that any function closely represents a quadratic close to the solution. Hence, the problem is of the form

$$\text{Minimise } \frac{1}{2}x^T Cx + b^T x + a \quad (3.12)$$

where, C is a positive definite matrix and b a vector. Transforming this quadratic function so that it is the sum of perfect squares, enables the optimum to be found exactly after n single variable searches. Equation (3.12) is expressed in terms of a

new co-ordinate system. From a graphical viewpoint, consider the general quadratic function with cross terms. The transformation chooses a new vector set, i.e. co-ordinate axis, to coincide with the major and minor axes of the quadratic. Single variable searches, therefore, correspond to searching along each of the principle axes of the quadratic.

Given a symmetric matrix C , two vectors u_1 and u_2 are said to be C -orthogonal or conjugate with respect to C if $u_1^T C u_2 = 0$. It can be shown that the new co-ordinate axes define a set of n column vectors u_1, \dots, u_n that are a mutually orthogonal set of conjugate directions with respect to C and are the eigenvectors of C (Ref. 43). These vectors are linearly independent and can, therefore, represent any other vector in n -space. Given an initial estimate $x^{(0)}$ to the minimum, the actual solution can be expanded in terms of the orthogonal set and the initial approximate as

$$x^* = x^{(0)} + \sum_{i=1}^n a_i u_i \quad (3.13)$$

where $a_i, i = 1$ to n , is a coefficient.

In summary, if a suitable set of conjugate directions can be obtained, the optimum of a quadratic function can be found by exactly n single variable searches with respect to each conjugate direction $u_i, i = 1, \dots, n$.

Powell devised a method to generate the set of conjugate directions using objective function values only. For the general case a quadratic approximation to the objective function and gradient information can be employed to generate conjugate directions. The most common approach is the conjugate gradient method.

The Conjugate Gradient Method

The conjugate gradient method sequentially builds a set of conjugate gradients as the algorithm progresses, as opposed to specifying the set beforehand. Each search direction is selected at the current point such that they show conjugacy to those previously selected. At the k^{th} iteration the current negative gradient vector is added to a linear combination of the previous direction vectors to obtain a new conjugate direction along which to move. It can be shown (Ref. 69) that the general form of each search direction is

$$s(x^{(k)}) = -\nabla f(x^{(k)}) + \left[\frac{\|\nabla f(x^{(k)})\|^2}{\|\nabla f(x^{(k-1)})\|^2} \right] s(x^{(k-1)}) \quad (3.14)$$

This is the suggested form by Fletcher and Reeves.

Using the conjugate gradient method the quadratic model will be solved in n steps. An approximated model may require further iterations, but in general this method has proved efficient. Potential difficulties arise if a search direction generated is linearly dependent. The n -dimensional solution space for future iterations is reduced and the actual solution possibly eliminated. Periodic restarts of the method are sometimes introduced to overcome the derivation of a dependent direction.

The method of parallel tangents, or the partan method, due to Shah, Buehler and Kempthorne (Ref. 77) uses the conjugate gradient method in conjunction with steepest descent steps. The motivation behind the method is that the solution to steepest descent lies close to where the two straight lines meet which bound the zigzag path of the steepest descents iterative steps.

Recent research by Liu, Hu and Storey (Ref. 46, 52) has proposed a new generalised conjugate gradient algorithm and compared it, via application to six functions, to other modified conjugate gradient and quasi-Newton methods. The new method uses a combination of the previous search direction and the current gradient to form the new direction per iteration, as in the conventional approach. The new direction is formed such that it is conjugate to a vector perpendicular to the current vector, instead of the new direction being conjugate to the previous

direction as before. The results show promise in terms of efficiency in the number of iterations and storage requirements.

Quasi-Newton Methods

Quasi-Newton methods are a further class of techniques using principles lying somewhere between the method of steepest descent and Newton's method. The motivation behind the quasi-Newton technique is to overcome the impracticality of evaluating the inverse Hessian at each step. Specifically, an approximation to the Hessian is made using only first derivative information.

Quasi-Newton methods are based primarily on quadratic functions as for conjugate gradients. The expectation is that success with quadratic functions will lead to success with general non-linear function. The methods differ in their manner and complexity of approximating the inverse.

The classical modified Newton's method is the simplest in the class of techniques. The true inverse Hessian is established at the initial point, denoted by $\mathbf{F}(x^{(0)})^{-1}$. The Hessian evaluated at the initial point is then used throughout the iterative process

$$x^{(k+1)} = x^{(k)} - \alpha^{(k)} \mathbf{F}(x^{(0)})^{-1} \nabla f(x^{(k)})^T \quad (3.15)$$

The effectiveness of this approach is governed by the magnitude of the third derivatives of the objective function.

Most quasi-Newton methods are more complicated than the classical approach and aim to use information gathered from each iteration as the algorithm progresses. At the k^{th} stage the previous Hessian is used to define the new descent direction. The intention is that as the sequence proceeds the approximated inverse Hessian tends toward the actual inverse Hessian derived about the optimal point

$$\mathbf{H}^{-1} \Rightarrow \nabla^2 f(x^*)^{-1} \quad (3.16)$$

Many such techniques are called variable metric methods due to the inverse Hessian changing at each iteration.

The most widely used method to approximate the inverse Hessian is the Davidon-Fletcher-Powell (DFP) method (Ref. 69). It can be shown (Ref. 17) that

$$\mathbf{A}^{(k)} = \mathbf{A}^{(k-1)} + \frac{\Delta \mathbf{x}^{(k-1)} \Delta \mathbf{x}^{(k-1)T}}{\Delta \mathbf{x}^{(k-1)T} \Delta \mathbf{g}^{(k-1)}} - \frac{\mathbf{A}^{(k-1)} \Delta \mathbf{g}^{(k-1)} \Delta \mathbf{g}^{(k-1)T} \mathbf{A}^{(k-1)}}{\Delta \mathbf{g}^{(k-1)T} \mathbf{A}^{(k-1)} \Delta \mathbf{g}^{(k-1)}} \quad (3.17)$$

where,

$$\begin{aligned} \Delta \mathbf{x}^{(k)} &= \mathbf{x}^{(k+1)} - \mathbf{x}^{(k)} \\ \Delta \mathbf{g}^{(k)} &= \nabla f(\mathbf{x}^{(k+1)}) - \nabla f(\mathbf{x}^{(k)}) \end{aligned}$$

where, $\mathbf{A}^{(k)}$ is an estimate to the inverse of the Hessian. In the main, $\mathbf{A}^{(0)} = \mathbf{I}$. The DFP formula maintains the symmetric positive definite characteristics of the \mathbf{A} matrix.

Broyden, Fletcher and Shanno (Ref. 43) proposed an alternative formula to update the inverse Hessian that has gained significant approval.

A common difficulty of quasi-Newton techniques is the tendency of $\mathbf{A}^{(k+1)}$ to be ill conditioned. A means to lessen this problem is to incorporate a restart procedure into the algorithm.

3.3.3 Constrained Gradient Techniques

The general constrained optimisation problem introduces additional complexities. The abundance of research and available methods to cope with unconstrained optimisation problems and the linearly constrained case encourage transformation of the constrained problem in order to apply such traditional techniques. These modifications fall into two categories.

The penalty concept reformulates the general constrained model into a sequence of unconstrained problems via the addition of specifically derived penalties should constraints be violated.

A second approach is to convert the general problem into a constrained but linear model. In the main, a Taylor expansion of the objective function and any non-linear constraints is carried out about the current point and truncated linearly. This opens up a vast array of applicable optimisation techniques. These are, in the main, from the class of linear programming techniques. Thus, the issue becomes an LP problem.

The standard approach to such linearisation methods is to approximate a linear model at an initial point $x^{(0)}$ carry out the LP optimisation technique and, thus, establish a new improved point $x^{(1)}$. The dilemma with this approach is that the Taylor approximation is only deemed accurate in the vicinity of the point about which it is derived. The estimated value of the new point is open to a large degree of inaccuracy.

A means to overcome inaccuracies associated with linearisation is to apply what are termed direction-generation methods based on linearisation. A linear approximation (i.e. the gradient) is evaluated at an initial point to determine a direction of search. The actual objective function value and those of the constraints are used to guide the search along this direction. The following section introduces the prominent gradient-based techniques for general constrained problems.

The Method of Feasible Directions

The most basic concept is the method of feasible directions. Starting from an initial point, a direction is specified. Steps are then taken through the search region from the initial point in the specified direction to a new point.

$$x^{(k+1)} = x^{(k)} + a^{(k)}d^{(k)} \quad (3.18)$$

where $d^{(k)}$ is the direction vector. The difference between this and the unconstrained gradient methods is that it must be ensured that the steps in the specified direction are feasible, i.e. there exists $\bar{a} > 0$ such that, for all $0 \leq a \leq \bar{a}$, $x^{(k+1)}$ is still in the feasible region.

This basic approach has a couple of major pitfalls. For certain problems a linear direction vector emanating from a point may never be feasible. Consider, for example, a problem with non-linear equality constraints. A second problem is that the feasible direction methods are subject to jamming, i.e. convergence to a point other than a local minimum. This occurs because the steps that the method takes become shorter and shorter as the direction vectors alternate between closely adjacent boundaries due to different constraints coming into play.

A means to combat jamming is to partition constraints into active and inactive sets, creating what is termed a working set. An inequality constraint $m_i(x) \leq 0$ is said to be active at a feasible point if $m_i(x) = 0$ and inactive if $m_i(x) < 0$. To maintain consistency an equality constraint is active at any feasible point. The idea is that within the neighbourhood of x active constraints restrict the domain of feasibility, whereas inactive constraints have no influence at this point. Constraints treated as inactive are, therefore, ignored. The working set is defined as those constraints, which are active at the current point, i.e. a subset of the total constraints.

Active set methods extend feasible directions to incorporate the partitioning of constraints, thus defining the working set. The algorithm proceeds from an initial point to move on the surface defined by the working set of constraints, i.e. the working surface, to an improved point. As steps are taken on the working surface different constraints come into play. The algorithm is, therefore, systematically dropping and adding constraints in the working set.

The Gradient Projection Method

Several methods are developed from this active set strategy. The gradient projection method is a commonly used and successfully applied example. The negative gradient established at a point via the conventional method of steepest descent for unconstrained problems is projected onto the working surface to define a direction of movement in the following manner.

The algorithm is initiated from a feasible point x . $W(x)$ is defined to be the working set at the given feasible point. The direction vector to be used is the

projection of the negative gradient onto this subspace (Ref. 43). To compute this projection use

$$\mathbf{P} = \left(\mathbf{I} - \mathbf{W}^T (\mathbf{W}\mathbf{W}^T)^{-1} \mathbf{W} \right) \quad (3.19)$$

where \mathbf{P} is the projection matrix corresponding to the working set, \mathbf{W} is the matrix composed of the rows of the working set, and the direction vector d is

$$d = -\mathbf{P}\nabla f(x)^T \quad (3.20)$$

The second step involves selecting both \bar{a} and a^* , where \bar{a} is the value such that the feasible line segment, $f(x + ad)$, is terminated and a^* finds the optimum value of the function along the line. If the optimal value occurs at the end of the segment, that is $\bar{a} = a^*$, a new constraint comes into play and is added to the working set.

In the main, we are dealing with an algorithm that solves equality constraints only. Thus, Lagrange multipliers and Kuhn-Tucker sufficient and necessary conditions at a minimum corresponding to the active constraints are required to solve the problem and define constraints to be added to and dropped from the working set. The reader is referred to Reference 6 for a discussion of Lagrange multipliers and Kuhn-Tucker conditions.

The gradient projection method is extended to deal with non-linear constraints. As previously, the projection matrix and gradient at the current point is used to define a direction of search. Equation (3.19) is, however, more complicated as linearisation of the constraints to create the working set matrix is required. Steps are taken in the direction of the projection matrix on the linearised working surface, the first step moving to point y . These steps are feasible only for the approximated constraints. Movement is then made in a perpendicular direction to the tangent plane to intersect the actual non-linear constraint and hence, return to the feasible region, as shown in Figure 3.6. Difficulties arise when steps taken along the tangent plane are too large. The result is that the perpendicular projections coincide with non-linear constraints that are no longer active. The point of intersection is, therefore, removed from the feasible region. In such

circumstances, an interpolation scheme must be used to return to the active constraints (Ref. 43).

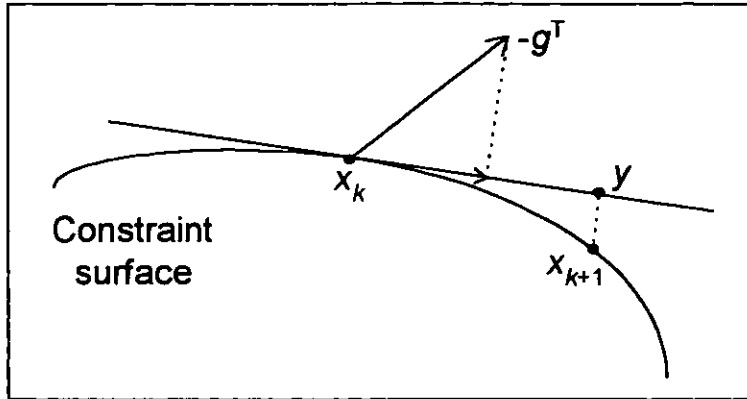


Figure 3.6 GPM with Non-linear Equality Constraints

The Reduced Gradient Method

Theoretically the reduced gradient method (RGM) (Ref. 69) behaves in a similar manner to the gradient projection method. Many ideas and attributes behind this method are, however, borrowed from the simplex method.

The reduced gradient method is an extension of the convex simplex method.

Consider a typical LP problem

$$\begin{aligned} \text{Minimize} \quad & f(x) = cx \\ \text{Subject to} \quad & Ax = b \quad x \geq 0 \end{aligned} \tag{3.21}$$

Applying the simplex method, the variable set x is partitioned into m basic variables, denoted by \hat{x} , which are all positive and $n - m$ non-basic variables, denoted by \bar{x} , which are all zero. In the same way the coefficient matrix A is partitioned into B and \bar{A} and the objective coefficient vector c into \hat{c} and \bar{c} . A basic feasible solution is derived. As the simplex algorithm proceeds the relative cost coefficients, \tilde{c} , are used to choose which non-basic variable is to enter the matrix

$$\tilde{c} = \bar{c} - \hat{c}B^{-1}\bar{A} \tag{3.22}$$

The non-basic variable corresponding to the smallest relative cost coefficient is chosen. The minimum ratio rule then selects the basic variable to leave the basis.

This approach is modified for the convex simplex method by replacing the linear objective coefficients with the linearised form of the non-linear objective function. Hence, the modified relative cost factor about a basic feasible solution $x^{(0)}$ is

$$\nabla \tilde{f}(x^{(0)}) = \nabla \bar{f}(x^{(0)}) - \nabla \hat{f}(x^{(0)}) \mathbf{B}^{-1} \mathbf{A} \quad (3.23)$$

where $\nabla \hat{f}$ relates to the partial derivatives with respect to the basic variables, $\nabla \bar{f}$ the non-basic variables and $\nabla \tilde{f}$ the relative change of the objective function at $x^{(0)}$ taking into account linear constraints.

If the non-basic variable corresponding to the largest negative $\nabla \tilde{f}(x^{(0)})$ component is increased, this will result in a reduction in the objective function. Its increase will simultaneously drive a basic variable to zero. In contrast to the general simplex procedure, the fact that all derivatives are evaluated at the current point means that as the basis is altered so to are the values of the derivatives. A minimum of $f(x)$ may, therefore, occur before the adjacent corner point is reached. The implication is that a line search should be carried out between the current point and the selected adjacent corner point.

A problem arises in that positive non-basic variables may occur. To accommodate this problem and enable continued selection of a non-basic variable to enter the basis the following constructions must be considered

$$\begin{aligned} \beta_r &= \min \{0, \nabla \tilde{f}_i : i = 1, \dots, n-m\} \\ \gamma_s &= \max \{0, \nabla \tilde{f}_i x_i : i = 1, \dots, n-m\} \end{aligned} \quad (3.24)$$

If $|\beta_r| > \gamma_s$, then \bar{x}_r should be increased, adjusting only the basic variables until some basic variable goes to zero. If $|\beta_r| \leq \gamma_s$, then \bar{x}_r should be decreased, adjusting only the basic variables until either the basic variable or \bar{x}_r itself goes to zero, whichever occurs first.

In summary, the convex simplex method requires the specification of an initial feasible point $x^{(0)}$, a partition of the problem variables $x = (\hat{x}, \bar{x})$ and linearisation

of the objective function. It is then established which non-basic variable to increase using equations (3.23) and (3.24) with the corresponding conditions. The target point, that is the adjacent corner point, is determined and a line search executed. The new point is, therefore established and if necessary the basis is updated.

A drawback to the convex simplex approach is that, unless a minimum to $f(x)$ occurs at the adjacent corner point, the basis will remain the same. This results in a static method. At worst the initial partition essentially amounts to the selection of a reference co-ordinate system in the non-basic variables.

The RGM is an extension of this approach that focuses on nullifying the one at a time search emphasis of the convex simplex method. This is achieved by altering a number of non-basic variables simultaneously. It is termed the RGM as it can be viewed as a typical gradient approach carried out in a reduced variable space. The partition of basic and non-basic variables and associated array manipulations are used to specify a feasible descent direction. The search direction is selected and refined such that it will lead to an improved feasible point. The method continues until convergence to a Kuhn-Tucker point is reached.

The means to calculate the descent direction is modified in later algorithms to accelerate the convergence rate. As oppose to using the ordinary steepest descent approach, conjugate directions and quasi-Newton constructions are employed. The generalised reduced gradient method (GRGM) enables the application to problems with both non-linear objective and constraint functions. In graphical terms, the descent direction is determined as for the RGM. An additional sub-routine is incorporated to drop perpendicular projections from the descent direction, thus establishing the nearest point on the feasible constraint surface.

Many similarities exist between the RGM and the GPM. The partitioning of active and inactive constraints in GPM correlates to the reduced variable space arising from the basic and non-basic partition of the RGM. In graphical terms the extension to the non-linear case for each method is fundamentally the same. However, the routine to carry out the perpendicular projection differs. Both techniques have proven to be a reliable and effective tool for solving a wide range of non-linear optimisation problems (Ref.'s 73, 74).

3.3.4 Summary

Gradient based approaches provide exceedingly reliable, robust and efficient tools for optimisation of a vast array of problems. Derivative information of the objective function and constraints determine the search direction, thus guiding the steps through the search region. The degree of knowledge of the function and its derivatives is, however, an essential and often restricting factor. Methods employing gradient information require that, in the least, $f(x)$ and $\nabla f(x)$ exist and are continuous. Second order methods include the assumption of existence and continuity of $\nabla^2 f(x)$

An integer restricted optimisation problem is by convention a grid of discrete sampling points. Hence, we cannot strictly formulate the partial derivatives. To apply any kind of gradient approach it must first be considered that a smooth curve is used to link all discrete points to give the marginal distribution of f as a function of x .

The GRGM has been implemented to several reliability optimisation problems (Ref. 88). It is stated that the decision variables must be considered as continuous. They are rounded to their nearest integer value following termination of the iterative procedure to obtain a feasible integer design vector solution. Such an approach would be prone to much rounding error in the safety system design variable. The variables are small and rounding or truncating is, therefore, significant.

Direction-generation methods based on linearisation require the formation of a linear objective function about a feasible point. The linear approximate of the objective function and derivative information is used to determine a direction of movement. The optimal step-size in the determined direction is governed by function evaluations of the actual objective function. For the safety system optimisation problem derivation of a linear objective function at a point is possible, see chapter 8 of this thesis. Assuming continuity of the decision variables, the corresponding gradient information can be used to define a direction of search. Problems arise when steps are taken from the current point. Movement away from the current point in the specified direction renders the linearised objective function inaccurate. Away from the point no explicit objective function

exists to determine the value of the function. As stated, the search region is governed by a black box type objective function, which requires an integer design vector to evaluate the probability of system unavailability at a specific point. A line search in the descent direction is, therefore, out of the question. The alternative is to take a fixed step in the descent direction. Multiplying the direction vector by some fixed value and adding this to the current point will result in a new design point. It must then be ensured that the new point is both an improvement over the current solution and feasible. There is no guarantee, in fact it would be a remarkable coincidence, if the new design point were integer and each variable remained within their specified bounds. Rounding non-integer elements of the new design vector and returning non-feasible elements within their specified bounds must, hence precede any actual system evaluations.

In conclusion the discrete nature, low value and small bounded range associated with the majority of design variables for the safety system optimisation problem render any gradient-based approaches impractical. The assumption of continuity introduces the need to round or truncate non-integer variables. The rounding procedure is a combinatorial problem in itself and prone to a huge degree of error. A practical use of gradient information as regards optimisation of the safety system is described in chapter 8 of this thesis. In this case, derivatives are used to approximate the objective function at a point via a numerical difference scheme. A grid sampling technique is then carried out in the direct vicinity of this point to ensure the integrality and feasibility of the design vectors analysed.

3.4 Direct Search Methods

Direct search methods establish a direction of search, carry out a linear search, or take fixed steps, in the designated direction and thus, iteratively produce estimates of the minimum (or maximum) of $f(x)$. The process is iterative as each direction is not necessarily directly toward the minimum. The fundamental concept behind search methods is similar to the gradient-based approaches. The difference being that search methods use the evaluation of function values only to define directions of search. Such methods do not require continuity of the objective function or existence of derivatives.

Search methods can be categorised into two classes. (1) *Heuristic methods*, these rely on geometric intuition and offer no performance guarantee. (2) *Theoretically based methods*, these have a mathematical foundation and mathematical proof of convergence may be possible. The *simplex search* and *pattern search* techniques are contrasting examples of the heuristic type. *Conjugate directions* are an example of the theoretically based approaches. These techniques are described in the following section.

3.4.1 Unconstrained Direct Search

Heuristic Direct Search Methods

The most basic direct search approach is to adjust one variable at a time whilst keeping the others fixed until a minimum is reached. This method is termed the *one at a time search*. It is ensured that each search direction is independent and spans the entire domain of $f(x)$. Each step of one overall iteration is a linear search parallel to each of the n co-ordinate axes, where n is the dimension of the search space. This approach often leads to almost exhaustive enumeration of the search space and is impractical for all but the most basic of functions, i.e. those with low dimensionality and very little interaction between variables.

Simplex Method

Spendly, Hext and Himsforth introduced a more structured approach involving the evaluation of the performance index in some pattern about a base design point (Ref. 69). It uses the fact that in n dimensions a regular simplex is a polyhedron composed of $n + 1$ mutually equidistant points which form its vertices. For example an equilateral triangle is a simplex in two dimensions. A new simplex can then be generated on any face of the old one by projecting a chosen vertex the proper distance through the centroid of the remaining vertices of the old simplex. The old vertex is deleted, the remaining points and newly projected 'reflection point' form the new simplex. Utilising these ideas the simplex method was formed.

To carry out the simplex algorithm basic calculations are required to generate a regular simplex about a base point. Reflection through the centroid involves calculation of the centroid itself. For a problem with n dimensions

$$x_c = \frac{1}{n} \sum_{\substack{i=0 \\ i \neq j}}^n x_i \quad (3.25)$$

where x_c is the centroid and x_j the point to be reflected. All points on the line from x_j through x_c are given by

$$x = x_j + \lambda(x_c - x_j) \quad (3.26)$$

The value λ determines the distance from the centroid where $\lambda = 1$ corresponds to the centroid itself and $\lambda = 2$ the reflected point chosen to retain regularity of the simplex.

Rules are required to overcome two modes of oscillation. *Straddling* the minimum occurs when the newly generated point is also the largest of the new simplex. It would consequently be reflected back to form the original simplex. To combat this the second largest point of the new simplex is reflected. The second type of oscillation occurs when the simplex rotates about one vertex, resulting in repetition of the simplex pattern. This is termed *cycling* and occurs in the vicinity of the minimum. When repeated use of one vertex is detected, the size of the simplex is reduced by some factor termed the reduction factor. Termination occurs when the size of the simplex is reduced below a specified criterion or variation in the function value over a few iterations is negligible. The reduction factor and termination criteria are generally user-specified parameters.

The steps of the simplex method are such that an initial point $x^{(0)}$, reduction factor α and termination parameter β are defined (assume a minimisation problem):

- (1) Select $n + 1$ mutually equidistant points to define a regular simplex.
- (2) Evaluate the function at each vertex. Locate the vertex with the largest function value, $x_h^{(0)}$.
- (3) Check for either mode of oscillation. Update $x_h^{(0)}$ if necessary.

- (4) Reflect the point $x_h^{(0)}$ through the centroid calculated from the remaining points. To retain the regularity of the simplex the reflected point lies at an equal distance on the opposite side of the centroid to $x_h^{(0)}$. Should cycling occur, reduce the size of the simplex by a factor α .
- (5) If the size of the simplex is greater than the termination parameter return to step 2 and continue with the newly formed simplex. Else stop.

The simplex method is limited in that its progress is slow. No previous history from past iterations is used to generate accelerated moves toward the minimum. On the contrary, once a simplex has been reduced there is no simple way to expand it again.

Nelder and Mead introduced modifications to combat these problems (Ref. 26). The modified algorithm begins with a regular simplex in n dimensions. The crux of the algorithm, however, is that there is no need to maintain regularity of the simplex as the search proceeds. The actions of contraction and expansion are introduced. Basically, the simplex is reflected as previously stated. Depending on the relation of the function value at the reflected point to the old vertex and the remaining vertices of the new simplex the point search along the line of equation (3.26) is expanded, contracted or maintained. The value λ in this equation is replaced by α , β or γ representing normal reflection, contraction and expansion respectively. Nelder and Mead recommend that the values of $\alpha = 1$, $\beta = 0.5$ and $\gamma = 2$ be employed.

Pattern Search

Pattern search is a second heuristic approach devised by Hooke and Jeeves (Ref.'s 43, 69). This method emphasises more clearly working with a set of direction vectors to guide the search as opposed to manipulating a pattern of trial points. It is made up of two stages namely 'exploratory' moves and 'pattern' or acceleration moves. Past history amalgamated from the sequence of iterations is incorporated into the algorithm.

The exploration phase involves a one at a time search about a base point, i.e. a local search in n dimensions of set increments about a point vector. The value of the objective function is established at the base point $f(x^{(0)})$. Each variable is

then incremented by Δx in rotation. If $f(x^{(0)} + \Delta x_i) < f(x^{(0)})$, where $i = 1, \dots, n$, the change in element x_i is accepted, else x_i is incremented by $-\Delta x_i$. The lowest function value, i.e. $f(x^{(0)} + \Delta x_i)$, $f(x^{(0)})$ or $f(x^{(0)} - \Delta x_i)$ is accepted. The exploratory search thus establishes the lowest point, $x_{\min}^{(k)}$, about the base point, $x^{(k)}$.

To describe the algorithm further the following notation is employed.

- $x^{(k)}$ current base point
- $x^{(k-1)}$ previous base point
- $x_p^{(k+1)}$ pattern move point
- $x^{(k+1)}$ next (i.e. new) base point

The second phase involves movement from the minimum about $x^{(k-1)}$ through the minimum about $x^{(k)}$ to an equal distance the other side of the latter point. This is termed a pattern move and defines the temporary pattern point $x_p^{(k+1)}$. An exploratory search is then carried out about $x_p^{(k+1)}$ and its lowest associated point established. If the resulting point is an improvement over the current minimum the pattern move is accepted and the minimum about $x_p^{(k+1)}$ becomes the new base point. Otherwise the pattern move is rejected is chosen to be the minimum about the current base point.

An exploratory search may fail in that the base point proves also to be the lowest. If this is the case the step size increment in each search direction is reduced and the search repeated. Termination occurs when the increment is smaller than a set criterion or the function values vary insignificantly over a specified number of iterations.

Numerous modifications have been made to the Hooke-Jeeves method. The most widely used is the introduction of expansion and contraction steps in both the exploratory search and the pattern move should function values prove better or worse respectively.

Conjugate Directions

Conjugate directions are an example of a theoretically based search method. Such methods require the objective function to be approximated as a quadratic in n dimensions.

$$f(x) = \frac{1}{2} x^T A x + b^T x + c \quad (3.27)$$

An orthogonal co-ordinate frame for the n dimension space is defined by the conjugate direction vectors associated with the matrix A . In particular the eigenvectors of A define the conjugate directions which represent the set of orthogonal principle axes for the quadratic surface defined by $f(x)$. (A more detailed explanation of conjugate directions is given in section 3.3.2).

As a direct search method a non gradient-based technique is required to determine linearly independent conjugate direction vectors. Powell proposed an iterative method (Ref. 43). The idea is to build up the vector set sequentially. The procedure begins with a one at a time search along each co-ordinate axes. The minimum about the initial point defines a new direction to replace one of the original co-ordinate axes. Specifically the direction is $x^{(k+1)} - x^{(k)}$, where $x^{(k+1)}$ is the minimum associated with the base point $x^{(k)}$. In addition, a check is proposed based on the determinant of the matrix of direction vectors to ensure that any direction vector introduced maintains linear independence of the matrix. The direction matrix, therefore, continues to span the n -dimensional space.

Rosenbrock introduced a two-phase method termed the method of rotating co-ordinates (Ref. 1). As the name suggests the co-ordinate system is rotated such that one of the search directions lies in the expected direction of the minimum. The first phase of each iteration defines a direction of search in a similar manner to a step of Powell's sequential development of conjugate directions. A linear search in each n dimension is added as oppose to a sole fixed step. The second phase changes the search direction vectors to align with the direction vector determined in phase one by a process of orthogonalisations.

3.4.2 The Constrained Case

The previous section considers only the unconstrained optimisation problem. These unconstrained methods can be modified to handle the more complex constrained scenario. The modifications used fall into two categories. Explicit modification of the constraints considers the detailed structure of the actual constraint functions. The implicit approach penalises the violation of any constraint by adding a sequence of penalty functions to the objective function. The latter, sequential unconstrained techniques are dealt with elsewhere and will only be touched upon here. The following section describes primarily the former explicit approach.

Given a feasible point x the most basic constrained method ensures that any deviation from this point remains in the feasible region via evaluation of the constraints. Should any constraints be violated the infeasible point is manipulated such that it is returned to the feasible state.

Consider the Hooke-Jeeves pattern search. In the exploratory phase a step in a particular co-ordinate direction may cross a constraint boundary, thus leaving the feasible region. As a consequence, one of two courses of action could be taken. The feasible point could be treated as a point with a very high objective function and thus rejected. Conversely, the linear search along the direction vector in question must be reduced by a factor until the feasible region is re-entered. In a similar manner if a pattern move results in an infeasible tentative base point, the acceleration step can be reduced by a fixed fraction until a feasible point is found. In practice, however, this simple approach introduces significant problems. Primarily, there exists the inability to deal with non-linear equality constraints. The factor reduction approach is a crude line search. A non-linear equality constraint may not be feasible at any point along the line. Contraction is, therefore, of no consequence. Secondly, slow convergence or at worst premature convergence may result. The search can degenerate into a series of line searches in a fixed set of directions. In particular, all n -dimensional searches from a base point may either lead outside the feasible region or to points with a worse performance value.

Rosenbrock incorporated an interesting extension to this fundamental constrained approach, introduced in conjunction with the method of rotating co-ordinates (Ref. 1). It was developed specifically to include constraints of the form $x_{L,i} \leq x_i \leq x_{U,i}, i = 1, 2, \dots, l > n$, where the first n constraints limit the range of the parameters x_1, \dots, x_n . The remaining constraints are of the form $x_{L,i} \leq f_i(x) \leq x_{U,i}$ and are few in number.

The extension involves the inclusion of a boundary region associated with each constraint. The basic idea being that a width δ_i beyond the limits of each boundary is accepted as part of the feasible region. A formula defines a depth of penetration, say λ , for the lower and upper boundary. The depth of penetration acts as a local penalty function, thus altering the objective function in the vicinity of the constraint within the boundary region. This approach enables an enhanced ability to search the region close to the constraint boundary.

To rectify the basic approach for dealing with constraints it is necessary to adjust the search directions whenever further progress is stymied. The intention is to redefine the set of search directions such that they move along or parallel to the constraint surface. A dilemma arises due to the assumption that direct searches are able to use only function values, that is they have no access to gradient information.

The Multiple Gradient Summation Technique

Klingman and Himmelblau address this problem to a degree with the introduction of the multiple gradient summation technique (Ref. 43). On encountering a constraint it is assumed that the minimum lies near to or on the constraint boundary. A linear combination of the function gradient and violated constraint gradients, $i = 1, \dots, p$, is defined by

$$-\frac{\nabla f^T}{|\nabla f|} - \sum_{i=1}^p \frac{\nabla g_i^T}{|\nabla g_i|} \quad (3.28)$$

and evaluated at the current point. It is envisaged that analytical evaluation of the violated constraint function gradients ∇g_i is possible. The gradient of the

objective function ∇f is approximated by the direction vector of the predicted pattern move.

The Complex Algorithm

An alternative approach is a modification of the simplex method referred to as the complex method proposed by Box (Ref. 12). The motivation behind this method is to use scattered or random search directions to mitigate the fact that the search directions do not adapt to the constraint surfaces. The original approach initiates a regular simplex about an initial point using basic mathematical calculation.

Considering the constrained case, deviation of the simplex may result in some infeasible vertices. To overcome this a set of p feasible points are chosen randomly, where $p \geq n + 1$, n being the dimension of the problem. The algorithm then proceeds in the normal manner. The performance value at of each point p is evaluated. Reflection of the largest of these points is carried out about the centroid of the remaining points. As in the pattern move of the Hooke-Jeeves pattern search, the reflected point may be infeasible. When this occurs the point is adjusted for feasibility. Firstly each variable is checked to ensure any element exceeding its bounds is modified to the closest bound limit, i.e.

$$\begin{aligned} \text{if } x_i^m < x_i^{(L)}, \text{ set } x_i^m &= x_i^{(L)} \\ \text{if } x_i^m > x_i^{(U)}, \text{ set } x_i^m &= x_i^{(U)} \end{aligned}$$

where x_i^m is the reflected point, $x_i^{(L)}$ and $x_i^{(U)}$ the lower and upper bound limits associated with the i th variable, $i = 1, \dots, n$. If the resulting vector is feasible the algorithm continues as normal. Otherwise, x^m is retracted half the distance to the centroid until it is feasible.

Box recommends the number of trial points $p = 2n$. A large number of vertices prevents the simplex from collapsing and flattening along the constraint boundary. In addition, it is recommended that the algorithm undergoes a specified number of restarts prior to termination. This combats premature convergence of the method due to the restricted search about the boundary of the region.

3.4.3 Further Research

Misra in 1973 (Ref. 59) demonstrates the application of the sequential simplex search to a reliability optimisation problem. The paper considers the optimal number of redundancies at various stages of the system to ensure maximum reliability, within the limits imposed on resources. Linear constraints only are considered. It is stated, however, that the inclusion of non-linear constraints is simple due to the requirement of function values alone.

Constraints are added to the objective function for system reliability in the form of a least squares model. It is suggested that the actual problem is to seek a solution that offers a maximum value for the objective function and at the same time uses the available resources to a maximum extent. A least squares penalty concept is, hence, used.

The redundancy allocation is an integer-programming problem where the allocations are allowed to take only integer values, which lie within a small range at a low value. A further point of interest from this study is that the decision variables are treated as continuous for solving the simplex method. A final optimal solution is obtained by rounding of the final solution obtained. The algorithm proved successful on the 4-stage system example.

3.4.4 Summary

In general direct search methods are heuristic in nature. That is, they are intuitively based, unsupported by rigorous theory and, hence, offer no guarantee of convergence. Conjugate directions have a modicum of mathematical foundation. In contrast, however, they stipulate the necessity that the objective function be approximated as a quadratic. This limits the flexibility of application of conjugate directions. The quadratic enforcement also reduces the global capabilities of conjugate directions, enforcing a somewhat local optimisation procedure.

A potential disadvantage of direct search methods is the intensive dependence on function evaluations. Problems, which require significant computer time and

effort to evaluate objective and constraint functions, render direct search methods impractical. The demand for function evaluations is directly related to the dimensionality of the problem. In particular, many direct search algorithms rely on several restarts to combat premature convergence others utilise a line search in each exploratory direction. The complex algorithm relies on a large number of trial points to prevent the simplex collapsing and flattening along the constraint border.

Further problems arise when considering the integer stipulation, restricted variable range and small values assigned to most decision variables in the safety system design optimisation problem. The assumption of divisibility is inherent in the formula to locate the centroid of a set of trial points. Evaluation of the centroid generally requires that the resulting non-integer elements are rounded or truncated. Problems with rounding concerning the safety system design problem have been discussed previously (section 3.2.2). Secondly, contraction and expansion along search directions are prevalent, particularly in the constrained case. In such circumstances, integrality can be maintained via choice of an integer parameter to carry out contraction and expansion. Concerning the safety system design, however, difficulties arise due to the small restricted variable range associated with most of the variables. Consider, for example, pattern search, increasing the acceleration parameter by the smallest possible integer value, i.e. 1, causes many decision variables to step beyond their upper bound limit. The simplex method and pattern search require the pattern of points and directional movements to grow and develop within the search region from iteration to iteration. The integer stipulation in conjunction with the small variable range for most of the design variables infers that an integer increment is too great.

Misra's paper in 1973 previously described (Ref. 59) uses a simplex algorithm to optimise a problem with a variable set having similar characteristics to the safety system design problem. The method uses rounding to obtain an integer solution and proves to be effective. Rounding is only introduced after termination of the algorithm. Following each iteration the algorithm proceeds with non-integer valued variables. This is possible as performance is expressed as an explicit objective function. Taking into account the whole process, rounding contributes only a very minor part and is therefore negligible. The safety system design problem does not have an explicit objective function. Integer values of the

decision variables are required to evaluate the performance of each design implicitly. Utilising the simplex method for the safety system design optimisation would require rounding to be carried out after every iteration. In addition, boolean variables constitute part of the variable set in the safety system design. A variable type which deems rounding illogical.

Conversely, the heuristic nature of direct search methods enables greater flexibility in their application to varying problem types. Direct search methods rely on function evaluations only and generally pay little attention to the structure of either objective or constraint functions. Such an approach is able to deal with the 'black box' type function of the safety system design problem. The lack of an explicit objective function over the whole search region is a major limitation of the safety system model. This attribute of most direct search methods is, thus, a particularly encouraging characteristic.

The multiple gradient summation method uses gradient information from the constraint functions only. As regards the safety system, there exists a differentiable objective function for the maintenance down time and cost constraints. It is an advantage to use any available gradient information. Handling the spurious trip constraint would, however, require further manipulation and innovation.

The idea of a 'fuzzy' boundary in Rosenbrock's constrained approach shows potential regarding the mdt, cost and spurious trip constraint of the safety system design problem. Allowing constraints to be violated should significant improvement in system reliability result, enables the algorithm to follow the real decision making progress.

The method of rotating co-ordinates and the complex method showed consideration of variable bounds. Random sampling procedures to generate values for variables within the specific stated ranges proffers a reasonable approach to maintain point feasibility. This fact is a motivation for the random search techniques considered in the next section.

3.5 Random Search Methods

Many optimisation problems are not represented by a well-structured objective function with convex or unimodal properties. Conversely, in practical situations there is little or no 'a priori' information about the search region. Frequently the search space under consideration possesses more than one local minimum. Local optimisation techniques may improve the solution by a few percent. Moving to a different area of the search space can result in a very significant change in performance, disregarded by the local approach. When little is known about the function in question the aim is to devise an optimisation algorithm that will distinguish between the local minima and locate the best local minimum. That is, to devise an algorithm to deal with the global optimisation problem.

The demand on function evaluations enforced by a thorough systematic search of the region is intolerable. The means to tackle global optimisation problems fall into 2 broad classes namely 'deterministic' and 'probabilistic'. As stated by Gomulka (Ref. 27), theoretical reasoning indicates it is unlikely that efficient deterministic algorithms for global optimisation could be discussed for general functions. Such approaches are only possible for a very restricted class of functions. It is expected that application of global optimisation methods will be probabilistic in nature.

3.5.1 Pure Random Search

The simplest probabilistic approach is the pure (or crude) random search. Trial points are generated via a random sampling procedure. The simplest approach is to use pseudo-random number generation routines available on most computers. Specifically, when decision variables are limited to a restricted variable range, each can be assigned a value using

$$x_i = x_i^{(L)} + r_i(x_i^{(U)} - x_i^{(L)}) \quad i = 1, \dots, n \quad (3.29)$$

where r_i is a random number distributed uniformly on the interval (0,1). To maintain integrality of the point vector ensure that

$$r_i = \frac{R_i}{(x_i^{(U)} - x_i^{(L)})} \quad (3.30)$$

where $0 \leq R_i \leq (x_i^{(U)} - x_i^{(L)})$ and integer. The objective and constraint values (if constraints are present) are calculated for each trial point to ensure feasibility. Each feasible trial point is then compared and the best retained. A specified number of trial points are analysed prior to termination. This simultaneous search is robust, simple and flexible in application. However, it is, in other than exceptional circumstances, impractical.

The positive attributes of a crude random search can be inter-twined with other optimisation techniques and used in an intelligent manner to achieve a robust yet efficient algorithm. The following section introduces several innovative techniques, which use a random search as their backbone.

3.5.2 Controlled Random Search

In the extreme a function may be quasi-random, i.e. there exists little or no correlation between the values of the function at neighbouring points. It is usual, however, to assume some correlation does exist even when a function is discontinuous or the variables discrete. The controlled random search (CRS) is motivated by this concept. It is reasonable to assume that point vectors with high performance, that is a low function value in the case of a minimisation problem, lie in a sub-domain containing at least a local minimum. Attention should, therefore, be focused on high quality sub-domains of the search space.

The basic CRS proposed by Price (Ref. 65) proceeds as follows. It is supposed that each of the n variables lies within specific stated ranges, thus defining the search domain.

- Step 1. N trial points are chosen at random from the feasible search space. Each trial point and corresponding function value are stored in array A .
- Step 2. P is selected from a set of potential trial points derived in the following manner

- a) $n + 1$ points are chosen at random from N . These points form a simplex in n -space with vertices R_1, \dots, R_{n+1} , where R_{n+1} is chosen arbitrarily as the pole.
- b) The centroid of the simplex, G , is calculated.
- c) P is the image point of the pole with respect to G , that is

$$P = 2G - R_{n+1}$$

In total ${}^N C_{n+1} \times (n+1)$ potential trial points could be chosen, thus creating the set of primary trial points.

Step 3. Check P for feasibility and establish f_p .

Step 4. Establish the point, M , resulting in the greatest function value, f_M , of the N points stored.

Step 5. If $f_p < f_M$ and P is feasible, replace M in the array with P .

Step 6. If $f_p > f_M$ or P is not feasible return to step 2 and repeat with an alternative primary trial point.

It is required that the probability of success of $f_p < f_M$ strikes a balance between a pure random search, thus introducing global diversity, and a structured algorithm, ultimately tending toward convergence. If the percentage of successes from the total number of trials at any stage falls below 50% a secondary trial point, Q , is introduced, where

$$Q = \frac{(G + R_{n+1})}{2} \quad (3.31)$$

Primary trial points are search orientated, whereas secondary trial points are conducive to convergence. A certain number of function evaluations may specify the stop criterion of the CRS algorithm. Else, termination may occur when either N points fall within a sufficiently small region of n -space or all function values stored in the array are within so many significant figures of one another.

Price introduced a new version, known as CRS2, of the original CRS algorithm (Ref. 66). CRS2 differs in that primary trial points only are included. No consideration is given to secondary trial points. Secondly, CRS2 differs in application of the Nelder-Mead type simplex application. Specifically, the pole of the simplex is not chosen randomly. The pole, R_1 , is chosen to be the point corresponding to the lowest function value of the N points in store. This reduces

the set of potential primary trial points to $n \times {}^{N-1}C_n$, n being the dimensionality of the simplex.

CRS2 biases the search to the region around the lowest function value in store in an attempt to achieve more rapid convergence than that attained by the original CRS algorithm.

A further modification was introduced resulting in CRS3 (Ref. 67). CRS3 speeds the final convergence of the optimisation by combining a local optimisation algorithm (LOC) with the global search procedure. CRS is based on the geometry of the simplex. It was, therefore, considered appropriate to base the LOC extension on the Needle-Mead simplex procedure.

LOC is carried out in a separate subroutine either at the end of the original CRS2 global search stage or within when certain criteria result. It was suggested to introduce LOC whenever CRS2 generates a new point, which falls within the bottom one tenth of the array of stored values.

To run LOC the best $n + 1$ points of the N points stored in array A are chosen to form a simplex. The worst point, W , is established and the centroid of the remaining n points generated. The points P , Q and R are then calculated, where

$$\begin{aligned} P &= 2G - W \\ Q &= (G + W)/2 \\ R &= 4G - 3W \end{aligned} \tag{3.32}$$

representing normal reflection, contraction and expansion with respect to the centroid respectively. The function and constraint values at point P are established. Consequently, contraction or expansion is carried out to determine the most optimal point to replace W in the set of stored points. Rules governing these steps reflect those of the Nelder-Mead simplex procedure.

CRS portrays a decrease in convergence ability as the region of the global minimum is approached. A recent paper by Ali and Storey (Ref. 4) attempts to remedy this defect. As oppose to carrying out LOC, the region around the current best point is explored using a β -distribution. In the same paper Ali and Storey compared this to a gradient based local search, implemented on detection of a new

best point following application of CRS. The motivation behind this is that gradient type techniques are generally preferable, even when the gradients have to be computed numerically. The paper shows that the new algorithms portray substantial improvement over the original CRS approaches.

Summary of CRS

The CRS is designed for thoroughness of search rather than speed of convergence. The original CRS exhibits slow convergence due to its treatment of each region in a non-preferential manner, thus tending toward a pure random search. The later algorithms aim to rectify this. Due to their inherent random nature, the algorithms are prone to selecting repeated points, thus introducing degeneracy and encouraging susceptibility to premature convergence. In consequence, the modified searches may sometimes show bias toward the region of a local minimum, which is not necessarily the global minimiser.

CRS3 encounters problems when the global minimum lies on a constraint boundary. Due to its simplex framework LOC does not operate effectively at the edge of the search region.

As regards the safety system design model, the fact that the CRS algorithms are based on the Nelder-Mead simplex approach implies the disadvantages associated with the application of the simplex approach will take effect. In particular, the inherent assumption of divisibility when calculating the centroid and the limited ability to carry out contraction and expansion due to the integer stipulation and restricted variable range.

Despite this, CRS has been considered in view of many potential features of merit for application to the safety system design model. The later CRS algorithms, in the main, achieve the balance between a direct and random search. The random influence stretches the search across the region and the direct aspect introduces the history of previous events, reinforcing continued search in promising areas. In addition, generating a population of feasible integer points enables a more optimistic outlook as regards searching the whole search region. Thus, preventing the search getting stuck in local optima. Specifically, the algorithm shows

capability of pattern recognition, resulting in clustering about high quality sub-regions.

Function values only are used. The algorithm is not dependent on the structure of the objective function or presence of derivatives. It is, therefore, applicable to a black-box type function. The algorithm is able to deal with constraints and is particularly designed to incorporate variables with an upper and lower bound, thus defining the boundaries of the search region. In conclusion the algorithm is versatile. It is able to adapt to the problem in hand. The basic concepts of CRS establish the vicinity of the global minimum. A local technique adapted to the problem in question is then used to refine the latter stages of the search.

3.5.3 Partitioned Random Search

A further sophisticated random search approach was introduced by Tang and termed the partitioned random search (PRS) (Ref. 84). The purpose of the PRS is to partition the search region of the objective function into a certain number of sub-regions. (CRS assumes this partitioning indirectly through the formation of clusters, whereas with PRS the sub-regions are directly defined and enforced). Vector points are sampled from each sub-region. The associated function values are used to estimate the 'promise' of each sub-region.

An independent and identically distributed random sampling scheme is employed for each of the sub-regions. The observed function values evaluated within each partition are used to assign a density function to the sub-region in question. The idea is to limit fruitless function evaluations in unpromising sub-regions using a partitioned sequential sampling approach.

An optimal sequential sampling procedure is derived and consists of two components. The first concern is to determine a stopping rule, i.e. a decision as to whether further function evaluations should be taken or whether the algorithm should terminate. The second concern is the sampling rule, i.e. if the algorithm continues, from which sub-region should the next value be taken? The sampling rule is governed by each sub-regions promising index. The promising index is derived from the density function associated with the particular sub-region. The

reader is referred to References 84 and 85 for a more detailed explanation of the mathematical definition of the optimal sequential sampling procedure and estimation of the promising index. The idea behind the method is of interest here. In addition, mathematical analysis in the paper describes that the random search with partitions works better than the random search with no partitions. With 2 sub-regions alone the PRS demonstrates superiority over the crude random search. The more partitions the greater the effect.

As the PRS algorithm proceeds the number of function evaluations per iteration increases. Initially the region is partitioned into k sub-regions. By choosing, for example, one function evaluation from each partition, the most promising sub-region is established. The next step partitions the most promising sub-region into e further partitions. The number of function evaluations required to compare sub-regions is now $k + e$. A modification to the PRS was introduced by Tang and termed the adaptive partitioned random search (APRS) (Ref. 85). In a similar manner, the region is partitioned into k sub-regions and the most promising of these determined. The APRS differs in that, having established the expected best partition, the rest of the sub-regions are aggregated together into one region. The most promising sub-region is then further divided into k parts. The search space, thus, consists of $k + 1$ partitions and following the initial iteration the number of function evaluations remains fixed at $k + 1$.

Summary of PRS

PRS deserves attention in view of its simple yet effective application. The PRS method makes very few assumptions about the objective function, is able to handle constraints and is particularly suitable for variables within bounds. PRS initiates the search from a set of feasible integer points. This fact is indicative of the robustness of the search, thus combating the convergence to local optima. As is the case in CRS. Additionally, PRS is able to maintain integrality of the design points throughout all steps of the algorithm. This is of particular importance regarding the safety system design problem. Addition and subtraction of points is not an issue as in the simplex-based approaches. Consequently, variable bounds are not exceeded and specifically Boolean variables maintain logical values.

As previously stated in the linear programming section, PRS mimics the branch and bound approach in that sub-regions of the search are systematically analysed and evaluated. The branch and bound technique assigns an upper bound to each sub-region. Whereas, PRS is a probabilistic approach and determines the expectation of each partition to contain a high quality point.

A disadvantage of PRS is the intensive demand on function evaluations combined with an unstructured convergence property. APRS goes some way to alleviate the high demand, however the number of function evaluations required to reach a specific criteria is still something of an unknown quantity. In addition, it must be ensured that the amalgamated region in the APRS is not neglected should it contain the global minimum.

3.5.4 The Combinatorial Heuristic Method

The combinatorial heuristic method is an interesting approach regards the integer realm (Ref. 69). The method relies on discretising the range of each independent variable then carrying out a randomised search over the finite grid of possible solutions. The latter part of the algorithm is ideally suited to an optimisation problem involving integer restricted decision variables within a specified range. The steps of the algorithm proceed as follows.

Step 1. The initial point $x^{(0)}$ is randomly generated. $F_{\min} = f(x^{(0)})$ is evaluated.

Step 2. Each variable i , where $i = 1, \dots, n$, is considered in turn and the following loop executed.

- a) Randomly select q values of the i^{th} variable. Hold the other variables fixed. Establish $f(x_{i,j}^{(0)}) = f(x^{(0)} + \Delta x_{i,j})$, where $j = 1, \dots, q$. If $f(x^{(0)}) < f(x_{i,j}^{(0)})$ for all j , reinitiate step 2a with variable $i + 1$.
- b) Identify the lowest of $f(x_{i,j}^{(0)})$. Set this function value to T_{\min} .
- c) Incorporate interaction with the next variable, i.e. carry out a look-ahead search.
 - i) Randomly select p values of the variable $i + 1$. For each $x_{i,j}^{(0)}$ of step 2a evaluate the new function value with the random selection of p , i.e. for $j = 1, \dots, q$, evaluate $f(x_{i,j}^{(0)} + \Delta x_{(i+1),k})$ for k

= 1, ... p. Retain each function value showing an improvement over T_{\min} .

ii) Fix the i^{th} variable at a value corresponding to the point vector resulting in the lowest value in step 2ci.

d) If $i = n - 1$ go to step 3, else return to 2a with $i = i + 1$.

Step 3. Randomly choose q values for the n^{th} variable. Hold the rest fixed and retain the best function value.

Step 4. Check for termination. Either stop or reinitiate step 2 with $i = 1$.

Further enhancements to the method consider various termination criteria, variable reordering, more elaborate look-ahead procedures and variable reduction strategies.

Summary of the Combinatorial Heuristic Method

As regards application to the safety system design optimisation problem, the obvious attribute of the combinatorial heuristic method is the specific orientation toward solving an integer problem. Decision variables are modified within bounds and assume only integer values. In addition, little or no assumption is made concerning the structure of the objective or constraint functions. Constraint evaluations can be easily incorporated at each step and infeasible designs penalised or rejected. The random sampling prevents points getting stuck in local optima whilst simultaneously avoiding an exhaustive search over all discrete values of the design variables.

Again the obvious disadvantage is the high demand on function evaluations. The randomising effect prevents a restricted search of the region yet, in contrast, the random structure introduces an unknown quantity regards convergence of the algorithm.

The combinatorial heuristic method portrays striking similarities to Misra's approximation method in section 3.2.6. The innovative inclusion of random sampling and a look-ahead search could be combined with Misra's use of tolerable slacks about the constraints to introduce a more effective algorithm. Misra's approach would first evaluate the constraint functions defined by an

explicit objective function, thus requiring little computer effort. Should these values fall within the tolerable slacks the more complex implicit function are evaluated. A shift in emphasis in computer effort is, therefore made towards those functions which are well-structured and more easily handled.

CHAPTER 4

GENETIC ALGORITHMS

4.1 Introduction to Evolutionary Computation

In general, any optimisation problem can be perceived as a search for the best point through a space of potential solutions. Classical analytic or exhaustive optimisation techniques usually suffice for small, simple spaces. However, searching a complex space often involves a trade-off between two important issues: exploiting the best solutions and exploring the search space. Exploitation assesses information supplied by a good solution in the hope that incremental changes to the single structure will lead to an improved variant. Hill climbing is an example of such an exploitation strategy, which proceeds to the detriment of exploration of the search space. This short-sightedness makes hill climbing vulnerable to being trapped in local optima, often far removed from the optimal solution. Locally optimum solutions often prove insufficient for real world engineering problems. Conversely, random search is an example of a blind strategy, which explores the search space but ignores information about the problem domain. Such blind sampling is in the main intolerably inefficient.

Evolutionary computation is an umbrella term used to describe a class of search strategies that solve complex problems by simulating evolution via a computer algorithm. Such strategies combine elements of directed and stochastic search, which can strike a remarkable balance between exploitation and exploration of the space.

There are currently three main avenues of research in simulated evolution: genetic algorithms, evolution strategies and evolutionary programming. Common principles governing these are that each algorithm initialises a population of contending trial solutions for the particular problem. Each trial solution is then assessed by an objective measure of performance and used in conjunction with a selection mechanism to determine which solutions continue into subsequent generations. Specific processes are introduced which create new solutions via the alterations of existing solutions. The algorithms differ in the emphasis that they put on particular

aspects of the evolutionary process. Evolution strategies and evolutionary programming emphasise the behavioural link between existing and newly formed solutions (Ref.'s 6,21). In contrast, genetic algorithms stress the genetic link and it is with these that this chapter is concerned.

4.2 Introduction to Genetic Algorithms

John Holland, his colleagues and students developed genetic algorithms (GAs) in the 1970's at the University of Michigan. Holland's work is based on two central themes: the encoding of complicated structures by simple representations (bit strings) and the ability to improve such structures by simple transformations. His motivation rose from the study of natural adaptive systems (Ref. 44).

In evolution the problem each species faces is to search for beneficial adaptations to a complicated and changing environment. The knowledge gained by each species is embodied in the genetic makeup (or chromosomes) of its members. When parents *reproduce* this chromosomal material may be altered. *Crossover* involves the exchange of genetic material between two parents. *Mutation* randomly alters specific areas of the chromosomal string. *Inversion* alters segments of the chromosome. GAs are a class of optimisation procedures which use principles mimicking those of natural selection and genetics, specifically genetic inheritance and Darwinian strife for survival.

The terminology used in GA literature is a mixture of the natural and the artificial. In a biological organism a chromosome is the structure used to encode the description that specifies how the organism is to be constructed. A gene encodes a particular feature of the individual and together the set of genes makes up the chromosome. The location, or locus, of the gene within the chromosome structure determines the order of the characteristics represented by the gene. A gene may adopt several different values to describe its characteristic. Each value of a gene is termed an allele. The complete set of values of all genes constitutes the gene pool. The correspondence of GA terms and optimisation terms is summarised in table 4.1.

Genetic Algorithms	Explanation
Chromosome (string, individual)	Solution (coding)
Gene	Variable contributing to solution
Locus	Position of gene (variable)
Allele	Value of gene (variable)
Gene pool	Entire search space

Table 4.1 An explanation of GA terms

The structure of the GA is such that each solution is coded as a string of parameter values. Each solution being analogous to a chromosome in nature, each parameter a gene, and their specific value an allele. The method then works with a population of strings. Each string is assigned a measure of its fitness in the environment. Selection (or reproduction as it is also known) then exploits this available fitness information. The greater the fitness value the higher the strings chance of being selected to enter the next generation.

The whole process is influenced by the action of genetic operators, typically crossover and mutation. These perturb the parameter information on each string and allow for greater exploration about the search space.

As suggested by Goldberg (Ref. 37), GAs differ from more traditional optimisation techniques in that they work from a coding of the parameter set and not the parameters themselves. They search from a population of points, not a single point. They use objective (i.e. fitness) rather than auxiliary information (e.g. gradients) and the transition rules are probabilistic not deterministic.

In 1975 De Jong combined the theories of Holland with his own computational experiments in a function optimisation setting (Ref. 22). His study bore great importance in the subsequent development of GAs, in particular as an optimisation tool. GAs have since received considerable attention regarding their potential to solve a wide range of practical optimisation problems in the engineering world. Amongst many others, such applications include reliability design, scheduling and sequencing, vehicle routing and transportation.

GAs lend themselves well as a function optimiser due to their simplicity of implementation. They do not have much mathematical requirement. GA's find good solutions by a blind manipulation of their contents, where blind refers to the fact that the GA will search for solutions without regard to the specific inner workings of the problem. The objective function can thus be viewed as a black box that provides only evaluation of the strings. As a result the GA can handle any kind of objective function and requires very little prior knowledge of the fitness domain. In particular, GAs are able to cope with multi-modal regions. From a function optimisation point of view the GA tries to locate and scale high peaks on the fitness landscape. More specifically, selection acts to drive populations uphill towards peaks while recombination operators produce drift.

4.3 The Simple Genetic Algorithm

The SGA is described by Goldberg (Ref. 37) and is used here to illustrate the basic components of the GA. The general structure of the SGA is shown in figure 4.1. Following initialisation and evaluation of a population of strings, operators act to select, recombine and mutate the population, which evolves over subsequent generations. Prior to application of the GA the user must determine a representation scheme, define the fitness measure, define the parameters and variables for controlling the algorithm and designate a performance measure and a criterion for terminating a run. Using this outline, the following section describes the basic stages of the traditional GA in a more detailed fashion.

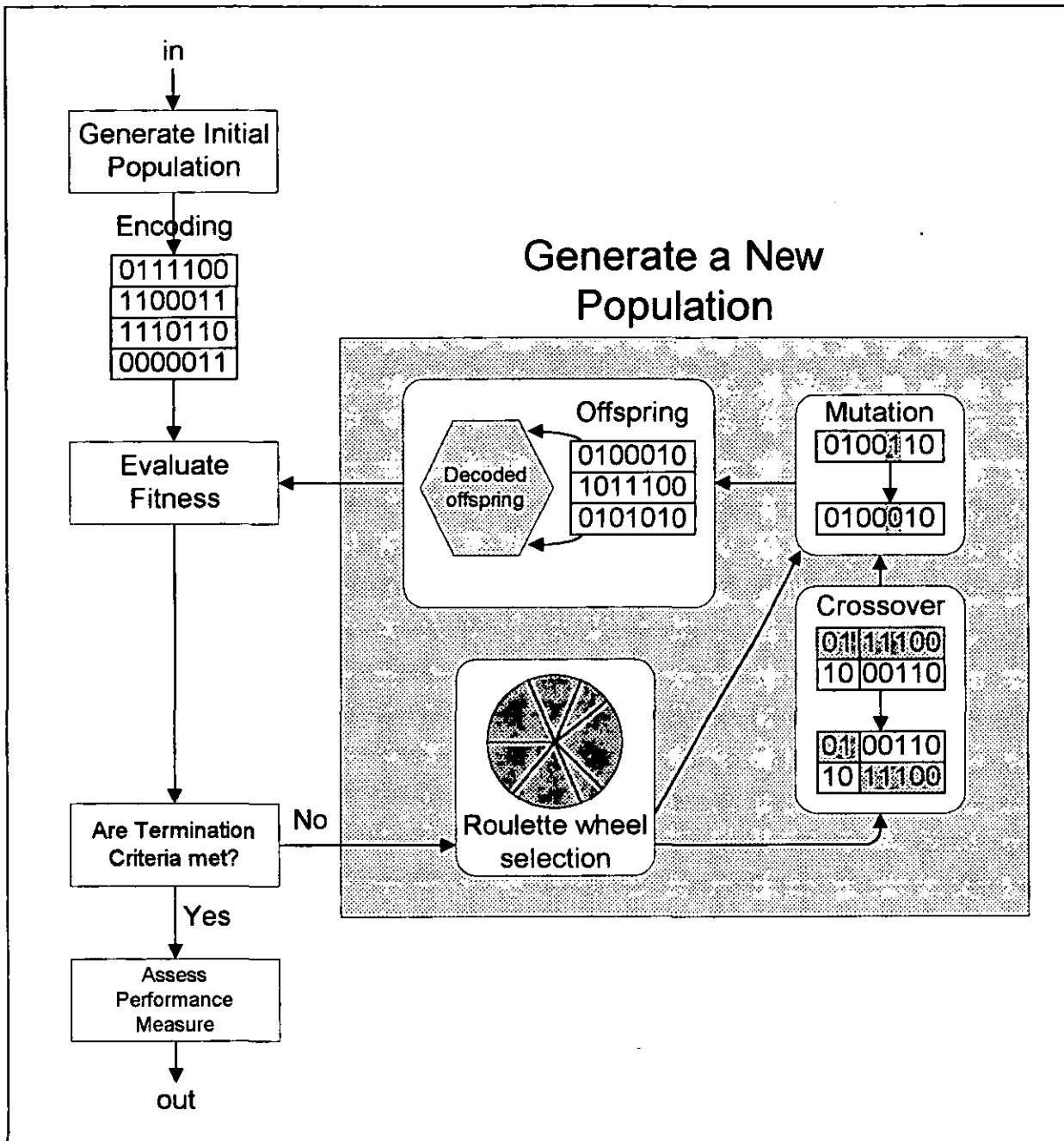


Figure 4.1 The Structure of Holland's Traditional Genetic Algorithm.

4.3.1 Representation

To create an individual the parameters of an optimisation problem are coded as a finite length string over some finite alphabet. Traditionally GAs use either binary or

Gray coded strings. Both of these have a 2-character alphabet, 0 or 1. Considering the binary code: the right most bit of a typical 8-bit byte in the computer has the value 2 to the power 0, i.e. 2^0 . The bit directly to the left is 2^1 , the next 2^2 and so on.

Hence, the binary number

$$10010 = 1 \times 2^4 + 0 \times 2^3 + 0 \times 2^2 + 1 \times 2^1 + 0 \times 2^0 \\ = 16 + 0 + 0 + 2 + 0 = 18$$

Gray code differs from binary in that the hamming distance between successive numbers is minimised. More specifically, adjacent integers have Gray code representations that differ in only one bit position. In essence, a Gray code takes a binary sequence and shuffles it to form a new sequence with this 'adjacency property'. Consequently, for a series of integers, multiple Gray codes exist. One of the most common goes by the name 'binary reflected Gray codes' and is shown in comparison to the standard binary code in table 4.2.

Integer	Binary	Gray
0	000	000
1	001	001
2	010	011
3	011	010
4	100	110
5	101	111
6	110	101
7	111	100

Table 4.2 Binary and Gray Code

The parameter set of each individual is represented by a single chromosomal string of allele values. Each parameter is encoded as its bit string equivalent with its own particular sub-length. Each encoded section is then concatenated to form a single multi-parameter string of 0's and 1's, where each string represents a point in the search space. The ensemble of possible values the parameter set can take defines the entire region and is comparable to a gene pool.

The GA works on the search space of the coded population. Manipulation is carried out on encoded decision variables, not the decision variables themselves, except where a real-valued representation scheme is used. However, examination of the chromosome string in isolation yields no information about the problem being solved. The chromosome must be decoded to apply any meaning to the binary representation. Having decoded the chromosome representation into its decision variable domain it is possible to assess the performance of the individual.

4.3.2 Initialisation

As previously stated, a GA performs a multi-directional search by maintaining a population of potential solutions. The population to population approach attempts to ensure the search does not get trapped in local optima. Initialisation routines vary. For research purposes, a random population is useful since the evolution from a disordered population to a well-structured highly fit set of points is a good test of the GA. To initialise a binary population of n individuals, whose chromosomes are l bits long, $n \times l$ values from the set $\{0,1\}$ are randomly generated. If prior knowledge of the problem is available it may be useful to incorporate it into the initial population. However, this should be tempered with diversity to ensure regions of the search space are not neglected.

4.3.3 Fitness evaluation

A criteria is required to determine how 'good' an individual is. An objective function is generally used to provide a measure of how individuals perform in the problem domain. The objective function value is a raw measure of performance that is usually not in an appropriate form for use in the algorithm. For this reason it is only used as an intermediate stage in determining the performance of individuals. For example, when finding a global minimum the fittest individuals will have the lowest numerical values of the associated objective function. Another function is, therefore, required to transform the objective function value into a measure suitable for application of the selection operator. This mapping is generally referred to as the fitness function and

determines the fitness of the individual. Transformation of the raw objective function value is essentially the first stage of selection. The mechanics of the transformation process depends on the problem being solved and the selection operator used.

4.3.4 Selection

Selection is the first operator concerned with generating the next population. The purpose of selection is to increase the probability of reproducing individuals that have higher fitness values, thus directing search towards promising regions in the search space. The particular selection method used determines how many and which strings make up the new population. In some versions the whole population of strings is replaced, in others only a subset. If the whole population, n say, is to be replaced the new chromosomes are generated by choosing strings from the original population and reproducing them. It should be noted that selection alone does not introduce any new points for consideration from the search space. Selection copies individuals without change into the next generation. The traditional approach implements selection using Holland's proportionate method, or a biased roulette wheel as it is better known. Basically each individual in the population is allocated a slice of the roulette wheel that is directly proportional to the individuals fitness. Specifically, individual i with fitness f_i has a selection probability p_i of being reproduced where

$$p_i = \frac{f_i}{\sum_{j=1}^n f_j} \quad (4.1)$$

A wheel is made according to each string's probability and the selection process is based on spinning the wheel n times. Figure 4.2 illustrates the application of the biased roulette wheel to a population of 4 strings. Strings with a higher fitness occupy a greater percentage of the wheel and thus, have a higher probability of contributing one or more offspring to the next generation.

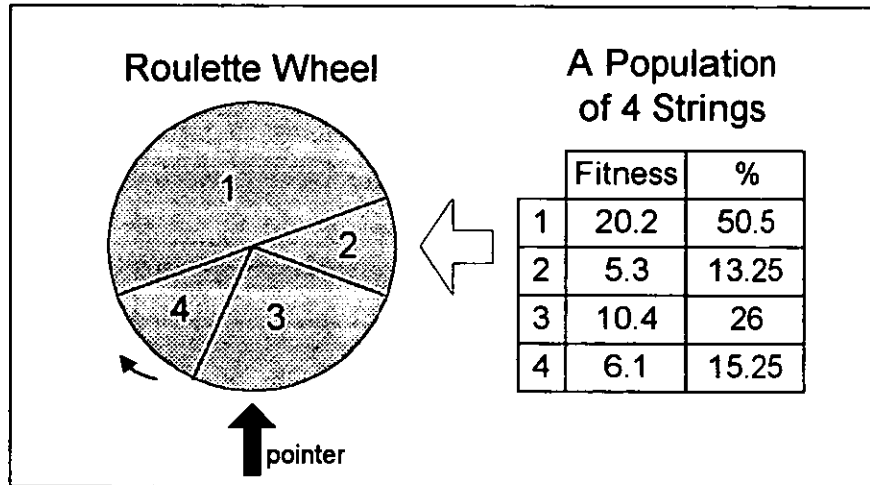


Figure 4.2 A Biased Roulette Wheel

4.3.5 Genetic Operators

Individuals already selected to enter the next generation are recombined according to specified recombination rates. The genetic operation of crossover is the exchange of genetic material between two parent chromosomes and allows new individuals (i.e. new points in the search space) to be created and tested. The traditional GA operator is called one-point crossover. A cutting point is randomly chosen such that the genetic material beyond that point is exchanged between two previously selected parents to create two children. One-point crossover with the random crossover point chosen to be 3 gives

Parent 1: 011|010 \Rightarrow Child 1: 011100
 Parent 2: 101|100 \Rightarrow Child 2: 101010

In addition to its role as a recombination operator, binary crossover also has the potential to generate new parameter values. Consider a chromosome made up of a concatenated set of parameter values in binary form. Each time the selected crossover point is within the boundaries of a parameter, part of the parameter's bits from one parent are combined with those of another with the result that a new parameter value may be encoded. Crossover is often described as the key to the

power of the GA. The performance of the GA depends to a great extent on the performance of the crossover operator used.

Mutation is generally the random alteration of a parameter value on the solution string. As regards a binary coded string this simply means changing a 1 to a 0 or vice versa. A slightly different approach randomly regenerates the bit in question. The new bit will, therefore, differ from the old one 50% of the time. In essence, this is merely equivalent to a reduction in the mutation rate. Given that mutation is generally applied uniformly to an entire population of strings it is possible that a given binary string may be mutated at more than one point. Mutation ensures all alleles have the potential to enter the population.

Reproduction and crossover effectively explore the search space. Occasionally, however, they may become overzealous and lose some potentially useful genetic material. Mutation is usually conceived to be a background operator, which helps maintain sufficient diversity in the population.

4.3.6 Result Designation and Termination Criteria

It is required that a termination criterion is specified for each run of the GA. Due to the fact that the GA is a stochastic search method it is difficult to formally specify convergence criteria. Application of convergence termination criteria is problematic as fitness of a population may remain static for a number of populations before a superior individual is found (Ref. 29)

A popular criterion is to state the maximum number of generations evolved. An alternative approach is to achieve a particular degree of convergence portrayed in the binary encoding of the string population. There is no general rule. The method of termination is problem dependent. In addition, there must be a means to monitor the behaviour of the GA, i.e. a method of result designation. One frequently used method is to designate and store the best individual obtained in any generation of the population during the run (i.e. the best-so-far). Keeping track of the best individual from one generation to the next is usually described as an off-line measure. The

average fitness of each population may also be evaluated and stored to give an on-line indication of population convergence.

A major advantage of the GA is that it can be stopped and restarted at any point in the search and information to that point can be determined and analysed. In addition, the GA can provide a number of potential solutions to a given problem, where the choice of final solution is left to the user. This is particularly useful in the case of multiobjective optimisation for which the problem does not necessarily have a unique solution. The GA is able to explore the trade-offs between multiple objectives with a minimum of effort and identify alternative solutions simultaneously (Ref. 57).

4.3.7 GA Parameters

Prior to each run important global program variables that affect the operation of the GA need to be defined. These are referred to as the GA parameters. The most common GA parameters govern the population size, the maximum number of generations, the probability of crossover and the probability of mutation. The greater the population size and number of generations the more potential there is to both explore the search space and introduce diversity from the onset. However, this is generally at the expense of efficiency and a balance for the particular problem must be attained.

The crossover rate controls the expected number of chromosomes to undergo the crossover operation per generation. A higher crossover rate allows exploration of more of the solution space and reduces the chances of settling for a false optimum: but if this rate is too high, it results in the wastage of a lot of computation time in exploring unpromising regions of the solution space.

The mutation rate controls the rate at which each bit is changed to its opposing value. If it is too low, many genes that would have been useful are never tried out: but if it is too high, there will be much random perturbation, the offspring will start losing their resemblance to the parents and the algorithm will lose the ability to learn from the history of the search (Ref. 34).

Establishing the best values for the GA parameter set is an optimisation problem in itself. This area is discussed further in chapter 6.

4.4 The Theoretical Foundation

The theoretical foundation behind GA's relies on the notion of a schema (Ref.'s 37,56). Holland defines a schema to be a similarity template, which describes a subset of strings with similarities in certain string positions. Geometrically, a schema can be considered as a hyperplane in the search space. Holland used the concept of a schema with binary encoding to derive the schema theorem. In essence, the schema theorem states that if a better understanding of the problem domain is desired it is essential to use information gained from the similarities between strings as well as their fitness. To build a schema the notational device * (or don't care symbol) is introduced forming a ternary alphabet (0,1,*). A schema represents a subset of strings that match in all positions other than '*'. For example the schema (**1*010) matches the following strings (1111010), (0111010) and (1010010) but does not match (1101010).

Schemata are described using two properties: order and defining length. The order of a schema is the number of fixed positions, i.e. 0's and 1's, in its vector. The defining length is the distance between the first and last fixed positions of the schema.

Each binary string of length l is an instantiation of 2^l possible schemata. Holland speculated that the evaluation of each string actually offers partial information about the expected fitness of all possible schemata in which that string resides. Such parallel processing is considered a combinatorial explosion working to positive effect and is termed implicit parallelism.

Intuition implies that highly fit strings will contain a high proportion of fit schemata, where a schema's fitness is defined to be the average fitness of all strings in the population matched by the schema in question. A schema's fitness is altered as the GA progresses. The effect exerted by selection is straightforward. Strings are

selected from the population with a probability relative to their fitness. Above average schema will, therefore, receive an increasing number of strings in the next generation. Conversely, the proportions of less desirable schema will decrease. The effect of crossover and mutation is a little more complex. As regards crossover, if the crossover point chosen cuts the schema, the pattern of the schema will be disrupted. Obviously, the shorter the defining length the more likely the schema is to survive crossover. Mutation is most likely to destroy a schema containing more fixed bits. Hence, short, low order, above average schemata receive exponentially increasing trials in subsequent generations of the GA.

4.5 The Constrained Case

Most practical optimisation methods must have the ability to handle both equality and inequality constraints. As regards the constrained problem, if the objective fitness value of a decoded chromosome lies outside the feasible region the chromosome is deemed to be infeasible. Equality constraints can be dealt with inherently within the black box objective function. The concern is, therefore, to cope with inequality constraints.

The simplest approach is to run the GA in the normal fashion and evaluate both the objective fitness and the constraints. Should any constraints be violated the solution is infeasible. The string is, therefore, assigned no fitness. It is, in effect, discarded from the evolutionary process. This is often classified as the rejecting strategy. Difficulties arise with the rejecting strategy when the problem is highly constrained or the search space non-convex. In a highly constrained environment, finding a feasible point is a feat in itself. If infeasible points are rejected outright, no information concerning the space is gained and much time and effort wasted. In addition, simple rejection could seriously limit the diversity of the initial population. As regards a non-convex feasible space, it is often more efficient to reach the optimum if it is possible to 'cross' an infeasible region. Constraint management techniques allowing movement through feasible regions of the search space tend to yield more rapid optimisation and produce better trial solutions than do limiting search trajectories only to feasible regions of the search space (Ref. 35)

Additionally, in the constrained case the optimum typically occurs at the boundary of the feasible space. The ability to approach the optimum from both feasible and infeasible regions is, therefore, beneficial.

A penalising strategy enables consideration of infeasible points. As in conventional techniques, the penalty method penalises infeasible solutions, thus modifying the constrained case into an unconstrained problem. The problem is transformed such that in the limit its solution tends to the optimal solution of the original problem. The challenge is to strike a balance between eliminating unpromising information and retaining useful schemata.

There are very few general guidelines to assist in the derivation of penalty formulae. The penalty term can modify the value of the objective function in two ways. One approach is to add the respective penalty to the fitness function, that is

$$\text{mod}(x) = f(x) + p(x) \quad (4.2)$$

where x is the chromosome, $f(x)$ the objective function and $p(x)$ the penalty term. If no constraints are violated $p(x) = 0$, else $p(x) > 0$ for a minimisation problem or $p(x) < 0$ to find a maximum.

A penalty can be classed as being constant or variable, where the variable type tends to be more effective for complex problems. Variable penalties are based either on the extent to which constraints are violated or the iteration number of the GA. Typically, the degree of constraint violation depends on the absolute distance of the particular infeasible solution from the feasible region. The penalty becomes more severe as the absolute distance increases. As an example, Homaifer, Qi and Lai (Ref. 45) applied an additive variable penalty to a typical non-linear programming where $p(x) = 0$ if x is feasible, else

$$p(x) = \sum_{i=1}^m r_i g_i^2(x) \quad (4.3)$$

where r is the penalty coefficient for the i th constraint. For a greater array of penalty function methods the reader is referred to reference 34.

A further constraint consideration is a characteristic termed illegality of chromosomes. An illegal chromosome has a parameter set that does not represent a solution to the given problem.

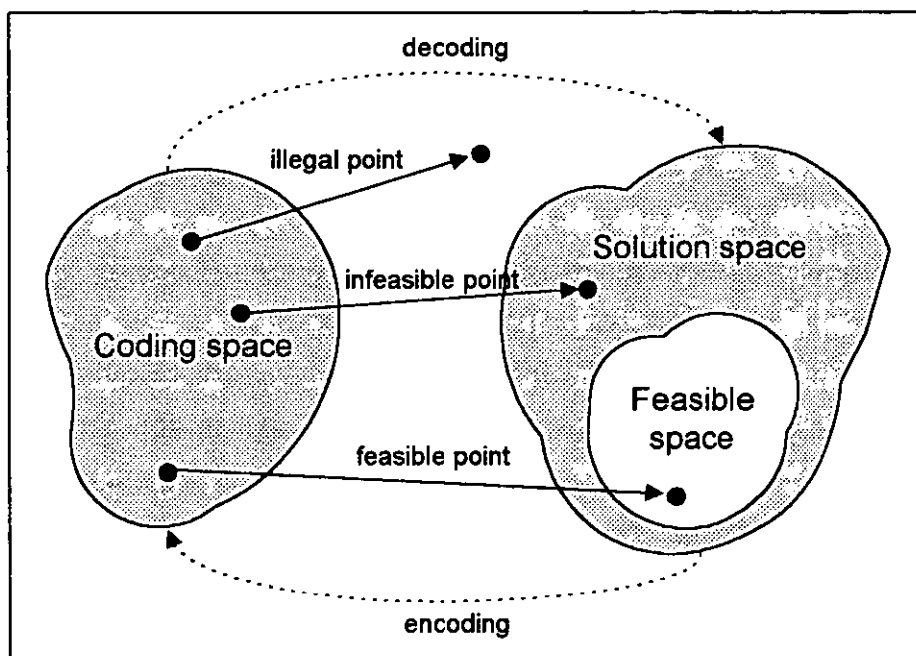


Figure 4.4 Infeasible and Illegal Solutions

The occurrence of illegal chromosomes is directly related to the manner in which the parameter set is encoded. In the main, illegal chromosomes arise due to the action of genetic operators, typically crossover and mutation. An illegal chromosome cannot be decoded to a feasible solution and, thus, cannot be evaluated (see figure 4.4). As such the penalty approach is inapplicable in this situation.

There are three common approaches to deal with illegality: the rejecting strategy, the repairing strategy and the modification of genetic operators. The rejecting strategy proceeds as stated previously. The repairing strategy applies some repairing procedure, which generates a feasible chromosome from an infeasible one. The repaired chromosome can be used solely for the purpose of fitness evaluation, or it can actually replace the original illegal chromosome in the population. Empirical

testing shows the repairing strategy to be both effective in terms of speed and performance. The final approach is to modify the action of the genetic operators such that the feasibility of chromosomes is maintained. This method has also proved effective although it is highly problem dependent (Ref. 21).

4.6 Premature Convergence

Above average strings are favoured by the selection operator of the GA. As the population evolves over successive generations, fit strings become more abundant and below average strings are 'weeded out'. Ultimately a pattern of particularly fit schemata emerges, enforcing population convergence. All too often, however, highly fit strings dominate the population too early and bring about what is known as premature convergence.

While Holland's schemata theory points to sampling rates and search behaviour in the limit, any implementation necessarily uses a finite population. Estimates based on finite samples inevitably have a sampling error associated with them. Finite sampling in conjunction with the high-variance associated with stochastic selection can result in offspring production that is markedly different from theoretical predictions. De Jong cited both aspects as primary causes in the loss of diversity in the population gene pool. As a result the search converges to a sub-optimal solution, i.e. demonstrates premature convergence. In the biological arena, this is termed genetic drift, where allele frequencies vary due to chance over successive generations and may result in some alleles becoming 'extinct'. In such situations, recombination is less likely to produce individuals in certain regions than in others. In particular, standard crossover simply regenerates the current point.

Many innovative techniques have been proposed to combat premature convergence. De Jong's study in 1975 is one of the earliest to address this issue and in fact, set the ball rolling in terms of alternative approaches to selection and recombination operators. These techniques are considered in more detail in the next section.

A paper by Booker considers a preventative approach in that he looks for indicators that occur before loss of diversity takes effect and carefully implements search operators to avoid the situation (Ref. 9).

4.7 Modifications to the GA

Recent research has made a departure from the SGA with the aim to improve its performance as an optimisation tool. Attention has been focussed on each stage of the algorithm: specifically, representation, selection, fitness evaluation and the recombination operators. This section looks at each of these areas in a little more detail.

4.7.1 Representation

The classical GA generally uses a fixed length binary string representation. The motivation behind this is related to the idea of schemata. Both binary and non-binary strings code the same number of solution alternatives. Holland speculated, however, that it is beneficial to maximise the number of schemata being sampled, thus maximising implicit parallelism. This is achieved using the lowest cardinality alphabet, i.e. binary. In addition, bit string representations fit with the standard versions of crossover and mutation.

There are, however, problems with a binary representation. Frequently the cardinality of a parameter set is not a power of 2. In such cases, enforcing a binary representation can increase the size of the representation space. Consequently, genetic operators will invariably produce some illegal chromosomes, thus instigating the need for coping strategies. A further problem arises when GAs find points in the vicinity of the optimum but fail to reach the best. This may be explained by the notion of Hamming cliffs. A Hamming cliff corresponds to a pair of numbers in integer space whose bit representations are complementary, for example 7 and 8 are adjacent with bit string representations 0111, 1000. A small change in the parameter value, therefore, requires a radical change in its corresponding binary representation.

In such circumstances the GA is said to ‘Lack the Killer Instinct’. The adjacency property of Gray coded strings may nullify this problem.

4.7.2 Fitness Evaluation

The GA uses the numerical fitness values obtained from the objective function in association with the parameter set to guide the search. As such, it is vital that an individual’s fitness, i.e. converted raw objective data, within the finite population, accurately describes the search space. Problems can arise at the start of the GA if the population contains a few highly fit individuals amidst a set of mediocre colleagues. The fitness advantage gained by the fit strings, relative to the rest of the population, is misleading and in consequence they dominate the population within a few generations, resulting in premature convergence. In later generations the opposite problem may occur. There is a tendency that a population showing diversity, which is legitimately dominated by fit strings, has a population average fitness close to the population best fitness. Consequently the expected number of offspring for the average and best strings are almost equivalent. As a result each string contributes equally to the subsequent population despite differences in fitness. A possible means of mitigating both problems is to use fitness scaling (or normalising procedures as they are also known). The use of scaling retains an individual’s relative performance and also attempts to bias the selective pressure towards better individuals although still allowing relatively unfit individuals the potential to reproduce (Ref. 16).

Fitness scaling maps raw objective function values to some positive real values, which are then used to determine the selection probability for each chromosome. If the raw fitness is represented by f and the scaled fitness by f' for chromosome k

$$f'_k = g(f_k) \tag{4.4}$$

where g is the scaling function. The scaling function can take a variety of different forms. The most common are linear scaling, sigma truncation, power law scaling and logarithmic scaling. Linear transformations offset the objective function and are often used prior to fitness assignment to ensure the resulting fitness values are non-

negative. If there is little deviation in the population, then linear scaling provides only a small bias towards the fittest individuals. In contrast, power law scaling is used to shrink or stretch the range of fitness values required, the degree of effect being dependent on the value of the power. The performance of the GA is highly sensitive to the normalising procedure used. The reader is referred to reference (Ref. 34) for further details regarding these scaling mechanisms.

4.7.3 Selection

Holland's traditional GA selects chromosomes to enter the next population using a biased roulette wheel. Selected strings are probabilistically recombined and inserted into the new population. If recombination does not occur, offspring identical to their parents are inserted directly into the new population. In biological terms this is asexual reproduction. Selection is generally based on the whole population, which maintains a constant size from one generation to the next. As illustrated in figure 4.5.

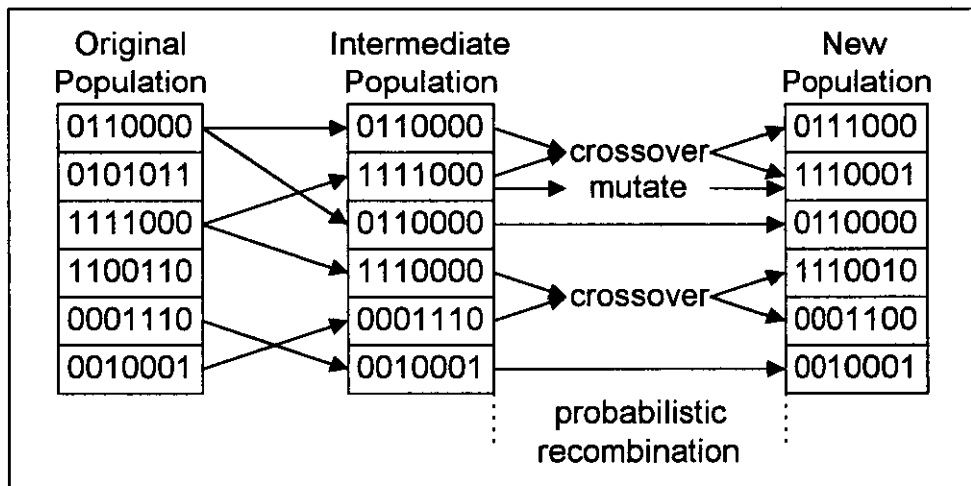


Figure 4.5 Constant Sampling Space

An alternative approach creates an enlarged sampling space. Chromosomes are selected and stochastically recombined in the original manner. The method differs in that the resulting offspring are not inserted immediately into the new population. An enlarged intermediate sampling space containing both selected parents and offspring is formed from which the new population is then chosen. As illustrated in figure 4.6.

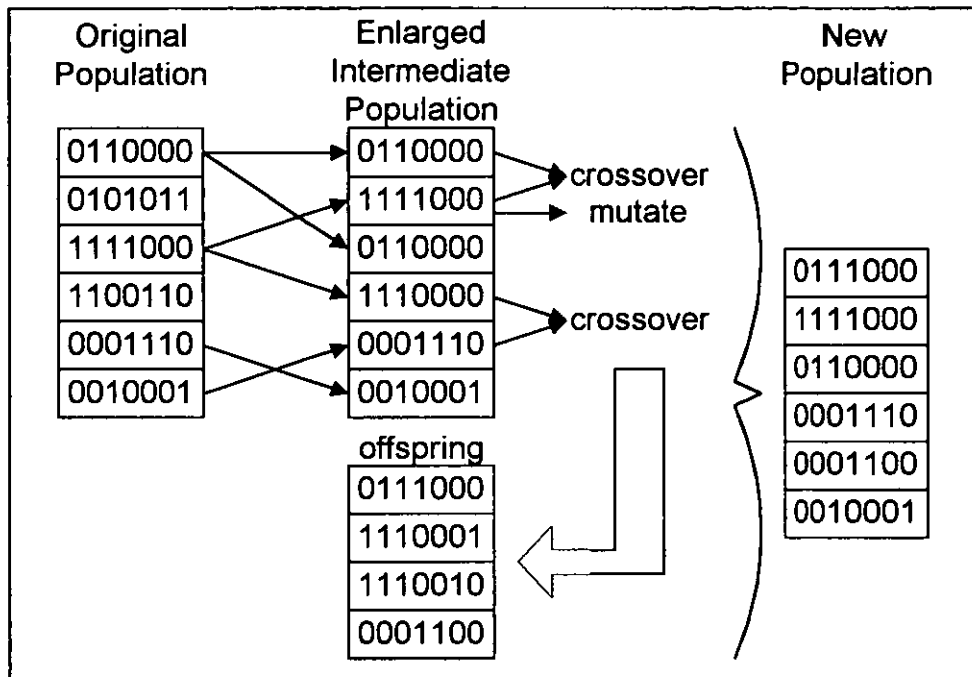


Figure 4.6 Enlarged Sampling Space

Selection pressure is a critical factor in the balance between promoting diversity and controlling rapid convergence. Many studies have proposed, examined and compared various selection methods. A study carried out by De Jong in 1975 is of particular importance. This study considers a test bed of five functions covering a broad range of characteristics and examines the effect of four different selective pressures on GA performance (Ref. 22).

Holland's biased roulette wheel is an example of stochastic sampling. The number of offspring attributed to a particular individual is approximately directly proportional to the individual's performance. As cited by De Jong, the roulette wheel is, however, open to much variance resulting in a fair amount of scatter between the expected and actual number of offspring. Consequently, there is no constraint either on the upper or lower bound of an individual's offspring production. This allows the potential for super component dominance or, conversely, denial of many inferior individuals. Baker introduced a modification to Holland's roulette wheel called stochastic universal sampling (Ref. 7). Markers are equally spaced around the edge of the roulette wheel, the number being in accordance with the population size. One spin of

the wheel then matches each marker with a segment allocated to a particular individual. The premise is to prevent super component dominance and maintain diversity.

A deterministic approach to compute the expected number of offspring for each string uses the rank of its fitness in the population as oppose to the magnitude. There are a variety of means to assign the offspring based on ranking but it must be ensured that a monotonically increasing relationship with respect to increasing performance values is maintained and the population size retained. Ranking gives a certain degree of control over percentage offspring production of the population, thus restricting reproduction of highly fit strings. Problems with objective functions having large offsets are also eliminated. However, convergence to better solutions using the ranking method is generally slow. A further disadvantage is that the ranking method does not allow allele proportions to change when warranted. The algorithm's ability to exploit promising schemata is compromised by ignoring all knowledge about the relative fitness of strings in the population (Ref. 9).

Besides ranking, a variety of deterministic approaches using the concept of expected number have been proposed. As an example Brindle's approach (Ref. 34) allocates offspring according to the integer part of each chromosome's expected offspring production. The fractional remainder of this value is used to create a list in descending order. Individuals are then added to the new population from the sorted list until the sample space is full.

Elitist selection is another deterministic approach. It ensures the best structure is passed into the next generation, if it is not chosen through the usual means. De Jong suggests that an elitist strategy improves local search at the expense of global perspective.

A method, which contains both random and deterministic sampling, is tournament selection (Ref. 34). This involves the random choice of a set of chromosomes from the current population. The best one of these is then chosen for reproduction. The smallest set containing only two chromosomes is referred to as a binary tournament. Tournament selection can be incorporated within the biased roulette wheel approach.

Successive pairs of individuals are chosen using the roulette wheel in the normal manner. The best of these is then inserted into the new population.

4.7.4 Recombination Operators

The recombination operators provide the key to exploration of the search space and as such are of vital importance to the GA process.

Crossover is often characterised as a distinguishing feature of the GA. It capitalises on familiar areas of the space that show promise whilst simultaneously stepping out to unexplored regions. The conventional approach proposed by Holland, termed one-point crossover, reinforces this statement using Holland's building block hypothesis. An obvious extension of one-point crossover is multi-point crossover. If the chromosome is thought of as a circle with the first gene immediately following the last then it becomes immediately clear that there are in fact 2 crossover points: one fixed at position zero and the other randomly selected. An immediate generalisation to the present crossover operator is to allow both crossover points to be randomly selected (Ref. 23).

In this way two-point crossover is formed. This was the philosophy behind the fifth modification introduced in De Jongs' study, which he termed a generalised crossover approach. Figure 4.7 illustrates the effect of four-point crossover. By Holland's analysis, multi-point crossover becomes a random shuffle breaking up higher order schemata. Later studies have shown an improvement in off-line performance to the detriment of on-line convergence (Ref. 9).

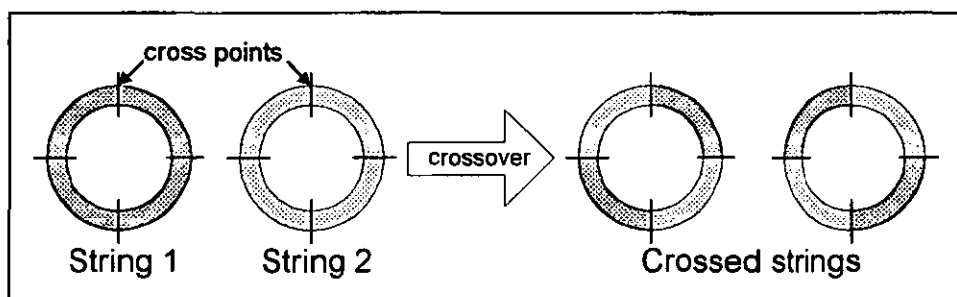


Figure 4.7 4-Point Crossover

Uniform crossover is an approach given by Syswerda (Ref. 80), which swaps a bit from one parent to the other with equal probability. Uniform crossover involves the generation of a random crossover mask, which is made up of a binary string of identical length to the chromosome. Allele values are exchanged between parents according to the pattern of the mask. This procedure is demonstrated in figure 4.8. Theory suggests that one-point crossover will perform better as it maintains blocks of good code. In practice varying results have been obtained. Syswerda shows uniform crossover to be effective when used with real-coded strings in a combinatorial problem setting. When used with real-valued alleles uniform crossover is usually referred to as discrete recombination.

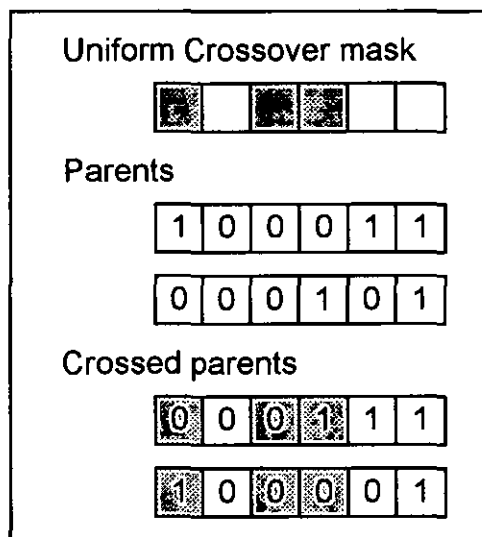


Figure 4.8 Uniform Crossover

One-point crossover is likely to disrupt sections of code that lie at opposite ends of the chromosome. Highly fit building blocks that span a significant portion of the string are, thus, unlikely to be exploited. This motivated Holland to develop the inversion operator. Inversion involves the random selection and reversal of a segment of the chromosome. To recap, a chromosome is made up of a set of genes, i.e. the problem variables, which lie at specific loci along the string. Each gene is given a specific value, i.e. an allele, within the range of the variable it represents. Inversion considers the gene and its allele as an ordered pair and dissociates the gene from its loci. In other words, the segment reversal switches the position of the gene on the string but does not alter its value. For example, consider a specific

chromosomal string, where the numbers above correspond to the particular gene that each allele describes and the two inversion sites are randomly chosen to be 2 and 6

$$\begin{array}{cccccc|cc} 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 \\ 1 & 0 & 0 & 1 & 1 & 1 & 0 & 1 \end{array}$$

The inversion operator gives

$$\begin{array}{cccccc|cc} 1 & 2 & 6 & 5 & 4 & 3 & 7 & 8 \\ 1 & 0 & 1 & 1 & 1 & 0 & 0 & 1 \end{array}$$

As can be seen, each gene retains its original value. The fitness of the string is, therefore, unaltered. Although inversion has no effect on the genotypical information its affect becomes apparent when used in conjunction with crossover. It may be that the current population has a bad ordering. Parts of the chromosome representation that contribute most to the performance of a particular individual may not necessarily be contained in adjacent sub-strings. As a result these fit building blocks are constantly disrupted. Inversion, in its role as a reordering operator, can rearrange the gene placement, providing the opportunity for more effective propagation of the reordered building block.

Following inversion, crossover must fix the position of the genes on each parents' chromosomal string and cross only the allele information. If crossover proceeds with the gene and allele as an ordered pair it cannot be guaranteed that either offspring will contain a full gene complement. An alternative approach is to cross only homologous strings, i.e. those with identical genotypical structure, but this is a little restrictive.

4.7.5 Hybridisation

In practical design applications every numerical optimisation technique is characterised by advantages and drawbacks when viewed in comparison to one another. The GA robustly explores the search space. However, a common conception is that GAs are good at global optimisation but bad at fine detailed local

searching. Hybridisation has the potential to maintain and exploit favourable features of the GA whilst simultaneously enhancing their ability to refine good solutions. Hybridisation is the combination of the evolutionary procedure with an optimisation technique of different nature, possibly suited to the domain of the chosen application, or characterised by a complementary behaviour (Ref. 68).

The simplest means to apply a hybrid method involves optimising in two steps. First, the GA is used to locate the area of the suspected global optimum and then subsequently switched to other methods for further tuning. An improvement is to incorporate the new optimisation technique among the set of operators as the algorithm evolves, without necessarily changing the structure of the GA.

A recent paper by Yang and Douglas (Ref. 93) combines the Nelder and Mead simplex downhill method with a traditional GA. The simplex downhill method (introduced in section 3.4.1) is not as robust as the GA but is well tuned for local searching. The idea of the simplex approach is to speed up search to the optimum once promising areas are established. For this reason the expansion operator is discarded and an averaging operator added.

An iteration of the GA creates an intermediate population of size n via selection, recombination and mutation in the normal manner. A second intermediate population is achieved by applying the simplex approach n times on a randomly selected sub-committee from the initial intermediate population. A new coding system, which normalises the variable within the range $(0,1)$ replaces the traditional bit string code. This enables application of the individuals to both methods.

Both intermediate populations are ranked in descending order of fitness. Individuals of like position are compared and the best inserted into the new population. A form of elitism. The result is an effective, efficient, robust and fast converging global optimum procedure.

A further study (Ref. 68) couples a GA to a gradient based optimisation technique. Again an intermediate population is created via selection, crossover and mutation; some of its included individuals are then selected and fed into a hill-climbing operator

for refinement. Three alternative schemes are proposed to choose these individuals. In this study the traditional binary string is used to encode a solution. Prior to application of the hill-climbing operator, the binary string is converted into the real number list, then converted to continue the GA. The number of individuals selected for local refinement is limited due to considerable computer overhead of gradient evaluation. In addition, to keep computer effort to a practical level, the hill-climbing operator generally proceeds for no more than 2 iterations. In conclusion, the hybrid GA shows performance enhancement for both single and multiobjective problems.

CHAPTER 5

SYSTEM ANALYSIS

5.1 Introduction to HIPS System

Safety systems are designed to operate when certain conditions occur and act to prevent their development into a hazardous situation. Failure of a safety system for a potentially hazardous industrial system or process may have catastrophic consequences, possibly injuring members of the workforce or public and occasionally resulting in loss of life. The purpose of this thesis is to describe a design optimisation scheme, which uses available resources to the best possible advantage to obtain an optimal safety system design.

5.1.1 Description of HIPS System

To demonstrate the practicalities of the optimisation process it has been applied to a simple High-Integrity Protection System (HIPS). The basic features of the HIPS are shown in figure 5.1. Its function is to prevent a high-pressure surge passing through the system. In this way protection is provided for processing equipment whose pressure rating could be exceeded. The high pressure originates from a production well of a not normally manned offshore platform and the pieces of equipment to be protected are located downstream on the processing platform.

The first level of protection is to be the ESD (emergency shutdown) sub-system. Pressure in the pipeline is monitored using pressure transmitters (PTs). When the pipeline pressure exceeds the permitted value then the ESD system acts to close the Wing and Master valves on the well together with any ESD valves that have been fitted.

To provide an additional level of protection a second level of redundancy can be incorporated by inclusion of a HIPS (high-integrity protection system). This works in a similar manner to the ESD system but is completely independent in operation.

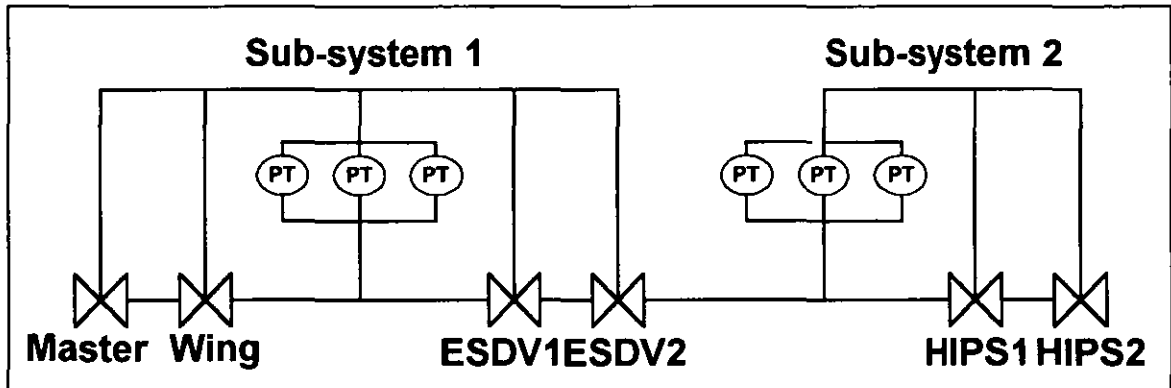


Figure 5.1 High-Integrity Protection System

Even with a relatively simple system such as this there are a large number of options for the designer to consider. In the example it is required to determine values for the design variables that represent the following:

Designer Options	Design Variable
• How many ESD valves are required (0, 1, 2)?	E
• How many HIPS valves are required (0, 1, 2)?	H
• How many pressure transmitters for each sub-system (0, 1, 2, 3, 4)?	N_1, N_2
• How many transmitters are required to trip?	K_1, K_2
• Which of two possible ESD/HIPS valves to select?	V
• Which of two possible pressure transmitters to select?	P
• Maintenance test interval for each sub-system (1 week – 2 years)?	θ_1, θ_2

In all there are 42,831,360 combinations of the twelve design variables. An exhaustive evaluation of each potential design is impractical. In addition, understanding the interaction between design variables is a complex task. It is impossible for the design engineer to predict the effect that changes in each design parameter will have on system performance.

For this example, each considered hardware component can fail in one of two modes: dormant and spurious failure. Dormant failure is the inability of the component to carry out its desired task on demand. Spurious failure results from the component carrying out its desired function when its operation is not required. In consequence, normal working conditions on the processing platform are inappropriately disrupted. The failure rate and mean repair time for each component in both failure modes is shown in table 5.1.

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

Table 5.1 Component Failure Data

The choice of system design is not, however, unrestricted. Limitations have been placed on the design such that:

1. The total system cost must be less than 1000 units. Hardware costs for each item in the system are given in table 5.1.
2. The average time each year that the system resides in the down state due to preventive maintenance must be less than 130 hours. Times taken to service each component at each maintenance test are also shown in table 5.1.
3. The number of times that a spurious system shutdown occurs would be unacceptable if it was more than once per year.

5.2 Safety System Analysis

A criteria must be determined to quantify how 'good' each system design actually is. The most important feature of a safety system is that it works when the demand arises. The objective is, therefore, to minimise system unavailability (i.e. the probability of system failure on demand) and as such provides a measure of system performance.

Consideration must also be given, however, to the available resources. The HIPS is limited by cost, maintenance effort and spurious trip frequency. The design options need to be adjusted in order to improve system performance without violating the constraints. This will involve a trade-off between the effects any design changes will have. For example consider a gas detection system. In the presence of a gas leak, the detectors fitted initiate action of the protective system. Failure to do so results in failure of the system. The more gas detectors fitted the more robust the system. However, Gas detectors can fail in two modes. Increasing the number of detectors increases the likelihood of registering a spurious leak. This in turn spuriously activates the protection system, production is disrupted and excess cost incurred.

In order to assess each design option a means to evaluate each constraint is required.

5.2.1 Evaluate the System Unavailability

No explicit objective function can be formulated. Incorporating an added level of redundancy requires a new term to be added to the objective function, therefore altering its characteristics entirely. As such, fault trees are used to quantify the system unavailability of each potential design. It is, however, a time consuming, impractical task to construct a fault tree for each design variation. Such an approach could be viable if automatic fault tree construction methods were available. Although much attention has been given to this area of fault tree analysis, a reliable and effective approach has not been achieved.

To resolve this difficulty house events can be used to enable the construction of a single fault tree capable of representing causes of the system failure mode for each possible system design. House events in the fault tree, which are either TRUE or FALSE, are utilised to switch on or off different branches to model the changes in the causes of failure for each design alternative.

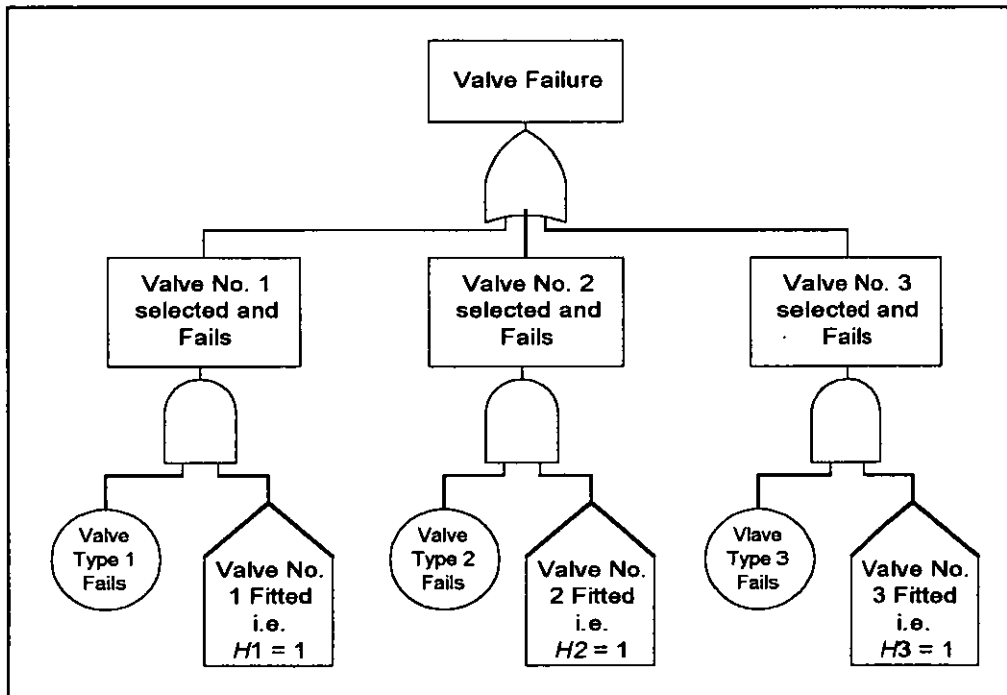


Figure 5.2 Fault Tree Structure for Component Select

Consider, for example, the choice of a valve type, V1, V2 or V3. The structure of the part of the tree that deals with valve failure is as shown in figure 5.2. If valve type 1 is selected the house event, H1, corresponding to the selection of this valve is set to TRUE. House events H2 and H3, corresponding to the selection of valves 2 and 3 are conversely set to FALSE. A contribution to the top event arises from the left most branch only. The two right most branches are in effect switched off. Levels of redundancy are handled in a similar manner.

The particular house events utilised in construction of a single fault tree representing the probability of system unavailability for each HIPS design alternative are discussed in detail in the next section.

5.2.1.1 Construction of the System Unavailability Fault Tree

The top event of the fault tree representing the causes of system unavailability is defined as 'Safety system fails to protect'. The top event will occur if all the valves along the pipeline fail to close.

Each valve is of the 'air to open', fail safe type. When pressure in the pipeline is at an acceptable level, computer logic maintains the solenoid of the valve in an energised state, the pneumatic line remains pressurised and the associated actuator retains the valve in the open state. An increase in pipeline pressure is detected by pressure transmitters, which transmit a signal to the computer. If the pressure increase exceeds that which is acceptable, the computer causes the circuit of the output channel to the solenoid to open. Two relay contacts are available to break the circuit and, thus, introduce a level of redundancy. Consequently, the solenoid is de-energised and a vent valve activated. This results in a drop of pressure to the actuator, which causes the valve to close.

The immediate, necessary and sufficient sub-events to the top event related by AND logic are 'Wing and master valves fail to protect', 'ESD valves fail to protect' and 'HIPS valves fail to protect', as shown in figure 5.3. The system unavailability fault tree construction is described via development of each branch in turn.

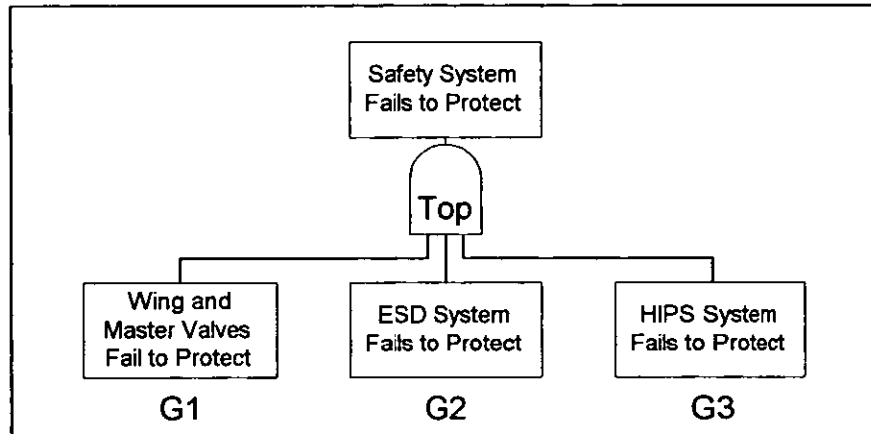


Figure 5.3. Top Event of System Unavailability Fault Tree

Wing and Master Valves Fail to Protect

Certain components in subsystem 1 are fixed and as such constitute part of each potential design variation. Specifically, these are the wing and master valves, their solenoids, the relay contacts and the computer logic.

Closure of the wing or master valve alone will protect against a high-pressure surge in the pipeline. The event 'Wing and master valve fails to protect' will, therefore, only occur if both the wing and master fail. Consider first the wing valve. The wing valve fails to protect if either the wing valve itself fails (basic event 'WV') or the pneumatic line to the actuator of the wing valve remains pressurised (an intermediate event that must be developed further). The pneumatic line remains pressurised due to failure of the solenoid valve associated with the wing valve (basic event 'SVW') or as a result of the solenoid staying energised. A similar tree structure develops from the alternative branch 'Master valve fails to protect'. This causal relationship described above is shown in figure 5.4.

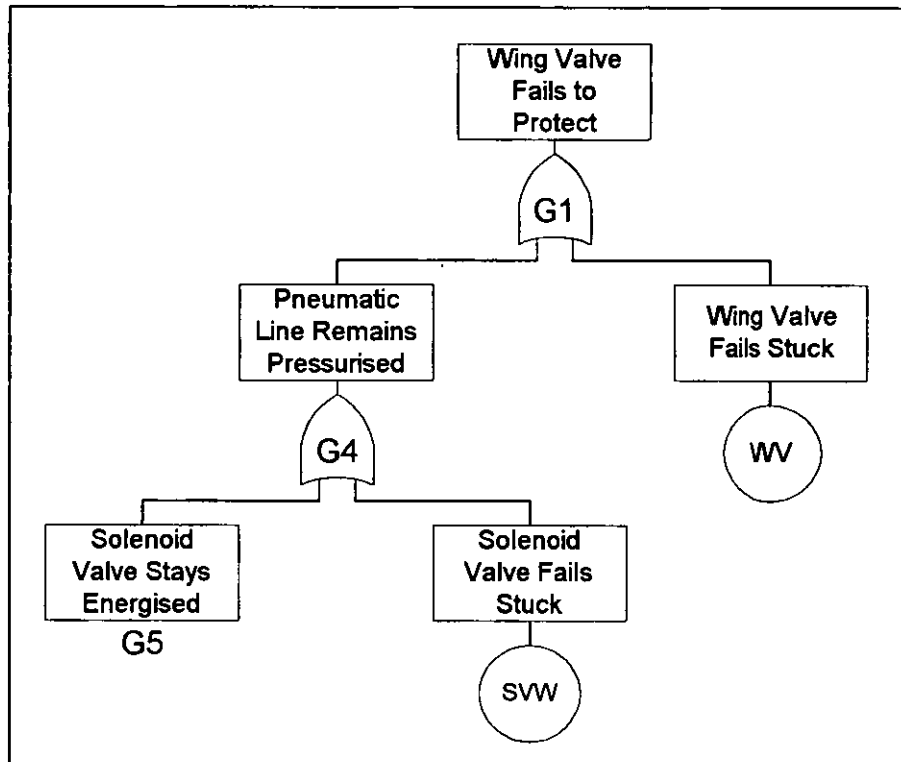


Figure 5.4 Wing Valve Fails to Protect

The event 'Solenoid stays energised' is common to branches developed from both the wing and master valves. This event represents a failure in the flow of fault logic from detection of increased gas pressure by the pressure transmitters to de-energising of the solenoids of the wing and master valves. This intermediate event must be further developed. The solenoid stays energised if both relay contacts fail to break the circuit from the computer (basic events 'R 1/1' and 'R1/2') or if the computer fails to send the trip. In turn, the 'computer fails to send the trip' if the computer logic itself fails (basic event 'PLC 1') or if the computer does not receive input to signify an increase in gas pressure, i.e. if the trip signal to subsystem 1 is not received. The causal relationship from gate 5 to the 'failure of subsystem 1 to indicate trip' is shown in figure 5.5.

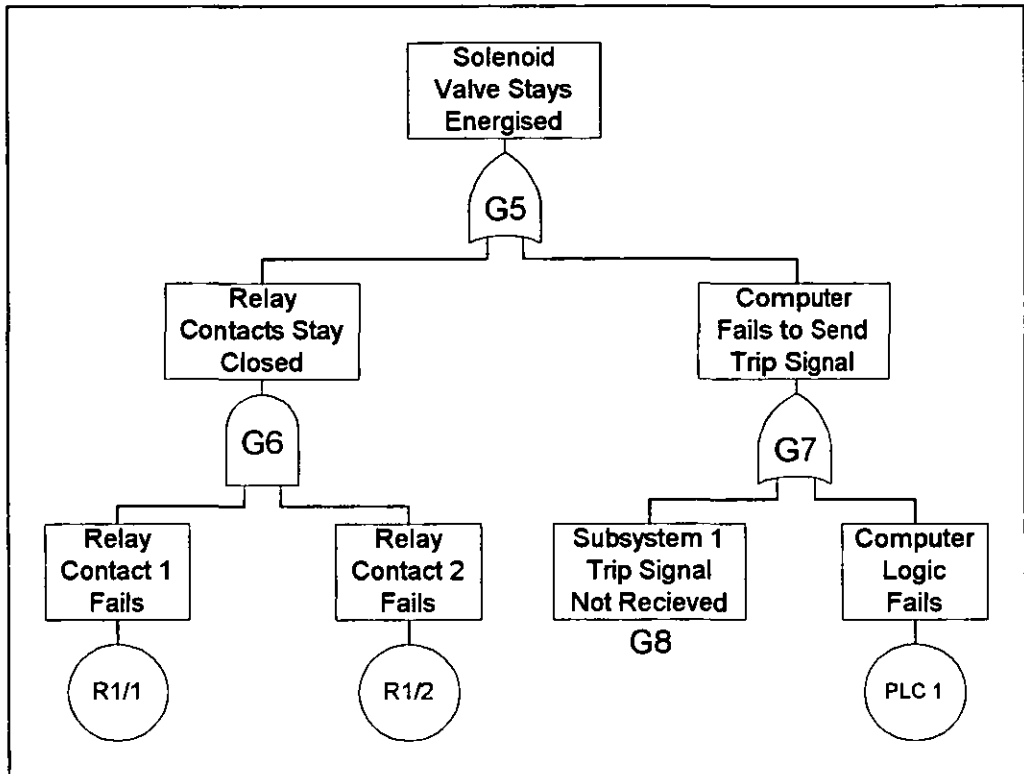


Figure 5.5 Solenoid Valve Stays Energised

The pressure transmitters initiate the safety protection system into action should the demand arise. Failure of the pressure transmitters of subsystem 1 to detect a gas pressure beyond the acceptable limit prevents action of the computer logic associated with subsystem 1 and hence, any subsequent events.

The number of pressure transmitters for subsystem 1 (N_1) is a variable with range 1 to 4. The structure of the tree will alter according to the value of this variable. To model these structural changes the following house events are introduced:

- NEN1: the number of pressure transmitters for subsystem 1 is not 1
- NEN2: the number of pressure transmitters for subsystem 1 is not 2
- NEN3: the number of pressure transmitters for subsystem 1 is not 3
- NEN4: the number of pressure transmitters for subsystem 1 is not 4
- EN1: subsystem 1 has 1 pressure transmitter fitted
- EN2: subsystem 1 has 2 pressure transmitters fitted
- EN3: subsystem 1 has 3 pressure transmitters fitted

EN4: subsystem 1 has 4 pressure transmitters fitted

If the condition holds, the house event is set to true, i.e. assigned a probability of 1. For example, if the design is such that subsystem 1 has 3 pressure transmitters fitted, house events NEN1, NEN2, NEN4 and EN3 would be assigned a probability of 1 and the rest set to 0.

The casual relationship between sub-events and house events developed from 'subsystem 1 fails to indicate trip' is shown in figure 5.6.

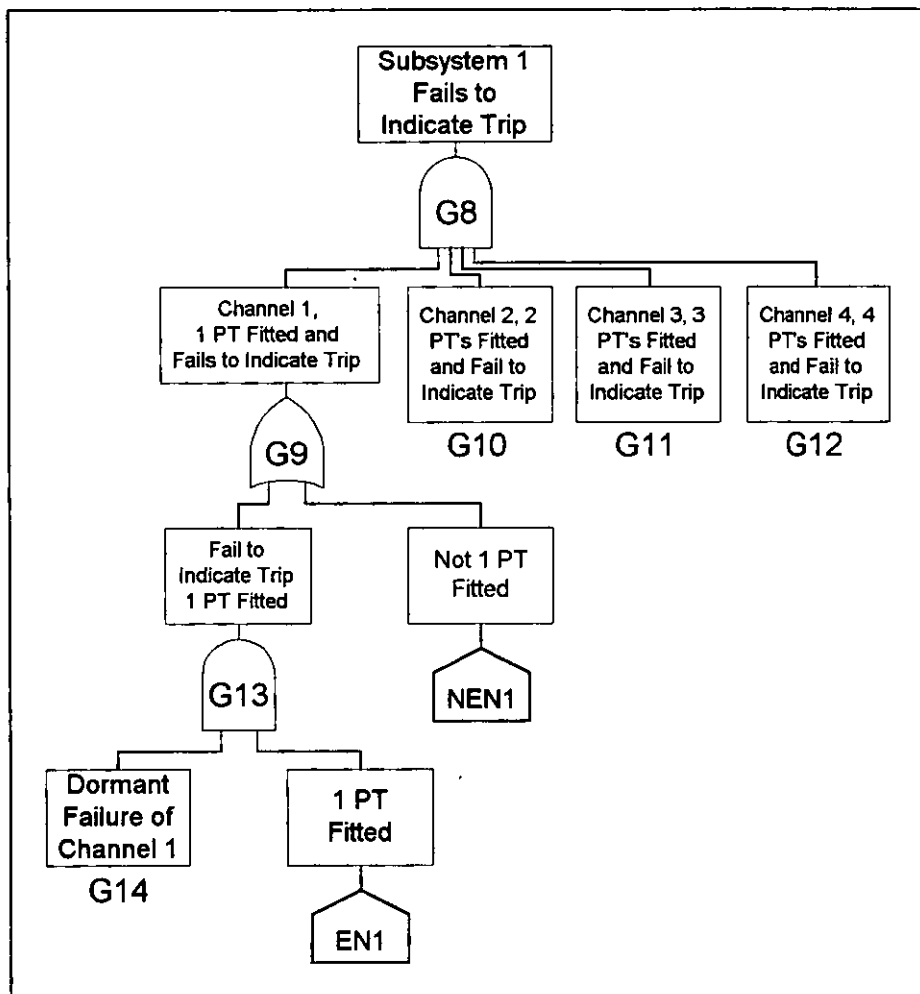


Figure 5.6 Subsystem 1 Fails to Indicate Trip

Each sub-event represents a channel failure, where channel 1 models a design with 1 pressure transmitter alone for subsystem 1, channel 2 models a design exactly with 2

pressure transmitters, channel 3 with 3 pressure transmitters and so on. Channel failure to indicate a trip can occur in two ways: either the channel does not model the design in question or the channel fails to indicate a trip due to failure of the pressure transmitters fitted. In consequence, for each analysis three channels will automatically fail because they do not model the design in question. The higher event will only occur if all four channels fail. The significant channel is, therefore, that which models the design.

Consider the example previously cited. A design has 3 pressure transmitters and the house events are fixed accordingly. Channels 1, 2 and 4 input fault logic to 'System fails to indicate trip' due to the house events NEN1, NEN2 and NEN4 respectively. The house events achieve their desired purpose in that they eliminate irrelevant branches that do not model the considered design. Control is effectively given to channel 3. If channel 3 inputs fault logic the output event will occur. As a result of the assigned house event probabilities, channel 3 will only fail to indicate a trip if the 3 pressure transmitters fitted fail to carry out their required task.

The number of pressure transmitters in subsystem 1 is not the only variable. Consideration must be given to the number of pressure transmitters required to trip the system (K_1 , with range 1 to N_1) and the pressure transmitter type (P_1 or P_2). Each possible variation must be modeled in each channel.

To model further structural changes in the fault tree additional house events are defined:

- EK1: 1 pressure transmitter is required to trip subsystem 1
- EK2: 2 pressure transmitters are required to trip subsystem 1
- EK3: 3 pressure transmitters are required to trip subsystem 1
- EK4: 4 pressure transmitters are required to trip subsystem 1
- P11: Pressure transmitter type 1 is fitted
- P12: Pressure transmitter type 2 is fitted

Channel 1 models a design with 1 pressure transmitter for subsystem 1. Hence, K_1 must be 1. Consequently EK1 holds true and is set to 1. Channel 1 is further developed as shown in figure 5.7.

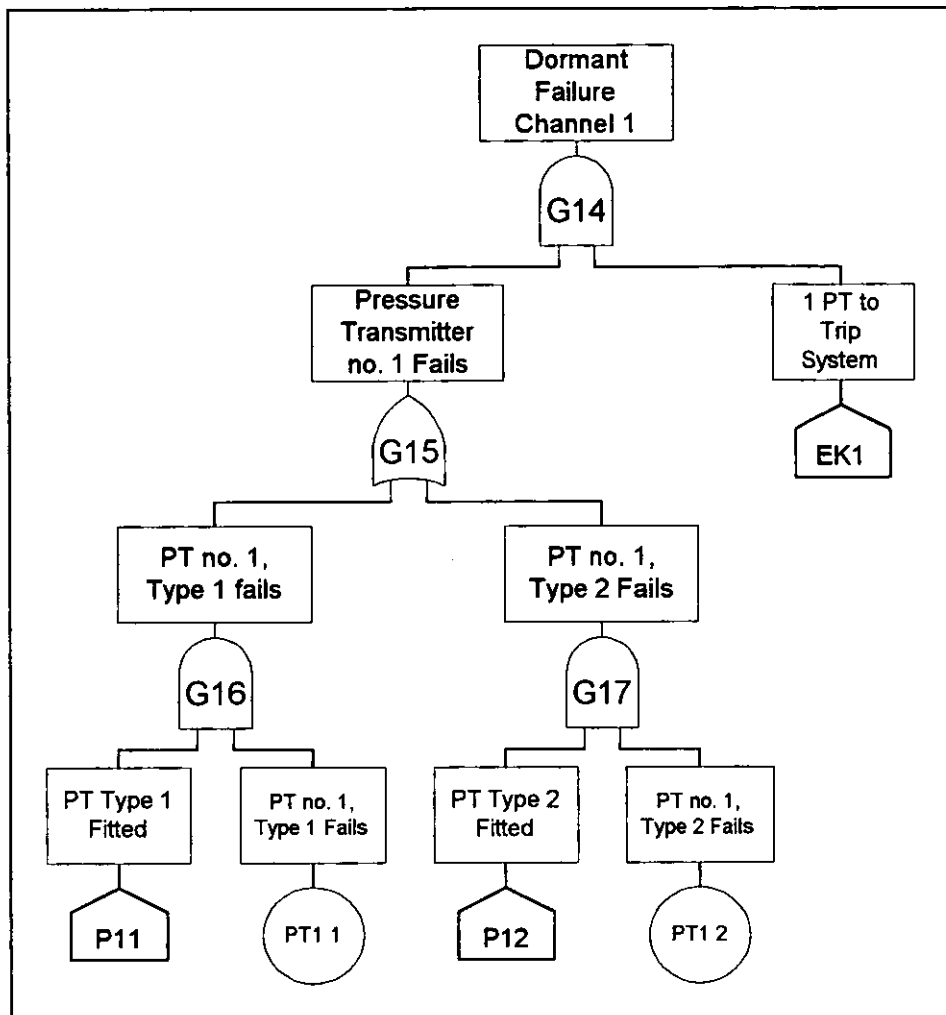


Figure 5.7. Dormant Failure of Channel 1

Channel 1 terminates with house events modelling a component type design alternative. This structural relationship is common throughout the tree to model the inclusion of different valve and pressure transmitter types. Specifically, pressure transmitter number 1 fails if either ‘Pressure transmitter type 1 is fitted’ (house event ‘P11’) AND ‘Pressure transmitter number 1 type 1 fails’ (basic event ‘PT1 1’) OR if ‘Pressure transmitter type 2 is fitted’ (house event ‘P12’) AND ‘Pressure transmitter

number 1 type 2 fails' (basic event 'PT1 2'). It is assumed that, when more than one pressure transmitter is fitted, they are of the same type.

Channel 2 models system designs with 2 pressure transmitters for subsystem 1. In this case, K_I can be either 1 or 2. If only one pressure transmitter is required to trip the system, they act as redundant components and both must fail to trigger channel 2 failure. Conversely, if 2 pressure transmitters must register an increase in pressure, failure of either will be sufficient to fail the channel. Hence, 'Dormant failure of channel 2' occurs as a result of 2 from 2 pressure transmitters failing or 1 from 2 transmitters failing, where $K_I = 1$ and $K_I = 2$ respectively. Additional basic events 'PT2 1' and 'PT2 2' representing failure of pressure transmitter number 2 type 1 or 2 are introduced. Figure 5.8 shows further development of channel 2.

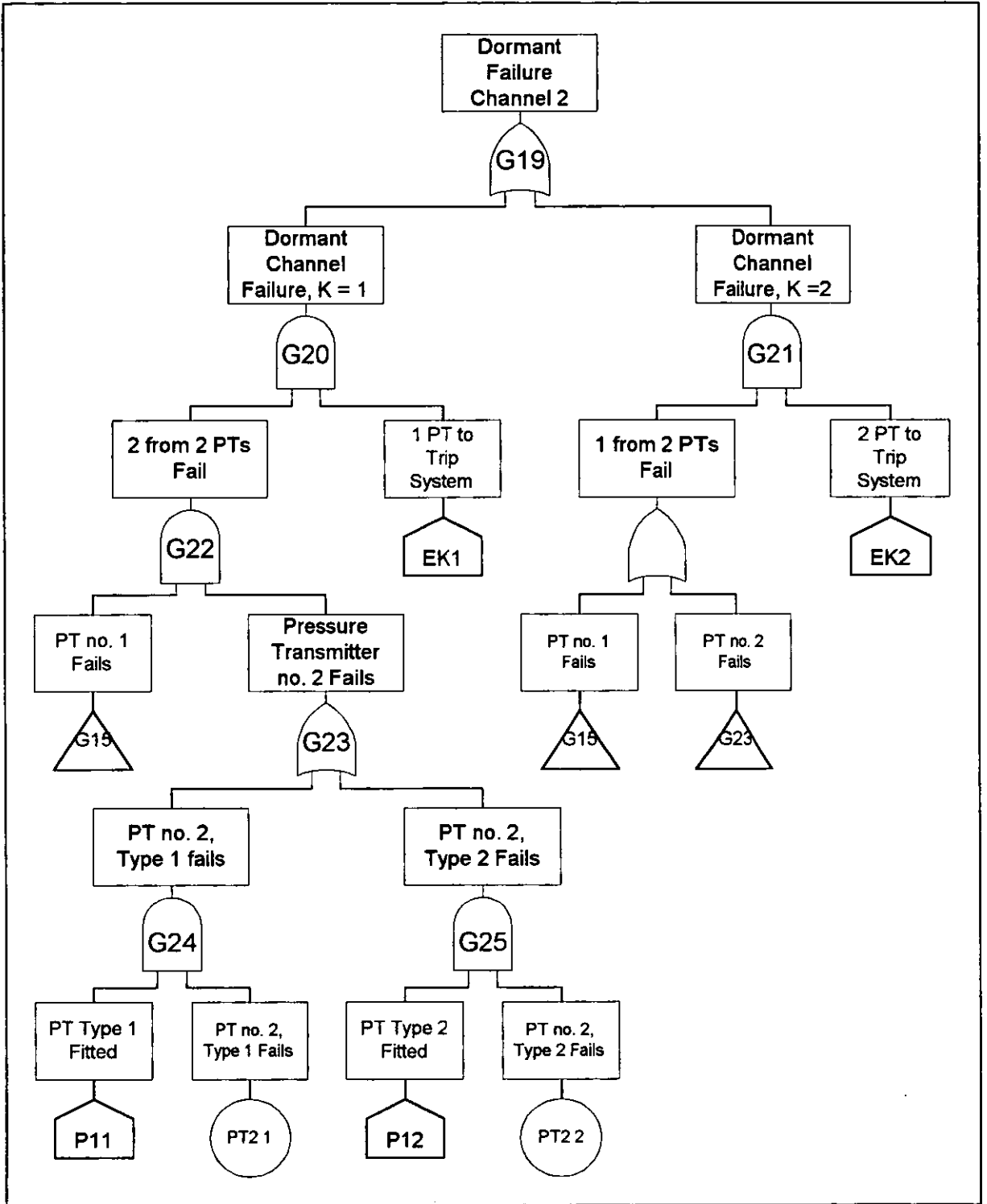


Figure 5.8. Dormant Failure of Channel 2

In a similar manner, channel 3 must model K_1 equal to 1, 2 or 3. Either 3 from 3, any combination of 2 from 3 or failure of 1 pressure transmitter alone will trigger dormant failure of channel 3, where $K_1 = 1$, $K_1 = 2$ or $K_1 = 3$ respectively. Additional basic events 'PT3 1' and 'PT3 2' representing failure of pressure transmitter number 3 type 1 or 2 are introduced. Channel 4 must account for 1 through 4 pressure transmitters being able to trip the system. This section of the tree is developed in a similar manner to that already described.

ESD Subsystem Fails to Protect

Consideration is now given to the event heading the second branch in figure 5.3. The event 'ESD valves fail to protect' is developed in a very similar manner to failure of the wing and master valve. However, structural differences arise due to the fact that ESD valves are not fixed aspects of the system design. The number of ESD valves (E) is a variable ranging from 0 to 2 and as such house events are included to switch on and off relevant parts of the tree.

The ESD subsystem fails to protect if ESD valve 1 and ESD valve 2 fail to protect. Failure of each ESD valve can occur if either the ESD valve is not fitted or the ESD valve is fitted and fails. Figure 5.9 shows the causal relationship represented with house events.

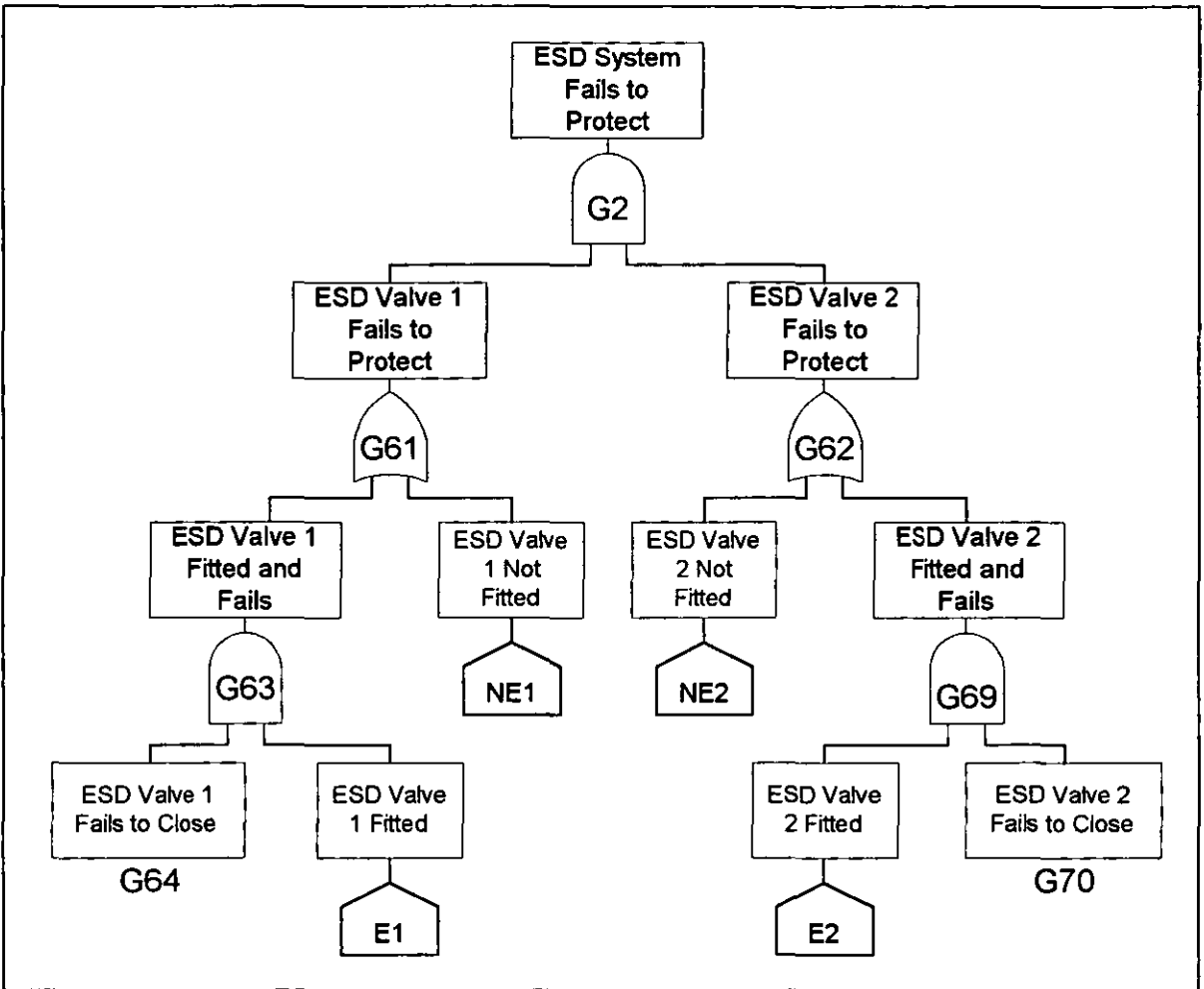


Figure 5.9 ESD System Fails to Protect

Failure of the ESD valve to close occurs as a result of the ESD valve itself failing stuck or the pneumatic line remaining pressurised. Failure of the ESD valve depends on the type fitted (V_1 or V_2). The pneumatic line remains pressurised due to failure of the solenoid of the ESD valve or the solenoid staying energised.

The passage of fault logic from detection of hazardous gas levels to de-energising the solenoid of the ESD valve utilises the computer logic, relay contacts and any pressure transmitters fitted. These basic events are common to those, which transmit fault logic and ultimately trigger the wing and master valve to close. The event 'Solenoid stays energised' is, therefore, identical to that described in the previous section and its branching structure is merely replicated within the ESD structure of the tree. Figure 5.10 shows the fault tree structure representing the failure of ESD valve 1 to close.

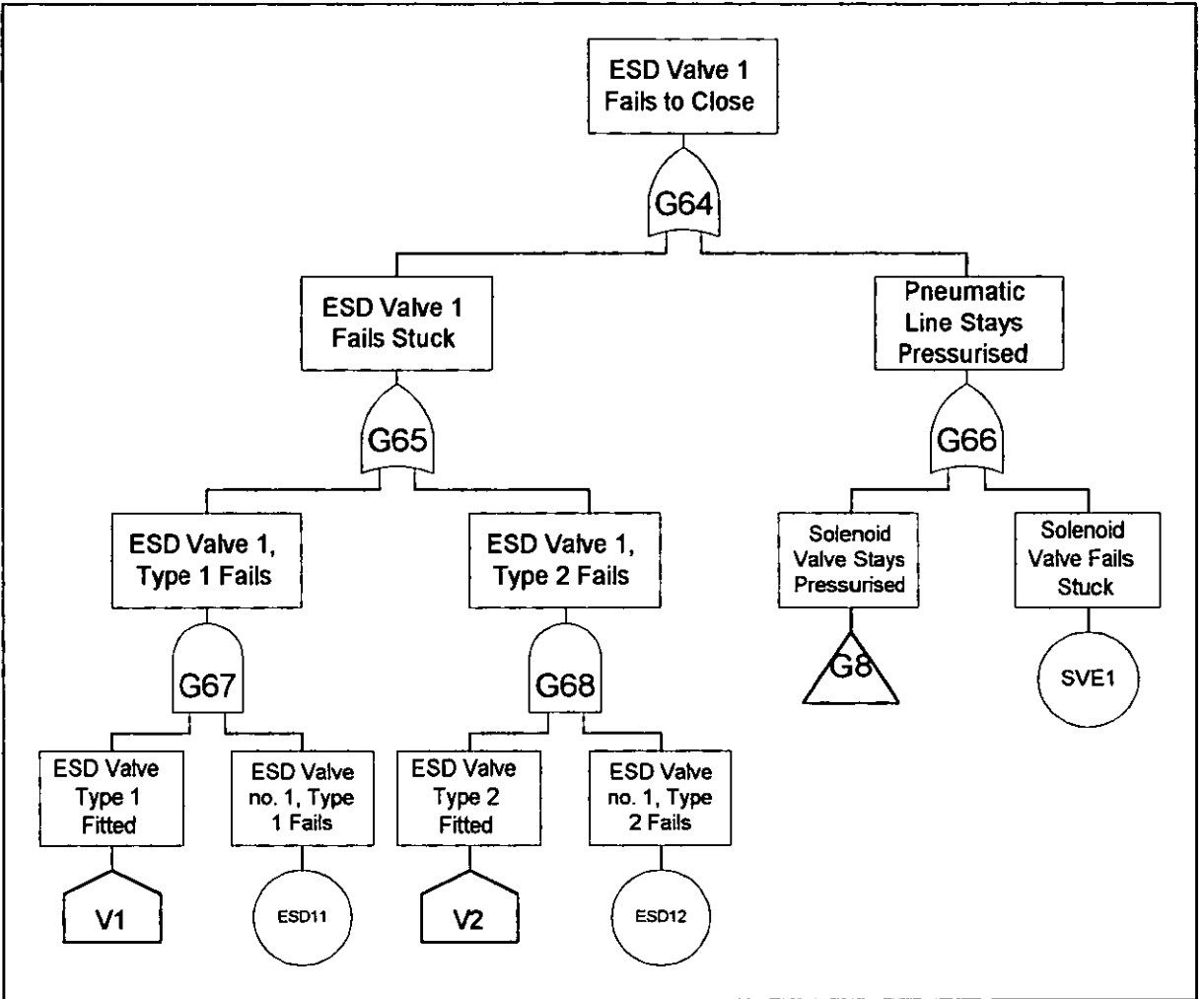


Figure 5.10 ESD Valve 1 Fails to Protect

HIPS Fails to Protect

The HIPS subsystem works in an identical manner to that of the ESD subsystem. As such the HIPS subsystem fails if all fitted HIPS valves fail. The HIPS subsystem is, however, completely independent in operation to the ESD, wing or master valves.

Failure logic for the HIPS system is developed in an identical manner to that described for the ESD subsystem to complete the fault tree development.

5.2.1.2 Data Input and Analysis

Fault tree construction and analysis is carried out in the software package FAILTTREE+ (described in section 2.5). In total the system unavailability fault tree consists of 88 primary events and 169 gates. Of the 88 events, 44 are basic events and 44 house events. The system under consideration is a safety system, which works only when the demand arises. As such, each basic event is a dormant failure model type, i.e. each requires input of the associated failure rate (λ), mean time to repair (τ) and inspection interval (θ).

Failure and repair data are given in table 5.1. This data is fixed. It does not vary from one design variation to the next. In contrast, the value of a component's inspection interval varies. A component is associated with either subsystem 1 or subsystem 2 and as such, variables θ_1 and θ_2 respectively supply the inspection interval for the particular design. Evaluation of component unavailability (q) is given by equation (2.47) and hence the value of q for each basic event varies from one design to the next, due to variations in θ .

Data for each house event is specified by the house event model type. This model type requires either the value 1 or 0. The value of the house event is specific to the design in question and hence, must be specified prior to each particular design quantification.

The system unavailability fault tree is termed 'Unavail'. On selection of the analysis option two ascii files, 'Unavail.ats' and 'Unavail.aqd', are created

5.2.1.3 Conversion of the Fault Tree to a BDD

The use of house events overcomes the need to construct a separate fault tree for every possible design alternative. To specify the tree structure to a particular design, house events are assigned the appropriate Boolean states. Although preferable to reconstruction of the tree, manipulating house events is an involved process. As tree

size increases the task can become very time consuming and prone to human error. In addition, the ability to incorporate the fault tree technique within an optimisation procedure is restricted due to dependence on the fault tree analysis software package. Since the source code is not available it requires each analysis to be performed manually and the results entered into the optimisation scheme. Due to the large number of system assessments required for design optimisation this approach is not a practical proposition. For this reason an analysis procedure that can be embedded within the optimisation code is needed.

Conversion of the fault tree to its BDD equivalent can introduce significant advantages into the quantitative process. As stated previously, quantitative calculations using the BDD structure do not require prior determination of the minimal cut sets and top event parameters are calculated exactly. The possibility of gross inaccuracies is eliminated. In addition, calculations work directly from the BDD and as such the technique is very flexible and lends itself well to incorporation within other analysis procedures. The following section describes how the BDD analysis is incorporated within the optimisation procedure.

5.2.1.3.1 BADD

A computational method to convert the failure logic represented by the fault tree to a BDD structure is implemented in a program called BADD (Binary Algorithm Decision Diagrams) (Ref. 79), previously developed at Loughborough University.

The causal relations between the gates and events of the fault tree are represented in the 'ats' ascii file format (described in section 2.5.2). The '*.ats' file is read directly into the BADD code, where '*' is the name of the specific fault tree structure input file. The 'ats' file is created by the FAULTTREE+ package, thus removing the need to develop a graphics package for fault tree development. FAULTTREE+ is also used to produce the minimal cut sets during the fault tree validation stage.

The input routine then assigns each basic event a unique integer termed its index. For ease of computation the fault tree is re-configured to a binary tree, i.e. each gate has

only two inputs. Inputs to a vote gate are left unchanged and dealt with by a specialized subroutine.

In essence, BADD employs the *ite* (if-then-else) procedure on the binary tree. The program proceeds from the bottom of the tree in a step-by-step fashion, evaluating the *ite* structure of each gate until the top gate is reached. Once the *ite* structure for a particular gate has been evaluated, the gate is given an index. The algorithm is executed until all gates in the tree are indexed. Calculation of the *ite* structure of a vote gate is handled in a separate subroutine. For a more detailed description of the computer algorithm the reader is referred to reference 96.

The *ite* structure of each gate is entered in an *ite* table. The gate's index corresponds to the position in the table where the gate's *ite* structure is stored. The *ite* table forms the bridge for BDD construction, in that certain rows hold the nodes of the BDD. Each node consists of 3 parts and is defined in the table as an ordered triple. The first column indicates the index value of the associated basic event for the node. The second column states the position of the node on its 1 branch and the third column the position of the 0 branch node. The order of the table is such that the first n rows correspond to the n basic events in the fault tree and the next m rows to the gates of the tree. The $n+m^{\text{th}}$ row defines the root node of the BDD. Any additional rows describe the intermediate events in the *ite* process. Only a portion of the rows in the *ite* table constitute nodes in the BDD.

BADD writes the results of the computation to an output file. The output file provides the non-minimal *ite* table, the number of *ite* calculations performed and the position of the top event. (The minimal *ite* table requires the BDD to undergo a minimising procedure and is used to determine the system's minimal cut sets. This facility is not required in this study.)

Row Number	Basic Event Value	1 Branch Position	0 Branch Position
1	1	-1	0
2	2	-1	0
3	3	-1	0
4	4	-1	0
5	5	-1	0
6	6	-1	0
7	7	-1	0
8	5	6	0
9	1	8	0
10	3	7	0
11	3	16	4
12	1	11	0
13	2	3	0
14	1	17	0
15	1	19	0
16	4	-1	7
17	2	18	0
18	3	8	0
19	2	20	11
20	3	21	4
21	4	-1	22
22	5	23	7
23	6	-1	7

Table 5.3 Ite Table for Fault Tree Example

To illustrate the nature of the ite table the fault tree in figure 2.6 is converted to its ite format, as shown in table 5.3 (-1 is used to distinguish a 1 state terminal vertex from the basic event index value 1). Table 5.4 states the index value associated with each basic event.

Basic Event	Index Value
E1	1
E3	2
E4	3
H1	4
E5	5
H2	6
E2	7

Table 5.4 Basic Event Index Associated with Fault Tree Example

The ite table is essentially a set of instructions to build the equivalent BDD of fault tree 'example'. The output file states the position of the top event in the table and as such defines the root node of the BDD. This sets a start point for BDD construction. The 1 and 0 branches emanating from the root node link to other nodes, whose positions are defined in columns 1 and 2 respectively. In turn, the nodes corresponding to the 0 branch and 1 branch nodes are interpreted and the BDD unfolds. BDD construction terminates when all paths conclude in terminal vertices. The BDD for fault tree example is illustrated in figure 5.11. The index associated with each node is converted back to the name for that basic event. The position in the ite table of each node is also included in the figure.

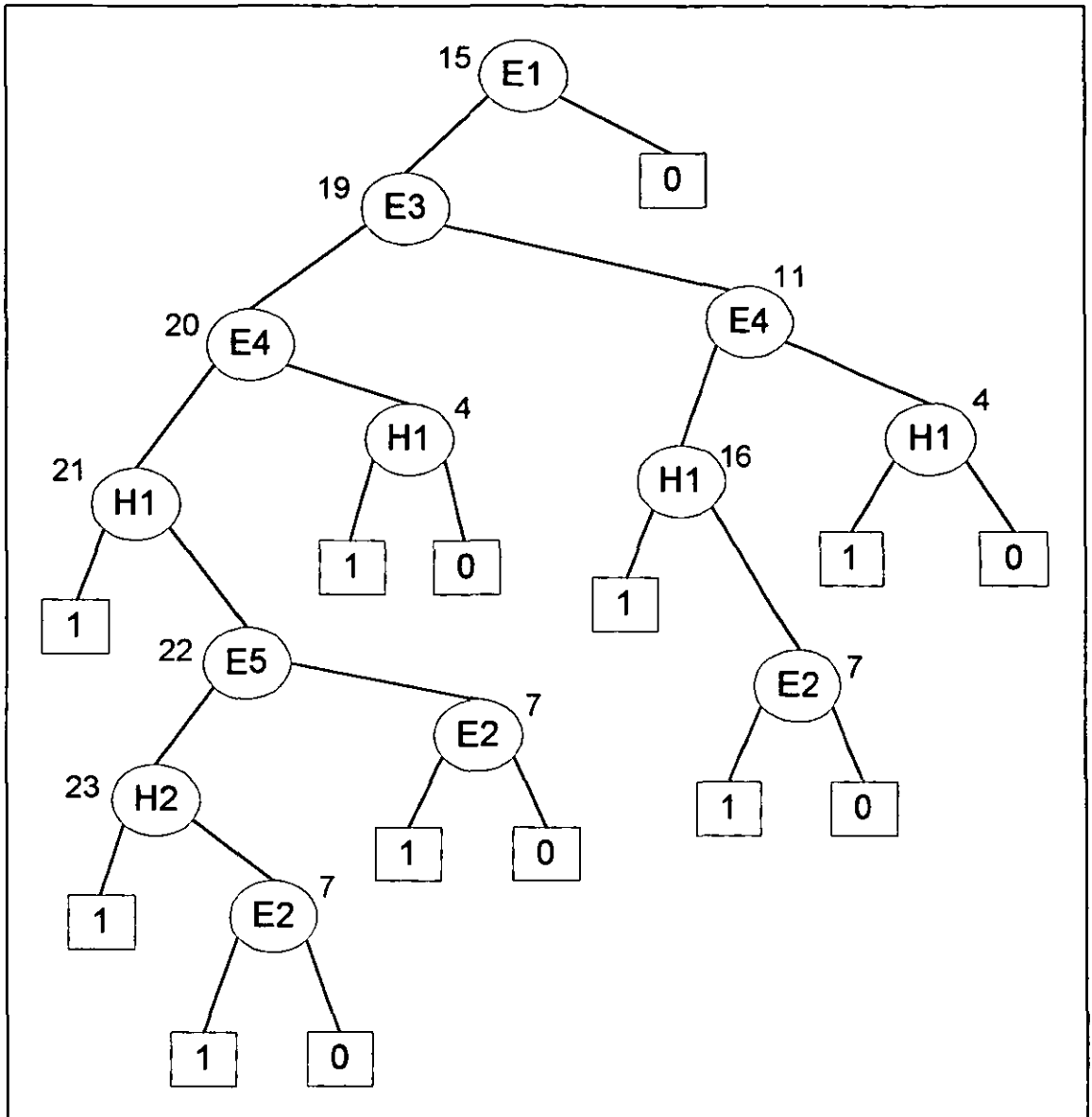


Figure 5.11 BDD for Example

5.2.1.3.2 BDD Structure to Represent System Unavailability

The file structure created from the system unavailability fault tree within FAULTTREE+, *Unavail.ats*, is read into BADD (i.e. a non-minimal ite table consisting of 10888 ite statements). Each primary event is assigned an index value and the root node position defined to be in row 303.

5.2.1.4 Computational Method for BDD Quantification

A computational method is used to calculate the top event probability for each specified safety system design alternative from the system unavailability BDD structure. The program is composed of several subroutines forming a step-by-step computational procedure. The first step involves input of the BDD file structure in the form of the ite table. Data, both fixed and variable, for each primary event is then entered to define the unavailability of each node in the BDD. Where the BDD node represents the occurrence or non-occurrence of a House Event the associated probability is 0 or 1. The final step tracks paths through the BDD, with included data, to evaluate the system unavailability of the considered design. This step-by-step procedure is illustrated in the flowchart in figure 5.12. The following sections explain each aspect of the flowchart in greater detail.

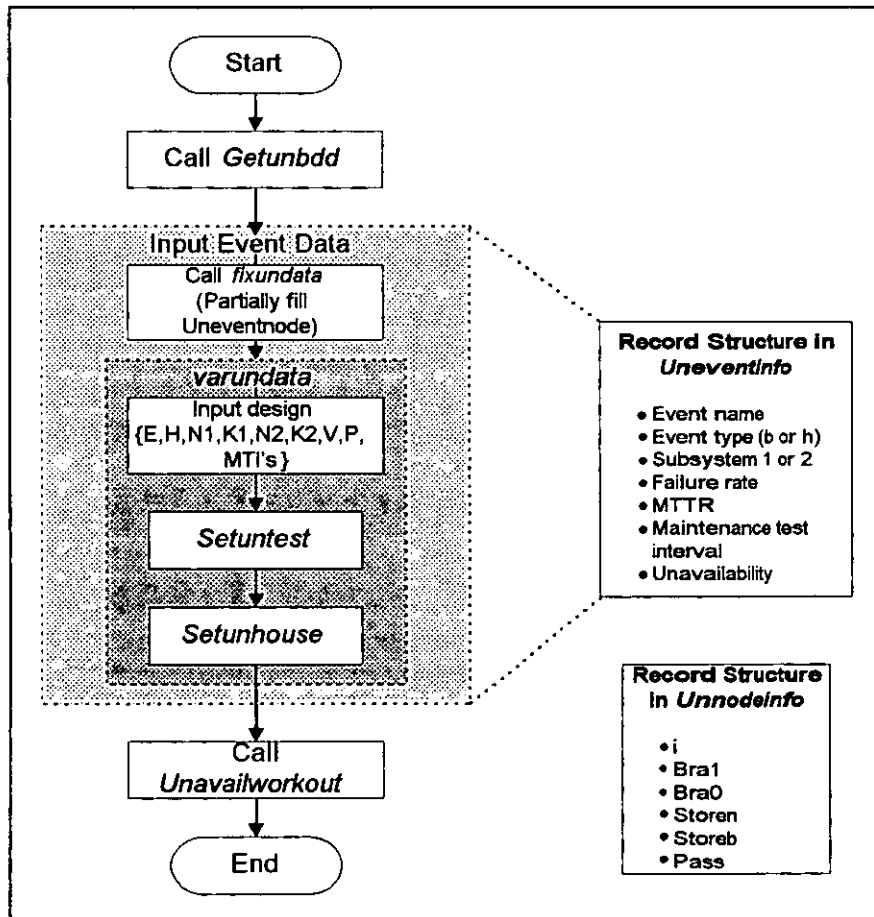


Figure 5.12 Flowchart of the Top Event Probability Computational Method

5.2.1.4.1 Input of the BDD File Structure

On execution of the computerised approach a subroutine called *Getunbdd* reads in the ite table associated with the system unavailability fault tree (determined previously using BADD). A database containing node information, termed *unnodeinfo*, is created with a record number equivalent to the number of rows in the ite table. Each record stores the basic event index value (*i*), the node on the 1 branch (*bra1*) and the 0 branch node (*bra0*). In addition, the root node position, the number of primary events and the number of ite calculations are specified and stored for use in the path tracking routines.

5.2.1.4.2 Input Event Data

The next part of the method compiles a database of records describing each primary event, termed *uneventinfo*. The structure of each record is shown in figure 5.12. Primary events are stored in the database in the order defined by the index associated with the BDD structure (the number of records ranges from 1 to 88).

The subroutine *fixundata* enters all fixed data associated with each event, thereby partially filling *uneventinfo*. The event name, event type and associated subsystem are input in each record. In addition, failure and repair data are specified for each basic event. All other fields are initialised to zero.

The subroutine *varundata* specifies the BDD for a particular design before analysis commences. The user is prompted to enter a particular safety system design parameter set. *varundata* then carries out two separate functions: *setuntest* and *setunhouse*. The former evaluates and stores the unavailability of each basic event using the currently specified inspection interval values whilst the latter assigns appropriate house event probabilities. *Setunhouse* involves a simple if then else structure. This is illustrated in the algorithm in figure 5.13 for the parameter governing the number of pressure transmitters in subsystem 1 (N_1).

Varundata completes the primary event database. The unavailability of each event is defined, where house events are treated as basic events with restricted probabilities of 0 or 1. Data can be acquired for each node in the BDD structure via a partnership between the node's basic event index value and the data stored in the record corresponding to this index point within *uneventinfo*.

```

Setunhouse
  consider  $N_1$ 
    if  $N_1 = 1$ 
       $EN1 = 1$ 
       $NEN2 = NEN3 = NEN4 = 0$ 
    else if  $N_1 = 2$ 
       $EN2 = 1$ 
       $NEN1 = NEN3 = NEN4 = 0$ 
    else if  $N_1 = 3$ 
       $EN3 = 1$ 
       $NEN1 = NEN2 = NEN4 = 0$ 
    else
       $EN4 = 1$ 
       $NEN1 = NEN2 = NEN3 = 0$ 

```

Figure 5.13 Algorithm for N_1 in Subroutine *Setunhouse*

5.2.1.4.3 Analysing the BDD Structure

Having completed data input of *uneventinfo* specific to the safety system design under consideration, quantification of the BDD proceeds. *Unavailworkout* is a routine that tracks each possible path through the tree from the top node to the terminal vertices. The product of probabilities along each path is calculated and the sum of these products established.

To describe *Unavailworkout* the following variables need to be defined:

- *SN* defines the node under consideration at a given moment. Initially *SN* is defined to be the root node of the BDD structure.
- *NNI* defines the 1 branch node of the node being considered, *SN*.
- *NN0* defines the 0 branch node of *SN*.
- *PATH* stores the value derived from the product of each branch on a particular route from the top node to *SN*. *PATH* is initially set to 1 and continually updated as the algorithm progresses. Passing from *SN* to *NNI* multiplies *PATH* by the component unavailability associated with node *SN*. Conversely passing from *SN* to *NN0* multiplies *PATH* by the respective component availability. It is required that *PATH* is rectified by division of the appropriate branch's probability as movement is traced back up the tree.
- *QSYS* stores a running total of the sum of all paths to a terminal 1 vertex. *QSYS* is initially set to 0. Each time a terminal 1 vertex is encountered the value of *PATH* is added to *QSYS*. Ultimately *QSYS* represents the system unavailability of the design as it defines the sum of all product paths resulting in the failed state.

A node is 'complete' if all paths from that node to accessible terminal vertices have been explored. Having traced a specific route to a terminal vertex it is necessary to backtrack and branch out at the most immediate 'incomplete' node onto a different path. The implication is that the tracking procedure requires a memory. As such, the storage structure associated with each node, *unnodeinfo*, is extended. Specifically variables *PASS*, *STOREN* and *STOREB* are introduced. *PASS* keeps a running total of the number of times a path passes through the node. If *PASS* is even the emanating 1 branch of *SN* is checked. Else if *PASS* is odd the 0 branch is traversed. At *SN* the previous node and branch passed (0 or 1) to reach *SN* are stored, i.e. *STOREN* and *STOREB* respectively. Movement back up the BDD is merely a case of accessing these stored values, where the value of *PASS* is used to determine whether a node is 'complete'.

Unavailworkout consists of three main subroutines: *check1branch*, *check0branch* and *search*. The flowchart in figure 5.14 exhibits the general framework of the overall path tracking routine. *Unavailworkout* is executed until both the 1 and 0 branch of each node has been explored.

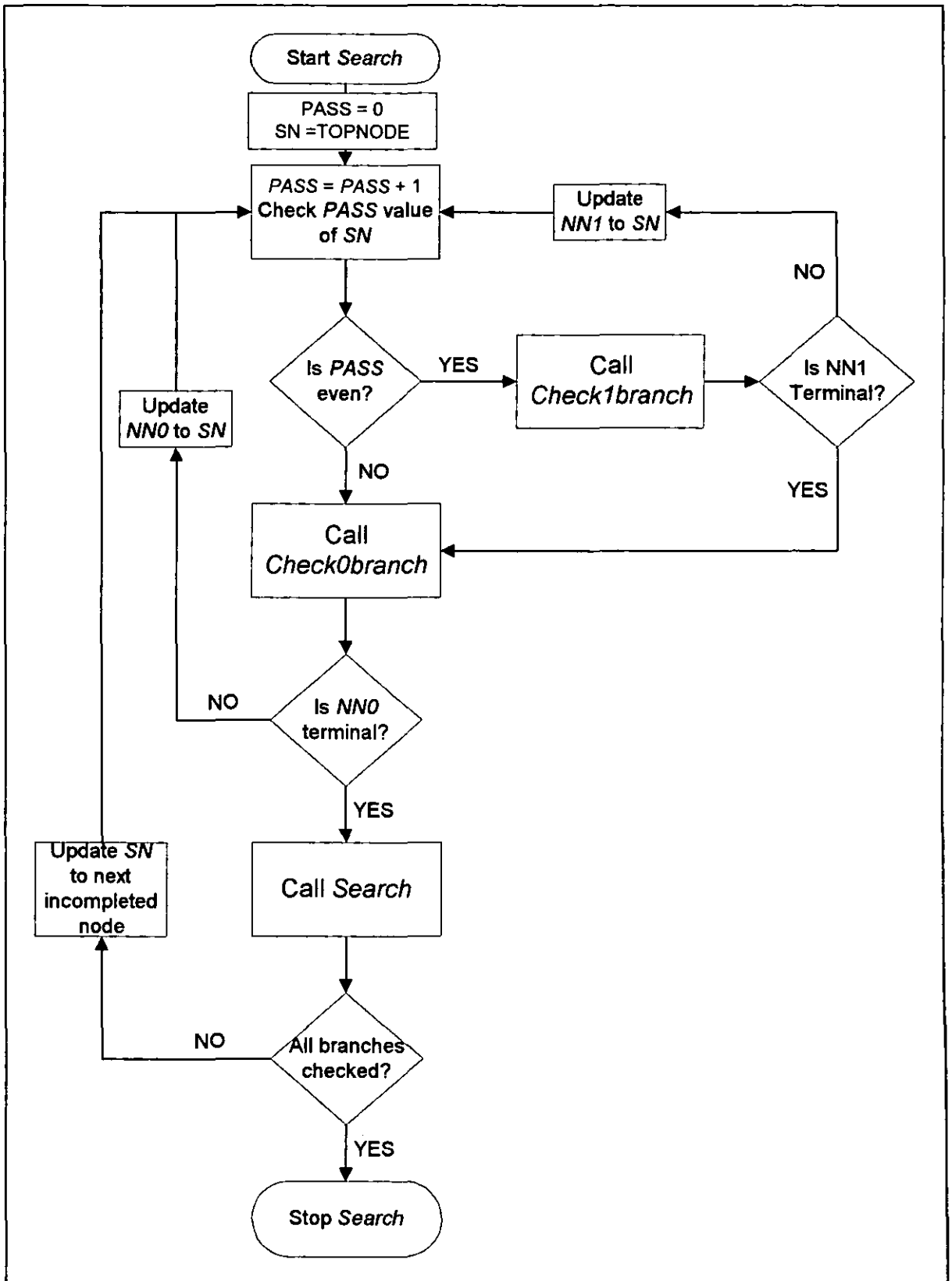


Figure 5.14 Flow Chart illustrating Subroutine *Unavailworkout*

Unavailworkout considers node SN. If PASS associated with SN is even check1branch is called. An initial check to determine whether NN1 is a house event is performed. This checking process is represented in pseudocode in figure 5.15.

```

while NN1 = 'h'
  if  $q_{NN1} = 1$ 
    update NN1: NN1 = 1branch node of NN1
  if  $q_{NN1} = 0$ 
    update NN1: NN1 = 0 branch node of NN1

```

Figure 5.15 Algorithm to Check for House Events

If NN1 is a house event the probability of passing down one of its branches is absolute. This branch is, therefore, traversed and the other eliminated. Information concerning passage across the house event is not stored. Movement back up the tree skips over the house event and hence, the other branch will never be checked.

The resulting node, NN1, which exits the checking loop is not a house event. Subsequent action depends on this node type. The steps taken if NN1 is a basic event are shown in the algorithm in figure 5.16. Flow logic then returns the updated node, SN, to the beginning of *Unavailworkout*, its PASS value is checked and the process continues.

```

if NN1 = 'b'
   $PATH = q_{NN1} \times PATH$ 
  STOREN = SN
  STOREB = 1
  update SN: SN = NN1
  break

```

Figure 5.16 Algorithm if NN1 is a Basic Event

Conversely, if NN1 is a terminal vertex the process followed is as illustrated in figure 5.17. Flow logic then sends SN to subroutine check0branch.

```

if  $NN1 = -1$ 
     $PATH = q_{NN1} \times PATH$ 
     $QSYS = PATH + QSYS$ 
     $PATH = PATH / q_{NN1}$ 
    break
else if  $NN1 = 0$ 
    break

```

Figure 5.17 Algorithm if NN1 is a Terminal Vertex

Subroutine *check0branch* proceeds in a similar manner. It differs, however, when *NN0* results in a terminal vertex. In this case, both branches of *SN* have been checked and *SN* has proved 'complete'. Subroutine search is, therefore, executed.

```

let  $TEMPNODE = STOREN$  of  $SN$ 
while  $PASS$  value of  $TEMPNODE$  is even

    move back up tree and update  $PATH$ 
    if  $STOREB$  of  $SN = 1$ 
         $PATH = PATH / q_{TEMPNODE}$ 
    else if  $STOREB$  of  $SN = 0$ 
         $PATH = PATH / (1 - q_{TEMPNODE})$ 

     $SN = TEMPNODE$ 
     $TEMPNODE = STOREN$  of  $TEMPNODE$ 
    if  $TEMPNODE = root$  node
        stop

if  $STOREB$  of  $SN = 1$ 
     $PATH = PATH / q_{TEMP-NODE}$ 
else if  $STOREB$  of  $SN = 0$ 
     $PATH = PATH / (1 - q_{TEMPNODE})$ 
 $SN = TEMPNODE$ 

if  $SN = root$  node
    if  $PASS$  value of root node is even
        stop
    else
        continue

```

Figure 5.18 Algorithm for Subroutine Search

Figure 5.18 shows the pseudocode to describe search. Stored values associated with each node are used to update the value of *PATH* and find the next ‘incomplete’ node. Search terminates *Unavailworkout* if all branches have been checked. Termination is realised if *SN* is again defined as being the root node and the root node has an even *PASS* value.

5.2.2 Cost and MDT Evaluation

Constraints fall into two categories: those that can be determined from an explicit function of the design variables and are, therefore, easily evaluated, and those that cannot be expressed as a function and can only be evaluated by a full analysis of the system. The former are termed explicit constraints and the latter referred to as implicit.

The spurious trip frequency of the HIPS is an example of an implicit constraint. This is evaluated in a similar manner to the probability of system unavailability and is considered in section 5.2.3.

Cost and MDT are explicit constraints. Total cost is the sum of the cost of subsystem 1 and subsystem 2 and is represented by the following equations

$$\text{COST} = \text{COST}(\text{SUBSYS1}) + \text{COST}(\text{SUBSYS2}) \leq 1000 \quad (5.1)$$

$$\text{COST}(\text{SUBSYS1}) = E(V_1 C_{V1} + V_2 C_{V2} + C_S) + N_1(P_1 C_{P1} + P_2 C_{P2}) + 261 \quad (5.2)$$

$$\text{COST}(\text{SUBSYS2}) = H(V_1 C_{V1} + V_2 C_{V2} + C_S) + N_2(P_1 C_{P1} + P_2 C_{P2}) + 21 \quad (5.3)$$

where

C_{V1}, C_{V2} = costs of the two valve types

C_{P1}, C_{P2} = Costs of the two pressure transmitter types

C_S = cost of the solenoid valves

E = the number of ESD valves fitted

H = the number of HIPS valves fitted

Subsystem 1 necessarily includes a wing and master valve, their solenoid valves, the computer and control relays. Hence, the fixed cost of 261 units included in equation (5.2). The extra cost depends on the number and type of ESD valves and the number and type of pressure transmitters. Subsystem 2 has a fixed cost of 21 units due to the computer and control relays, hence the constant in equation (5.3).

Similarly average MDT is a sum of subsystem 1 and subsystem 2, as shown in the following equations

$$\text{MDT} = \text{MDT}(\text{SUBSYS1}) + \text{MDT}(\text{SUBSYS2}) \leq 130 \quad (5.4)$$

$$\text{MDT}(\text{SUBSYS1}) = \frac{52}{\theta_1} [E(V_1 M_{V1} + V_2 M_{V2} + M_S) + N_1(P_1 M_{P1} + P_2 M_{P2}) + 47] \quad (5.5)$$

$$\text{MDT}(\text{SUBSYS2}) = \frac{52}{\theta_2} [H(V_1 M_{V1} + V_2 M_{V2} + M_S) + N_2(P_1 M_{P1} + P_2 M_{P2}) + 13] \quad (5.6)$$

where

M_{V1}, M_{V2} = test times of the two valve types

M_{P1}, M_{P2} = test times of the two pressure transmitter types

M_S = test time of the solenoid valves

E and H are the number of ESD and HIPS valves fitted respectively

The constant 47 in equation (5.5) is the test time for the wing and master valve, their solenoids, the computer and control relay for subsystem 1. The test time for the computer and control relay for subsystem 2 is 13 units, as stated in equation (5.6).

5.2.3 Evaluate the Frequency of Spurious Trip Occurrences

5.2.3.1 Construction of the Spurious Trip Fault Tree

Due to the constraint limiting the number of spurious system trips permitted, attention is given to a second system failure mode, i.e. spurious activation of the HIPS. The specific fault tree resulting in this culminating event must be developed. The top event will occur if any one of the valves included along the pipeline closes spuriously.

Each valve has air-to-open fail safe characteristics. When there is no demand on the system to act, failure of any component such that the air supply to a valve is removed (or failure of the valve itself) will result in a spurious trip.

The immediate, necessary, and sufficient sub-events to the top event, related by OR logic, are 'Wing or Master Valves Fail Spuriously', 'ESD Subsystem Fails Spuriously' OR 'HIPS Subsystem Fails Spuriously'. This causal relationship is illustrated in figure 5.19.

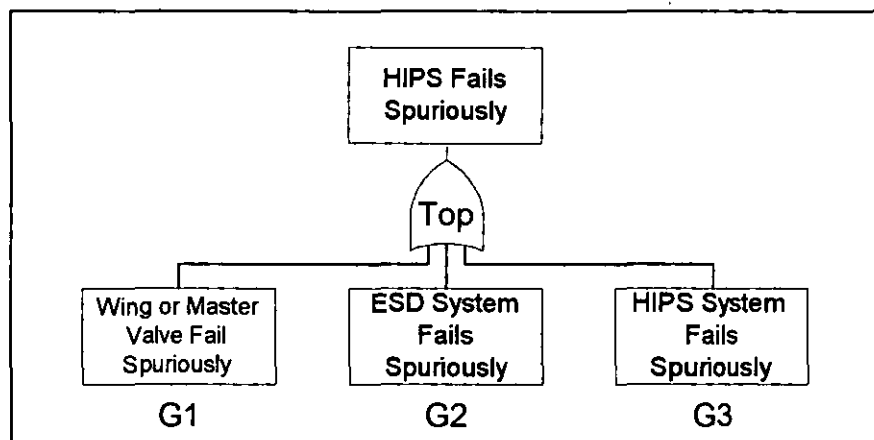


Figure 5.19 Top event of Spurious Trip Fault Tree

Clearly, the major difference between this and the system unavailability fault tree is the lack of redundancy in the first level of the tree structure.

It should be noted that house events incorporated in the spurious trip fault tree structure are consistent with those in the system unavailability fault tree. The failure mode of the design under consideration is different but the structural characteristics of the design remain the same in each case.

Wing or Master Valve Fails Spuriously

Spurious failure of either the wing or master valve results in a spurious system failure. This gives rise to the next level intermediate events 'Wing Valve Closes Spuriously' OR 'Master Valve Closes Spuriously'. Further development of the intermediate wing valve event closely resembles that of the wing valve failing stuck in the system unavailability fault tree. Components and their casual relationships involved in the structure of the branch are almost the same. The branch differs in the failure modes of the included basic events. Additionally, spurious action of either relay contact instigates the flow of fault logic and trips the system (see figure 5.20).

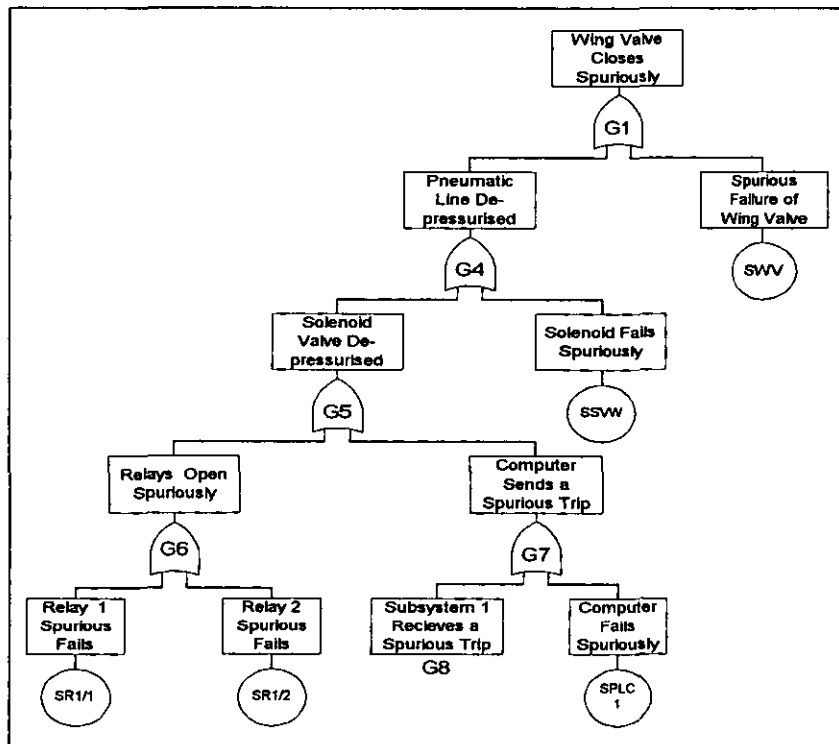


Figure 5.20 Wing Valve Fails Spuriously

Structural differences arise below the intermediate event, 'Subsystem 1 Receives a Spurious Trip', G8. Sub-events are again partitioned into 4 separate channel failures, where channel 1 indicates inclusion of 1 pressure transmitter etc. However, Or logic associates the channels and spurious channel failure occurs if the relevant number of pressure transmitters are fitted (as defined by house events EN1, EN2, EN3 or EN4) and the relevant trip combination occurs (i.e. house events EK1, EK2, EK3 or EK4). The mutually exclusive house event pairings (e.g. NEN1 and EN1) are, therefore, not required, as shown in figure 5.21.

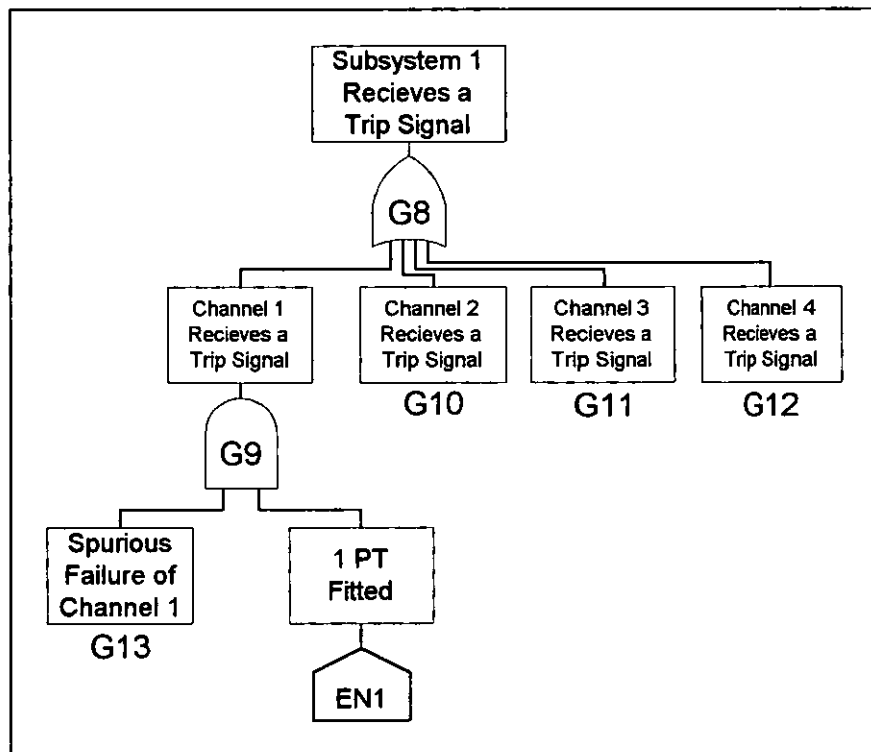


Figure 5.21. Subsystem 1 Receives a Trip Signal

Further structural differences exist within each channel. Consider channel 2. In this case K_j can be either 1 or 2. If only 1 pressure transmitter is required to trip the system, spurious failure of either transmitter will initiate spurious system action. Conversely, if 2 pressure transmitters must register an increase in pressure, spurious failure of both is required to cause spurious system failure. This is the reverse scenario to dormant failure of channel 2. The other channels are developed in a similar vein.

ESD and HIPS Subsystems Fail Spuriously

Spurious failure of any included ESD valve will cause the ESD subsystem to fail spuriously. Further development of the resulting intermediate events: 'ESD Valve 1 Closes Spuriously' OR 'ESD Valve 2 Closes Spuriously', mimics the structure in the system unavailability fault tree.

5.2.3.2 Data Input

In total the spurious trip fault tree consists of 74 primary events and 150 gates. Of the 74 primary events, 44 are basic events and 30 house events. The reduction in house events in relation to the system unavailability fault tree is due to the removal of house events that verify a component's non-existence. The basic events correspond to the components in the system unavailability tree, however, they represent the component's spurious failure. A spurious failure results in system activation when no demand is given. As such, spurious failures are instantaneously revealed and repair can be instigated immediately. The constant failure and repair rate model is used to quantify each basic event, i.e. the failure rate (λ) and repair rate (ν) (where ν is the reciprocal of the MTTR) are required.

The dependency of each basic event's probability of spurious failure on its associated maintenance test interval is eliminated. Consequently, spurious failure data for each basic event remains fixed from one design to the next. The parameter set of a design modifies only the house event probabilities.

5.2.3.3 Conversion of the Spurious Trip Fault Tree to a BDD

The file structure representing the spurious trip fault tree structure, *spurious.ats*, is read into BADD. A non-minimal ite table consisting of 10788 calculations results. Each primary event is assigned an index value and the root node position defined to be in row 289.

5.2.3.4 A Computational Method for the Top Event Unconditional Failure Intensity of the BDD

A computational method is implemented to determine the unconditional failure intensity of the BDD structure corresponding to the spurious trip fault tree. Integrating the resulting value over an allocated time gives the expected number of trip occurrences over that period.

The step-by-step computational procedure is similar to that used to quantify the top event probability. Initially the relevant BDD file structure is read. The BDD is then modified to correspond to a specific safety system design. Finally the appropriate path tracking technique is activated to evaluate the unconditional failure intensity. These procedures are described in detail below.

5.2.3.4.1 Input of BDD File Structure and Event Data

On execution of the computerised approach a subroutine called *Getspbdd* reads in the ite table associated with the spurious trip fault tree. A database containing node information, termed *Spnodeinfo*, is created with a record number equivalent to the number of rows in the ite table. In addition, the root node position, the number of primary events and the number of ite calculations are obtained and stored.

Step 2 compiles a database of records which describe each primary event, termed *Speventinfo*. The fields in each record are akin to those in *Uneventinfo*. However, failure and repair data correspond to spurious failure, the maintenance test interval field is removed and a field to store each components unconditional failure intensity is added. Primary events are stored in *Speventdata* in the index order associated with the spurious BDD file structure.

Subroutine *Spfixdata* mimics *Unfixdata* in that it enters all fixed data associated with each primary event. Due to that fact that spurious failure is instantaneously revealed, both component unavailability and unconditional failure intensity remain constant from one design to the next and as such, can be evaluated and stored within *Spfixdata*.

5.2.3.4.2 Analysing the BDD Structure

Evaluating the top event unconditional failure intensity requires the determination of the criticality function for each primary event. This is achieved through summation of the criticality function of each related node within the BDD, where the criticality function of each node requires the determination of *Probpost0*, *Probpost1* and *Probprev* values, i.e. $po_{xi}^0(q)$, $po_{xi}^1(q)$ and $pr_{xi}(q)$ respectively (introduced in section 2.6.6.2). Routine *Spuriousworkout* determines the top event unconditional failure intensity using subroutines *Nodecf*, *Eventcf* and *Sysufi*.

Evaluating the Criticality Function of Each Node

Prior to describing *Nodecf* consideration is given to each record within *Spnodeinfo*. Fig 5.22 illustrates the structure of each record. The first six fields are equivalent to those in *Unnodeinfo* in *Unavailworkout*. Fields *i*, *BRA1* and *BRA0* are specified in *Getspbdd* and define the spurious BDD structure. *STOREN*, *STOREB* and *PASS* enable storage of the path traced through the BDD, in effect associating a memory with the path tracking procedure. The extra fields are specific to system unconditional failure intensity evaluation. *CHECKPOST* is a binary marker that basically states whether the node has been assigned *Probpost* values (a 1 indicator) or not (a 0 indicator). *PROBPREV*, *PROBPOST1*, *PROBPOST0* and *CF* store the relevant values for the particular node, where *CF* denotes the criticality function.

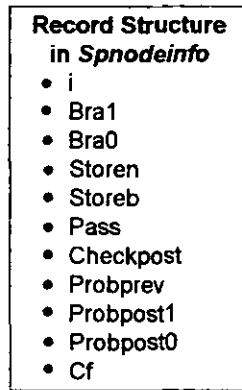


Figure 5.22 Structure of a Record in *Snodeinfo*

Nodecf traces a path through the BDD using a similar procedure to that of *Unavailworkout*, primarily to determine each node's *Probprev* value. *Nodecf* differs in the manner it calculates and stores *PATH*. *Probprev* for node *Xi* is concerned with the path section from the root node to *Xi*. Within *Check1branch*, if *NNI* is a basic event, $pr_{NN1}(q)$ is updated using the current *PATH* value, as demonstrated in figure 5.23.

```

within Check1branch
  If NNI = 'b'
     $PATH = PATH \times q_{SN}$ 
     $pr_{NN1}(q) += PATH$ 
    STOREN of NNI = SN
    STOREB of NNI = 1
    SN = NNI
  break

```

Figure 5.23 Algorithm for the *PATH* tracking section of *Check1branch*

If *NNI* is a terminal vertex no action regarding *PATH* is taken and the subroutine to check the 0 branch is called.

Similarly, within *Check0branch*, if *NNI* is a basic event, the probability of the preceding 0 branch is incorporated in $pr_{NN1}(q)$. To achieve this *PATH* is first updated via multiplication by the preceding 0 branch probability, $(1 - q_{SV})$. If *NNI* is terminal, subroutine *Search* is implemented to backtrack up the BDD and find the most immediate 'incomplete' node.

Each time a node is updated to SN within *Nodecf*, a check is carried out to determine whether the *Probpost* values, i.e. $po_{SN}^1(q)$ and $po_{SN}^0(q)$, of that node have been evaluated. If the check proves negative the node is sent to subroutine *Post*. *Post* incorporates two functions: *Post1func* and *Post0func*. Initially the 1 branch node, $NN1$, of SN is determined. $NN1$ is passed to *Post1func*. *Post1func* mimics *Unavailworkout* with $NN1$ as the top event. $QSYS$, i.e. the product of all paths to 1 from $NN1$, is stored in $po_{SN}^1(q)$. Subsequently, the 0-branch node, $NN0$, of SN is passed to *Post0func*. Similarly, *Post0func* mimics *Unavailworkout*, where $NN0$ is the top event, and $QSYS$ is stored in $po_{SN}^0(q)$. A flowchart depicting *Post* and its relation to *Nodecf* is shown in figure 5.24. On exit of *Nodecf* the values $po_{xi}^0(q)$, $po_{xi}^1(q)$ and $pr_{xi}(q)$ have been evaluated for each node.

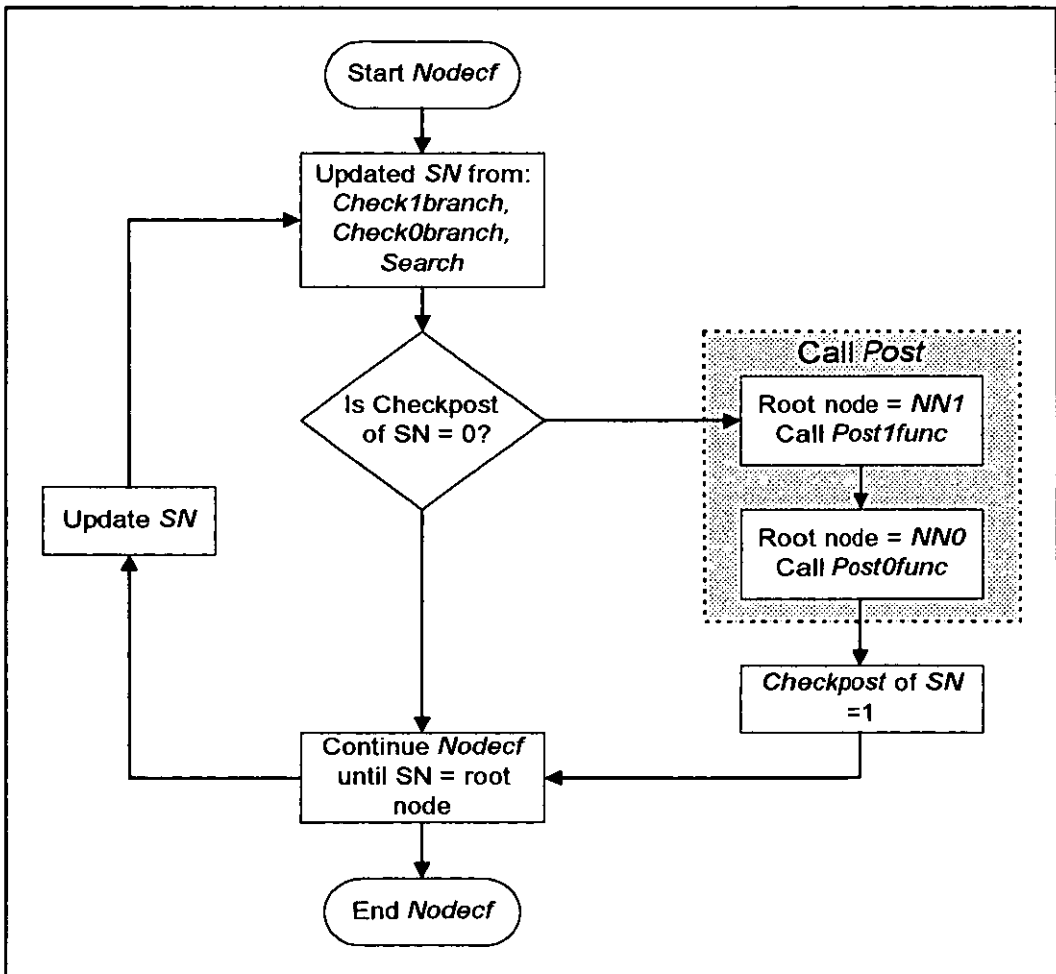


Figure 5.24 Flowchart for Subroutine *Post*

Evaluate the Criticality Function of Each Basic Event

Subroutine *Eventcf* calculates the criticality function (introduced in section 2.6.6.2) for each basic event as shown in figure 5.25, using equation (2.52). The calculation of $G_{Xi}(q)$ for each node simply requires the values $po_{Xi}^1(q)$, $po_{Xi}^0(q)$, and $pr_{Xi}(q)$. The calculation of $G_i(q)$ for each basic event is the summation of associated node criticality function values.

```
set  $G_i(q) = 0$  for all  $i$ 
do for all nodes  $X_i$ 
     $G_{Xi}(q) = pr_{Xi}(q) | po_{Xi}^1(q) - po_{Xi}^0(q) |$ 
do for all basic events  $i$ 
    check all nodes  $X_i$ 
        if  $X_i$  is associated with  $i$ 
             $G_i(q) = G_i(q) + G_{Xi}(q)$ 
```

Figure 5.25 Algorithm to Evaluate the Criticality Function $G_i(q)$

Evaluate the System Unconditional Failure Intensity

A final subroutine *Sysufi* to determine the system unconditional failure intensity (introduced in section 2.6.6.2) is illustrated in figure 5.26, using equation (2.44). An allocated time period, input by the user, subsequently determines the expected number of top event occurrences over that period.

```
Set  $w_{SYS}(t) = 0$ 
Do for all basic events  $i$ 
     $w_{SYS}(t) += G_i(q) \times w_i$ 
Time period =  $T$ 
 $W(0, T) = \text{integral}_0^T (w_{SYS}(t))$ 
```

Figure 5.26 *Sysufi* Algorithm

5.2.4 Accuracy Comparison with FAULTTREE+

To compare the accuracy of the computerised BDD technique with the conventional Kinetic Tree Theory approach of FAULTTREE+, three safety system designs were analysed. Table 5.2 defines the parameter set for each considered design, where the final rows state the corresponding system unavailability and spurious trip occurrence using both quantitative techniques. In addition, the cost and maintenance down time of each design is portrayed.

Variables	Design 1	Design 2	Design 3
E	0	1	2
K₁/N₁	1/2	3/4	1/1
H	2	0	1
K₂/N₂	2/3	0/0	4/4
V	1	2	2
P	1	1	2
θ₁	40	50	30
θ₂	30	34	30
MDT	130.4	58.2	164.7
Cost	922	561	992
Q_{sys} FAULTTREE+	9.76e-3	4.288e-2	5.87e-3
Q_{sys} BDD	9.70e-4	4.287e-2	5.5e-3
F_{sys} FAULTTREE+	0.561	0.242	0.544
F_{sys} BDD	0.551	0.241	0.542

Table 5.2 Quantification Results of Three Safety System Designs

CHAPTER 6

IMPLEMENTING THE GA TO OPTIMISE THE HIGH-INTEGRITY PROTECTION SYSTEM

6.1 Introduction

It was decided to investigate the effectiveness of Genetic Algorithms (GAs) to optimise safety system availability. The GA was initially applied to the High Integrity Protection System (HIPS). SGA_C is a C-language translation and extension of the original Pascal Simple Genetic Algorithm (SGA) code presented by Goldberg (Ref 37). This package was used as a framework to build the GA software for the HIPS optimisation called GASSOP (Genetic Algorithm Safety System Optimisation Procedure).

GAs are iterative procedures that maintain a population of candidate solutions to some objective criteria. Each safety system design is indicative of a specific set of parameter values representing a point in the search space. GASSOP commences with a diverse population of designs. Each design's performance is evaluated using a preconceived criteria, in this case availability. The set of performance measures is subsequently used within a selection procedure to create a new population of candidate solutions, which enable greater exploration of the search space. A second iteration commences using the new population. Each iteration is termed a generation. The iterative procedure terminates after a pre-set number of generations. An algorithm summarising the steps of GASSOP is given in figure 6.1.

```

Procedure GA
  gen = 0
  input GA parameters
  initialize P(0)
  evaluate P(0)
  do while gen ≤ maxgen
    select P(gen + 1) from P(gen)
    genetic operator action
    evaluate P(gen + 1)
    gen += 1
  Report

```

Figure 6.1 A Simple Genetic Algorithm

6.2 Computer Implementation of the GA to Optimise the HIPS – GASSOP

The following is an outline of the routines constituting GASSOP and the subroutines they contain.

6.2.1 Input to GASSOP

Initialise is the central initialisation routine called on execution of GASSOP and consists of subroutines *Initdata*, *Memory* and *Initpop*.

The first step, *Initdata*, prompts the user for SGA parameters. The user may choose parameters governing:

- Population size (*popsize*),
- Number of generations (*maxgen*),
- Crossover rate (*pcross*),
- Mutation rate (*pmutate*).

Memory handles dynamic memory management. The data entered in *Initpop* is used to dynamically allocate space for the GA population. Each individual, i.e. design, in the population accumulates a certain amount of descriptive data throughout the GA

run. As such, on execution of GASSOP, enough storage space must be allocated for data storage in the form of a record per individual. The structure of each record is shown in figure 6.2.

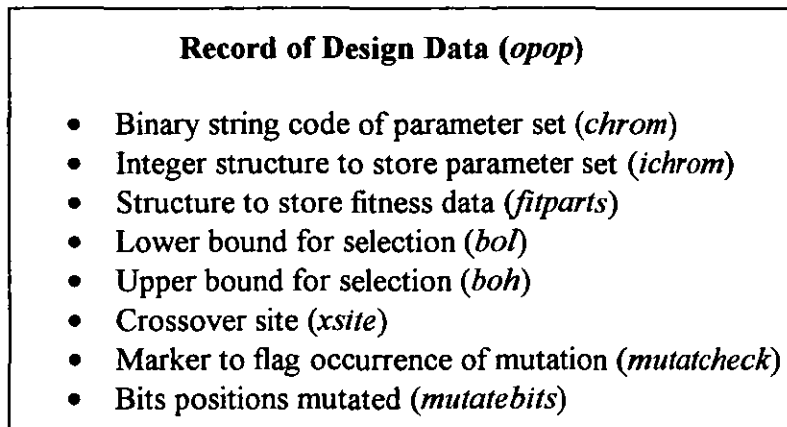


Figure 6.2 Record to Store Data for Each Design - *opop*

Each record itself contains two further structures: *ichrom* and *fitparts*. *Ichrom* consists of a set of bit fields to store each parameter value for the considered design in its integer form. In contrast, *chrom* stores the parameter set as a concatenated string of integer values in binary form. (Both *ichrom* and *chrom* will be considered in greater detail when describing the subroutine *Initpop*). *Fitparts* is a structure to store all fitness data associated with the design and will be considered further in section 6.2.3. *Bol* and *boh* store the upper and lower values to partition the roulette wheel, as described later within the *Selection* routine. Fields *xsite*, *mutatcheck* and *mutatbits* store the relevant data to report action taken by the genetic operators. The database of records describing each individual is termed *opop*.

A replica storage structure is allocated simultaneously. This space will store individuals chosen from the original population for reproduction and is termed *npop*. Individuals chosen in the selection stage are transferred to *npop*. Genetic operators subsequently act on designs stored within *npop*. The fitness information of the modified designs is updated and the new population is transferred back to the original storage space, *opop*, prior to the next iteration.

6.2.2 Coding and Initialisation

Each solution string represents a particular system design depending on the values assigned to each of its 10 parameters, where each parameter lies within a specific stated range. It was decided to initialise a population of strings randomly and represent parameter values in binary code.

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

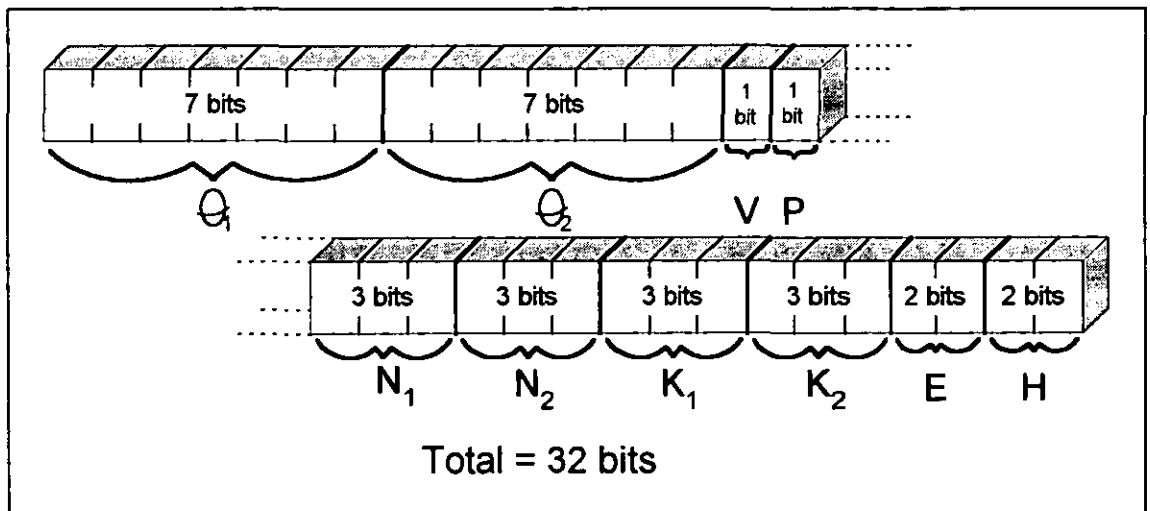


Figure 6.3 A Design's Parameter Set Coded as a Binary String

For ease of computation, the order of parameters on the string was chosen primarily to ensure that parameters did not cross the boundaries of a 16 bit word-length and each followed directly from the last, leaving no gaps.

A means to compile each solution string is to initialise random strings of 0's and 1's directly as in Goldberg's approach. However, the restricted range of each parameter is not necessarily equivalent to its corresponding binary range defined by the

allocated bit field length. For example, consider N_1 , the parameter governing the number of pressure transmitters in subsystem 1. The design can accommodate 0 through 4 pressure transmitters, thus 3 bits are allocated to this parameter. Random initialisation of these three bits as described above will decode to a value in the range 0 to 7. This can yield 3 infeasible results. To overcome this the initialisation procedure termed *Initpop* creates the binary string in two stages.

The first stage initialises the string in integer form parameter by parameter using a random number function, which only generates numbers that are feasible. These integer values are stored in their respective bit field within the *ichrom* structure. Presence or absence of the HIPS valve determines the state of subsystem 2. If no HIPS valve is fitted there is no redundant subsystem. A function is, therefore, incorporated to check the value of H . If necessary, i.e. if $H = 0$, the associated parameters, i.e. N_2 , K_2 , θ_2 , are modified to zero. The range of N_2 and K_2 is hence 1 to 4, where they only assume a zero value as dictated by H .

Next subroutine *Binconversion* uses a left shift and masking technique to insert each initialised value, in its binary form, into the 32 bit solution string. The binary string covers two bytes using a computer with 16 bit word-length. Hence, *chrom* is an integer array of size 2. To demonstrate *Binconversion* figure 6.4 illustrates compilation of the first 16 bits of the 32 bit string. The final 16 bits are formed in the same manner.

```

Binconversion for string j
chrom[0]j =  $\theta_1$ 
chrom[0]j <shift to left> 9
temp =  $\theta_2$ 
temp <shift to left> 2
chrom[0]j <OR-logic> temp
temp = V
temp <shift to left> 1
chrom[0]j <OR-logic> temp
temp = P
chrom[0]j <OR-logic> temp

```

Figure 6.4 Algorithm to Compile Binary String - *Binconversion*

Initpop is carried out *popsiz*e times to create a complete population of feasible individuals.

6.2.3 Evaluating String Fitness

A simple explicit objective function to calculate the fitness of each design does not exist. String fitness comprises of system unavailability plus the respective penalty should any of the constraints be violated, i.e. a penalty for exceeding the cost limit of the design envelope, a penalty for exceeding the total maintenance down time constraint and a penalty for exceeding the allocated trip frequency allowance per year.

A routine called *Fitness* obtains a raw fitness value for each design. The data fields that constitute the structure *fitparts*, associated with each individual, are as follows:

- Cost of subsystem 1 (*csub1*),
- Cost of subsystem 2 (*csub2*),
- Total system cost (*tcost*),
- Maintenance down time for subsystem 1 (*mdt1*),
- Maintenance down time for subsystem 2 (*mdt2*),
- Total maintenance down time (*tmdt*),
- Spurious trip frequency (F_{SYS}),
- Probability of system unavailability (Q_{SYS}),
- Penalty due to excess cost (*penC*),
- Penalty due to excess MDT (*penM*),
- Penalty due to excess spurious trip frequency (*penS*),
- Penalised system unavailability (Q'_{SYS}),
- Roulette wheel percentage (*p*).

Fitness contains subroutines to evaluate and store this data and follows the algorithm illustrated in figure 6.5. (Determination of each string's roulette wheel percentage is considered within the *Selection* routine).

```

Fitness
  Unavail
    if gen = 0
      Getunbdd
      Fixundata
    do for all strings j
      Varundata
      Unavailworkout
      QSYS = QSRSj
  Spurious
    if gen = 0
      Getspbdd
      Fixspdata
    do for all strings j
      Varspdata
      Spuriousworkout
      FSYS = FSRSj
do for all j
  evaluate MDT
  evaluate cost
  Penaltys
Genaverage
Genbest

```

Figure 6.5 Fitness Evaluation Algorithm

Fitness first evaluates each fitness aspect, i.e. Q_{SRS} , F_{SRS} , cost and MDT. Calculating each string's cost and MDT is fairly straightforward. Both these constraints use an explicit formula in which the respective data values and specific parameter set are entered to give the required results. Evaluating the system unavailability and the spurious trip frequency is a more in depth process. A BDD representing each failure mode is incorporated into GASSOP and used directly to quantify the system failure parameters. The routines to achieve this are discussed in chapter 5. These routines are integrated within *Unavail* and *Spurious* with slight modifications in order to handle a population of strings within the GA environment, as shown in figure 6.5. Fixed data, i.e. the BDD structure and component failure and repair data, need only be entered once and as such, are carried out on the first iteration alone. Each time a new design is considered the variable data must be specified prior to quantification of the BDD. Consequently, automatic manipulation of House Event probabilities and

maintenance test interval parameters establishes the unavailability of each primary event for a particular design. As such, the method is capable of specifying the probabilities of each node, in both the system unavailability and spurious trip BDD, automatically within the GA software package for each design alternative.

String fitness must, however, be represented in the GA as a single value. This value is subsequently used to determine which members of the population will be reproduced into the next generation. The system unavailability is the value in which we are ultimately interested. Resources are not, however, inexhaustible. In consequence, the values for cost, maintenance down time (MDT) and spurious trip frequency are used to penalise the probability of system unavailability if, and only if, they exceed their respective limits:

- Cost > 1000 units
- MDT > 130 hours
- $F_{sys} > 1$ per year

Penalty formulae were, thus, derived for cost, MDT and spurious trip and implemented within the subroutine *Penaltys*.

6.2.3.1 Derivation of Penalty Formulae

During MDT components in the safety system are being inspected rendering the safety system unavailable. As such, MDT above 130 hours directly produces a contribution to the unavailability of the system. 130 hours or below is feasible and incurs no penalty. Hence, if the MDT of a particular string exceeds 130 hours the respective penalty is

$$M_p = \frac{(\text{MDT} - 130)}{8760} \quad \text{for MDT} > 130 \quad (6.1)$$

where M_p denotes the penalty due to excess MDT and 8760 the number of hours per year.

Consideration is now given to the cost constraint. This proves a little more difficult to formulate as excess cost does not directly translate to system unavailability. The method utilised tries to form a direct relationship between cost and performance. If cost exceeds its allocated limit by 100 units, i.e. 10% of the total budget, a corresponding increase in performance of at least 10% is expected.

For this approach it is required to know the level of performance associated with a system which costs 1000 units. Here an assumption has to be made that a typical system of this cost would have an unavailability of about 0.02. Therefore, if modifications to the system cause the cost to increase to 1100 units, it is expected that system unavailability will decrease to 0.018. String designs with excessive cost will not be adopted so the more the constraint violation the heavier the penalty. This is implemented using an exponential relationship of the form $y = x^{5/4}$, chosen by trial and error. Consequently, the penalty function for excess cost is

$$C_P = \left(\frac{\text{COST} - 1000}{100} \right)^{5/4} \times 0.002 \quad (6.2)$$

where C_P denotes the penalty due to excess cost. Figure 6.6 demonstrates application of the penalty function for excess cost.

Consider string A:

$\text{Cost} = 1150 \quad Q_{\text{SYS}} = 0.011$

- $C_P = \left[\frac{(1150 - 1000)}{100} \right]^{5/4} \times 0.002$
 $= 3.3 \times 10^{-3}$
- $Q'_{\text{SYS}} = 0.011 + 3.3 \times 10^{-3}$
 $= 0.0143$

where Q'_{SYS} denotes the penalised system unavailability

Figure 6.6 Evaluation of the Cost Penalty

The third constraint, excess spurious trip occurrence, is also related to cost. If a spurious trip occurs, production ceases causing a financial loss. It is assumed that the cost per hour for loss of production is 100 units. On average a spurious trip requires 36 hours to repair and only one such occurrence a year is acceptable. If 2 trips were to occur in one year an excess of 36 hours downtime must be tolerated, i.e. a loss of 3600 cost units.

Once the constraint violation has been expressed in terms of cost, the formula given in equation 1 can be used to relate this to the system unavailability. The algorithm of the calculations required to find the spurious trip penalty for a particular design is illustrated in figure 7.

Consider string A:

$$F_{SYS}^A = 1.3845 \text{ per year}$$

- *Exceeds spurious limit by 0.3485*
- *Repair time,*

$$0.3485 \times 36 = 12.546 \text{ hours}$$
- *In money terms,*

$$12.546 \times 100 = 1254.6 \text{ units,}$$

i.e. EXCESS COST = 1254.6 units
- *Using the cost penalty formula*

$$S_P = \left(\frac{1254.6}{100} \right)^{\frac{5}{4}} \times 0.002$$

$$= 0.047$$
where S_p denotes the spurious penalty

Figure 6.7 Evaluation of the Spurious Trip Penalty

Each penalty is subsequently added to the system unavailability to give

$$Q'_{SYS} = Q_{SYS} + M_P + C_P + S_P \tag{6.3}$$

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

6.2.3.2 Fitness Information per Generation

Two final routines analyse the penalised system unavailability of each design in the population. *Genaverage* determines the population average fitness value. *Genbest* determines the fitness of the best design. The best design is stored in a structure termed *Bestever*. *Bestever* stores both the specific parameter set and the fitness information of the best design.

6.2.4 Selection

Selection is the process of determining the number of times a particular design is chosen to enter the next generation. Selection consists of two phases. The first phase is concerned with the transformation of a design's fitness value, i.e. Q'_{SYS} , into a real-valued expectation of an individual's probability of reproducing, p . This is dealt with in a routine called *Fitconversion*. The second phase is the probabilistic selection of individuals for reproduction using the assigned reproduction probabilities. This is carried out in the *Selection* routine.

6.2.4.1 Fitness Conversion Method

Each string enters the selection procedure with an associated fitness value, Q'_{SYS} . In the safety system optimisation problem the smaller Q'_{SYS} , the fitter the string and hence, the greater should be its chance of reproduction. In such cases, a possible approach is to use a reproduction probability based on availability concepts rather than unavailability. This, however, does not provide a useful measure of selection potential. Designs of ultimate interest have unavailability values that lie in the range (0, 0.1). Subtracting these values from one results in numbers in the range (0.9, 1.0). This loses any sensitivity to distinguish between the capabilities of any design. The

fitness information available to the GA is distorted and the GA's ability to differentiate between designs nullified.

The implication is that a specialized conversion method is required. A non-linear translation is required to enforce the selection likelihood of the very good design (with availabilities over 99%) in comparison with poor designs (with availabilities around 90%) and average designs (with availabilities around 98%). This is achieved by a conversion method, which allocates each string to one of three categories according to its fitness value. The range of values covered by each category are represented in table 6.1.

Fitness value domain				
Upper limit		Category		Lower limit
1.0	\geq	3	$>$	0.2
0.2	\geq	2	$>$	0.1
0.1	\geq	1	\geq	1.0

Table 6.1 Categories Represented in the Fitness Domain

Each string in category 3 is automatically allocated zero percent. This category comprises of poor system designs. If they do not feature on the roulette wheel they will be eliminated from the succeeding generation.

Category 2 contains average merit designs. It is, however, important to retain a little diversity in the population. To preserve sensitivity each string is subtracted from the upper category limit, i.e. 0.2, and subsequently allocated some portion of a total of 5% of the roulette wheel.

The strings, which fall into category 1, are of ultimate interest. To enhance their fitness values each string is subtracted from the upper category limit, 0.1. A particular amount of the remaining 95% of the roulette wheel is then allocated to each string, dictated by how fit they are in relation to the other strings in the category.

The algorithm to implement *Fitconversion* is illustrated in figure 6.8. A check function is incorporated to ensure that the roulette wheel totals 100% if either categories 1 or 2 are empty. If all strings fall into category 3 their fitness values are subtracted from one and the entire roulette wheel apportioned accordingly. The latter check is not illustrated in figure 6.8.

```

Fitconversion
do for all j
  if  $Q'_{srsj} \leq 0.1$ 
    temp =  $0.1 - Q'_{srsj}$ 
    sum95 += temp
    n95 += 1
  else if  $0.1 < Q'_{srsj} \leq 0.2$ 
    temp =  $0.2 - Q'_{srsj}$ 
    sum5 += temp
    n5 += 1
  else
     $p_j = 0$ 

check function
if n5 = 0
  mult95 = 100
else
  mult95 = 95
if n95 = 0
  mult5 = 100
else
  mult5 = 5

do for all j
  if  $Q'_{srsj} \leq 0.1$ 
     $p_j = [(0.1 - Q'_{srsj}) / \text{sum95}] \times \text{mult95}$ 
  else if  $0.1 < Q'_{srsj} \leq 0.2$ 
     $p_j = [(0.2 - Q'_{srsj}) / \text{sum5}] \times \text{mult5}$ 

```

Figure 6.8 Algorithm for Converting String Fitness to Reproduction Probability

6.2.4.2 Sampling

Prior to selection a routine termed *Bounds* is implemented. *Bounds* assigns each individual a segment of a line, corresponding to its reproduction probability. The limit of each segment is defined in terms of an upper and lower bound, i.e. *bol* and *boh* respectively, for design *i*. As an example *Bounds* is applied to a population of 4 strings, as shown in figure 6.9.

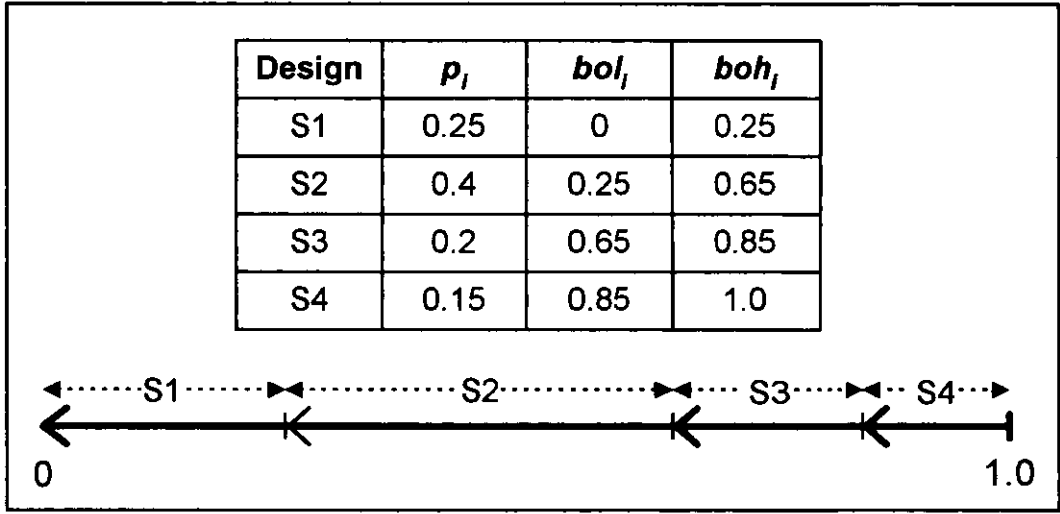


Figure 6.9. An Example of Bounds

Selection in the GA builds a new population of constant size. A random number between 0 and 1 is generated. The resulting number falls within a segment allocated to a particular string. This string is subsequently chosen to enter the new population and is stored in the first record of *npop*. This cycle is repeated until the new population is complete. The algorithm to implement selection is depicted in figure 6.10.

```

Selection
  do for  $i = 1$  to popsize
    generate random number 1 to 1000 = rand
    rand = rand / 1000
    do for all j
      if rand  $\geq 0.5$ 
        if  $bol_j \geq rand$ 
          if  $bol_j \leq rand$ 
            npop $i$  = opop $j$ 
        else
          if  $bol_j \leq rand$ 
            if  $boh_j \geq rand$ 
              npop $i$  = opop $j$ 

```

Figure 6.10 Algorithm to Implement Selection

6.2.5 Action of Genetic Operators

Designs chosen to enter the new population are merely replicas of previously explored design points. The action of genetic operators aims to reach unexplored areas of the search space. In the GA this is typically carried out using crossover and mutation.

Crossover produces new individuals, which have some parts of both parent's genetic material. The GA uses the simplest form, single-point crossover. This mating process is carried out in the routine *Crossover*. Crossover considers every other string in the population. Consider string j . A random number between 0 and 1 is generated. If this number lies below the crossover rate the mating process proceeds and string j is mated with string $j + 1$ about a randomly chosen crossover point. Else both strings remain unchanged and consideration is given to string $j + 2$. As the new population is determined using a probabilistic selection method, mating pairs are in effect randomly chosen.

The algorithm to implement crossover uses a combination of shifting and Boolean logic to achieve its objective. Having deduced that string j will be crossed the subsequent process is illustrated in figure 6.11.

```

Crossover
  crosspoint = random number 1 to 32
  if crosspoint ≤ 16
    byte = 0
    goto Funccross with byte and crosspoint value
  else
    byte = 1
    crosspoint = crosspoint - 16
    goto Funccross with byte and crosspoint value

Funcross
  temp1 = chrom[byte]j+1
  temp1 <shift to right> UNITSIZE - crosspoint
  temp1 <shift to left> UNITSIZE - crosspoint

  temp2A = 0
  for i = 0 to (UNITSIZE - crosspoint - 1)
    Temp2A <shift to left> 1
    Temp2A += 1
  temp2 = temp2A <AND-logic> chrom[byte]j

  temp3 = chrom[byte]j
  temp3 <shift to right> UNITSIZE - crosspoint
  temp3 <shift to left> UNITSIZE - crosspoint

  temp4 = temp2A <AND-logic> chrom[byte]j+1

  chrom[byte]j = temp1 <OR-logic> temp2
  chrom[byte]j+1 = temp3 <OR-logic> temp4

```

Figure 6.11 Algorithm to Implement Crossover

Mutation is a random process where one allele of a gene is replaced by another to produce a new genetic structure. Using binary code, mutation flips the value of the bit at the loci selected to be the mutation point. In the GA, mutation is applied with uniform probability to the entire population of strings. It is possible, therefore, that a given binary string may be mutated at more than 1 point. Mutation occurs with a probability determined by the mutation rate, typically in the range 0.001 to 0.1.

Each string enters the routine *Mutation*. Each bit of the string is then considered in turn. If the random number generated dictates that the considered bit is to be mutated the string is sent to a subroutine termed *Funcmutate*. *Funcmutate* first determines the value assigned to the specific loci, i.e. 0 or 1. A mask is then used in conjunction with AND or OR logic, where the bit value is 0 and 1 respectively, and the bit is flipped. The algorithm to mutate a bit is given in figure 6.12, where *mutatepoint* specifies the particular bit under consideration. An example of *Funcmutate* applied to the 6th bit of a simplified string is given in figure 6.13.

```

Funcmutate
  let maskA = 65535
  let maskB = 1

  determine value of bit to be flipped
  mask1 = maskA <shift to right> mutatepoint
  flag1 = mask1 <OR-logic> chrom[byte]j
  mask2 = maskA <shift to right> (UNITSIZE - mutatepoint - 1)
  mask2 <shift to left> (UNITSIZE - mutatepoint + 1)
  flag2 = mask2 <OR-logic> flag1

  create mask
  maskB <shift to left> (UNITSIZE - mutatepoint)

  if flag2 = 65535
    chrom[byte]j = chrom[byte]j <AND-logic>  $\overline{\text{maskB}}$ 
  else
    chrom[byte]j = chrom[byte]j <OR-logic> maskB

```

Figure 6.12 Algorithm to Implement Mutation

Mutatepoint = 6 on Chrom[byte]_j = 11100110:

Step1) Determine value of bit to be flipped,

mask1 = maskA >> 6 = 00000011
flag1 = mask1 <OR> chrom[byte]_j
= 00000011 <OR> 11100110
= 11100111
mask2 = maskA >> 3 = 00011111
mask2 = maskA << 3 = 11111000
flag2 = mask2 <OR> flag1
= 11111000 <OR> 11100111
= 11111111

Step 2) Create a mask

maskB = maskB << 2 = 00000100

Step 3) Flip bit,

flag2 = 65535 so,
chrom[byte]_j = chrom[byte]_j <AND> $\overline{\text{maskB}}$
= 11100110 <AND> 11111011
= 11100010

Figure 6.13 Example to Demonstrate Funcmutate

6.2.6 Update Design Data

Mutation and crossover directly manipulate the binary code of each design. As a result the record of data concerning each design which has undergone crossover or mutation will be distorted. For each modified design the integer parameter set must be modified to correspond to the binary string. In addition, the string's fitness information must be updated.

A routine called *Updateichrom* extracts the specific integer values of a design's parameter set from its binary code. A shifting process in conjunction with OR logic

is used. Figure 6.14 demonstrates the first part of the algorithm to implement *Updateichrom*, depicting extraction from bits 1 to 16 of the binary string.

```

Updateichrom
extract parameters from byte 0
temp = chrom[0]j <AND-logic> 0XFE00
ichrom.θ1 = temp <shift to right> 9
temp = chrom[0]j <AND-logic> 0X01FC
ichrom.θ2 = temp <shift to right> 2
temp = chrom[0]j <AND-logic> 0X0002
ichrom.V = temp <shift to right> 1
temp = chrom[0]j <AND-logic> 0X0001
ichrom.P = temp

```

Figure 6.14 Algorithm to Extract a String's Integer Parameter Set

Genetic operator action can modify a parameter value is within its feasible binary range, as dictated by the number of bits allocated to the parameter. Crossover or mutation may produce an infeasible design. Either a parameter value may be altered such that it lies outside its designated range or interactions between parameters may render the design infeasible.

Having extracted the parameter set, designs are checked for feasibility. The checks made and subsequent corrective action taken is carried out to:

- Ensure neither N_1 or N_2 exceed 4. If so, reduce the respective value to 4.
- Ensure neither N_1 or K_1 equal 0. If so, set to 1.
- Ensure neither E or H equal 3. If so, set to 2.
- Check subsystem 2 feasibility:
 - If $H = 0$, ensure N_2 , K_2 and θ_2 are 0.
 - If $H \neq 0$, ensure N_2 , K_2 and θ_2 are not 0. If so set the respective value to 1.
- Ensure K_1 is less than or equal to N_1 . If not, set $K_1 = N_1$.
- Ensure K_2 is less than or equal to N_2 . If not, set $K_2 = N_2$.

A check marker is flagged if a design requires any feasibility modifications. These designs then enter a routine to rectify their binary string (implemented in a similar manner to *Binconversion*).

6.2.7 Update Fitness Data

The next step updates the fitness information of the modified designs. Each crossed or mutated string enters the fitness function. The probability of system unavailability, the spurious trip frequency, MDT, cost, respective penalties and associated penalised system unavailability are re-evaluated.

A new population consisting of both explored and unexplored points has been formed. The population is analysed using *Genaverage* and *Genbest*, calculating the population's average penalised system unavailability and determining the fittest design. The best individual is then compared with that in the *Bestever* structure. If the individual in the current population proves fitter it replaces the best design up to that point. As such, the best design obtained throughout the GA run is recorded.

6.2.8 Check For Termination

The new population, *npop*, is transferred to storage within *opop*. The generation number is increased by 1. If the maximum number of generations is reached the loop is terminated and the GA run completed. Else, the next iteration commences with the selection process carried out on the *opop* structure .

6.2.9 Output from GASSOP

GASSOP writes data associated with each generation to an output file. Each generation provides the following information:

- 1) The parameter set of each individual expressed in integer from at the start of the iteration number.
- 2) A table of raw fitness values: MDT, cost, spurious trip and system unavailability of each design.
- 3) Each design's penalised system unavailability with a breakdown of penalty contributions.

- 4) The reproduction probability assigned to each string.
- 5) The population of designs chosen during selection alongside their binary code.
- 6) Action carried out in crossover and mutation, i.e. those strings which have undergone either crossover or mutation, expressed in binary, and verification of the crossover and mutation points.
- 7) An updated new population in integer form, plus corresponding fitness information

This data file is compiled throughout the GA run. Additionally, on completion of the program run a summary of results is written to an output file called *Bestdata*. This summarises the GA run and states

- 1) A table specifying the average penalised system unavailability of each population.
- 2) A table specifying the fitness of the best design in each population.
- 3) A description of the best design obtained throughout the GA run, i.e. the parameter set and associated fitness values.

6.2.10 Results from a Run of GASSOP

GASSOP was implemented with a population of 20 strings. A maximum of 100 generations was allowed along with a mutation rate of 0.01 and crossover rate 0.7. In total, therefore, 1000 system evaluations were performed in determining the best design. The running time of the program was an order of minutes.

The average and best string fitness were two dominant factors considered in each generation. These results are represented graphically in figure 6.14.

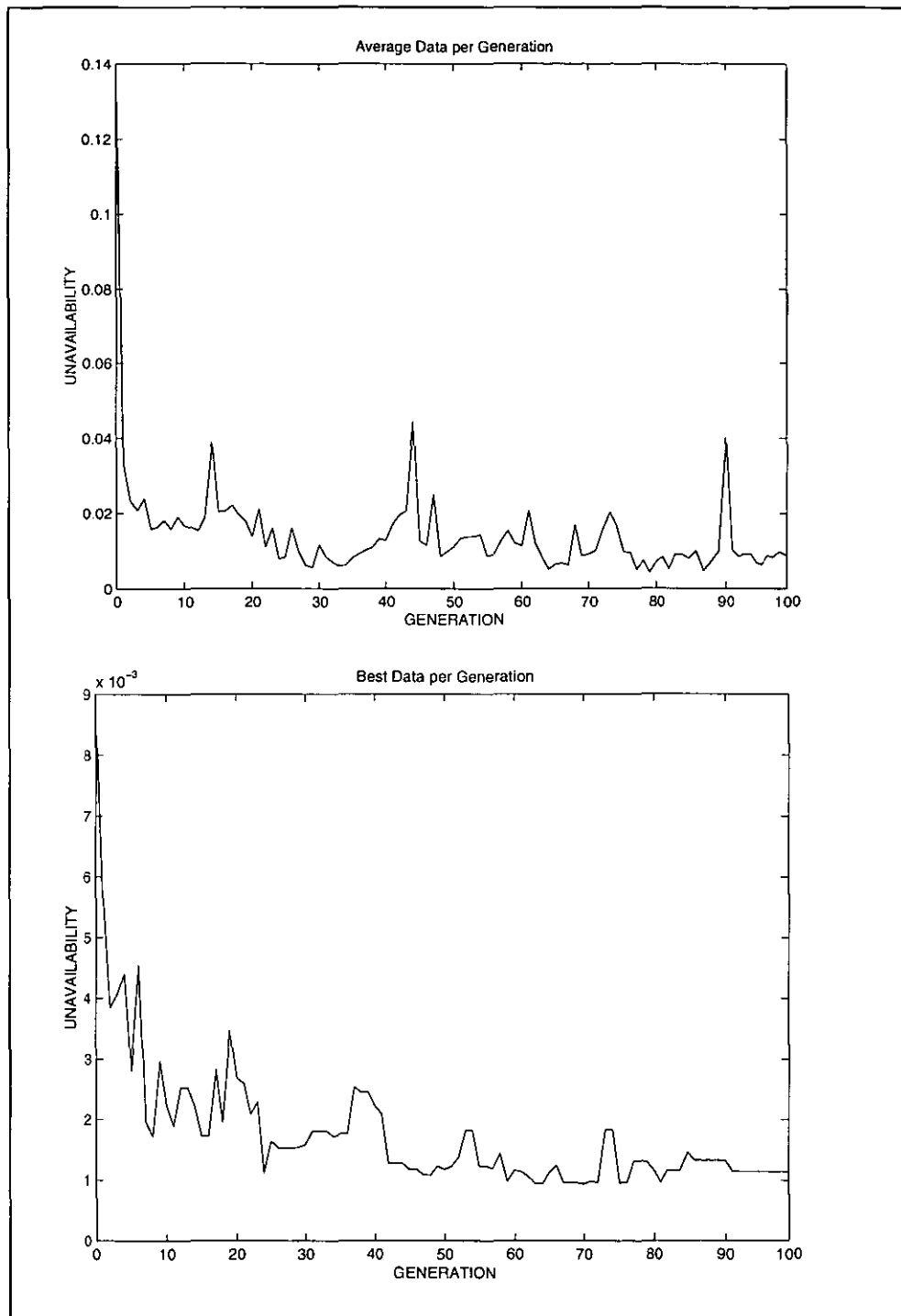


Figure 6.14 Average and Best String Data

The initial population produced a set of strings with fitness values ranging between 0.699 and 0.0087. Over successive generations the action of the GA successfully produced string fitness convergence.

A dramatic improvement in average population fitness was portrayed in the first 30 generations. Generation 30 has fitness values in the range 0.144 to 0.0017. Generation 50 onwards represents dominance by highly fit strings.

The fittest string from the entire process arose in generation 70. The characteristics of this design are specified in table 6.2. The design does not exceed any design limitations and as such is not penalised, i.e. $Q_{SYS} = Q'_{SYS}$.

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

Table 6.2 Characteristics of the Best Design

Generation 91 onwards gives rise to a slightly less fit best string with a fitness value of 0.001141. This design differs from that above in that it requires 3 pressure transmitters to trip subsystem 1 and has test interval parameters of 46 and 26 for subsystems 1 and 2 resulting in a MDT of 114.35.

6.3 GA Parameters

The GA requires the following selection parameters to be set:

- population size,
- crossover rate,
- mutation rate,
- number of generations.

Values entered for these parameters have a marked affect on the action of the GA (see section 4.3.7). Using GASSOP as previously described, an analysis was carried out to investigate the effect of changing these parameter values. De Jong, amongst others, has researched this area with a test bed of five trial functions, and two criteria of goodness defined (Ref. Goldberg). Regarding his concluding discussion a limited set of values for each parameter was chosen:

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

In total 36 runs of the GA were carried out, thus ensuring each possible combination of the values above was analysed. The penalised system unavailability of the best overall string per run was then investigated for each parameter set.

To obtain an indication of the effect of setting each parameter to a particular value the best penalised system unavailability obtained for all of the other parameter values were summed and averaged. A summary of these results are given in table 6.3, for the mutation rate, crossover rate and population size respectively.

It is interesting to note that as the population size increases the beneficial effect of a larger mutation and crossover rate is diminished. A possible deduction from this is that the mutation and crossover rates do not have a significant effect if the population size is sufficiently large. A population of only 10 strings may not incorporate enough diversity from the onset and for this reason a high degree of mutation moves to areas in the search space, which would otherwise not be explored.

In conclusion a balance in the diversity, and processing time introduced into the GA has to be established. For further analysis of the SGA it was decided to use a population size as high as is appropriate, a crossover rate of 0.7 and mutation rate 0.01.

6.4 Further Testing

To test the GA method further 10 runs with a population of 20 strings over 100 generations were carried out. The mutation and crossover rate for each run was entered as 0.01 and 0.7 respectively.

The GA portrays a significant convergence in average population fitness. This is demonstrated in table 6.4, which shows the population average fitness value of the first and last generation for each run. Typically, the greatest convergence occurs in the first 30 generations. Figure 6.15 demonstrates the average data per generation for runs 3, 4 and 8. The latter part of each run shows fluctuation about a low average population fitness.

GA Run No.	Initial Population Fitness Average	Final Population Fitness Average
1	0.1729	0.0305
2	0.1391	0.0087
3	0.0928	0.0069
4	0.1241	0.0101
5	0.1184	0.0242
6	0.0863	0.0184
7	0.0673	0.0093
8	0.1155	0.0188
9	0.1345	0.0190
10	0.0997	0.0113
Average fitness		
Σ	0.1151	0.0157

Table 6.4 To Demonstrate Population Average Fitness Convergence

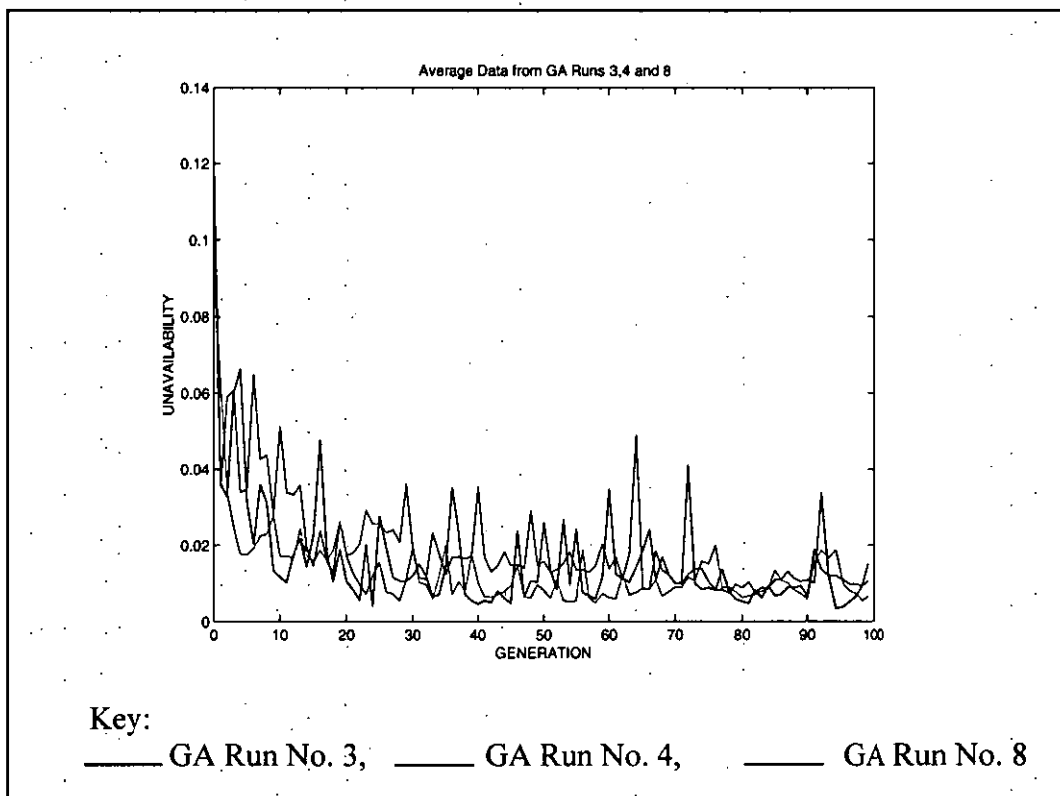


Figure 6.15 Average Data From GA Runs 3,4 and 8

Table 6.5 shows the characteristics of the best design resulting from each run and table 6.6 the corresponding fitness data. These designs vary quite markedly. Each design is, however, particularly fit.

GA Run	E	K_1/N_1	H	K_2/N_2	V	P	θ_1	θ_2	Q_{sys}
1	0	3/4	2	2/4	1	1	51	37	1.56e-3
2	0	2/4	2	1/1	2	1	45	22	9.36e-4
3	0	1/2	3	1/1	2	1	33	37	1.14e-4
4	0	2/3	2	1/1	1	1	42	28	1.13e-3
5	0	2/4	2	3/3	2	1	31	35	1.26e-3
6	0	2/4	2	2/4	1	1	38	37	1.13e-3
7	0	1/1	2	1/3	2	1	45	23	9.71e-4
8	0	1/1	2	1/1	2	1	28	29	8.31e-4
9	0	1/2	1	4/4	1	1	26	46	2.34e-3
10	0	1/2	2	1/1	2	1	36	28	9.44e-4

Table 6.5 Characteristics of the Best Designs from Each Run

GA Run	Cost	MDT	Spurious Trip	Q_{sys}	Q'_{sys}
1	982	107.85	0.29	1.56e-3	1.56e-3
2	822	127.74	0.85	9.36e-4	9.36e-4
3	1000	130.32	0.94	1.14e-4	1.14e-4
4	902	131.24	0.42	1.13e-3	1.23e-3
5	862	122.26	0.72	1.26e-3	1.26e-3
6	982	122.16	0.55	1.13e-3	1.13e-3
7	802	125.30	0.97	9.71e-4	9.71e-4
8	762	131.54	0.72	8.31e-4	1.03e-3
9	672	108.52	0.54	2.34e-3	2.34e-3
10	782	119.48	0.85	9.44e-4	9.44e-4

Table 6.6 Performance Measures of the Best Designs

The final population resulting from each run constitutes fit, yet somewhat varied designs.

6.4.1 Discussion of Results

Convergence to a fit design through the GA is not necessarily smooth. The potential to introduce an unfit gene at any point arises due to the inherent random nature of the GA. As a result sudden fluctuations in average population fitness are to be expected. Conversely, a particularly fit string may be produced in an early generation due to this random nature.

It is, however, anticipated that the GA will show significant convergence to a particular design. In consequence, the final population will be dominated by a design similar, if not identical, to the best overall string. In addition, the fittest designs from one run to the next will show very little structural variance.

GASSOP is achieving its objective to find fit domains in the search space. The GA, however, lacks the ability to 'home in' on the fittest designs in a highly fit population. For this reason significant variety is retained in the latter generations. There exists, therefore, the potential for significant fluctuation regarding design variations within a certain fitness band and this is further encouraged by genetic operator action.

In addition, the maintenance effort allocated to a particular design is a determining factor in the resulting system unavailability. A particularly fit design may be investigated with a pair of test intervals, which do not enable good use of the MDT resource. Consequently, a design that has less potential but portrays a low system unavailability by virtue of its maintenance effort may be preferred to a potentially fitter design with a relatively poor MDT allocation. These issues are addressed in the next chapter.

CHAPTER 7

MODIFICATIONS TO THE GA

7.1 Introduction

The focus of this chapter is to investigate potential improvements to the simple GA introduced in chapter 6, such that the GA is more efficient and effective in obtaining highly fit solution strings. Having implemented the GA to perform the HIPS optimisation, areas of improvement became apparent. . The only information directly used by the GA regarding each design is the reproduction probability allocated to the string and used within the selection process. It must therefore be ensured that the reproduction probability calculated for each string accurately reflects the string's fitness. An improvement in the accuracy of the string's reproduction probability will improve the overall performance of the GA.

The system unavailability plus any penalty due to violation of cost, MDT or spurious trip frequency, should any of these constraints be violated, constitute the string's fitness. The fitness value is then converted to its corresponding reproduction probability. This process is depicted in figure 7.1.

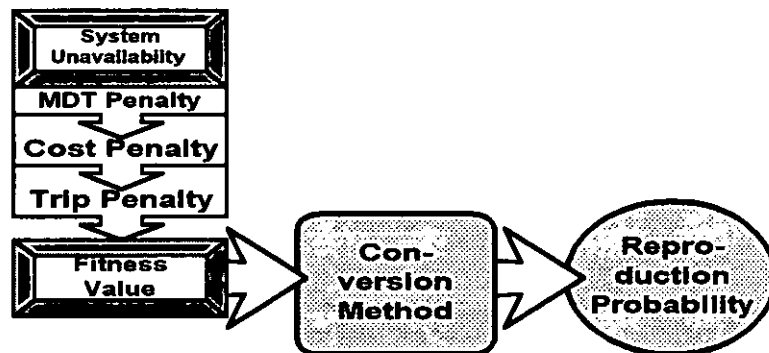


Figure 7.1 Factors Contributing to the Reproduction Probability

All these factors are of great importance. An inaccurate calculation of any one will result in a reproduction probability that does not precisely reflect the fitness of the string. This chapter considers each aspect in more detail.

7.2 Utilisation of the MDT Resource

The Maintenance test interval (MTI) parameters for each subsystem directly affect the system unavailability of a design. They also contribute to the calculation of a system's maintenance down time, which exerts a penalty on the system unavailability if 130 hours is exceeded.

Fully utilising the 130 hour MDT resource and splitting the available effort between the two subsystems to the best advantage will result in the most optimal system unavailability. It was noted that very few designs approached this limit and, hence, are not fully utilising their available resources.

An extension of the quantitative analysis described in section 6.3 was carried out to establish the effect of the GA parameters, i.e. population size, crossover and mutation rate, on the ability of the GA to result in a design which tends toward its MDT limit. The analysis was undertaken in a similar manner to that previously described. The value of interest being in this case the MDT of the best system design generated. A summary of the average MDT of the best designs for each GA parameter combination is shown in table 7.1.

		MUTATION RATE		
		M1 = 0.1	M2 = 0.01	M3 = 0.001
AVERAGE MDT		122.8	122.2	115.4

		POPULATION SIZE		
		P1 = 10	P2 = 20	P3 = 50
AVERAGE MDT		118.9	120.4	124.9

		CROSSOVER RATE			
		C1 = 0.5	C2 = 0.6	C3 = 0.7	C4 = 0.8
AVERAGE MDT		116.5	116.5	121.6	124

Table 7.1. A Summary of the Average MDT Values

A higher mutation rate, crossover rate and larger population size generate greater diversity in each population. The analysis shows that the greater the variation in each population, the more likely the MDT is to converge to its limit.

Each parameter fixed at its largest value leads to the most optimum MDT. However, the data implies that the crossover rate has the greatest individual effect.

The MDT is governed by the values assigned to the MTI values for subsystem 1 and 2. These parameters span a much greater range of values than the other parameters and hence occupy a greater proportion of the string.

The test parameters have a much larger area of the search space to explore. This knowledge supports the evidence from our quantitative analysis, implying a need for greater variation within this area of the string.

Three methods to improve the exploration of the range of values allowed by the MTI parameters have been explored.

7.2.1 MDT Modification Method 1

In method 1 the GA executes in the normal manner and a best overall string is deduced. This best system design is then sent to an additional routine. Within this routine a loop checks the system unavailability for each feasible combination of test intervals for subsystems 1 and 2. Any non-feasible combination, i.e. a MDT exceeding 130 hours, is ignored. The combination of maintenance test intervals resulting in the most optimal system performance is retained. The algorithm to achieve this is shown in figure 7.2.

```

set  $Q_{STORE} = 1.0$ 
do for  $1 \leq \theta_{BEST}^1 \leq 104$ 
  do for  $1 \leq \theta_{BEST}^2 \leq 104$ 
    evaluate MDT using best design
    if  $MDT < 130$ 
      evaluate  $Q_{SYS}$ 
      if  $Q_{BEST} < Q_{STORE}$ 
         $\theta_{STORE}^1 = \theta_{BEST}^1$ 
         $\theta_{STORE}^2 = \theta_{BEST}^2$ 
         $MDT_{STORE} = MDT_{BEST}$ 
         $Q_{STORE} = Q_{BEST}$ 

 $\theta_{BEST}^1 = \theta_{STORE}^1$ 
 $\theta_{BEST}^2 = \theta_{STORE}^2$ 
 $MDT_{BEST} = MDT_{STORE}$ 
 $Q_{BEST} = Q_{STORE}$ 

```

Figure 7.2 Algorithm to Carry Out MDT Modification Method 1

7.2.2 MDT Modification Method 2

Method 2 introduces greater search into the MTI area of the string, without affecting the rest of the design. This method uses a specialised mutation operator.

The specialised mutation approach involves the inversion of a segment of the string. It must be noted that this is not equivalent to action carried out by the inversion operator. During inversion the segment reversal switches the position of genes on the string but does not alter their values, i.e. inversion retains the value of design parameters on the string (see section 4.2.6.4). To describe the modified mutation operator consider the string, S,

$$S = 10001111101110$$

Choose $a = 2$ and $b = 7$ say,

$$S' = 10111001101110$$

The string, S' , is the original string with the points 2 through to 7 inverted.

Crossover and traditional binary mutation is carried out on the latter part of each string pair only, i.e. between bits 15 and 32 (parameters V through to H , see figure 6.3). Consequently, following the action of the two genetic operators the MTI area of the string is unaffected. Each new string is then further processed. The string is first decoded to obtain the MTI values for each subsystem and stored as integer values in the normal manner. The MTI area of each string is subsequently inverted between two randomly generated points a and b , where $0 \leq a < b \leq 14$. The range of these cut points is limited if the design under consideration has no redundant subsystem, i.e. $0 \leq a < b \leq 7$, as $\theta_2 = 0$. Each end of the MTI section is isolated into its respective parts, i.e. 0 through to a , a to b , and b to 14. The middle section to be inverted is termed *invsect*. To invert *invsect* each separate digit is placed in an array, termed *sect*, and the section rebuilt in reverse. The algorithm to achieve this is illustrated in figure 7.3.

Once reversed *invsect* is re-instated in its original position in the MTI section, sandwiched between the end segments. Following the inversion process the MTI values are extracted and stored alongside the modified binary section in a separate structure, within the record of the considered individual.

```

l = b - a + 1

set leftshift = 0
do for i = 0 to (l - 1)
    leftshift = leftshift + 2i

set startshift = 0
set digit = 0
set rightshift = b - a
do for loop = startshift to 1
    tempsect = leftshift<AND-logic> invsect
    tempsect<shift to right> rightshift
    sect[digit] = tempsect
    tempsect = 0
    leftshift = leftshift - 2<to the power>rightshift
    digit = digit + 1

set invsect = 0
do for loop = (l-1) to 0
    Sect[loop]<shift to left>loop
    Invsect<OR-logic>sect[loop]

```

Figure 7.3 Algorithm to Implement the Inversion Process

Four potential MTI parameter combinations for the 2 subsystems now exist:

$$[\theta_1, \theta_2] [\theta'_1, \theta_2] [\theta_1, \theta'_2] [\theta'_1, \theta'_2]$$

where θ' denotes the mutated test interval values. The resulting combinations are considered in one of two ways:

Method 2a evaluates the system unavailability in addition to the MDT for each MTI combination. Should any pair of test intervals exceed 130 hours MDT the respective penalty is added. The test intervals resulting in the lowest penalised system unavailability are retained.

Method 2b evaluates the MDT associated with each MTI combination using the MDT formula. The pair of values whose MDT is closest to, but not greater than, 130 hours is retained. If all resulting MDT values are greater than 130 hours the test intervals

associated with the lowest value are kept. In this case the MDT penalty is enforced as in the original GA.

7.2.3 MDT Modification Method 3

Method 3 is a mixture of methods 1 and 2. Crossover and mutation is carried out in the latter part of each string only, prior to being processed in the MTI area. Each string enters a loop, as with method 1, that checks the system unavailability of the design for every feasible combination of test parameters in the range. The pair of test parameters resulting in the best system performance for each system is retained. The population of strings, therefore, enters the next generation with each design fully utilising all the available MDT resources.

The MTI parameters are established in the same manner for the initial population, thus differing from the original GA. The structural system design variables are randomly initialised, the test parameters are then set to the optimal value for the particular design.

Method 3 is computationally intensive. Consideration of each string in each population requires an extensive amount of MDT and system unavailability evaluations, where the latter aspect is of primary concern. A means to combat this is to assume that MDT values below a certain criteria will not result in an optimal system unavailability. As such, the algorithm is modified so that only those test interval combinations resulting in a MDT between 120 and 130 are sent to the fitness function for system unavailability evaluations. MDT values outside this band are ignored. This could be taken one step further in that the MDT of all MTI combinations is evaluated and the test intervals whose MDT is closest to 130 hours retained. This combination is then used to evaluate the system unavailability and as such only one system unavailability evaluation per string is required.

7.2.4 Results of the MDT Modification Methods

The GA with MDT modification methods 1, 2a, 2b and 3 were tested 10 times. Each run 20, 0.7 and 0.01 were entered for the population size, crossover rate and mutation rate respectively. Each run was terminated after 100 generations. Table 7.2 shows the original MDT and corresponding fitness for the best designs resulting from the original GA. Stated alongside are the MDT and fitness values for each best design using the modified methods. The average fitness values arising from each method are also given. (Method 3 was tested with a full analysis of every test interval combination. A lower limit of 120 hours MDT was subsequently placed on the designs requiring system unavailability evaluations. The results proved identical for each approach.)

Original SGA		MDT modification Method 1		MDT modification				MDT modification Method 3	
				Method 2a		Method 2b			
MDT	FIT	MDT	FIT	MDT	FIT	MDT	FT	MDT	FT
107.85	1.56e-3	129.82	1.12e-2	128.26	8.65e-4	122	9.26e-4	129.78	8.36e-4
127.74	9.36e-4	129.98	1.05e-2	123.39	9.81e-4	127.94	7.88e-4	129.56	7.81e-4
130.32	1.14e-3	129.75	1.09e-2	127.06	8.91e-4	128.37	1.05e-3	129.89	7.80e-4
131.24	1.23e-3	129.77	8.36e-4	127.27	8.17e-4	122.02	9.44e-4	129.65	7.24e-4
122.26	1.26e-3	129.56	7.84e-4	129.46	7.87e-4	129.63	8.14e-4	129.45	7.99e-4
122.16	1.13e-3	129.63	8.06e-4	127.94	7.86e-4	126.75	8.43e-4	129.89	7.81e-4
125.30	9.71e-4	129.57	9.74e-4	128.77	8.65e-4	128.77	8.65e-4	129.72	9.35e-4
131.54	1.03e-3	129.89	1.08e-3	128.32	1.07e-3	128.59	8.61e-4	129.89	7.81e-4
108.52	2.34e-3	129.54	1.04e-3	123.5	1.07e-3	114.4	1.31e-3	129.88	8.56e-4
119.48	9.44e-4	129.87	9.91e-4	129.63	8.04e-4	129.63	8.04e-4	129.72	7.59e-4
Average value from each column									
122.64	1.25e-3	129.74	9.78e-4	127.36	8.94e-4	125.82	9.20e-4	129.74	8.03e-4

Table 7.2 Results from the Original GA and the MDT Modification Methods

The routine used to check the system unavailability of the resulting design for each feasible combination of test intervals (introduced in section 7.2.1) was incorporated into both methods 2a and 2b. This enables the resulting best designs to fully utilise available resources and hence, enhance their performance. These updated designs are given in table 7.3.

Method 2a combined with Method 1		Method 2b combined with Method 1	
MDT	Fitness	MDT	Fitness
130	8.37e-4	129.56	7.82e-4
129.75	8.78e-4	129.61	7.59e-4
129.63	8.07e-4	129.18	1.04e-3
129.56	7.84e-4	129.63	8.07e-4
129.89	7.81e-4	129.72	7.62e-4
129.61	7.56e-4	129.63	8.07e-4
129.77	8.36e-4	129.77	8.36e-4
130	1.04e-3	130	8.38e-4
129.72	9.65e-4	129.84	1.01e-3
129.72	8.04e-4	129.63	8.04e-4
Average value form each column			
129.75	8.49e-4	129.66	8.45e-4

Table 7.3 Method 2 Combined with Method 1

The average string fitness and best string fitness in each generation for the fittest string resulting from each modified method (i.e. those underlined in table 2) are shown in figure 7.4. The characteristics of each of these designs are given in table 7.4.

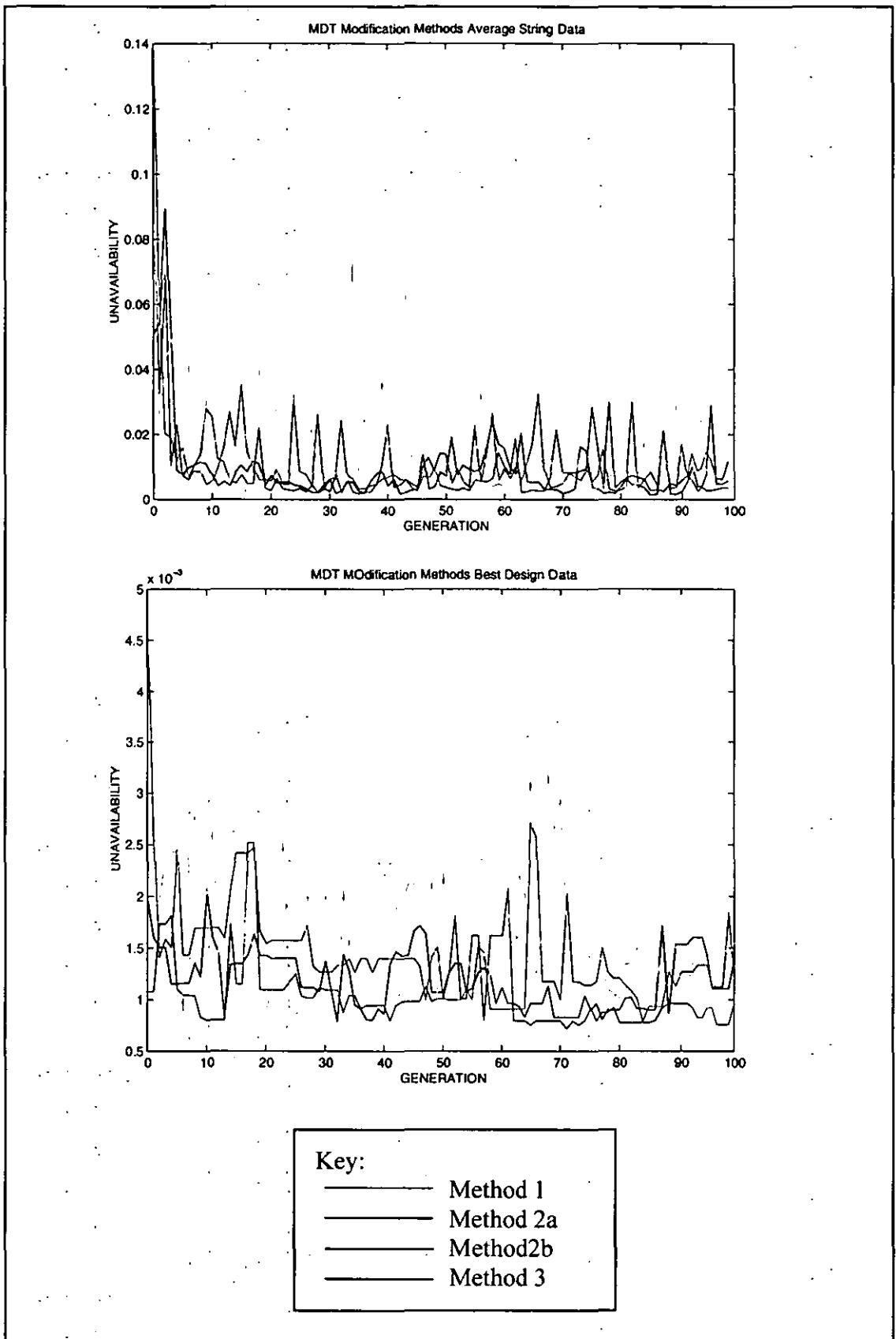


Figure 7.5 Average and Best Data for the Fittest Designs per Generation

	E	K_1/N_1	H	K_2/N_2	V	P	θ_1	θ_2
Method 2a	0	2/4	2	1/2	2	1	30	32
Method 2b	0	3/4	2	1/2	2	1	30	32
Method3	0	1/2	2	1/2	1	0	36	29

Table 7.4 The Best Design Characteristics Resulting From Each Method

7.2.5 Discussion of the MDT Modification Results

Each method improved the utilisation of MDT resources, thus improving the performance of the safety system design.

The first method does not affect the GA process directly. The MDT is considered after as opposed to during the design stage, as is typical of most engineering disciplines. Such an approach, however, neglects to incorporate the MTI parameters in the design stage and thus, is unable to positively influence decisions such as, which valve or pressure transmitter to choose.

Method 2 achieves the desired intention to introduce greater variety into the test interval parameter area of the string. Full use of the MDT resource is not guaranteed but the general case results in convergence towards the limit. Application of method 1 to the final design ensures full use of all available MDT resources. The advantage of the second method is that it is incorporated within the design process and does not require an excessive amount of extra system evaluations.

Method 2a puts a greater demand on system evaluations than method 2b, but achieves a more thorough analysis of the test interval combinations. Method 2a considers both the system MDT and the effectiveness of its distribution between subsystems.

Method 2b considers the former aspect alone. The results for each are very similar, however, there is a slight bias in preference toward method 2a, particularly regarding MDT usage.

Method 3 is in effect an exhaustive search of every feasible test interval combination for each subsystem. It ensures that all MDT resources are used at all times and are distributed between both subsystems to the best advantage. As such, potentially fit designs are not eliminated due to inferior MTI values. This method, however, relies on a very large amount of system unavailability evaluations and may be considered impractical for larger systems. Analysing the system unavailability of only those test interval combinations within a specific range of the MDT limit, effectively reduces computer effort. Adapting the exhaustive approach to calculate the MDT values only, thus incurring no extra demand on system unavailability evaluations is an alternative method with much less demands on processing time. This distribution of test interval values between each subsystem may not, in this case, be optimal.

For each MDT modification approach the GA portrays convergence to a fit population. The convergence rate is similar to that of the original GA, however, a slightly fitter population is generally attained in methods 2 and 3. Fluctuation occurs as a result of the introduction of novel design points via mutation and crossover. If these points prove ineffective they are eliminated from subsequent generations.

Methods 2 and 3 enable a more thorough exploration of the MTI parameter values. As such, the best designs constituting the latter generations generally exhibit fitter performance measures. The best design from one generation to the next portrays significant variety. The GA still demonstrates an inability to differentiate the fittest strings from a highly fit population. The graphs show fluctuation within a highly fit band of designs, however, a particular design is not maintained.

In conclusion, method 3 achieves the best results for the HIPS system. It is probable that method 2a or the less computer intensive adaptation of method 3 would be more practical for larger systems. These explore extra system evaluations but not to excess.

7.3 Derivation of a Modified Cost Penalty Formulae

If the performance of a design is significantly improved due to a comparatively small excess in one or more of the constraints, the design in question deserves further consideration. An excessive abuse of the limits with only a small degree of performance improvement conversely implies the design be discarded. It is essential that an appropriate penalty be exerted to the system unavailability of a design should it exceed certain limits.

The MDT modification methods go some way to alleviating the importance of the MDT penalty formula. A spurious trip results in loss of production of the industrial system and hence, loss of profit. For this reason a spurious trip is expressed in terms of excess cost and the cost penalty formula used. The penalty formula under consideration is, therefore, that regarding cost.

The cost penalty in the original GA is derived from the formula:

$$COST\ PENALTY = (EXCESS\ COST/100)^{5/4} \times 0.002 \quad (7.1)$$

A base level in system performance is assumed, i.e. a system unavailability of 0.02. This was based on expectation of improved performance, which is proportional to increase in cost, as described in section 6.2.3.1. To illustrate this, consider the example in figure 7.6.

The comparative penalty for the fitter string is much greater. The penalty should take the fitness value of the system to be penalised into consideration.

An alternative cost penalty formula is introduced which takes into account both the cost violation and the system unavailability of the design being penalised. This is achieved using a multiplying factor which, rather than being fixed, varies according to the system unavailability of the design.

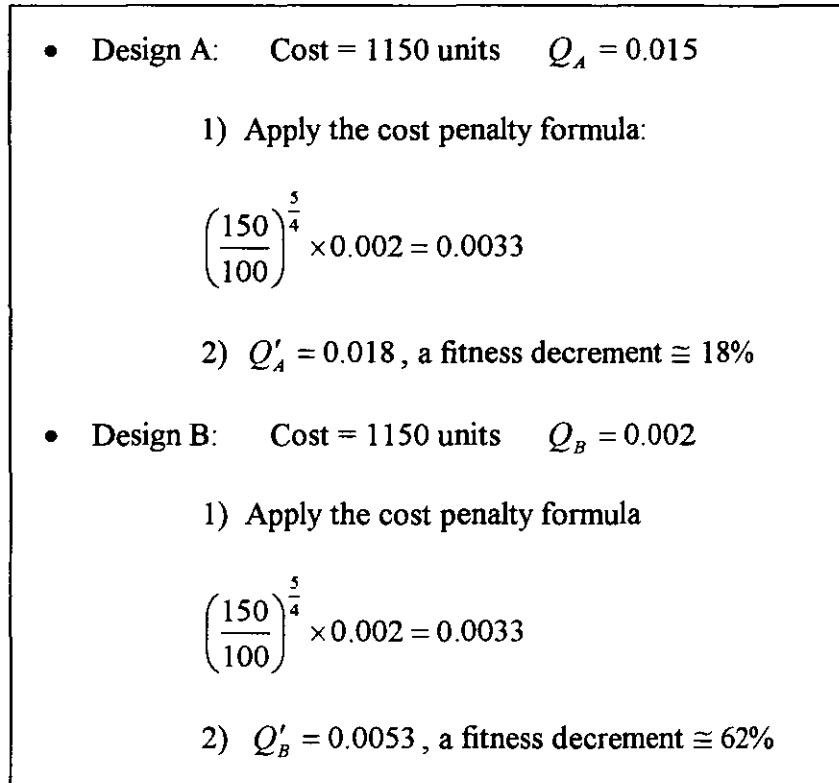


Figure 7.6 Two Example Designs

Consider a particular design with cost C . The cost, C , exceeds 1000 units. The percentage excess of the system's cost is calculated, X^C say. The multiplying factor is derived by calculating X^C percent of the system unavailability of the design under consideration. In the designs A and B in figure 6, both designs exceed 1000 units by 15 percent, i.e. $X^C = 15\%$. The system unavailability of design A is 0.015, of which 15% is 0.0025. The multiplying factor to be used in the penalty formula for this design is, therefore 0.0025. Applying the penalty formula;

$$COST\ PENALTY = (150/100)^{5/4} \times 0.0025 = 0.0041 \quad (7.2)$$

The penalised fitness value is 0.0191, a 22 percent decrement. The system unavailability of design B is 0.002, of which 15% is 0.0003. Hence, applying the cost penalty;

$$COST\ PENALTY = (150/100)^{5/4} \times 0.0003 = 0.00049 \quad (7.3)$$

The penalised fitness value is 0.0025, a 20 percent decrement.

7.3.1 Results of the GA Using the Modified Cost Penalty

The GA was tested with ten runs using the modified cost penalty formula. For each run 20, 0.7, 0.01 and 100 were entered for the population size, crossover rate, mutation rate and number of generations respectively.

Table 7.5 shows the total cost, spurious trip and corresponding fitness value for each run of the original GA. Stated alongside is the equivalent information regarding the modified cost penalty method for the best design.

The average fitness value shows a small improvement using the modified cost penalty formula. It is not appropriate to compare the average cost of the best systems. Designs that cost below 1000 units are not affected by the penalty formula. The modified method should not encourage convergence towards 1000 units. The important information is that highly fit designs that slightly overstep the cost constraint but show significant performance improvement are tolerated.

Original SGA			Modified cost penalty formula		
Cost	Trip rate	Fitness	Cost	Trip rate	fitness
982	0.29	1.56e-3	1062	0.55	1.39e-3
822	0.85	9.36e-4	782	0.85	1.08e-3
1002	0.94	1.14e-3	1022	0.68	1.60e-3
902	0.42	1.23e-3	652	0.93	1.14e-3
862	0.72	1.26e-3	982	0.81	1.16e-3
982	0.55	1.13e-3	882	0.98	1.04e-3
802	0.98	9.71e-4	712	0.80	1.29e-3
762	0.72	1.03e03	882	0.42	1.38e-3
672	0.54	2.34e-3	882	0.45	8.27e-4
782	0.85	9.44e-4	1023	0.58	1.04e-3
Average Fitness original		1.25e-3	Average Fitness modified		9.69e-4

Table 7.5 A comparison of the original and modified cost penalty formula

7.3.2 Discussion of the Modified Cost Penalty

The GA generates populations with successively fitter strings. Quite frequently this pattern continues up to a point until a very fit string oversteps a boundary by a very small amount. By virtue of the penalty formula the system's fitness value becomes substantially less fit. The string's chance of re-selection is virtually impossible and thus, is weeded out of the population. The modified penalty formula ensures that the penalty applied to a string is in accordance with the string's system unavailability.

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

7.4 The Conversion Method

Each string receives a measure of its fitness in the environment. This is the string's penalized system unavailability. As described in the previous section, this value is not in an appropriate form to be used in the selection process of the GA.

A specialized conversion method is required. It is imperative that the conversion method gives rise to reproduction probabilities in accordance with the fitness value of each string. The distribution of percentages should accurately reflect the distribution of fitness values. A system whose performance is twice as good as another should have a percentage allocation double that of the other design. This shall be referred to as the proportional representation of a population's fitness values following conversion. The original 3-category conversion method is described in section 6.2.4.1.

7.4.1 Application of the original method

Two populations from a run of the GA were chosen to portray the effectiveness of the conversion method, as shown in table 7.6. (Treatment of the same population using the modified conversion method described in the next section is shown alongside). The first population considered is the initial population in the run. This population covers a wide ranging set of fitness values. The second population is from a later generation that has benefited from the action of the GA. The designs are highly fit and a comparatively small area of the fitness domain covered.

First Population			Second Population		
Fitness Value	Original RW%	Modified RW%	Fitness Value	Original RW%	Modified RW%
9.7e-3	11.5	14.2	8.14e-4	5.08	7.23
1.2e-2	11.2	13.7	8.17e-4	5.08	7.20
1.42e-2	10.9	13.1	8.31e-4	5.08	7.07
2.26e-2	9.9	10.7	8.42e-4	5.08	6.96
3.41e-2	8.4	7.6	9.29e-4	5.08	6.13
3.45e-2	8.3	7.5	1.20e-3	5.07	5.74
3.67e-2	8.1	6.9	1.33e-3	5.07	5.47
3.9e-2	7.8	6.2	1.34e-3	5.07	5.45
5.17e-2	6.1	5.7	1.34e-3	5.07	5.45
5.62e-2	5.6	5.2	1.34e-3	5.07	5.45
6.79e-2	4.1	3.9	1.37e-3	5.07	5.38
8.23e-2	2.3	2.4	1.37e-3	5.06	5.38
9.29e-2	0.1	1.3	1.38e-3	5.06	5.37
0.98e-2	0.003	0.05	1.38e-3	5.06	5.37
0.118	0.3	0.05	2.07e-3	5.05	4.46
0.123	0.2	0.04	2.47e-3	5.04	3.85
0.355	0	0	3.03e-3	5.02	3.13
0.389	0	0	3.13e-3	5.02	2.95
0.421	0	0	5.51e-3	4.96	1.33
0.422	0	0	4.63e-2	3.92	0.59

Table 7.6 Two Example Populations

The conversion method is rather crude. Its effectiveness is dependent on the range and distribution of fitness values being converted. A population of values ranging uniformly between, for example 0.009 and 0.4, produces a set of roulette wheel percentages with a fairly accurate proportional representation. A much fitter population, with most systems having a fitness value less than 0.01, gravitates toward a set of almost equal percentages, although the actual fitness values vary quite

markedly. The tendency is that the conversion method is insensitive to small yet significant differences between the fitness values of fitter strings.

In addition problems occur when either a very high, or very low, proportion of strings fall into a particular category. The percentage allocated to each category is fixed and, therefore, independent of the number of strings it contains. Note the discrepancy in the allocation to the less fit strings of the first population.

7.4.2 A modified conversion method

An alternative method is required which is able to cope with very diverse populations and is simultaneously able to show sensitivity to a highly fit set of strings.

Initially nine categories are depicted which cover the area of the fitness domain of importance, i.e. below 0.2, and each category is assigned a particular weight, as shown in table 7.7. As the category gets fitter, its weight increases in size.

Fitness domain					Weight
Upper limit		Category		Lower limit	
0.2	>	9	≥	0.1	1 ²
0.1	>	8	≥	0.05	2 ²
0.05	>	7	≥	0.01	3 ²
0.01	>	6	≥	0.005	4 ²
0.005	>	5	≥	0.004	5 ²
0.004	>	4	≥	0.003	6 ²
0.003	>	3	≥	0.002	7 ²
0.002	>	2	≥	0.001	8 ²
0.001	>	1	≥	0	9 ²

Table 7.7 Nine Categories Plus Weights

The fundamental steps of the modified method are then as follows:

- 1) Firstly each category is designated a particular percentage of the roulette wheel depending on:
 - (a) The number of strings, n , of the total population, N , in the category
 - (b) The weight assigned to the category.

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

The following sections describe steps 1 and 2 in greater detail.

Establish the Percentage Allocation for Each Category

Each string in the population is considered in turn. It is first established into which category the string falls. As in the original method the fitness value is enhanced by subtracting its value from the upper limit of the least fit category, namely 0.2. The enhanced fitness value is then multiplied by the weight associated with the category into which the string falls. The 'weighted' values of the strings in each separate category are summed. The result is that each category is designated a particular value directly related to the number of strings it contains and its weight. Using these designated values the relative percentage of each category is calculated.

Distribute the Percentage Allocated to Each Category Between its Strings

Having established the percentage allocated to each category. The average percentage allocated to each string within each category can be evaluated. As the categories get fitter, the average percentage allocated to each string should increase.

It is imperative to ensure that strings in fitter categories are given a larger percentage than those in less fit categories. For each category, the average percentage allocated

to each string in the most previous non-empty category is ascertained. As regards the 9th category this is always zero.

The total percentage appointed to each category is split into two parts, a ‘used’ and an ‘excess’ percentage. Each string in a category, X say, must be given a percentage at least that of the average allocated to each string in the most previous non-empty category, as previously determined. If category X contains n from N strings in the population, the ‘used’ percentage is:

$$Cat_{Xused} = n \times prevav_X$$

Where,

Cat_{Xused} = The ‘used percentage of category X ,

$prevav_X$ = The average % allocated to each string in the most previous non-empty category.

The ‘excess’ percent for a category is, therefore, its total minus its ‘used’ percentage. Each string in a category is first given the average percentage allocation of each string in the most previous non-empty category. An additional portion of the categories’ ‘excess’ percentage is then given to each string.

Distribution the ‘Excess’ Percentage of Each Category Between its Strings

As categories get fitter the difference between the string’s fitness values within the categories is very small. These differences are, however, significant, particularly regarding predominantly fit populations.

Distributing the ‘excess’ percentage of a category between its contained strings again requires fitter strings to be represented by a larger value. To retain the sensitive distribution between the fitness values of the strings they are subtracted from the

upper limit of the category in question. These resulting values for each string are then summed and their relative percentage of the categories' 'excess' percent calculated.

7.4.3 An Example of The Modified Conversion Process

To demonstrate application of the modified conversion method the roulette wheel percentage associated with each string in a population of 10 individuals is established.

The first step determines the percentage allocated to each category. Table 7.8 states the penalised system unavailability of each string (Q'_{srsj}), the associated category (c_j), the enhanced fitness value, i.e. $Q'_{srsj} - 0.2$ (e_j) and multiplication of e_j by the weight associated with the respective category (w_j). The final columns are considered later.

String j	Q'_{srsj}	c_j	e_j	w_j	Sub_j	p_j
1	0.0204	7	0.1796	1.6166	0.0296	9.6
2	0.0043	5	0.1957	4.8933	7.3e-4	30.2
3	0.0792	8	0.1208	0.4831	0.0208	1.9
4	0.0058	6	0.1942	3.1065	4.5e-3	19.2
5	0.0508	8	0.1492	0.5967	0.0492	4.6
6	0.0619	8	0.1381	0.5525	0.0381	3.6
7	0.0135	7	0.1865	1.6785	0.0365	11.0
8	0.7090	-	-	-	-	0
9	0.0246	7	0.1754	1.5789	0.0254	8.7
10	0.0130	7	0.1870	1.6832	0.0370	11.1

Table 7.8 Design Data Calculated During the Modified Conversion Process

Table 7.9 considers each category. The second column states the number of strings in the population falling within the category limits. Column 3 represents the sum of each strings w_j in the respective category. Column 3 totals 16.188. The values in each row of column 3 are evaluated as a percentage of the column total and hence, gives the percentage allocated to each category.

Category No.	n_{C_j}	$\sum w_j$ in Category	Category %	Average	Used	Excess	Subsum _{C_j}
9	0	0	0	0	0	0	0
8	3	1.632	10.1	3.4	0	10.1	0.108
7	4	6.557	40.5	10.12	13.6	26.9	0.128
6	1	3.106	19.2	19.2	10.1	9.1	4.16e-3
5	1	4.893	30.2	30.2	19.2	11.0	7.3e-4
4	0	0	0	0	0	0	0
3	0	0	0	0	0	0	0
2	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0

Table 7.9 Category Values Established During the Modified Conversion Process

Step 2 involves the distribution of the percentage allocated to each category between the strings it contains. Columns 5, 6 and 7 in table 8 state the average, ‘used’ and ‘excess’ values associated with each category. For each string Q'_{SYS} is subtracted from the upper limit of its associated category, as represented in table 7 (Sub_j). The Sub_j values of each string falling in the same category are summed, as shown in the table 7.9 ($Subsum_{C_j}$).

The roulette wheel percentage (p_j) allocated to each string is subsequently derived using:

$$p_j = \left(\frac{Sub_j}{Subsum_{C_j}} \times excess_{C_j} \right) + used_{C_j} \times 100 \quad (7.4)$$

and represented in table 7.8.

Table 7.6 shows the application of the modified method to two very different populations. The first is quite diverse in the performance of the systems it contains. The second contains predominantly fit strings. The percentages arising from the original method are stated alongside as a comparison.

7.4.4 Results Comparing Both Conversion Methods

The GA was tested with ten runs using the modified conversion method. For each run 20, 0.7, 0.01 and 100 were entered for the population size, crossover rate, mutation rate and number of generations respectively.

Table 7.10 shows the best penalised system unavailability for each run of the original GA. Stated alongside is the equivalent information regarding the modified conversion method.

Conversion method	
Original	Modified
1.56e-3	1.31e-3
9.36e-4	8.14e-4
1.14e-4	9.57e-4
1.23e-3	1.19e-3
1.26e-3	9.55e-4
1.13e-3	1.25e-3
9.71e-4	8.32e-4
1.03e-3	8.93e-4
2.34e-3	7.92e-4
9.44e-4	8.25e-4
Average fitness	
1.25e-3	8.64e-4

Table 7.10 Results from Both Conversion Methods

The results show that overall system performance was generally improved using the modified method. Most importantly the new approach has an enhanced ability to converge on very a fit design.

7.4.5 Discussion of the Conversion Method Results

The original method is not able to cope adequately with highly fit populations of system designs. The fitness values of the strings are not accurately represented and hence, the information used by the GA is not in accordance with the actual population fitness information.

The modified method is able to differentiate between the strings in the fitter population, in addition to retaining the ability to handle a varied population. Essential information is not lost in the conversion process. Consequently, the modified approach portrays an enhanced ability to maintain a specific best design over the latter generations.

7.5 The Modified GA

A modified GA was created via amalgamation of the MDT modification method 2a (plus application of MDT modification method 1 to the resulting design), the modified cost penalty and the modified fitness conversion method.

10 runs of the GA were implemented with parameters set at values consistent with those used in previous tests (i.e. population size, crossover rate, mutation rate and the number of generations equal to 20, 0.7, 0.01 and 100 respectively). Table 7.11 states the fitness information associated with the best design resulting from each run. In each case no limits were violated and as such $Q_{SYS} = Q'_{SYS}$.

Run No.	Cost	MDT	Spurious Trip	Q_{SYS}
1	882	126.83	0.716	8.4e-4
2	862	129.89	0.847	7.8e-4
3	822	128.43	0.717	7.6e-4
4	862	129.56	0.717	7.81e-4
5	822	128.43	0.717	7.6e-4
6	842	127.94	0.716	7.88e-4
7	842	130	0.847	7.92e-4
8	882	128.69	0.978	8.13e-4
9	822	128.26	0.586	8.62e-4
10	782	130.06	0.847	8.08e-4

Table 7.11 Results from the Modified GA

The best overall design resulted in both the 3rd and 5th trial and had the characteristics shown in table 7.12. Figure 7.7 portrays the best design data associated with each generation of the 5th run.

Subsystem 1	No. of ESD valves	0
	No. of PTs	2
	No. of PTs to trip system	1
	MTI	29
Subsystem 2	No. of HIPS valves	2
	No. of PTs	3
	No. of PTs to trip system	2
	MTI	32
Valve type		2
PT type		1

Table 7.12 Characteristics of the Best Design

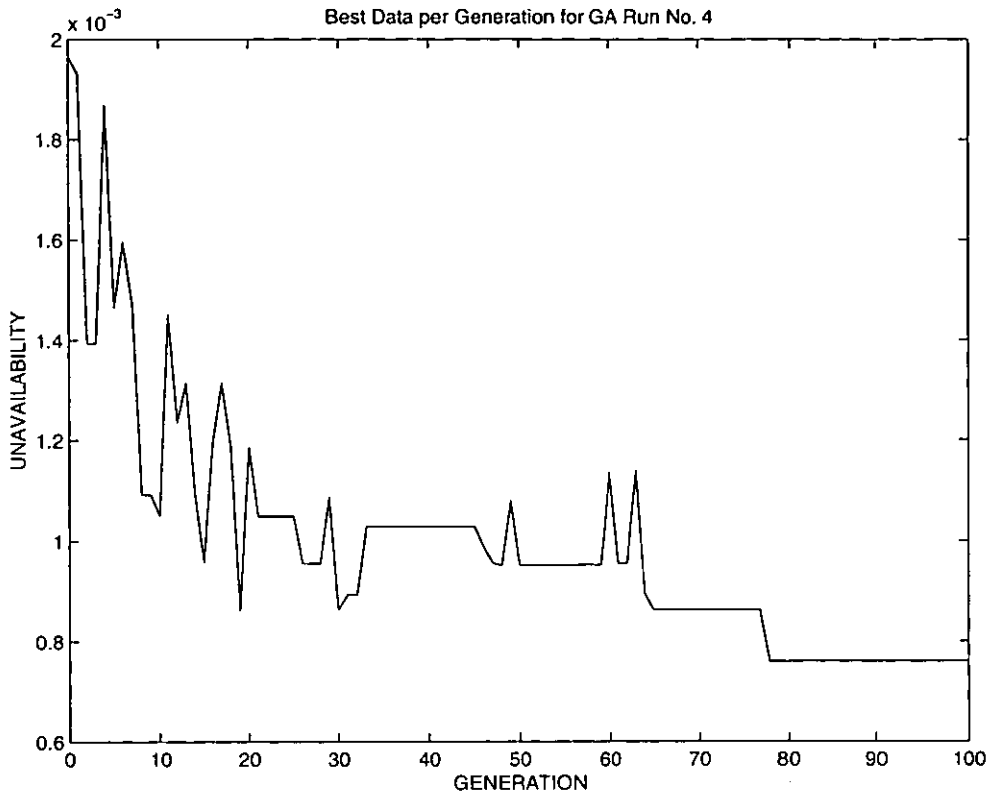


Figure 7.7 Best Data per Generation from GA Run 4

7.5.1 Discussion of Results from the Modified GA

The modified GA demonstrates the ability to find and explore the fittest areas of the search space.

Greater exploration of the MTI parameter values is introduced using the modified MDT methods 2 and 3. There is, thus, less chance that potentially fit designs are overlooked or eliminated due to poor MTI parameter values that do not fully utilize the MDT resource.

Modification of the cost penalty formula ensures that the penalty incurred takes into account the fitness of the string being penalized. This avoids instant elimination of highly fit strings from future generations that are potentially close to the optimal design but overstep a constraint boundary by a small amount.

The original conversion method, used in the SGA, is unable to accurately reflect the small but significant differences in the fitness of strings in highly fit populations (generally those in the latter generations). As a result each string is allocated approximately the same area of the roulette wheel and the GA is thus limited in its ability to distinguish and hence, converge toward the optimal design. The modified conversion method enables accurate evaluation of each strings roulette wheel percentage for both highly fit populations and populations that cover a greater range of fitness values. The GA is thus able to consistently select the fittest strings with greater probability.

The modified GA is able to differentiate between highly fit strings as the algorithm progresses, without losing potentially useful information. It has an enhanced ability to distinguish and retain the best designs over latter generations and hence, convergence to the optimal design is more probable.

It should be noted that significant fluctuation in average string fitness of the population exists throughout the entire run of the GA, as can be seen in figure 7.5. This is due to the action of the genetic operators. Crossover and mutation randomly introduce diversity into each solution string and as such modifications to the designs may be 'good or bad'. Advantageous modifications are capitilised on and selected to enter future generations. Negative modifications adversely affect population average fitness and are subsequently eliminated from the GA run.

CHAPTER 8

THE GRID-SAMPLING OPTIMISATION TECHNIQUE

8.1 Introduction

This chapter describes an alternative approach to optimise the HIPS system referred to as the grid-sampling technique. There is no explicit objective function that defines how the system unavailability is related to the design variables. This alternative optimisation method works on the basis that some form of an objective function is assumed to estimate system unavailability and a region defined over which this approximate function is considered accurate.

An initial design is chosen and an objective function derived such that it is feasible within a restricted neighbourhood of the initial design point. In a similar manner, an objective function is assumed for the implicit spurious trip frequency constraint. An efficient, computerised procedure can then analyse each point within the restricted design space to obtain the enclosed optimal design.

The procedure results in an iterative scheme where the optimal solution is approached by solving a sequence of optimisation problems. Each problem in the sequence produces a new improved solution and the next problem to solve is defined by moving the feasible solution space to the neighbourhood surrounding the new solution and re-evaluating the approximate objective functions. This is represented diagrammatically in figure 8.1, where x_0 is the initial design and x^j , $j = 1, 2$ and 3 , each subsequent point about which an optimisation problem is established.

An approach by which optimal performance can be obtained using the fault tree analysis method to determine the availability of each system design was described in a paper in 1994 (Ref. 95). The methodology represented in this chapter improves the efficiency of that used in the 1994 paper by incorporating BDDs, as described in chapter 5 of this thesis. Using BDDs enables a totally automated approach.

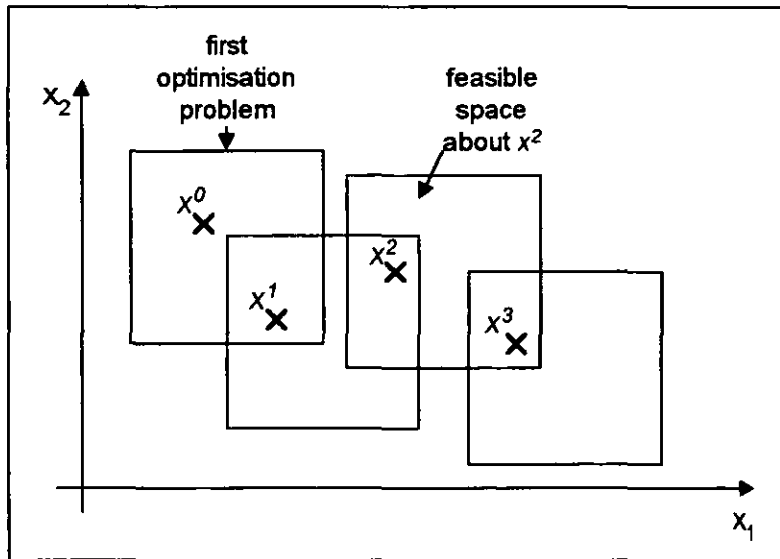


Figure 8.1 The Iterative Scheme of the Optimisation Procedure

8.2 The Optimisation Algorithm

The framework of the optimisation algorithm to implement the grid-sampling approach is depicted in figure 8.2.

An initial design is produced by the design engineer, denoted by x^0 . It is then ensured that the chosen design does not violate the MDT, cost or spurious trip limitations. If any of these constraints are violated a new start point is selected, else the design's system unavailability is calculated. (The spurious trip and system unavailability are evaluated using the aforementioned BDD technique.)

Forms of the objective function for both the system unavailability and spurious trip frequency are assumed about the initial design point and a restricted neighbourhood defined within which these functions are assumed accurate. Each integer point within the restricted space is subsequently analysed automatically using the approximate objective functions and the explicit formulae for cost and MDT.

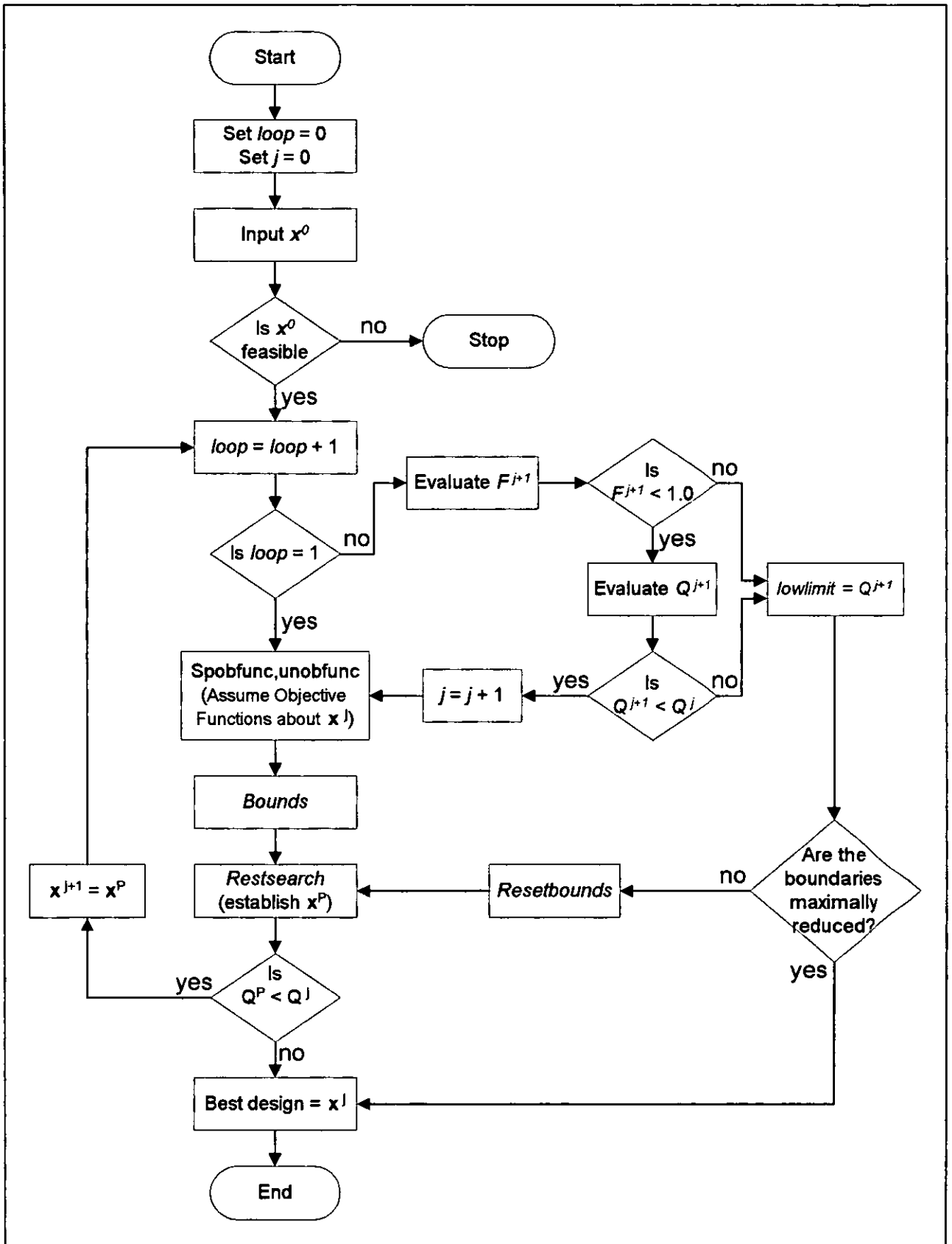


Figure 8.2 Algorithm to Implement the Grid-Sampling Method

The system unavailability of the best predicted point resulting from the restricted search, denoted by Q_{SYS}^P is compared with the system unavailability of the initial design, Q_{SYS}^j . If the predicted point shows potential for improvement it is accepted for further consideration and defined as \mathbf{x}^{j+1} , where j is iteration number (initially set to 0) and denotes the number of predicted designs that are accepted.

The actual values F_{SYS}^{j+1} and Q_{SYS}^{j+1} are evaluated using the appropriate BDD. If the spurious trip rate is less than 1 and the system unavailability verifies improvement over the previous design, the new design point is accepted. If either the spurious trip is greater than 1 or the system unavailability less than Q_{SYS}^j , the predicted point is rejected. In such circumstances, the current design point (\mathbf{x}^j) is retained, the boundaries of the restricted neighbourhood of the search space are reduced, a *lowlimit* equivalent to the previous predicted value is set and the search process repeated within the reduced restricted neighbourhood. The restricted search space is continually reduced until either the actual performance values of a predicted design show improvement over the current design or the boundaries about the search space are reduced to contain only the current point. The latter scenario terminates the algorithm and the current vector is deemed the most optimal design.

At any point the algorithm terminates if the search within the restricted neighbourhood fails to predict a design which shows improvement over the current best. The resulting optimal design is, therefore, \mathbf{x}^j .

The final value assumed by j specifies the number of optimisation problems solved. In contrast, the final value of *loop* states the number of predicted points, which were checked for their actual performance values. The difference between j and *loop*, therefore, represents those designs that were inaccurately predicted.

The following sections describe the steps of the grid-sampling optimisation algorithm in more detail.

8.2.1 Formulation of the Objective Function

To explore the design space around the initial design and progress to an improved point a means to express the system performance as a function of the design is required, i.e.

$$Q_{STS} = f(\mathbf{x})$$

where

$$\mathbf{x} = (E, N_1, K_1, H, N_2, K_2, V, P, \theta_1, \theta_2)$$

Assumption of an objective function form uses the Taylor expansion about the current design point (x) giving:

$$f(x + \Delta x) = f(x) + g^T \Delta x + \frac{1}{2} \Delta x^T H \Delta x + \dots \quad (8.1)$$

where Δx is the change in the design vector, g the gradient vector and H the Hessian matrix. Taylor series approximates the value of points sufficiently close to x , however, there is no function that can represent $f(x + \Delta x)$ for the entire design space.

Linear truncation of the Taylor expansion means that $f(x + \Delta x)$ can be evaluated providing that the gradient vector can be obtained. That is $\partial f / \partial x_i$ for each design parameter, where the partial derivatives show the rate of change of system performance with respect to the respective parameter. Strictly formulation of $\partial f / \partial x_i$ cannot occur as integer variables are being considered. To overcome this it is assumed that a smooth curve has been used to link all discrete points to give the marginal distribution of f as a function of x then $\partial f / \partial x_i$ can be obtained for the smooth curve.

Finite differences can be used to estimate $\partial f / \partial x_i$. For a linear objective function the partial derivatives are evaluated to specify the terms in the function using the central difference formula if possible:

$$\frac{\partial Q_{SYS}}{\partial x_i} = \frac{Q_{SYS}(x_1, x_2, \dots, x_{i-1}, x_i + dx_i, x_{i+1}, \dots, x_n) - Q_{SYS}(x_1, \dots, x_{i-1}, x_i - dx_i, x_{i+1}, \dots, x_n)}{2dx_i} \quad (8.2)$$

The BDD is used to obtain $f(x_i \pm \Delta x_i)$ for each variable, provided $x_i \pm \Delta x_i$ is in the required range. If the range of variable $x_i \pm \Delta x_i$ is infeasible, this is overcome using either the forward or backward scheme, equation (8.3) or (8.4) respectively:

$$\frac{\partial Q_{SYS}}{\partial x_i} = \frac{Q_{SYS}(x_1, x_2, \dots, x_{i-1}, x_i, x_{i+1}, \dots, x_n) - Q_{SYS}(x_1, \dots, x_{i-1}, x_i - dx_i, x_{i+1}, \dots, x_n)}{dx_i} \quad (8.3)$$

$$\frac{\partial Q_{SYS}}{\partial x_i} = \frac{Q_{SYS}(x_1, x_2, \dots, x_{i-1}, x_i + dx_i, x_{i+1}, \dots, x_n) - Q_{SYS}(x_1, x_2, \dots, x_{i-1}, x_i, x_{i+1}, \dots, x_n)}{dx_i} \quad (8.4)$$

The gradient vector associated with the current design point and integrated within the system unavailability objective function is derived in a routine called *Unobfunc*. The partial derivatives associated with each parameter are stored in an array termed *unob* ($E, N_1, K_1, H, N_2, K_2, V, P, \theta_1, \theta_2$ correspond to positions 0 through 9 respectively).

The range of the parameter being dealt with and its value in the current design determines the difference scheme used. Boolean variables, i.e. V and P , are necessarily restricted to either forward or backward differences. The algorithm in figure 8.3 illustrates the estimation of partial derivatives for E , the parameter governing the number of ESD valves with range 0 to 2.

```

Unobfunc
  if E = 0
    evaluate  $Q(x_{E+1}, \mathbf{x})$ 
    evaluate obun[0] using forward differences
  if E = 1
    evaluate  $Q(x_{E+1}, \mathbf{x})$  and  $Q(x_{E-1}, \mathbf{x})$ 
    evaluate obun[0] using central differences
  if E = 2
    evaluate  $Q(x_{E-1}, \mathbf{x})$ 
    evaluate obun[0] using backward differences

```

Figure 8.3 An Algorithm to Estimate the Partial Derivative w.r.t E

Incrementing a parameter value within its range may still render a design infeasible due to interactions with other parameters. Consider, for example, a design with $N_1 = K_1 = 4$. Estimating the partial derivative with respect to N_1 uses backward differences. However, $N_1 = 3$ and $K_1 = 4$ is infeasible. To overcome this K_1 must also be modified. The algorithm to estimate $\partial f / \partial x_{N_1}$ is portrayed in figure 8.4.

Problems arise when the design under consideration has no redundant system, i.e. $H = 0$. Utilising forward differences to determine the partial difference with respect to H increments H by 1. To ensure the resulting design is feasible N_2 , K_2 , and θ_2 cannot remain at 0. To overcome this the forward difference scheme is applied to an alternative design with $H = N_2 = K_2 = 1$ and $\theta_2 = 50$ to estimate $\partial f / \partial x_H$.

```

Unobfunc
if  $N_1 = 1$ 
    goto Fordiff
if  $N_1 = 2$ 
    if  $K_1 = 1$ 
        goto Cendiff
    if  $K_1 = 2$ 
        goto Fordiff
if  $N_2 = 3$ 
    if  $K_1 = 1$  or  $2$ 
        goto Cendiff
    if  $K_1 = 3$ 
        goto Fordiff
if  $N_1 = 4$ 
    if  $K_1 = 1, 2$  or  $3$ 
        goto Bacdiff
    if  $K_1 = 4$ 
        evaluate  $Q(x_{N_1-1}, x_{K_1-1}, \mathbf{x})$ 
        evaluate obun[1] using backward differences

Fordiff
    evaluate  $Q(x_{N_1+1}, \mathbf{x})$ 
    evaluate obun[1] using forward differences

Bacdiff
    evaluate  $Q(x_{N_1-1}, \mathbf{x})$ 
    evaluate obun[1] using backward differences

Cendiff
    evaluate  $Q(x_{N_1+1}, \mathbf{x})$  and  $Q(x_{N_1-1}, \mathbf{x})$ 
    evaluate obun[1] using central differences

```

Figure 8.4 An Algorithm to Estimate the Partial Derivative w.r.t N_I

Having evaluated each derivative the linearly truncated Taylor expansion takes the form:

$$Q_{SYS}(\mathbf{x} + \Delta\mathbf{x}) = Q_{SYS}^j + \sum_{i=1}^n \left(\frac{\partial Q_{SYS}}{\partial x_i} \right)^j dx_i \quad (8.5)$$

(where x^j is the current design point) to define the objective function for system unavailability.

The objective function to approximate the frequency of spurious system occurrences is derived in an identical manner. The routine to achieve this is termed *Spobfunc* and the storage array of the gradient vector *spob*.

To obtain numerical estimates of the partial derivatives of system unavailability with respect to each design variable at most $2n$ system unavailability evaluations are required. Similarly, at least n spurious trip evaluations are necessary to approximate the objective function for this failure mode. In addition, each objective function must be remodelled at each iteration for the current design point. Returning to the fault tree software, setting the correct house events and re-analysing the tree for each evaluation is a tedious process (Ref. 95). The BDD technique can be integrated within the optimisation algorithm source code. As such, alterations in the design parameter values are set automatically within the program, the associated probabilities in the BDD set accordingly and the necessary system evaluations established.

8.2.2 Limiting the Scope of the Objective Functions

The nature of the HIPS system restricts the design space. Specific ranges allocated to design variables define upper and lower bounds. Constraint functions remove other design alternatives. Additionally, since variables must assume integer values, only integer points within the design space are feasible. A feasible design space for a 2-variable problem is illustrated in figure 8.5. The intersection points on the lattice represent feasible integer points within the space.

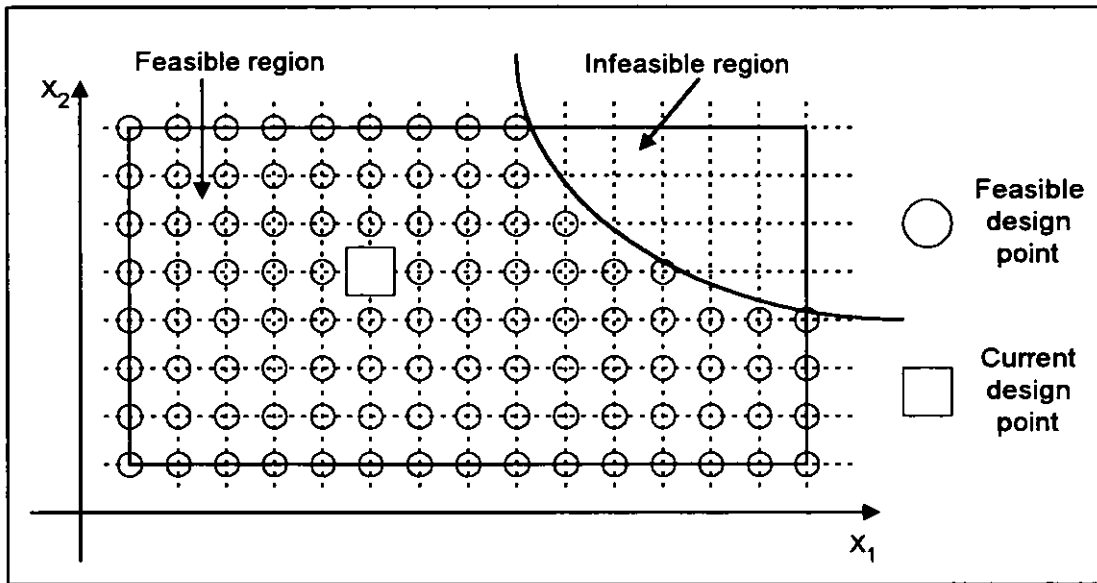


Figure 8.5 Feasible Design Space

Linear truncation of the Taylor series enables only an approximation of $Q_{SYS}(\mathbf{x} + \Delta\mathbf{x})$. As such, a restricted solution space in the neighbourhood of the current point, where the approximate solution is deemed to be acceptably accurate, must be defined. To enforce the restricted neighbourhood each variables range is limited further through the introduction of additional constraints, i.e.

$$x_i^j - \Delta x_{iL} \leq x_i^T \leq x_i^j + \Delta x_{iU}$$

where x_i^T represents the variable of a design point within the restricted space, x_i^j represents design variables at the j^{th} accepted design point and Δx_{iL} and Δx_{iU} are the lower and upper limits by which x_i is allowed to change.

The function to create a restricted neighbourhood about the current design is termed *Bounds*. Assigning further restrictions to each parameter is somewhat limited as a result of their already small range and the parameter's integer enforcement. Consequently, where possible additional constraints are set one unit either side of the current value of the design variable. The algorithm to apply bounds about N_1 , the parameter governing the number of pressure transmitters for subsystem 1, is illustrated in figure 8.6. Bounds about the other parameters are defined in a similar manner.

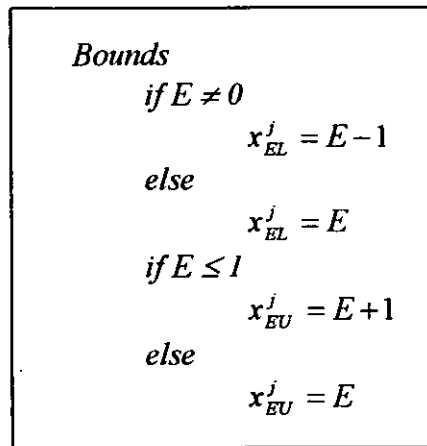


Figure 8.6 An Algorithm to Define Bounds about E

The MTI parameters are an exception. The range of values covered by the test parameters is much greater and hence, has scope to assign more strict bounds is less restricted. Initial bounds are set 12 units (weeks) either side of the actual test interval values for the j^{th} design.

8.2.3 Searching the Restricted Design Space

Having defined a restricted area about the current design within which the objective functions are assumed accurate, the set of finite feasible points in the space can be analysed. The analysis is an efficient automated process, requiring only the evaluation of explicit objective functions. The routine to carry out the analysis of each intersection point on the grid is termed *Restsearch* and the implementation algorithm is illustrated in figure 8.7.

```

Restsearch
set  $Q_{SYS}^P = 1.0$ 
do for each point in the grid,  $\mathbf{x}^T$ 
  goto Analysis
  if  $Q_{SYS}^P < Q_{SYS}^j$ 
    search successful
     $Q_{SYS}^P = Q_{SYS}^{j+1}$ 
  else
    search unsuccessful
    best design is  $\mathbf{x}^j$ 
    STOP

Analysis
  evaluate cost and MDT of  $\mathbf{x}^T$ 
  if cost  $\leq 1000$  & MDT  $\leq 130$ 
    goto Increment
    goto Estimatespur
    if  $F_{SYS}^T \leq 1$ 
      goto Estimateun
      if  $Q_{SYS}^T > \text{lowlimit}$ 
        if  $Q_{SYS}^T < Q_{SYS}^P$ 
           $Q_{SYS}^P = Q_{SYS}^T$ 
           $\mathbf{x}^P = \mathbf{x}^T$ 

```

Figure 8.7 The Algorithm to Implement *Restsearch*

Consider each integer design point within the space in turn, \mathbf{x}^T say. The cost and MDT of the design is first checked for feasibility. If either limit is violated the design is rejected and the next point considered. If no violation occurs the design (\mathbf{x}^T) is sent to a routine termed *Increment*. *Increment* establishes the difference of each parameter value (x_i^T) from the current design (x_i^j) and stores these values in an array termed *inc*. The gradient vector associated with design \mathbf{x}^j has been evaluated previously in *Spobfunc* for the spurious objective function. The values in *inc* specific to design \mathbf{x}^T are inserted into the spurious objective function in the routine *Estimatespur*. An estimate of the design's spurious trip frequency results. If this estimated value (F_{SYS}^T) is greater than 1 the design is rejected and the next point on the lattice considered. If no spurious trip violation occurs the increment array is used

in a routine termed *Estimateun* to estimate the design's system unavailability (Q_{SYS}^T). The algorithm to implement *Estimateun* is illustrated in figure 8.8.

<p><i>Estimateun</i> set $sumgrad = 0$ do for $i = 0$ to 9 $sumgrad = inc[i] \times unob[i]$ $Q_{SYS}^T = Q_{SYS}^j + sumgrad$</p>

Figure 8.8 The Algorithm to Implement *Estimateun*

The aim of *Restsearch* is to obtain the most optimal predicted point in the restricted space, excluding \mathbf{x}^j . The best point is recorded in vector \mathbf{x}^P . Q_{SYS}^P is then compared with Q_{SYS}^j . If $Q_{SYS}^j \leq Q_{SYS}^P$ it is assumed \mathbf{x}^j is the optimal design and the algorithm is terminated. If $Q_{SYS}^P < Q_{SYS}^j$, \mathbf{x}^P is temporarily accepted as the new design point \mathbf{x}^{j+1} . Actual system unavailability and spurious trip calculations, i.e. F_{SYS}^{j+1} and Q_{SYS}^{j+1} respectively, are undertaken to verify the estimated performance measures. If these measures are accurately predicted the new point is accepted and a new iteration about this point commences. If either measure proves significantly inaccurate, i.e. $F_{SYS}^{j+1} > 1$ or $Q_{SYS}^{j+1} > Q_{SYS}^j$, attention is returned to the current design point \mathbf{x}^j and a repeated search about this point proceeds. Prior to reinitiating *Restsearch* a lower bound on the predicted system unavailability must be established, termed *lowlimit*. *Lowlimit* is set equal to the predicted system unavailability of the previously rejected point, thus ensuring this point is not reselected. In addition, the restricted area around \mathbf{x}^j must be reduced further to ensure accuracy of performance predictions.

8.2.4 Reducing the Restricted Design Space

The routine to reduce the size of the restricted neighbourhood about \mathbf{x}^j is termed *Resetbounds*. The bounds to be reduced refer only to the restricted range about the MTI parameters. Bounds about the other parameters are maximally reduced due to the integer restriction. Each consecutive time a predicted improvement ($Q_{SYS}^{j+1} > Q_{SYS}^j$) proves invalid, the distance of the enforced upper and lower bound about the actual test parameter value is halved. On the first occasion

$\theta^j - 12 \leq \theta^T \leq \theta^j + 12$ is reduced to $\theta^j - 6 \leq \theta^T \leq \theta^j + 6$, the second to $\theta^j - 3 \leq \theta^T \leq \theta^j + 3$. On the fourth and fifth occasion the bounded interval is 4 and 2 units wide respectively so as not to violate the integer restriction. The first prediction that proves to be accurate is accepted as the new point about which the next iteration proceeds. A restricted neighbourhood is established about this new vector, where the MTI bounds are relaxed to a 24-unit interval. If, however, a predicted point is not accepted prior to the 6th trial about \mathbf{x}^j the algorithm terminates and \mathbf{x}^j is deemed to be the optimal design.

8.2.5 Application of the Grid-Sampling Method

The following section demonstrates the application of the grid-sampling optimisation method to a high pressure protection optimisation.

The initial design is stated in the first row of table 8.1. The following rows state the consecutive designs, which were predicted to show improvement over the current design. Those designs that were accepted when their actual performance measures were evaluated are represented in bold. As can be seen, the first design accepted as an improvement over the initial vector is that predicted 1st. The most optimal design from the grid sampling process using the specified initial design is described in the 5th row.

The fitness values associated with each design are stated in table 8.2. The values predicted for system unavailability and spurious trip frequency are also given. (Actual values only are determined for the initial design.) The initial design has a system unavailability of 3.95×10^{-3} . The system unavailability of the final design shows significant improvement specifying a value of 7.97×10^{-4} .

Predicted Design No.	Accepted Design No.	E	K_1/N_1	H	K_2/N_2	V	P	θ_1	θ_2	Actual Q_{SYS}
Initial Design		1	2/2	1	1/1	1	1	40	50	3.95e-3
1	1	0	1/3	1	2/2	1	1	30	39	9.34e-4
2	-	0	2/2	2	1/3	2	1	30	30	9.35e-4
3	-	0	2/2	2	1/3	2	1	28	33	9.62e-4
4	2	0	2/3	2	1/3	2	1	27	36	7.97e-4
5	-	0	1/4	2	1/2	2	1	30	31	9.69e-4
6	-	0	1/4	2	2/2	2	1	31	30	9.7e-4
7	-	0	1/4	2	2/2	2	1	28	34	9.91e-4
8	-	0	1/4	2	2/2	2	1	27	36	1.01e-4
9	-	0	1/4	2	2/3	2	1	28	35	8.04e-4

Table 8.1 Characteristics of Each Predicted Design

Predicted Design No.	Accepted Design No.	Predicted Q_{SYS}	Predicted F_{SYS}	Actual Q_{SYS}	Actual F_{SYS}	MDT	Cost
Initial Design		-	-	3.95e-3	0.420	101.66	882
1	1	-1.48e-3	0.812	9.34e-4	0.942	129.33	922
2	-	6.83e-4	0.716	9.35e-4	0.847	130	822
3	-	6.91e-4	0.716	9.62e-4	0.847	129.16	822
4	2	7.32e-4	0.846	7.97e-4	0.847	129.04	842
5	-	5.01e-4	0.651	9.69e-4	0.977	129.78	842
6	-	5.08e-4	0.651	9.7e-4	0.977	129.44	842
7	-	5.09e-4	0.651	9.91e-4	0.977	129.67	842
8	-	5.25e-4	0.651	1.01e-3	0.977	129.52	842
9	-	5.32e-4	0.781	8.04e-4	0.976	129.63	862

Table 8.2 Fitness Values of Each Predicted Design

An objective function is first assumed about the initial design to approximate both the system unavailability and spurious trip frequency. Table 8.3 states the gradient vector

values and the difference scheme used for each parameter for each failure mode (F,B and C denote forward difference, backward difference and central difference respectively). In addition, the upper and lower bounds of the restricted neighbourhood established about each parameter are specified. (If the HIPS valve is set to 0 parameters N_2 , K_2 , and θ_2 are modified accordingly.)

Parameter	Difference Scheme	Spurious $\partial f / \partial x_i$	Unavailability $\partial f / \partial x_i$	Upper Bound	Lower Bound
E	C	8.73e-3	-1.30e-4	0	2
N_1	F	2.83e-4	-8.77e-4	1	3
K_1	B	-0.261	8.82e-4	1	3
H	F	8.73e-3	1.75e-3	0	2
N_2	F	0.131	2.54e-4	1	2
K_2	F	-0.262	5.07e-4	1	2
V	F	0.166	7.94e-4	1	2
P	F	0.482	4.90e-3	1	2
θ_1	C	0	9.73e-5	28	52
θ_2	C	0	7.59e-5	38	62

Table 8.3 Objective Function Coefficients Associated with \mathbf{x}^0 , Evaluated Using Finite Differences

The restricted neighbourhood about the initial design (\mathbf{x}^0) was analysed and an improved design predicted. The actual performance values associated with this predicted design proved to be both feasible and fitter than the initial vector, giving rise to \mathbf{x}^1 . A second pair of objective functions was consequently established about \mathbf{x}^1 . The associated partial derivatives and parameter bounds are stated in table 8.4.

Parameter	Difference Scheme	Spurious $\partial f / \partial x_i$	Unavailability $\partial f / \partial x_i$	Upper Bound	Lower Bound
E	F	8.69E-3	-5.61E-5	0	1
N_1	C	0.130	-2.45E-7	2	4
K_1	F	-0.391	1.465E-6	1	2
H	F	8.69E-3	-2.6E-4	0	2
N_2	F	0.130	-6.53E-5	1	3
K_2	B	-0.261	2.6E-4	1	3
V	F	0.165	3.26E-5	1	2
P	F	2.368	1.37E-5	1	2
θ_1	C	0	3.25E-5	18	42
θ_2	C	0	2.45E-5	27	51

Table 8.4 Objective Function Coefficients Associated with \mathbf{x}^1

The restricted neighbourhood about \mathbf{x}^1 was analysed and an improved design predicted. The actual system unavailability of this design proved less fit than Q_{SYS}^1 and was, therefore, rejected. The MTI boundaries were reduced to $24 \leq \theta_1 \leq 36$ and $33 \leq \theta_2 \leq 45$ and this reduced neighbourhood re-analysed. Similarly the next predicted design proved less fit and the MTI boundaries were further reduced to $27 \leq \theta_1 \leq 33$ and $36 \leq \theta_2 \leq 42$. A further analysis of the area produced a design whose actual values showed improvement over Q_{SYS}^1 and was therefore accepted, giving rise to \mathbf{x}^2 . The feasible solution space was moved to the neighbourhood surrounding this new solution (with MTI bounds relaxed to a 24-unit interval) and the objective function coefficients re-evaluated, as shown in table 8.5.

Parameter	Difference Scheme	Spurious $\partial f / \partial x_i$	Unavailability $\partial f / \partial x_i$	Upper Bound	Lower Bound
E	F	0.0913	-4.32e-5	0	1
N_1	C	0.130	-1.10e-4	2	4
K_1	C	-0.195	1.64e-4	1	3
H	B	0.0914	-1.03e-3	1	2
N_2	C	0.130	-2.49e-7	2	4
K_2	F	-0.392	1.49e-6	1	2
V	B	0.165	2.51e-5	1	2
P	F	1.435	2.36e-5	1	2
θ_1	C	0	3.07e-5	15	39
θ_2	C	0	2.31e-5	24	48

Table 8.5 Objective Function Coefficients associated with \mathbf{x}^2

Analysis of the region surrounding \mathbf{x}^2 produced 5 consecutive designs predicted to improve Q_{SYS}^2 . The actual system unavailability of each prediction proved, however, to be inferior. Following the fifth prediction each MTI's bounded interval had been maximally reduced and the algorithm is terminated. The most optimal design resulting from this test was \mathbf{x}^2 .

8.2.6 Results Using the Grid-Sampling Approach

The Grid-Sampling approach was tested using 8 alternative initial designs. Table 8.6 specifies each initial design in bold and shows the designs produced at each subsequent iteration for each test. Table 8.7 summarises the optimal design resulting from each run.

Iteration	E	K_1/N_1	H	K_2/N_2	V	P	θ_1	θ_2	Q_{SYS}	F_{SYS}	Cost	MDT
Test 1												
0	1	2/2	1	1/1	1	1	40	50	3.95e-3	0.419	882	101.6
1	0	1/3	1	2/2	1	1	30	39	9.34e-4	0.942	922	129.3
2	0	2/3	2	1/3	2	1	27	36	7.97e-4	0.847	842	129
Test 2												
0	1	1/1	1	1/1	2	1	45	34	3.27e-3	0.716	762	90.3
1	0	1/2	2	1/2	2	1	34	26	7.23e-4	0.977	802	129.6
Test 3												
0	1	2/2	1	1/1	2	1	60	25	3.64e-3	0.586	782	86.3
1	0	1/3	2	1/2	1	1	58	25	1.20e-3	0.942	922	129.5
2	0	2/2	2	1/3	2	1	46	22	1.06e-3	0.847	822	129.2
3	0	1/3	2	2/2	2	1	41	23	9.96e-4	0.846	822	129.9
4	0	2/2	2	1/3	2	1	35	26	9.46e-4	0.847	822	129.9
5	0	1/3	2	2/2	2	1	32	28	9.35e-4	0.846	822	130
6	0	2/2	2	1/3	2	1	31	29	9.34e-4	0.847	822	130
Test 4												
0	2	1/1	0	0/0	2	1	50	0	4.85e-2	0.464	721	70.7
1	2	1/2	0	0/0	2	1	38	0	3.23e-2	0.594	741	94.4
2	2	1/3	0	0/0	2	1	28	0	2.39e-2	0.725	761	130
3	1	1/3	0	0/0	2	1	22	0	1.89e-2	0.633	541	130
4	0	1/3	0	0/0	1	1	16	0	1.43e-2	0.411	301	126.7
Test 5												
0	2	1/1	1	1/1	2	1	63	33	4.42e-3	0.808	982	86.1
1	1	1/1	2	1/1	2	1	52	23	1.19e-3	0.808	982	129.9
2	0	1/2	2	1/2	2	1	49	21	8.58e-4	0.977	802	128.1
3	0	1/2	2	1/2	2	1	43	22	7.82e-4	0.977	802	129.9
4	0	1/2	2	1/2	2	1	40	23	7.58e-4	0.977	802	129.8
5	0	1/2	2	1/2	2	1	38	24	7.5e-4	0.977	802	129.2
Test 6												
0	0	1/1	2	3/4	1	2	45	35	2.23e-3	0.899	872	120.8
1	0	1/1	2	2/4	2	1	34	27	8.53e-4	0.586	822	129.4
2	0	1/2	2	3/4	2	1	32	29	7.62e-4	0.716	842	129.7
3	0	1/3	2	2/3	2	1	33	28	7.57e-4	0.847	842	129.9
Test 7												
0	0	2/3	1	1/3	1	1	40	40	2.51e-3	0.67	672	85.5
Test 8												
0	2	1/1	1	1/1	1	1	40	50	4.2e-3	0.807	982	108.2

Table 8.6 Results from each Iteration of Tests 1 to 8

Test No.	E	K_1/N_1	H	K_2/N_2	V	P	θ_1	θ_2	Q_{SYS}	F_{SYS}	Cost	MDT
1	0	2/3	2	1/3	2	1	27	36	7.97e-4	0.847	842	129
2	0	1/2	2	1/2	2	1	34	26	7.23e-4	0.977	802	129.6
3	0	2/2	2	1/2	2	1	31	29	9.34e-4	0.847	822	130
4	0	1/3	0	0/0	1	1	16	0	1.43e-2	0.411	301	126.7
5	0	1/2	2	1/2	2	1	38	24	7.5e-4	0.977	802	129.2
6	0	1/3	2	2/3	2	1	33	28	7.57e-4	0.847	842	129.9
7	0	2/3	1	1/3	1	1	40	40	2.51e-3	0.67	672	85.5
8	2	1/1	1	1/1	1	1	40	50	4.2e-3	0.807	982	108.2

Table 8.7 A Summary of the Best Design from Each Test

Tests 1 to 6 produced a fitter design than initially chosen. The optimal designs resulting from tests 1, 2 5 and 6 are very fit. A fitter design was not found on either test 7 or 8. The optimal designs produced in each run are somewhat varied.

8.2.7 Discussion of Results Using the Linearised Grid-Sampling Method

The optimisation procedure proves to be very effective if an appropriate initial design is chosen. Tests 2 and 5 produce very similar results, as shown in table 8.7. The only difference is that the maintenance allocated to the 5th test is not as effective. An unavailability of 7.57×10^{-4} results from the optimal design in the 6th test. This design differs from tests 2 and 5 in that both sub-systems have 3 as opposed to 2 pressure transmitters and the HIPS sub-system is initiated by 2 of the 3 transmitters being activated. Although the cost is slightly more for the latter design, it may be preferred due to the lower spurious trip rate.

Table 8.6 confirms the dependence of the optimisation procedure on the initial design vector. The search is focused about a single point in the entire design space. Search progresses toward the optimum area in the vicinity of the point. The result is that the method is somewhat local in scope. This is illustrated in test 3, where the local

optimum has an unavailability of 9.34×10^{-4} . Fitter areas of the design space, resulting in designs with performance values approximately 7.5×10^{-4} , remain unexplored.

In addition, problems arise due to interactions between parameter values rendering the design infeasible. The extreme case is when the HIPS valve is equal to 0. As such, finite differences with respect to H require special treatment and incur greater inaccuracy in the numerical estimations. This is illustrated in test 4. The resulting design is comparatively poor. Inaccurate estimations of predicted designs in the restricted neighbourhood of the current point prevent more optimal points designs from being selected. This occurs to a lesser extent between other parameters, e.g. the number of pressure transmitters fitted and the number required to trip the system, and is possibly a factor in the poor performance of tests 7 and 8.

In the main, the optimisation procedure enables full utilisation of the MDT resource, distributed across each sub-system to the best advantage. Bounds governing the feasible design over which the optimisation proceeds are relaxed for the test parameters. Greater exploration of the test parameter combinations for each design in the space, therefore ensues.

8.3 Other Formulations of the Objective Function

The accuracy of the objective function is a determining factor in the effectiveness of the grid-sampling technique. A potential means to improve the technique involves consideration of extra terms in the Taylor expansion. As such, two variations in assumption of the objective function are investigated.

8.3.1 The Pure Quadratic Objective Function

The first variation involves incorporation of the pure quadratic contributions from an additional term in the Taylor expansion. The objective function, therefore, assumes the form:

$$Q_{SYS}(\mathbf{x} + \Delta\mathbf{x}) = Q_{SYS}^j + \sum_{i=1}^n a_i \Delta x_i + \frac{1}{2} \sum_{i=1}^n a_{i+n} (\Delta x_i)^2 \quad (8.6)$$

where

$$a_i = \left(\frac{\partial Q_{SYS}}{\partial x_i} \right)^j \quad i = 1, \dots, n$$

and

$$a_{i+n} = \left(\frac{\partial^2 Q_{SYS}}{\partial x_i^2} \right)^j \quad i = 1, \dots, n$$

The first partial derivatives are derived numerically, as described previously for the linear case. The second derivatives are evaluated using the current design vector j and the previous design point $j + 1$:

$$a_{i+n} = \frac{\left(\frac{\partial Q}{\partial x_i} \right)^{j-1} - \left(\frac{\partial Q}{\partial x_i} \right)^j}{x_i^{j-1} - x_i^j} \quad i = 1, \dots, n \quad (8.7)$$

As such, the initial form of the objective function about the initial design vector is linear. Having, accomplished the first iteration, sufficient information is available to evaluate the second order terms, which enable use of the quadratic objective function.

Incorporating the pure quadratic objective function within the implementation algorithm is straightforward. Once a new design is established the gradient vectors evaluated about the previous point are stored. The algorithm then proceeds in the normal manner and the gradient vectors about the new design are calculated. The first order partial derivatives from points j and $j - 1$ are subsequently used to establish the second order coefficients in equation (8.6). In addition, *Estimateun* and *Estimatespur* are modified to incorporate the quadratic term in the Taylor expansion.

8.3.1.1 Application using the Pure Quadratic Objective Function

Application of the grid-sampling technique using the pure quadratic objective function is demonstrated by means of the following example.

The initial design about which the algorithm proceeds is stated in the first row of table 8.8. The following rows specify each predicted design, where the accepted designs are stated in bold, in a similar manner to application of the linear scheme. The fitness values associated with each predicted design are given in table 8.9. The initial design has a system unavailability of 3.95×10^{-3} and the final resulting design assumes the fitter performance value of 9.34×10^{-4} .

Predicted Design No.	Accepted Design No.	E	K_1/N_1	H	K_2/N_2	V	P	θ_1	θ_2	Actual Q_{SYS}
Initial Design		1	2/2	1	1/1	1	1	40	50	3.95e-5
1	1	0	1/3	2	1/2	1	1	30	39	9.34e-4
2	-	0	2/2	2	1/2	2	1	26	35	9.49e-4
3	-	0	2/2	2	1/2	2	1	27	34	9.57e-4
4	-	0	2/2	2	1/2	2	1	27	36	1.02e-3
5	-	0	2/2	2	1/2	2	1	28	37	1.08e-3
6	-	0	2/2	2	1/2	1	1	29	39	1.14e-3

Table 8.8 Characteristics of Each Predicted Design

Predicted Design No.	Accepted Design No.	Predicted Q_{SYS}	Predicted F_{SYS}	Actual Q_{SYS}	Actual F_{SYS}	MDT	Cost
Initial Design		N/A	N/A	3.95e-5	0.419	101.7	882
1	1	-1.48e-3	0.812	9.34e-4	0.942	129.3	922
2	-	8.28e-4	0.586	9.49e-4	0.716	130	802
3	-	8.34e-4	0.585	9.57e-4	0.716	128.6	802
4	-	8.46e-4	0.586	1.02e-3	0.716	125.7	802
5	-	8.75e-4	0.586	1.08e-3	0.716	121.6	802
6	-	9.04e-3	0.421	1.14e-3	0.551	129.9	902

Table 8.9 Fitness Values of Each Predicted Design

A linear form of the objective function is first assumed about the initial design, identical to that of the linear scheme. The values of each gradient vector are specified in table 8.10.

x_i	Initial Design (x^0)		$x_i^0 - x_i^1$	Design x^1			
	$\left(\frac{\partial Q_{SYS}}{\partial x_i}\right)^0$	$\left(\frac{\partial F_{SYS}}{\partial x_i}\right)^0$		$\left(\frac{\partial Q_{SYS}}{\partial x_i}\right)^1$	$\left(\frac{\partial F_{SYS}}{\partial x_i}\right)^1$	$\left(\frac{\partial^2 Q_{SYS}}{\partial x_i^2}\right)^1$	$\left(\frac{\partial^2 F_{SYS}}{\partial x_i^2}\right)^1$
E	-1.30e-3	8.73e-3	1	-5.61e-5	8.69e-3	-7.414e-5	3.80e-5
N_1	-8.77e-3	2.82e-4	-1	-2.45e-7	0.130	8.77e-4	0.130
K_1	8.82e-3	-0.261	1	1.46e-6	-0.391	8.80e-4	0.129
H	-1.75e-2	8.73e-3	-1	-8.72e-4	8.69e-3	8.74e-4	-3.70e-5
N_2	-2.54e-4	0.141	-1	-6.53e-5	0.130	1.88e-4	-4.94e-4
K_2	5.07e-4	-0.262	0	2.6e-3	-0.261	0	0
V	7.94e-4	0.166	0	3.26e-5	0.165	0	0
P	4.90e-3	0.482	0	1.37e-5	2.36	0	0
θ_1	9.74e-5	0	10	3.25e-5	0	6.49e-6	0
θ_2	7.59e-5	0	11	2.44e-5	0	4.68e-6	0

Table 8.10 First and Second Order Derivative Information

The first design predicted to show improvement over the initial vector was accepted, i.e. \mathbf{x}^1 . A pure quadratic form of each objective function was established about \mathbf{x}^1 .

The first step derived each gradient vector in the normal manner. $\left(\frac{\partial f}{\partial x_i}\right)^1$ and $\left(\frac{\partial f}{\partial x_i}\right)^0$

were subsequently used in equation (8.7) to evaluate the second partial derivatives.

Values of the first and second order derivatives are specified in table 8.10.

Each design predicted to improve upon Q_{SYS}^1 was found to be inferior, with the result that the restricted neighbourhood was maximally reduced and \mathbf{x}^1 deemed the optimal design.

8.3.1.2 Results Using the Pure Quadratic Objective Function

The grid-sampling algorithm was applied to 8 initial designs using the pure quadratic objective function. The initial designs were chosen to be the same as those tested using the linear scheme to enable direct comparisons. Table 8.11 specifies each initial design in bold and shows the design produced at each subsequent iteration for each test.

Iteration	E	K_1/N_1	H	K_2/N_2	V	P	θ_1	θ_2	Q_{STS}	F_{STS}	Cost	MDT
Test 1												
0	1	2/2	1	1/1	1	1	40	50	3.95e-4	0.419	882	101.7
1	0	1/3	2	1/2	1	1	30	39	9.34e-4	0.942	922	129.3
Test 2												
0	1	1/1	1	1/1	2	1	45	34	3.27e-3	0.716	762	90.3
1	0	1/2	2	1/2	2	1	34	26	7.23e-4	0.977	802	129.6
Test 3												
0	1	2/3	1	1/1	2	1	60	25	3.64e-3	0.585	782	86.3
1	0	1/3	2	1/2	1	1	58	25	1.20e-3	0.942	922	129.5
Test 4												
0	2	1/1	0	0/0	2	1	50	0	4.85e-2	0.463	721	70.7
1	2	1/2	0	0/0	1	1	38	0	3.23e-2	0.428	841	108.1
2	1	1/2	0	0/0	1	1	26	0	2.22e-2	0.420	571	118
3	0	1/3	0	0/0	1	1	16	0	1.43e-2	0.411	301	126.7
Test 5												
0	2	1/1	1	1/1	2	1	63	33	4.41e-3	0.808	982	86.1
1	1	1/1	2	1/1	2	1	52	23	1.19e-3	0.808	982	129.9
2	0	1/2	2	1/2	2	1	47	21	8.2e-4	0.977	802	129.8
3	0	1/2	2	1/2	2	1	43	22	7.82e-4	0.977	802	129.9
4	0	1/2	2	1/2	2	1	40	23	7.58e-4	0.977	802	129.8
5	0	1/2	2	1/2	2	1	34	26	7.23e-4	0.977	802	129.6
Test 6												
0	0	1/1	2	3/4	1	2	45	35	2.23e-3	0.899	872	120.8
1	0	1/1	2	2/4	2	1	34	27	8.53e-4	0.586	822	129.4
2	0	1/2	2	2/4	2	1	32	29	7.59e-4	0.717	842	129.7
3	0	1/2	2	2/3	2	1	35	26	7.46e-4	0.717	822	129.9
Test 7												
0	0	2/3	1	1/3	1	1	40	40	2.51e-3	0.673	672	85.8
1	0	1/4	2	2/2	1	1	34	35	1.22e-3	0.812	942	129.6
2	0	2/3	2	1/2	2	1	30	31	7.62e-4	0.717	822	128
Test 8												
0	2	1/1	1	1/1	2	1	40	50	4.2e-3	0.807	982	108.2

Table 8.11 Results from each Iteration of Tests 1 to 8 (Pure Quadratic)

8.3.2 The Cross Quadratic Objective Function

A second variation in the objective function is investigated by inclusion of the cross product terms in the Taylor expansion. The objective function is then of the form:

$$Q(\mathbf{x} + \Delta\mathbf{x}) = Q_{SYS}^j + \sum_{i=1}^n a_i \Delta x_i + \sum_{i=1}^n \sum_{\substack{k=1 \\ k \neq i}}^n b_{ik} \Delta x_i \Delta x_k + \frac{1}{2} \sum_{i=1}^n c_i (\Delta x_i)^2 \quad (8.8)$$

where

$$a_i = \left(\frac{\partial Q_{SYS}}{\partial x_i} \right)^j - \sum_{k=1}^n \left(\frac{\partial^2 Q_{SYS}}{\partial x_i \partial x_k} \right)^j \Delta x_k \quad i = 1, \dots, n$$

$$b_{ik} = \left(\frac{\partial^2 Q_{SYS}}{\partial x_i \partial x_k} \right)^j \quad i = 1, \dots, n \quad k = 1, \dots, n \quad k \neq i$$

and

$$c_i = \left(\frac{\partial^2 Q_{SYS}}{\partial x_i^2} \right)^j \quad i = 1, \dots, n$$

Each partial derivative is evaluated using finite differences. Derivation of the first derivatives is discussed previously in section 8.2.1. The pure second partial derivatives are evaluated using the finite difference formula:

$$\frac{\partial^2 Q_{SYS}}{\partial x_i^2} = \frac{Q(x_i + \Delta x_i) - 2Q(\mathbf{x}) + Q(x_i - \Delta x_i)}{\Delta x_i^2} \quad (8.9)$$

where

$$Q(x_i + \Delta x_i) = Q(x_1, x_2, \dots, x_{i-1}, x_i + \Delta x_i, x_{i+1}, \dots, x_n)$$

and

$$Q(x_i - \Delta x_i) = Q(x_1, x_2, \dots, x_{i-1}, x_i - \Delta x_i, x_{i+1}, \dots, x_n)$$

The system unavailability terms are evaluated automatically by perturbing the variable x_i within the program source code and using the BDD technique. Derivation of the cross product terms uses the finite difference formula:

$$\frac{\partial^2 Q_{sys}}{\partial x_i \partial x_k} = \frac{[Q(x_i + \Delta x_i, x_k + \Delta x_k) - Q(x_i + \Delta x_i, x_k - \Delta x_k)] - [Q(x_i - \Delta x_i, x_k + \Delta x_k) - Q(x_i - \Delta x_i, x_k - \Delta x_k)]}{4\Delta x_i \Delta x_k} \quad (8.10)$$

where both variables x_i and x_k are perturbed.

Subroutines *Unobfunc* and *Spobfunc* are extended to evaluate the second derivatives. As regards the pure quadratic terms, the derivative is 0 if the specific value of the respective parameter does not allow the use of central differences. The limited range of some parameters enforces a zero valued second derivative regardless of its specific value in the current design \mathbf{x}^j , i.e. parameters V , P and H . Evaluation of the cross terms requires both considered parameters to allow the use of central differences, else the respective cross term value is zero.

Numerical estimates of the partial derivatives are subsequently used to establish values for the Taylor coefficients. The linear coefficients are calculated within *Estimatum* and *Estimatspur*. They are specific to the design point being analysed (\mathbf{x}^T) within the feasible neighbourhood of \mathbf{x}^j due to their dependence on the distance of \mathbf{x}^T from \mathbf{x}^j (i.e. the values established in *Increment*).

8.3.2.1 Application using the Objective Function with Cross Terms

To demonstrate the formulation of the quadratic objective function with cross terms the grid-sampling technique is applied to the initial design specified in row 1 of table 8.12. Again, each accepted design in the program run is portrayed in bold and the associated fitness of each design specified in table 8.13.

Predicted Design No.	Accepted Design No.	E	K_1/N_1	H	K_2/N_2	V	P	θ_1	θ_2	Actual Q_{SYS}
Initial Design		0	1/1	2	3/4	1	2	45	35	2.23e-3
1	1	0	1/1	2	2/4	2	1	34	27	8.53e-4
2	-	0	1/2	2	1/4	2	1	32	29	-
3	2	0	1/2	2	3/4	2	1	32	29	7.62e-4
4	3	0	1/3	2	2/3	2	1	33	28	7.57e-4
5	-	0	1/2	2	1/4	2	1	32	29	-
6	-	0	2/2	2	1/4	2	1	32	29	9.65e-4
7	-	0	1/2	2	1/4	2	1	31	30	-
8	-	0	2/2	2	1/4	2	1	31	30	9.67e-4
9	-	0	1/2	2	1/4	2	1	30	31	-

Table 8.12 Characteristics of Each Predicted Design

Predicted Design No.	Accepted Design No.	Predicted Q_{SYS}	Predicted F_{SYS}	Actual Q_{SYS}	Actual F_{SYS}	MDT	Cost
Initial Design		N/A	N/A	2.23e-3	0.899	120.8	872
1	1	-6.07e-3	0.585	8.53e-4	0.586	129.4	822
2	-	7.66e-3	0.718	-	1.24	129.7	942
3	2	7.68e-4	0.196	7.62e-4	0.716	129.7	842
4	3	2.23e-4	0.847	7.57e-4	0.847	129.9	842
5	-	2.41e-4	0.652	-	1.24	129.7	842
6	-	2.42e-4	0.261	9.65e-4	0.977	129.7	842
7	-	2.48e-4	0.652	-	1.24	129.5	842
8	-	2.49e-4	0.261	9.67e-4	0.977	129.5	842
9	-	2.56e-4	0.652	-	1.24	129.7	842

Table 8.13 Fitness Values of Each Predicted Design

The partial derivatives about the initial design are derived using finite differences (equations (8.2),(8.3) or (8.4) for the first derivatives and (8.8) and (8.9) for the

second derivatives). These values are specified in table 8.14. (The cross terms not specified are equal to 0. Note that the value of each cross term in the spurious objective function is zero.)

The cross and pure second derivatives correspond directly to b_{ik} and c_i in equation 8 respectively. The coefficients constituting a_i are dependent on incremental information, i.e. $\Delta x_i = x_i^T - x_i^0$, where \mathbf{x}^T is the point being analysed in the neighbourhood of \mathbf{x}^0 . The values of a_i at the next accepted point, i.e. $\mathbf{x}^T = \mathbf{x}^1$ are stated in table 8.15.

Pure Partial Derivatives					Cross Partial Derivatives			
x_i	$\frac{\partial Q_{SYS}}{\partial x_i}$	$\frac{\partial F_{SYS}}{\partial x_i}$	$\frac{\partial^2 Q_{SYS}}{\partial x_i^2}$	$\frac{\partial^2 F_{SYS}}{\partial x_i^2}$	x_i, x_k	$\frac{\partial^2 Q_{SYS}}{\partial x_i \partial x_k}$	x_i, x_k	$\frac{\partial^2 F_{SYS}}{\partial x_i \partial x_k}$
E	-1.13e-4	8.69e-3	0	0	$K_2\theta_1$	5.99e-5	-	-
N_1	-8.25e-4	0.607	0	0	$K_2\theta_2$	6.96e-5	-	-
K_1	1.65e-3	-1.214	0	0	$\theta_1\theta_2$	1.57e-6	-	-
H	-1.95e-3	8.7e-3	0	0				
N_2	-0.0651	4.6e-5	0	0				
K_2	2.64e-3	-9.13e-3	4.95e-3	2.44e-6				
V	6.54e-5	0.165	0	0				
P	7.77e-4	0.479	0	0				
θ_1	5.08e-5	0	6.07e-8	7.25e-8				
θ_2	6.91e-5	0	3.22e-7	9.94e-8				

Table 8.14 Derivative information about \mathbf{x}^0

Parameter	\mathbf{x}^T	Δx_i	a_i	
			F_{SYS}	Q_{SYS}
E	0	0	8.69e-3	-1.14e-4
N_1	1	0	0.607	-8.24e-4
K_1	1	0	-1.214	1.65e-3
H	2	0	8.69e-3	-1.94e-3
N_2	4	0	4.6e-5	-0.652
K_2	2	-1	9.03e-3	8.8e-3
V	2	1	0.165	6.54e-5
P	1	1	0.479	7.77e-4
θ_1	34	-11	0	1.24e-4
θ_2	27	-10	0	1.58e-4

Table 8.15 The Value of Coefficients Constituting a_i

A second iteration about \mathbf{x}^1 subsequently proceeds. Each objective function is re-evaluated. The value of each derivative is specified in table 8.16.

x_i	$\frac{\partial Q_{SYS}}{\partial x_i}$	$\frac{\partial F_{SYS}}{\partial x_i}$	$\frac{\partial^2 Q_{SYS}}{\partial x_i^2}$	$\frac{\partial^2 F_{SYS}}{\partial x_i^2}$
E	-4.93e-5	0.0915	0	0
N_1	-1.01e-4	0.131	0	0
K_1	2.02e-4	-0.261	0	0
H	-1.14e-3	0.0916	0	0
N_2	-0.034	8.47e-4	0	0
K_2	1.23e-6	-0.26	2.44e-6	0.52
V	2.08e-5	0.166	0	0
P	3.73e-4	0.496	0	0
θ_1	2.16e-5	0	7.25e-8	0
θ_2	3.25e-5	0	9.94e-8	0

x_i, x_k	$\frac{\partial^2 Q_{SYS}}{\partial x_i \partial x_k}$
$K_2 \theta_1$	3.76e-8
$K_2 \theta_2$	8.81e-8
$\theta_1 \theta_2$	9.95e-7

\mathbf{x}^T (\mathbf{x}^2)	a_i	
	Q_{SYS}	F_{SYS}
0	-4.93e-5	0.0915
2	-1.01e-4	0.131
1	2.02e-4	-0.261
2	-1.14e-3	0.0916
4	-0.034	8.47e-4
3	-1.32e-6	-0.781
2	2.08e-5	0.166
1	3.73e-4	0.496
32	2.42e-5	0
29	3.42e-5	0

Table 8.16 Derivative Information about \mathbf{x}^1

Similarly, objective functions are established about the new improved solutions \mathbf{x}^2 and \mathbf{x}^3 . Each predicted point within the neighbourhood of \mathbf{x}^3 either violates the spurious trip constraint or has an inferior system unavailability to the current best. Design \mathbf{x}^3 is, therefore, deemed the most optimal design and assumes an unavailability of 7.57×10^{-4} .

8.3.2.2 Results Using the Quadratic Objective Function with Cross Terms

The grid-sampling algorithm was applied to 8 initial designs using the quadratic objective function with cross terms. The initial designs were chosen to be the same as those tested using the linear scheme to enable direct comparisons. Table 8.17 specifies each initial design in bold and shows the design produced at each subsequent iteration for each test.

Iteration	E	K_1/N_1	H	K_2/N_2	V	P	θ_1	θ_2	Q_{SRS}	F_{SRS}	Cost	MDT
Test 1												
0	1	2/2	1	1/1	1	1	40	50	3.95e-4	0.419	882	101.7
Test 2												
0	1	1/1	1	1/1	2	1	45	34	3.27e-3	0.716	762	90.3
1	0	1/2	2	1/2	1	1	42	31	1.05e-3	0.812	902	123.8
2	0	1/1	2	1/3	2	1	33	27	8.27e-4	0.977	802	129.2
3	0	1/2	2	1/2	2	1	34	26	7.23e-4	0.977	802	129.6
Test 3												
0	1	2/3	1	1/1	2	1	60	25	3.64e-3	0.585	782	86.3
Test 4												
0	2	1/1	0	0/0	2	1	50	0	4.85e-2	0.463	721	70.7
1	2	1/2	0	0/0	1	1	38	0	3.23e-2	0.428	841	108.1
2	1	1/2	0	0/0	1	1	26	0	2.22e-2	0.420	571	118
3	0	1/3	0	0/0	1	1	16	0	1.43e-2	0.411	301	126.7
Test 5												
0	2	1/1	1	1/1	2	1	63	33	4.41e-3	0.807	982	86.1
Test 6												
0	0	1/1	2	3/4	1	2	45	35	2.23e-3	0.899	872	120.8
1	0	1/1	2	2/4	2	1	34	27	8.53e-4	0.586	822	129.4
2	0	1/2	2	3/4	2	1	32	29	7.62e-4	0.716	842	129.7
3	0	1/3	2	2/3	2	1	33	28	7.57e-4	0.847	842	129.9
Test 7												
0	0	2/3	1	1/3	1	1	40	40	2.51e-3	0.673	672	85.8
Test 8												
0	2	1/1	1	1/1	2	1	40	50	4.2e-3	0.807	982	108.2

Table 8.17 Results from each Iteration of Tests 1 to 8 (Cross Quadratic)

8.3.3 Discussion Using Other Formulations of the Objective Function

The pure quadratic formulation of the objective function attains very similar results to that of the linear approximation. Its performance is inferior regarding tests 1 and 3. An inability to predict an accurate improvement over relatively poor designs is portrayed, where the linear approach enabled further exploration in each case. In contrast, the quadratic approach showed slightly better performance in tests 5 and 6. Test 5 achieves a slightly better use of maintenance effort. The resulting design in test 6 differs only in that the number of pressure transmitters for subsystem 1 is 2 as

opposed to 3. The difference between the two system unavailabilities is negligible but the design resulting from the quadratic approach is both cheaper and less prone to spurious trips. Test 7 is unusual in that the quadratic objective function substantially improves performance. A design vector with an unavailability of $7.62e-4$ results.

Both the linear and pure quadratic formulations proved to be more effective than the cross term objective function. Tests 1, 3 and 5 showed an inability to proceed to a second iteration. The remaining tests gave identical results to those of the linear approximation.

In the main, the results show that incorporation of the quadratic terms adds little accuracy to the objective function. This can be explained by the way that the system unavailability is affected by changes in redundancy levels. Lower order minimal cut sets consist of components most likely to cause system failure. Adding an extra level of redundancy increases the order of these minimal cut sets by one. Consequently, other components in the system will most likely provide more significant contributions to system failure. Further increasing the redundancy levels does not significantly reduce the system unavailability. Therefore, the rate of improvement predicted by the objective function does not match that actually achieved, even when quadratic terms are included.

Formulation of the linear objective function is the easiest to achieve. The pure quadratic form requires a few extra evaluations, yet no further BDD evaluations are enforced. The complexity and computer effort exerted by the cross terms approach is by far the greatest.

In conclusion, the difference obtained in both computer effort and effectiveness between the linear and the pure quadratic objective function is negligible. But what appears to be of vital importance to the success of the scheme is the proximity of the initial design to the optimal.

CHAPTER 9

IMPLEMENTING THE OPTIMISATION PROCEDURE TO A FIREWATER DELUGE SYSTEM

9.1 Introduction

The optimisation procedure using GA's has been demonstrated on a relatively simple High-Integrity Protection System. However, many systems are much more complex and have a much larger number of design variables. As a result this chapter is concerned with an implementation of the optimisation process to a larger, more detailed system, to test the effectiveness of the GA approach in these circumstances.

This chapter first introduces and describes the features of a deluge system design typical of those used on offshore platform. Attention is then given to analysis of the safety system. Subsequently, the safety system analysis is incorporated within a GA optimisation procedure.

9.2 Description of the Firewater Deluge System

To test the effectiveness of the optimisation process in dealing with larger design problems it has been applied to a Firewater Deluge System (FDS) on an offshore platform. The basic features of the deluge system are shown in figure 9.1. Its function is to supply, on demand, water and foam at a controlled pressure to a specific area on the platform protected by a deluge system. As such, the FDS comprises of a deluge skid, firewater pumps and their associated equipment and ringmains, Aqueous Film-Forming Foam (AFFF) pumps and their associated equipment and ringmains. The following section describes the systems further.

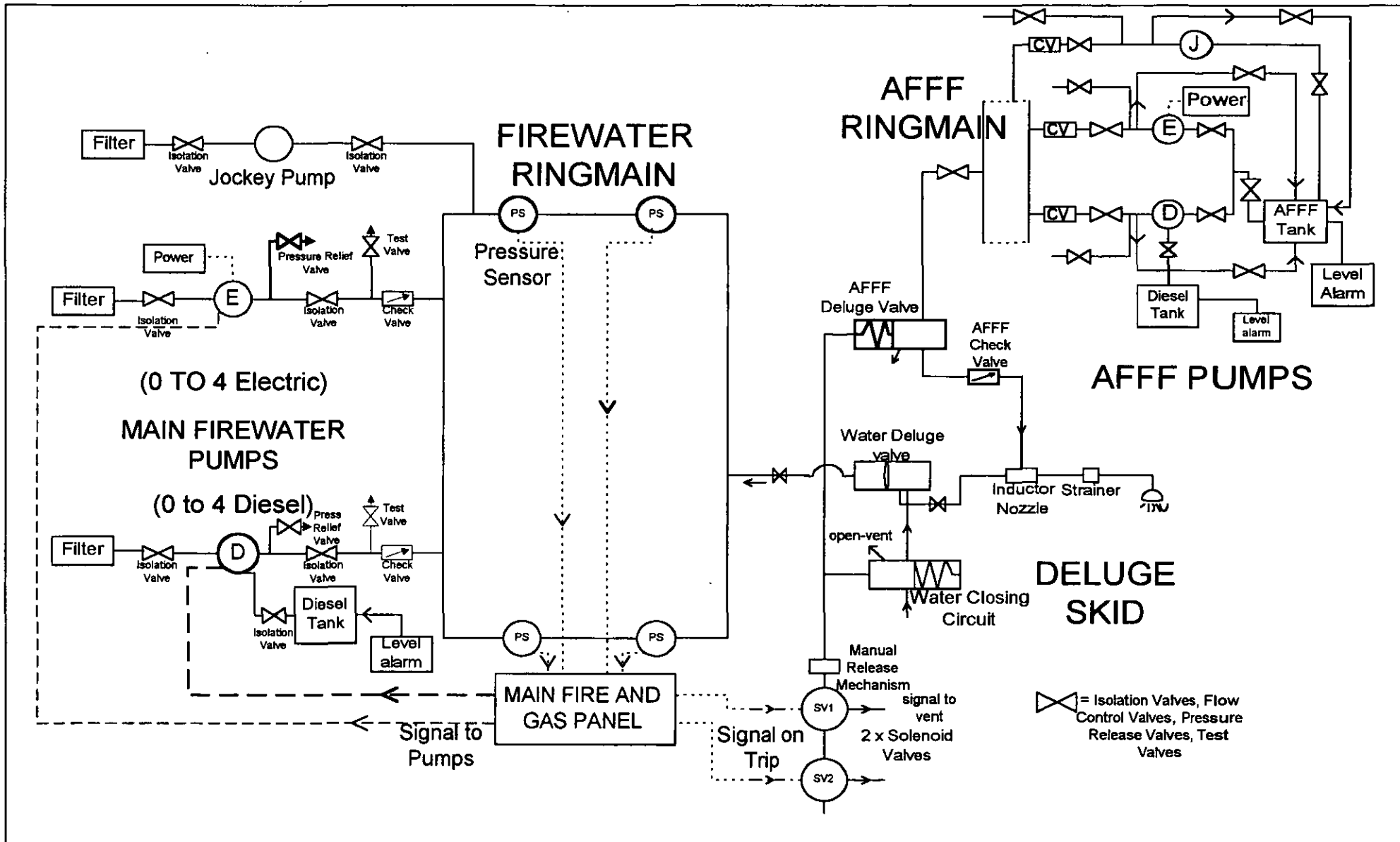


Figure 9.1 The Firewater Deluge Skid

9.2.1 The Deluge System

The deluge valve set including all associated equipment is mounted on a fabricated steel framework called a skid. Skids are situated on the processing platform where an incident can occur and its associated equipment act to spray water onto the affected area.

The deluge valve set comprises three main elements: the main distribution line, a water closing circuit and a control air circuit. Upon receipt of a signal from the Main Fire and Gas Panel (MFGP), the solenoid valves are de-energised and open thus releasing air pressure from the control air circuit. The air pressure drop allows the valmatic release valve to open, and water from the water closing circuit runs to drain. This causes the pressure on the deluge valve diaphragm to fall. When the pressure on the diaphragm has fallen sufficiently, the firewater main pressure acting on the underside of the deluge valve clack overcomes the load imposed by the diaphragm, allowing water to flow into the distribution pipes, through the nozzles and onto the hazard.

The system may also be operated manually by opening the system local manual release valve on the skid. This allows air to escape from the control air circuit and the system operates as described above.

The deluge valve set is also fitted with an AFFF supply line. Instrument air pressure maintains the valmatic release valve and AFFF valve closed. When the air pressure drops in the control air circuit, due to the solenoid valves being de-energised (the same components as those used to activate the water deluge valve), the AFFF valve and valmatic release valve open simultaneously. As the water flows through the foam inductor in the main distribution line, foam concentrate is induced from the AFFF line via the foam proportioner. The solution of water and approximately 3% foam then feed into the distribution network, through the nozzles and onto the hazard.

Events considered in the reliability assessment of the deluge skid are specified in table 9.2. Event type 'HE' states that the event is a human error. In contrast 'CO' denotes that the event is a component failure. Components are of 'wear-out' or 'non

wear-out' type, denoted by 'W' and 'NW' respectively. Preventative maintenance is only carried out on components of wear-out type. The tendency for corrosion build-up in the system was noted. For this reason corrosion resistant components were introduced, where 'o' and 'n' correspond to non-corrosion resistant and corrosion resistant materials respectively. The failure and repair data, maintenance effort and costs associated with each component are specified in table 9.3. The key to notation in table 9.3 is expressed in table 9.1. As can be seen from table 9.3, events due to human error require only specification of the probability of occurrence. SV1, SV2 and WVRF are the only components that fail spuriously.

Notation	Description
λ_D	Dormant failure rate
λ_S	Spurious failure rate
τ_D	Dormant mean time to repair
τ_S	Spurious mean time to repair
H_T	Number of hours manual work required to test the component
C_{HT}	Cost per hour of manual work to test the component
C_R	Number of hours manual work required to repair the component
C_{HR}	Cost per hour of manual work to repair failure (dormant or spurious)
C_{SR}	Cost of spares for each repair carried out (dormant or spurious)
H_P	Number of hours manual work required to carry out preventative maintenance
C_{SP}	Cost of spares each time preventative maintenance is undertaken
C_{HP}	Cost per hour of manual work to carry out preventative maintenance
N_S	No. of spares stored
C_S	Storage costs per component
C_I	Initial cost

Table 9.1 Notation Used

Not- ation	Event Description	Event Type	Rates
SI	Failure of MFGP to correctly select and send a close signal to the solenoid valve.	CO	NW
WBS	Strainer, located upstream of the water deluge valve, blocked.	CO	NW
WBN (old)	Deluge nozzle on the waterspray system blocked, old type material.	CO	NW
WBN (new)	Deluge nozzle on the waterspray system blocked, new type material.	CO	NW
WIVB	Blockage of the locked open butterfly valve, one upstream and one downstream of the water deluge valve.	CO	NW
WIVO	Operator leaves the normally locked open butterfly valve in the shut position (one upstream and one downstream of the water deluge valve).	HE	-
WV1	Water deluge valve type 1 fails to open.	CO	NW
WV2	Water deluge valve type 2 fails to open.	CO	NW
WV3	Water deluge valve type 3 fails to open.	CO	NW
MRM	Manual release mechanism fails to dump instrument air.	CO	NW
SV	Solenoid activated valve fails to dump instrument air on receipt of the signal from the MFGP. (there are 2 of Solenoid Valves).	CO	NW
WVR (old)	Valmatic relief valve sticks closed on activation, old type material.	CO	NW
WVR (new)	Valmatic relief valve sticks closed on activation, new type material.	CO	NW
AIVB	Normally locked open butterfly valve on AFFF distribution line blocked (only one isolation valve on AFFF line).	CO	NW
AIVC	Operator leaves the normally locked open butterfly valve on the AFFF distribution line in the shut position.	HE	-
AINB (old)	The foam supply into the firewater distribution line is blocked by the inductor nozzle, old type material.	CO	NW
AINB (new)	The foam supply into the firewater distribution line is blocked by the inductor nozzle, new type material.	CO	NW

Table 9.2 Events Involved in the Reliability Assessment of the Deluge Skid

Not-ation	Description	Event Type	Rates
ACVB	The check valve in the AFFF injection line is blocked.	CO	NW
AV1	AFFF deluge valve type 1 fails to open on demand.	CO	NW
AV2	AFFF deluge valve type 2 fails to open on demand.	CO	NW
AV3	AFFF deluge valve type 3 fails to open on demand.	CO	NW

Table 9.2continued.

Event	λ_D	τ_D	λ_S	τ_S	H_T	C_{HT}	C_R	C_{HR}	C_{SR}	N_S	C_S	C_I
SI	2e-7	6e-6										
WBS	2.8e-5	1.2e-5			2	30	12	30	50	4	75	100
WBN (o)	3.0e-5	1.2e-5			2	30	12	30	100	3	300	1000
WBN (n)	5e-6	1.2e-5			2	30	12	30	300	3	300	3000
WIVB	1.8e-6	1.8e-6			2	30	18	30	200	2	300	400
WIVO	Q = 0.01											
WV1	4.0e-5	1.8e-5			2	30	18	30	200	2	200	400
WV2	3.5e-5	1.8e-5			2	30	18	30	250	2	200	500
WV3	2.8e-5	1.8e-5			2	30	18	30	300	2	200	600
MRM	1.0e-5	1.2e-5			2	30	12	30	300	1	300	600
SV1	3e-6	1.2e-5	3e-6	1.2e-5	2	30	12	30	200	2	300	400
SV2	2.0e-5	1.2e-5	2.0e-5	1.2e-5	2	30	12	30	125	2	300	250
WVR (o)	5e-6	1.2e-5	5e-6	1.2e-5	2	30	12	30	300	1	300	600
WVR (n)	2e-6	1.2e-5	2e-6	1.2e-5	2	30	12	30	450	1	300	900
AIVB	1.8e-5	1.8e-5			2	30	18	30	200	2	300	400
AIVC	Q = 0.01											
AINB (o)	3.0e-5	1.2e-5			2	30	12	30	100	3	300	1000
AINB (n)	5e-6	1.2e-5			2	30	12	30	300	3	300	3000
ACVB	2.5e-5	1.8e-5			2	30	18	30	300	2	300	600
AV1	4.0e-5	1.8e-5			2	30	18	30	150	2	150	300
AV2	3.5e-5	1.8e-5			2	30	18	30	200	2	150	400
AV3	2.8e-5	1.8e-5			2	30	18	30	250	2	150	500

Table 9.3 Data Associated with Events in Table 2

9.2.2 Firewater Supply and Distribution System

The deluge systems are connected to a pressurised ringmain network. The ringmain pressure is maintained by a jockey pump drawing water from the sea. Falling pressure is detected by the pressure transducers, which subsequently send a signal to the MFGP. In turn, the MFGP activates the firewater pumps to supply water direct from the sea at sufficient pressure to meet the deluge requirements. Pumps not needed remain in inactive standby.

It is possible to start each pump manually, both locally and at the fire control panel.

The fire pumps are arranged in two sets, one set being powered from the main electric power plant and the other from their own dedicated diesel engines. The diesels have a day tank sized for a 24 hour supply. The tank has a low level alarm fitted, alarming in the Central Control Room.

Events considered in the reliability assessment of the firewater supply and distribution are specified in tables 9.4, 9.5 and 9.6. Table 9.4 lists the events associated with each fire pump and as such, constitute a single pump line (see figure 9.1). The electricity supply (ESF) is global to all electric pumps and a single diesel tank supplies all fitted firewater diesel pumps. Events associated with the electric and diesel supply, in addition to the different pump types available are specified in table 9.5. Table 9.6 considers failure events associated with the distribution network.

Not- ation	Description	Event Type	Rates
FB	The pump, which includes seawater filter, is blocked by debris.	CO	NW
IVB	Firewater pump isolation valve being blocked. The butterfly isolation valve operates on the header from pump to ringmain.	CO	NW
IVC	The firewater pump isolation valve is left closed after pump test.	HE	-
PRVO	Pressure relief valve on header from pump to ringmain fails open.	CO	NW
SVO	The flow control valve fails open on demand. (used to dump excess flow from the pumps to ringmain)	CO	NW
DVO	Test line, used to dump flow from firewater pumps overboard during test, is left open after completion.	HE	-
CVB	Check valve on header between the pump and ringmain blocked.	CO	NW

Table 9.4 Events associated with each firewater pump

Not- ation	Description	Event Type	Rates
ESF	Failure of electricity supply to electric driven firewater pumps.	CO	NW
DIVB	Diesel engine supply is blocked.	CO	NW
DIVC	Diesel supply is inadvertently left isolated after maintenance.	HE	-
LAF	Diesel tank level switch fails to signal flow level to control room.	CO	NW
OAF	Operator fails to notice or misinterprets the low level tank alarm.	HE	-
E 100	Failure of electric pump with 100% capacity.	CO	W
D 100	Failure of diesel pump with 100% capacity.	CO	W
E1 50	Failure of electric pump type 1 with 50% capacity.	CO	W
E2 50	Failure of electric pump type 2 with 50% capacity.	CO	W
D1 50	Failure of diesel pump type 1 with 50% capacity.	CO	W
D2 50	Failure of diesel pump type 2 with 50% capacity.	CO	W
E1 33	Failure of electric pump type 1 with 33 1/3% capacity.	CO	W
E2 33	Failure of electric pump type 2 with 33 1/3% capacity.	CO	W
D1 33	Failure of diesel pump type 1 with 33 1/3% capacity.	CO	W
D2 33	Failure of diesel pump type 2 with 33 1/3% capacity.	CO	W
E/DM	Probability of maintenance being carried out on a pump.	-	-

Table 9.5 Failure Events Associated with the Firewater Pumps

Not-ation	Description	Event Type	Rates
FSU	Failure of fire pump selector unit to initiate start of the standby pump in sequence, on detection of failure of duty pump/pumps to restore ringmain pressure.	CO	NW
OE	Designated duty pump/pumps inadvertently left in a mode other than auto start at the end of the test.	HE	-
PBF	Manual push button in the control room failing to initiate pump start when pressed.	CO	NW
PT1	Failure of ringmain low pressure sensor type 1 to indicate low ringmain pressure.	CO	NW
PT2	Failure of ringmain low pressure sensor type 2 to indicate low ringmain pressure.	CO	NW
PT3	Failure of ringmain low pressure sensor type 3 to indicate low ringmain pressure.	CO	NW

Table 9.6 Failure Events Associated with the Distribution Network

Associated component data for the events specified in tables 9.4, 9.5 and 9.6 is given in tables 9.7, 9.8, and 9.9 respectively. (The key to notation in these tables is supplied in table 9.1). Note that the pumps are of wear-out type and as such, the Weibull distribution is used (introduced in section 2.4.4.2). Hence, the value of the Weibull parameters, β and η , are specified. In addition, PBF is a component failure with its probability of occurrence specified directly.

Event	λ_D	τ_D	H_T	C_{HT}	C_R	C_{HR}	C_{SR}	N_S	C_S	C_I
FB	2.8e-5	1.2e-5	2	30	12	30	50	4	150	100
IVB	1.8e-5	1.8e-5	2	30	18	30	400	2	300	400
IVC	Q = 0.01									
PRVO	1.2e-5	1.8e-5	2	30	18	30	250	2	300	500
SVO	1.8e-5	2.4e-5	2	30	24	30	400	2	300	800
DVO	Q = 0.01									
CVB	2.5e-5	1.8e-5	2	30	18	30	300	2	300	500

Table 9.7 Data associated with the Events in Table 4

Event	λ_D	τ_D	H_T	C_{HT}	C_R	C_{HR}	C_{SR}	H_P	C_{HP}	C_{SP}	N_S	C_S	C_I
ESF	5.0e-5	2e-6	2	45	2	45	200						1000
DIVB	3.0e-5	8e-6	2	30	8	30	200				2	300	400
DIVC	Q = 0.01												
LAF	3.0e-5	6e-6	2	30	6	30	100				2	200	200
OAF	Q = 0.01												
Event	β	η	H_T	C_{HT}	C_R	C_{HR}	C_{SR}	H_P	C_{HP}	C_{SP}	N_S	C_S	C_I
E 100	2	16667	2	30	72	30	1500	72	30	300	1	1000	3000
D 100	2	14035	2	30	72	30	1450	72	30	290	1	1000	2900
E1 50	3/2	22857	2	30	48	30	900	48	30	180	2	900	1800
E2 50	3/2	26667	2	30	48	30	1000	48	30	200	2	900	2000
D1 50	3/2	20000	2	30	48	30	750	48	30	150	2	900	1500
D2 50	3/2	22857	2	30	48	30	900	48	30	180	2	900	1800
E1 33	3/2	32000	2	30	36	30	600	48	30	120	2	800	1200
E2 33	3/2	40000	2	30	36	30	700	48	30	140	2	800	1400
D1 33	3/2	28571	2	30	48	30	500	48	30	100	2	800	1000
D2 33	3/2	33333	2	30	48	30	550	48	30	110	2	800	1100
E/D M	Q = 1/25												

Table 9.8 Data Associated with the Events in Table 5

Event	λ_D	τ_D	λ_S	τ_S	H_T	C_{HT}	C_R	C_{HR}	C_{SR}	N_S	C_S	C_I
FSU	8e-6	2.4e-5			1	45	24	45	200	1	200	2000
OE	Q = 0.01											
PBF	Q = 0.01											
PT1	7e-6	4e-6	7e-6	4e-6	1	45	4	45	50	2	100	500
PT2	1.4e-5	4e-6	1.4e-5	4e-6	1	45	4	45	20	2	100	200
PT3	2.1e-5	4e-6	2.1e-5	4e-6	1	45	4	45	10	2	100	100

Table 9.9 Data Associated with Events in Table 6

9.2.3 AFFF Supply and Distribution

The foam concentrate is stored in a stainless steel tank and is distributed through a stainless steel ringmain network. The tank has a low level alarm fitted, alarming in the Central Control Room.

The foam system is kept at approximately the same pressure as the firewater system by a continuously running air driven jockey pump. The AFFF pumps are either motor driven, supplied from the platform power plant, or diesel driven. AFFF pumps start automatically when any firewater pump starts to supply foam at sufficient pressure to meet design requirements. Pumps not needed remain in standby. The diesel supply to the firewater diesel pumps is separate from that of the AFFF diesel pumps.

Failure Events on each AFFF pump line are identical to those described in table 9.4, where the associated component data is supplied in table 9.7. Further events involved in the AFFF supply and distribution are described in table 9.10.

Not- ation	Description	Event Type	Rates
ATIVB	Normally locked open ball valve on AFFF tank outlet blocked.	CO	NW
ATIVC	AFFF supply left isolated after maintenance.	HE	-
AESF	Failure of electric supply to the electric driven AFFF pumps.	CO	NW
ADIVB	Normally locked open ball valve on diesel tank outlet blocked.	CO	NW
ADIVC	Diesel supply to AFFF pumps left isolated after maintenance.	CO	NW
ALAF	Diesel tank level switch fails to signal low level to control room.	CO	NW
AOAF	Operator fails to notice AFFF diesel tank low level alarm.	HE	
AE 100	Failure of AFFF electric pump with 100% capacity.	CO	W
AD 100	Failure of AFFF diesel pump with 100% capacity.	CO	W
AE 50	Failure of AFFF electric pump with 50% capacity.	CO	W
AD 50	Failure of AFFF diesel pump with 50% capacity.	CO	W
AE/DM	Probability of maintenance being carried out on an AFFF pump.	CO	W

Table 9.10 Failure Events Associated with AFFF Supply and Distribution

Data associated with each of the events in table 9.10 is given in table 9.11. (Key to notation in table 9.11 is supplied in table 9.1.) Note, again, that the AFFF pumps are of wear-out type. As such, the Weibull distribution is used and Weibull parameters, β and η , are specified.

Event	λ_D	τ_D	H_T	C_{HT}	C_R	C_{HR}	C_{SR}	H_P	C_{HP}	C_{SP}	N_S	C_S	C_I
ATIVB	1.8e-5	1.8e-5	2	30	18	30	200				2	300	400
ATIVC	Q = 0.01												
ESF	5.0e-5	2e-6	2	45	2	45	200						1000
DIVB	3.0e-5	8e-6	2	30	8	30	200				2	300	400
DIVC	Q = 0.01												
LAF	3.0e-5	6e-6	2	30	6	30	100				2	200	200
OAF	Q = 0.01												
Event	β	η	H_T	C_{HT}	C_R	C_{HR}	C_{SR}	H_P	C_{HP}	C_{SP}	N_S	C_S	C_I
AE 100	2	16667	2	30	72	30	750	72	30	150	1	800	1500
AD 100	2	14035	2	30	72	30	725	72	30	145	1	800	1450
AE 50	3/2	22857	2	30	48	30	450	48	30	90	2	600	900
AD 50	3/2	20000	2	30	48	30	375	48	30	75	2	600	750
AE/D M	Q = 1/25												

Table 9.11 Data Associated with Events in Table 10

9.2.4 Design Variables

As regards the FDS it is necessary to determine values for the design variables that represent the following:

Designer Options	Design Variable
• How many pressure transmitters on the ringmain (0,1,2,3,4)?	N
• How many pressure transmitters are required to trip?	K
• Which of three possible pressure transmitters to select?	P
• How many firewater pumps are required (1 to 8)?	F
• Of these firewater pumps how many are electrically powered (0 to 4)? (The rest being diesel driven).	F_E
• What percentage capacity to choose for the firewater pumps (100%, 50% or 33 1/3%)?	F_P
• Which of two possible pump types to select? (For the 50% and 33 1/3% capacity pumps only)	F_T
• How many AFFF pumps are required (1 to 4)?	A
• Of these AFFF pumps how many are electrically powered (0,1,2)? (The rest being diesel driven).	A_E
• What percentage capacity to choose for the AFFF pumps (100%, 50%)?	A_P
• Which of three possible water deluge valve to select?	W
• Which of three possible AFFF deluge valve to select?	D
• Which of two possible materials to use for certain components (as specified in table 9.2)?	C
• Maintenance test interval for the firewater and AFFF pump system (1 to 28 days)?	θ_P
• Maintenance test interval for the ringmain (1 to 24 weeks)?	θ_R
• Maintenance test interval for the deluge skid (3 to 18 months in 3 monthly intervals only)?	θ_D
• Preventative maintenance on components of wear-out type (3 to 18 months in 3 monthly intervals only)?	θ_{PM}

It should be noted that all pumps in the firewater system are to be of the same capacity, as are all pumps in the AFFF system. In addition, electric and diesel pumps

of 100% capacity in the firewater system are of one type only, as are both 100 and 50% pumps in the AFFF system.

The design costs a certain amount to build, termed its initial cost. When in situ the FDS must be tested at regular intervals. Any failures found must be repaired. In addition, certain components are of wear-out type. These must undergo preventative maintenance at regular intervals. Knowledge of components comprising the FDS enable predictions to be made about the expected cost of system testing, repairs and maintenance effort. The initial cost plus cost of maintaining the system yield the life cycle cost. The choice of design is not unrestricted. Limitations have been placed on the design such that:

- 1) Total life cycle costs must be less than an average of 125000 units per year (i.e. initial cost plus total cost of maintenance, part 4 stated below).
- 2) Total cost of testing the system must be less than 20500 units per year.
- 3) Total cost of preventative maintenance effort must be less than 13500 units per year.
- 4) Total cost of maintenance effort must be less than 44000 units. (i.e. cost of corrective maintenance due to repair of dormant and spurious failure plus 2 and 3 stated above).
- 5) The number of times that a spurious system shutdown occurs would be unacceptable if it were to occur on average more than 0.75 times per year.

9.3 Safety System Analysis

The FDS is a primary safety system on the platform designed to mitigate the consequences of jet and pool fires, in addition to reducing overpressures in the event of an explosion. Failure in the event of a hydrocarbon release could result in fatalities. It is imperative, therefore, that the FDS works when the demand arises. Applying the same principles as in the HIPS optimisation, the objective is to minimise system unavailability whilst giving consideration to the available resources.

9.3.1 Evaluate the System Unavailability

No explicit objective function exists. A fault tree using house events is, therefore, constructed to model each possible design alternative. This fault tree is then converted to its BDD equivalent and integrated within the GA source code.

9.3.1.1 Construction of the System Unavailability Fault Tree

The top event of the fault tree representing the causes of system unavailability is defined as the 'Firewater Deluge System Fails to Activate on Demand'. This top event will occur if either the firewater or AFFF pump mechanisms are not activated, the firewater or AFFF pumps themselves fail or the water or foam deluge systems fail, as indicated in figure 9.2. The FDS system unavailability fault tree construction is described via development of each of these sub-events in turn.

Failure to Initiate Pump Mechanisms (Firewater and AFFF)

Failure to initiate the firewater and AFFF pump mechanisms occurs if both automatic and manual intervention fails. 'Manual Start' fails if either the push button on the MFGP fails or if the operator fails to push the button. 'Auto Start' fails if either the fire pump selector unit fails or the low pressure sensing on the firewater ringmain fails. This causal relationship is portrayed in figure 9.2. Failure of the low pressure sensing depends on the number of pressure transmitters fitted (N) and the number of pressure transmitters required to trip the system (K). House events are used to model each possible design alternative in a similar manner to that described in section 5.2.1.

Failure of the Firewater Pumps and Lines

The FDS fails to supply sufficient water to the ringmain if failure of the firewater pump mechanisms or lines means that the pumps cannot supply 100% required pressure. Events resulting in this scenario depend on the values assigned to the

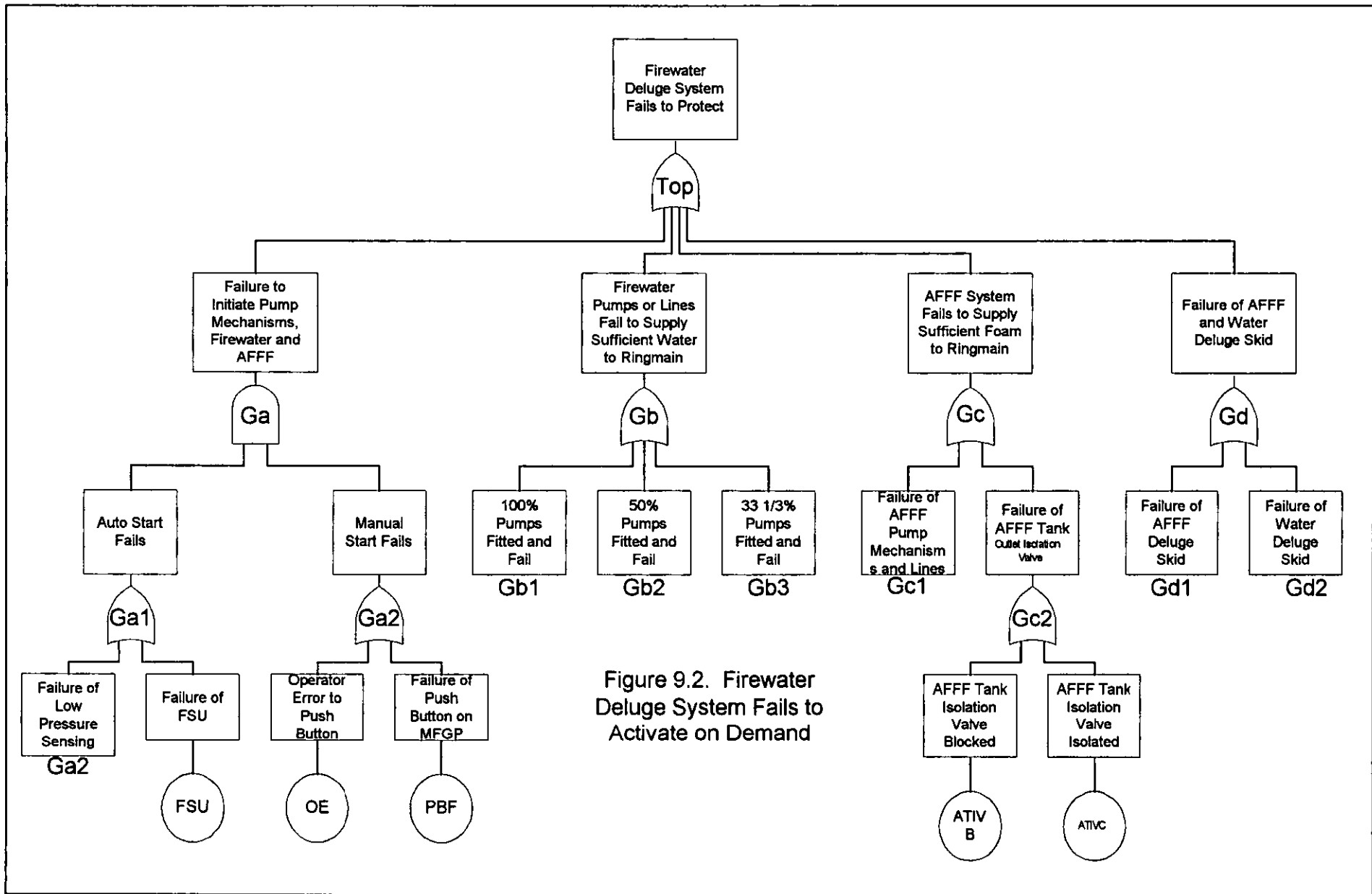


Figure 9.2. Firewater Deluge System Fails to Activate on Demand

variables F_P , F , F_E and F_T . As illustrated in figure 9.2, 'Failure of Firewater Pumps or Lines' will occur if either the firewater pumps are of 100% capacity and fail, if firewater pumps are of 50% capacity and fail or if firewater pumps are of 33 1/3% capacity and fail.

1 to 8 100% capacity firewater pumps can be fitted. As such 8 sub-events are developed from gate 'Gb1', where the left most branch considers that '1 100% pump only is fitted and fails', the next that '2 100% pumps are fitted and fail', and so on. These branches are further developed. Consider the event '2 100% pumps are fitted and fail'. Of the 2 pumps fitted, each possible combination of electric and diesel pumps must be modelled and further developed to specify the events that will trigger failure resulting from the specific combination. This causal relationship is portrayed in figure 9.3. H2P is a house event that is set to true if $F = 2$. HE2, HE1, HE0 are true if $F_E = 2$, $F_E = 1$ or $F_E = 0$ respectively.

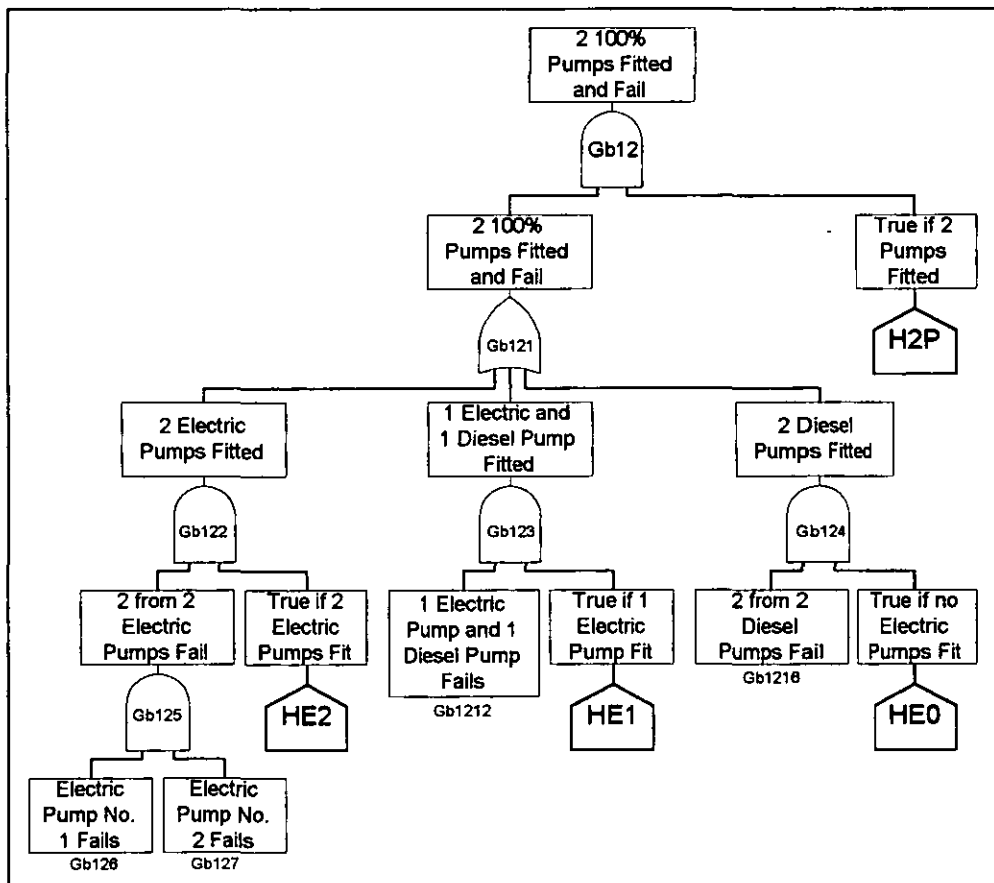


Figure 9.3 2 100% Pumps Fitted and Fail

Pump failure occurs if the pump itself fails or if components on the pump line fail. Events that will trigger 'Electric Pump no. 1 Fails' are depicted in figure 9.4. (The abbreviations in figure 9.4 are described in table 1. The appended 'E1' specifies each component to occurrence in electric pump line number 1 alone).

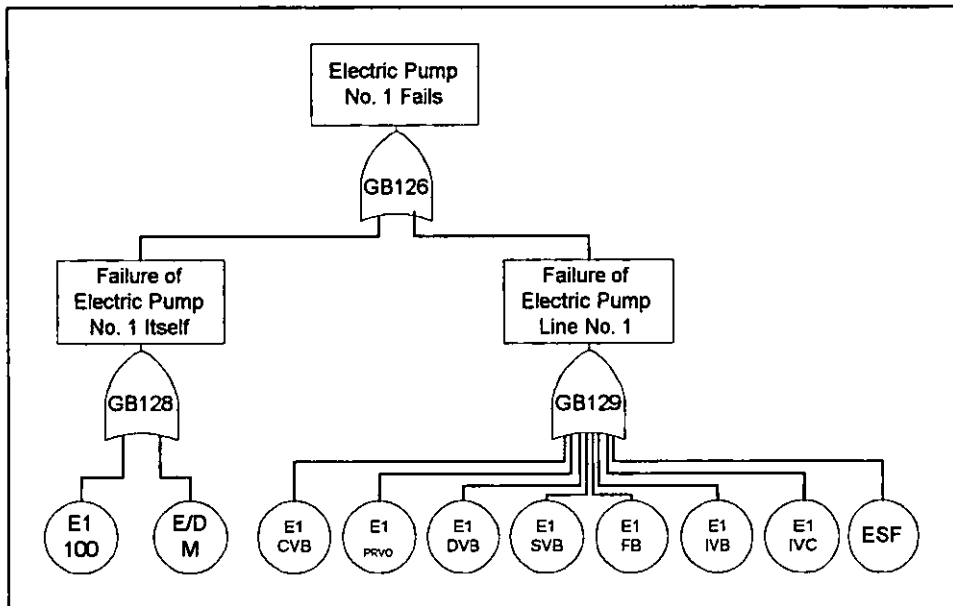


Figure 9.4 Electric Pump No. 1 Fails

The event '50% Pumps are Fitted and Fail', gate 'Gb2', is developed in a similar manner to that described above. It differs in that at least 2 pumps must be fitted to ensure 100% pressure is attained. As such, the sub-event '1 50% pump is fitted and fails' is infeasible. Developing the 50% branch is a little more complex as regards the combinations of pumps resulting in overall pump failure. Consider, for example, the next two levels in the tree structure below the event '3 50% Pumps Fail' illustrated in figure 9.5.

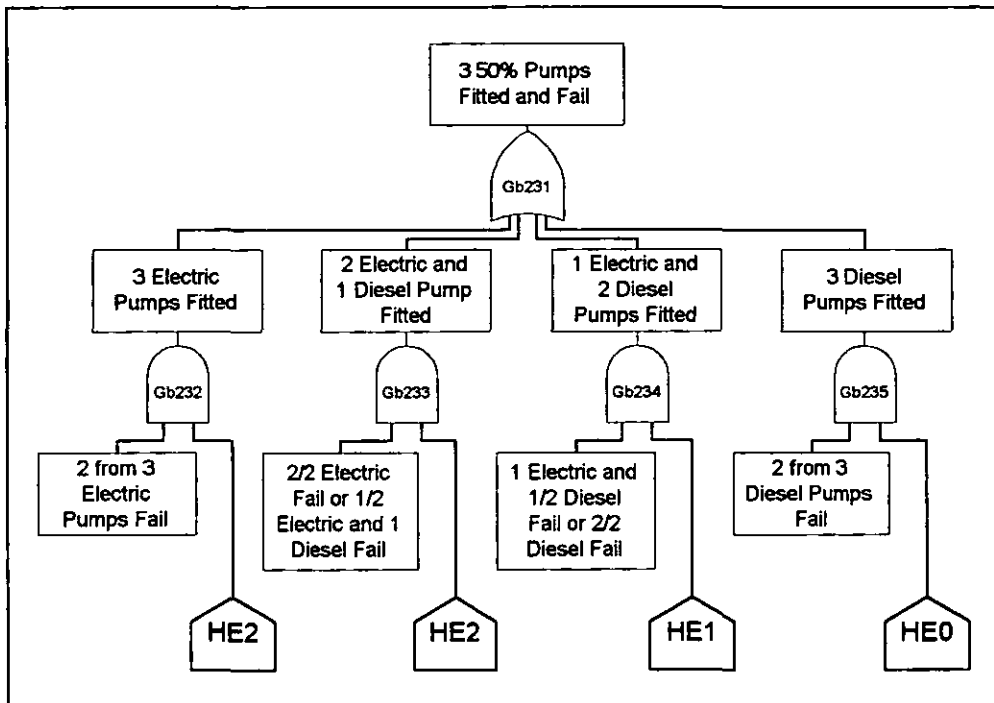


Figure 9.5 3 50% Pumps Fail

The combinations become still more complex when developing '33 1/3% Pumps are Fitted and Fail'. However, construction follows a similar pattern. As regards the 33 1/3% branch at least 3 pumps must be fitted to ensure 100% pressure is attainable and as such only 6 sub-events constitute the level immediately below gate 'Gb3'.

Failure of the AFFF Pumps and Lines

The AFFF pump system fails to supply sufficient foam to the ringmain as a result of failure to the AFFF pump mechanisms or lines or isolation of the AFFF tank, as shown in figure 9.1. The tree structure below gate 'Gc1' is similar to that of the firewater pump system. Pumps of 33 1/3% capacity are, however, not considered.

Failure of the AFFF or Water Deluge Skid

'Failure of the AFFF or Water Deluge Skid' occurs if either 'Failure of the Water Deluge Skid' or 'Failure of the AFFF Deluge Skid' occur. Considering the former

event. 'Failure of the Water Deluge Skid' occurs if either of the waterspray isolation valves fail, the strainer or nozzle becomes blocked or the deluge valve fails to open. Developing further 'The Water Deluge Valve Fails to Open' requires consideration of events that restrict activation of the deluge valve or failure of the deluge valve itself. These two scenarios are related using OR logic. 'Failure to Activate the Water Deluge Valve' occurs if the signal to the solenoids fails, both fitted solenoid valves remain energised or the valmatic release valve fails. 'Failure of the AFFF Deluge Skid' is developed in a similar manner. It differs primarily in that the blocked nozzle is replaced by blockage of the inductor nozzle and the strainer by a blocked AFFF check valve in the sequence of events described above.

The causal relationship describing each aspect of the FDS unavailability fault tree is depicted in full in Appendix V.

9.3.1.2 Converting the System Unavailability Fault Tree to a BDD

Construction of the fault tree in FAULTTREE+ exceeds the limit for both the number of gates and the number of basic events. In addition, BADD is unable to convert some larger structures to their BDD equivalent. For this reason, the fault tree is split into 17 separate parts. This is possible due to the fact that each part has independent events. The 'ats' file created from each separate part is read into BADD and the non-minimal ite table describing the corresponding BDD structure produced. Table 9.12 describes the fault tree structure of each separate part with reference to figure 1 and states the size of the resulting BDD. (The abbreviation FPF & F denotes Firewater Pumps Fitted and Fail).

Name of 'ats' File	Top Event	Top Event Description	No. of gates	No. of Events	BDD Number	No. of ite Statements
INIT	Ga	Failure to activate pump mechanisms	29	61	1	723
AFFF	Gc	Failure of AFFF pump mechanisms and lines	62	83	2	6046
DEL	Gd	Failure of AFFF and water delude skids	36	33	3	157
B100	Gb1	100% FPF & F	88	89	4	30574
B502	Gb22	2, 50% FPF & F	55	32	5	467
B503	Gb23	3, 50% FPF & F	72	46	6	2493
B504	Gb24	4, 50% FPF & F	89	60	7	8159
B505	Gb25	5, 50% FPF & F	88	64	8	9068
B506	Gb26	6, 50% FPF & F	87	62	9	6836
B507	Gb27	7, 50% FPF & F	86	54	10	4357
B508	Gb28	8, 50% FPF & F	85	45	11	2420
B303	Gb33	3, 33 1/3% FPF & F	72	42	12	849
B304	Gb34	4, 33 1/3% FPF & F	89	59	13	5582
B305	Gb35	5, 33 1/3% FPF & F	88	66	14	8694
B306	Gb36	6, 33 1/3% FPF & F	87	67	15	8513
B307	Gb37	7, 33 1/3% FPF & F	86	58	16	6102
B308	Gb38	8, 33 1/3% FPF & F	85	46	17	3366

Table 9.12 A Description of Each Separate Tree Structure and Associated BDD

BDD's 4 to 17 are all sub-events comprising the level directly below 'Failure of the Firewater Pump Mechanisms or Lines' (see figure 9.1). To model a particular design only one of these branches will be relevant. For example, for a design with $F = 5$ and a firewater pump capacity of 50% the tree structure termed B505 is used to determine the probability that system failure occurs as a result of firewater pump failure.

Consequently, probability values are derived for each event that can directly cause the top event. As such, each of these events can be dealt with in an identical manner to basic events and evaluation of the probability of top event occurrence becomes a very

simple process. This is illustrated in figure 9.6, where the event ‘Firewater Pumps and Lines Fail to Supply Sufficient Water to the Ringmain’ accounts for the tree structures associated with BDD’s 4 through to 17.

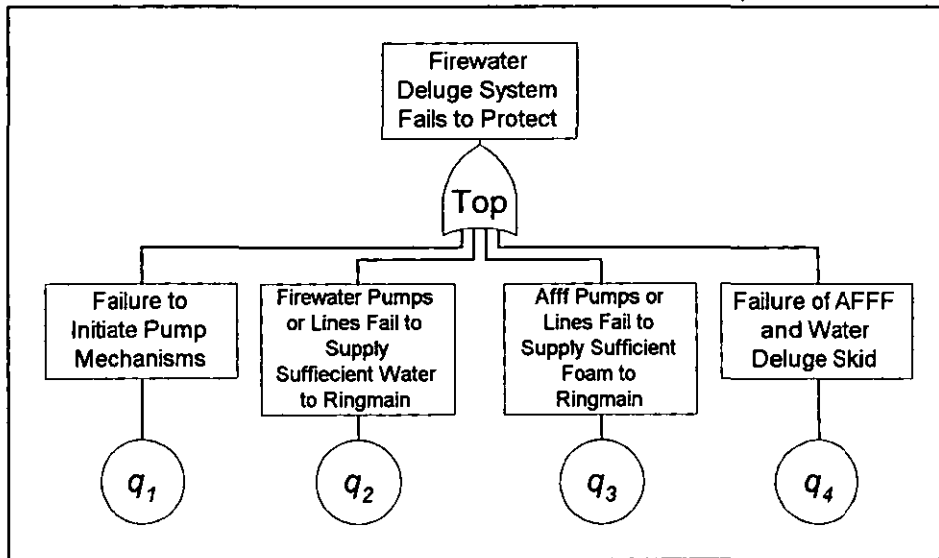


Figure 9.6 Top Event Quantification

As each ‘basic event’ featured in figure 6 is independent and mutually exclusive

$$Q_{SYS} = 1 - \prod_{i=1}^4 (1 - q_i) \quad (9.1)$$

where Q_{SYS} is the probability that the FDS fails to work on demand.

9.3.1.3 Computational Method for BDD Quantification

A computational approach to calculate the system unavailability of the HIPS is described in section 5.2.1. The basic framework is applicable for quantification of the FDS. However, certain modifications and extensions are required.

The first step in the computational procedure involves input of each BDD file to represent the total system structure (see table 9.12). This is implemented in subroutine *Getumbdds*. A database storing the *ite* table of each BDD is created, where

unnodeinfo1 refers to the database storing the structure of BDD number 1, *unnodeinfo2* to the structure of BDD number 2, and so on.

The next step compiles further databases to describe the events comprising each BDD. The records are stored in order of the index associated with the respective BDD. *Uneventinfo1* refers to the database of events associated with BDD number 1, *uneventinfo2* is associated with BDD number 2, and so on. The structure of each record is as shown in figure 5.15 with 3 additional fields: 'Modified Maintenance Test Interval', 'Wear-out or Non Wear-out (basic event)' and 'Preventative Maintenance Test Interval'. In addition, the field 'Subsystem 1 or 2' is omitted.

Fixundata enters all fixed data associated with each event in each database, thereby partially filling each *uneventinfo*. The event name and event type (i.e. basic, human error or house) are input in each record. Each basic event is specified as being of wear-out or non wear-out type and its associated failure and repair data entered. In addition, as regards events resulting from human error the probability of occurrence is specified directly.

Having entered all fixed information the unavailability of each separate part of the system, i.e. q_1 , q_2 , q_3 , and q_4 (see figure 9.6), is evaluated for the specific design. This is achieved in subroutines *Findq1*, *Findq2*, *Findq3* and *Findq4* respectively. System unavailability is subsequently evaluated using equation (9.1). Within each of the *Find* routines *Unavailworkout* (described in section 5.2.1.4.3) is used to analyse each BDD. Modifications are however necessary in that it must first be decided which BDD is undergoing quantification. In addition, the means to achieve the probability of each node within the respective BDD is a little more involved. The following sections describe the modifications in more detail and an outline of the whole process is portrayed in figure 9.7.

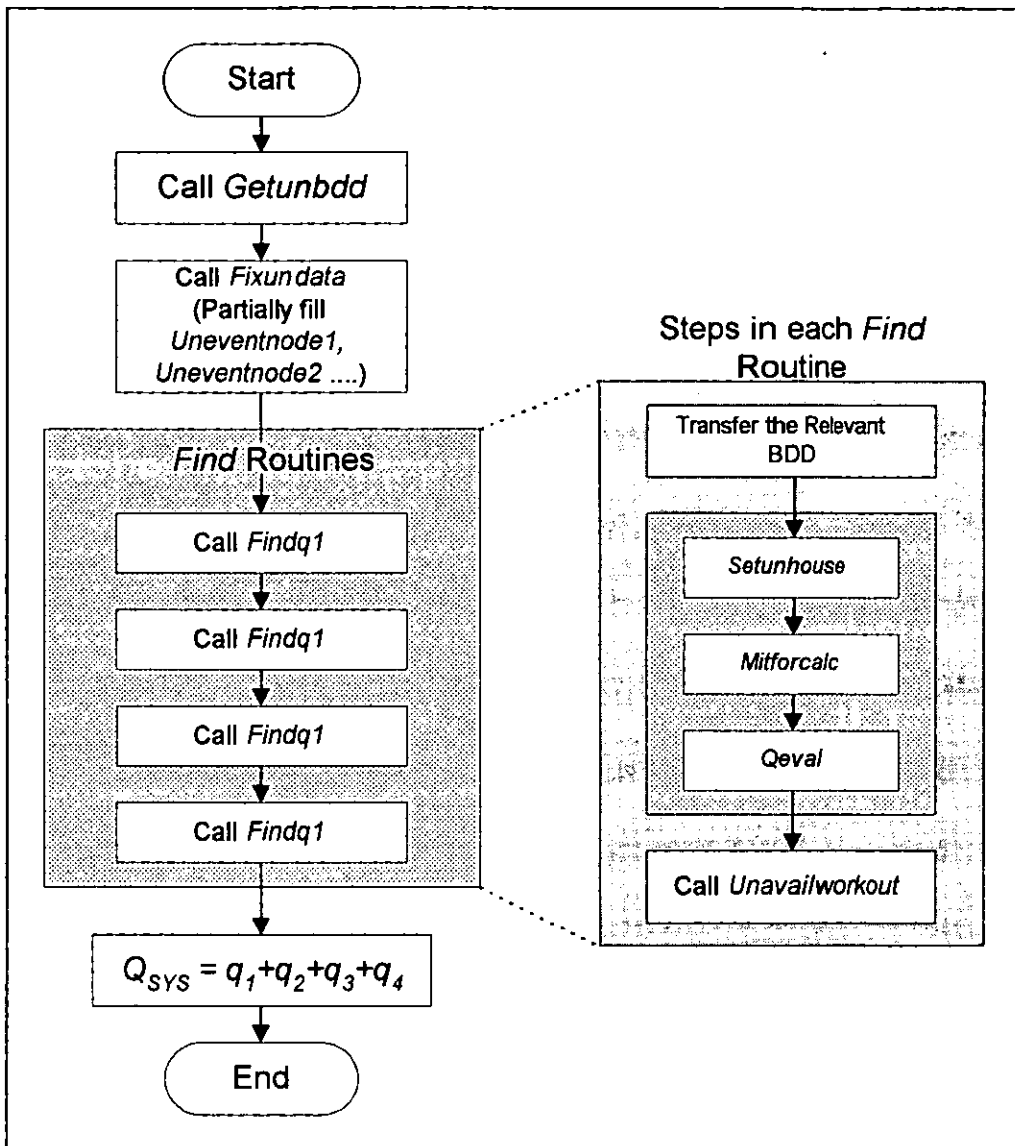


Figure 9.7 The Computational Method to Evaluate FDS Unavailability

9.3.1.3.1 Transferring the BDD

On execution of each of the *Find* routines the first step involves transferring the relevant BDD to the structures used within *Unavailworkout*, i.e. the general structures *unnnodeinfo* and *uneventinfo*. As regards *Findq1*, *Findq3* and *Findq4* this is a straightforward process. A single BDD for each part of the system associated with these routines, models each possible design variation. More specifically, BDD number 1 models all the possible design alternatives achieved through variation of the

values assigned to N , K and P . As such, *Findq1* transfers the description of BDD number 1 stored in *unnodeinfo1* and its associated event database, *uneventinfo1*, to the general structures *unnodeinfo* and *uneventinfo* respectively, in preparation of BDD quantification.

Findq2 is a little more complex in that 14 BDD's are required to model each possible design alternative arising from manipulation of the firewater pump variables. To model a specific design, however, only one of the 14 BDD's is necessary. Consequently, an if then else structure is used to determine which BDD description file, and hence associated event database, is required. For example, if the design under consideration has firewater pumps of 100% capacity the structure of BDD number 4 is retrieved. Conversely, if the particular design has 3 firewater pumps of 50% capacity it is required that BDD number 6 is retrieved.

9.3.1.3.2 Derivation of Component Unavailabilites

The next step in each *Find* routine is to define the unavailability of each node in the relevant BDD specific to the considered design.

House events are dealt with in an identical manner to that previously described in *Setunhouse* in 5.2.1.4.2.

Each component in the system undergoes scheduled maintenance. As such, the average unavailability of each basic event (omitting those due to human error) is dependent on the interval between system testing. Components comprising the firewater and AFFF pump lines are tested every θ_p days, components comprising the distribution network every θ_r weeks and components on the deluge skids every θ_D months. Maintenance tests carried out on separate parts of the system are not, however, independent. The pumps and lines are inadvertently tested whenever ringmain tests are carried out. Normal operation of the water and AFFF ringmains necessarily requires an adequate supply of water or foam from the pump lines. Similarly, both the pump lines and ringmains are indirectly tested whenever there is a deluge skid test. Normal operation of the deluge skid necessarily requires that

sufficient water is pumped through the system. As such, θ_P and θ_R are generally pessimistic. θ_P must be modified to incorporate the influence of θ_R and θ_D prior to use within *Findq2* and *Findq3*. θ_R must, in turn, be modified due to the influence of θ_D prior to use within *Findq1*. θ_D is not, however, affected and can be used directly within *Findq4*.

Modification of θ_P and θ_R

Modified values for θ_P and θ_R , denoted as θ_P^M and θ_R^M respectively, are derived in a routine called *Mtiforcalc*. For ease of computation a year is considered to be 48 weeks, i.e. 336 days.

The step-by-step algorithm to modify θ_R is:

- 1) Convert θ_R and θ_D to days.
- 2) Establish the number of deluge tests carried out in a year, denote as $N\theta_D$.
- 3) Set up an array to store each day in the year on which the deluge skid is tested, denote as $A\theta_D$.
- 4) Divide each member of $A\theta_D$ by θ_R . Eliminate each member that divides exactly. Let the number eliminated be stored in *divrd*. *Divrd* specifies the number of days when a ringmain and deluge test coincide.
- 5) Establish the number of ringmain tests carried out in a year, denote as $N\theta_R$.
- 6) Sum $N\theta_D$ and $N\theta_R$ and subtract *divrd*.
- 7) To derive θ_R^M divide 336 by the number resulting from step 6. Divide this result by 7 to convert θ_R^M back to weeks.

Further steps to derive θ_P^M are:

- 1) Set up an array to store each day in the year on which the ringmain is tested, denote as $A\theta_R$.
- 2) Append the values in $A\theta_D$ (the reduced version) to $A\theta_R$.
- 3) Establish the number of pump tests carried out in a year, denote as $N\theta_P$.

- 4) Divide each member of $A\theta_R$ by θ_P and determine the number of these that divide exactly, denote as $divpr$.
- 5) Sum $N\theta_D$, $N\theta_R$ and $N\theta_P$ then subtract $divrd + divpr$.

This algorithm is illustrated by means of an example in figure 9.8.

Let $\theta_D = 3$ months, $\theta_R = 8$ weeks and $\theta_P = 16$ days

Evaluate θ_R^M :

- 1) $\theta_D = 84$ days, $\theta_R = 56$ days
- 2) $N\theta_D = 336 \div 84 = 4$ occurrences
- 3) $A\theta_D = \{84, 168, 252, 336\}$
- 4) Both 168 and 336 are directly divisible by 56
so these values are eliminated from $A\theta_D$. $Divrd = 2$.
- 5) $N\theta_R = 336 \div 56 = 6$ occurrences
- 6) Do $N\theta_D + N\theta_R - divrd = 4 + 6 - 2 = 8$
- 7) $\theta_R^M = 336 \div 8 = 42$ days, i.e. 6 weeks

Evaluate θ_P^M :

- 1) $A\theta_R = \{56, 112, 168, 224, 280, 336\}$
- 2) $A\theta_R$ appended = $\{56, 84, 112, 168, 224, 252, 280, 336\}$
- 3) $N\theta_P = 336 \div 16 = 21$ occurrences
- 4) Days 112, 224 and 336 are divisible by 16.
- 5) Hence, $divpr = 3$
- 6) Do $N\theta_D + N\theta_R + N\theta_P - (divrd + divpr) = 31 - 5 = 26$
- 7) $\theta_P^M = 336 \div 26 = 12.92$ days

Figure 9.8 An Example to Demonstrate Derivation of θ_R^M and θ_P^M

It should be noted that θ_R^M and θ_P^M are used only for evaluation purposes. The actual maintenance test intervals assigned to the design remain unchanged.

Wear-out Components

For a component with a non-constant failure rate the wear-out phase of the bath-tub curve applies and as such, the time to failure distribution is the Weibull distribution (2.37).

Consider a wear-out component that is tested at intervals of θ and undergoes preventative maintenance at θ_{PM} , as shown in figure 9.9. Only after preventative maintenance is the component considered as good as new.

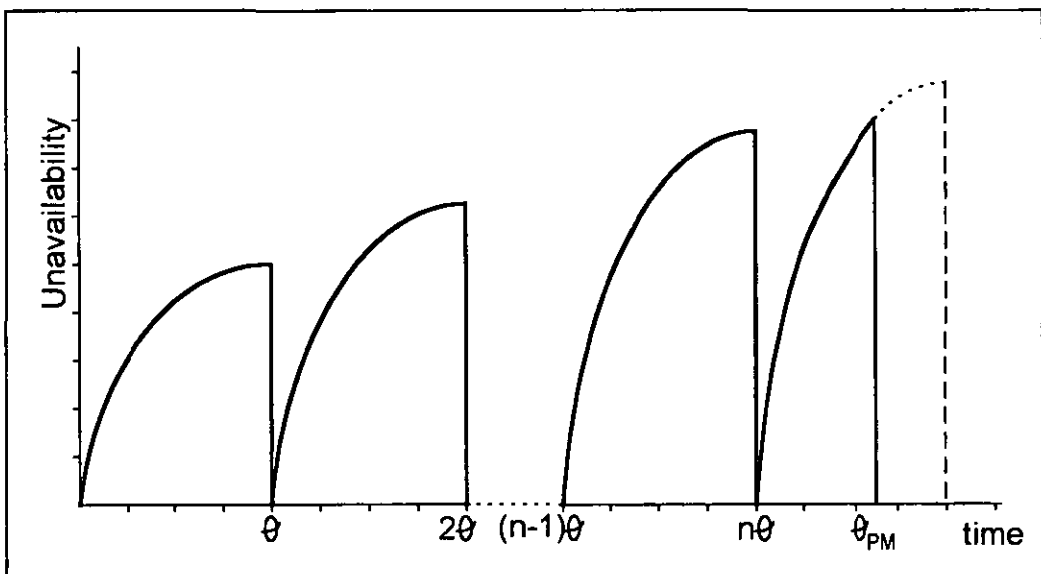


Figure 9.9 Failure with Periodic Inspection and Preventative Maintenance

Average component unavailability is, thus

$$q_{AV} = \frac{q_{AV1} + q_{AV2} + \dots + q_{AV(n-1)} + q_{AVn}}{\theta_{PM}} \quad (9.3)$$

where

$$q_{AV1} = \int_0^{\theta} \left(1 - e^{-\left(\frac{t}{\eta}\right)^{\rho}} \right) dt$$

$$q_{AV2} = \int_0^{\theta} \left(1 - e^{\left(\frac{1}{\eta}\right)^{\beta} [(t+\theta)^{\beta} - \theta^{\beta}]} \right) dt$$

$$q_{AV(n-1)} = \int_0^{\theta} \left(1 - e^{\left(\frac{1}{\eta}\right)^{\beta} [(t+(n-1)\theta)^{\beta} - ((n-1)\theta)^{\beta}] } \right) dt$$

$$q_{AVn} = \int_0^{(\theta_{PM} - n\theta)} \left(1 - e^{\left(\frac{1}{\eta}\right)^{\beta} [(t+n\theta)^{\beta} - (n\theta)^{\beta}] } \right) dt$$

The failure rate is time dependent and as such, $t + i\theta$, where $i = 0$ to n , is incorporated within equation (9.3) for sections q_{AV1} , q_{AV2} and q_{AVn} respectively.

A numerical approximation method, specifically Simpson's Rule, is used to evaluate each term in equation (9.3). To reduce the effects of ill-conditioning the exponential terms are first expanded in terms of their power series. This gives

$$q_{AV1} = \int_0^{\theta} \left((at)^{\beta} - \frac{1}{2!}(at)^{2\beta} + \frac{1}{3!}(at)^{3\beta} - \dots \right) dt$$

$$q_{AV2} = \int_0^{\theta} \left(\alpha^{\beta} \left((t+\theta)^{\beta} - \theta^{\beta} \right) - \frac{\left(\alpha^{\beta} \left((t+\theta)^{\beta} - \theta^{\beta} \right) \right)^2}{2!} + \dots \right) dt$$

$$q_{AVn} = \int_0^{(\theta_{PM} - n\theta)} \left(\alpha^{\beta} \left((t+n\theta)^{\beta} - (n\theta)^{\beta} \right) - \frac{\left(\alpha^{\beta} \left((t+n\theta)^{\beta} - (n\theta)^{\beta} \right) \right)^2}{2!} + \dots \right) dt$$

where

$$\alpha = \frac{1}{\eta} \tag{9.4}$$

The equations can be simplified if the Weibull parameters, β and η , are known.

The only components of wear-out type in the FDS are the firewater and AFFF pumps. As such, the value of θ in equation (9.3) is θ_P^M . The Weibull parameters, β and η , for each pump are specified in tables 9.8 and 9.11.

The step-by-step algorithm to evaluate q_{AV} for a component of wear-out type is:

- 1) Convert both θ_P^M and θ_{PM} to hours
- 2) Establish the number of tests carried out prior to θ_{PM} , denote as n .
- 3) Establish the size of the end section, i.e. $\theta_{ES} = \theta_{PM} - n\theta$.
- 4) Set $\beta = 2$ if F_P is 100%, else $\beta = 3/2$.
- 5) Evaluate α using equation (9.4).
- 6) Use Simpson's Rule with 6 strips to evaluate the area of each section, where $i = 0$ to n (corresponding to each section):
 - i) Evaluate:

$$q_{AV(i+1)} = \int_0^{\theta'} \left(\alpha \left((t + i\theta_P^M)^\beta - (i\theta_P^M)^\beta \right) - \frac{\alpha^2 \left((t + i\theta_P^M)^\beta + (i\theta_P^M)^\beta \right)^2}{2!} + \dots \right) dt \quad (9.5)$$

where $\theta' = \theta_P^M$ for $i = 0$ to $n-1$ and $\theta' = \theta_{ES}$ for $i = n$

- ii) It must be ensured that evaluation of each section is sufficiently accurate. As such, Simpson's Rule is first implemented using 1 term in equation (9.5), then a second time using two terms. If the difference between the two results is less than 1×10^{-8} the latter value is accepted. Else the integral is evaluated again using an additional term. This process continues until the difference from one Simpson's evaluation to the next is less than the specified error criteria.
- 7) Evaluate q_{AV} using:

$$q_{AV} = \frac{\sum_{i=0}^n q_{AVi}}{\theta_{PM}} \quad (9.6)$$

A demonstration of the step-by step algorithm to evaluate the unavailability of a wear-out component is illustrated in figure 9.10. (The value of θ_{PM} used in the example is neither practical nor feasible for the FDS. A small value is chosen in order to simplify the demonstration of the algorithm.)

Having implemented *Setunhouse* and *Qeval* probabilities have been established for each node in the relevant BDD. As such, execution of *Unavailworkout* proceeds to calculate the probability of occurrence of the respective system part.

F_P is 100%

$$\lambda = 2 \times 10^{-6}$$

$$\theta_P^M = 4 \text{ days}$$

$$\theta_{PM} = 10 \text{ days}$$

1) $\theta_P^M = 96 \text{ hours}$, $\theta_{PM} = 240 \text{ hours}$

2) 2 tests prior to θ_{PM} , $\therefore n = 2$

3) $\theta_{ES} = 240 - (2 \times 96) = 48 \text{ hours}$

4) $\beta = 2$

5) $a = \left(\frac{1}{400000} \right)^2 = 6.25 \times 10^{-12}$

6) Consider each section:

• Section 1, $i = 0$

i) Evaluate:

$$q_{AV1} = \int_0^{96} \left(at^2 - \frac{a^2 t^4}{2!} + \frac{a^3 t^6}{3!} - \dots \right) dt$$

ii)

- Using 1 term $q_{AV1}^1 = 2.765 \times 10^{-6}$
- Using 2 terms $q_{AV1}^2 = 2.764 \times 10^{-6}$
- Test for accuracy,
 $q_{AV1}^1 - q_{AV1}^2 = 1 \times 10^{-9} < 1 \times 10^{-8}$
- Therefore, $q_{AV1} = 2.764 \times 10^{-6}$

• Section 2, $i = 1$

i) Evaluate:

$$q_{AV2} = \int_0^{96} \left(a(t^2 + \theta_P^M t) - \frac{a^2 (t^2 + \theta_P^M t)^2}{2!} + \dots \right) dt$$

ii)

- Using 1 term $q_{AV2}^1 = 1.10592 \times 10^{-5}$
- Using 2 terms $q_{AV2}^2 = 1.105916 \times 10^{-5}$
- Test for accuracy,
 $q_{AV2}^1 - q_{AV2}^2 = 4 \times 10^{-11} < 1 \times 10^{-8}$
- Therefore, $q_{AV2} = 1.105916 \times 10^{-5}$

• Section 3, $i = 2$

i) Evaluate:

$$q_{AV3} = \int_0^{48} \left(a(t^2 + 2\theta_P^M t) - \frac{a^2 (t^2 + 2\theta_P^M t)^2}{2!} + \dots \right) dt$$

ii)

- Using 1 term $q_{AV3}^1 = 2.73792 \times 10^{-5}$
- Using 2 terms $q_{AV3}^2 = 2.73791 \times 10^{-5}$
- Test for accuracy,
 $q_{AV3}^1 - q_{AV3}^2 = 1 \times 10^{-10} < 1 \times 10^{-8}$
- Therefore, $q_{AV3} = 2.73791 \times 10^{-5}$

7) $q_{AV} = \frac{2.764 \times 10^{-6} + 1.105916 \times 10^{-5} + 2.73791 \times 10^{-5}}{240} = 1.717 \times 10^{-7}$

Figure 9.10 Evaluation of q_{AV} For a Component of Wear-out Type

9.4 Evaluate the Frequency of Spurious Trip Occurrences of the FDS

9.4.1 Constructing the Spurious Trip Fault Tree

As a result of the constraint limiting the number of spurious system occurrences permitted, spurious activation of the FDS must be established. The specific fault tree to quantify causes of this failure mode must first be developed.

The top event occurs if either of the solenoid valves fail spuriously, the valmatic release valve opens spuriously or the signal from the MFGP to the solenoid valves is interrupted. The latter event occurs as a result of spurious activation of the ringmain pressure sensors. The causal relationship of events directly causing the top event is depicted in figure 9.11.

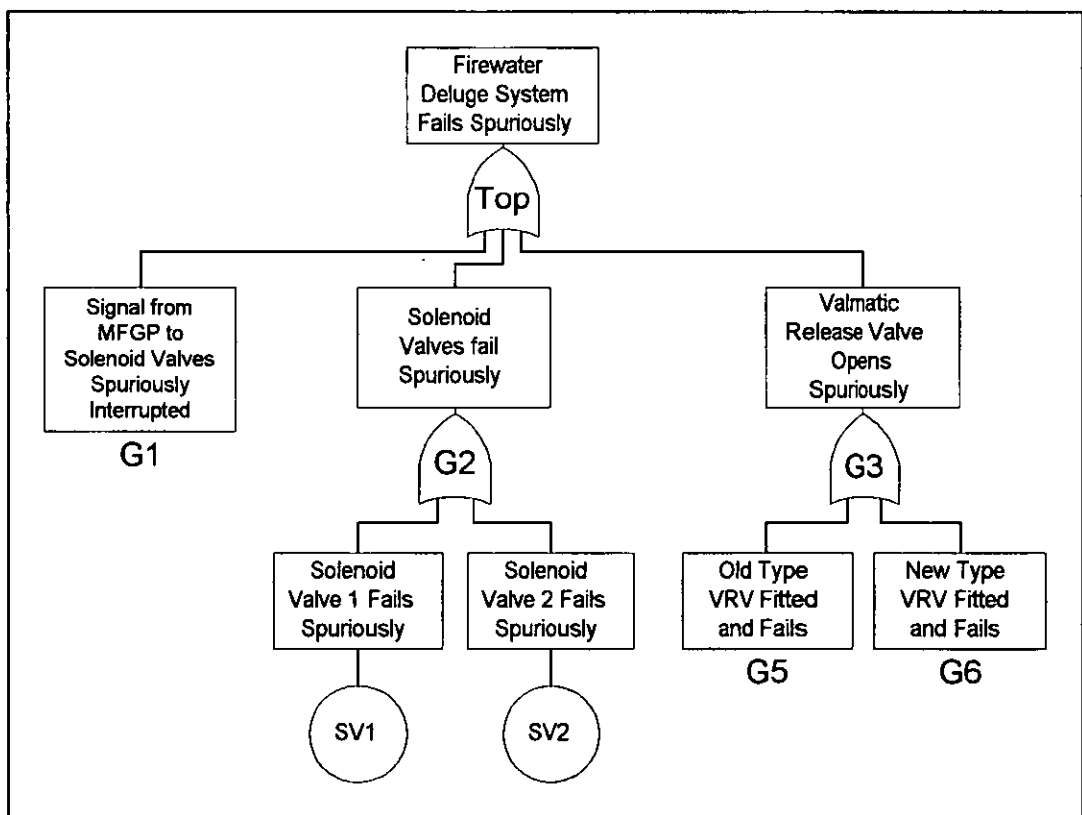


Figure 9.11 Spurious Trip Fault Tree

Gate 'G1', i.e. 'Signal from MFGP to the Solenoid Valves is Spuriously Interrupted', is modelled in a similar manner to that described in section 5.2.3.1. The causal relationship describing each aspect of the FDS spurious trip fault tree is depicted in full in Appendix VI.

9.4.2 Conversion of the Spurious Trip Fault Tree to a BDD

In total the spurious trip fault tree of the FDS consists of 65 primary events and 28 gates. Conversion of the associated 'ats' fault tree structure file to its equivalent BDD proceeds in an identical manner to that of the spurious fault tree used to quantify the HIPS system. A non-minimal *ite* table consisting of ? statements results.

9.4.3 Evaluating the Top Event Unconditional Failure Intensity of the BDD

All components featured in the spurious trip fault tree for the FDS have constant failure rates. In addition, spurious failures are instantaneously revealed and hence, the probability of failure of each basic event is independent of its associated maintenance test interval. As a result the computational approach to evaluate the top event unconditional failure intensity of the BDD for the FDS is identical to that of the HIPS (described in section 5.2.3.4).

9.5 Life Cycle Costs

Due to constraints imposed on the FDS, consideration must be given to the available resources regarding cost.

To build the FDS an initial cost is incurred. Once built further running costs must also be taken into account. Running costs include only maintenance activity, i.e. the cost of system testing at regular intervals, the cost of corrective maintenance to repair any problems highlighted by system testing and the cost of preventative maintenance carried out at regular intervals on components that exhibit wear-out. The following

section describes the means to evaluate each of these aspects. In this study running costs are evaluated over a period of 1 year, i.e. 8760 hours. (Data regarding the costs associated with each component are specified in tables 9.3, 9.7, 9.8, 9.9 and 9.11).

5.1 Initial Cost

Each component has an initial purchase cost. A storage cost is also associated with each component, which depends on the number of spare items stored and the cost to store each item.

Initial cost (including storage cost) of the FDS (SIC) is given by

$$\text{SIC} = \text{ICFPL} + \text{ICAPL} + \text{ICR} + \text{ICDS} \quad (9.7)$$

Where ICFPL, ICAPL, ICR and ICDS denote initial cost of the firewater pumps and lines, initial cost of the AFFF pumps and lines, initial cost of the ringmain and initial cost of the deluge skid respectively.

Considering the ICFPL further

$$\text{ICFPL} = [F_E \times \text{ICelec}] + [(F - F_E) \text{ICdies}] + \text{ICelecsup} + \text{ICdiessup}$$

where $(F - F_E)$ denotes the number of diesel driven pump lines, ICelec and ICdies represent the initial cost of each electric pump and diesel pump line respectively, ICelecsup and ICdiessup represent the initial cost of the electric supply and the initial cost of the diesel supply respectively.

Each pump line consists of certain fixed components, i.e. a filter (F), check valve (CV), isolation valve (IV), spill-back valve (SPV) and a pressure relief valve (PRV). The initial plus storage cost of the CV (CV_{IS}) is

$$\begin{aligned} \text{CV}_{\text{IS}} &= C_1 + (C_s \times N_s) \\ &= 500 + (2 \times 300) = 600 \end{aligned}$$

(Notation is described in table 9.1). The initial plus storage cost for each component is evaluated in a similar manner. Thus, the initial cost of each electric pump line (ICelec) is

$$\text{ICelec} = 5400 + \text{FE}_{\text{IS}}$$

where 5400 is the sum of the initial plus storage cost of each fixed component on an electric pump line and FE_{IS} is the initial plus storage cost of the particular electric pump chosen (i.e. 100%, 50% or 33 1/3%, type 1 or 2). ICdies is developed in a similar manner.

ICelecsup comprises the electricity supply (ES) alone. In contrast, ICdiessup comprises both the diesel tank isolation valve (DIV) and the low level alarm (LA). The initial cost of the firewater pumps and lines can, therefore, be expressed as

$$\text{ICFPL} = [F_E (5400 + \text{FE}_{\text{IS}})] + [(F - F_E)(5400 + \text{FD}_{\text{IS}})] + 2600 \quad (9.8)$$

where 2600 is the sum of ICelecsup and ICdiessup.

ICAPL is given by

$$\begin{aligned} \text{ICAPL} = & [A_E \times \text{ICelec}] + [(A - A_E) \times \text{ICdies}] \\ & + \text{ICelecsup} + \text{ICdiessup} + \text{ICaffsup} \end{aligned}$$

where $(A - A_E)$ denotes the number of diesel driven pumps in the AFFF system and ICaffsup represents the initial cost of the AFFF tank isolation valve. Applying the same principles as for the firewater pumps and lines, ICAPL can be expressed as

$$\text{ICAPL} = [A_E \times (6400 + \text{AE}_{\text{IS}})] + [(A - A_E) \times (6400 + \text{AD}_{\text{IS}})] + 3600 \quad (9.9)$$

where 6400 is the sum of the initial plus storage cost of each fixed component on an electric or diesel pump line (this differs from the firewater pump lines due to an additional isolation valve), AE_{IS} and AD_{IS} are the initial plus storage cost of the

particular electric or diesel pump chosen and 3600 is the sum of ICelecsup, ICdiessup and ICaffsup.

The fire pump selector unit (FSU) is the only fixed component in the ringmain. The ICR is thus

$$ICR = 2200 + P_{IS} \quad (9.10)$$

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

Fixed components on the deluge skid are the manual release mechanism (MRM), solenoid valve 1 (SV1), solenoid valve 2 (SV2), strainer (S), water deluge isolation valve (WIV), AFFF check valve (ACV) and AFFF deluge isolation valve (AIV). Variable components in the deluge system are a valmatic release valve (VRV), a nozzle (N) and an inductor nozzle (IN). These are made of corrosion resistant or non-corrosion resistant material. In addition, the skid comprises a water deluge valve (WV) and an AFFF deluge valve (AV). Both valves are of type 1,2 or 3. Thus, the ICDS is given by

$$ICDS = 7350 + VRV_{IS} + N_{IS} + IN_{IS} + WV_{IS} + AV_{IS} \quad (9.11)$$

where 7350 is the sum of the initial plus storage cost of the fixed components. The latter terms correspond to the initial plus storage cost of the particular type of variable components chosen.

Initial cost of the system (SIC) can, therefore, be expressed as

$$\begin{aligned} SIC = & [F_E (5400 + FE_{IS}) + (F_P - F_E) (5400 + FD_{IS}) + 2600] \\ & + [A_E (6400 + AE_{IS}) + (A_P - A_E) (6400 + AD_{IS}) + 3600] \\ & + [2200 + P_{IS}] + [7350 + VRV_{IS} + N_{IS} + IN_{IS} + WV_{IS} + AV_{IS}] \end{aligned} \quad (9.12)$$

9.5.2 Corrective Maintenance Cost

Cost of corrective maintenance for each component depends on the expected number of failures and the cost to repair each failure. Specifically, corrective maintenance of component i (CM_i) is given by

$$CM_i = (W_i^D + W_i^S) \times (C_R \times C_{HR} \times C_{SR}) \quad (9.13)$$

where W_i^D and W_i^S denote the expected number of dormant and spurious failures for component i respectively over the allocated time period (i.e. 8760 hours). Further notation is described in table 9.1. As an example, consider the check valve. Using equation (9.13) the corrective maintenance cost of the check valve (CV_{CM}) is

$$\begin{aligned} CV_{CM} &= W_{CV}^D [(18 \times 30) + 300] \\ &= W_{CV}^D \times 840 \\ &= W_{CV}^D \times CV_{CM,rep} \end{aligned}$$

where $CV_{CM,rep}$ is the cost incurred each time the check valve is repaired (i.e. 840 units). Note that the CV does not fail spuriously.

For component i of non wear-out type the unconditional failure intensity (w_i) is given by

$$w_i = \lambda_i (1 - q_i) \quad (9.14)$$

The expected number of failures is then determined using equation (2.21), where $t = 8760$ hours.

For component i of wear-out type the failure rate is time dependent and as such, is defined by the Weibull distribution, equation (2.37). The expected number of failures per year is, therefore,

$$W_i(0,8760) = \left(\frac{8760}{\theta_{PM}} \right) \int_0^{\theta_{PM}} \left(\frac{\beta_i}{\eta_i} \right) \left(\frac{t}{\eta_i} \right)^{\beta_i-1} (1 - q_i) \quad (9.15)$$

where θ_{PM} is converted to hours.

Cost incurred by the FDS due to corrective maintenance (SCMC) is given by

$$SCMC = \mathbf{CMCFPL} + \mathbf{CMCAPL} + \mathbf{CMCR} + \mathbf{CMCDS} \quad (9.16)$$

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

In each of equations (9.17), (9.18), (9.19) and (9.20) to follow, which describe the terms in equation (9.16), variable components are represented in bold, in order to distinguish them from the fixed components. Consider, for example, equation (9.17). \mathbf{W}_{FE}^D is in bold as it denotes the expected number of dormant failures of the particular electric pump chosen (i.e. 100%, 50%, 33 1/3% of type 1 or 2). \mathbf{AE}_{CMrep} represents the cost per repair associated with that electric pump. In contrast, W_{CV}^D denotes the expected number of dormant failures for the check valve, where there is only one such component type that achieves this purpose. As a result the cost per repair of the CV (840 units) can be specified directly.

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

$$\mathbf{CMCFPL} = [F_E \times \mathbf{CMCelec}] + [(F - F_E) \times \mathbf{CMCdies}] \\ + \mathbf{CMCelesup} + \mathbf{CMCdiessup}$$

where **CMCelec** and **CMCdies** represent the corrective maintenance costs of each electric and diesel pump line respectively, **CMCelesup** and **CMCdiessup** represent the corrective maintenance cost of the electric and diesel supply respectively. Using equation (9.13) to define the corrective maintenance cost associated with each component, **CMCFPL** can be expressed as

$$\begin{aligned}
\text{CMCFPL} = & \left[F_E \times (\text{Fix} + (\mathbf{W}_{FE}^D \times \mathbf{FE}_{\text{CM rep}})) \right] \\
& + \left[(F - F_E) \times (\text{Fix} + (\mathbf{W}_{FD}^D \times \mathbf{FD}_{\text{CM rep}})) \right] \\
& + [W_{ES}^D \times 290] + [(W_{DIV}^D \times 400) + (W_{LA}^D \times 280)]
\end{aligned} \tag{9.17}$$

where

$$\text{Fix} = (W_{CV}^D \times 840) + (W_{PRV}^D \times 790) + (W_{SV}^D \times 1120) + (W_{FB}^D \times 410) + (W_{IV}^D \times 740)$$

CMCAPL is established using the same principles as those above to give

$$\begin{aligned}
\text{CMCAPL} = & \left[A_E \times (\text{Fix} + (\mathbf{W}_{AE}^D \times \mathbf{AE}_{\text{CM rep}})) \right] \\
& + \left[(A - A_E) \times (\text{Fix} + (\mathbf{W}_{AD}^D \times \mathbf{AD}_{\text{CM rep}})) \right] \\
& + [W_{ES}^D \times 290] + [(W_{DIV}^D \times 400) + (W_{LA}^D \times 280)] + [W_{AIV}^D \times 740]
\end{aligned} \tag{9.18}$$

where

$$\text{Fix} = (W_{CV}^D \times 840) + (W_{PRV}^D \times 790) + (W_{SV}^D \times 1120) + (W_{FB}^D \times 410) + 2(W_{IV}^D \times 740)$$

CMCR is given by

$$\text{CMCR} = (W_{FSU}^D \times 1280) + ((\mathbf{W}_{PT}^D + \mathbf{W}_{PT}^S) \times \mathbf{PT}_{\text{CM rep}}) \tag{9.19}$$

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

In a similar vein, CMCDS involves summation of the corrective maintenance cost of each associated component. This gives

$$\begin{aligned}
\text{CMCDS} = & \left((W_{SV1}^D + W_{SV1}^S) \times 560 \right) + \left((W_{SV2}^D + W_{SV2}^S) \times 485 \right) \\
& + (W_{MRM}^D \times 660) + (W_S^D \times 410) + 2(W_{IV}^D \times 740) + (W_{CV}^D \times 840) \\
& + \left((W_{VRV}^D + W_{VRV}^S) \times \text{VRV}_{\text{CM rep}} \right) + (W_N^D \times N_{\text{CM rep}}) + (W_{IN}^D \times \text{IN}_{\text{CM rep}}) \\
& + (W_{WV}^D \times \text{WV}_{\text{CM rep}}) + (W_{AV}^D \times \text{AV}_{\text{CM rep}})
\end{aligned} \tag{9.20}$$

Note that the solenoid valves and the valmatic release valve can fail spuriously.

9.5.3 Preventative Maintenance Costs

A component with a constant failure rate does not experience wear-out throughout its lifetime. At any time it is equally as likely to fail as when it was new and as such, preventative maintenance is futile. The preventative maintenance cost incurred by a non wear-out component is, therefore, zero.

Establishing the cost incurred by the FDS due to preventative maintenance (SPMC) involves summation of the preventative maintenance cost incurred by each fitted pumps (i.e. each component of wear-out type).

The preventative maintenance cost per year of each wear-out component depends on the number of times preventative maintenance is carried out in the year and the cost per time. Preventative maintenance cost incurred by component i (PMC_i) is, thus

$$PMC_i = \left(\frac{8760}{\theta_{PM}} \right) \left((H_P \times C_{HP}) + C_{SP} \right) \tag{9.21}$$

Notation is described in table 9.1 (θ_{PM} is converted to hours). Consider, for example, the preventative maintenance cost of the firewater pump of 100% capacity ($E100_{PMC}$). Using equation (9.21) gives

$$\begin{aligned}
E100_{PM} &= \left(\frac{8760}{\theta_{PM}} \right) [(72 \times 30) + 300] \\
&= \left(\frac{8760}{\theta_{PM}} \right) \times 2460 \\
&= \left(\frac{8760}{\theta_{PM}} \right) \times E100_{PM \text{ rep}}
\end{aligned}$$

where $E100_{PM}$ is the cost each time preventative maintenance is carried out on the pump (i.e. 2460 units).

As a result, SPMC is given by

$$\begin{aligned}
SPMC &= [F_E \times PMCFelec] + [(F_P - F_E) \times PMCFdies] \\
&\quad + [A_E \times PMCAelec] + [(A_P - A_E) \times PMCADies]
\end{aligned}$$

where $PMCFelec$ and $PMCFdies$ denote the preventative maintenance costs of each firewater electric and diesel pump line respectively, $PMCAelec$ and $PMCADies$ denote the preventative maintenance costs of each AFFF electric and diesel pump line respectively. Using equation (9.21), SPMC can be expressed as

$$SPMC = \left(\frac{8760}{\theta_{PM}} \right) \left[(F_E \times FE_{PM \text{ rep}}) + ((F_P - F_E) \times FD_{PM \text{ rep}}) \right. \\
\left. + (A_E \times AE_{PM \text{ rep}}) + ((A_P - A_S) \times AD_{PM \text{ rep}}) \right] \quad (9.22)$$

where $FE_{PM \text{ rep}}$ and $FD_{PM \text{ rep}}$ represent the cost per preventative maintenance of the particular firewater electric and diesel pump type chosen respectively, $AE_{PM \text{ rep}}$ and $AD_{PM \text{ rep}}$ represent the cost per preventative maintenance of the particular AFFF electric and diesel pump type chosen respectively.

9.5.4 Testing Cost

System tests are carried out on each pump line, the distribution network and deluge skid as dictated by θ_P , θ_R and θ_D respectively.

A pump line test examines the pump and all other components on that line simultaneously. Similarly, a single ringmain and deluge skid test examines all associated components. The cost of testing must only be considered once per group of components. (It is assumed that components tested simultaneously require the same specialized labour (i.e. C_{HT} , see table 1) and have the same test time (i.e. H_T , see table 9.1). If this condition does not hold modifications are required). As such, the cost incurred due to system testing per year (STC) is given by

$$\text{STC} = \text{TCFPL} + \text{TCAPL} + \text{TCR} + \text{TCDS} \quad (9.23)$$

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

TCFPL depends on the number of test carried out per year, the cost of testing each pump line plus the additional cost of testing the electric supply. The latter requires skilled electricians and as such, is considered separately. It is assumed that the diesel supply is tested indirectly via the diesel pumps. Each component on the pump lines has a test time of 2 hours ($H_T = 2$), with a cost of 30 units per hour ($C_{HT} = 30$). In contrast, the cost per hour to test the electricity supply is 40 units ($C_{HT} = 40$) Hence,

$$\begin{aligned} \text{TCFPL} &= F_p \times [\text{Cost to test each pump line per year}] + [\text{Cost to test ES per year}] \\ &= F_p \left[\frac{8760}{\theta_p} (H_T \times C_{HT}) \right] + \left[\frac{8760}{\theta_p} (H_T \times C_{HT}) \right] \\ &= F_p \left[\frac{8760}{\theta_p} (2 \times 30) \right] + \left[\frac{8760}{\theta_p} (2 \times 45) \right] \\ &= \frac{8760}{\theta_p} [(F_p \times 60) + 90] \end{aligned} \quad (9.24)$$

Using similar principles, TCAPL is given by

$$\text{TCAPL} = \frac{8760}{\theta_p} [(A_p \times 60) + 90] \quad (9.25)$$

The cost of testing the ringmain per year is

$$\begin{aligned} \text{TRC} &= \frac{8760}{\theta_R} (H_T \times C_{HT}) \\ &= \frac{8760}{\theta_R} \times 45 \end{aligned} \quad (9.26)$$

As regards the ringmain components H_T and C_{HT} are given in table 9.9.

The cost of testing the deluge skid per year is

$$\begin{aligned} \text{TCDS} &= \frac{8760}{\theta_D} (H_T \times C_{HT}) \\ &= \frac{8760}{\theta_D} \times 60 \end{aligned} \quad (9.27)$$

As regards the deluge skid components H_T and C_{HT} are given in table 9.3.

9.5.5 Evaluate the Life Cycle Cost

The Life Cycle Cost of the FDS (LCC) is the sum of equation (9.7), (9.16), (9.22) and (9.23), i.e.

$$\text{LCC} = \text{SIC} + \text{SCMC} + \text{SPMC} + \text{STC} \quad (9.28)$$

Cost due to Total Maintenance Effort (TMEC) is the sum of the time dependent terms in equation (9.28), i.e.

$$\text{TMEC} = \text{SCMC} + \text{SPMC} + \text{STC} \quad (9.29)$$

9.5.6 Implementing Life Cycle Cost Evaluation within the Computational Method

The life cycle cost formulae are implemented in a routine termed *Lifecosts*, which follows both unavailability and spurious BDD quantification routines.

To evaluate the corrective maintenance cost it is required that the expected number of dormant failures (W^D), and where necessary spurious failures (W^S), is derived for each component. Evaluation of W_i , both dormant and spurious, for component i is dependent on the component's probability of failure (q_i).

As regards system unavailability, the probability of failure of a component is dependent on the test intervals assigned to the particular design. As such, component unavailability is not a fixed value but varies from one design to the next and is evaluated specifically for the purpose of BDD quantification. Consequently, it is necessary to evaluate and store the expected number of failures for each component during the BDD quantification process.

To achieve this a subroutine termed *Wfort1* is integrated within routine *Findq1*. *Wfort1* establishes and stores the expected number of failures for each component associated with the ringmain network using component unavailability data derived in *Findq1*. Similarly, *Wfort2* is a subroutine integrated within *Findq2*. *Wfort2* establishes the expected number of failures for each component associated with the firewater pumps and lines. *Wfort3* is integrated within *Findq3* and *Wfort4* within *Findq4*. The former derives W^D for each component associated with the AFFF pumps and lines and the latter, all components associated with the deluge skid.

In a similar manner, a subroutine termed *Wforspur* is incorporated within the spurious trip BDD quantification process to derive the expected number of spurious failures (W^S) for each component that fails in this mode.

W^D , and where necessary W^S , is calculated only for the components comprising the deluge system design under consideration. For example, if the pressure transmitters in the considered FDS are of type 2, W^D and W^S are evaluated and stored for

pressure transmitter type 2, whereas pressures transmitters type 1 and 3 are disregarded.

In addition, subroutines *Wfort1*, *Wfort2*, *Wfort3*, *Wfort4* and *Wforspur* use if then else structures to deduce and store values for all the variables within each of the life cycle formulae specific to the FDS under consideration.

Consider, for example, equation (9.7), which evaluates initial system cost. Amongst others, it is required that the initial plus storage cost associated with the pressure transmitter holds the value corresponding to the pressure transmitter type fitted in the design. The pressure transmitters comprise the distribution network and as such, *Wfort1* contains the algorithm illustrated in figure 9.11.

```
Wfort1  
if P = 'type 1'  
    PIS = 700  
if P = 'type 2'  
    PIS = 400  
if P = 'type 3'  
    PIS = 300
```

Figure 9.11 An Algorithm within *Wfort1*

On execution of the routine *Lifecosts* all variables within each of the life cycle formulae, i.e. equations (9.7), (9.16), (9.22) and (9.23), are set specific to the particular design. Evaluation of initial system cost, corrective maintenance cost, testing cost and preventative maintenance cost is, therefore, a straightforward process.

Figure 9.12 illustrates integration of the subroutines required to evaluate life cycle cost of the FDS within the BDD quantification process (See figure 9.7 for reference).

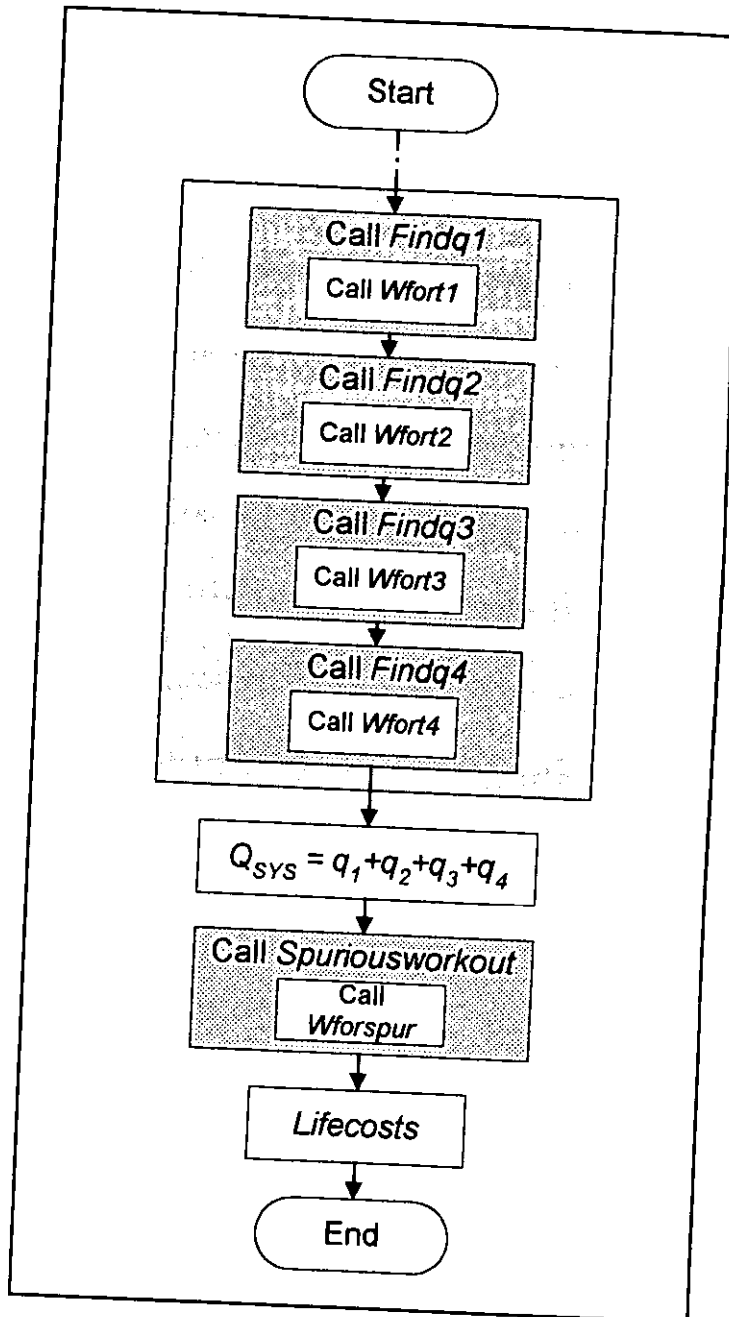


Figure 9.12 Integration of Life Cycle Cost Evaluation Routines

9.6 Computer Implementation of the GA to Optimise the FDS

The Genetic Algorithm Safety System Optimisation Procedure (GASSOP) is introduced in chapter 6 of this thesis. The framework of GASSOP in application to the FDS is predominantly the same as for the HIPS. The following is an outline of

the modifications that are required in order to allow for the increased complexity of the FDS. (Routines not discussed are implemented in an identical manner to that described in chapter 6). For discussion purposes GASSOP applied to the FDS is termed GASSOPm.

9.6.1 Memory

On execution of GASSOPm, *Memory* allocates space for a database of records that describe each individual in the GA population. This database is termed *opop*. Two replica structures are allocated simultaneously, *npopo* and *npopn*. Individuals chosen in the selection stage are transferred to *npopo*. Crossover and Mutation is subsequently carried out on a section of the string of each design stored in this structure. The maintenance test interval area of the string remains unaffected. Individuals within *npopo* are then copied to *npopn*, where genetic operator action manipulates the test interval parameters. The fitness information of the designs within each storage structure is updated and subsequently used to form a new population. The new population is transferred back to the original space, *opop*, prior to the next iteration. (genetic operator action is discussed in greater detail in section 9.6.6).

9.6.2 Coding and Initialisation

It was decided to create two solution strings to represent a particular FDS design. The second string accommodates all maintenance test interval parameters, i.e. θ_P , θ_R , θ_D , θ_{PM} , and the first string, all the other parameters remaining. The number of bits allocated to and the order in which each parameter is stored is depicted in figure 9.13. The first string is 29 bits in length, the second 16.

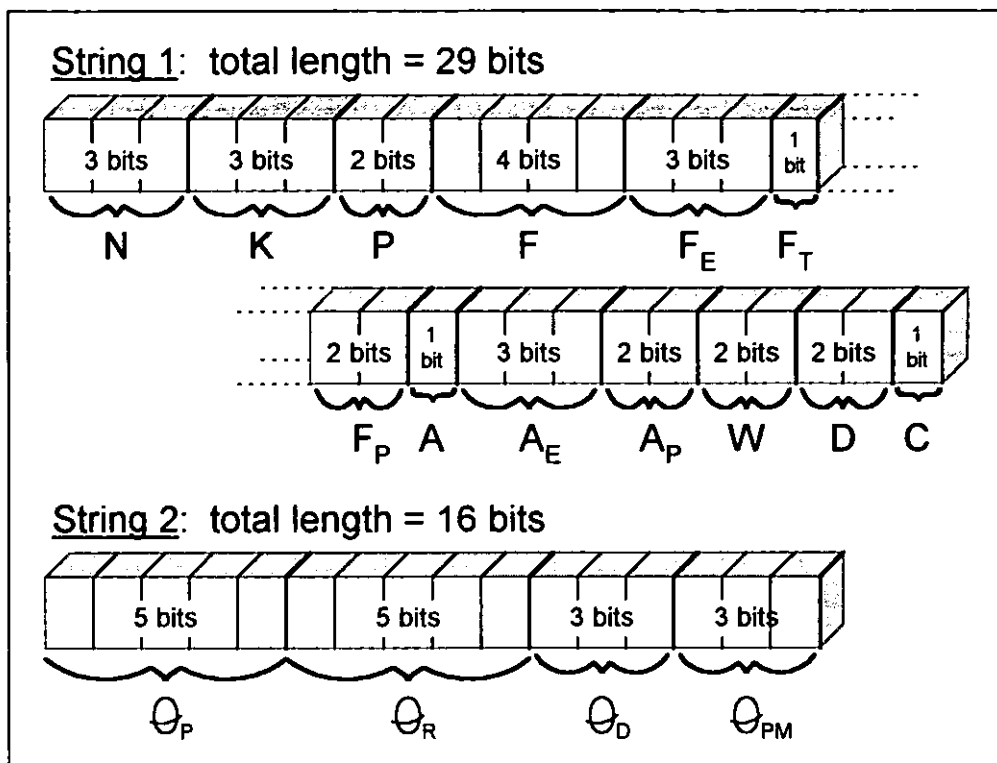


Figure 9.13 The FDS Parameter Set Coded as a Binary String

Representing each design by two solution strings eases implementation of the genetic operators (discussed in section 9.6.6).

Each design is initialised as described in section 6.2.2. An integer in the range 1 to 6 is generated for parameters θ_D and θ_{PM} . The resulting value is subsequently multiplied by 3, so that each parameter is assigned a value in accordance with its feasible range, i.e. 2 to 18 months in 3 monthly intervals.

9.6.3 Evaluating String Fitness

A simple explicit objective function to calculate the fitness of each deluge design does not exist. String fitness comprises of the system unavailability plus the respective penalty should any of the constraints be violated.

The routine Fitness obtains a fitness value for each design. The data fields that constitute the structure *fitparts*, associated with each individual, are as follows:

- Spurious trip frequency (F_{STS})
- Probability of system unavailability (Q_{STS})
- Penalised system unavailability (Q'_{STS})

Two further structures are stored within *fitparts*: *lifeparts* and *penparts*. The data fields that comprise each of these structures are:

- | <i>lifeparts</i> | <i>penparts</i> |
|--|---|
| • Life cycle cost per year of the FDS (LCC) | • Penalty due to excess LCC (LCC_P) |
| • Cost per year due to system testing (STC) | • Penalty due to excess STC (STC_P) |
| • Cost per year due to preventative maintenance (SPMC) | • Penalty due to excess SPMC ($SPMC_P$) |
| • Cost per year due to corrective maintenance (SCMC) | • Penalty due to excess TMEC ($TMEC_P$) |
| • Cost per year due to total maintenance effort (TMEC) | |

Fitness contains subroutines to evaluate and store the data in *fitparts* and follows the algorithm illustrated in figure 9.14.

```

Fitness
  if gen = 0
    Getunbdds and Getspbdds
    Fixundata and Fixspdata
  do for all strings j
    Unavail
      Findq1 ⇒ Wfort1
      Findq2 ⇒ Wfort2
      Findq3 ⇒ Wfort3
      Findq4 ⇒ Wfort4
       $Q_{SYS}^j = 1 - \prod_{i=1}^4 (1 - q_i)$ 
    Spurious
      Varspdata
      Spuriousworkout ⇒ Wforspur
       $F_{SYS}^j = FSYS$ 
    Lifecosts
    Penaltys
  Genaverage
  Genbest

```

Figure 9.14 Fitness Evaluation Algorithm for the FDS

Penalty formulae must be derived to compensate for any constraints that may be violated. These formulae are evaluated within the routine *Penaltys*.

9.6.4 Derivation of the Penalty Formulae

The FDS has four constraints regarding cost. For each constraint a penalty formula of the type described in sections 6.2.3.1 and 7.3 is applied. This gives

$$LCC_P = \left(\frac{\text{excess LCC}}{1250} \right)^{\frac{9}{8}} \times \left(\frac{Q_{SYS}}{100} \right) \quad (9.30)$$

$$STC_p = \left(\frac{\text{excess STC}}{205} \right)^{\frac{9}{8}} \times \left(\frac{Q_{SYS}}{100} \right) \quad (9.31)$$

$$SPMC_p = \left(\frac{\text{excess SPMC}}{135} \right)^{\frac{9}{8}} \times \left(\frac{Q_{SYS}}{100} \right) \quad (9.32)$$

$$TMEC_p = \left(\frac{\text{excess TMEC}}{440} \right)^{\frac{9}{8}} \times \left(\frac{Q_{SYS}}{100} \right) \quad (9.33)$$

The first term in each equation expresses the percentage by which the particular cost exceeds its constraint. The system unavailability of the considered design is then multiplied by the respective percentage excess to establish the appropriate penalty. Equations (9.30), (9.31), (9.32) and (9.33) are not independent. Excess life cycle cost invariably implies a violation in one or more of the other constraints. Similarly, both system testing and preventative maintenance comprise a proportion of total maintenance effort. As such, an excess in maintenance effort generally coincides with system testing and preventative maintenance cost violations. To compensate for this interaction to a certain extent the exponential relationship, i.e. $y = x^{\frac{9}{8}}$, is chosen to be smaller than that used previously without achieving linearity. An example of application of the cost penalty formulae is given in figure 9.15.

Occurrence of a spurious trip ceases production on the processing platform and causes financial loss. As a result the spurious trip constraint violation is expressed in terms of excess cost (described in section 6.2.3.1). The life cycle cost constraint formula, equation (9.30), can then be used to derive the penalty (S_p) applied to Q_{SYS} .

- Consider string A: $Q_{sys} = 0.0125$
 $LCC = 12700$ $STC = 20450$ $SPMC = 15000$
 $SCMC = 11000$ $TMEC = 46450$

- $Excess\ LCC = 2000$
 $Excess\ STC = N/A$
 $Excess\ SPMC = 1500$
 $Excess\ TMEC = 2450$

- Apply equation (31), (33) and (34):

$$LCC_p = \left(\frac{2000}{1250} \right)^{\frac{9}{8}} \times (1.25 \times 10^{-4})$$

$$= 2.12 \times 10^{-4}$$

$$SPMC_p = \left(\frac{1500}{135} \right)^{\frac{9}{8}} \times (1.25 \times 10^{-4})$$

$$= 1.9 \times 10^{-3}$$

$$TMEC_p = \left(\frac{2450}{440} \right)^{\frac{9}{8}} \times (1.25 \times 10^{-4})$$

$$= 8.5 \times 10^{-4}$$

- Therefore,

$$Q'_{sys} = Q_{sys} + LCC_p + SPMC_p + TMEC_p$$

$$= 0.0125 + 2.1 \times 10^{-4} + 1.9 \times 10^{-3} + 8.5 \times 10^{-4}$$

$$= 0.01546$$

Figure 9.15 Application of the Cost Penalty Formulae

Each penalty is subsequently added to the system unavailability to give a sole fitness value for each design, i.e.

$$Q'_{SYS} = Q_{SYS} + LCC_P + STC_P + SPMC_P + TMEC_P + S_P \quad (9.34)$$

9.6.5 Selection

The fitness conversion method used in the selection process is akin to that described in section 7.4. Due to the complexity of the FDS, however, the range of fitness values that represent a design of ultimate interest is larger. As such, an extra category is incorporated and the bounds of each category modified. The 10 categories and their corresponding weight are portrayed in table 9.13.

Fitness domain					Weight
Upper limit		Category		Lower limit	
0.4	>	10	≥	0.2	1 ²
0.2	>	9	≥	0.15	2 ²
0.15	>	8	≥	0.1	3 ²
0.1	>	7	≥	0.08	4 ²
0.08	>	6	≥	0.06	5 ²
0.06	>	5	≥	0.04	6 ²
0.04	>	4	≥	0.03	7 ²
0.03	>	3	≥	0.02	8 ²
0.02	>	2	≥	0.01	9 ²
0.01	>	1	≥	0	10 ²

Table 9.13 Ten Categories plus Weights

Sampling proceeds in an identical manner to that of the HIPS optimisation with the exception that each string selected is stored within the database *npopo*.

9.6.6 Genetic Operator Action plus Update Routines

In view of the comparatively large area of the search space occupied by the maintenance test interval (MTI) parameter values, it was decided to isolate the action of genetic operators on these variables. To ease computation, the MTI parameters are stored on a separate binary string, referred to as string 2. The other parameters are stored in binary on string 1 (see figure 9.13).

Crossover and mutation is applied in the normal manner (described in section) to string 1 of each design stored in *npopo*. *Updateichrom* is then implemented to extract the specific integer values of a design's parameter set from its binary code.

Following extraction, the parameter set of each design must be checked for feasibility. The checks made and subsequent corrective action taken is carried out to:

- Ensure N does not exceed 4 or N is not greater than K . If so, N is reduced to 4 or K is set to the value of N respectively.
- Ensure P is not greater than 2. If so, regenerate P in the range 0 to 2.
- Ensure F is feasible as regards F_P and the upper limit 8:
 - If F_P is 100%, F should be in the range 1 to 8.
 - If F_P is 50%, F should be in the range 2 to 8.
 - If F_P is 33 1/3%, F should be in the range 3 to 8

If the above do not hold regenerate F .

- Ensure F_E is feasible as regards its limits and F :
 - If $F \leq 4$, F_E must be in the range 0 to 4.
 - If $F = 5$, F_E must be in the range 1 to 4.
 - If $F = 6$, F_E must be in the range 2 to 4.
 - If $F = 7$, F_E must be in the range 3 to 4.
 - If $F = 8$, F_E must be equal to 4.

If the above do not hold regenerate F_E .

- Ensure A is feasible as regards A_P and the upper limit 4:
 - If A_P is 100%, A should be in the range 1 to 4. Else regenerate A .
 - If A_P is 50%, A should be in the range 2 to 4. Else regenerate A .
- Ensure A_E is feasible as regards its limits and A :
 - If $A \leq 2$, A_E must be in the range 0 to 2.

- If $A = 3$, A_E must be in the range 1 to 2.
- If $A = 4$, A_E must be equal to 2.

If the above do not hold regenerate A_E

- Ensure W or D are not greater than 2. If so, regenerate the respective parameter in the range 0 to 2.

On completion of *Updateichrom* the contents of *npopo* are copied to *npopn*.

Inversion (as described in section 7.2.2) is then carried out on string 2 of each design in *npopn*. A routine termed *Updatemti* is subsequently implemented to extract the specific integer values of each design's MTI parameter set from the respective binary code. *Updatemti* uses the same principles as *Updateichrom* to achieve extraction. Feasibility checks to ensure that the MTI parameters lie within the stated ranges are not made within *Updatemti*. It is assumed that the penalty formulae will weed out designs whose test interval values are inappropriate.

The next step requires that the fitness information of the modified designs in both *npopo* and *npopn* is updated. The probability of system unavailability, the spurious trip frequency. Life cycle costs and associated penalised system unavailability are re-evaluated.

A new population is formed from both *npopo* and *npopn*. The particular design in record j , where $j = 1$ to n , of *npopo* and *npopn* differs only in the values assigned to its test interval parameters. A function is used to compare the penalised system unavailability of the j^{th} design in *npopo* with that of the j^{th} design in *npopn*. The fittest design is selected to enter the new population and as such, is transferred to the original storage structure, *opop*.

The algorithm to illustrate genetic operator action and the update routines is given in figure 9.16.

```

Genetic Operator Action
  for j = 1 to n - 1
    apply Crossover to string 1 of npopoj
  aor j = 1 to n
    apply Mutation to string 1 of npopoj
  apply Updateichrom to npopo
  for j = 1 to n
    npopnj = npopoj
  for j = 1 to n
    apply Inversion to string 2 of npopnj
  apply Updatemti to npopn
  Updatefitness (both npopo and npopn)
  for j = 1 to n
    If  $Q'_{sys}$  of npopdj ≤  $Q'_{sys}$  of npopnj
      opopj = npopoj
    Else
      opopj = npopnj

```

Figure 9.16 Genetic Operator Action and Update Routines

9.6.7 Output From GASSOPm

Output from GASSOPm is in accordance with that described in section 6.2.9. The raw fitness values are, however, more comprehensive. The system unavailability of each design plus the respective contributions of q_1 , q_2 , q_3 and q_4 are first described. The life cycle cost and its respective parts, i.e. SIC, STC, SPMC, SCMC and TMEC are then listed.

9.6.8 Results

To test GASSOPm 10 runs with a population of 20 strings over 100 generations were carried out. The mutation and crossover rate for each run was selected as 0.01 and 0.7 respectively. Each run required several hours.

The GA portrays significant convergence in average population fitness similar to that of the HIPS optimisation. This is demonstrated in table 9.14, which shows the population average fitness value of the first and last generation for each run.

GA Run No.	Initial Population Average Fitness	Final Population Average Fitness
1	0.177	0.0172
2	0.298	0.0306
3	0.332	0.0156
4	0.2	0.0297
5	0.164	0.0243
6	0.265	0.0223
7	0.186	0.0380
8	0.276	0.0218
9	0.238	0.0273
10	0.264	0.0232
Average Fitness		
Σ	0.24	0.025

Table 9.14. To Demonstrate Population Average Fitness Convergence

Typically, the greatest degree of convergence occurs in the first 30 generations. Average population fitness then fluctuates about a low value for the latter part of each run. Average population fitness per generation for runs 1, 7 and 9 is illustrated in figure 9.17.

Runs 8, 9 and 10 were implemented over a further 100 generation, i.e. 200 generations in total. As regards run number 9, average fitness after 200 generations was 0.0302 in contrast to 0.0273 after 100 generations. Average population fitness over 200 generations of the 9th run is illustrated in figure 9.18. The best design arose in generation 77. Neither greater population average fitness convergence nor a more optimal best design was achieved over the extended analysis period. Similar results were attained for runs 8 and 10.

KEY:

- GA run number 1
- GA run number 7
- GA run number 9

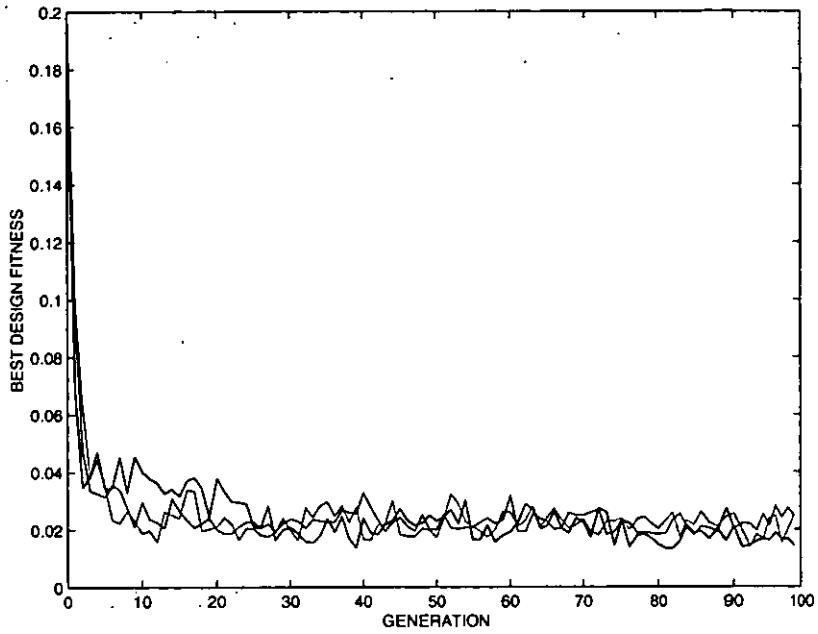


Figure 9.17 Average Population Fitness per Generation for Runs 1, 7 and 9

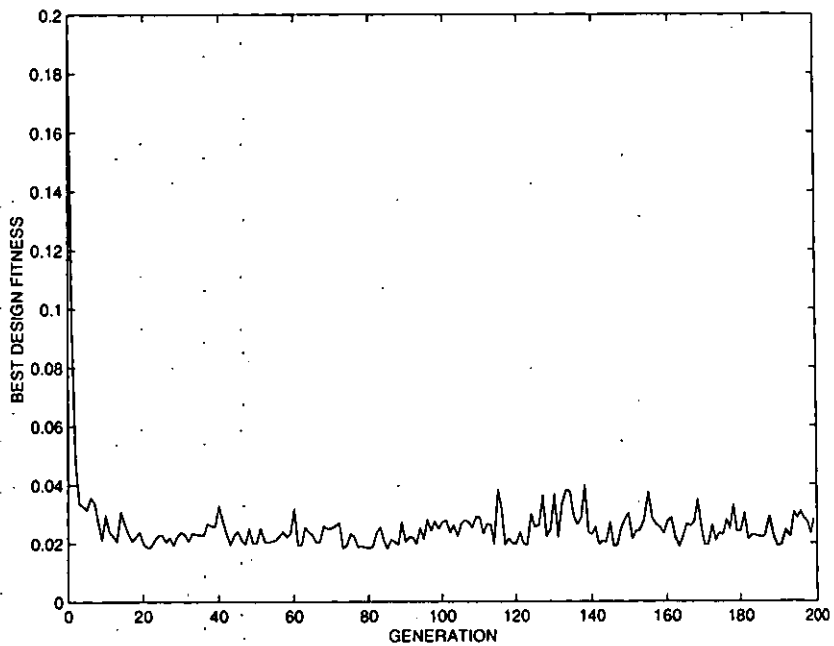


Figure 9.18 Average Population Fitness over 200 Generations for Run 9

The best string fitness of the fittest design in each generation typically fluctuates over the initial generations prior to leveling out towards the end of the run. Figure 9.19 illustrates the best string data for runs 1, 3 and 8.

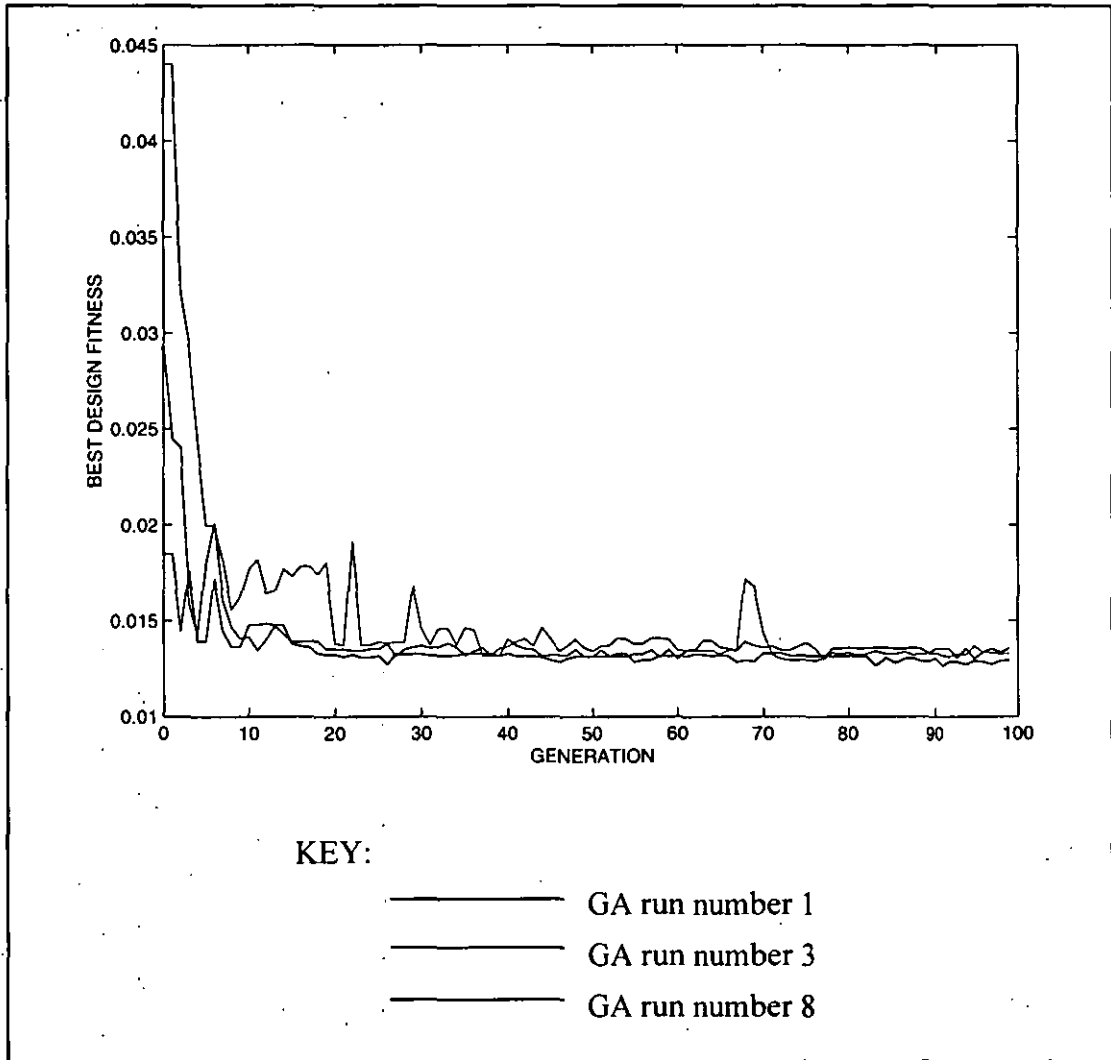


Figure 9.19 Best Design Fitness per Generation for Runs 1, 3 and 8

Table 9.15 shows the characteristics of the best design resulting from each run. The corresponding fitness data is portrayed in table 9.16. Table 9.17 specifies the amount and percentage by which the life cycle cost of each of the best designs varies about the life cycle cost limit, i.e. 125 000 units.

Best design resulting from run number...										
	1	2	3	4	5	6	7	8	9	10
<i>K/N</i>	1/1	1/3	1/2	1/3	1/4	1/2	3/4	1/4	1/2	2/3
<i>P</i>	1	1	2	1	1	2	1	1	1	3
<i>F_E/F</i>	3/6	3/6	3/5	1/3	1/3	2/4	1/3	2/5	1/3	3/4
<i>F_P</i>	50%	50%	100%	100%	100%	50%	100%	50%	100%	100%
<i>F_T</i>	1	2	1	2	1	2	2	1	1	2
<i>A_E/A</i>	1/2	1/2	1/2	2/4	2/4	2/4	2/4	1/2	2/4	1/2
<i>A_P</i>	100%	100%	100%	50%	50%	50%	50%	100%	50%	100%
<i>W</i>	3	3	3	2	3	3	3	1	2	2
<i>D</i>	3	3	2	3	3	2	3	3	3	2
<i>C</i>	2	2	2	2	2	2	2	2	2	2
<i>θ_P</i>	24	18	8	11	11	23	13	16	27	8
<i>θ_R</i>	1	1	3	1	1	1	1	1	1	1
<i>θ_D</i>	3	3	3	3	3	3	3	3	3	3
<i>θ_{PM}</i>	18	18	15	15	18	18	15	12	18	18

Table 9.15. Characteristics of the Best Designs

The best overall design for the FDS arose in the 2nd run and has a system unavailability of 1.263×10^{-2} . This design is over 98.73% available. The best design arising in the 1st has very similar characteristics. It differs primarily in that 1 as opposed to 3 pressure sensors are included and the firewater pump is of type 2.

	The fitness of the best designs in runs...									
	1	2	3	4	5	6	7	8	9	10
STC	9222.8	8759.3	12683.8	11688.7	9740.6	9132.2	11688.7	12074.3	9740.6	8795.5
SPMC	8994.6	11123.8	17467.9	16543.5	16543.5	10224.5	14399.5	10819.6	8284.9	16294.6
TMEC	26934.3	29640.7	38272.8	34647.1	33135.9	28164.4	32502.4	29345.5	24876.6	33916.1
LCC	116480	120386	120795	123643	122231	125237	121598	109691	113871	106928
F_{sys}	0.29	0.403	0.4641	0.243	0.243	0.464	0.2189	0.243	0.342	0.219
Q₁	4.12e-6	3.53e-6	1.07e-5	3.53e-6	3.53e-6	3.54e-6	3.53e-6	2.53e-6	3.53e-6	3.63e-6
Q₂	1.91e-3	1.89e-3	2.08e-3	2.19e-3	2.22e-3	2.9e-3	2.22e-3	2.36e-3	2.34e-3	2.18e-3
Q₃	1.12e-3	1.11e-3	1.13e-3	1.09e-3	1.1e-3	1.12e-3	1.1e-3	1.11e-3	1.13e-3	1.09e-3
Q₄	9.0e-3	9.63e-3	9.7e-3	9.7e-3	9.63e-3	9.7e-3	9.63e-3	9.75e-4	9.7e-3	9.8e-3
Q_{sys}	1.267e-2	1.263e-2	1.292e-2	1.3e-2	1.295e-2	1.374e-2	1.295e-2	1.32e-2	1.32e-2	1.3e-2
Q'_{sys}	1.267e-2	1.263e-2	1.292e-2	1.3e-2	1.295e-2	1.376e-2	1.295e-2	1.32e-2	1.32e-2	1.3e-2

Table 9.16 Fitness Corresponding to the Best Designs in Table 9.15

LCC – 125000	8520	4614	4205	1357	2769	-237	3402	15309	11129	18072
% diff	6.8	3.7	3.3	1.1	2.2	-0.2	2.7	12.2	8.9	14.4

Table 9.17 Difference of Each Designs LCC about the Life Cycle Cost Constraint

9.6.8.1 Discussion of Results

GASSOPm is achieving its objective to find fit domains in the search space. Many similar parameter combinations are repeated throughout the best designs portrayed in table 9.15. As regards the deluge system, both the water and AFFF deluge valve are predominantly of type 3. The pipework is consistently of the non-corrosion resistant material, i.e. type 2.

A recurring combination for the firewater pump system is the inclusion of 3 firewater pumps, 1 electrically powered and 2 diesel driven. As regards the AFFF pump system, two combinations repeatedly arising are the inclusion of 2 100% pumps, 1 electric and 1 diesel, and 4 50% pumps, 2 electric and 2 diesel. Typically, the fittest designs portray balance in the number of electric to diesel pumps, particularly regarding those of 50% capacity.

Failure of the distribution network, i.e. Q_I , is consistently very low (in the magnitude 3×10^{-6}). The contribution of Q_I to the overall system unavailability of the design is, therefore, less significant. This is a likely reason for the marked variety in K and N . The pressure transmitters are predominantly of type 1, thus, preventing the number of spurious trip occurrences from exceeding its limit of 0.75 per year.

A strong pattern arises in the values assigned to the maintenance test interval parameters. The maintenance test interval for the ringmain is set as 1 week for all but one of the best designs. The deluge skid is consistently tested at 3 monthly intervals. In contrast, the test interval between preventative maintenance tends to be at the higher end of its range, i.e. 15 to 18 months. Greater variations exist regarding θ_p .

The total life cycle cost of each of the best designs approaches the limit of 125000 units. As portrayed in table 9.17, the majority of the best designs make almost optimal use of the available resources.

CHAPTER 10

A SEARCH LOGIC APPROACH

10.1 Introduction

Having reviewed recent research and literature in the area of optimisation it is apparent that the safety system design optimisation problem is unique and as such requires an alternative approach to the traditional optimisation techniques. In the main difficulties arise due to the lack of an accurate explicit objective function over the entire, or even a relatively small part, of the search space. Further complications occur as a result of the integer stipulation assigned to all of the design variables, in particular when the integer values are low and their restricted range small. Attention must also be given to constraints limiting the available resources. In addition, certain design variables are dependent on the values assigned to other variables. An optimisation method, which manipulates the variables independently, may result in an infeasible optimal design.

10.2 Applicable Features of the Optimisation Literature

Certain optimisation techniques address one or more of the difficulties stated above. A specialised method, or methods, could, thus, be created to solve the safety system design optimisation problem using an amalgamation of applicable features. The following is a discussion of innovative features and the particular issue they lend themselves to overcoming.

The use of random sampling of decision variables within their specified range is applied in much of the random search literature. This feature is a simple yet effective means of establishing a feasible solution vector. It is both easy to program and incorporate within a general optimisation algorithm. In addition, random sampling encourages global consideration of the search space, thus combating premature convergence or a localised approach.

The combinatorial heuristic method (Ref. 69) introduces an innovative approach to fixing design variables using a look-ahead search, that is a means to incorporate interaction with the next variable. The look-ahead search is particularly applicable for integer design variables that lie within a small restricted range. Of most importance is that such an approach is able to consider an element of dependency between the system variables.

The use of tolerable slacks (or fuzzy boundaries) in both Misra's paper (Ref. 60) and the method of rotating co-ordinates developed further by Rosenbrock to include constraints (Ref. 1) introduces a certain leeway either side of the constraint boundaries, thus representing the real decision making process. In industry a small extension on resources is deemed acceptable if the resulting design shows significant improvement. As stated in reference 60, an optimal design will in the main fully utilise all available resources. For this reason the incorporation of tolerable slacks ensures that time and effort is not lost analysing the interior of the search space. In effect this mimics the simplex method used in linear programming without incorporating the mathematical complexity. In addition, the use of tolerable slacks shifts the emphasis of function evaluations from the objective function to the constraints. This is of particular importance in the absence of an explicit objective function.

Searching from a population of trial points significantly improves the global aspects of an approach. Techniques about a specific trial point have a tendency to lead to local hill climbing and attainment of a local minimum only. Searching from a population of points is particularly advantageous when little is known about the search space of the design optimisation problem. In such cases the use of global optimisation techniques are dictated.

Knowledge of the search space is established via evaluations of both constraints and objective functions. Computer time and effort is assigned to this task. It is wasteful to use only the best values calculated and reject all others. As regards the safety system optimisation our knowledge of the search space is accumulated from function values alone. A method that makes use of the history of function evaluations to steer the search is, therefore, essential. Both CRS and PRS (Ref's 4 and 84) use a clustering or pattern recognition type approach. Clusters of high performance designs imply a high quality area of the search space, which deserves

further attention. Conversely, a cluster of unfit design vectors implies that continued search in that area is most likely to be fruitless.

10.3 A Logical Search Approach

It was decided to apply a logical search approach to the HIPS optimisation using primarily motivation from Misra's technique (Ref. 59) and the combinatorial heuristic method (Ref. 69) introduced in sections 3.2.6 and 3.5.4 respectively. Specifically, random sampling and a look-ahead search are combined with the use of tolerable slacks about the constraints to improve the efficiency of the search.

The design variables and their specific stated ranges are specified in section 5.1 of this thesis. The range of each maintenance test interval parameter is however, modified to 1 to 26 as opposed to 4 to 104 weeks, where 1 corresponds to 4 weeks, 2 to 8 weeks and so on. This modified range is used to limit the number of function evaluations required yet retain consideration of the full range of test intervals.

A look-ahead search is carried out about each of the variables N_1 , N_2 , K_1 , K_2 , E and H . Interaction with the type variables, V and P , and each maintenance test interval parameter, θ_1 and θ_2 , is considered within each look-ahead search.

Tolerable slacks must be established about each of the constraints. It is decided to set upper bounds equivalent to each limit, therefore ensuring feasibility of the resulting optimal design. The bands are assigned as follows:

- $800 \text{ units} \leq \text{cost} \leq 1000 \text{ units}$
- $110 \text{ hours} \leq \text{MDT} \leq 130 \text{ hours}$
- $0.6 \text{ per year} \leq \text{spurious trip frequency} \leq 1 \text{ per year}$

10.3.1 The Logical Search Algorithm

The steps of the algorithm are automated in a routine termed *Search_logic* and proceeds as follows:

- Step 1.** Set the iteration number to 0, i.e. $j = 0$.
- Step 2.** The initial point $\mathbf{x}^{(j)}$ is randomly generated. $Q_{STS}^j = f(\mathbf{x}^{(j)})$ is evaluated. If $\mathbf{x}^{(j)}$ violates any constraint due to excess cost, MDT or spurious trip frequency Q_{STS}^j is penalised in the normal manner (described in section 6.2.3.1).
- Step 3.** Establish a variable ordering, i.e. the order in which the variables are considered in the look-ahead search. Initially the ordering is chosen to be $\{N_1, K_1, E, H, N_2, K_2, E, K_1, N_1, H, K_2, N_2\}$. The choice of ordering is somewhat arbitrary and as such offers a significant degree of leeway. The variables adjacent to one another are however, those that are most likely to interact, therefore aiding the look-ahead search. In the main an optimal design is established following a single iteration of each variable. An additional iteration of each variable is however, implemented and the variable order shuffled. This serves two purposes: consideration of further likely interactions and confirmation of termination.
- Step 4.** Each variable i , where $i = 1, \dots, n$ as denoted by the variable ordering in step 3, is considered in turn and the following loop executed:
- A) Establish the feasible range of i in view of the fixed values assigned to the other variables in the design vector $\mathbf{x}^{(j)}$. (these variables are fixed either as a result of the initial design vector or via a previous loop of step 4).
 - B) Conduct a look-ahead search involving variable $i + 1$ and either V or P . If the $i + 1^{th}$ variable concerns the pressure transmitters then P is involved in the look-ahead search else V is the variable considered. The groupings are specified below:

$i + 1^{th}$ Variable	Type Variable
N_1, N_2, K_1 or K_2	P
E or H	V

- i) Establish all possible combinations of the feasible values of i , the feasible values of $i + 1$ and the type variable (i.e. type 1 or type 2 for V or P).
- ii) For each combination, i.e. modified design vector $\mathbf{x}_{i,i+1,T}^{(j)}$, where T denotes the type variable P or V , the associated

system unavailability $Q_{i,i+1,T}^{(j)}$ is established in the following manner:

- a) Evaluate system cost, c . Check that the resulting value lies within the tolerable slacks for cost. If $c < 800$ or $c > 1000$ units assign a value of 1 to the system unavailability of the design in question and repeat step 4Biia with the next modified design vector. If all combinations have been considered go to step 4Biii.
 - b) Evaluate the spurious trip frequency, s . Ensure the resulting value lies within the tolerable slacks for the trip rate. If not proceed as in step 4Biia.
 - c) For each combination of maintenance test intervals calculate the design's MDT. If the MDT falls within the respective tolerable slacks evaluate the system unavailability associated with the design. Note that the number of system unavailability function evaluations is limited by the use of tolerable slacks and the modified range for the maintenance test interval design variables. Retain the maintenance test interval combination that results in the most optimal system unavailability for the design in question. Store the design vector and associated system unavailability.
- iii) Compare the system unavailability values, $Q_{i,i+1,T}^{(j)}$, associated with each modified design vector and select the best, denote as $\mathbf{x}_b^{(j)}$.
- iv) Let $\mathbf{x}^{(j)} = \mathbf{x}^{(j+1)}$. If $i = n - 1$ go to step 4Bvi, else fix the i^{th} variable and the type variable in $\mathbf{x}^{(j+1)}$ using their respective values in $\mathbf{x}_b^{(j)}$. Note that $\mathbf{x}_b^{(j)}$ may be fitter than $\mathbf{x}^{(j+1)}$ as the $i + 1^{\text{th}}$ variable is not fixed in the latter vector. A subsequent execution of step 4 fixes the $i + 1^{\text{th}}$ variable where its value is equivalent to that in $\mathbf{x}_b^{(j)}$ unless a better alternative is established.
- v) Return to step 4A with $i = i + 1$ and $j = j + 1$.
- vi) Fix the i^{th} , the $i + 1^{\text{th}}$ and the type variable in $\mathbf{x}^{(j+1)}$ using their respective values in $\mathbf{x}_b^{(j)}$.

vii) Increment j , i.e. $j = j + 1$ and store the resulting value in $Term$, i.e. $j = Term$. Proceed to step 5.

Step 5. The resulting best design, $\mathbf{x}^{(j)}$, considers maintenance test interval values for subsystems 1 and 2 in 4 weekly intervals. Optimise use of the available MDT resource for $\mathbf{x}^{(j)}$ over the full range of test interval values.

Step 6. Check for termination. Ensure design vectors $\mathbf{x}^{(j)}$, for $j = Term - 3, \dots, Term$ are consistent. Else re-initiate step 4 with $i = 1$.

The order in which the variables are considered will affect the steps of the algorithm and may result in convergence to a different optimal design vector. The ordering of variables is considered in more detail in section 10.3.3.

There are various means to terminate the algorithm. Rather than specifying a fixed number of iterations the program could be altered to terminate when, for example, the optimal design has consistently attained a specified termination criteria over, for example, k iterations, where the ordering is repeated up to this point.

10.3.2 Detailed Results from a Run of *Search_logic*

The following is a detailed run through of the *Search_logic* algorithm, specifying the output resulting from each step:

Step 1. $j = 0$

Step 2. Randomly generate $\mathbf{x}^{(0)}$

	E	K_1/N_1	H	K_2/N_2	P	V	θ_1	θ_2	Q_{SYS}	Q'_{SYS}
$\mathbf{x}^{(0)}$	1	2/3	1	2/2	1	1	23	36	0.00137	0.0061

Step 3. Establish variable ordering, $\{N_1, K_1, E, H, N_2, K_2, E, K_1, N_1, H, K_2, N_2\}$.

Step 4. Consider N_1 :

A) Feasible values of N_1 are $\{2, 3, 4\}$.

B) K_1 is the $i + 1^{th}$ variable and as such P is the type variable.

i) All possible combinations of $\mathbf{x}_{N_1, K_1, P}^{(0)}$ are shown in table 10.1.

Note that $E = 1, H = 1, K_2/N_2 = 2/2, V = 1$ remain fixed.

- ii) Fitness evaluations are shown in table 10.1. Note that in all but rows 5 and 11 the spurious trip frequency lies outside its respective tolerable slacks and hence, values are not assigned to the maintenance test interval parameters.

$\mathbf{x}_{N_1, K_1, T}^{(0)}$	K_1/N_1	P	θ_1	θ_2	Cost	F_{SYS}	MDT	Q_{SYS}
1	1/2	1	-	-	902	0.551	-	1
2	1/2	2	-	-	862	1.51	-	1
3	2/2	1	-	-	902	0.289	-	1
4	2/2	2	-	-	862	0.295	-	1
5	1/3	1	48	20	922	0.681	130	0.0016
6	1/3	2	-	-	872	2.11	-	1
7	2/3	1	-	-	922	0.289	-	1
8	2/3	2	-	-	872	0.301	-	1
9	3/3	1	-	-	922	0.290	-	1
10	3/3	2	-	-	872	0.292	-	1
11	1/4	1	52	20	942	0.812	126	0.0017
12	1/4	2	-	-	882	2.71	-	1
13	2/4	1	-	-	942	0.551	-	1
14	2/4	2	-	-	882	1.51	-	1
15	3/4	1	-	-	942	0.289	-	1
16	3/4	2	-	-	882	0.292	-	1
17	4/4	1	-	-	942	0.289	-	1
18	4/4	2	-	-	882	0.292	-	1

Table 10.1 Fitness Evaluations For Designs Considering N_1

iii) The best design, $\mathbf{x}_b^{(0)}$, arises in the fifth row of table 10.1.

iv) $\mathbf{x}^{(0)} = \mathbf{x}^{(1)}$. Fix $N_1 = 3$ and $P = 1$ in $\mathbf{x}^{(1)}$.

v) $i = 2$, i.e. consider K_1 and $j = 1$

Step 4. Consider K_1 :

	E	K_1/N_1	H	K_2/N_2	P	V
$\mathbf{x}^{(1)}$	1	2/3	1	2/2	1	1

A) Feasible values of K_1 are $\{1, 2, 3\}$.

B) E is the $i + 1^{\text{th}}$ variable and as such, V is the type variable.

- i) All possible combinations of $\mathbf{x}_{K_1, E, V}^{(1)}$ are shown in table 10.2, where $N_1 = 3$, $H = 1$, $K_2/N_2 = 2/2$ and $P = 1$ remain fixed.
- ii) Fitness evaluations are shown in table 10.2. Note that only the designs in row 3 and 4 lie in the tolerable slacks for cost and spurious trip frequency.

$\mathbf{x}_{K_1, E, V}^{(1)}$	K_1	E	V	θ_1	θ_2	Cost	F_{SYS}	MDT	Q_{SYS}
1	1	0	1	-	-	652	-	-	1
2	1	0	2	-	-	602	-	-	1
3	1	1	1	48	20	922	0.681	130	0.0016
4	1	1	2	48	16	822	0.847	130	0.0014
5	1	2	1	-	-	1192	-	-	-
6	1	2	2	-	-	1042	-	-	-
7	2	0	1	-	-	652	-	-	-
8	2	0	2	-	-	602	-	-	-
9	2	1	1	-	-	922	0.289	-	-
10	2	1	2	-	-	822	0.455	-	-
11	2	2	1	-	-	1192	-	-	-
12	2	2	2	-	-	1042	-	-	-
13	3	0	1	-	-	652	-	-	-
14	3	0	2	-	-	602	-	-	-
15	3	1	1	-	-	922	0.289	-	-
16	3	1	2	-	-	822	0.455	-	-
17	3	2	1	-	-	1192	-	-	-
18	3	2	2	-	-	1042	-	-	-

10.2 Fitness Evaluations for Designs Considering K_1

- iii) the best design, $\mathbf{x}_b^{(1)}$, arises in row 4 of table 10.2.
- iv) $\mathbf{x}^{(1)} = \mathbf{x}^{(2)}$ Fix $K_1 = 1$ and $V = 2$ in $\mathbf{x}^{(2)}$.
- v) $i = 3$, i.e. consider E and $j = 2$

	E	K_1/N_1	H	K_2/N_2	P	V
$\mathbf{x}^{(2)}$	1	1/3	1	2/2	1	2

Step 4. Consideration is given to each variable in turn. In total step 4 is carried out $n - 1$. The modifications to the initial design following each

iteration are specified in table 10.3. Note that Q_{SYS} is the system unavailability of the best design vector, $\mathbf{x}_b^{(j)}$, resulting from an iteration of step 4.

Fix	$\mathbf{x}^{(j)}$	E	K_1/N_1	H	K_2/N_2	P	V	$Q(\mathbf{x}_b^{(j)})$
-	$\mathbf{x}^{(0)}$	1	2/3	1	2/2	1	1	0.0061
N_1	$\mathbf{x}^{(1)}$	1	2/3	1	2/2	1	1	0.00162
K_1	$\mathbf{x}^{(2)}$	1	1/3	1	2/2	1	2	0.00143
E	$\mathbf{x}^{(3)}$	0	1/3	1	2/2	1	2	0.00093
H	$\mathbf{x}^{(4)}$	0	1/3	2	2/2	1	2	0.00079
N_2	$\mathbf{x}^{(5)}$	0	1/3	2	2/3	1	2	0.00079
K_2	$\mathbf{x}^{(6)}$	0	1/3	2	2/3	1	2	0.00079
E	$\mathbf{x}^{(7)}$	0	1/3	2	2/3	1	2	0.00079
K_1	$\mathbf{x}^{(8)}$	0	1/3	2	2/3	1	2	0.00079
N_1	$\mathbf{x}^{(9)}$	0	1/3	2	2/3	1	2	0.00079
H	$\mathbf{x}^{(10)}$	0	1/3	2	2/3	1	2	0.00079
$K_1 \& N_1$	$\mathbf{x}^{(11)}$	0	1/3	2	2/3	1	2	0.00079

Table 10.3 Design Alterations through execution of step 4

Step 5. The resulting optimal design is:

	E	K_1/N_1	H	K_2/N_2	P	V	θ_1	θ_2	Cost	F_{SYS}	MDT	Q_{SYS}
$\mathbf{x}^{(11)}$	0	1/3	2	2/3	1	2	40	24	842	0.847	130	7.92e-3

Step 6. It can be seen from table 10.3 that iterations 5 to 11 portray consistent predictions of the optimal design.

10.3.2.1 Further Results

Table 10.4 specifies the results of four further runs of *Search_logic*. In each case the initial design and the final optimal design achieved are stated, where the final design is given in bold. Table 10.4 specifies the fitness components associated with the optimal design resulting from each run. Execution of the program requires an order of minutes.

Run No.	E	K_1/N_1	H	K_2/N_2	P	V	θ_1	θ_2	Q_{SRS}	Q'_{SRS}	
1	$\mathbf{x}^{(0)}$	1	2/3	1	2/2	1	1	23	36	0.00137	0.0061
	$\mathbf{x}^{(1)}$	0	1/3	2	2/3	1	2	40	24	7.92e-4	7.92e-4
2	$\mathbf{x}^{(0)}$	1	4/4	1	3/3	1	1	70	50	0.0094	0.0094
	$\mathbf{x}^{(1)}$	0	1/4	2	2/4	1	2	28	36	8.26e-4	8.26e-4
3	$\mathbf{x}^{(0)}$	2	1/1	2	4/4	1	2	40	65	0.0036	0.0067
	$\mathbf{x}^{(1)}$	0	1/1	2	1/2	1	1	28	40	0.001	0.001
4	$\mathbf{x}^{(0)}$	1	1/3	1	2/3	2	2	89	28	0.004	0.014
	$\mathbf{x}^{(1)}$	0	1/4	2	2/4	1	2	28	36	8.26e-4	8.26e-4
5	$\mathbf{x}^{(0)}$	2	1/1	1	1/1	2	2	70	50	0.013	0.11
	$\mathbf{x}^{(1)}$	0	1/1	2	1/3	1	2	32	28	8.31e-4	8.31e-4

Table 10.4 Results from Runs of *Search_Logic*

Run No.	Cost	MDT	F_{SRS}	Q_{SRS}	Q'_{SRS}	
1	$\mathbf{x}^{(1)}$	842	130	0.847	7.92e-4	7.92e-4
2	$\mathbf{x}^{(1)}$	882	129.6	0.978	8.26e-4	8.26e-4
3	$\mathbf{x}^{(1)}$	882	129.1	0.681	0.001	0.001
4	$\mathbf{x}^{(1)}$	882	129.6	0.978	8.26e-4	8.26e-4
5	$\mathbf{x}^{(1)}$	802	128.6	0.977	8.31e-4	8.31e-4

Table 10.5 Fitness of Optimal Designs from Runs of *Search_logic*

The best design is achieved in run number 1 and has a system unavailability of 7.92×10^{-4} . The optimal design achieved in each run is highly fit, yet there is significant variety in the parameter set of each optimum design vector. The resulting design is highly dependent on the choice of the initial design point.

10.3.3 Order of Variables in *Search_logic*

The order in which the variables are considered in the look-ahead search determines the type of interactions in the parameter set that are analysed. The particular ordering used in section 10.3.1 is chosen primarily to focus on

interactions between the variables in subsystem 1 and interactions between the variables in subsystem 2. The interaction between the number of ESD valves and the number of HIPS valves is also investigated.

Altering the ordering in the automated Serach_logic algorithm is a straight forward process. As such, two alternative orderings are considered, termed *altord1* and *altord2*.

10.3.3.1 Alternative Ordering Number 1, *altord1*

It is noted that the ordering in section 10.3.1 analysis little interaction between the pressure transmitters of each subsystem. *Altord1* aims to rectify this and hence, adopts the variable ordering $\{N_1, K_1, E, H, N_2, K_2, E, K_1, N_1, N_2, K_2, H\}$.

Implementing *Search_logic* with *altord1* from the same design points as those in the previous results section gives the optimal designs specified in table 10.6. The associated fitness information for each optimal design is stated in table 10.7.

Run No.	E	K_1/N_1	H	K_2/N_2	P	V	θ_1	θ_2	Q_{SYS}	Q'_{SYS}	
1	$\mathbf{x}^{(0)}$	1	2/3	1	2/2	1	1	23	36	0.00137	0.0061
	$\mathbf{x}^{(1)}$	0	1/3	2	2/3	1	2	40	24	7.92e-4	7.92e-4
2	$\mathbf{x}^{(0)}$	1	4/4	1	3/3	1	1	70	50	0.0094	0.0094
	$\mathbf{x}^{(1)}$	0	1/3	2	2/3	1	2	40	24	7.92e-4	7.92e-4
3	$\mathbf{x}^{(0)}$	2	1/1	2	4/4	1	2	40	65	0.0036	0.0067
	$\mathbf{x}^{(1)}$	0	1/2	2	2/3	1	2	40	24	7.92e-4	7.92e-4
4	$\mathbf{x}^{(0)}$	1	1/3	1	2/3	2	2	89	28	0.004	0.014
	$\mathbf{x}^{(1)}$	0	1/3	2	2/3	1	2	40	24	7.92e-4	7.92e-4
5	$\mathbf{x}^{(0)}$	2	1/1	1	1/1	2	2	70	50	0.013	0.11
	$\mathbf{x}^{(1)}$	0	1/2	1	1/2	1	2	34	26	7.23e-	7.23e-4

Table 10.6 Results from Runs of *Search_Logic* Using *Altord1*

Run No.		Cost	MDT	F_{SYS}	Q_{SYS}	Q'_{SYS}
1	$\mathbf{x}^{(11)}$	842	130	0.847	7.92e-4	7.92e-4
2	$\mathbf{x}^{(11)}$	842	130	0.847	7.92e-4	7.92e-4
3	$\mathbf{x}^{(11)}$	822	128.7	0.717	7.92e-4	7.92e-4
4	$\mathbf{x}^{(11)}$	842	130	0.847	7.92e-4	7.92e-4
5	$\mathbf{x}^{(11)}$	802	129.3	0.977	7.23e-4	7.23e-4

Table 10.7 Fitness of Optimal Designs Using *Altord1* within *Search_logic*

Using *altord1*, consistently fit designs were achieved in each run. There is very little variety between the optimal design vectors, where in the main differences arise in the number of pressure transmitters and the number of transmitters required to trip the system. The fittest design arises in run number 5 and has a system unavailability of 7.23×10^{-4} .

10.3.3.2 Alternative Ordering Number 2, *Altord2*

Due to the fact that the look-ahead search incorporates only variables i and $i + 1$ certain combinations of pressure transmitters are never explored. In addition, it is noted in the previous results that little is achieved by considering variable H a second time. For this reason an ordering is established which considers a three-way interaction between N_1 , N_2 and K_2 in its latter stage. To compensate for the additional computer effort required n is reduced by 1. Hence, *altord2* is $\{N_1, K_1, E, H, N_2, K_2, E, K_1, N_1 \& N_2 \& K_2\}$, where the 9th iteration of step 4 in the search logic algorithm considers all possible combinations of N_1 , N_2 and K_2 .

Results Using *Altord2*

Implementing *Search_logic* with *altord2* ordering from the same design points as those in the previous results sections gives the optimal designs specified in table 10.8. The associated fitness information for each optimal design is stated in table 10.9.

Run No.		E	K_1/N_1	H	K_2/N_2	P	V	θ_1	θ_2	Q_{SYS}	Q'_{SYS}
1	$\mathbf{x}^{(0)}$	1	2/3	1	2/2	1	1	23	36	0.00137	0.0061
	$\mathbf{x}^{(1)}$	0	1/2	1	1/2	1	2	34	26	7.23e-4	7.23e-4
2	$\mathbf{x}^{(0)}$	1	4/4	1	3/3	1	1	70	50	0.0094	0.0094
	$\mathbf{x}^{(1)}$	0	1/2	1	1/2	1	2	34	26	7.23e-4	7.23e-4
3	$\mathbf{x}^{(0)}$	2	1/1	2	4/4	1	2	40	65	0.0036	0.0067
	$\mathbf{x}^{(1)}$	0	1/2	1	1/2	1	2	34	26	7.23e-4	7.23e-4
4	$\mathbf{x}^{(0)}$	1	1/3	1	2/3	2	2	89	28	0.004	0.0014
	$\mathbf{x}^{(1)}$	0	1/2	1	1/2	1	2	34	26	7.23e-4	7.23e-4
5	$\mathbf{x}^{(0)}$	2	1/1	1	1/1	2	2	70	50	0.013	0.11
	$\mathbf{x}^{(1)}$	0	1/2	1	1/2	1	2	34	26	7.23e-4	7.23e-4

Table 10.8 Results from Runs of Search_Logic Using Altord2

Run No.		Cost	MDT	F_{SYS}	Q_{SYS}	Q'_{SYS}
1	$\mathbf{x}^{(1)}$	802	129.3	0.977	7.23e-4	7.23e-4
2	$\mathbf{x}^{(1)}$	802	129.3	0.977	7.23e-4	7.23e-4
3	$\mathbf{x}^{(1)}$	802	129.3	0.977	7.23e-4	7.23e-4
4	$\mathbf{x}^{(1)}$	802	129.3	0.977	7.23e-4	7.23e-4
5	$\mathbf{x}^{(1)}$	802	129.3	0.977	7.23e-4	7.23e-4

Table 10.9 Fitness of Optimal Designs Using Altord2 within Search_logic

Each run results in the same optimal design vector. This design is highly fit, with a system unavailability of 7.23×10^{-4} .

10.3.4 Discussion of Results

In the main, the resulting designs are highly fit. An ordering that considers greater interaction between the design variables is beneficial. Altord1, considers an alternative ordering on the second iteration of each design variable as opposed to merely repeating the variable ordering. Interaction between the pressure

transmitters of each subsystem is, therefore considered. Altord2 considers interaction between a number of variables at once as opposed to simply looking ahead to the next variable. This requires greater computer effort, which would be exacerbated the more complex the system under consideration, however for this relatively simple system it proves beneficial.

The HIPS system has characteristics which lend themselves well to the search logic optimisation procedure. Firstly, the system has only a small number of design variables and the range covered by these variables is also small. In addition, the variables are segregated into two halves, those concerned with subsystem1 and those with subsystem 2. There is obviously interaction between all the design variables, however the simplicity of the system lends itself well to establishing an efficient order for the variables to be considered in the look-ahead search.

The search logic approach appears to be both highly efficient and accurate, however, its scope of applicability is limited. It is in effect a local approach, which is highly dependent on the choice of the initial design vector. This dependency on the start point will increase as the complexity of the optimisation problem increases. In addition, a high degree of knowledge concerning the system is required to establish both the order in which to consider the variables in the look-ahead search and the boundaries used for the tolerable slacks.

To summarise, the search logic approach has proven to be effective for the HIPS optimisation, however, its applicability to a larger array of problems is limited and it is likely that it will only be useful as a highly localised technique.

CHAPTER 11

A STATISTICALLY DESIGNED EXPERIMENT

11.1 Introduction

The engineering design process involves setting values of a potentially large number of decision variables. Studying the effects of these variables one at a time or by trial and error is the common approach to the decision process. This gives rise to an inefficient, unstructured design process.

An alternative to this *ad hoc* approach is to use a structured method such as that offered by *Statistically Designed Experiments* (SDE) to investigate the best settings for the variables (Ref.'s 10, 38). Statistical design of experiments means making many purposeful design changes at once and conducting several tests and evaluations before decisions are taken as to what the next steps in the development process should be. Experimental design is used in many industrial applications, one typical example being production processes. Usually the experimental results will be subject to variability or noise. In the applications presented in this chapter the experimental design concepts are being used to try to get a starting point for the optimisation. The better the starting point the more efficient the optimisation. In effect, this approach is expected to achieve 80% of the work required to reach an optimal design using 20% of the effort. To assess the design performance a computer analysis (fault trees) is being used and so the results are not subject to variability and each experiment only needs to be performed once.

The variables to be changed are called *factors*. Prior to conducting a statistically designed experiment (otherwise known as a *factorial experiment*) all factors must be identified and the possible values of each factor determined. These possible values are termed *levels*. The performance of the engineering system is characterised by a measurement of some aspect or function termed the *response*.

11.2 Orthogonal Arrays

Purposeful design changes are achieved using special matrices called *orthogonal arrays*. Consider, for example, a particular system consisting of 3 parameters: number of valves (A), valve type (B), and number of pressure transmitters (C). As regards each parameter 2 settings are chosen to cover the range of interest. These factor levels define the experimental region and are listed in table 11.1. The goal is to minimise system unavailability and as such the response is the probability of system failure (Q_{SYS}).

Factor	Levels	
	1	2
A. No. of valves	2	3
B. Valve type	1	2
C. No. of PT's	4	5

Table 11.1 Factor Levels

Expt. No.	Columns			Q_{SYS}
	1	2	3	
1	1	1	1	0.2
2	1	2	2	0.4
3	2	1	2	0.3
4	2	2	1	0.2

Table 11.2 $L_4(2^3)$ Orthogonal Array

The matrix experiment to define the experimental plan is stated in table 11.2. It consists of 4 individual experiment designs corresponding to the four rows. The 3 columns of the matrix represent the 3 factors. The entries in the matrix represent the levels of the factors, whose values are indicated in table 11.1. Thus, the first design consists of 2 valves of type 1 and 4 pressure transmitters. The matrix experiment is the standard orthogonal array L_4 (Ref. 64). For any pair of columns all combinations of factor levels occur and they occur an equal number of times. This is called the *balancing property* and it implies orthogonality. This balancing property inherent in orthogonal arrays enables the identification of which design changes make the difference to the performance of the design.

A matrix experiment consisting of all possible combinations of the factors is termed a 'full-factorial'. The run size for a 2-level full factorial design in k factors is 2^k , which quickly becomes prohibitive for a moderate amount of factors. The problem is exacerbated for factors with more than 2 levels. A subset or fraction of the full

factorial design (a fractional factorial) tends to be used unless there are only a few potentially important factors to be studied. In the example above the orthogonal array represents a 'one-half fraction' of the 2^3 design.

11.3 Steps in a Statistically Designed Experiment

An outline of the statistically designed experimental procedure is as follows:

- 1) Definition of the aim of the experimental plan and selection of the response variable.
- 2) Selection of the factors that will be changed and the levels that will be used.
- 3) Choice of a matrix experiment such that the plan has a systematic and balanced pattern.
- 4) Analysis of the data from the matrix experiment to determine the effects of the various factors. One of the benefits of using orthogonal arrays is the simplicity of data analysis. The effects of the various factors can be determined by computing simple averages. The estimates of the factor effects are then used to determine the optimum factor settings.
- 5) Summary of the results in a response table and effects plot. The use of engineering knowledge and common sense is essential in interpreting the results.
- 6) Validation of conclusions from the experiment using follow-up runs and confirmation testing. Experimentation is an important part of the learning process, which suggests an iterative approach. As an experimental plan progresses some variables are dropped, the region of exploration for some factors changed or new response variables considered.

11.4 Application of the Statistically Designed Experiment to Optimise the HIPS

The following sections describe the application of a statistically designed experiment to optimise the High-Integrity Protection System (HIPS) introduced in chapter 5 of this thesis.

11.4.1 Aim and Response Variable

The means by which performance of a safety system design is assessed (i.e. the response variable) is chosen to be the penalised system unavailability (Q'_{SYS}), introduced in section 5.2. The primary goal is to determine the optimum level for each parameter such that Q'_{SYS} is minimised. The secondary goal, which is addressed throughout the experimental procedure, is to determine the effect (or lack of) that each parameter has on Q_{SYS} , maintenance down time (MDT), cost and the spurious trip frequency (F_{SYS}). The SDE is essentially a global optimisation procedure which initially encompasses a large proportion of the entire search space. Attainment of the global minimum, i.e. the primary goal, with this somewhat crude approach is unexpected. However, it is expected, through attention to the latter goal, to determine promising areas in the search space and hence, a starting point, or points, from which to commence a more sophisticated approach.

11.4.2 Choice of Factors and Levels

It was decided to include all 10 parameters associated with the HIPS (described in section 5.1.1) to perform the initial experiment. The intention was to screen out the less important factors before carrying out a more detailed follow-up experiment. Levels of the factors should be chosen to be sufficiently far apart to cover a wide experimental region. This gives the ability to identify good and bad regions of the search space. During subsequent refinement experiments levels closer to one another can be chosen. Each factor and their alternate levels selected for consideration are listed in table 11.3. Note that parameters K_1 , N_1 , K_2 and N_2 are denoted as $K1$, $N1$, $K2$ and $N2$. This is to avoid confusion with the level of the parameter. For example $N1_1$ denotes level 1 of parameter $N1$.

Factor	Levels		
	1	2	3
<i>E</i>	0	1	2
<i>K1</i>	1	2	
<i>N1</i>	2	3	4
<i>H</i>	1	2	
<i>K2</i>	1	2	
<i>N2</i>	2	3	4
<i>V</i>	1	2	
<i>P</i>	1	2	
$\theta 1$	30	40	50
$\theta 2$	30	40	50

Table 11.3 Factor Levels Chosen for the Initial Experiment

Note also that the range of parameters *K1*, *N1*, *K2* and *N2* are restricted to ensure that each design specified in the matrix experiment is feasible. In order to preserve the benefits of using an orthogonal array it is important that all designs in the matrix are investigated. If designs corresponding to one or more rows are not analysed the balancing property and hence, the orthogonality is lost.

11.4.3 Matrix Experiment

An efficient way to study the effects of several factors simultaneously is to plan a matrix experiment using an orthogonal array. An orthogonal array is constructed from the knowledge of the number of factors, their levels and the desire to study interactions. In this initial experiment there are 5 2-level factors and 5 3-level factors. For the purpose of the screening experiment it was decided not to study specific interactions. Using the standard methods of constructing orthogonal arrays, the standard array $L_{27}(3^{13})$ was selected for the matrix experiment. The reader is referred to references 51 and 62 for further detail on array construction and standard arrays.

The $L_{27}(3^{13})$ orthogonal array is given in Appendix VII. It has 13 columns and 27 rows. Each row has 3 distinct entries, namely 1, 2 and 3 (i.e. each column is a 3-level column). As such, a 3 level factor can be assigned directly to any column. Five of the chosen factors have 2-levels. The dummy level technique was used to assign each 2-level factor to a 3-level column. The dummy level technique works on the basis that if a factor A has 2 levels A_1 and A_2 , it can be assigned to a 3-level column by creating a dummy level A_3 , which could be taken to be the same as either A_1 or A_2 . The choice between taking $A_3 = A_1$ or $A_3 = A_2$ depends on a number of issues. For a more detailed explanation of this technique refer to reference 6. In columns 1, 2, 4, 6 and 7 of the matrix experiment in table 11.4 level 3 has been replaced by level 1. Looking at these columns it is clear that they do not have the balancing property. However, for level 1 in column 3 there are 6 rows with level 1 in column 1 (and 2, 4, 6 and 7) and 3 rows with level 2 in column 1. Similarly for level 2 in column 3 there are 6 rows with level 1 in column 1 and 3 rows with level 2 in column 1. The same can be said about level 3 in column 1. This is called proportional balancing. It can be shown that proportional balancing is also a sufficient condition for a matrix experiment to be orthogonal. Thus, after applying the dummy level technique the resulting array is still proportionally balanced and hence, orthogonal.

The modified $L_{27}(2^5 3^8)$ array representing the 27 safety system designs to be investigated is illustrated in table 11.4. Parameter V is assigned to column 1, P to column 2 and so on. Empty columns in the matrix (i.e. column 11, 12, and 13) are removed. Note that keeping one or more columns empty does not alter the orthogonality property of the array. The entire matrix of table 11.4 is translated using the level definitions in table 11.3 to create a clear experimental plan, see table 11.5.

Design No.	Column Numbers and Factor Assignment									
	1-V	2-P	3-E	4-K1	5-N1	6-H	7-K2	8-N2	9-θ1	10-θ2
1	1	1	1	1	1	1	1	1	1	1
2	1	1	1	1	2	1	2	2	2	2
3	1	1	1	1	3	2	2	3	3	3
4	1	1	2	2	1	1	1	2	2	2
5	1	1	2	2	2	1	2	3	3	3
6	1	1	2	2	3	2	2	1	1	1
7	1	2	3	2	1	1	1	3	3	3
8	1	2	3	2	2	1	2	1	1	1
9	1	2	3	2	3	2	2	3	2	2
10	1	1	2	2	1	1	2	2	2	3
11	1	1	2	2	2	2	1	3	3	1
12	1	1	2	2	3	1	2	3	1	2
13	1	1	3	1	1	1	2	3	3	1
14	1	1	3	1	2	2	1	3	1	2
15	1	1	3	1	3	1	2	2	2	3
16	1	2	1	2	1	1	2	3	1	2
17	1	2	1	2	2	2	1	2	2	3
18	1	2	1	2	3	1	2	3	3	1
19	2	1	3	2	1	2	2	2	3	2
20	2	1	3	2	2	1	2	3	1	3
21	2	1	3	2	3	1	1	3	2	1
22	2	1	1	2	1	2	2	3	1	3
23	2	1	1	2	2	1	2	3	2	1
24	2	1	1	2	3	1	1	2	3	2
25	2	2	2	1	1	2	2	3	2	1
26	2	2	2	1	2	1	2	2	3	2
27	2	2	2	1	3	1	1	3	1	3

Table 11.4 Modified L_{27} Orthogonal Array for the Initial Matrix Experiment

Design No.	<i>V</i>	<i>P</i>	<i>E</i>	<i>K1</i>	<i>N1</i>	<i>H</i>	<i>K2</i>	<i>N2</i>	θ_1	θ_2
1	1	1	0	1	2	1	1	2	30	30
2	1	1	0	1	3	1	2	3	40	40
3	1	1	0	1	4	2	2	4	50	50
4	1	1	1	2	2	1	1	3	40	40
5	1	1	1	2	3	1	2	4	50	50
6	1	1	1	2	4	2	2	2	30	30
7	1	2	2	2	2	1	1	4	50	50
8	1	2	2	2	3	1	2	2	30	30
9	1	2	2	2	4	2	2	3	40	40
10	1	1	1	2	2	1	2	2	40	50
11	1	1	1	2	3	2	1	3	50	30
12	1	1	1	2	4	1	2	4	30	40
13	1	1	2	1	2	1	2	3	50	30
14	1	1	2	1	3	2	1	4	30	40
15	1	1	2	1	4	1	2	2	40	50
16	1	2	0	2	2	1	2	4	30	40
17	1	2	0	2	3	2	1	2	40	50
18	1	2	0	2	4	1	2	3	50	30
19	2	1	2	2	2	2	2	2	50	40
20	2	1	2	2	3	1	2	3	30	50
21	2	1	2	2	4	1	1	4	40	30
22	2	1	0	2	2	2	2	3	30	50
23	2	1	0	2	3	1	2	4	40	30
24	2	1	0	2	4	1	1	2	50	40
25	2	2	1	1	2	2	2	3	40	30
26	2	2	1	1	3	1	2	2	50	40
27	2	2	1	1	4	1	1	4	30	50

Table 11.5 Experimental Plan with Specified Factor Values

Each design, dictated by the matrix experiment, was analysed using the analysis method described in chapter 5.

Design No.	Cost	MDT	F_{SYS}	Q_{SYS}	Q'_{SYS}
1	632	110.9	0.803	1.4e-3	1.4e-3
2	672	85.8	0.673	2.51e-3	2.51e-3
3	982	91.5	0.813	2.07e-3	2.07e-3
4	922	110.5	0.681	2.98e-3	2.98e-3
5	962	90.5	0.29	3.59e-3	3.59e-3
6	1212	183.7	0.56	8.6e-4	7.46e-3
7	1152	116.5	2.45	8.28e-3	0.1104
8	1142	190.7	0.555	2.25e-3	9.67e-3
9	1432	174.2	1.53	1.25e-3	0.109
10	902	102.7	0.289	4.2e-3	4.2e-3
11	1212	142.1	0.690	1.12e-3	3.11e-3
12	982	140.8	0.552	1.74e-3	2.97e-3
13	1172	126.2	0.298	2.91e-3	2.91e-3
14	1502	199.8	1.22	8.98e-4	0.0217
15	1212	131.3	0.828	3.25e-3	5.19e-3
16	612	111.4	1.183	4.19e-3	0.0442
17	872	104.8	1.515	1.73e-3	0.1017
18	622	97.1	1.51	2.45e-3	0.1024
19	1242	117.3	0.638	4.1e-3	4.36e-3
20	1062	142.2	0.547	2.29e-3	4.32e-3
21	1102	130.4	1.33	2.13e-3	0.0573
22	822	105	0.455	1.6e-3	1.6e-3
23	642	90.1	0.364	2.31e-3	2.31e-3
24	622	68.6	0.886	3.87e-3	3.87e-3
25	1002	143.9	1.96	9.35e-4	0.1025
26	772	88.9	1.56	5.36e-3	0.1054
27	792	129	3.66	2.62e-3	0.1026
Averages	972.4	123.2	1.03	0.003	0.0342

Table 11.6 Fitness Data for each Design, plus Averages

Table 11.6 specifies the system unavailability (Q_{SYS}), MDT, cost, spurious trip frequency (F_{SYS}) and penalised system unavailability (Q'_{SYS}) for each design.

11.4.4 Data Analysis

Data analysis is concerned with estimating the effect of each parameter on the penalised system unavailability over the 27 designs investigated. The overall mean value of Q'_{SYS} for the experimental region defined by the factor levels in table 11.6 is 0.0342 and is denoted by m . The effect of a factor is defined to be the change in the response produced by a change in the level of the factor. This is commonly termed the *main effect*. For example, the main effect of the valve type parameter at level 1 (V_1) is given by

$$\begin{aligned}
 m_{V_1} &= \frac{1}{18} \sum_{i=1}^{18} Q'_{i\,SYS} \\
 &= 0.0299
 \end{aligned}
 \tag{11.1}$$

where m_{V_1} denotes the main effect of parameter V at level 1. Similarly, the main effect of the valve type parameter at level 2 (V_2) is given by

$$\begin{aligned}
 m_{V_2} &= \frac{1}{9} \sum_{i=1}^9 Q'_{i\,SYS} \\
 &= 0.0427
 \end{aligned}
 \tag{11.2}$$

Note that the parameter governing the number of ESD valves (E) assumes each of its possible values six times over designs 1 to 18. Similarly, for these experiments, the other parameters exhibit this balancing property. As the matrix experiment is based on an orthogonal array, all the averages are balanced. The process of estimating factor effects is sometimes called the Analysis of Means (ANOM).

The main effect of each factor at each of its levels is specified in table 11.7. These averages are shown graphically in figure 11.1 in what is termed an effects plot. The dotted line in this figure defines the overall mean (m).

Factor	Main Effect (ANOM)			Sum of Squares (ANOVA)	% ANOVA
	Level				
	1	2	3		
<i>V</i>	0.0299	0.0427		9.8e-4	2.1
<i>P</i>	4.43e-3	0.0876		0.0386	83.1
<i>E</i>	0.0291	0.0372	0.0361	3.5e-4	0.75
<i>K1</i>	0.0385	0.032		2.5e-4	0.5
<i>N1</i>	0.0305	0.0282	0.0437	1.3e-3	2.7
<i>H</i>	0.0316	0.0394		3.6e-4	0.8
<i>K2</i>	0.045	0.0287		1.6e-3	3.4
<i>N2</i>	0.027	0.0369	0.0385	7.0e-4	1.5
$\theta 1$	0.0218	0.0431	0.0376	2.2e-3	4.8
$\theta 2$	0.0321	0.0331	0.0373	1.4e-4	0.3

Table 11.7 Main Effects and Variance of each Factor

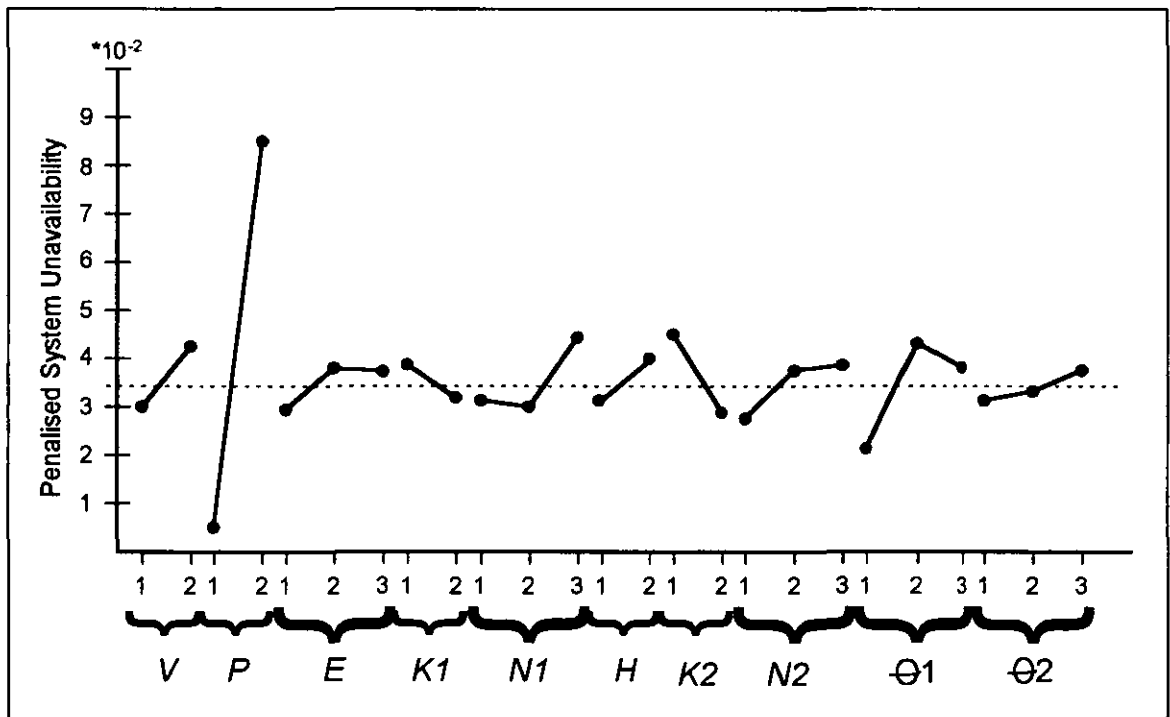


Figure 11.1 A Plot of each Factor's Main Effects

A better feel for the relative effect of the different factors can be obtained using the Analysis of Variance (ANOVA). The sum of squares due to factor *V* is equal to the

total squared deviation of each level of V from the overall mean. 18 designs are investigated with V at level 1 and 9 with V at level 2. Thus,

$$\begin{aligned} ss_V &= 18(m_{V_1} - m)^2 + 9(m_{V_2} - m)^2 \\ &= 9.8 \times 10^{-4} \end{aligned} \quad (11.3)$$

where ss_V is the sum of squares due to factor V . The sum of squares due to each factor is specified in table 11.7. As can be seen the parameter governing the pressure transmitter type (P) explains a major portion of the total variation of Q'_{STS} . In fact it is responsible for 83.1% of the variation. This is reinforced in figure 11.1.

11.4.5 Selecting Optimum Factor Levels

A primary goal in conducting a matrix experiment is to optimise the product design. That is, to determine the best or optimum level for each factor. As regards the HIPS, the optimum level for each factor is the level that gives the lowest value of Q'_{STS} in the experimental region. The estimated main effects can be used for this purpose. It can be seen from figure 11.1 that the settings $V_1, P_1, E_1, K1_2, N1_2, H_1, K2_2, N2_1, \theta1_1, \theta2_1$ would give the lowest penalised system unavailability.

The predicted best settings need not correspond to one of the rows in the matrix experiment, as is the case here. Confirmation of the actual performance characteristics associated with the estimated optimal design is given in table 11.8.

The value of Q'_{STS} realised for the predicted best settings is not as good as the best among the rows of the matrix experiment. This implies additional analysis and experimentation is needed.

Subsystem 1			Subsystem 2			<i>V</i>	<i>P</i>
<i>E</i>	<i>K1/N1</i>	θ	<i>H</i>	<i>K2/N2</i>	θ		
0	2/3	30	1	2/2	30	1	1

MDT	112.7
Cost	652
Spurious Trip Frequency	0.28
System Unavailability	0.0016
Penalised System Unavailability	0.0016

Table 11.8 Characteristics of the Estimated Optimal Design

11.4.6 The Influence of Parameter *P*

The main effects of the factors are their separate effects. If the effect of a factor depends on the level of another, then the two factors are said to have an interaction. The large effect of parameter *P* dominates the effect of the other parameters. In particular the number of pressure transmitters and the number to trip for each subsystem are likely to be effected by the pressure transmitter type.

Figure 11.1 clearly portrays that pressure transmitter type 1 is the optimum level for this factor. As such, the main effect of *K1*, *N1*, *K2* and *N2* was revised using only a subset of the designs in the L_{27} array. That is, those designs with *P* set at level 1. The main effects for each of these variables for *P* at level 1 are specified in table 11.9.

Factor	Main Effect for <i>P</i> at level 1		
	Level 1	Level 2	Level 3
<i>K1</i>	5.96e-3	8.17e-3	
<i>N1</i>	2.91e-3	6.26e-3	0.0131
<i>K2</i>	0.0151	3.62e-3	
<i>N2</i>	4.41e-3	2.9e-3	0.0149

Table 11.9 Main Effects with *P* at Level 1

The main effects at P_1 are much lower than those at P_2 . Figure 10.2 portrays the former effects.

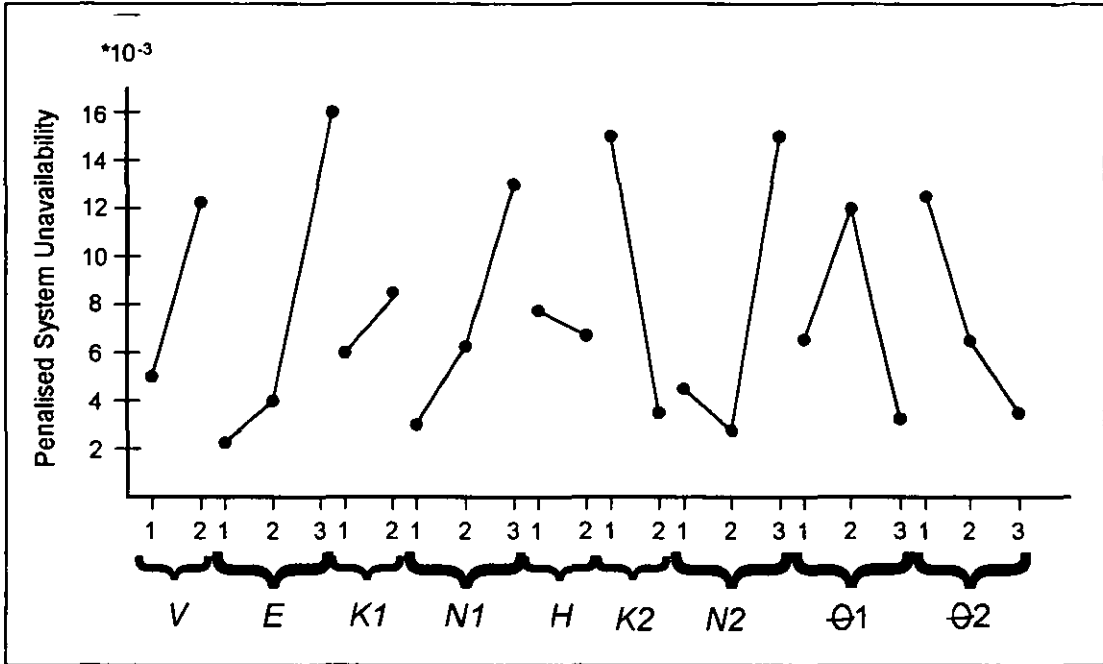


Figure 11.2 A Plot of each Factor's Main Effects with P at level 1

It can be seen that the optimal settings for $K1$, $N1$, $K2$ and $N2$ at P_1 are levels 1, 1, 2 and 2 respectively, i.e. a 1 from 2 trip configuration for subsystem 1 and a 2 from 3 trip configuration for subsystem 2. A confirmation run of the revised prediction specifying the best design gives:

- MDT = 112.7 hours
- Cost = 652 units
- Spurious trip frequency = 0.54 per year
- Probability of system unavailability = 0.0014 (no penalties incurred)

This is an improvement over the initial prediction and equals the best in the original experiments table.

11.4.7 Consideration of the Constraints

To carry out a follow-up experiment it must be determined which factors besides P have a strong effect on the response. It was noted that the main effects of each

parameter were biased by significantly large values of Q'_{SYS} , which resulted from the exertion of penalties due to constraint violations. As such, the effect of each parameter on system cost, MDT and spurious trip frequency was established.

In a similar manner to that described in section 11.4.4, the main effect of each factor level with respect to system cost was established. This data is given in table 11.10. In addition, table 11.10 specifies the sum of squares due to each factor and percentage of the variation for which the factor is responsible.

Factor	Main Effect with respect to Cost			ANOVA	% ANOVA
	Level 1	Level 2	Level 3		
<i>V</i>	961.3	994.5		6613	0.4
<i>P</i>	933.1	992		20815	1.1
<i>E</i>	719.8	973.1	1224.2	1144985	62.9
<i>K1</i>	970.9	923.4		43150	2.4
<i>N1</i>	939.8	982	995.3	15114	0.8
<i>H</i>	887.5	1142		588	0.03
<i>K2</i>	978.7	969.2		582627	32
<i>N2</i>	956.4	967.6	993.1	6367	0.3
<i>θ1</i>	973.1	973.1	970.9	29	0.002
<i>θ2</i>	970.9	973.1	973.1	29	0.002

Table 11.10 Main Effects and Variance of each Factor with respect to Cost

The main effect, sum of squares and percentage of the variation were also established with respect to system MDT and spurious trip frequency. This data is given in tables 11.11 and 11.12 respectively.

Factor	Main Effect with respect to MDT			ANOVA	% ANOVA
	Level 1	Level 2	Level 3		
<i>V</i>	128.3	112.9		1432.2	5.2
<i>P</i>	123.6	249.9		373.5	1.3
<i>E</i>	96.14	125.8	147.7	12052.3	43.7
<i>K1</i>	123	123.3		0.4	0.001
<i>N1</i>	116	126.2	127.4	707.1	2.6
<i>H</i>	114.7	140.2		3919.9	14.2
<i>K2</i>	123.6	123		2.2	0.008
<i>N2</i>	122.1	123.7	123.9	17	0.06
$\theta 1$	146	119.3	104.2	8064.4	29.2
$\theta 2$	123.9	121.9	112.7	1011.9	3.7

Table 11.11 Main Effects and Variance of each Factor with respect to MDT

Factor	Main Effect with respect to F_{SYS}			ANOVA	% ANOVA
	Level 1	Level 2	Level 3		
<i>V</i>	0.91	1.23		0.62	4.5
<i>P</i>	0.66	1.77		7.4	53.3
<i>E</i>	0.91	1.14	1.04	0.24	1.7
<i>K1</i>	1.31	0.89		1.06	7.6
<i>N1</i>	0.97	0.82	1.3	1.07	7.7
<i>H</i>	1.04	1.02		2.7e-3	0.02
<i>K2</i>	1.5	0.81		2.6	18.7
<i>N2</i>	0.85	1.11	1.13	0.44	3.2
$\theta 1$	1.06	1.02	1.01	0.13	0.09
$\theta 2$	0.9	0.99	1.2	0.43	3.1

Table 11.12 Main Effects and Variance of each Factor with respect to F_{SYS}

The main effects with respect to cost, MDT and spurious trip frequency are shown graphically in figures 11.3, 11.4 and 11.5 respectively. The dotted lines in figure 11.5 illustrate the main effects for each parameter for those designs with *P* at level 1 only.

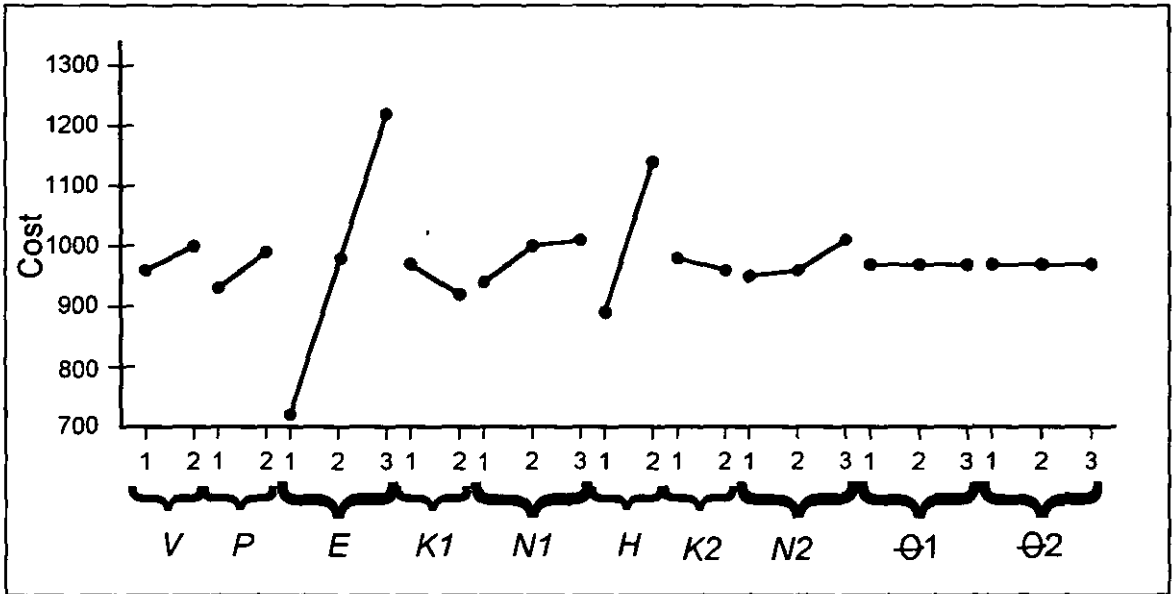


Figure 11.3 A Plot of the Factor's Main Effects Regarding Cost

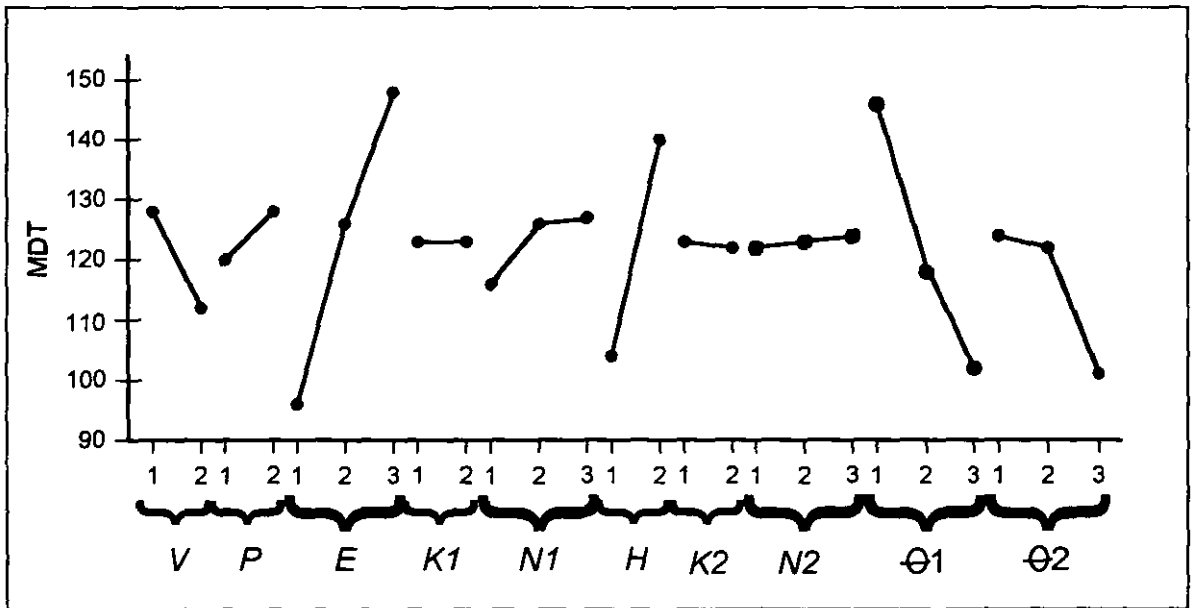


Figure 11.4 A Plot of each Factor's Main Effects regarding MDT

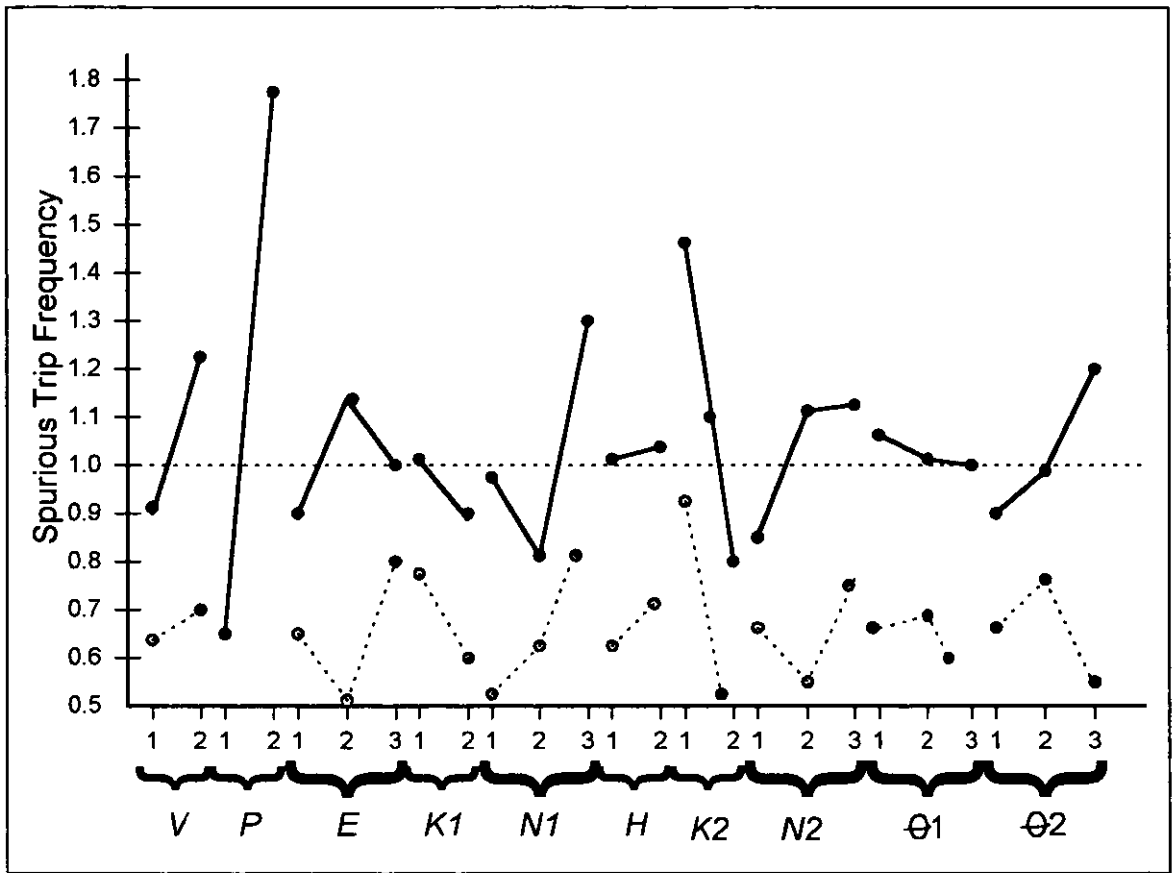


Figure 11.5 A Plot of the Factor's Main Effects regarding F_{sys}

Factors E and H explain a major portion of the total variation of cost, 63% and 32% respectively. Additionally, E and H account for a large portion of the total variation of MDT. $\theta 1$ and to a lesser extent V also contribute to the variation of MDT.

The major parameters which contribute to the variation in spurious trip frequency are those associated with the pressure transmitters. However, the parameter governing the pressure transmitter type accounts for over 50% of the variation alone. Typically, a spurious trip violation is heavily penalised. A design that constitutes pressure transmitters of type 2 generally has a high spurious trip frequency and hence, receives a substantial penalty. This explains the dominance of parameter P regarding variation of the penalised system unavailability. As can be seen in figure 11.5, significant variation in the trip frequency still exists in many of the parameters when only those designs in the matrix with pressure transmitter type 1 are considered. However, this

variation occurs below the spurious trip limit (i.e. 1.0) represented by the dotted line in the figure. As such, it is estimated that for P_1 the spurious trip values do not incur any penalty and therefore, do not affect the system unavailability.

Due to consideration given to the effect of the parameters on the constraints of the safety system design it was decided to carry out a follow-up experiment addressing parameters E , H , V , θ_1 and θ_2 specifically. (The calculated main effect of θ_2 on MDT was not significantly large, however, common sense dictates that there may be an error in the numerical information).

11.4.8 Follow-up Experiment

A follow-up experiment involving fewer factors can be used to explore the response-factor relationship in more detail. The purpose of this step is to verify that the optimum conditions suggested by the initial matrix experiment are indeed accurate and determine whether improvements can be made. It is possible that the verification experiment may identify strong interactions between the parameters and highlight potentially misleading information. The factors selected for the follow-up experiment and their alternate levels are specified in table 11.13.

Factors	Levels			
	1	2	3	4
V	1	2		
E	1	2		
H	1	2		
θ_1	25	30	35	40
θ_2	25	30	35	40

11.13 Factor Levels in the Follow-up Experiment

Observations from the screening experiment implied that the inclusion of 2 ESD valves is not beneficial and indicated the need to study the test intervals in more detail. Note that a trip configuration of 1 from 2 and 2 from 3 pressure transmitters

were included for subsystem 1 and 2 respectively. These parameter values are the best settings predicted in the screening experiment.

11.4.8.1 Design of a Matrix Experiment

In this study it was decided not only to estimate the main effects of the 5 factors listed in table 10.13 but also to estimate 4 key interactions. The four interactions considered to be most important were:

- 1) Valve type and ESD valve number, $V \times E$.
- 2) Valve Type and HIPS valve number, $V \times H$.
- 3) ESD valve number and HIPS valve number, $E \times H$.
- 4) Test interval 1 and test interval 2, $\theta_1 \times \theta_2$.

Construction of the Orthogonal Array

An appropriate orthogonal array must be constructed for the stated factors, their alternate levels and the specific interactions. We are dealing with a more complicated combinatoric problem than that of the screening experiment and as a result the construction of the orthogonal array is more complex. To ensure that the array fits the study it is necessary to count the total degrees of freedom. This determines the minimum number of experiments that must be performed to analyse all the chosen elements. In general, the number of degrees of freedom associated with each factor is equal to one less than the number of levels for that factor. The degrees of freedom for the follow-up experiment was calculated as follows

Source	Degrees of Freedom
Two 4-level factors: θ_1 and θ_2	$2 \times (4 - 1) = 6$
Three 2-level factors: V , E and H	$3 \times (2 - 1) = 3$
Three 2-factor interactions between 2-level columns	$3 \times (2 - 1)(2 - 1) = 3$
One 2-factor interaction between 4-level columns	$(4 - 1)(4 - 1) = 9$
Total	21

Since there are 2-level and 4-level factors it was preferable to use one of the 2-level standard arrays. The array must have 21 or more rows in account of the 21 degrees of freedom. The next smallest 2-level array is L_{32} (Ref. 63).

The interaction table given for the L_{32} array (Ref. 63) was used to establish columns for each interaction. Factors V , E and H were, thus, assigned to columns 1, 2 and 4 respectively. Two 4-level columns were required to accommodate θ_1 and θ_2 . This was achieved using the column merging method. Basically, to create a 4-level column in a standard 2-level orthogonal array, any two columns and their interaction column are merged (the reader is referred to reference 6 for a more detailed discussion of this approach). Columns 7 and 8 were arbitrarily chosen to form a 4-level column for θ_1 . Column 15, which represents the interaction between columns 7 and 8, was kept empty. Next, columns 9 and 16 were arbitrarily chosen to accommodate θ_2 . The interaction column 25 was kept empty.

Column 3 contains interaction between columns 1 and 2, so it can be used to estimate the interaction $V \times E$. Similarly, column 5 contains interaction between columns 1 and 4, thus estimating $V \times H$ and column 6 can estimate the interaction $E \times H$. Finally, column 22 estimates the interaction $\theta_1 \times \theta_2$. All the interaction columns were free from main effects. The assignment of factors to the columns of the L_{32} array is thus:

Factor	Column	Factor	Column
V	1	$V \times E$	3
E	2	$V \times H$	5
H	4	$E \times H$	6
θ_1	7,8,15	$\theta_1 \times \theta_2$	22
θ_2	9,16,25		

The final matrix experiment shown in table 11.14 (stating only the columns corresponding to the main effects) is an orthogonal array. In any pair of columns all components occur and they occur an equal number of times. The values of the levels of each factor are specified in the table as opposed to using their level code (namely 1

and 2 for a 2-level factor and so on). This gives a clearer indication of the design characteristics being investigated in each run.

Design No.	V	E	H	θ_1	θ_2
1	1	0	1	25	25
2	1	0	1	25	30
3	1	0	1	30	35
4	1	0	1	30	40
5	1	0	2	35	25
6	1	0	2	35	30
7	1	0	2	40	35
8	1	0	2	40	40
9	1	1	1	35	25
10	1	1	1	35	30
11	1	1	1	40	35
12	1	1	1	40	40
13	1	1	2	25	25
14	1	1	2	25	30
15	1	1	2	30	35
16	1	1	2	30	40
17	2	0	1	35	35
18	2	0	1	35	40
19	2	0	1	40	25
20	2	0	1	40	30
21	2	0	2	25	35
22	2	0	2	25	40
23	2	0	2	30	25
24	2	0	2	30	30
25	2	1	1	25	35
26	2	1	1	25	40

Table 11.14 Matrix Experiment

Cost	MDT	F_{SYS}	Q_{SYS}	Q'_{SYS}
652	135.2	0.542	9.7e-3	1.56e-3
652	126.2	0.542	1.15e-3	1.16e-3
652	106.2	0.542	1.63e-3	1.63e-3
652	101.4	0.542	1.85e-3	1.85e-3
922	153.6	0.551	7e-4	3.4e-3
922	137.8	0.551	8.42e-3	1.72e-3
922	119	0.551	1.13e-3	1.13e-3
922	110.5	0.551	1.3e-3	1.3e-3
922	141.7	0.551	1.29e-3	2.63e-3
922	132.7	0.551	1.53e-3	1.85e-3
922	115.3	0.551	2.03e-3	2.04e-3
922	110.5	0.551	2.32e-3	2.32e-3
1192	218.4	0.56	4.67e-4	1.08e-2
1192	202.4	0.56	5.62e-4	9.08e-3
1192	170.6	0.56	7.87e-4	5.76e-3
1192	162.1	0.56	9.02e-4	4.95e-3
602	89.1	0.625	2.33e-3	2.33e-3
602	85.2	0.625	2.6e-3	2.6e-3
602	94.4	0.625	1.94e-3	1.94e-3
602	87.1	0.625	2.31e-3	2.31e-3
822	134.6	0.72	7.15e-4	1.24e-3
822	127.9	0.72	8.22e-4	8.23e-4
822	142.5	0.72	6.1e-4	2.03e-3
822	130	0.72	7.36e-4	7.36e-4
822	143.5	0.72	1.56e-3	3.4e-3
822	139.6	0.72	1.77e-3	2.87e-3

Table 11.15 Fitness Data

27	2	1	1	30	25
28	2	1	1	30	30
29	2	1	2	35	35
30	2	1	2	35	40
31	2	1	2	40	25
32	2	1	2	40	30

Table 11.14 Contd.

822	137.3	0.72	1.34e-3	2.18e-3
822	130	0.72	1.61e-3	1.61e-3
1042	133.7	0.81	9.48e-4	1.38e-3
1042	127	0.81	1.09e-3	1.10e-3
1042	145.1	0.81	7.64e-4	2.5e-3
1042	132.6	0.81	7.22e-4	1.23e-3
855.1	132	0.63	1.26e-3	2.6e-3

Table 11.15 Contd.

Table 11.15 gives the cost, MDT, spurious trip frequency, system unavailability and penalised system unavailability of each design in the matrix experiment. The average values of each are specified in bold in the last row. The best design, also stated in bold in row 24 of table 11.14 and 11.15, has a system unavailability of 7.36×10^{-4} .

11.4.8.2 Data Analysis

The data in table 11.15 was analysed by the standard procedures introduced in section 11.4.4 to determine the main effect of each factor with respect to the penalised system unavailability. This data is given in table 11.16 and the effects plotted in figure 11.6.

Factor	Level			
	1	2	3	4
<i>V</i>	3.32e-3	1.46e-3		
<i>E</i>	1.93e-3	3.27e-3		
<i>H</i>	1.97e-3	3.22e-3		
θ_1	3.81e-3	2.59e-3	2.12e-3	1.87e-3
θ_2	3.37e-3	2.46e-3	2.32e-3	2.23e-3

Table 11.16 Main Effects of Factors

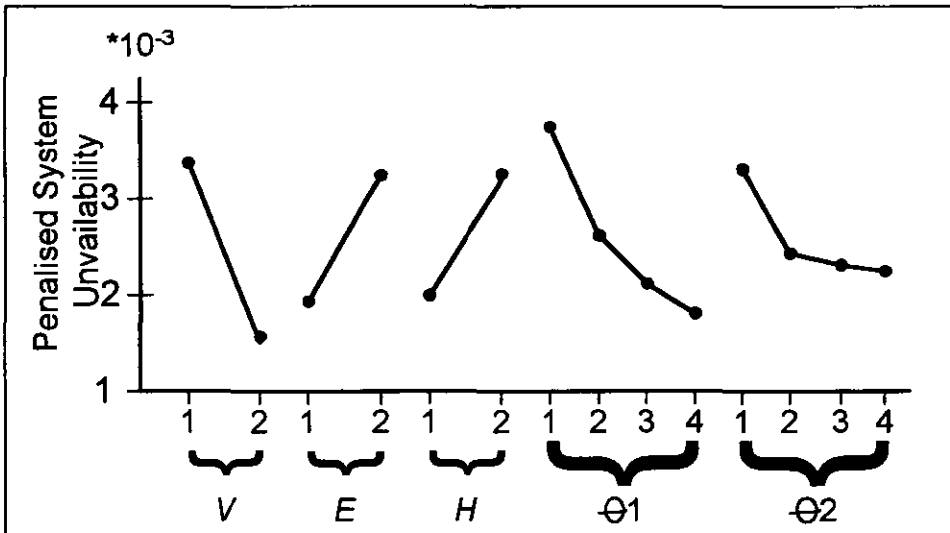


Figure 11.6 Main Effects of Factors in the Follow-up Experiment

It is estimated from the main effects that the settings V_2 , E_1 , H_1 , θ_{14} and θ_{24} would give the lowest penalised system unavailability. A confirmation run gives

- MDT = 78 hours
- Cost = 602 units
- Spurious trip frequency = 0.625 per year
- Probability of system unavailability = 0.0031 (no penalties incurred)

This design is comparatively poor and hence, consideration is given to the key interactions.

Table 11.17 represents the data from the interactions $V \times E$ and $V \times H$. Specifically, the main effect of factor E is calculated at the first level of factor V , then the second level of factor V . $V \times H$ is considered in a similar manner. Figure 11.7 illustrates these values graphically by plotting the response data in table 11.17 against factors E and H for both levels of factor V .

		<i>E</i>		<i>H</i>	
		1	2	1	2
V	1	3.65e-3	7.21e-3	1.8e-3	4.7e-3
	2	1.75e-3	1.99e-3	1.3e-3	2.4e-3

Table 11.17 $V \times E$ and $V \times H$ Interaction Effects

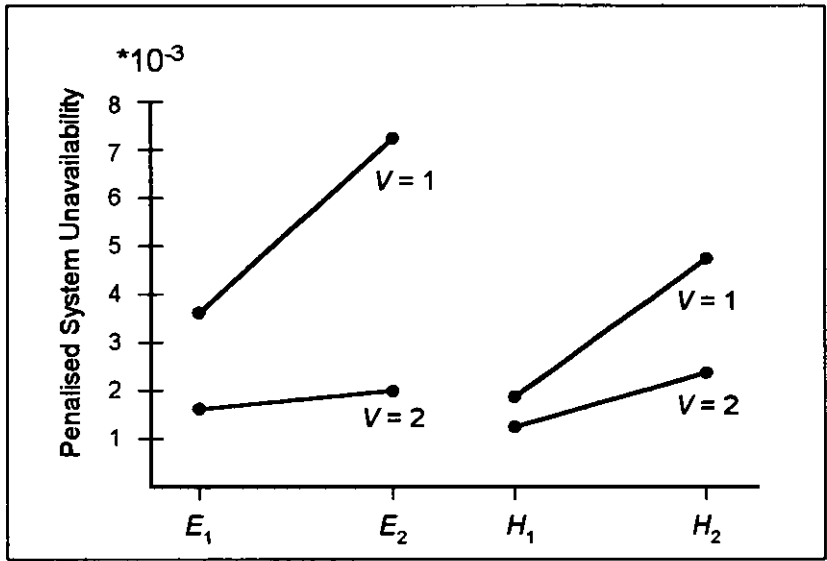


Figure 11.7 Effects Plot for the $V \times H$ and $V \times E$ Interactions

Effects were also calculated for the interactions $E \times H$ and $\theta 1 \times \theta 2$. These are given in tables 11.18 and 11.19 respectively. An effects plot of the these interactions is illustrated in figure 11.8.

		<i>E</i>	
		1	2
<i>H</i>	1	1.92e-3	2.32e-3
	2	1.54e-3	4.59e-3

11.18 $E \times H$ Interaction

Data

		$\theta 2$			
		1	2	3	4
$\theta 1$	1	6.16e-3	5.12e-3	1.84e-3	2.16e-3
	2	2.1e-3	1.17e-3	3.7e-3	3.4e-3
	3	3.01e-3	1.78e-3	1.86e-3	1.85e-3
	4	2.22e-3	1.77e-3	1.58e-3	1.81e-3

11.19 $\theta 1 \times \theta 2$ Interaction Data

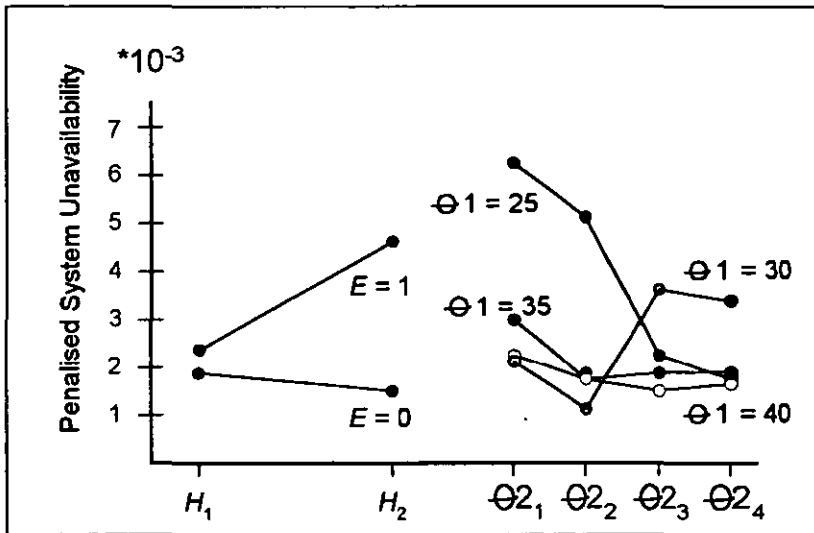


Figure 11.8 Effects Plot for the $E \times H$ and $\theta_1 \times \theta_2$ Interactions

11.4.8.3 Interpretation of Results

If there is no interaction between 2 factors an effects plot of one factor against both levels of the other will portray two parallel lines. No interaction implies that the change in the level of one factor will produce the same change in the response regardless of the level of the other factor. In this case, the optimum levels identified by the main effects are valid. If the lines on the effects plot are not parallel but have the same direction of movement then there is a presence of interaction. However, the optimum levels identified by the main effects are likely to still be valid. This is termed a synergistic interaction. However, an inconsistent direction of movement on the effects plot implies a strong interaction between the factors. In such cases the optimum level identified by the main effects can be misleading. This is termed an anti-synergistic interaction.

It is apparent from the main effect and interaction plots that:

- There is a synergistic interaction between parameters V and E and V and H . The interaction is stronger in the former. Data in table 11.17 verifies the main effects data (see figures 11.6 and 11.7), which states that the best settings for parameters

V , E and H are 2, 1 and 1 respectively (i.e. valve type 2 with 0 ESD valves and 1 HIPS valve).

- There is an anti-synergistic interaction between parameters E and H . This makes sense as both incur significant cost to the design. Data in table 11.18 contradicts that of the main effects data. It can be seen from figure 11.8 that when the number of ESD valves is 0 (its optimal setting) the best level for factor H is 2 (i.e. 2 HIPS valves fitted).
- Strong interaction exists between the maintenance test interval parameters and as such the main effects associated with θ_1 and θ_2 may be misleading. The implication from figure 11.8 is that the higher maintenance test interval values are generally beneficial. This could be due to the fact that such values do not incur a penalty as a result of a MDT violation. It is most likely that the fittest designs are associated with the lower test values however, such results are obscured by penalties due to MDT violation. Despite this, the best settings for both test parameters were predicted to be level 2 (i.e. 30 week intervals for each).

11.4.8.4 Optimum Factor Settings

The best settings of the factors suggested by the above results are identical to that of the best design in the design matrix of the follow-up experiment (note that the best settings of factors P , $K1$, $N1$, $K2$ and $N2$ are carried over from the initial experiment). As such, a verification experiment is not required to establish the response associated with the predicted best design. The performance characteristics of the resulting best design are summarised in table 11.20. The safety system is over 99.9% available and achieves full use of the MDT resource.

Subsystem 1			Subsystem 2			<i>V</i>	<i>P</i>
<i>E</i>	<i>K1/N1</i>	θ	<i>H</i>	<i>K2/N2</i>	θ		
0	1/2	30	2	2/3	30	2	1

MDT	130
Cost	822
Spurious Trip Frequency	0.72
System Unavailability	7.36e-4
Penalised System Unavailability	7.36e-4

Table 11.20 Characteristics of the Estimated Optimal Design

11.4.9 Discussion of Results

Data analysis of the initial experiment clearly dictated that the optimal safety system design should constitute pressure transmitters of type 1. It was further implied that the optimal trip configuration using this pressure transmitter is 1 from 2 for subsystem 1 and 2 from 3 for subsystem 2. It was noted that penalised designs were causing a bias in the main effects and as such, further analysis was carried out to determine which parameters had the greatest effect on constraint violation. The implication was parameters *E*, *H*, *V* and θ_1 are the most influential. Logic dictates that the parameter θ_2 is also influential as regards system MDT.

In light of this evidence a more detailed follow-up experiment was implemented using the factors stated above. A study of potential key interactions was also incorporated. It was found that parameters *V* and *E* and *V* and *H* portrayed a synergistic interaction. Whereas, *E* and *H* and both test parameters portrayed an anti-synergistic interaction. The interaction plots indicated that the optimal safety system design incorporates 0 ESD valves and 2 HIPS valves of type 2 with both test intervals set at 30 week intervals.

The optimum level predicted for each factor corresponded with the best design in the follow-up matrix experiment. The system unavailability of the design is 7.36×10^{-4} with no constraint violations.

11.5 Application of Local Optimisation Techniques

Common sense dictates that $K1$ and $N1$, in addition to $K2$ and $N2$ will portray significant interaction. A matrix experiment could be developed in a similar vein to that described in section 11.4.7 to investigate these interactions further. In the main, however, the experimental approach is a global technique that considers the entire search space to glean greater knowledge in order to determine potentially good (and bad) areas of the space. This knowledge can then be used to assist further, more sophisticated techniques.

The grid-sampling technique and logical search approach, introduced in chapters 8 and 10 respectively, are in contrast, localised, hill-climbing methods that rely on a good start point to achieve an optimal solution. As oppose to further refinement of the resulting design specified in table 11.20, denoted by \mathbf{x}^R , using SDE's, the design vector is used as an initial design point in the aforementioned localised approaches.

Results arising from application of the grid-sampling approach are specified in table 11.21 and 11.22. As can be seen, the first design predicted to be an improvement over \mathbf{x}^R is inaccurate in its approximation of the spurious trip frequency. The boundaries of the search space about the initial design vector are, therefore, reduced, a further design predicted and subsequently verified. Again, the predicted design's spurious trip frequency is inaccurate. This process continues until the boundaries are maximally reduced and hence, a safety system design more optimal then the initial design vector, \mathbf{x}^R , is not achieved. An explanation for this could be that \mathbf{x}^R is exceedingly fit. As such, the errors introduced via use of approximate objective functions to analyse the search space about the initial design point are large in comparison to the difference between the initial design vector and the global minimum.

Predicted Design No.	Accepted Design No.	E	K_1/N_1	H	K_2/N_2	P	V	θ_1	θ_2	Actual Q_{SYS}
Initial Design		0	1/2	2	2/3	1	2	30	30	7.36e-4
1	-	0	1/2	2	1/4	1	2	30	31	-
2	-	0	1/1	2	1/4	1	2	30	30	-
3	-	0	1/2	2	1/4	1	2	31	31	-
4	-	0	1/2	2	1/4	1	2	32	31	-

Table 11.21 Grid-sampling approach. Design Characteristics Using $\mathbf{x}^{(0)} = \mathbf{x}^R$

Predicted Design No.	Accepted Design No.	Predicted Q_{SYS}	Predicted F_{SYS}	Actual Q_{SYS}	Actual F_{SYS}	MDT	Cost
Initial Design		-	-	7.36e-4	0.72	130	822
1	-	5.08e-4	0.913	-	1.24	129.7	842
2	-	5.33e-4	0.782	-	1.11	130	822
3	-	5.34e-4	0.913	-	1.24	127.5	842
4	-	5.34e-4	0.913	-	1.24	125.4	842

Table 11.22 Grid-sampling Approach: Fitness Values of Each Predicted Design

The final design resulting from application of the search logic approach with variable ordering $\{N_2, K_2, H, E, N_1, K_1, H, K_2, N_2, E, K_1, N_1\}$ using $\mathbf{x}^{(0)} = \mathbf{x}^R$ is given in table 11.23. It can be seen that the resulting design has a 1 from 2 as opposed to a 2 from 3 pressure transmitter trip configuration constituting subsystem 2, in addition to modification of the maintenance test interval parameters. An improvement of 1.3×10^{-5} in system unavailability is achieved resulting in a safety system design which is over 99.92% available. The logical search approach implements an accurate investigation of the region directly surrounding the value of each variable in the initial design vector, whilst considering a degree of interaction between the design variables. The initial point proves, in this case, to be sufficiently fit such that the believed global optimum of the search space is found.

	E	K_1/N_1	H	K_2/N_2	P	V	θ_1	θ_2	Cost	F_{SYS}	MDT	Q_{SYS}
$\mathbf{x}^{(0)}$	0	1/2	2	2/3	1	2	30	30	822	0.72	130	7.36e-4
$\mathbf{x}^{(11)}$	0	1/2	2	1/2	1	2	34	26	802	0.977	129.3	7.23e-4

Table 11.23 The Logical Search Approach Using $\mathbf{x}^{(0)} = \mathbf{x}^R$

11.6 Conclusion and Summary

To summarise, the use of orthogonal arrays to develop an experimental design is an efficient way to study the effects of several control factors simultaneously. It is beneficial to adopt an iterative approach to acquire knowledge as the experimental plan progresses and thus, adapt the process accordingly. In the worst case scenario (i.e. if many factors exhibit strong interactions) good and bad regions of the search space are identified and an indication of the relative importance of each factor acquired. As such, further exploration can proceed using an appropriate local search technique based on the information established. In the best case scenario, the optimum level for some, if not all, the factors is determined. Importantly, the experimental effort required is much smaller when compared to other methods of exploration such as guess and test (trial and error), one factor at a time and full-factorial experiments.

CHAPTER 12

CONCLUSIONS AND FUTURE WORK

12.1 Conclusions

The aim of the thesis is to analyse optimisation techniques in order to produce optimal use of resources in safety system designs, such that the best performance possible, not just adequate performance, is obtained. During the study the following is determined:

- 1) The use of house events in a fault tree structure can be used to represent the failure causes of all potential designs.
- 2) Converting the fault tree structure to a Binary Decision Diagram results in all types of design parameters (component selection, redundancy levels and test intervals) being represented by 'probability' values in the analysis. The BDD also provides an efficient analysis method for all design options.
- 3) The use of a Simple Genetic Algorithm provides a means of optimisation which is capable of coping with all the requirements of the design problem and the approach was successfully tested on a High Pressure Protection System.
- 4) The original SGA highlighted potential areas of improvement. The modification methods have the following features:
 - Each method modifying the exploration technique of the MTI parameter values improves the use of the MDT resources available and thus, overall system performance.
 - Modifying the cost, and hence spurious trip penalty, enables a certain leeway regarding the cost constraint. Good system designs, which cost more than the allotted 1000 units, have a greater chance of being considered in later generations of the GA.
 - Finally, and most importantly, the modified conversion method provides an accurate representation of the fitness values of each system design to the GA.
- 5) The modified GA was successfully tested on a Firewater Deluge System.
- 6) The use of a Weibull distribution to model the relevant components within the modified GA approach enabled consideration of components of wear-out type.

- 7) While GA's have been applied here to two safety systems the procedure is generic and could equally be applied to optimise the performance of any engineering system whose failure causes can be represented by a fault tree.
- 8) The grid-sampling optimisation method is an iterative procedure that utilises the fault tree analysis technique to assess the performance of any given design. If a sufficiently fit initial design vector is used the resulting design makes the best use of available resources rather than achieving just an 'adequate' design.
- 9) While the grid-sampling method has been applied here to a safety system the procedure is generic and could equally be applied to optimise the performance of any engineering system whose failure causes can be represented by a fault tree.
- 10) The search-logic approach was successfully tested on a High Pressure Protection System, proving to be both efficient and effective
- 11) The statistically designed experiment (SDE) was successful in its ability to identify good and bad regions of the search space for the HIPS optimisation.
- 12) The iterative approach of the SDE, adapting the process accordingly as the experimental plan progresses, can be applied to optimise the performance of any engineering system whose failure causes can be represented by a fault tree.
- 13) The GA, grid-sampling and search-logic methods are automated, providing less of a reliance on engineering judgement. However, both the grid-sampling and the search-logic approach rely on identification of a sufficiently fit initial design vector. In addition, the search-logic approach requires identification of an appropriate variable ordering.
- 14) The SDE is adapted as the plan progresses, where each step depends on knowledge gained from previous steps. As such, this approach is not easily automated. However, it is efficient in that all information evaluated is used.
- 15) Constraints on the available resources do not inhibit application of any of the methods. Implicit or explicit constraints of the equality or inequality form can be incorporated.
- 16) Application of the SDE, specifically the development of an orthogonal matrix experiment, is restricted somewhat by infeasible design vectors, i.e. designs such that variables within the parameter set are non-sensical. For example, the number of pressure transmitters is 2 and the number of pressure transmitters required to trip the system is 3.

- 17) Maintenance activity can be incorporated into the optimisation scheme of each approach if required to provide maximum benefit from the procedure or the design can be considered in two stages, as is traditional, with the system structure being formulated first and the maintenance activity specified afterwards. The range of the maintenance test values that can be considered is, however, restricted in the SDE approach.
- 18) In application to the High Pressure Protection System the search-logic approach proved to be most efficient and effective. However, as regards a larger more complex system, such as the FDS, application of the modified GA in conjunction with a local search approach, such as the search-logic method, to the resulting design would be most beneficial.
- 19) The GA is particularly advantageous due to its lack of reliance on engineering judgement. It proceeds from a randomly generated population of points and as such, knowledge of the system is not required to initiate the search. The search continues in an automated fashion, which can be terminated at any point and a result obtained. In contrast, both the search-logic and the grid-sampling approach are local in scope and require good engineering judgement to determine an initial design vector. Similarly, the SDE requires a degree of knowledge to establish the initial matrix experiment and to plan further experiments as the search proceeds.

12.2 Future Work

Future work would involve application of each optimisation approach, i.e. GA's, grid-sampling, search logic and SDE, to a variety of different safety systems. Results from this could lead to the development of alternative optimisation approaches or modifications to the original techniques, therefore improving efficiency and accuracy. In addition, different optimisation strategies could be employed. As opposed to optimising system availability whilst constraining cost, cost could be optimised whilst constraining system availability.

Fault tree analysis has proved itself as a very flexible and useful technique for the assessment of system failure characteristics. Incorporating the use of alternative analysis techniques, such as Markov methods, could however reduce the assumptions necessary when using the fault tree method and, therefore, expand

the applicability of the optimisation techniques to a more diverse range of systems.

REFERENCES

1. P.R. Adby, M.A.H. Dempster, "Introduction to Optimisation Methods", Chapman and Hall, 1974.
2. K.K. Aggarwal, J.S. Gupta, and K.B. Misra, "A New Heuristic Criterion for Solving a Redundancy Optimisation Problem", *IEEE Transactions on Reliability*, Vol. R-24, No. 1, pp 86-87, 1975.
3. S.B. Akers, "Binary Decision Diagrams," *IEEE Trans. Computers*, Vol. C-27, No. 6, June 1978.
4. M.M. Ali, C. Storey, "Modified Controlled Random Search Algorithms", *International Journal of Computers and Mathematics*, Vol. 55, pp229-235, 1994.
5. J.D. Andrews and T.R. Moss, "*Reliability and Risk Assessment*," Longman Scientific and Technical, UK, 1993.
6. T. Back, "*Evolutinary Algorithms in Theory and Practice*", Oxford University Press, Oxford, 1997.
7. J. Baker, "Adaptive selection methods for genetic algorithms", in *Proceedings of the Second International Conference on Genetic Algorithms*, J. Grefenstette (Ed.), Lawrence Erlbaum Associates, Hillsdale, NJ, 1987.
8. L.M. Bartlett, J.D. Andrews, "Efficient Basic Event Orderings for Binary Decision Diagrams", *Proceedings of the Annual Reliability and Maintainability Symposium*, Anaheim, pp61-68, 1998.
9. L. Booker, "Improving search in genetic algorithms", in *Genetic Algorithms and Simulated Annealing*, L. Davies (ed.), Morgan Kaufmann Publishers, 1987, pp61-73.
10. G. Box, N. R. Draper, "Empirical Model-Building and Response Surfaces", John Wiley & Sons, 1987.
11. G. Box, W. Gordon, H. Stuart, "Statistics for Experimenters", John Wiley & Sons, 1987.
12. M.J. Box, "A New Method of Constrained Optimisation and a Comparison with Other Methods", *Computer Journal*, Vol. 8, pp42-52, 1965.
13. M. Bouissou, "An ordering heuristic for building binary decision diagrams from fault trees," *In Proceedings of RAMS'96 Conference*, Las Vegas, Nevada, pp208-214, 22-25 January 1996.
14. R.E. Bryant, "Graph-Based algorithms for Boolean function manipulation," *IEEE Trans. Computers*, Vol. C-35, No. 8, pp677-691, 1986 Aug.

15. P. Chatterjee, "Modularization of fault trees: A method to reduce the cost of analysis," *Reliability and Fault Tree Analysis*, SIAM, pp101-137, 1975.
16. A. Chipperfield, "Introduction to genetic algorithms", in *Genetic Algorithms in Engineering Systems*, A. M. S. Zalzala and P.J. Flemming (Eds.), The Institution of Electrical Engineers, 1997.
17. E.K.P. Chong, S.H. Zak, "An Introduction to Optimization", John Wiley & Sons, Inc, 1996.
18. S. Contini and G. de Cola, "A top-down approach to fault tree analysis using binary decision diagrams," *Jour. European des Systemes Automatises*, Vol. 30, No. 8, pp1115-1130, 1996.
19. O. Coudert and J.C. Madre, "A new method to compute prime and essential prime implicants of Boolean functions," In T. Knight and J. Savage, editors, *Advanced Research in VLSI and Parallel Systems*, pp113-128, March 1992.
20. P.A. Crosetti, "Fault tree analysis with probability evaluation," *IEEE Nuclear Power Systems Symposium*, Nov, pp465-471, 1970.
21. D. Dasgupta, Z. Michalewicz, "Evolutinary algorithms –an overview", in *Evolutionary Algorithms in Engineering Application*, D. Dasgupta and Z. Michalewicz (Eds.), Springer-Verlag Berlin, Heidelberg, 1997.
22. K. A. De Jong, "Analysis of the behaviour of a class of genetic adaptive systems", PhD thesis, Dept. of Computer and Communication Sciences, University of Michigan, Ann Arbour, 1975.
23. K. De Jong, "Genetic Algorithms: a 10 year perspective", *Proceedings of an International Conference on Gas and their Applications.*, 1985.
24. B.S. Dhillon, "Bibliography of literature of fault trees", *Microelectronics and Reliability*, Vol. 17, pp501-503, 1978.
25. A.K. Dhingra, "Optimal Apportionment of Reliability and Redundancy in Series Systems Under Multiple Objectives", *IEEE Transactions on Reliability*, Vol. 41, No. 4, pp577-582, 1992.
26. L.C.W. Dixon, "Nonlinear Optimisation", The English Universities Press Ltd, 1972.
27. L.C.W. Dixon, G.P. Szego, "Deterministic Vs Probabilistic", *Toward Global Optimisation 2*, North-Holland Publishing Company, 1978.
28. C.E. Ebeling, "*An Introduction to Reliability and Maintainability Engineering*," The McGraw-Hill Companies, 1997.

29. C. M. Fonseca and P.J. Fleming, "Multiobjective genetic algorithms", in *Genetic Algorithms in Engineering Systems*, A. M. S. Zalzal and P.J. Fleming (Eds), The Institution of Electrical Engineers, 1997.
30. L.R. Foulds, "Optimization Techniques. An Introduction", Springer-Verlag New York Inc, 1981.
31. S.J. Friedman and K.J. Supowit, "Finding the optimal variable ordering for Binary Decision Diagrams," *IEEE Trans. Computers*, Vol. 39, No. 5, pp710-714, May 1990.
32. J.B. Fussell and W.E. Vesely, "A new methodology for obtaining cut sets for a fault tree," *Trans. Amer. Nuc. Soc.*, Vol. 15, p262, 1972.
33. P.M. Ghare, R.E. Taylor, "Optimal Redundancy for reliability in Series Systems", *Operations Research*, Vol. 17, pp838-847, 1975.
34. M. Gen, R. Cheng, "*Genetic Algorithms and Engineering Design*", John Wiley and Sons, 1997.
35. F. Glover and H. Greenberg, "New approaches for heuristic search: a bilateral linkage with artificial intelligence", *European Journal of Operational Research*, Vol. 39, pp119-130, 1989.
36. D. Goldberg, B. Korb, and K. deb, "Messy genetic algorithms,: motivation, analysis and first results", *Complex Systems*, Vol. 3, pp493-530, 1989.
37. D. E. Goldberg, "*Genetic Algorithms in Search. Optimization and Machine Learning*", Addison-Wesley Publishing Company, 1989.
38. D. M Grove, T. P Davies, "Engineering Quality and Experimental Design", Longman Scientific and Technical, 1992.
39. M. Hamada, "Using statistically designed experiments to improve reliability and to achieve robust reliability", *IEEE Transactions on Reliability*, Vol. 44, No. 2, 1995, pp 206-215.
40. D.F. Hassl, N.H. Roberts, W.E. Vesely and F.F. Goldberg, "*Fault Tree Handbook*", US Nuclear Regulatory Commission, 1981, NUREG-0492.
41. E.J. Henley and H. Kumamoto, "*Reliability Engineering and Risk Assessment*," Englewood Cliffs, 1981.
42. F.S. Hillier, G.J. Lieberman, "Introduction to Mathematical Programming", 2nd ed., McGraw-Hill, Inc, 1995.
43. D.M. Himmelblau, "Applied Nonlinear Programming", McGraw-Hill, Inc, 1972.
44. J. Holland, "*Adaption in Natural and Artificial Systems*", University of Michigan Press, 1975.

45. A. Homaifar, C. Qi and S. Lai, "Constrained optimisation via genetic algorithms", *Simulation*, Vol. 62, no. 4, pp242-254, 1994.
46. Y. Hu, C. Storey, "Efficient Generalised Conjugate Gradient Algorithms, Part 2: Implementation", *Journal of Optimisation Theory and Applications*, Vol. 69, No. 1, pp139-152, 1991.
47. A. Joseph, S.I. Gass and N.A. Bryson, "Nearness and Bound Relationships Between an Integer-Programming Problem and its Relaxed Linear-Programming Model", *Journal of Optimisation Theory and Applications*, Vol.98, No. 1, pp55-63, 1998.
48. J. Kallrath, J.M. Wilson, "Business Optimisation Using Mathematical Programming", Macmillan Press Ltd, 1997.
49. C.Y. Lee, "Representation of switching circuits by binary decision diagrams," *Bell Syst. Tech. J.*, No. 38, pp985-999, July 1959.
50. N. Limnios and R. Ziani, "An algorithm for reducing the minimal cut sets in fault tree analysis," *IEEE Trans. Reliability*, Vol. R-35, No. 5, Dec, pp559-461, 1986.
51. D. K. J Lin, "Making full use of Taguchi's orthogonal arrays", *Quality and Reliability Engineering International*, Vol. 10, 1994, pp 117-121.
52. Y. Liu, C. Storey, "Efficient Generalised Conjugate Gradient Algorithms, Part 1: Theory", *Journal of Optimisation Theory and Applications*, Vol. 69, No. 1, pp129-137, 1991.
53. M.O. Locks, "Modularizing, minimizing, and interpreting the K & H fault-tree," *IEEE Trans. Reliability*, Vol. R-30, Dec, pp411-415, 1981.
54. D.G. Luenberger, "Linear and Nonlinear Programming", 2nd ed., Addison-Wesley Publishing, 1991.
55. D.W. McLeavey, J.A. McLeavey, "Optimisation of System Reliability by Branch and Bound", *IEEE Transactions on Reliability*, Vol. R-25, No. 5, pp327-329, 1976.
56. Z. Michalewicz, "*Genetic Algorithms + Data Structures = Evolution Programs*", Springer Verlag, 1992.
57. Z. Michalewicz, "A survey of constraint handling techniques in evolutionary computation methods", in *Evolutionary Programming IV*, J. McDonnell, R. Reynolds and D. Fogel (Eds.), MIT Press, Cambridge, MA, 1995.
58. S. Minato, N. Ishiura and S. Yajima, "Shared binary decision diagram with attributed edges for efficient Boolean function manipulation," *Proceedings of the 27th ACM/IEEE Design Automation Conf.*, pp52-57, June 1990.

59. K.B. Misra, "Reliability Optimisation through sequential simplex search", *International Journal of Control*, Vol. 18, No. 1, 1973.
60. K.B. Misra, U. Sharma, "An Efficient Algorithm to Solve Integer-Programming Problems Arising in System Reliability Design", *IEEE Transactions on Reliability*, Vol. 40, No. 1, pp81-91, 1991.
61. J.E. Mitchell, "Fixing Variables and Generating Classical Cutting Planes when using an Interior Point Branch and Cut Method to Solve Integer-Programming Problems", *European Journal of Operational Research* Vol. 100, pp623-628, 1997.
62. D. C. Montgomery, "Design and Analysis of Experiments", John Wiley & Sons, 3rd Edition, 1991.
63. M. S. Phadke, "Quality Engineering using Robust Design", P T R Prentice-Hall, 1995.
64. R. L. Plackett, J. P. Burman, "The design of optimal multifactorial experiments", *Biometrika*, Vol. 33, pp 305-325.
65. W.L. Price, "A Controlled Random Search Procedure For Global Optimisation", *Toward Global Optimisation 2*, North-Holland Publishing Company, 1978.
66. W.L. Price, "Global Optimisation by Controlled Random Search", *Journal of Optimisation Theory and Applications*, Vol. 40, No. 3, pp333-348, 1983.
67. W.L. Price, "Global Optimisation Algorithm for a CAD Workstation", *Journal of Optimisation Theory and Applications*, Vol. 55, No. 1, pp133-145, 1987.
68. D. Quagliarella and A. Vicini, "Coupling genetic algorithms and gradient based optimisation techniques", in *Genetic Algorithms and Evolution Strategy in Engineering and Computer Science*, D. Quagliarella, J. Periaux, C. Polini and G. Winter (Eds.), John Wiley and Sons, 1998.
69. G.V. Reklaitis, A. Ravindran and K.M. Ragsdell, "Engineering Optimisation. Methods and Applications", John Wiley and Sons, 1983.
70. D.M. Rasmuson and N.H. Marshall, "FATRAM – A core efficient cut set algorithm," *IEEE Trans. Reliability*, Vol. R-27, Oct, pp250-253, 1978.
71. A. Rauzy, "New algorithms for fault tree analysis," *Reliability Engineering and System Safety*, Vol. 40, pp203-211, 1993.
72. S. Ronald, "Robust encodings in genetic algorithms", in *Evolutionary Algorithms in Engineering Application*, Das, 1997.

86. Z.B. Tang, "Optimal Sequential Sampling Policy of Partitioned Random Search and its Approximation", *Journal of Optimisation Theory and Applications*, Vol. 98, No. 2, pp 431-448, 1998.
87. F.A. Tillman, "Optimisation by Integer Programming of Constrained Reliability Problems with Several Modes of Failure", *IEEE Transactions on Reliability*, Vol. R-18, No. 2, pp47-53, 1969.
88. F.A. Tillman, C. Hwang and W. Kuo, "Optimisation of Systems Reliability", Marcel Dekker, Inc, 1980.
89. T.J. Van Roy, L.A. Wolsey, "Solving Mixed 0-1 Programs by Automatic Reformulation", *Operations Research*, Vol. 35, pp45-57, 1987.
90. W.E. Vesely, "A time-dependent methodology for fault tree analysis," *Nucl. Eng. And Design*, Vol. 13, Aug, pp337-360, 1970.
91. W.E. Vesely and R.E. Narum, "PREP and KITT computer cose for the automatic evaluation of a fault tree," *Idaho Nuclear Corporation*, Idaho Falls, Idaho, IN-1349, 1970.
92. D.B. Wheeler at al, "Fault tree analysis using bit manipulation," *IEEE Trans. Reliability*, Vol. R-26, June, pp95-99, 1977.
93. J.M. Wilson, "Modularizing and minimising fault trees," *IEEE Trans. Reliability*, Vol. R-34, No. 4, Oct, pp320-322, 1985.
94. R. Yang and I. Douglas, "Simple genetic algorithm with local tuning: efficient global optimizing technique", communicated by W. B. Wong, *Journal of Optimisation Theory and Applications*, Vol. 98, No. 2, pp 449-465, August, 1998.
95. J.D. Andrews, "Optimal Safety System Design Using Fault Tree Analysis", *Proc. ImechE*, Vol. 208, pp123-131, 1994.
96. R.M. Sinnamon, "BDD for Fault Tree Analysis", PhD Thesis, 1996.

APPENDIX I - *.ats File Format

The fault tree structure file with extension 'ats' is an ASCII file in which each line contains a definition for a gate in the fault tree. The format of his file is as follows.

The name of the gate is stated at the start of the line. Ten columns are allocated to the gate name and should, therefore, consist of no more than 10 alphanumeric characters. The next 7 columns define the gate type. The type label is one of either AND, OR or VOTE m/n (stated in upper case). For example a 3 from 5 vote gate would be written VOTE3/5. Priority and inhibit gates are replaced by AND gates on creation of the tree structure analysis file. Columns 18 and 20 indicate the number of gate and base events. The limit is set at 9 of each. The remaining columns are considered in sets of 10 and are used to represent input names of gates and primary events. Names are left justified and gate names precede events. Gates may be defined in the file in any order. However, all gates named as inputs to other gates must be defined at some point in the file.

An example tree structure is shown below.

Gate1	AND	1	1	Gate2	X1				
Gate2	VOTE2/4	1	4	Gate3	X2	X3	X4		
Gate3	OR	0	2	X2	X4				

APPENDIX II – ‘*aqd’ File Format

The quantitative data file with the extension ‘aqd’ is an ASCII file in which each event and its associated data are defined on two lines. This file has the following format.

The event name and failure probability model are specified on the first line. The event name is allocated in the first 10 columns. The model type indicator is stated in the 11th column. The indicators F, R, M, P and H represent the ‘fixed’, ‘rate’, ‘MTTF’, ‘dormant’ and ‘house’ model types respectively. The second line specifies the model parameters of which there are 4 or 5 depending on the model type. The model parameters for each model type in the required order are:

- F: Unavailability, unconditional failure intensity, (q, w).
- R: Failure rate, repair rate, (λ, ν).
- M: Mean time to failure, mean time to repair, (μ, τ).
- P: Failure rate, mean time to repair, inspection intervals, (λ, τ, θ).
- H: Unavailability (0 or 1) followed by 3 dummy variables, (0, 1).

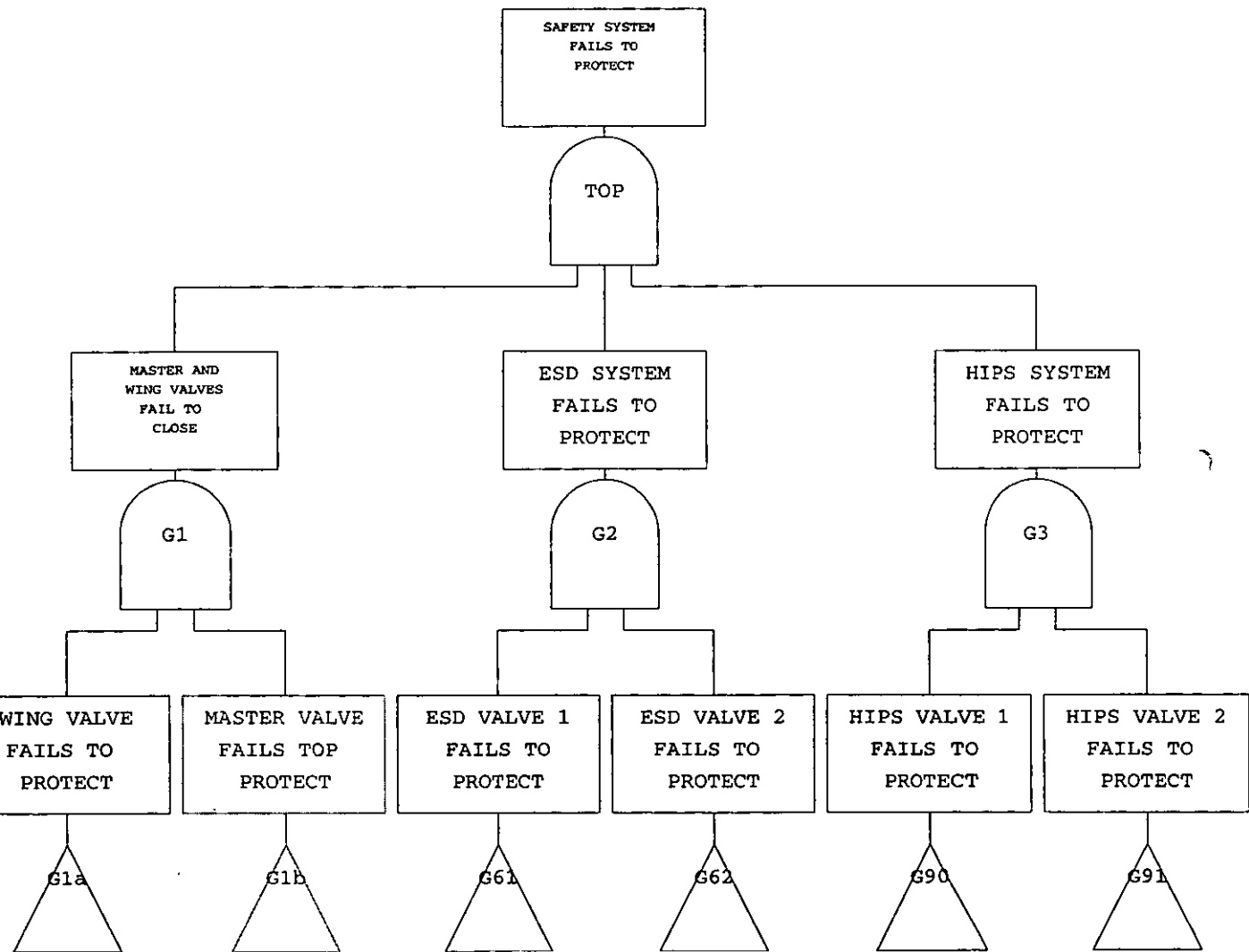
Consider the following example of the definition of an event in an ‘.aqd’ file:

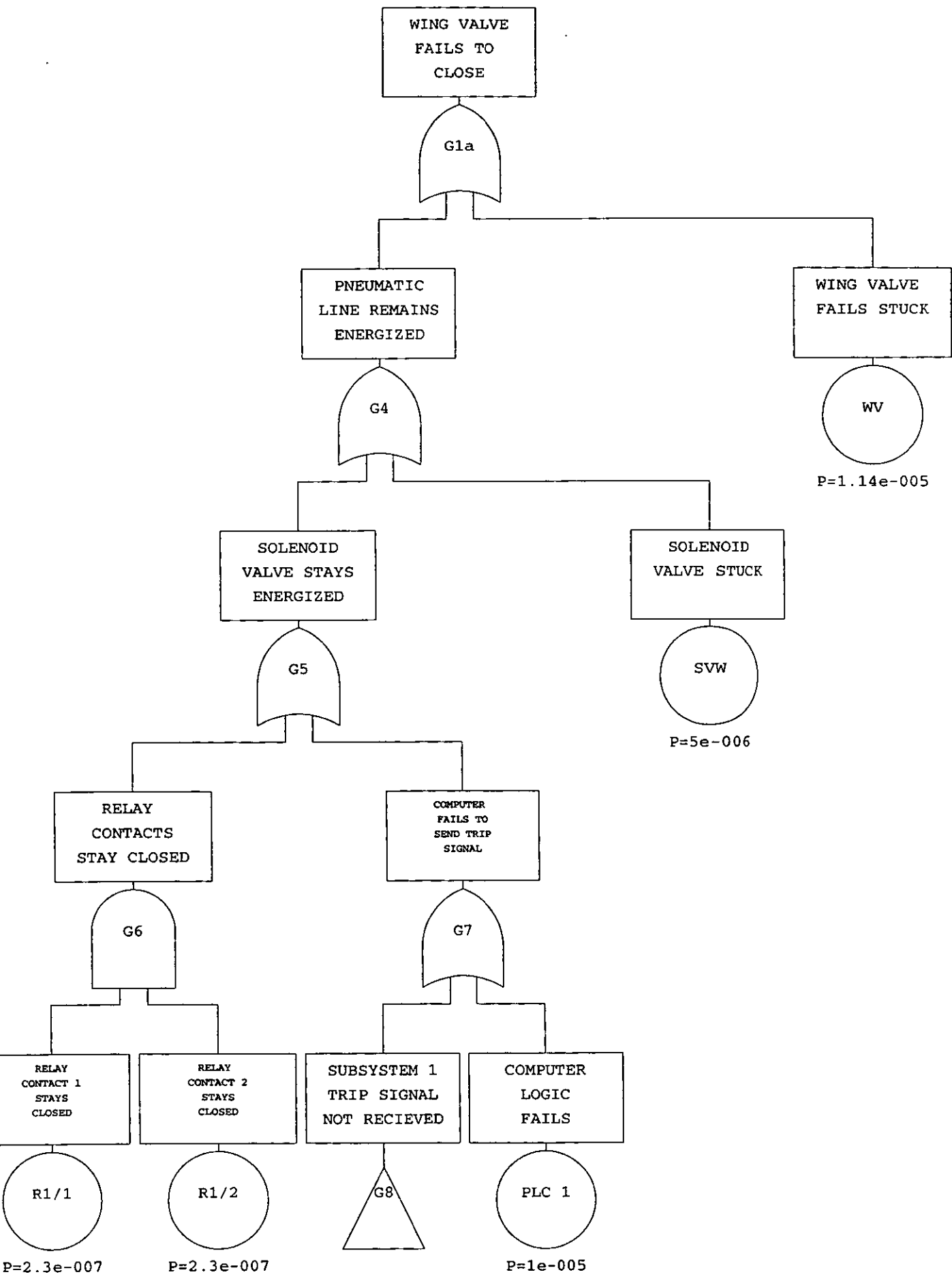
```
X1    R
0.12,140
```

This defines a basic event ‘X1’ with model code ‘R’, i.e. a ‘rate’ probability model is being used. The value of each quantitative parameter associated with the rate model type is specified in the second row. Specifically, 0.12 represents the failure rate and 140 the repair rate. No time units need be specified. Time units are assumed to be consistent.

APPENDIX III

FAULT TREE STRUCTURE FOR THE SYSTEM UNAVAILABILITY FAILURE MODE OF THE HIGH-INTEGRITY PROTECTION SYSTEM





WING VALVE
FAILS TO
CLOSE

G1a

PNEUMATIC
LINE REMAINS
ENERGIZED

G4

WING VALVE
FAILS STUCK

WV

P=1.14e-005

SOLENOID
VALVE STAYS
ENERGIZED

G5

SOLENOID
VALVE STUCK

SVW

P=5e-006

RELAY
CONTACTS
STAY CLOSED

G6

COMPUTER
FAILS TO
SEND TRIP
SIGNAL

G7

RELAY
CONTACT 1
STAYS
CLOSED

R1/1

P=2.3e-007

RELAY
CONTACT 2
STAYS
CLOSED

R1/2

P=2.3e-007

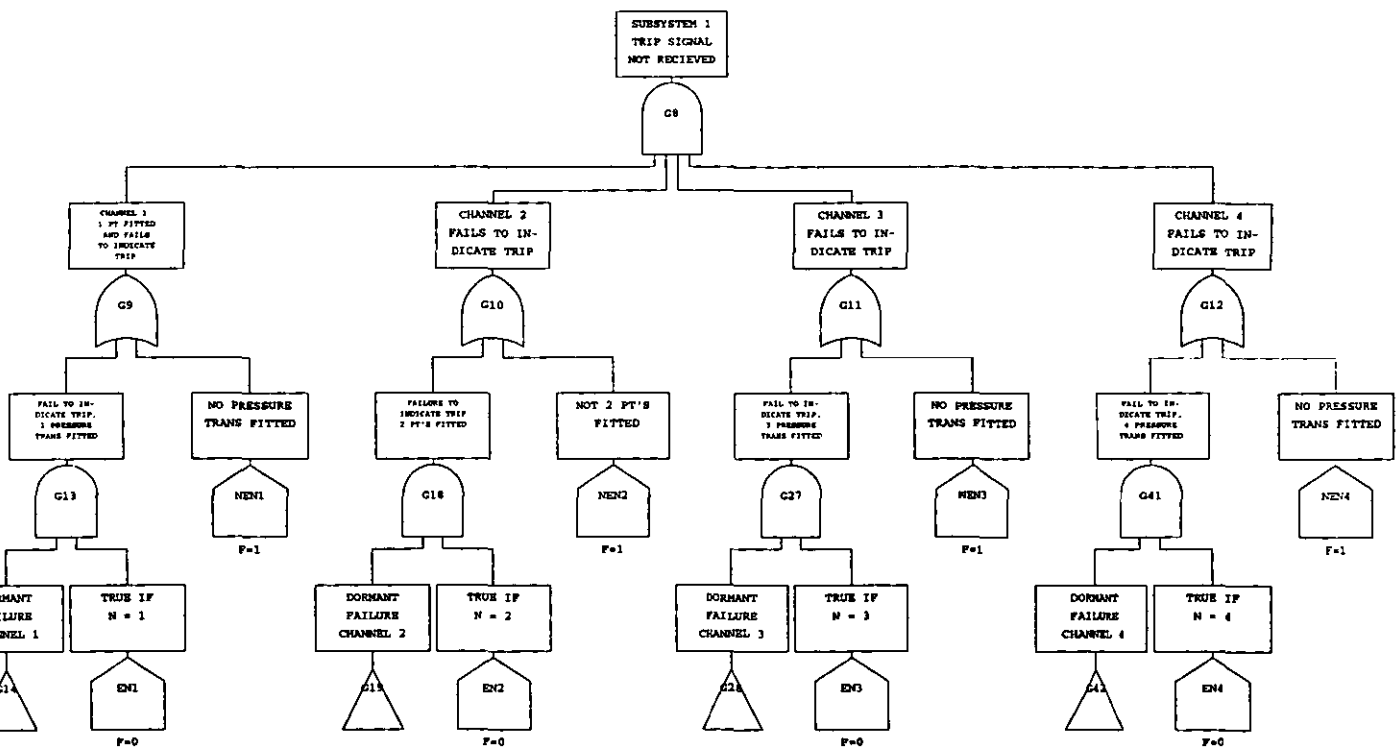
SUBSYSTEM 1
TRIP SIGNAL
NOT RECEIVED

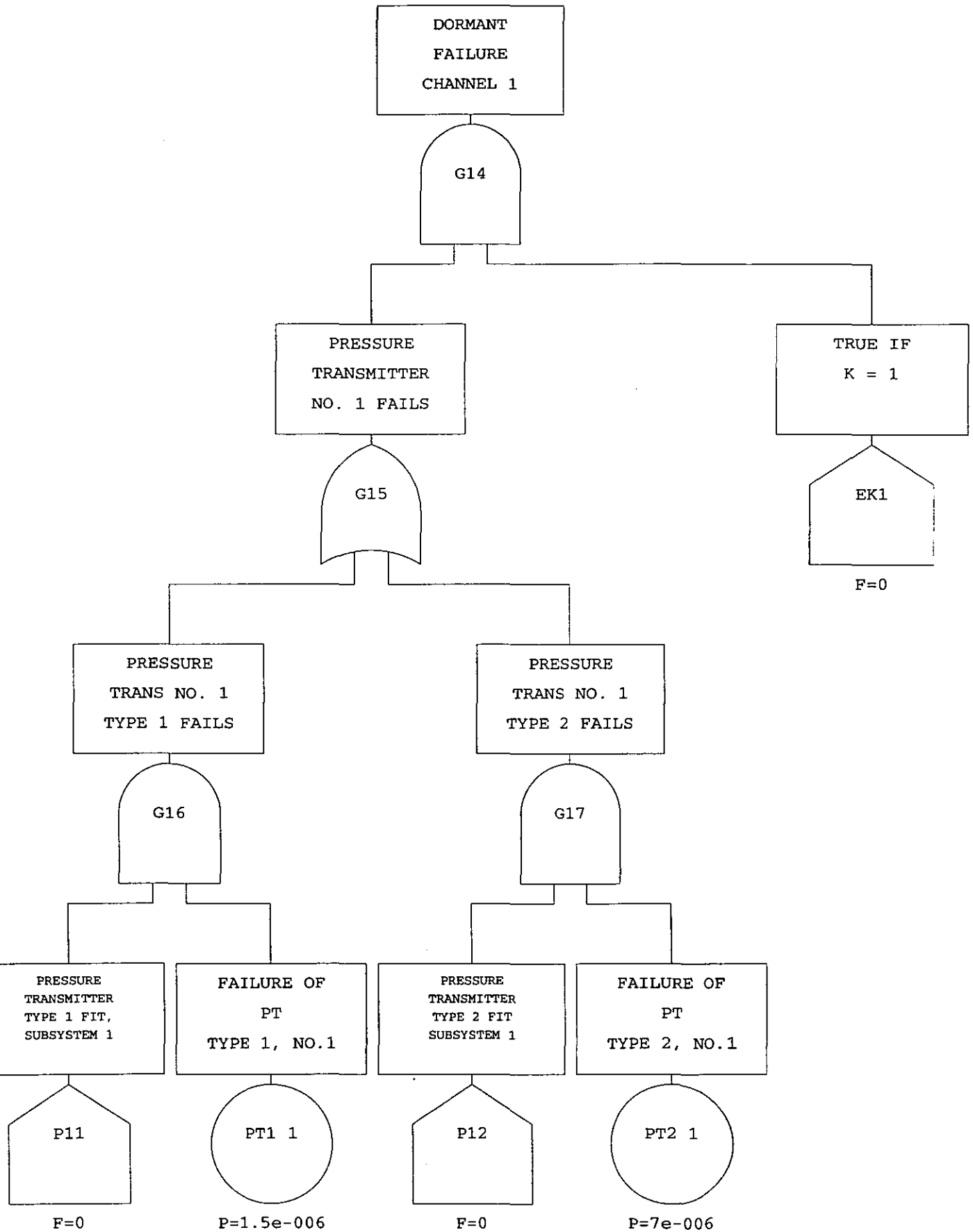
G8

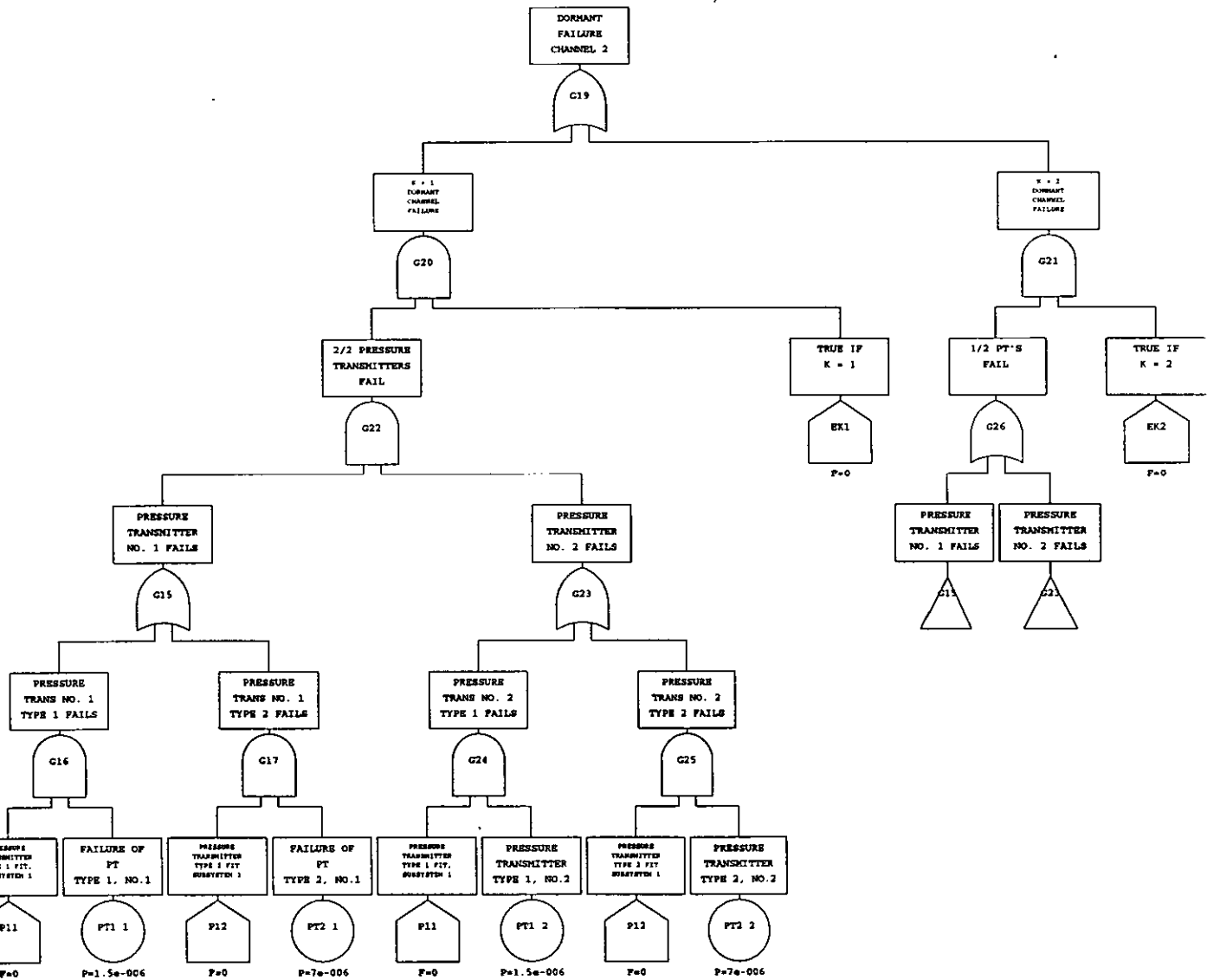
COMPUTER
LOGIC
FAILS

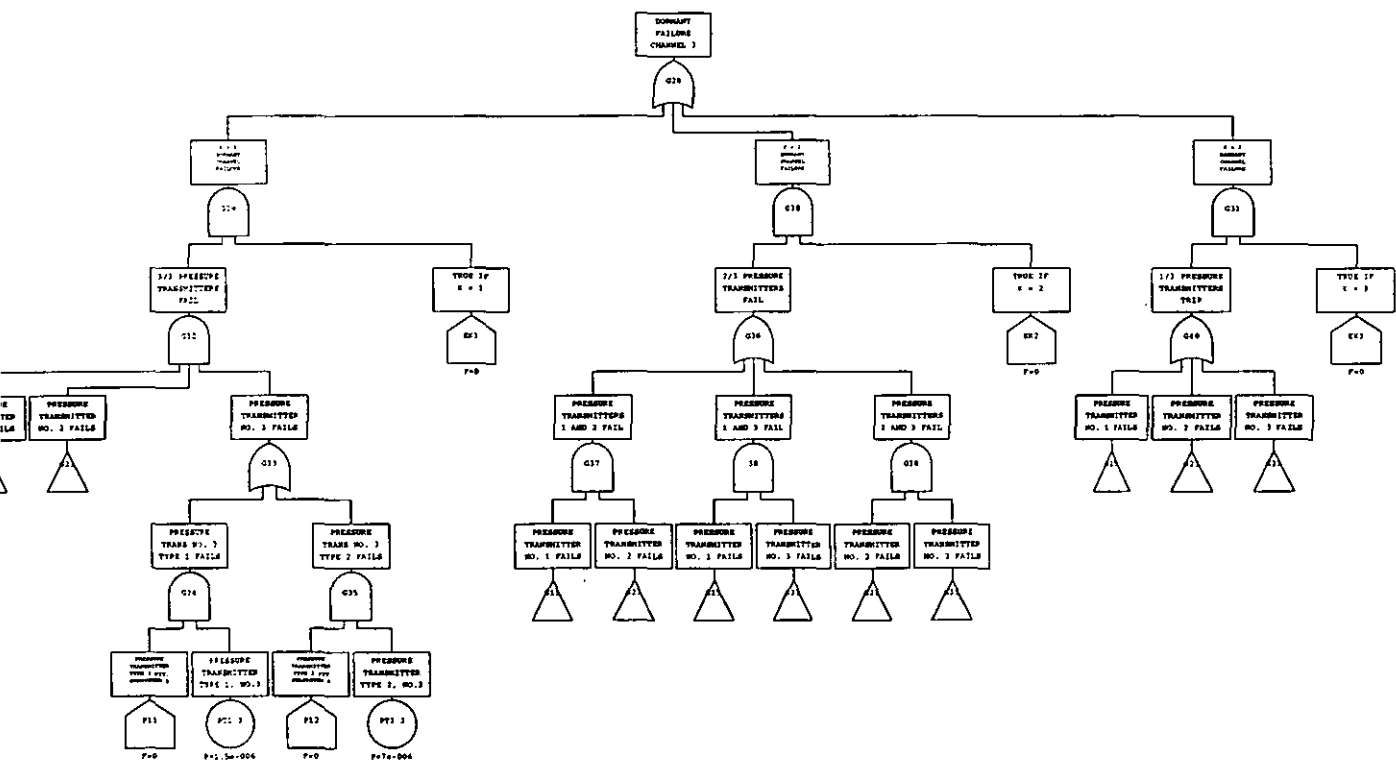
PLC 1

P=1e-005

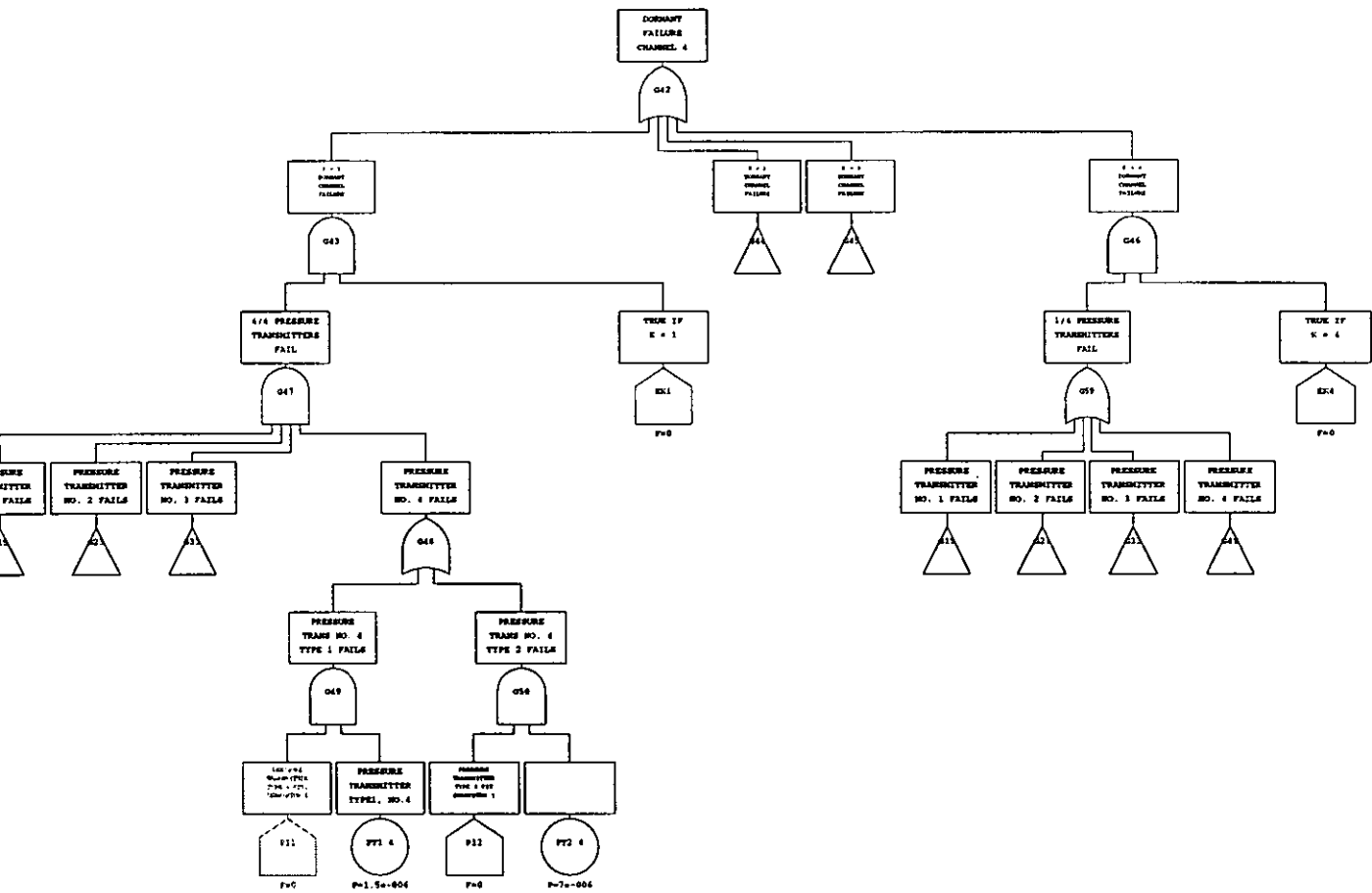


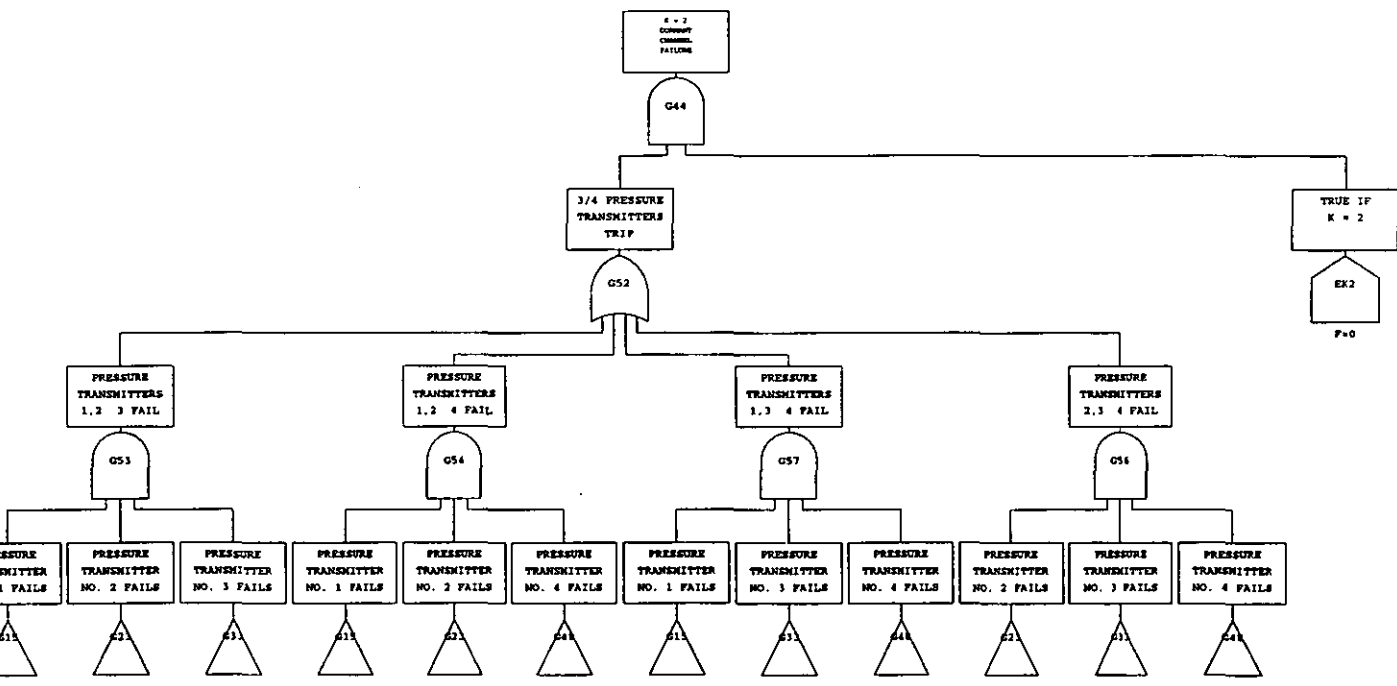


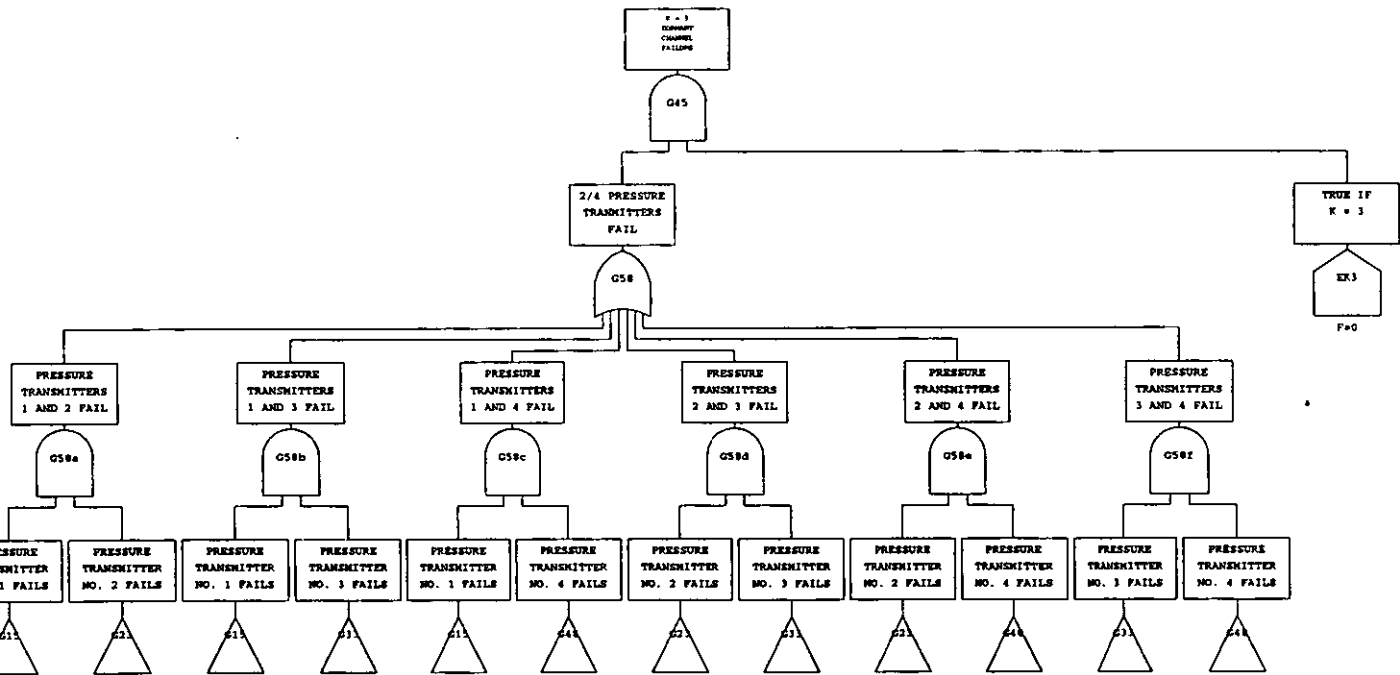




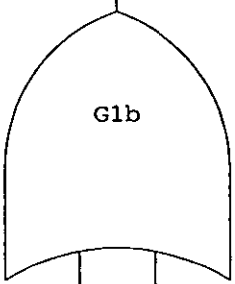
Can't read!





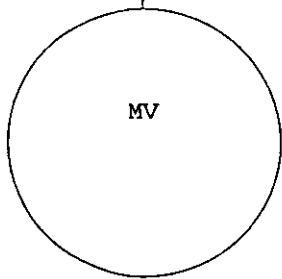
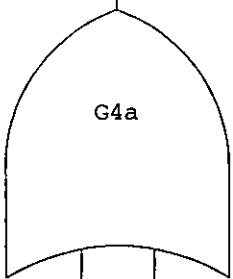


MASTER VALVE
FAILS TO
CLOSE



PNEUMATIC
LINE STAYS
ENERGIZED

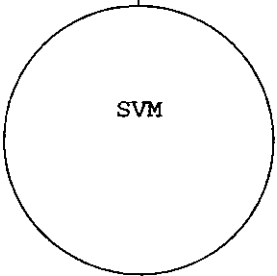
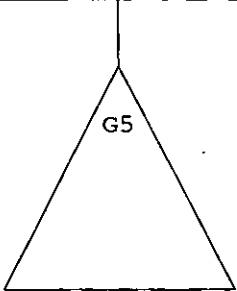
MASTER VALVE
FAILS STUCK



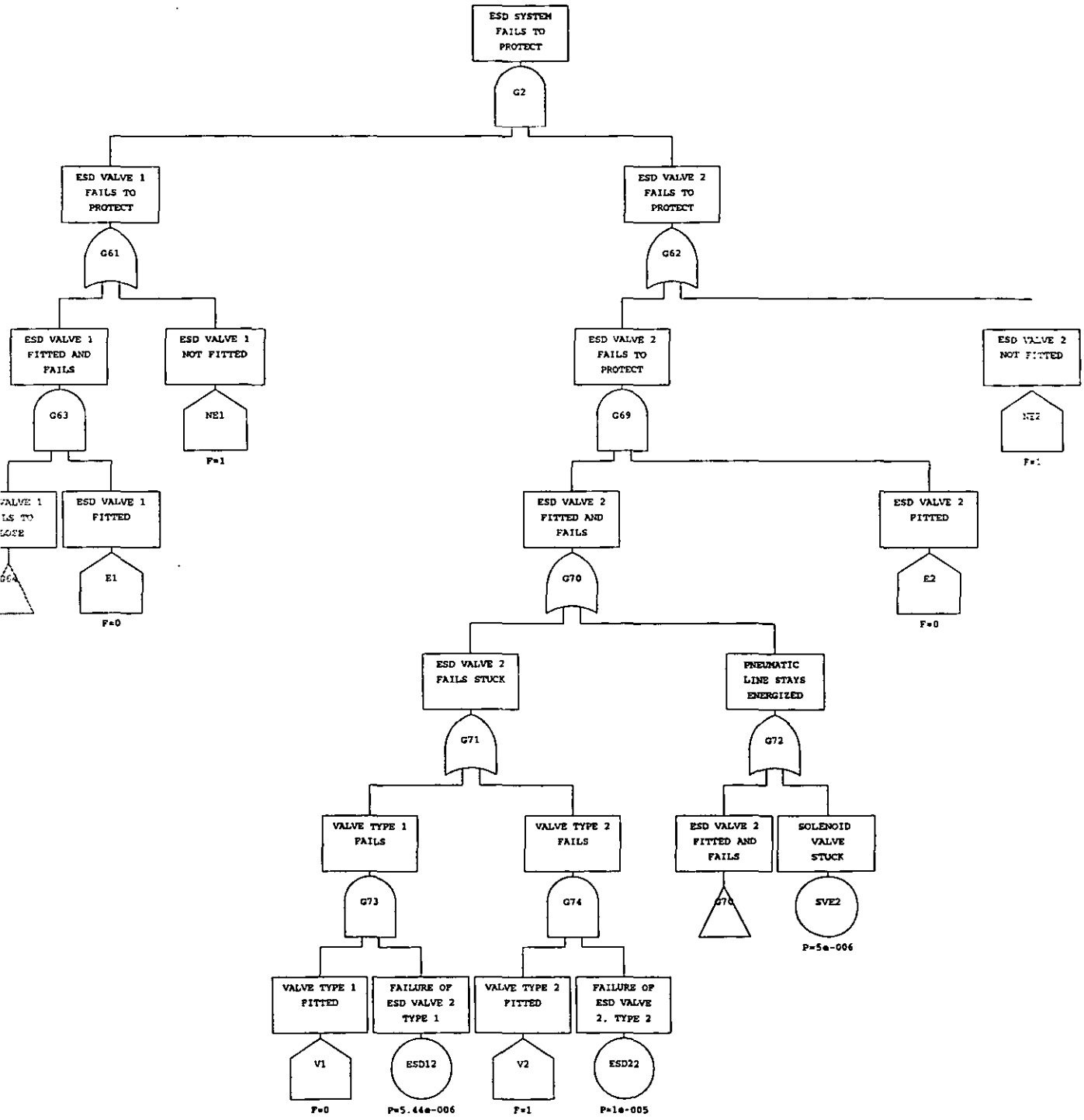
$P=1.14e-005$

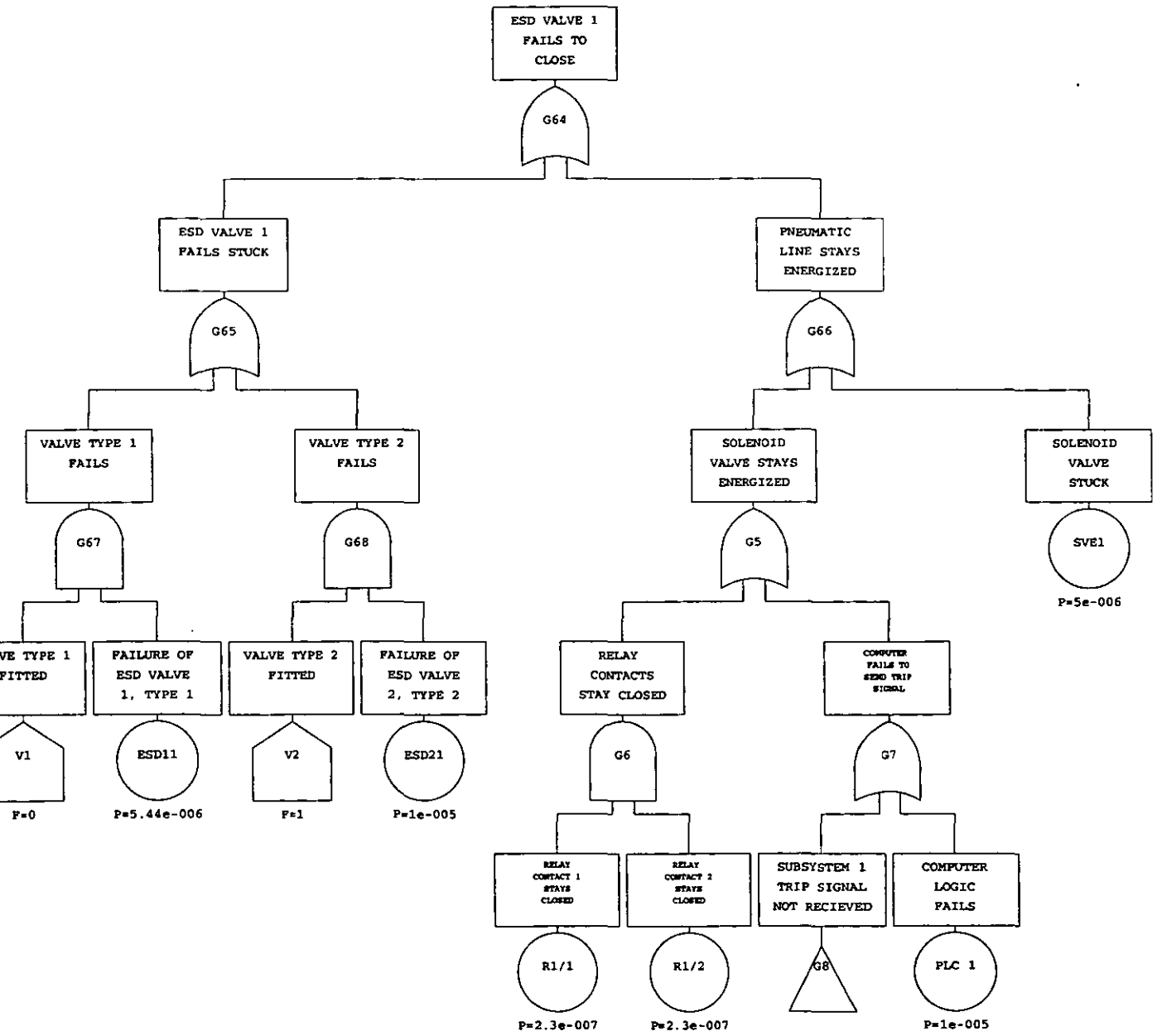
SOLENOID
VALVE STAYS
ENERGIZED

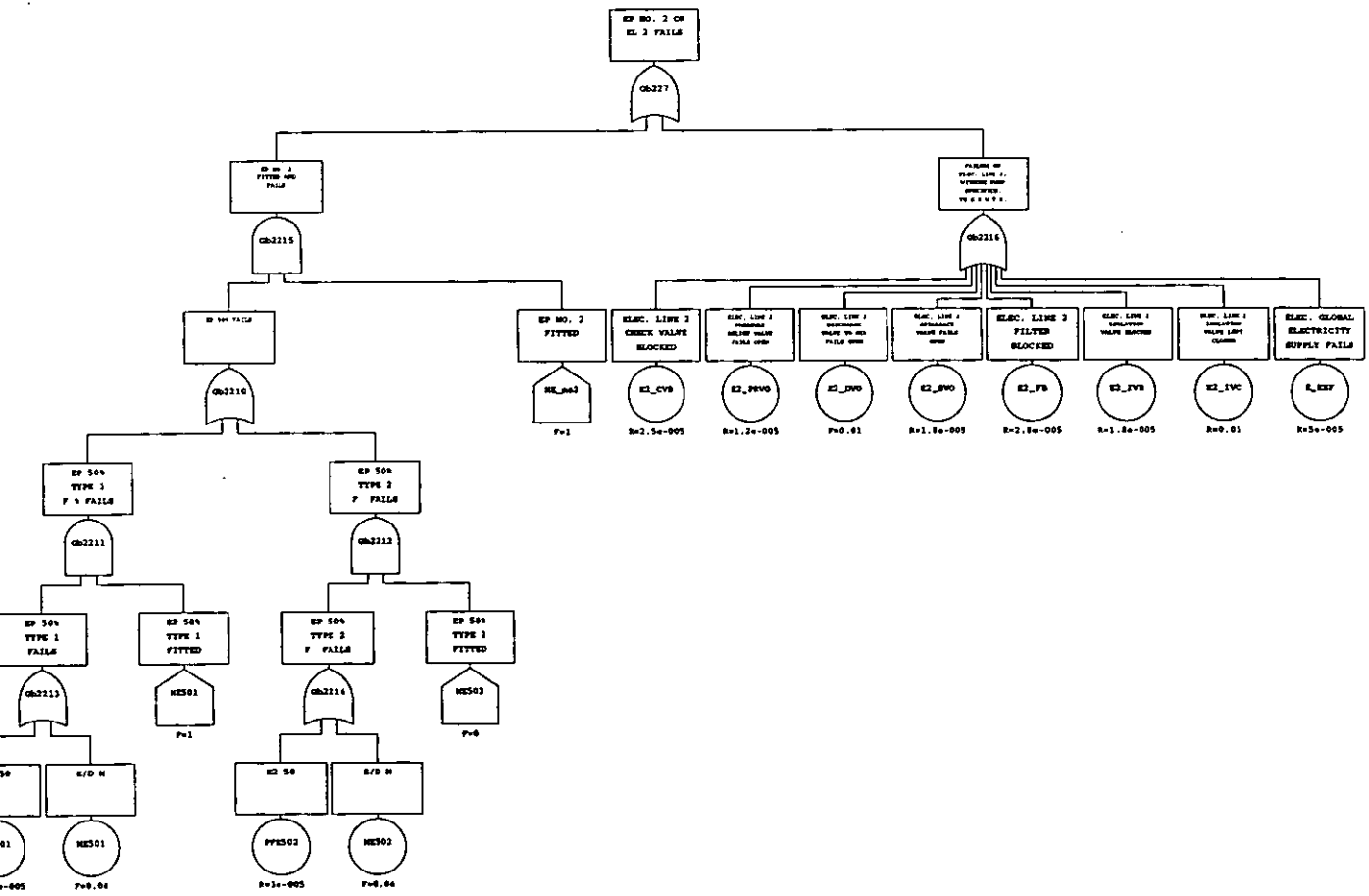
SOLENOID
VALVE
STUCK

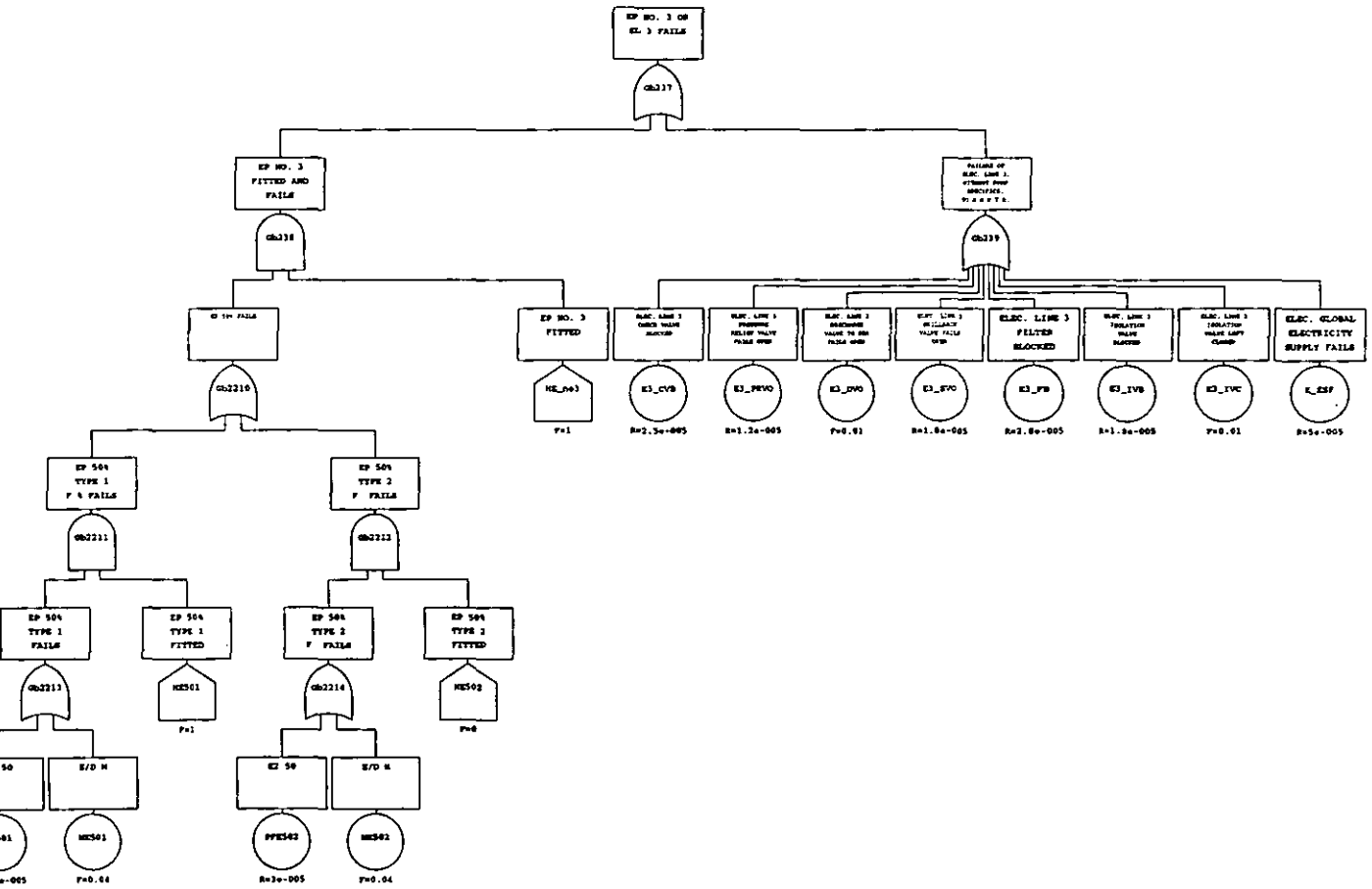


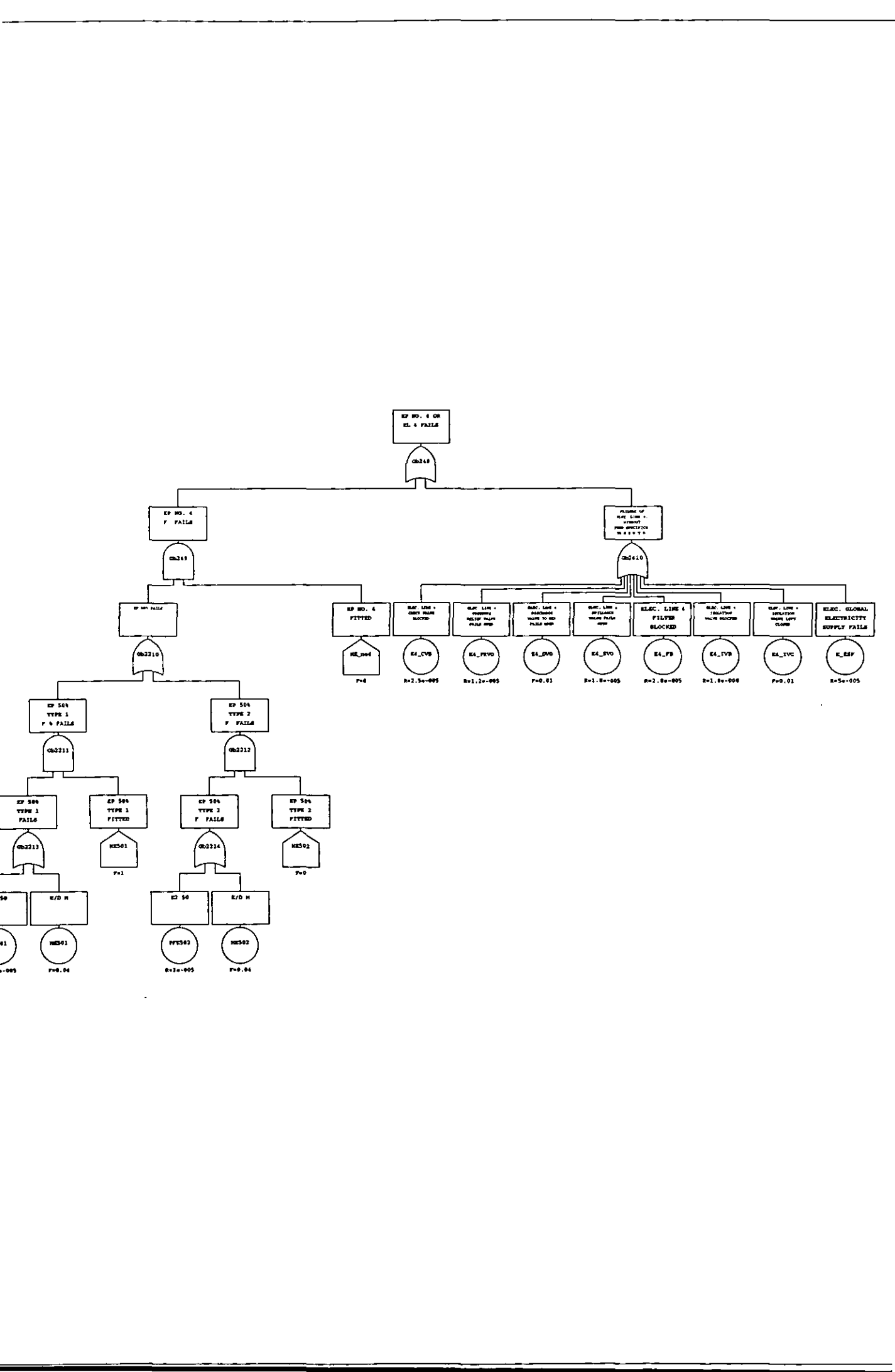
$P=5e-006$

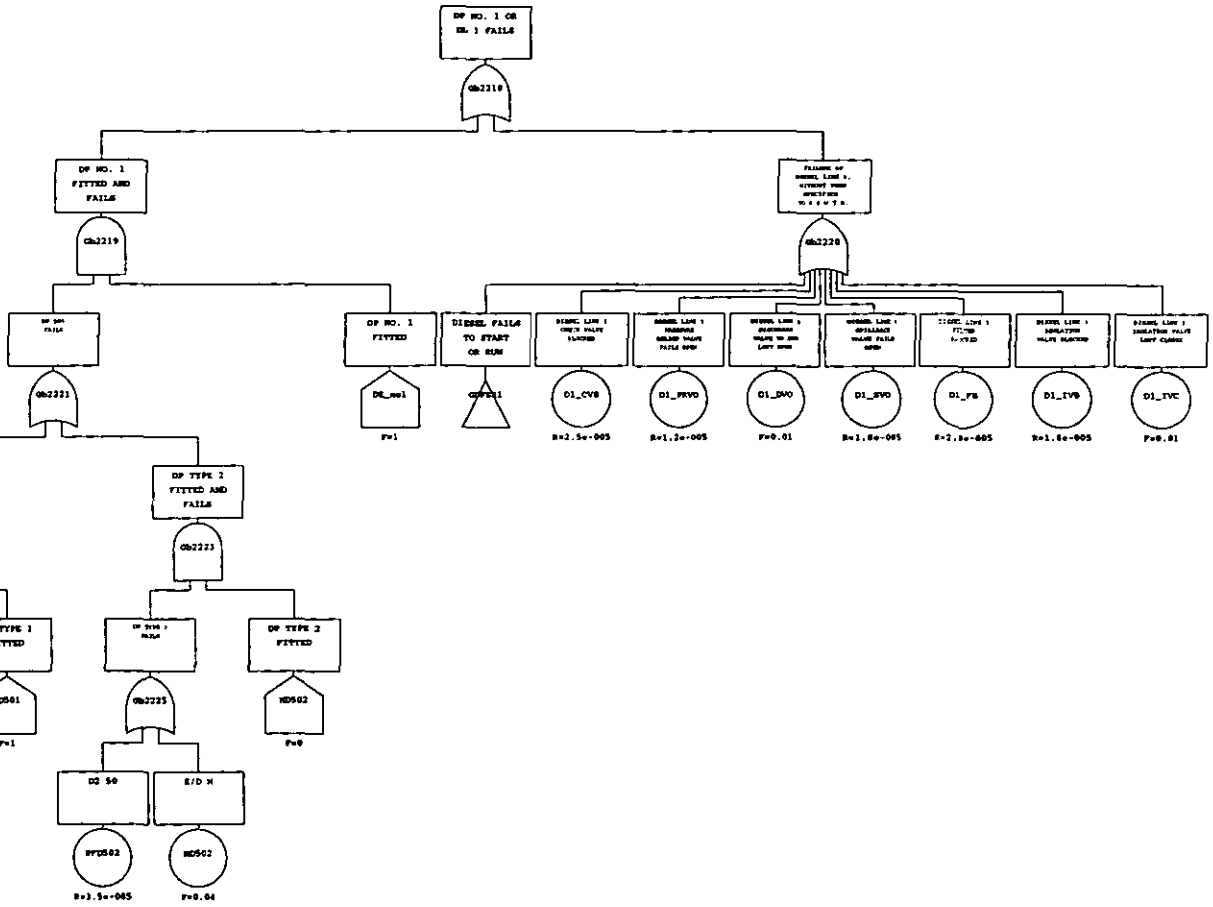


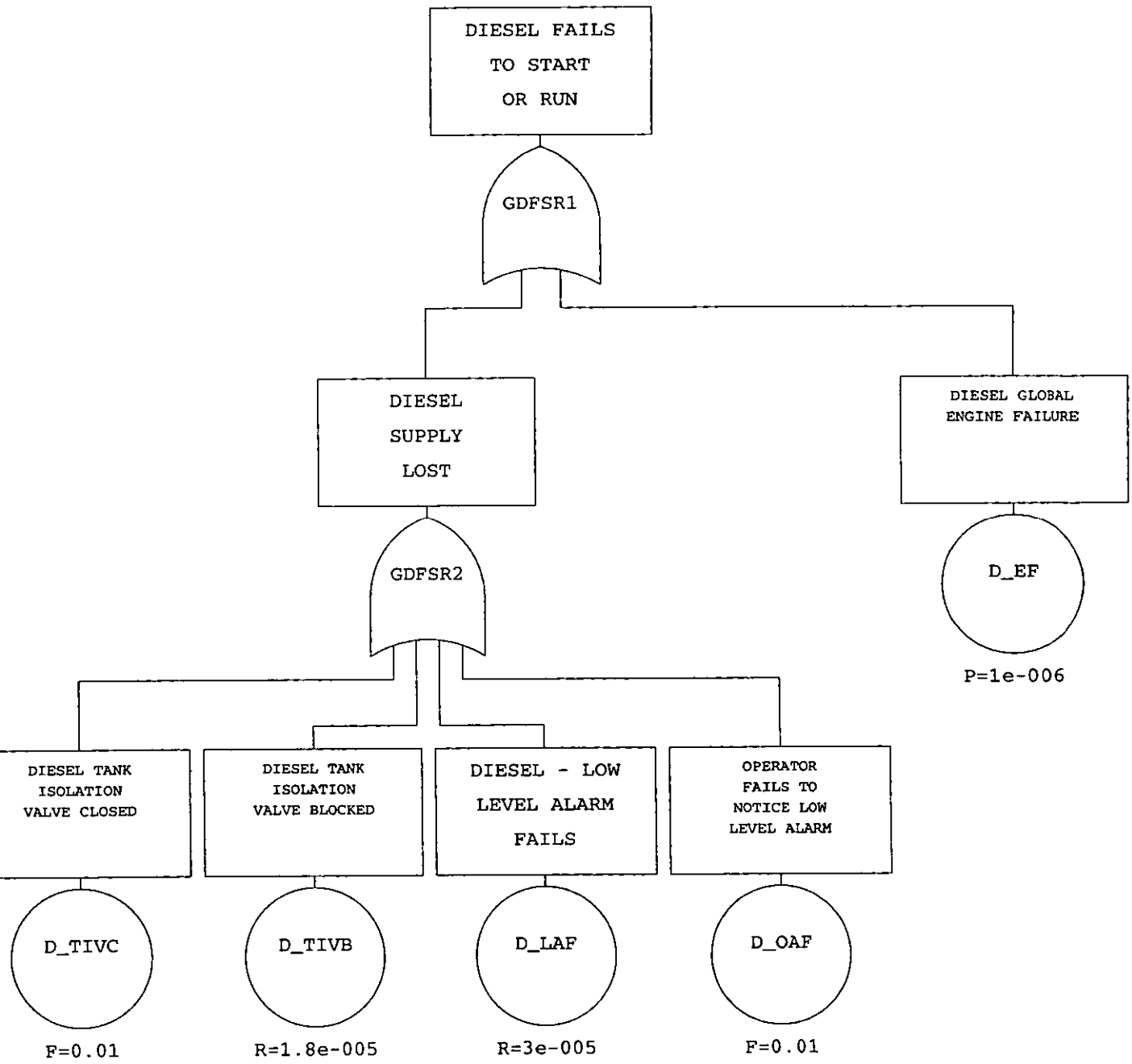


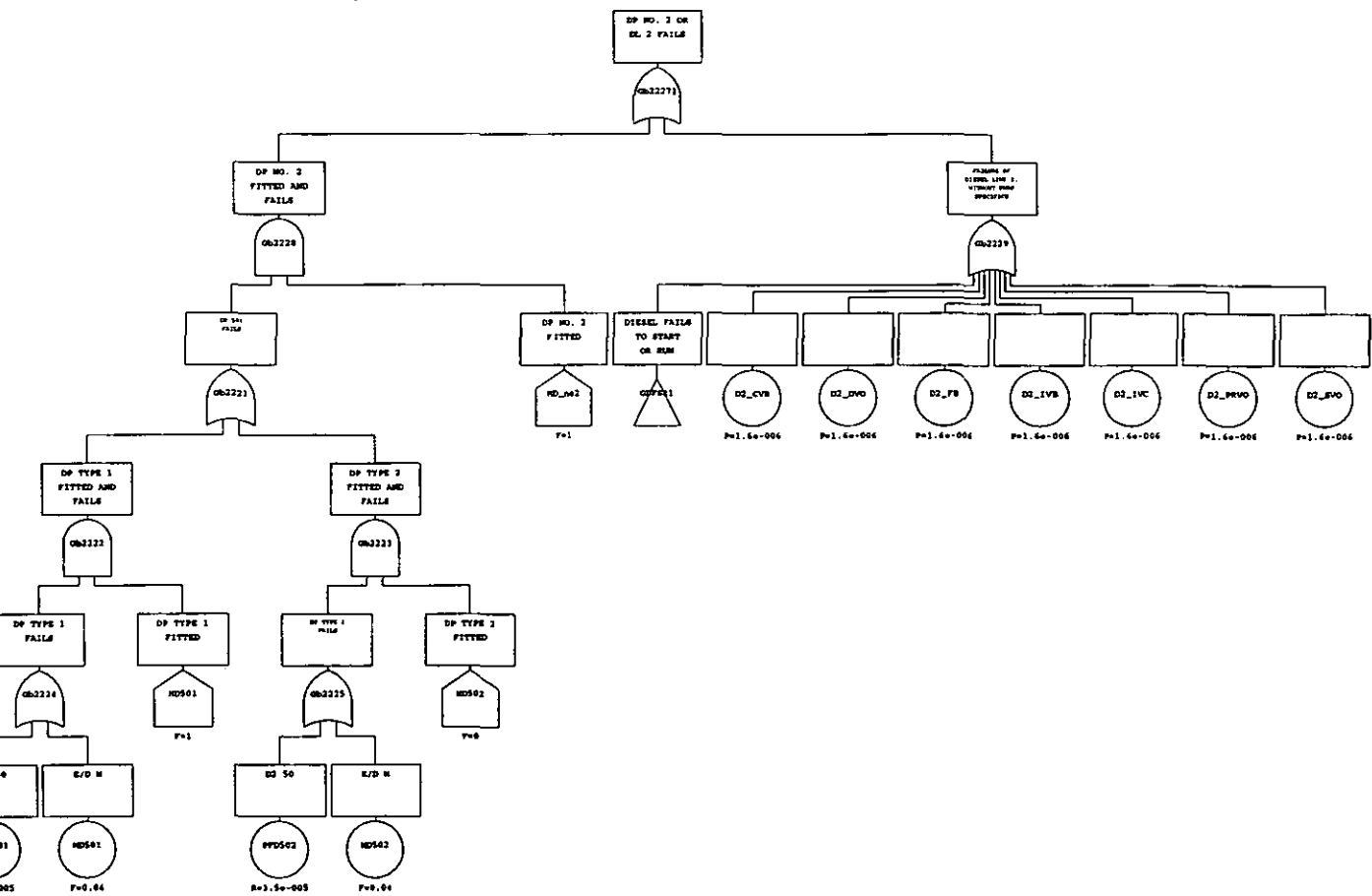


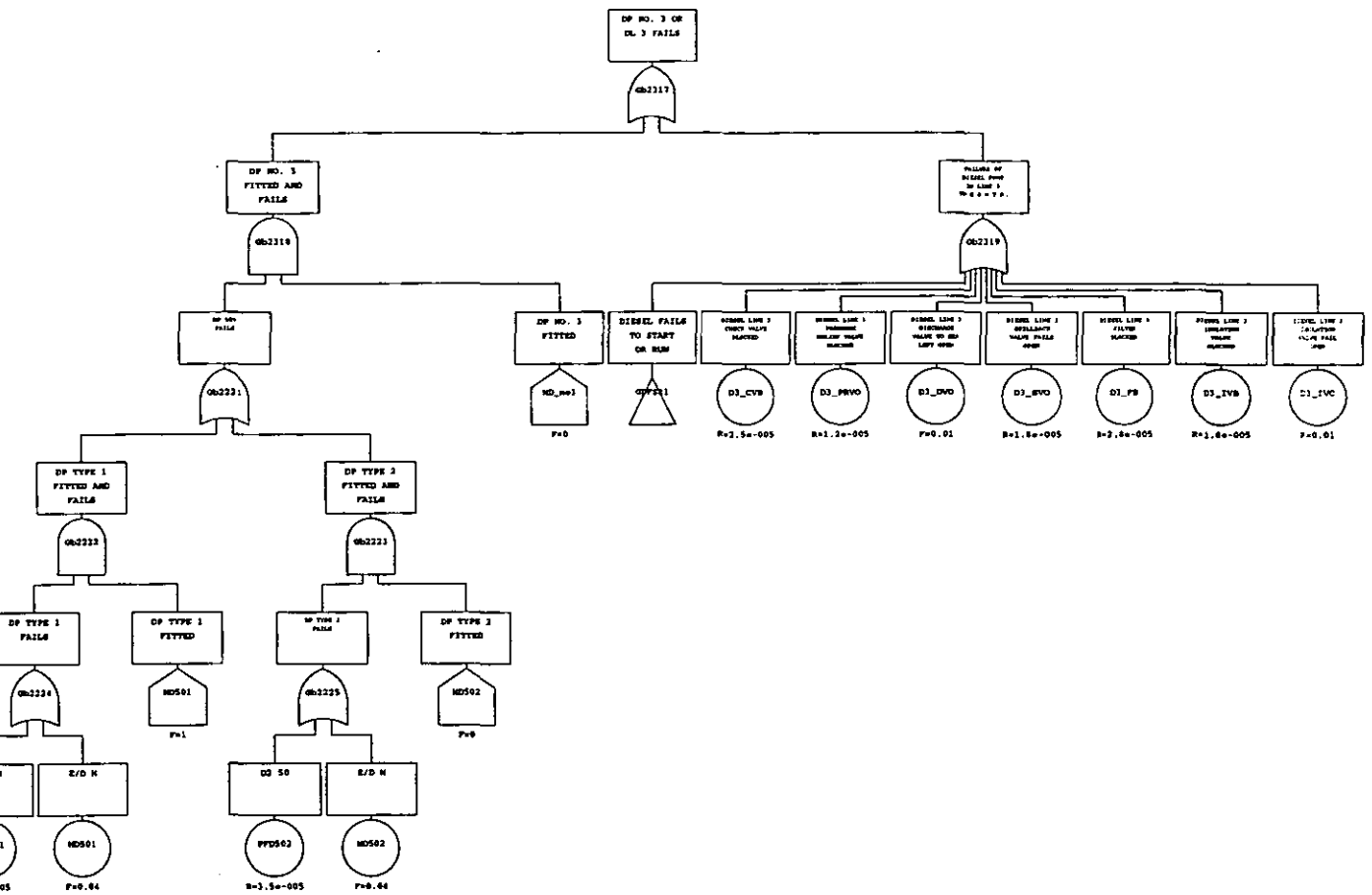


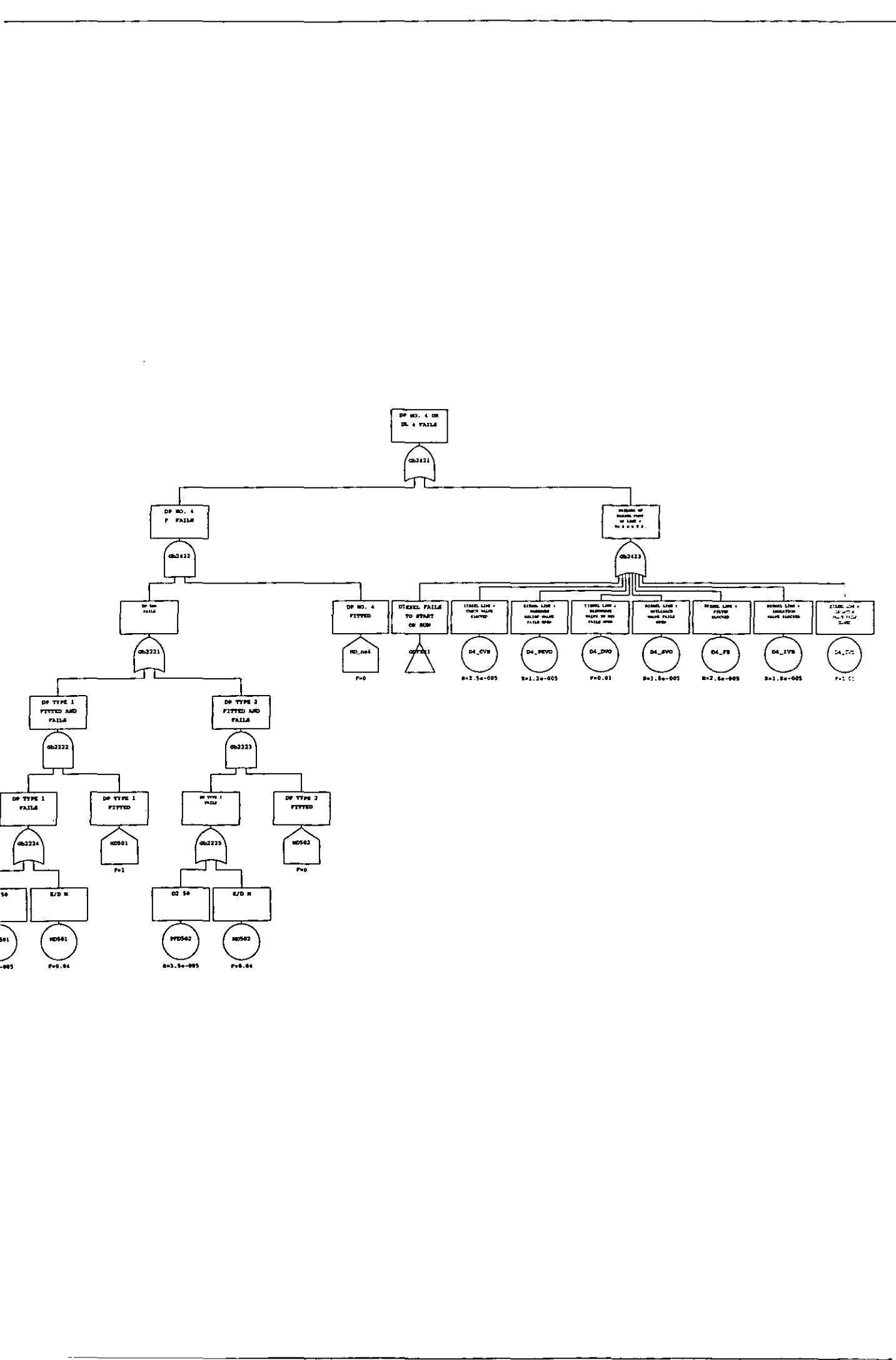


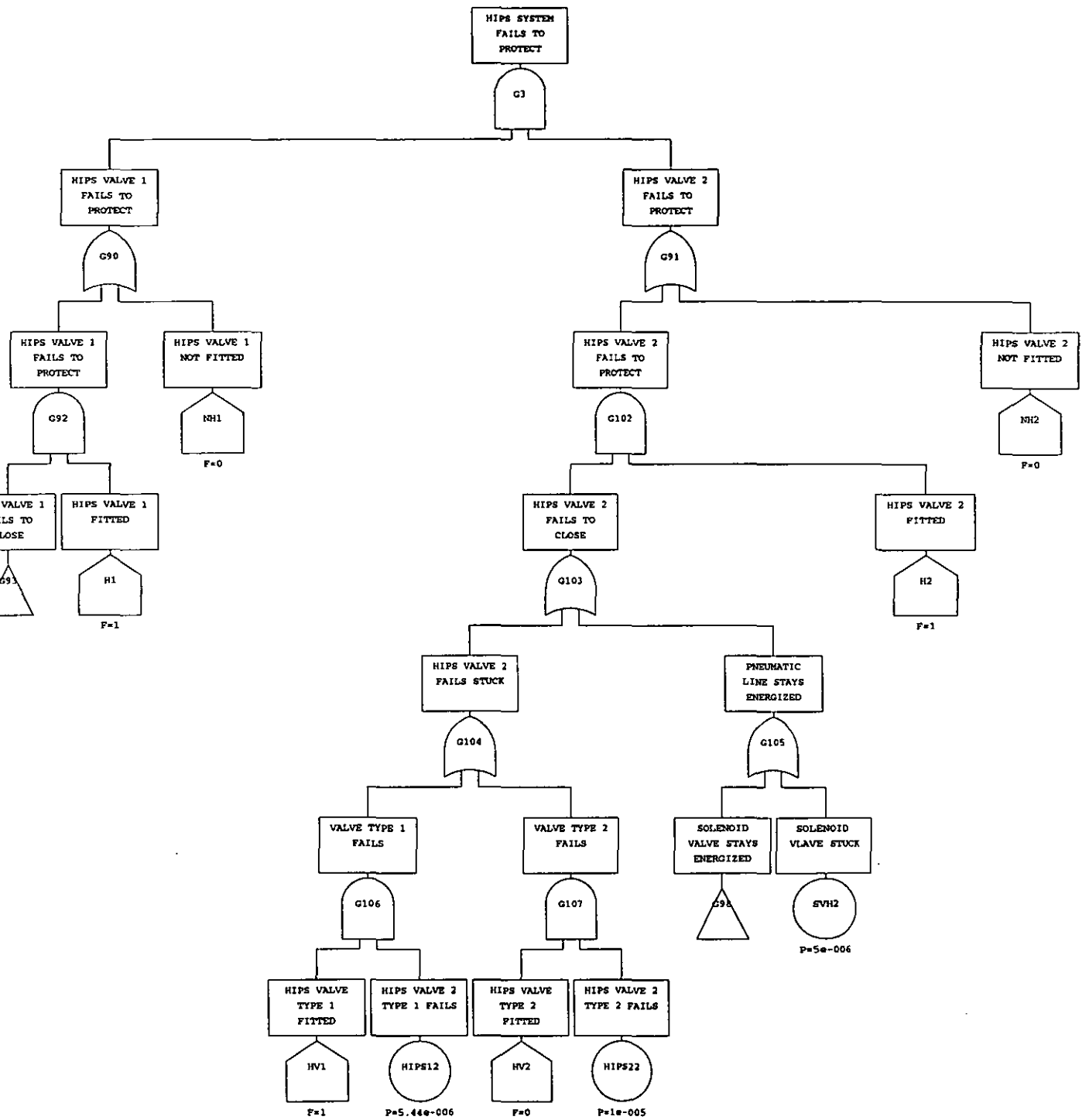


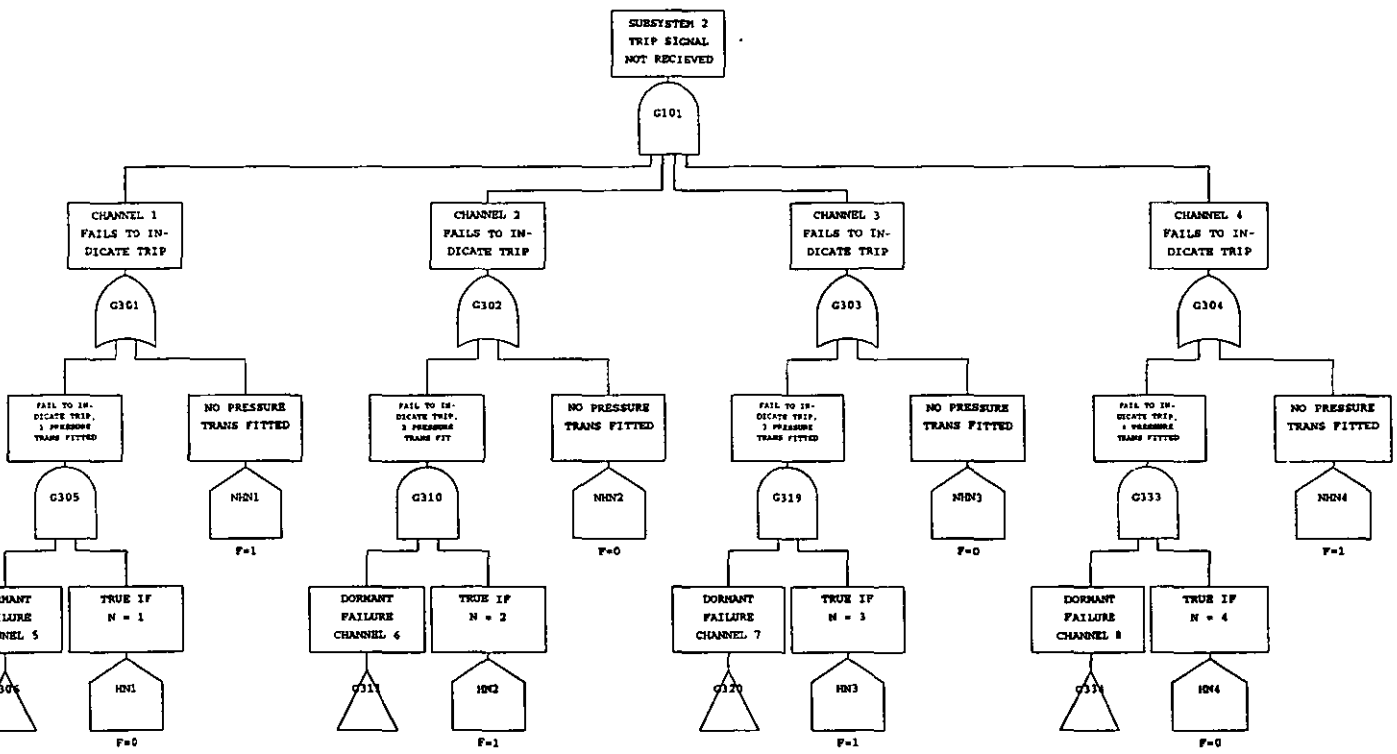


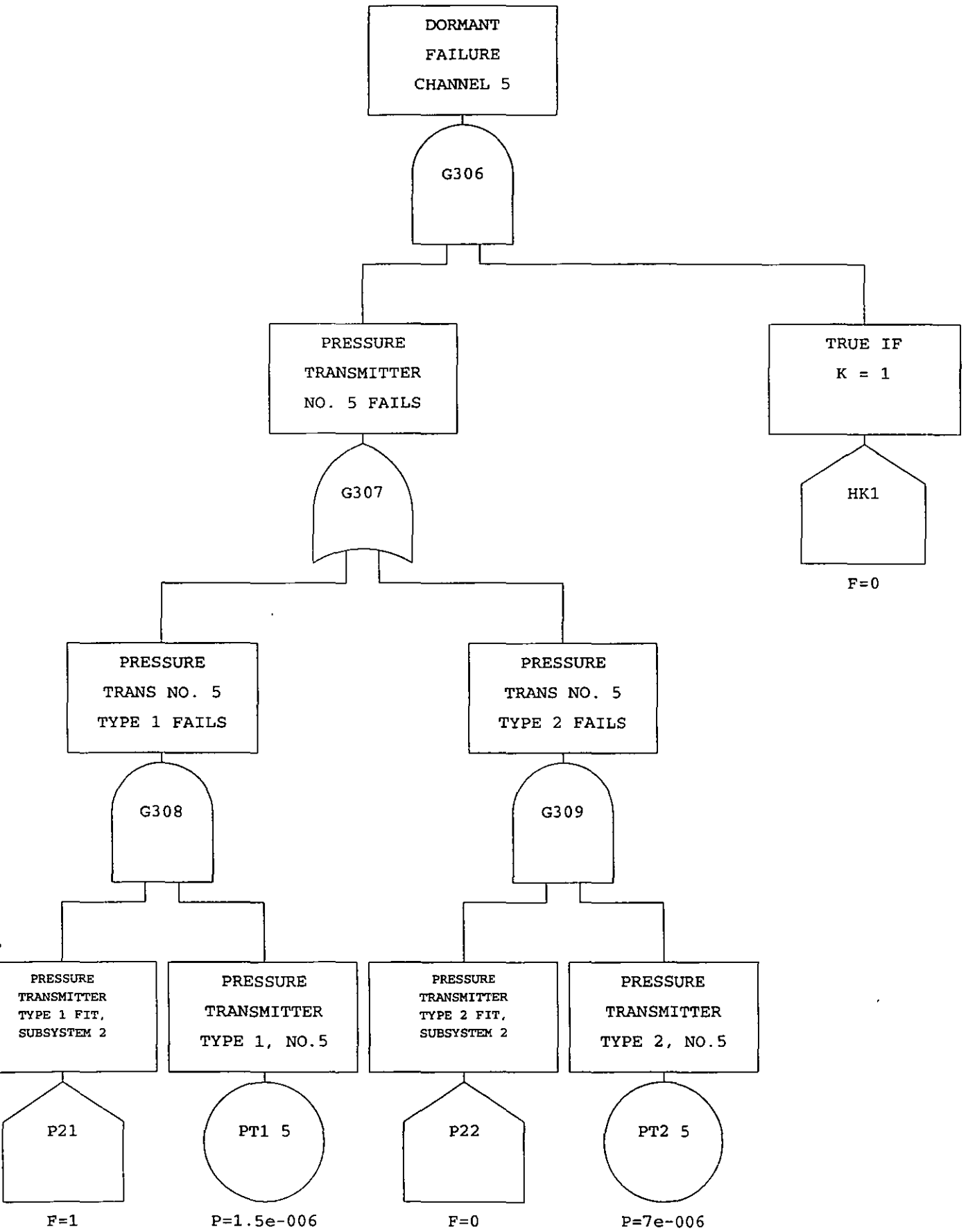


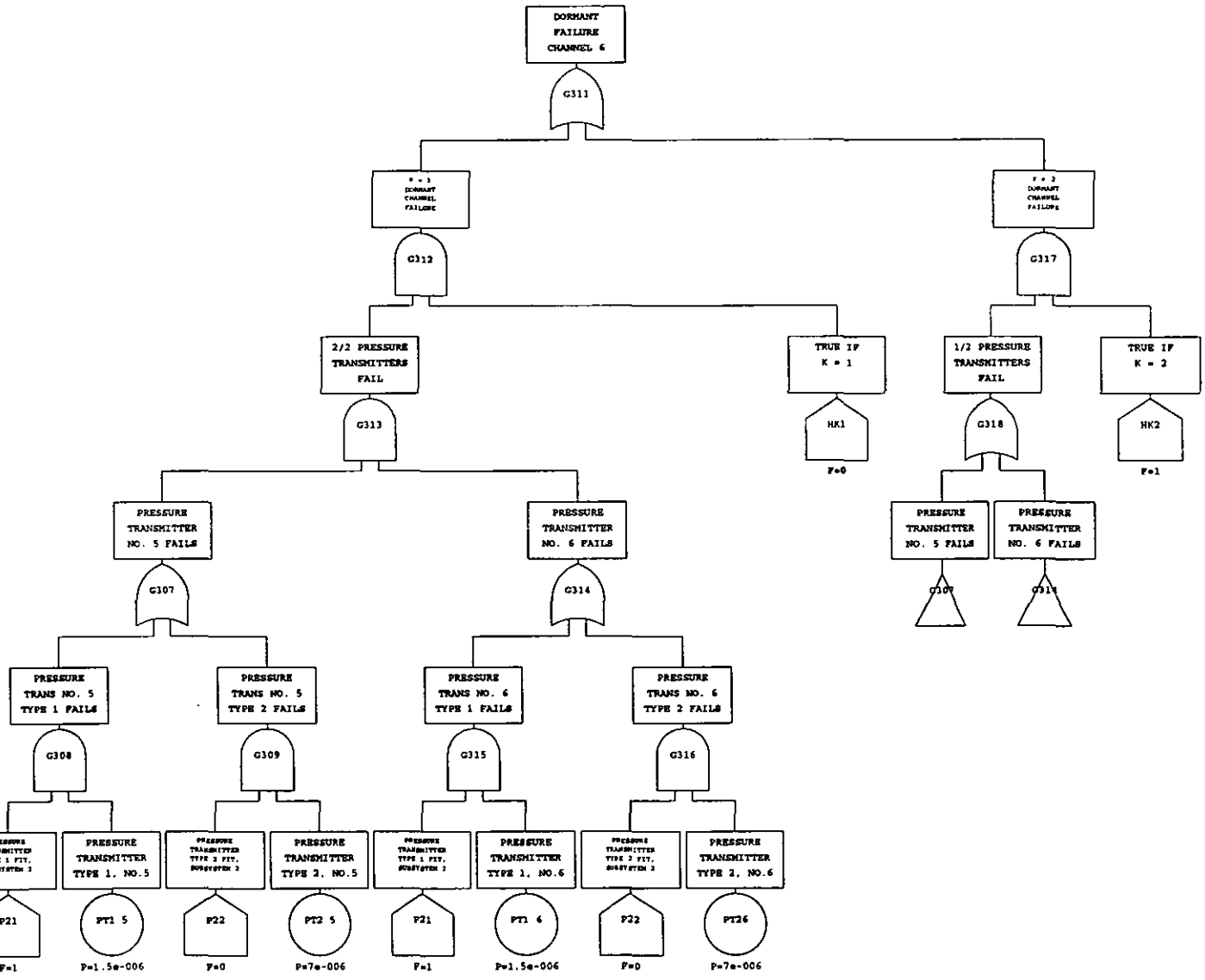


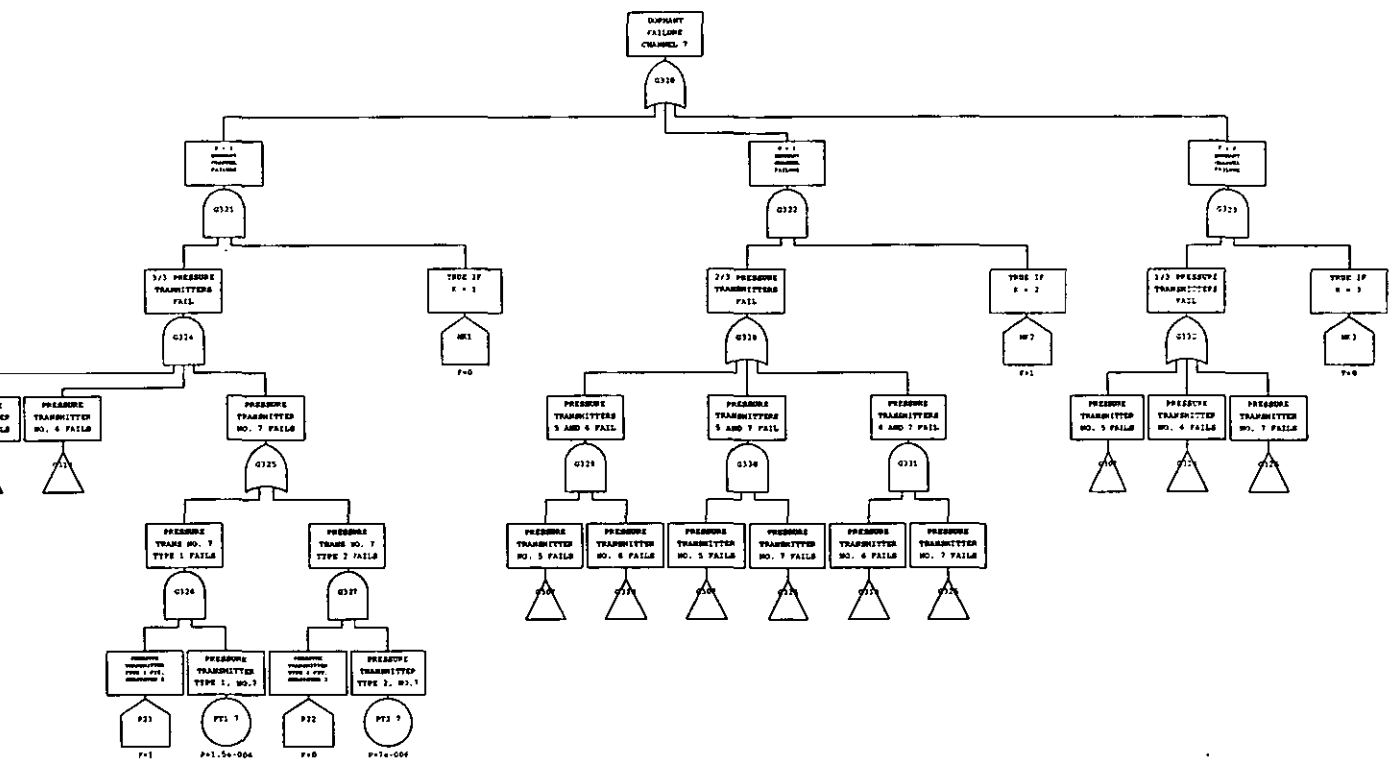


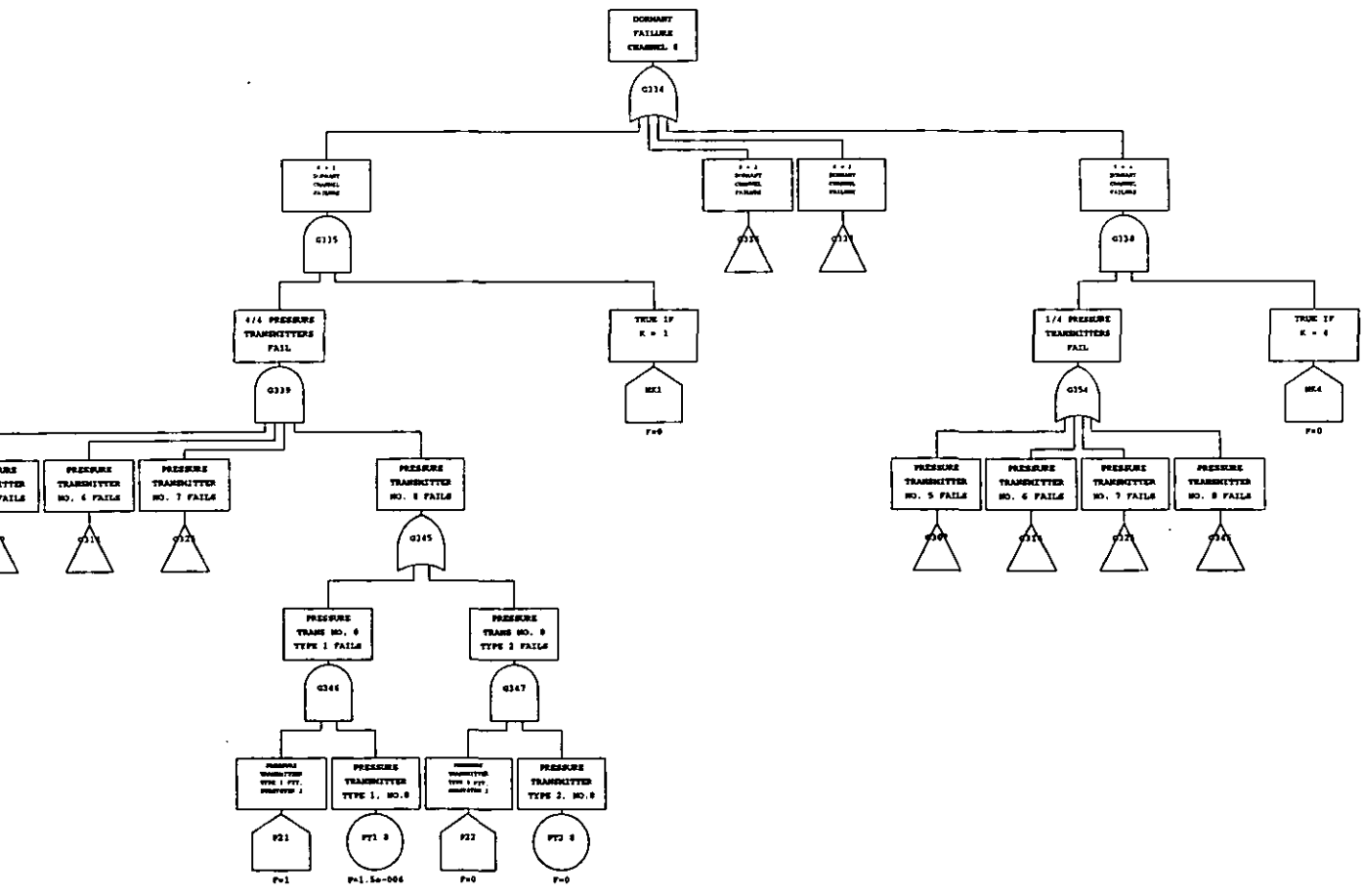


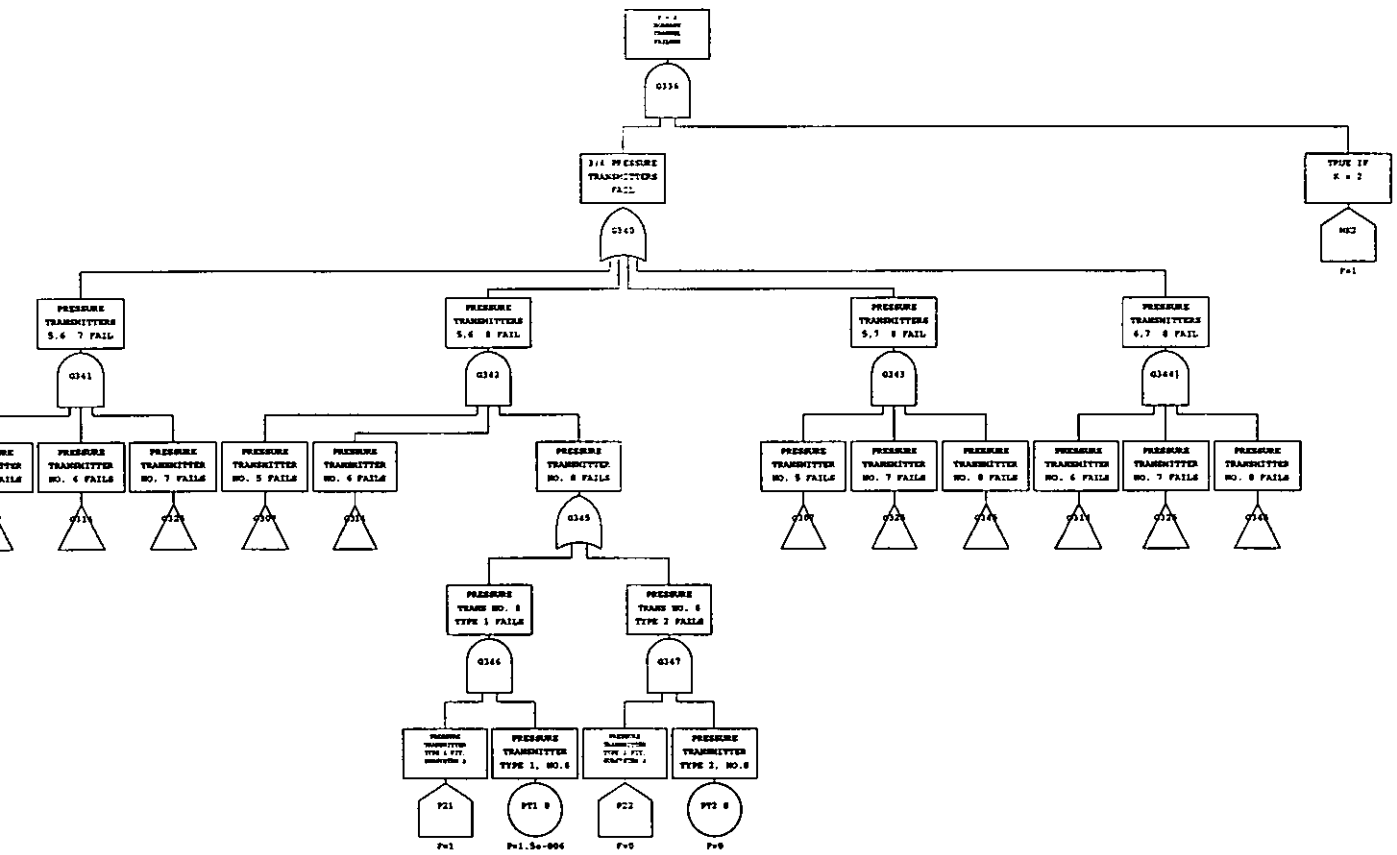


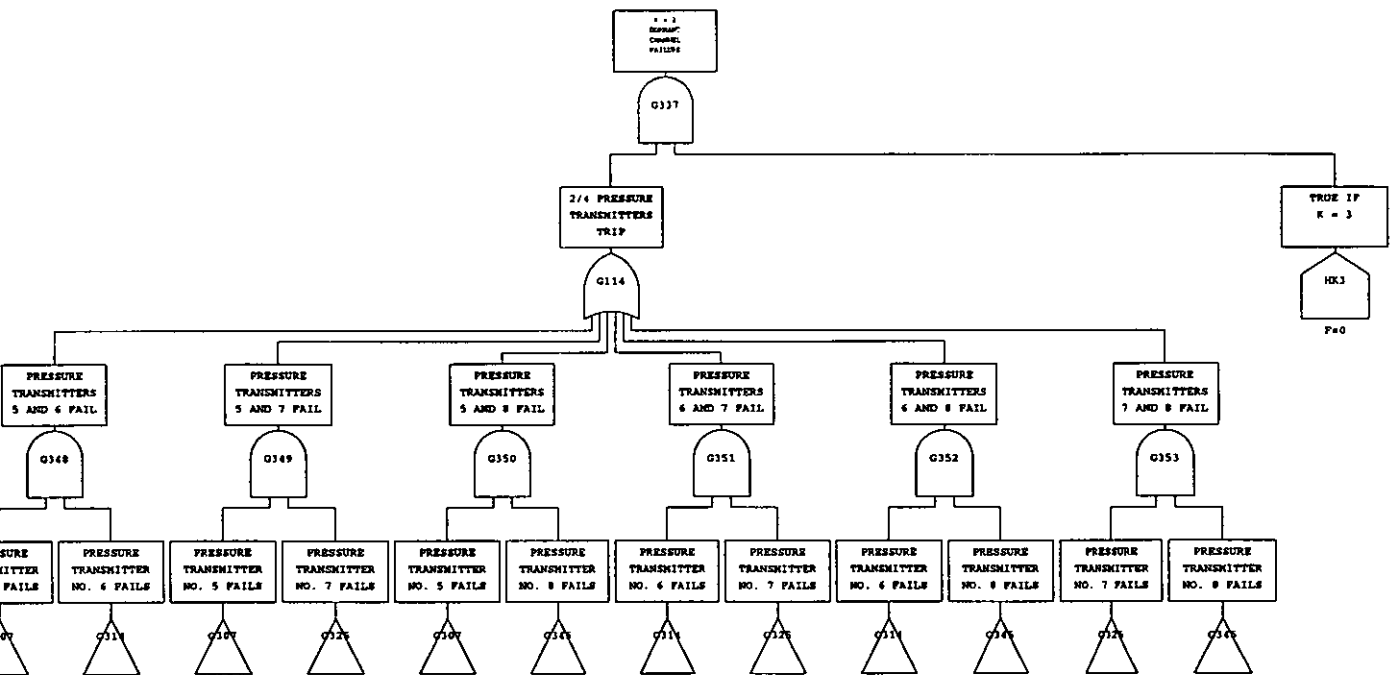


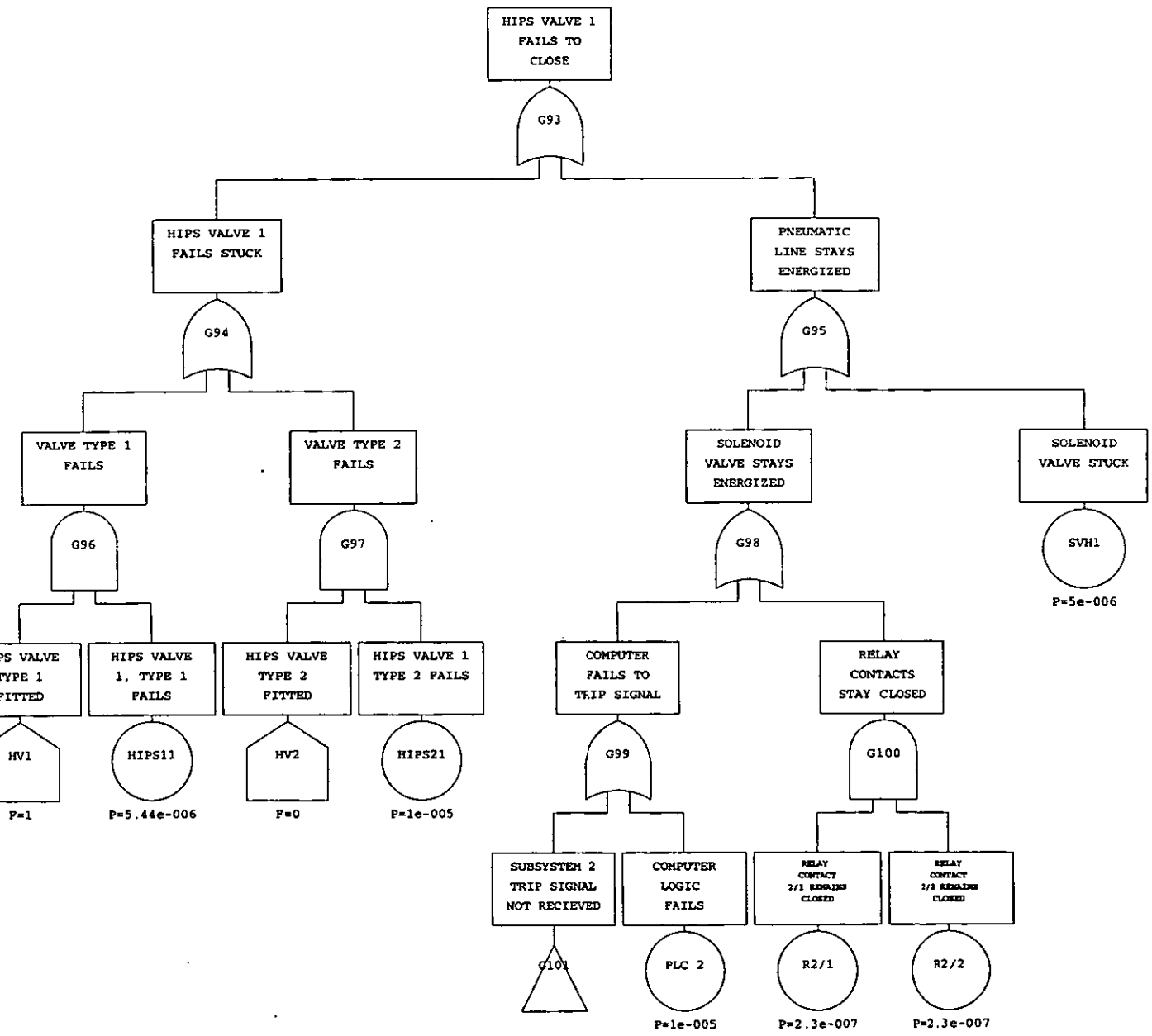






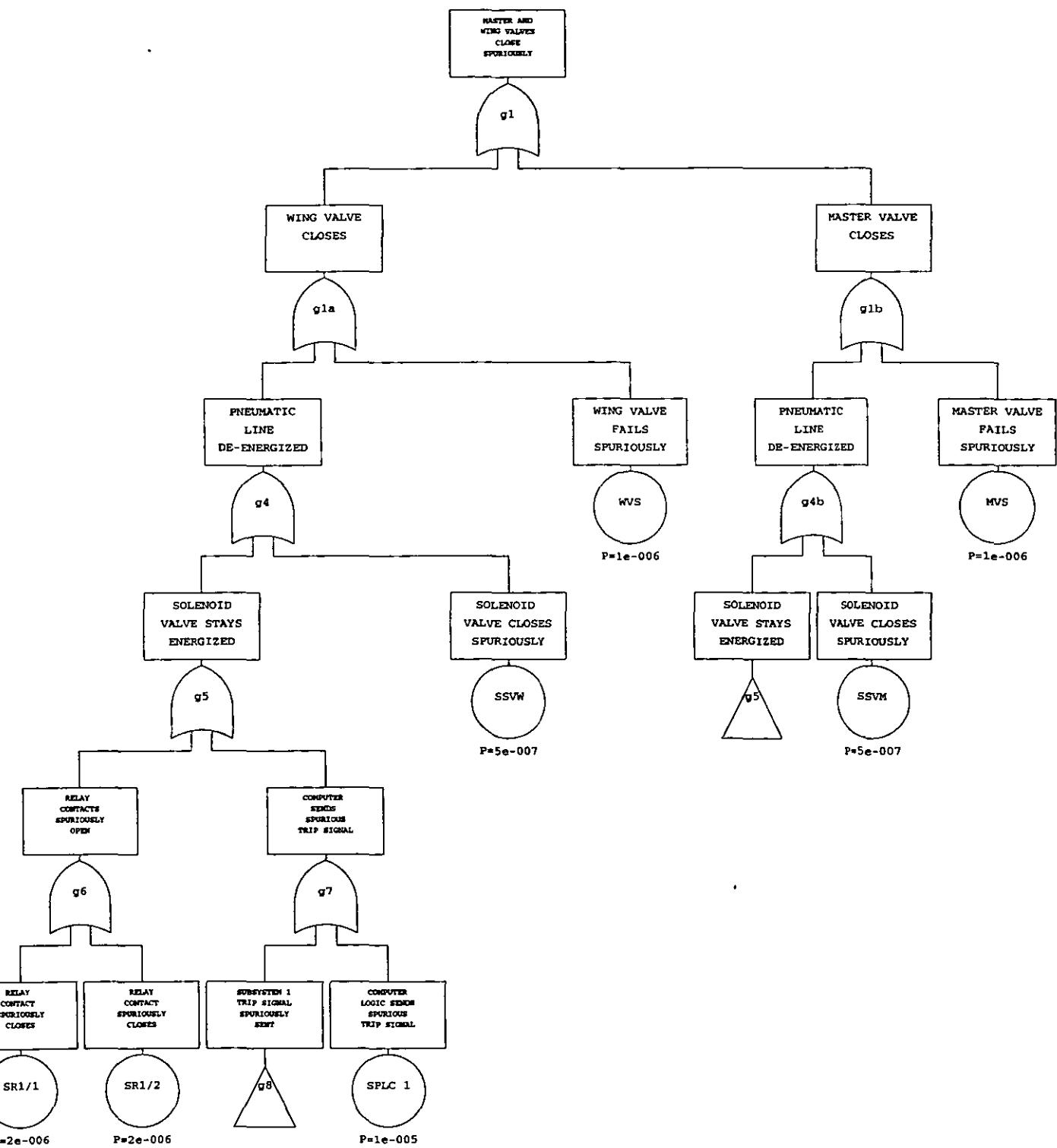


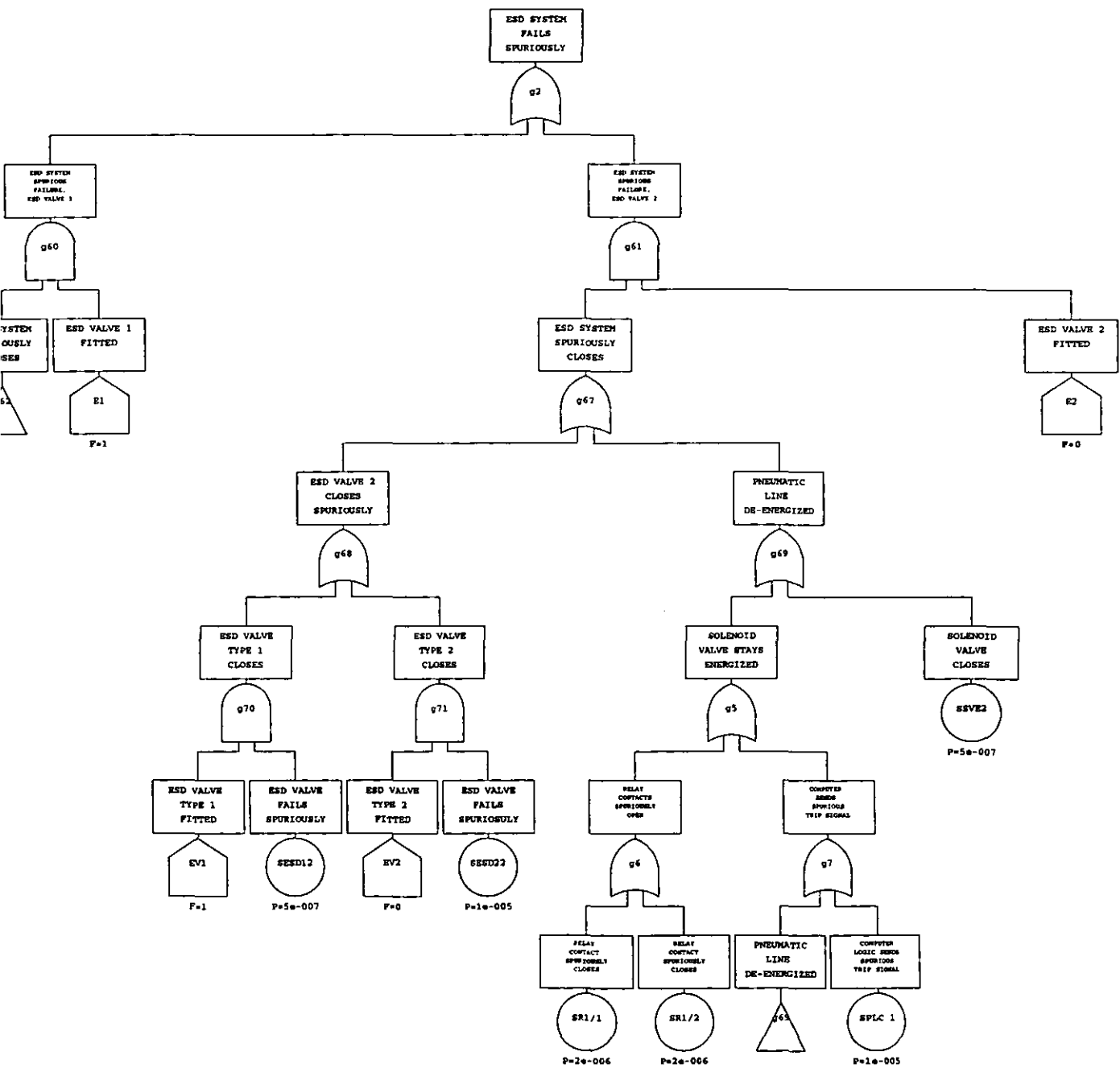


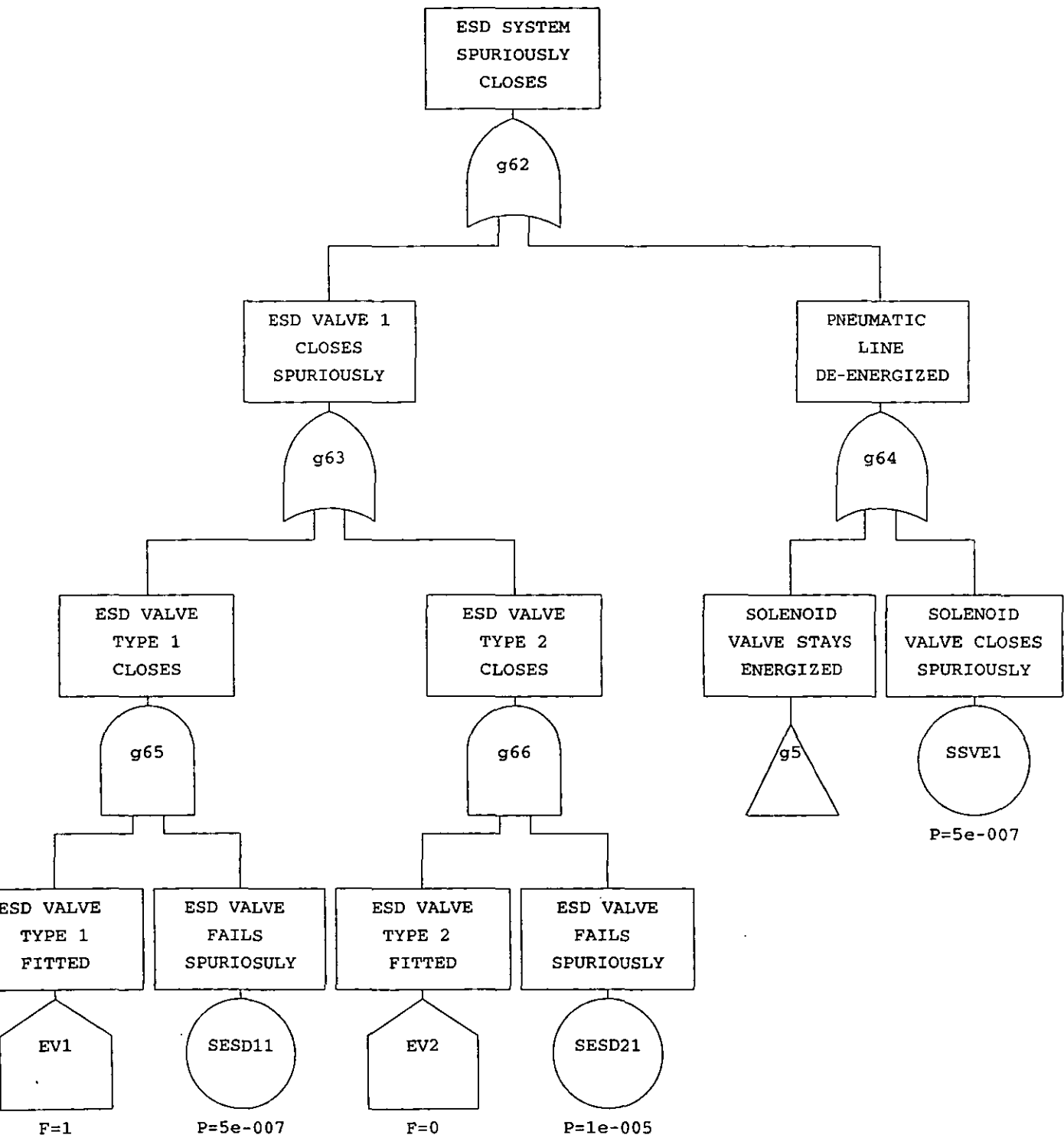


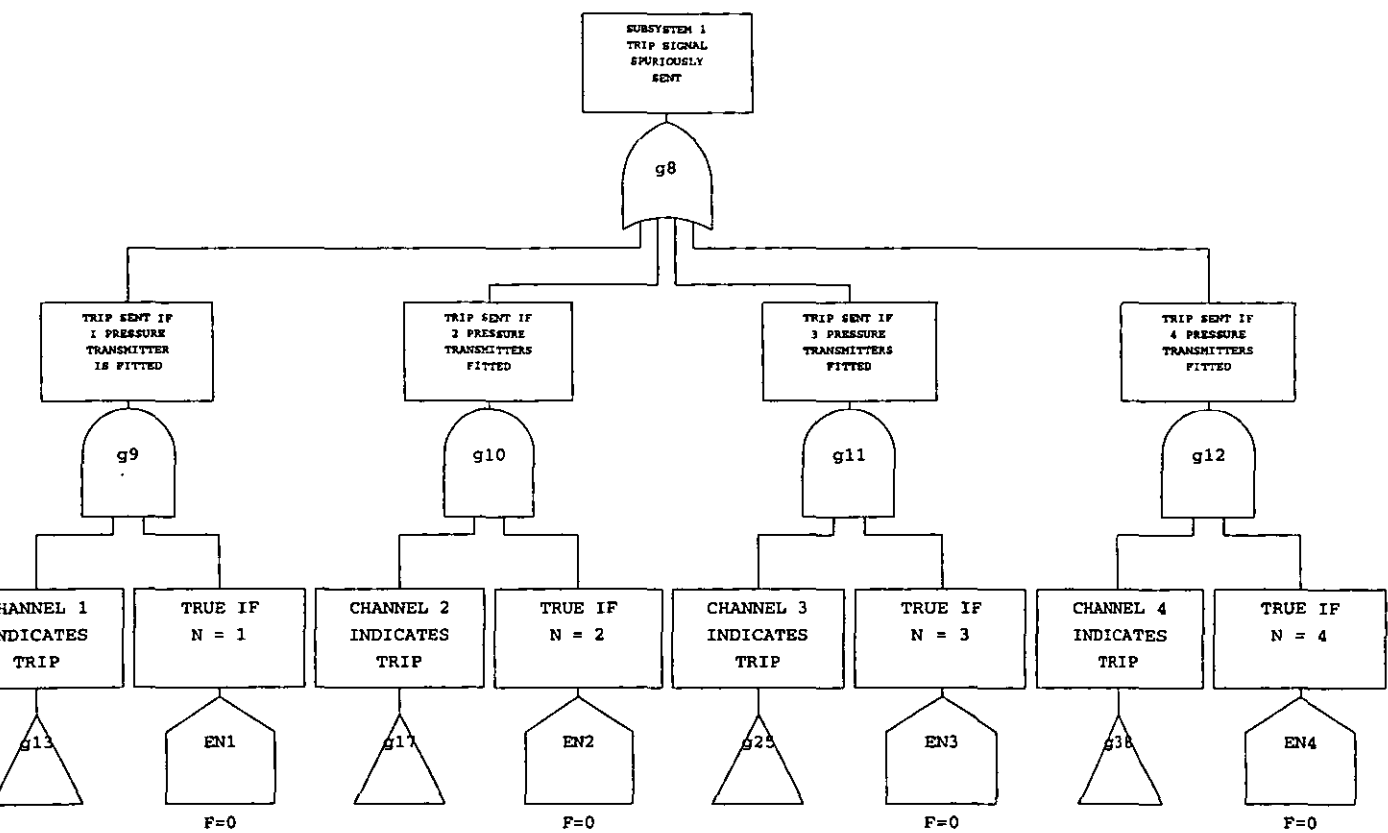
APPENDIX IV

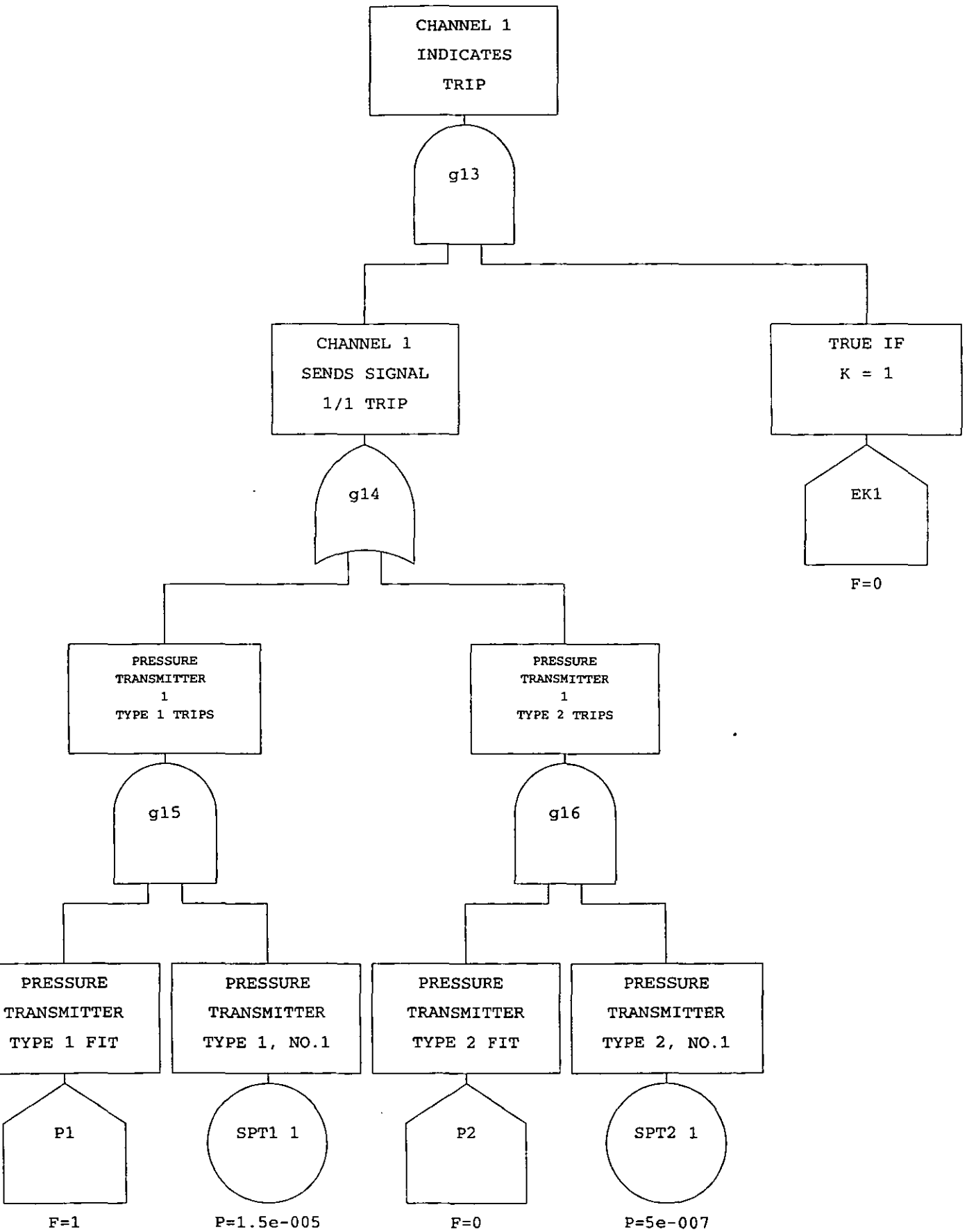
**FAULT TREE STRUCTURE FOR THE FOR THE SPURIOUS TRIP
FAILURE MODE OF THE HIGH-INTEGRITY PROTECTION SYSTEM**

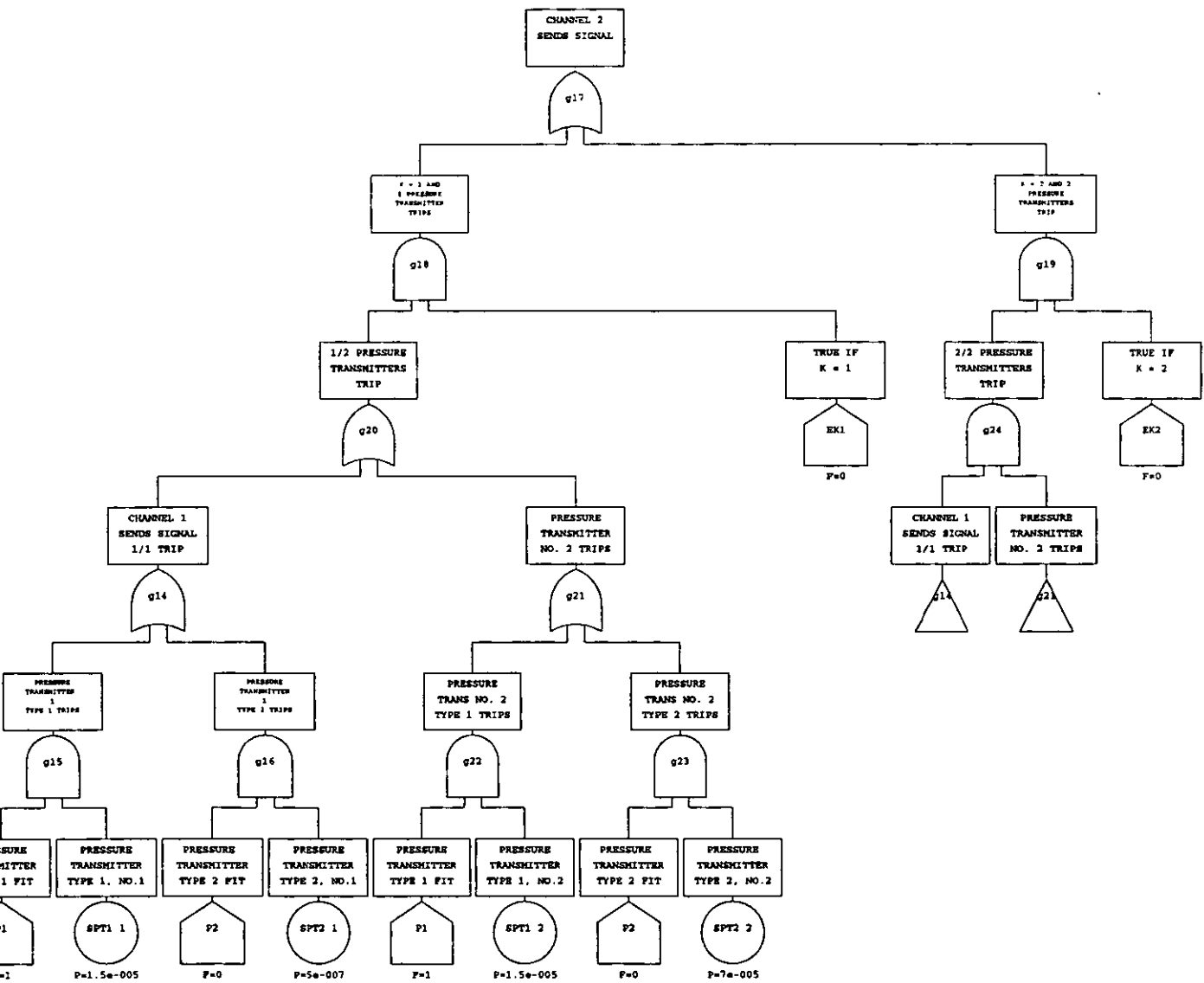


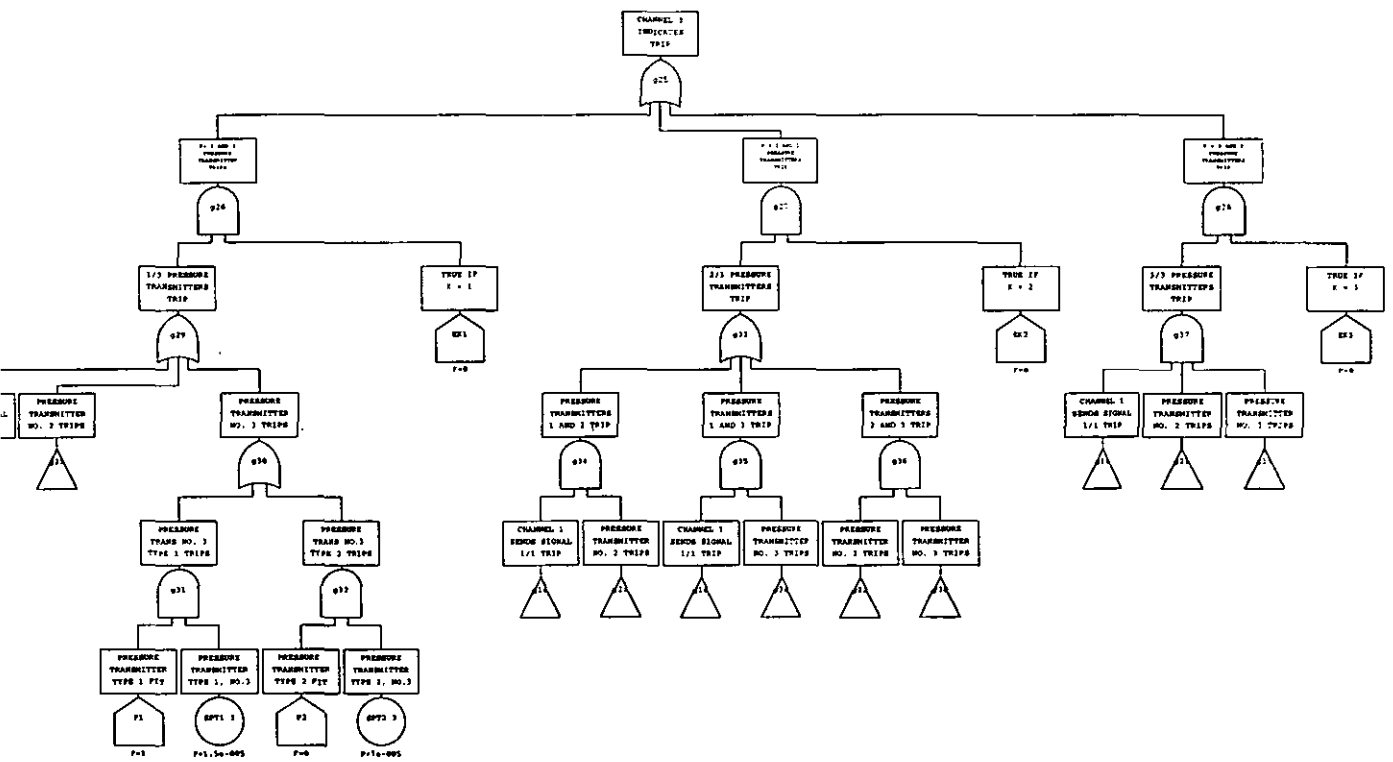


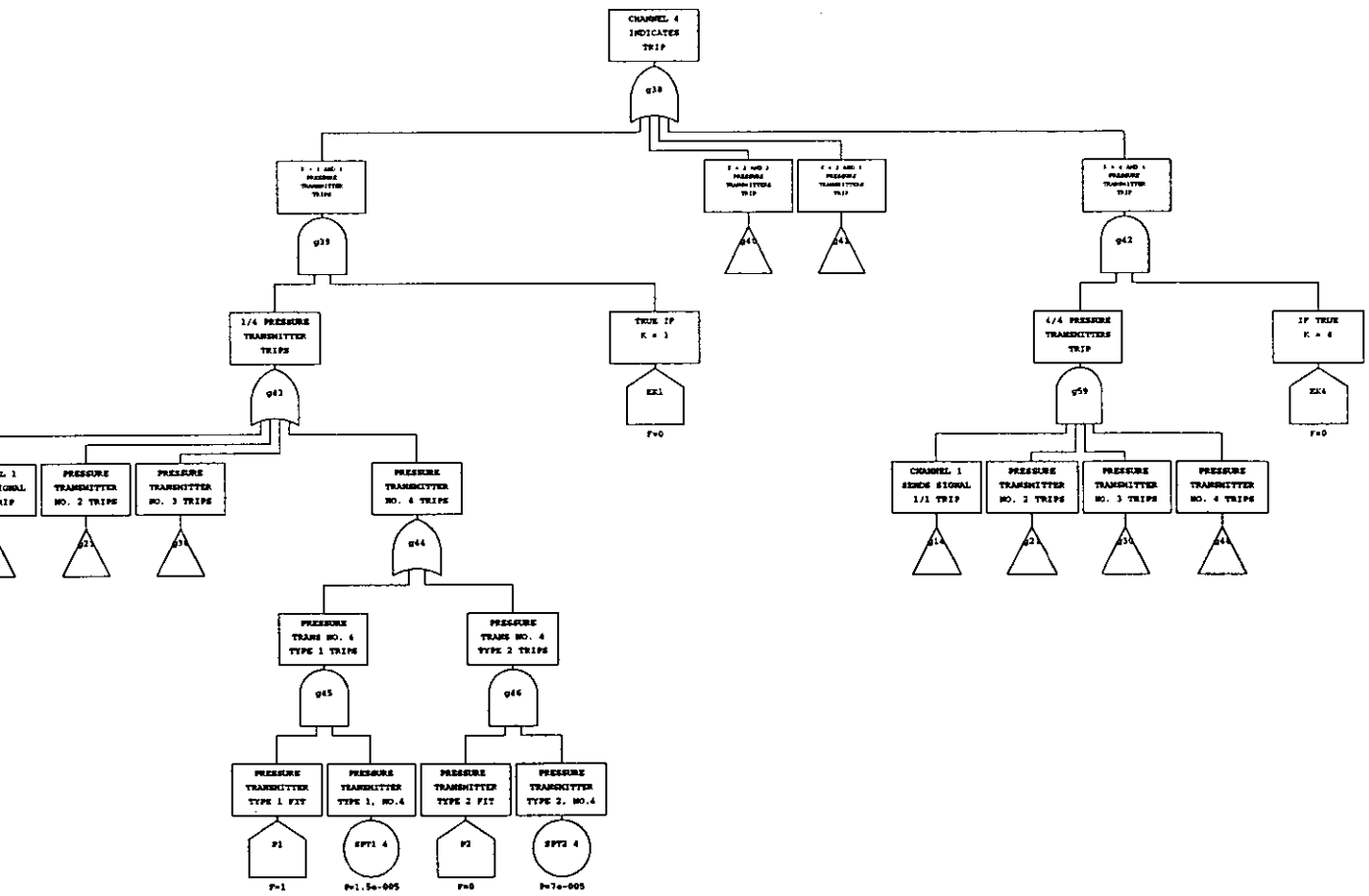


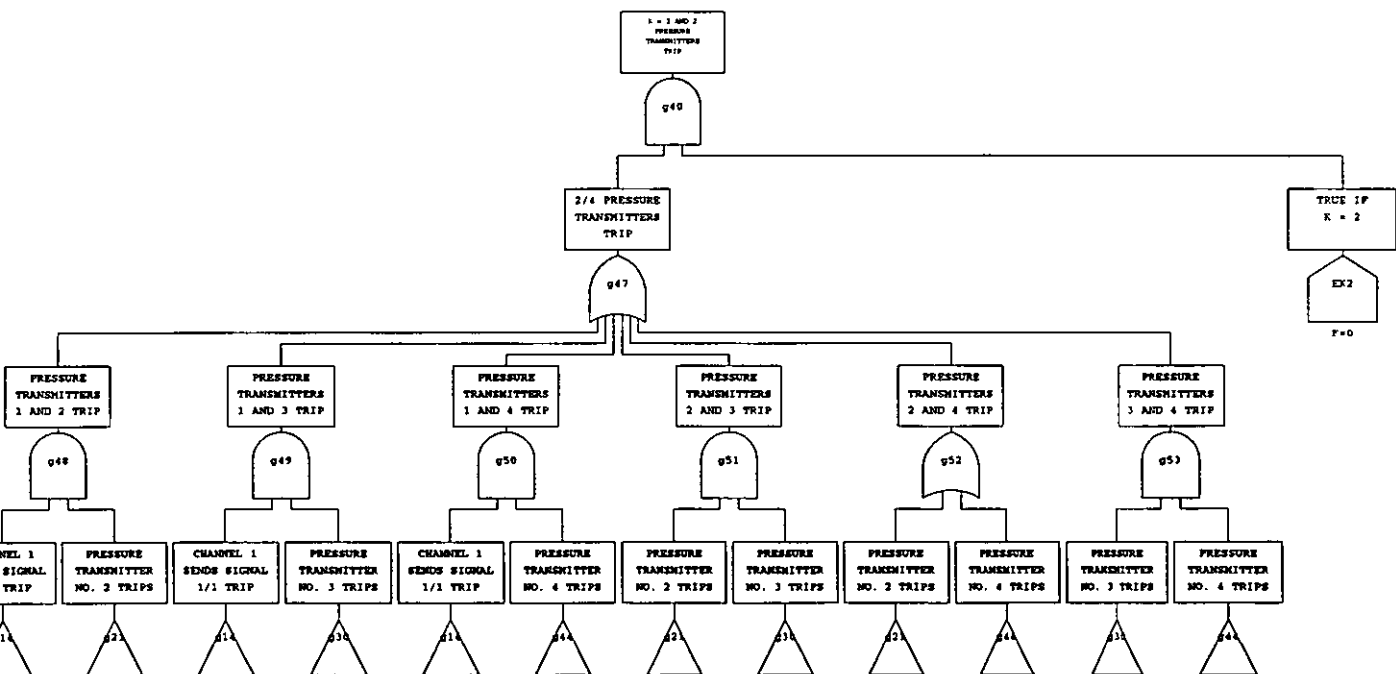


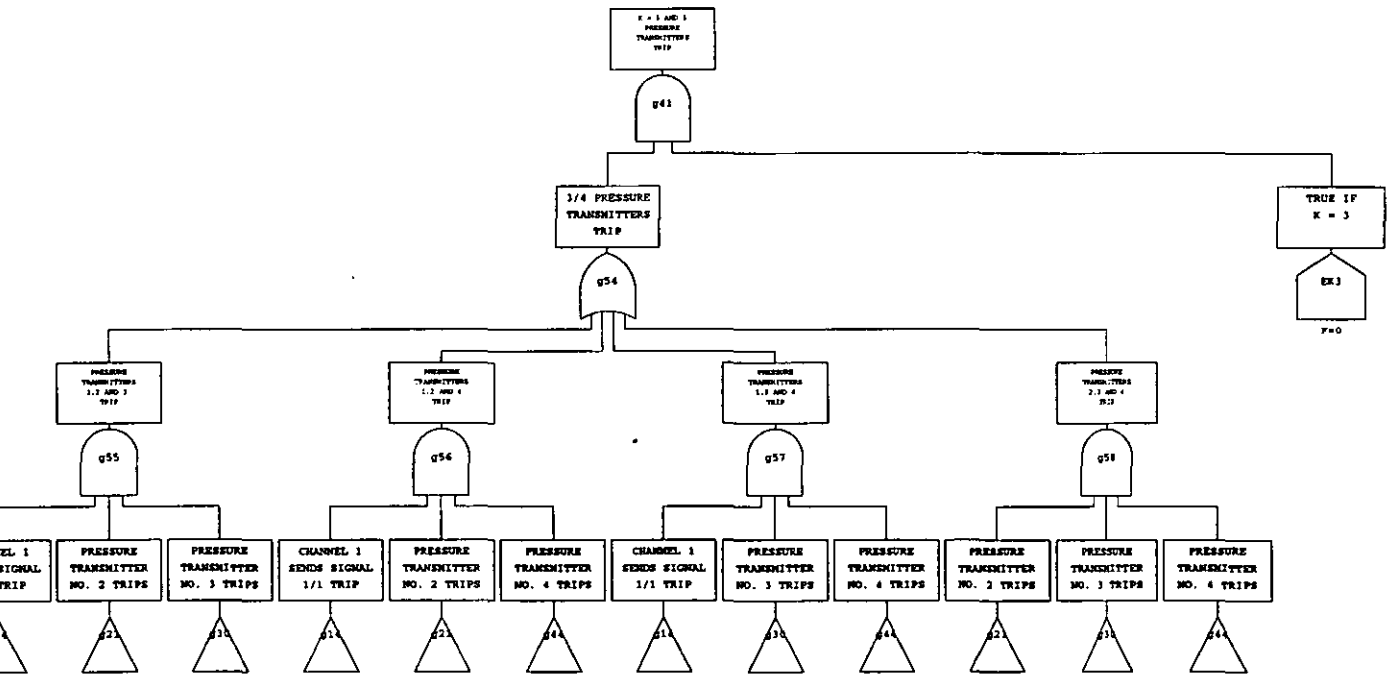


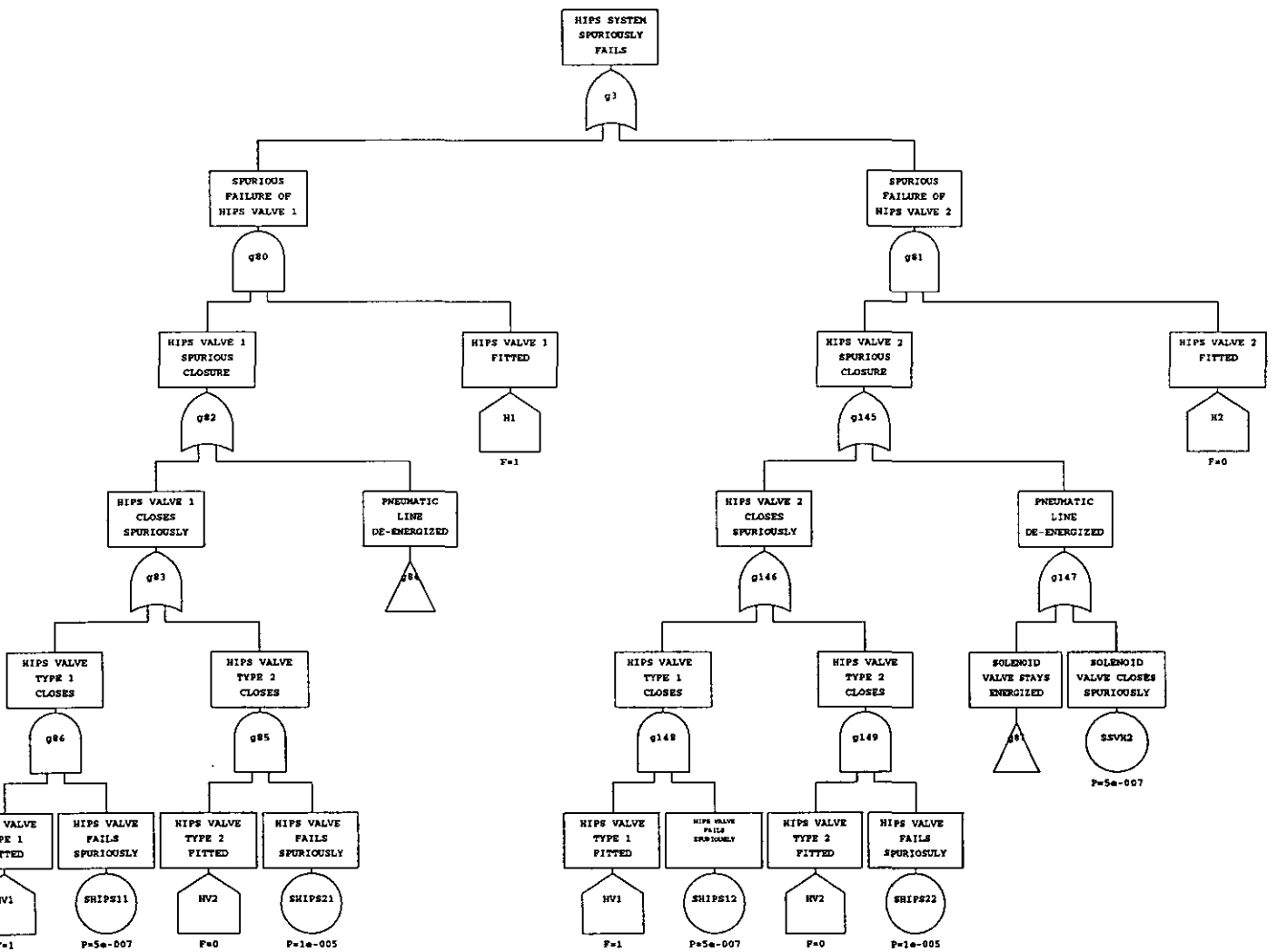


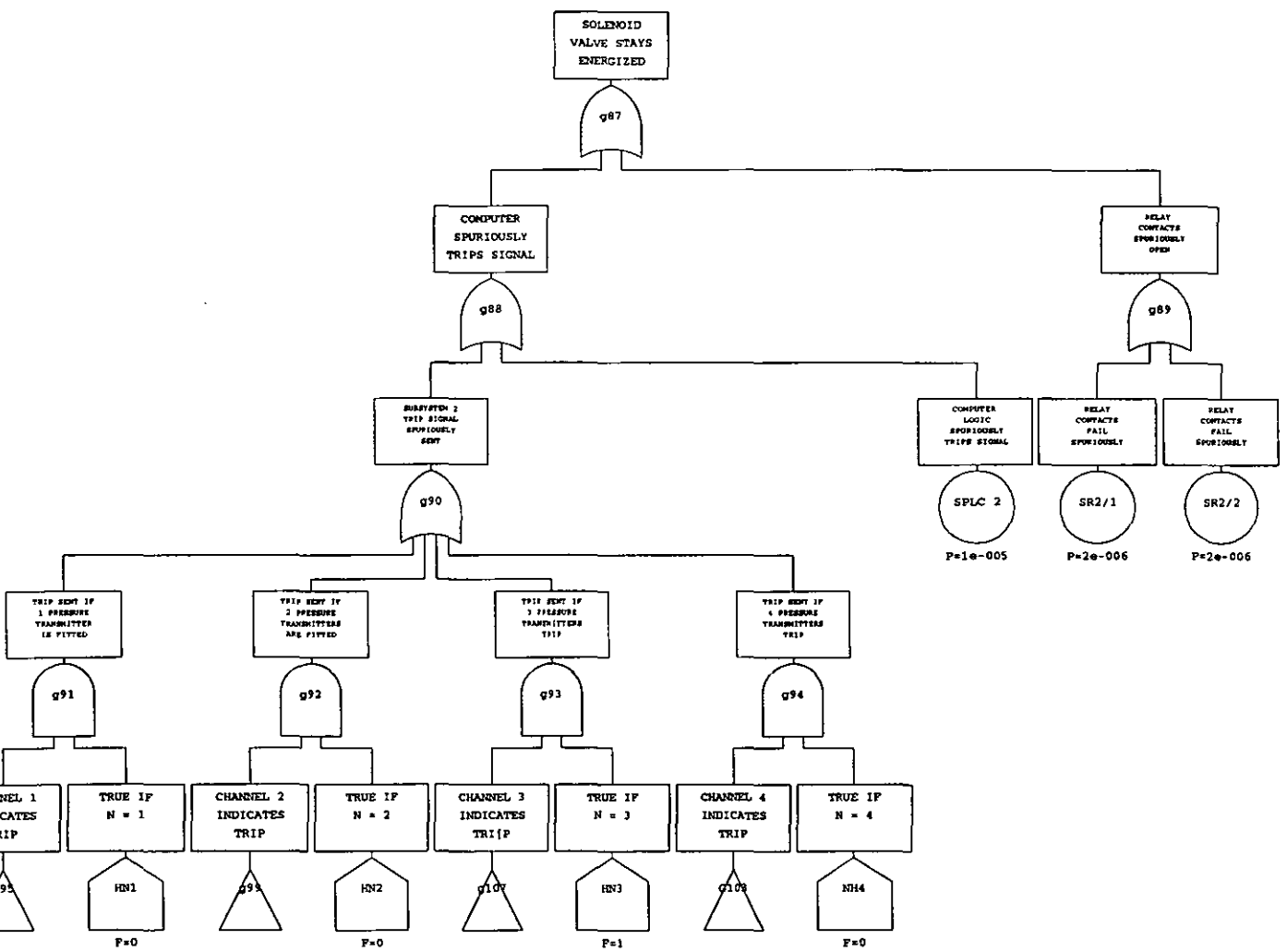


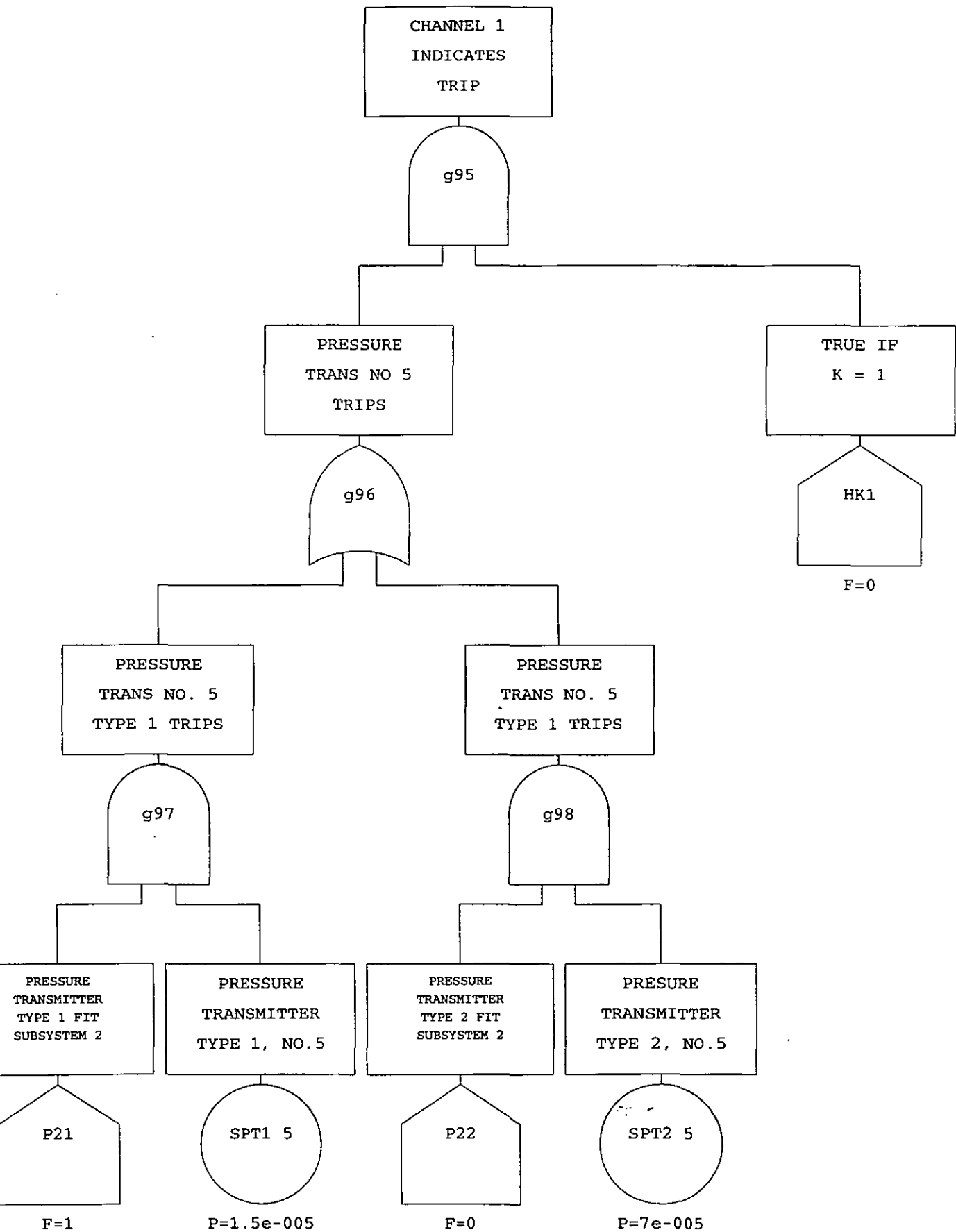












CHANNEL 1
INDICATES
TRIP

g95

PRESSURE
TRANS NO 5
TRIPS

TRUE IF
K = 1

g96

HK1

F=0

PRESSURE
TRANS NO. 5
TYPE 1 TRIPS

PRESSURE
TRANS NO. 5
TYPE 1 TRIPS

g97

g98

PRESSURE
TRANSMITTER
TYPE 1 FIT
SUBSYSTEM 2

PRESSURE
TRANSMITTER
TYPE 1, NO.5

PRESSURE
TRANSMITTER
TYPE 2 FIT
SUBSYSTEM 2

PRESURE
TRANSMITTER
TYPE 2, NO.5

P21

SPT1 5

P22

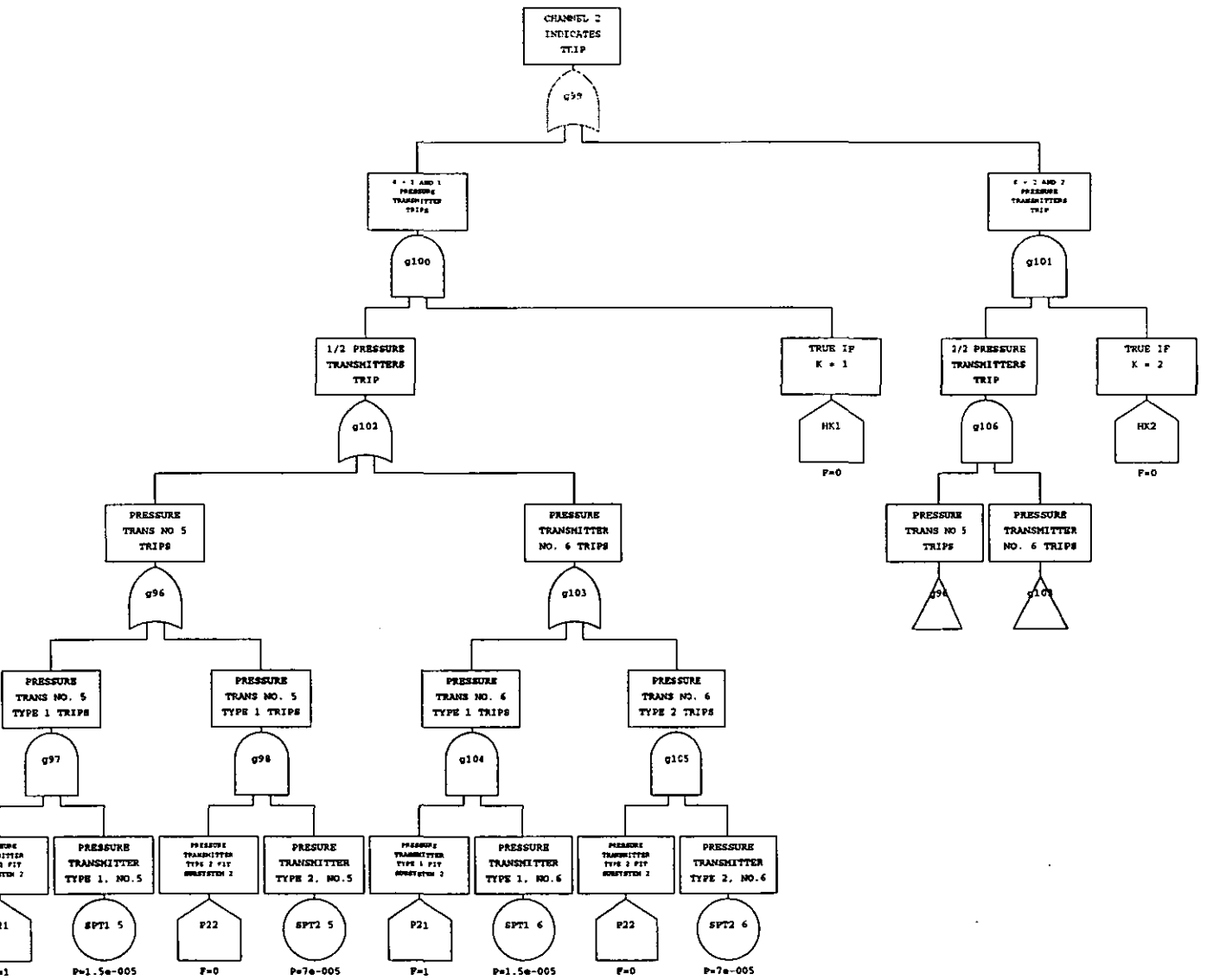
SPT2 5

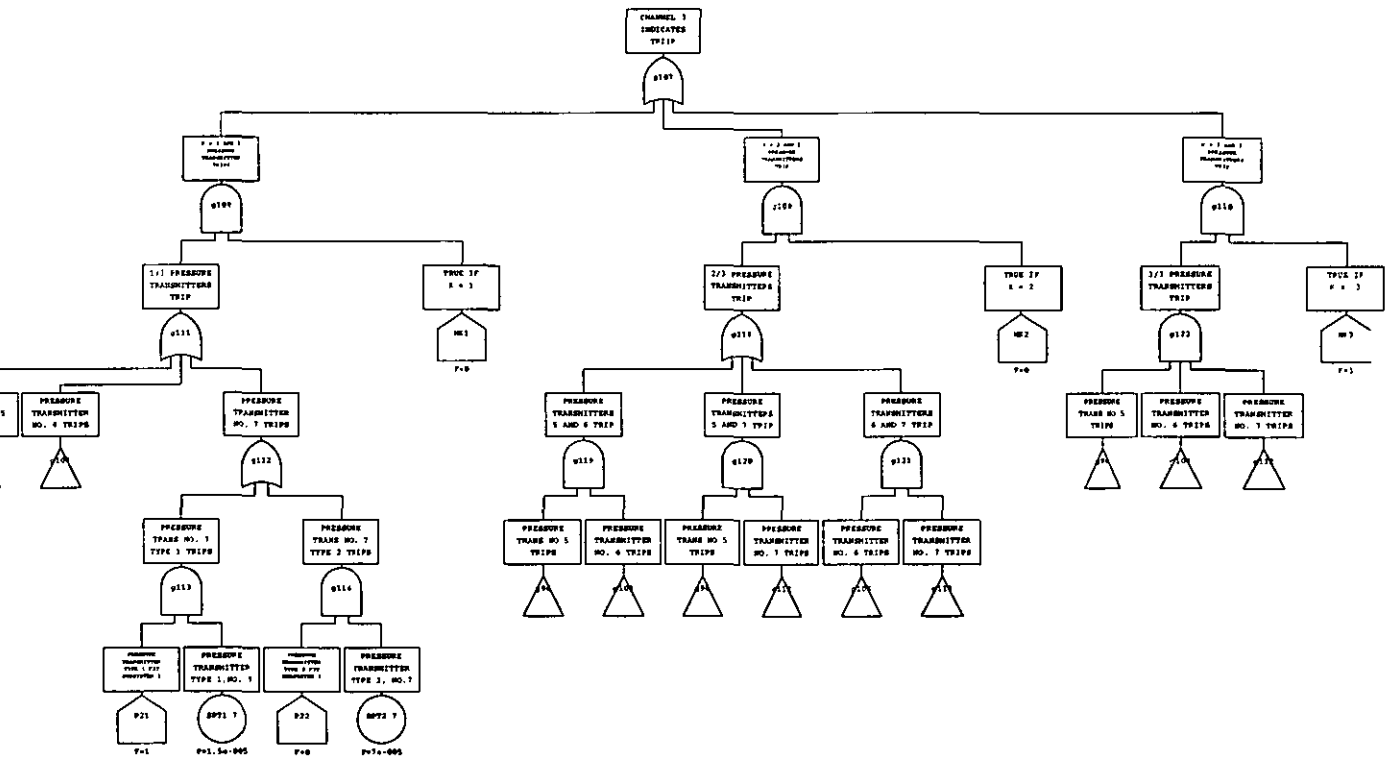
F=1

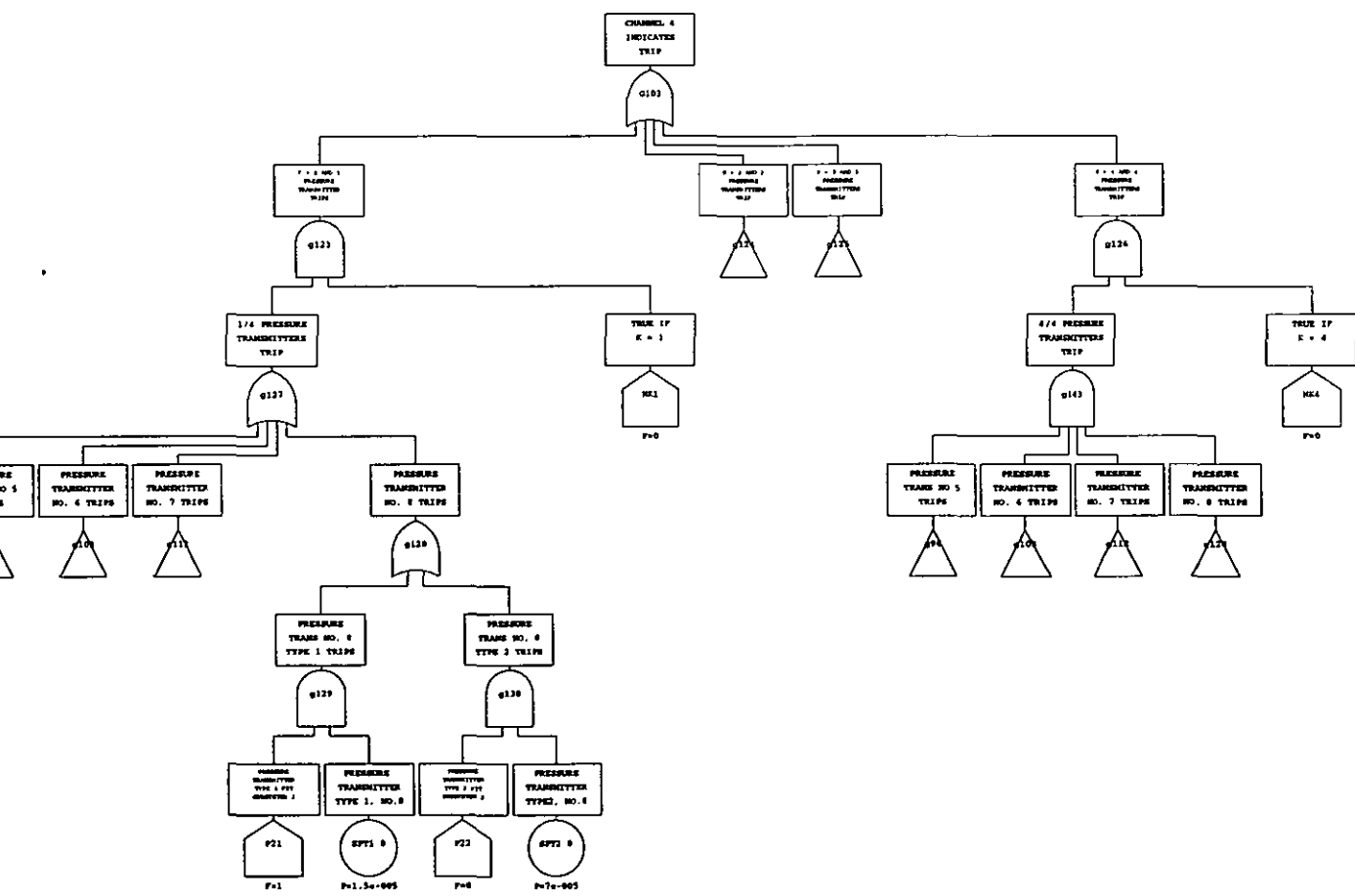
P=1.5e-005

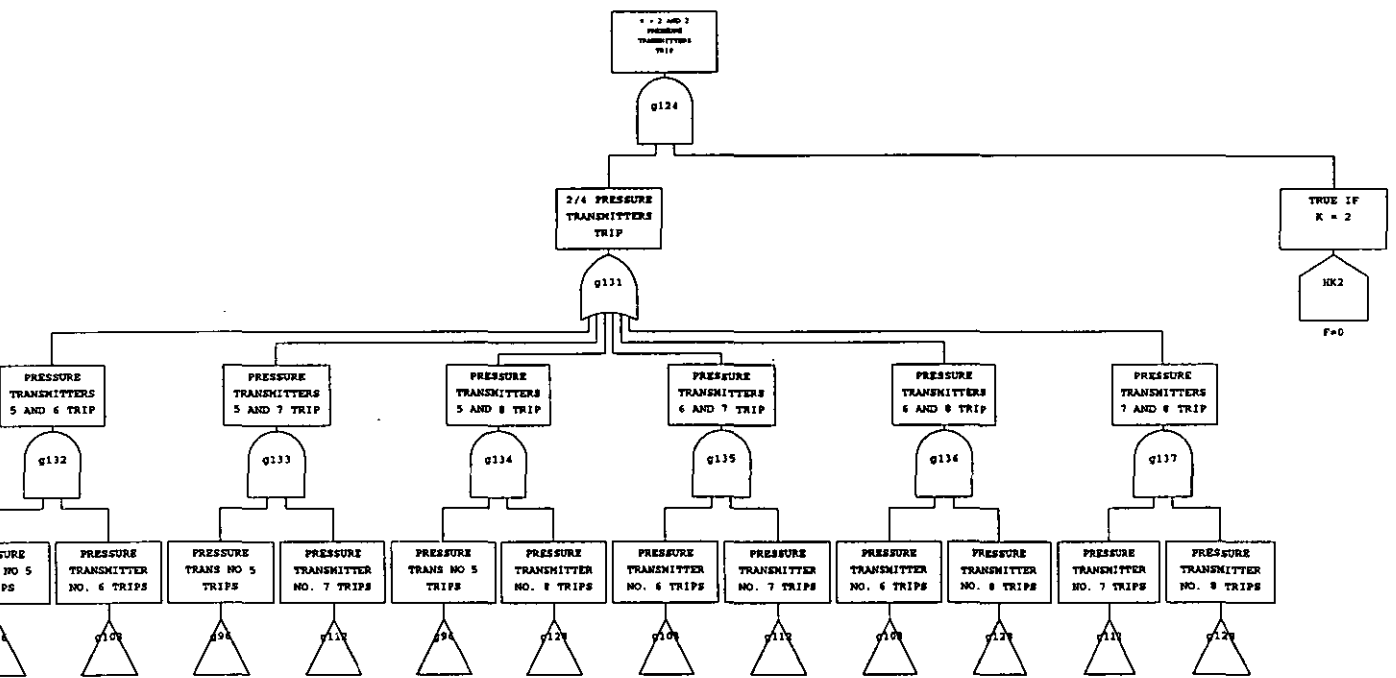
F=0

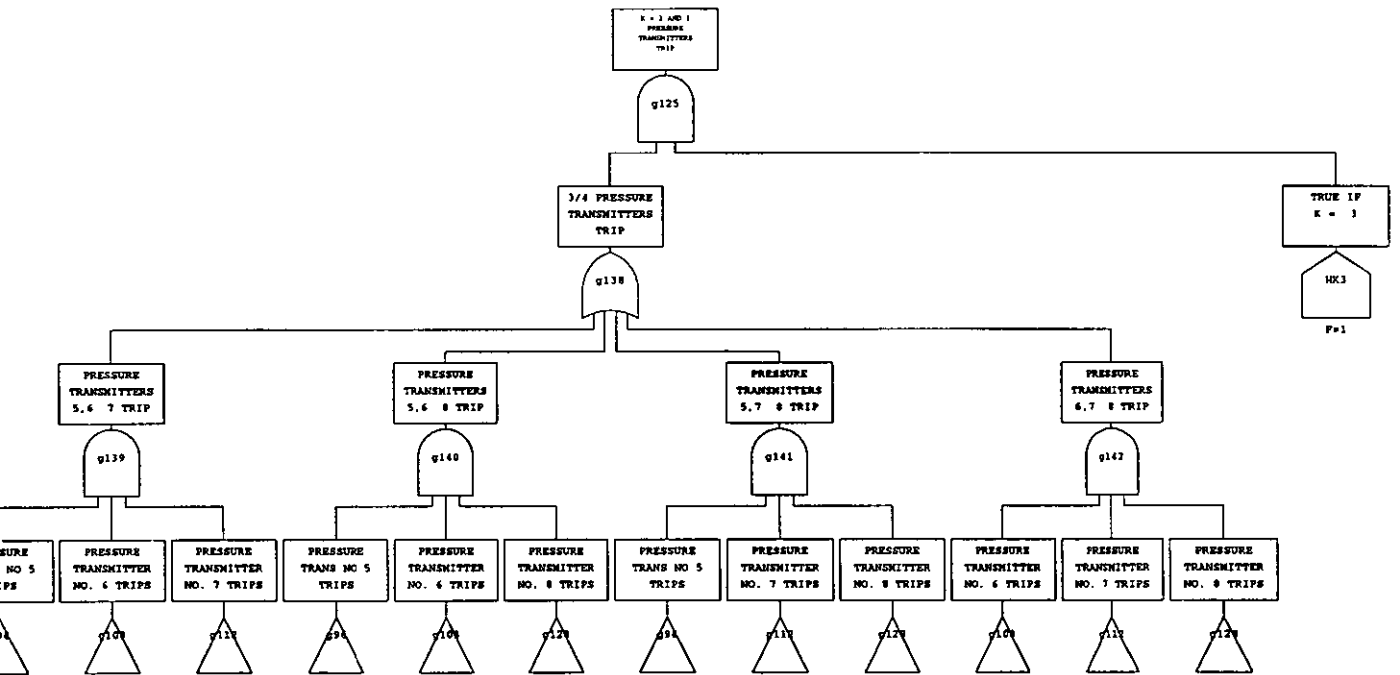
P=7e-005





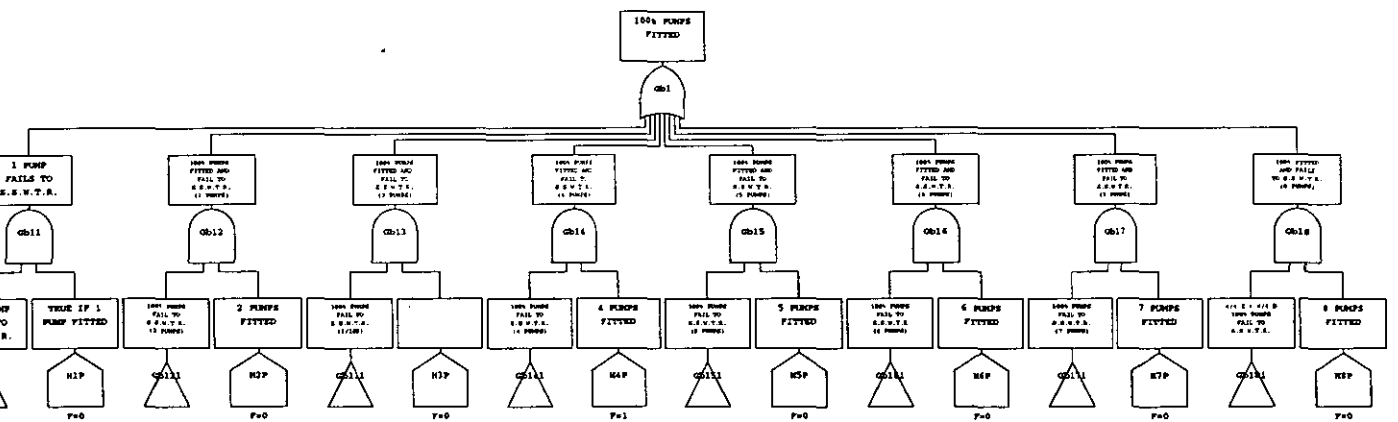


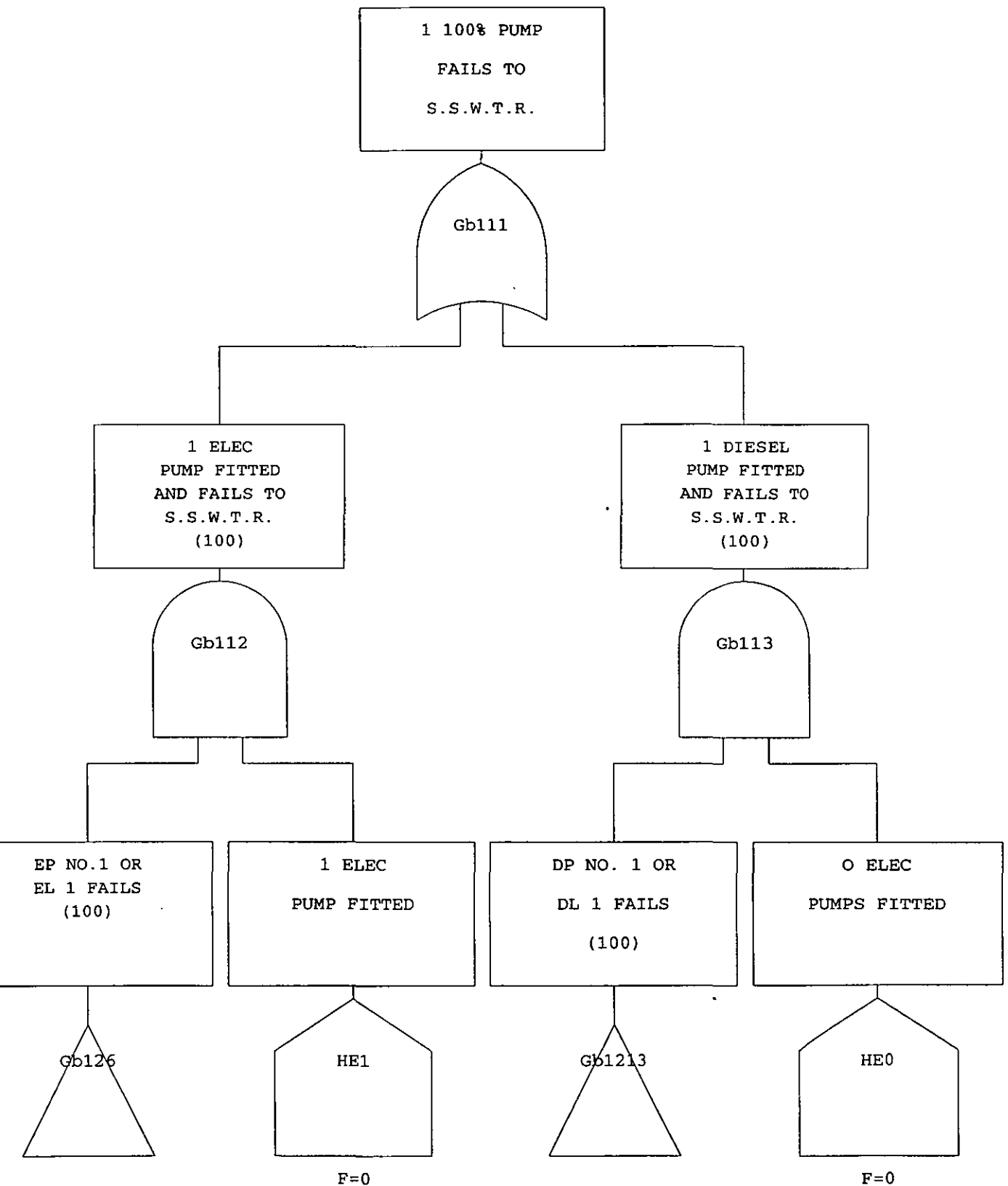


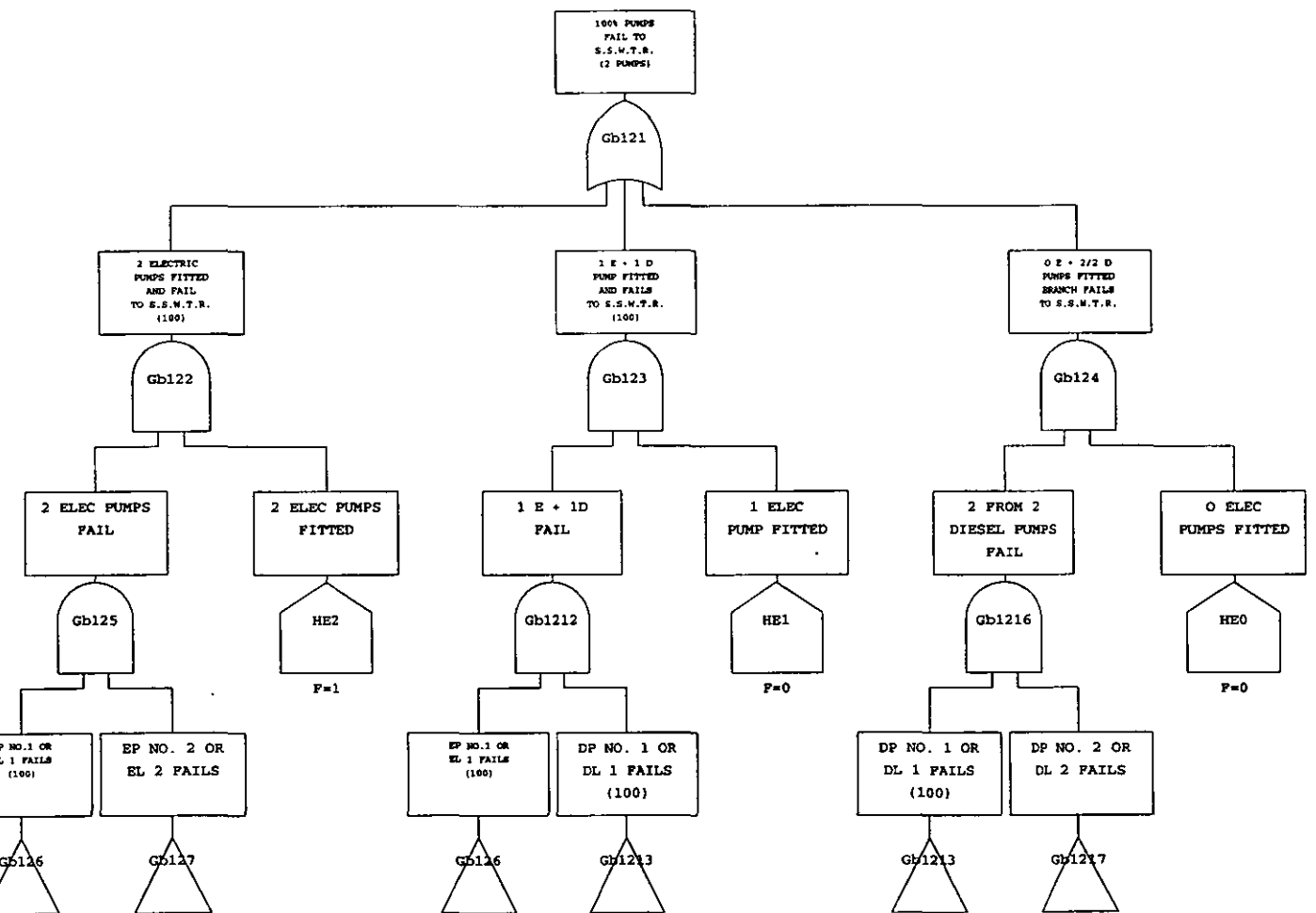


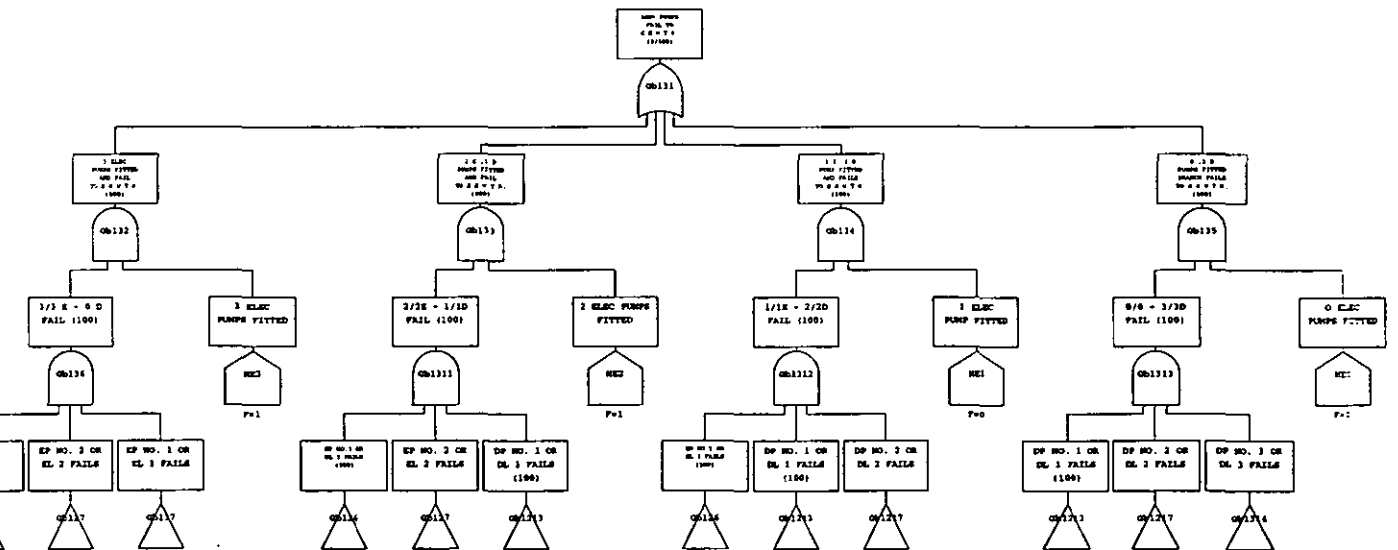
APPENDIX V

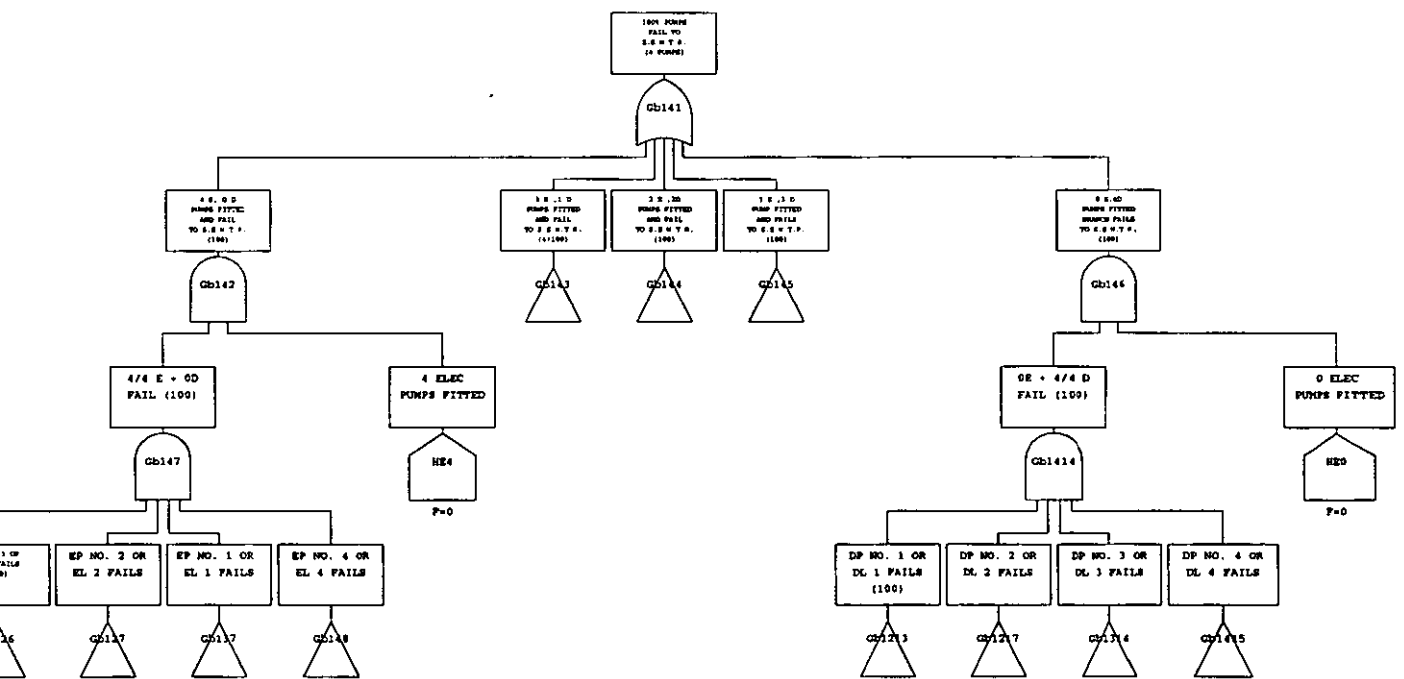
**FAULT TREE STRUCTURE FOR THE SYSTEM UNAVAILABILITY
FAILURE MODE OF THE FIREWATER DELIGE SYSTEM**

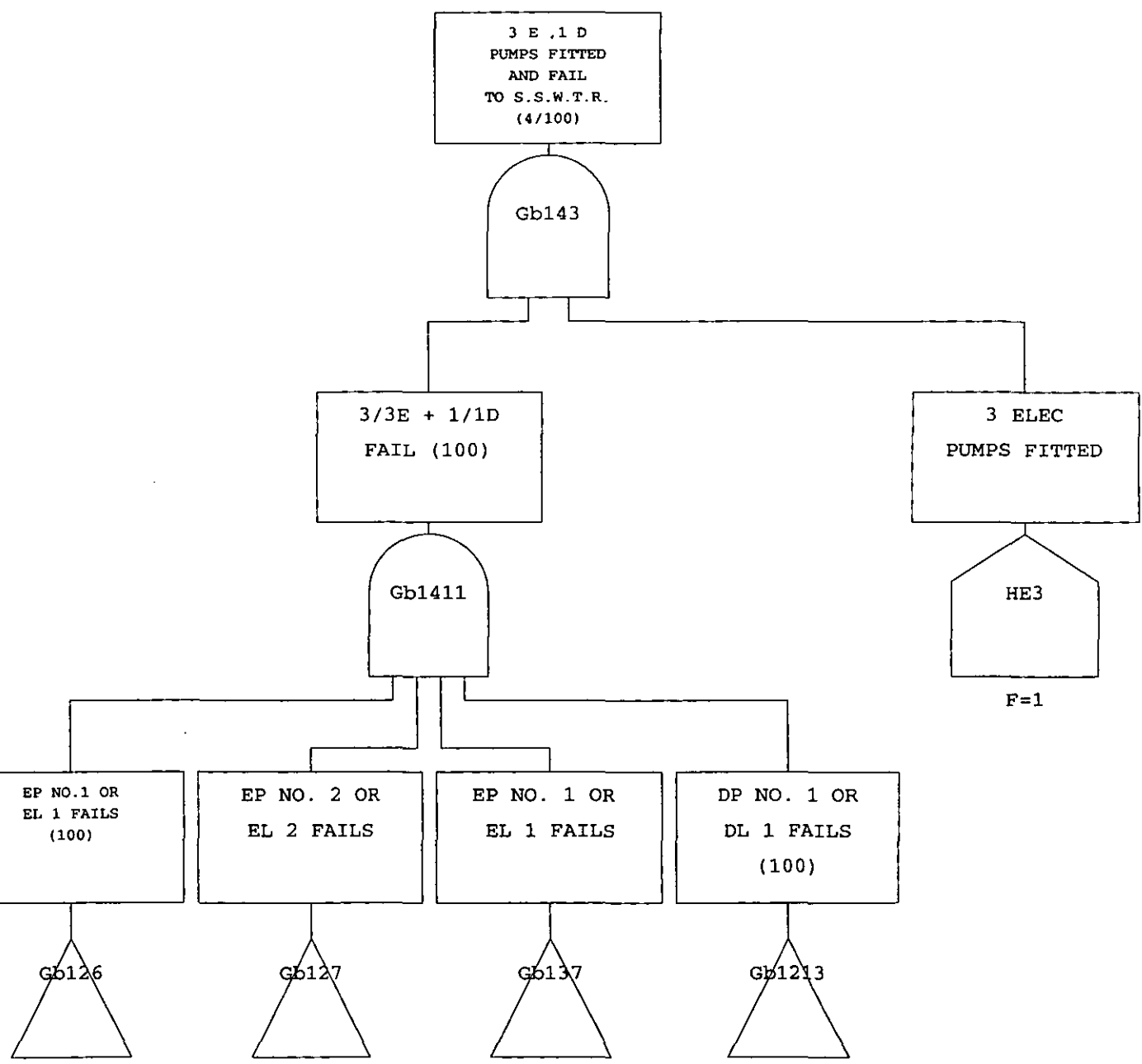












3 E, 1 D
PUMPS FITTED
AND FAIL
TO S.S.W.T.R.
(4/100)

Gb143

3/3E + 1/1D
FAIL (100)

3 ELEC
PUMPS FITTED

Gb1411

HE3

F=1

EP NO.1 OR
EL 1 FAILS
(100)

EP NO. 2 OR
EL 2 FAILS

EP NO. 1 OR
EL 1 FAILS

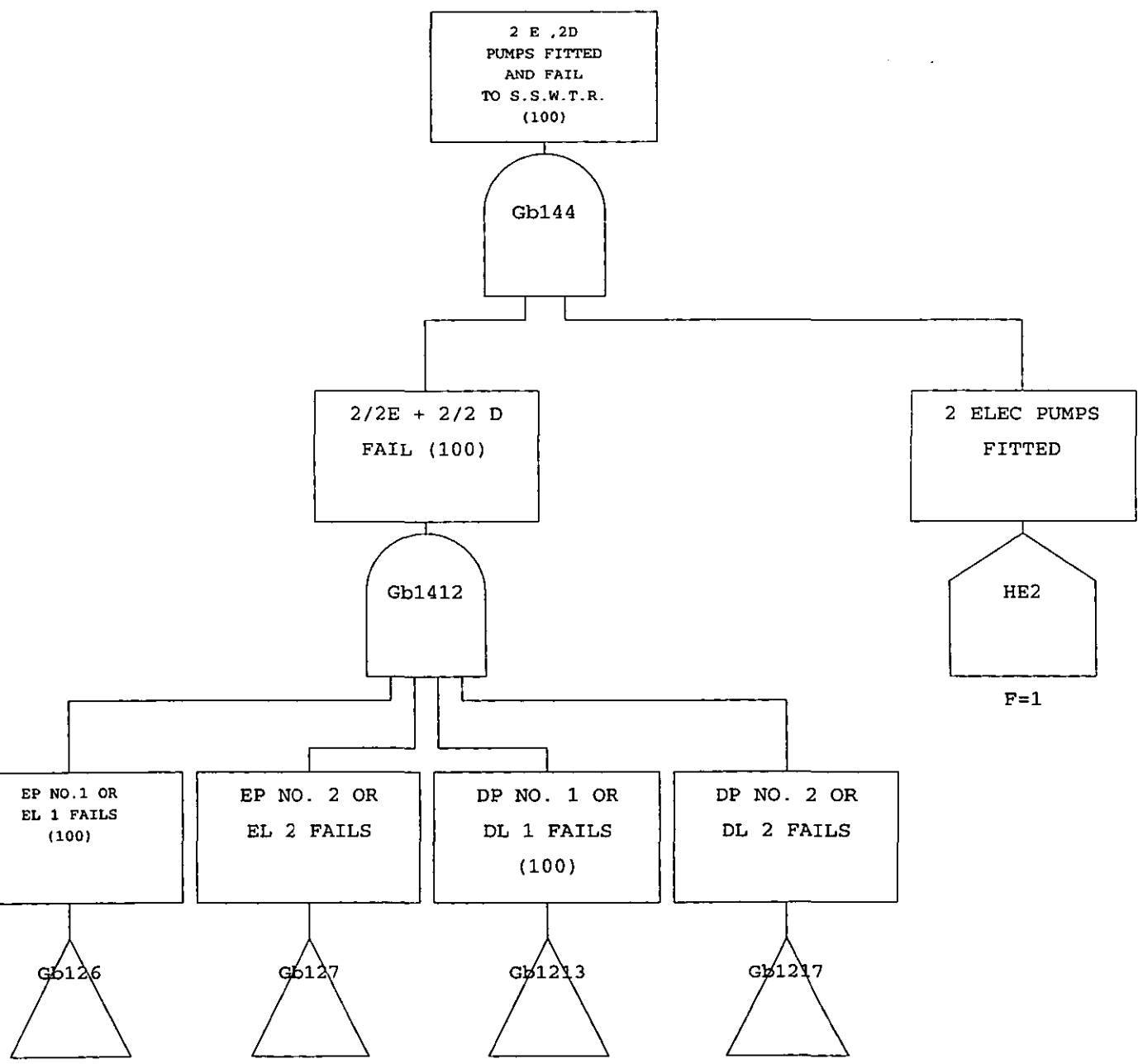
DP NO. 1 OR
DL 1 FAILS
(100)

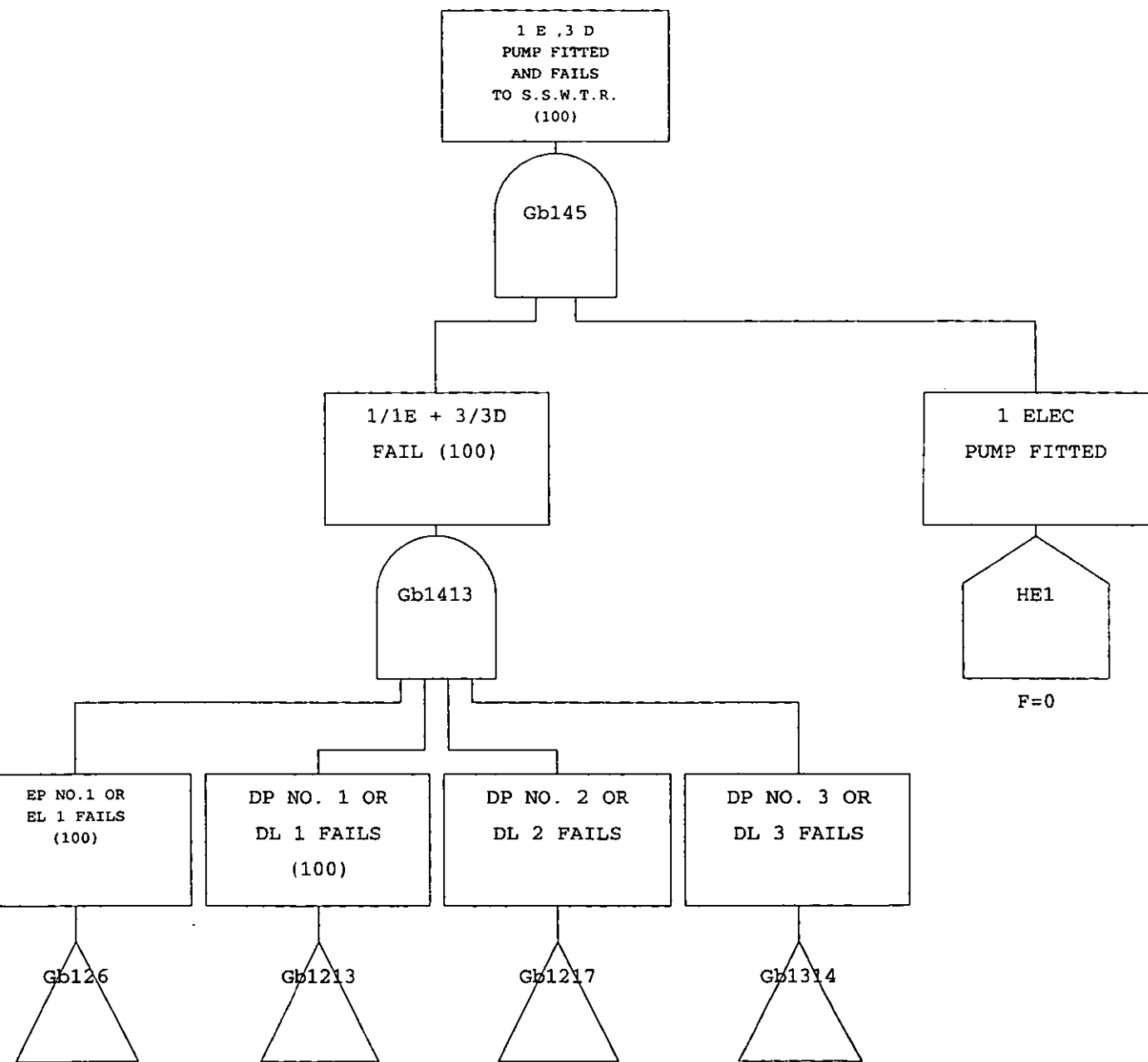
Gb126

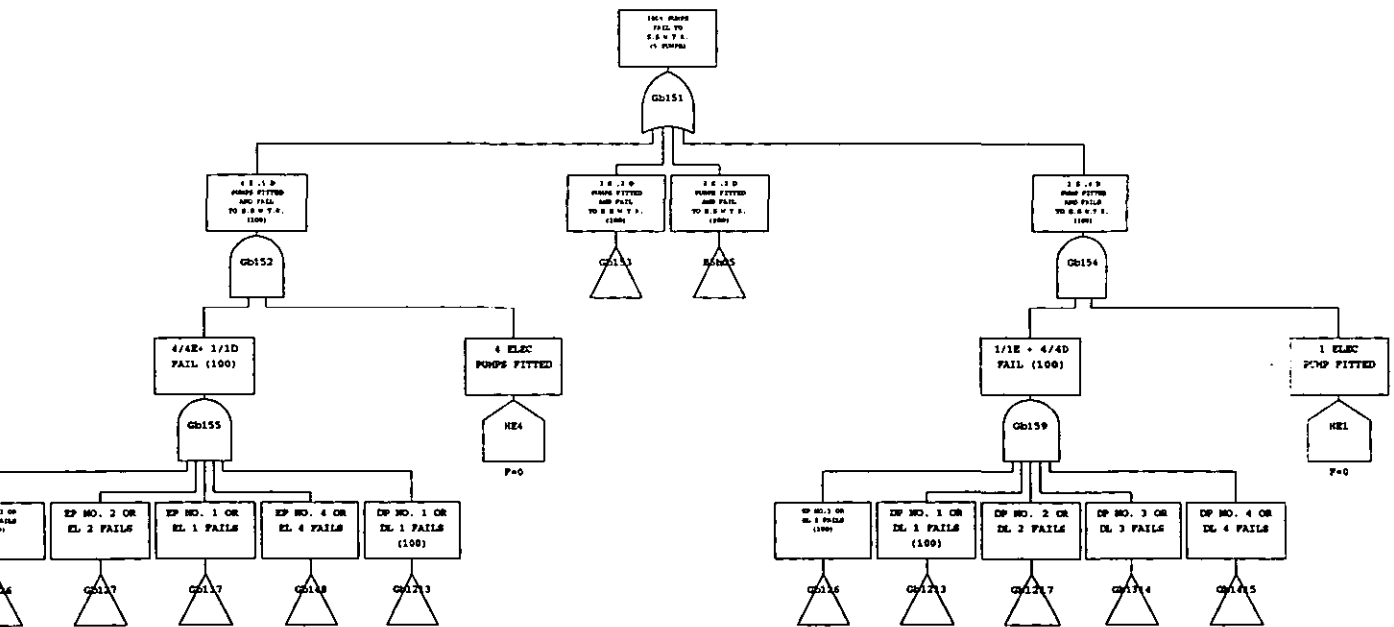
Gb127

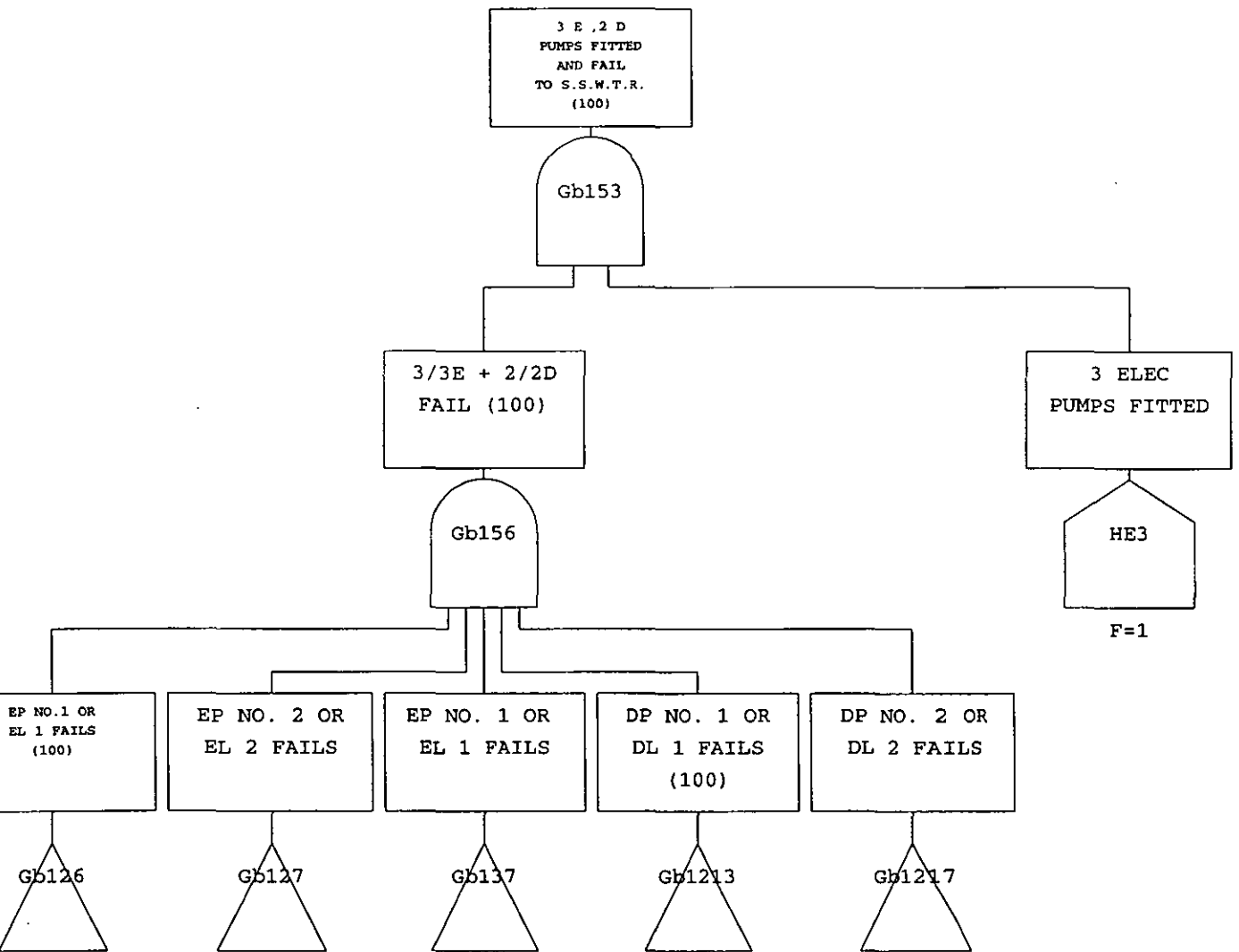
Gb137

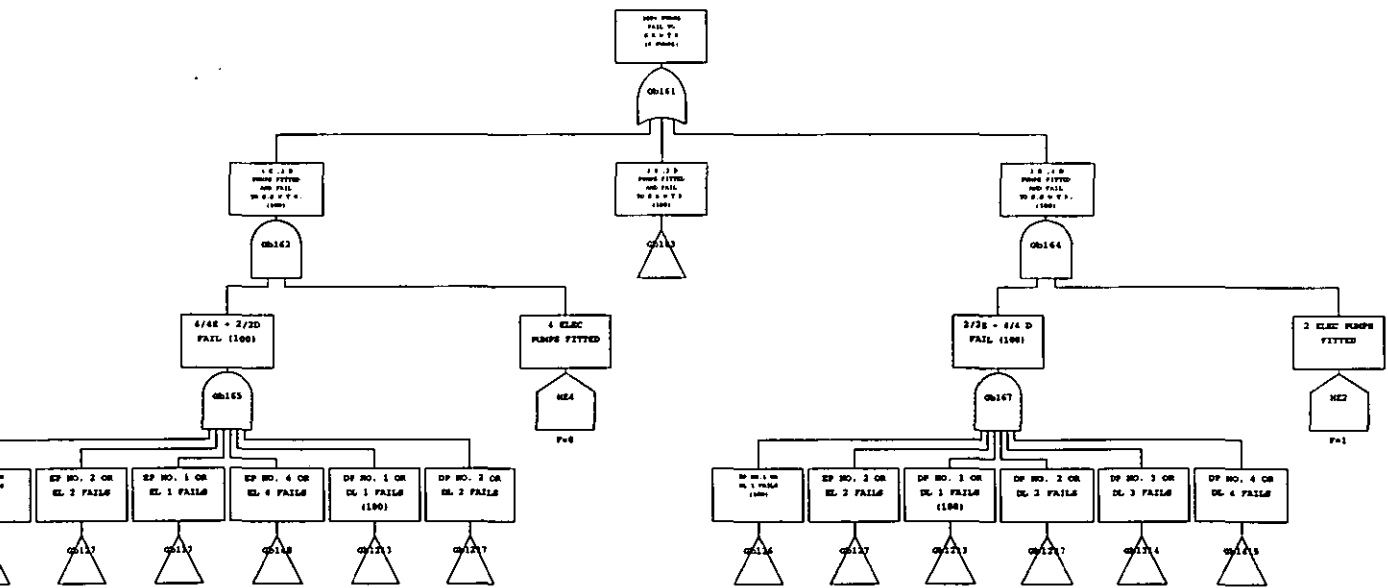
Gb1213

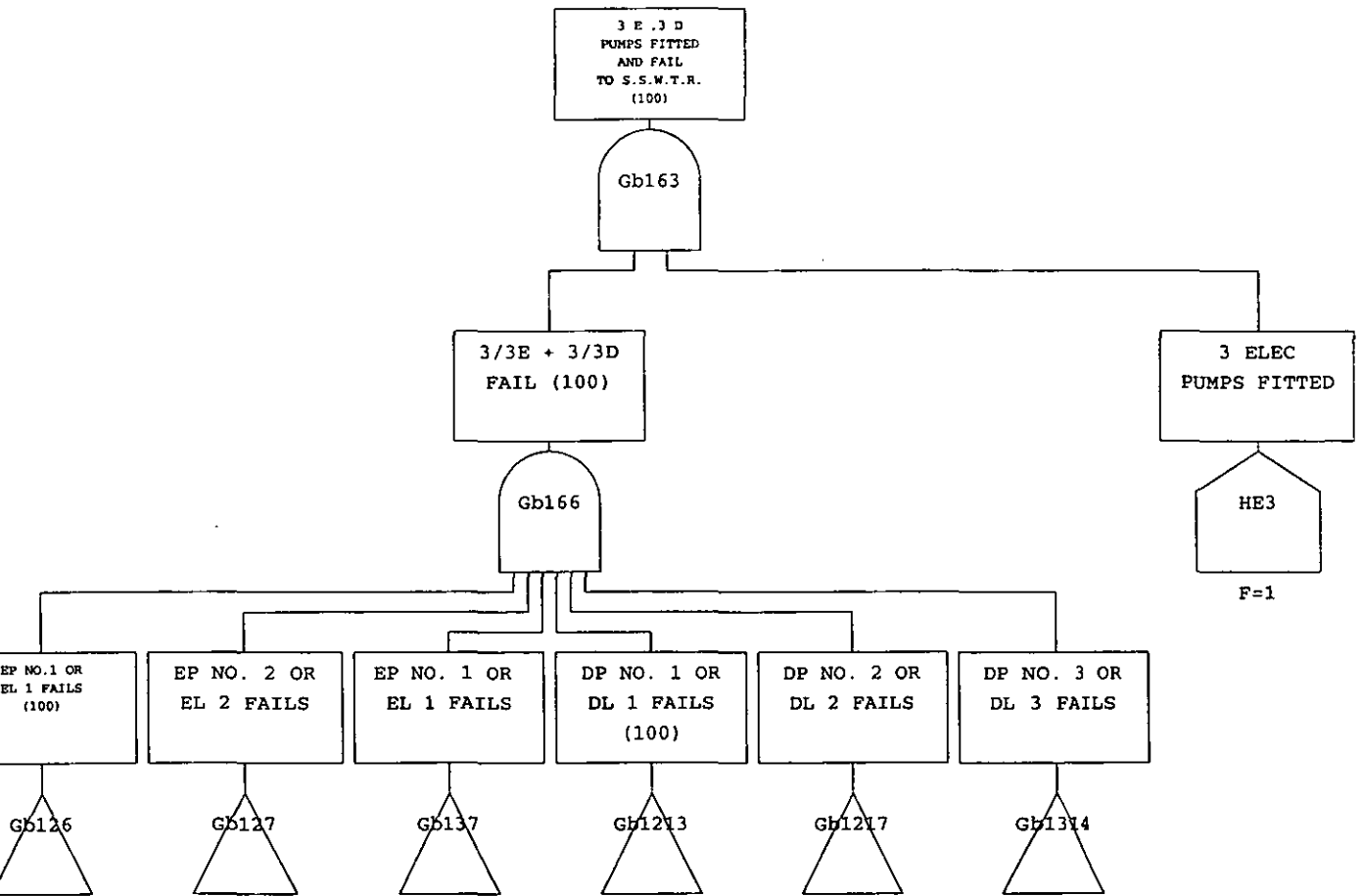


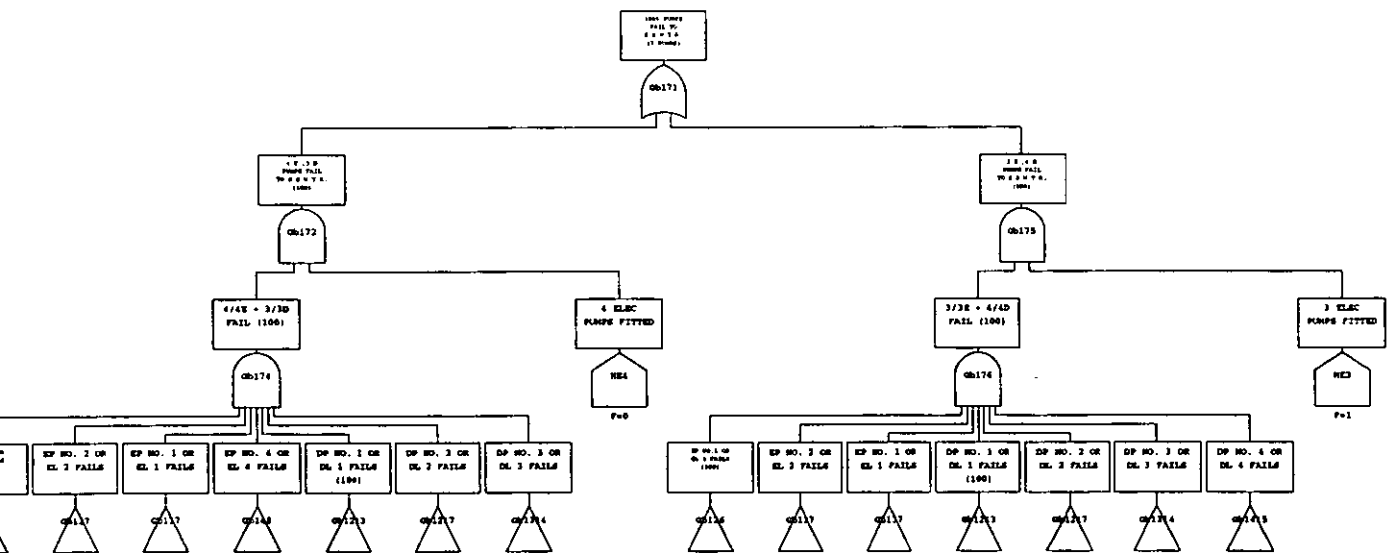


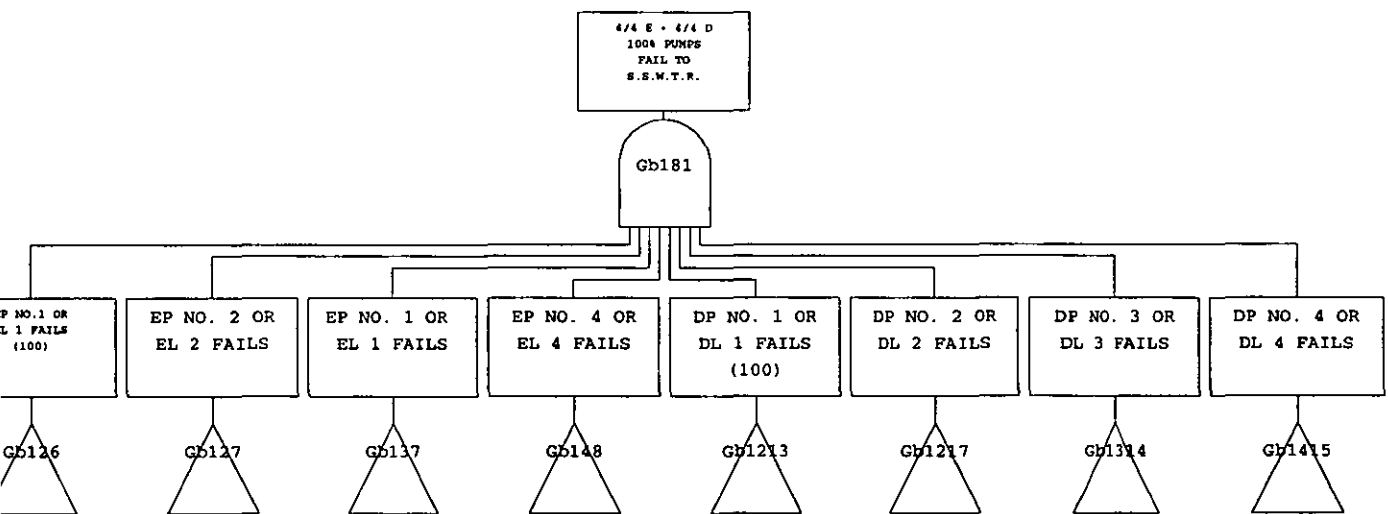


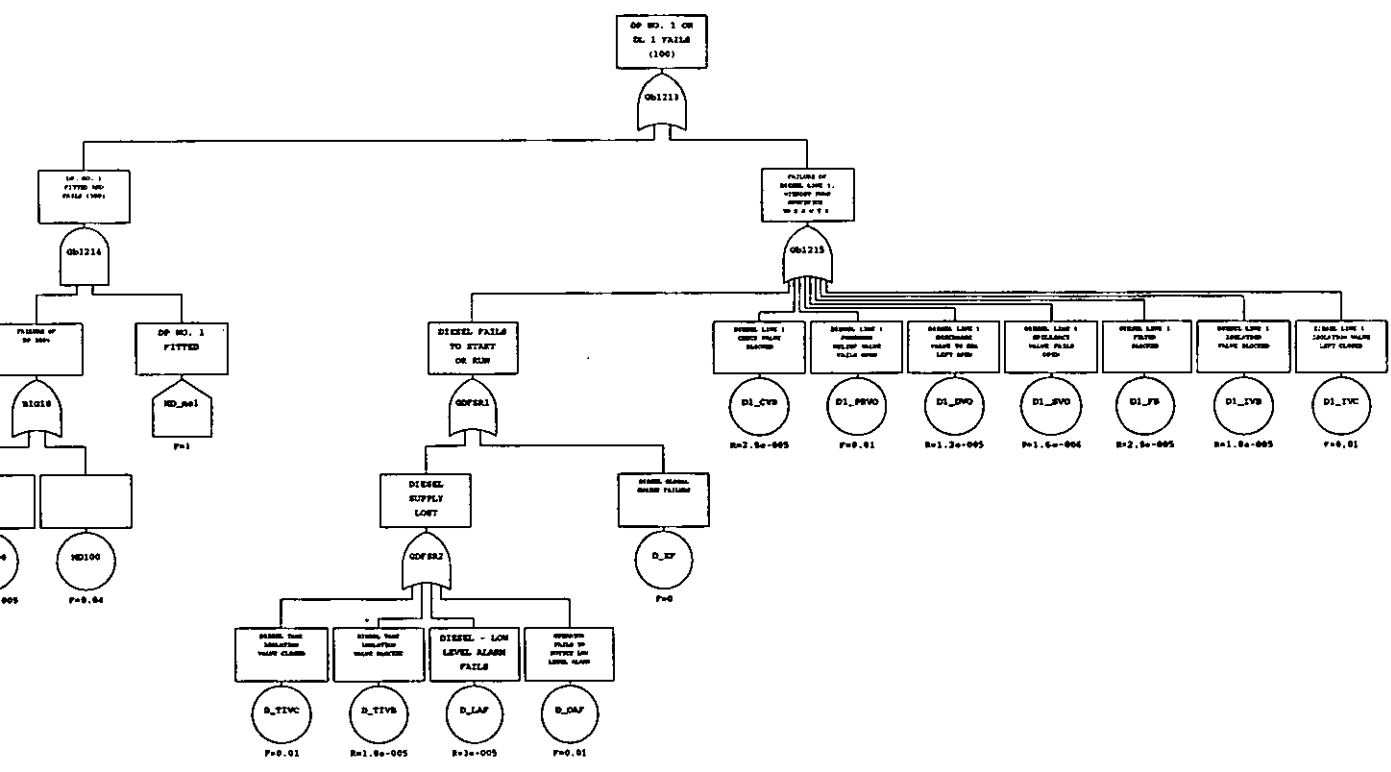


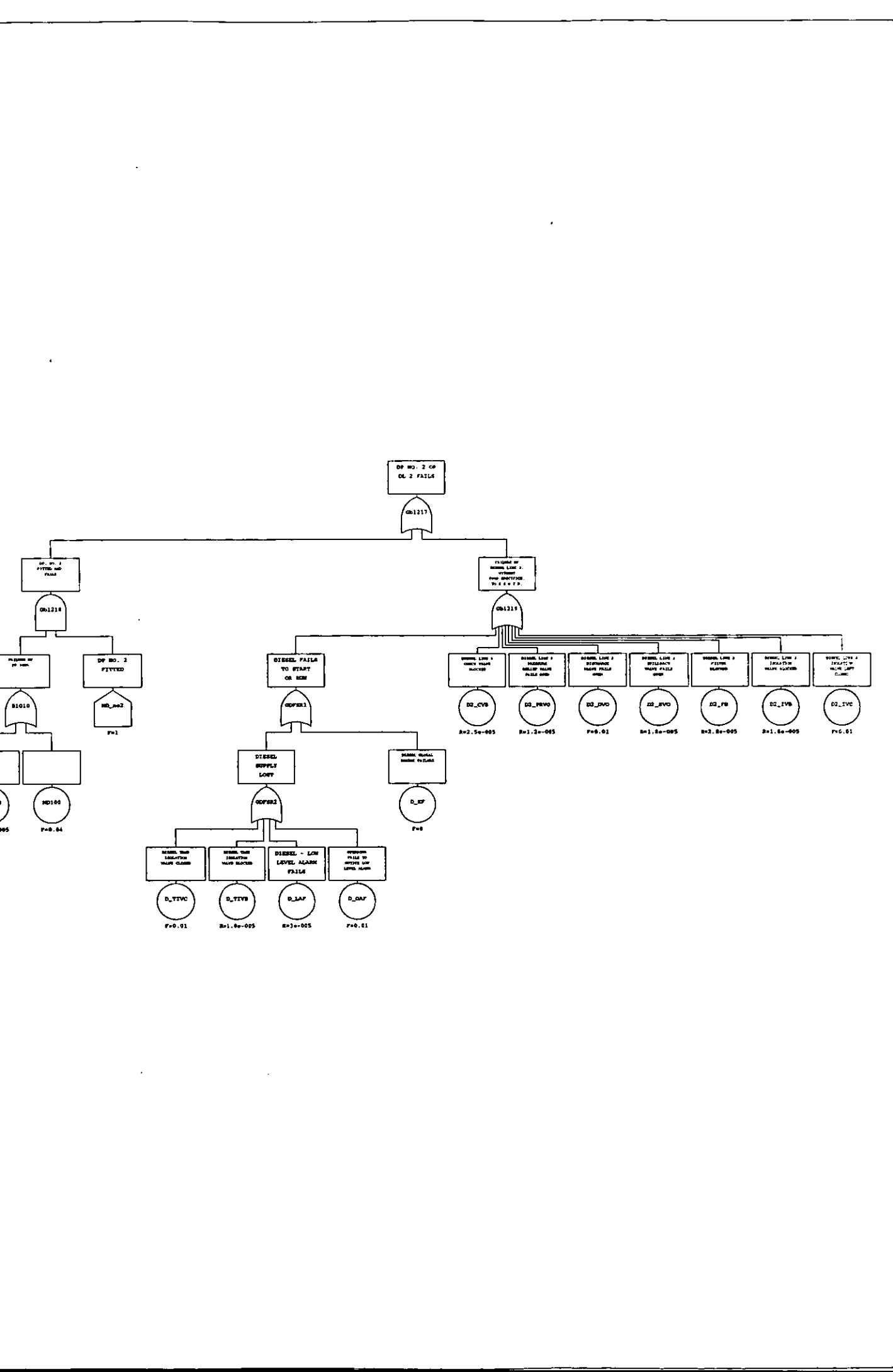


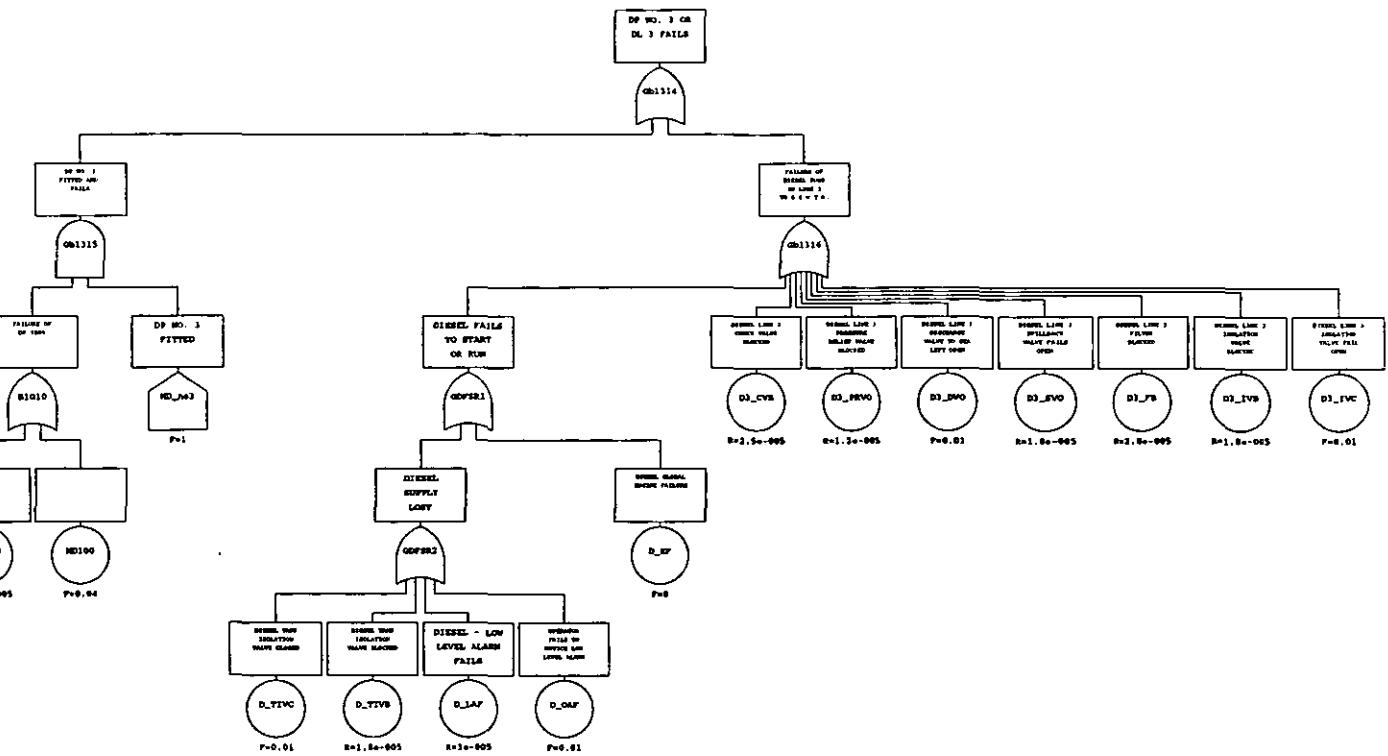


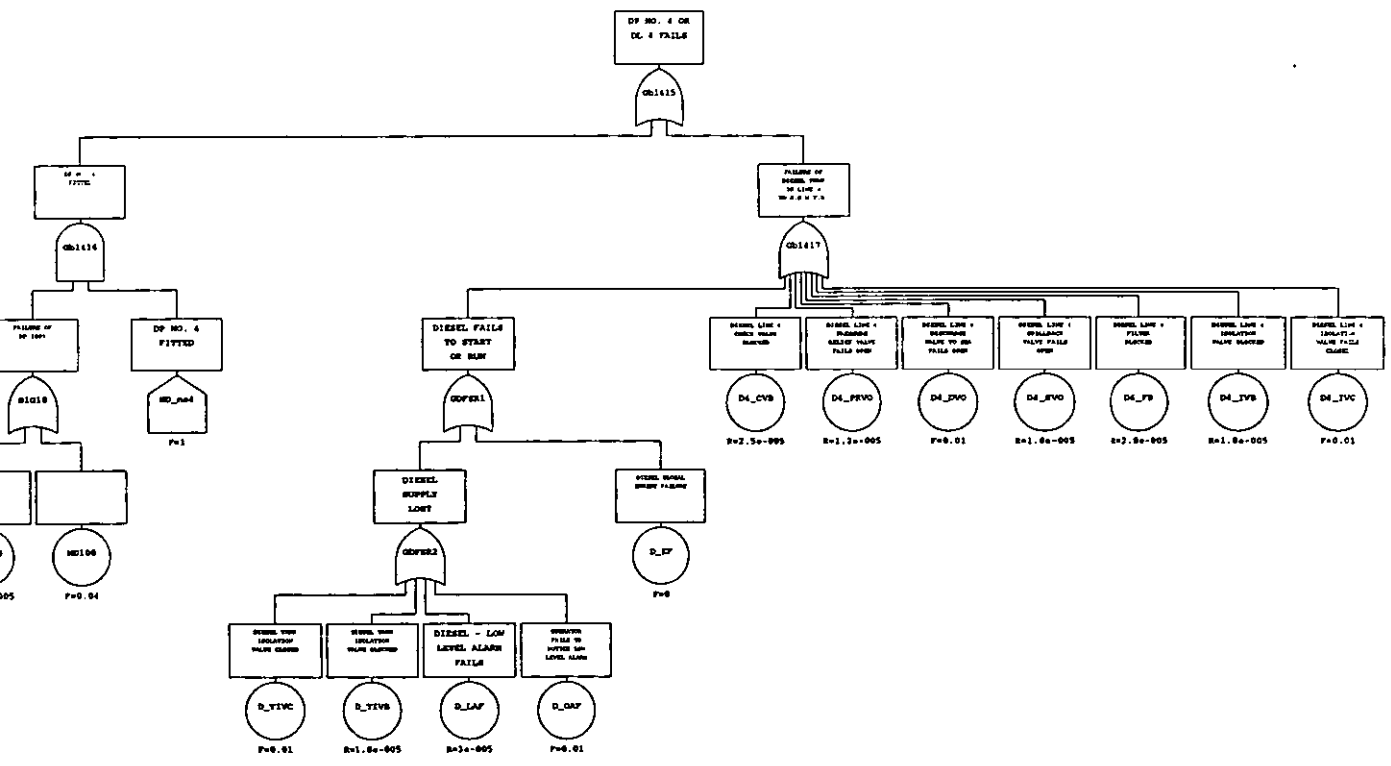


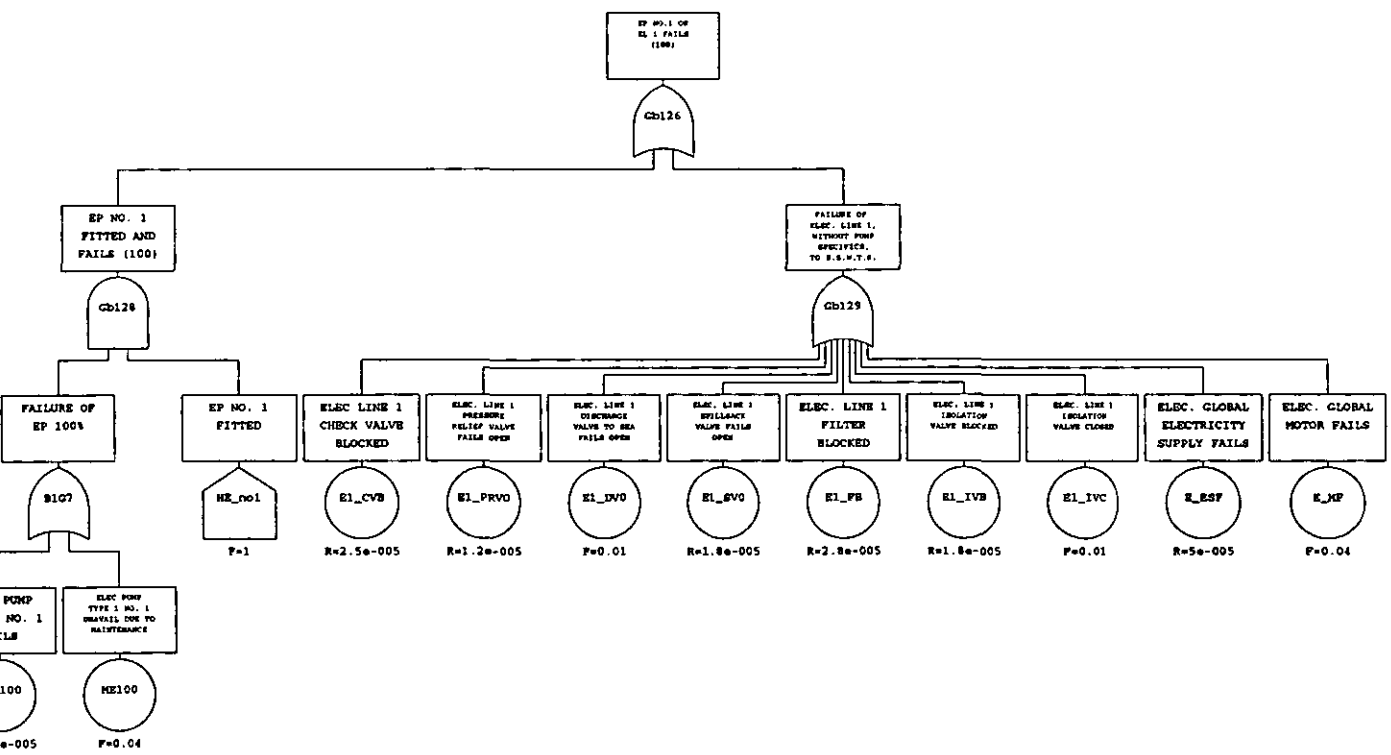


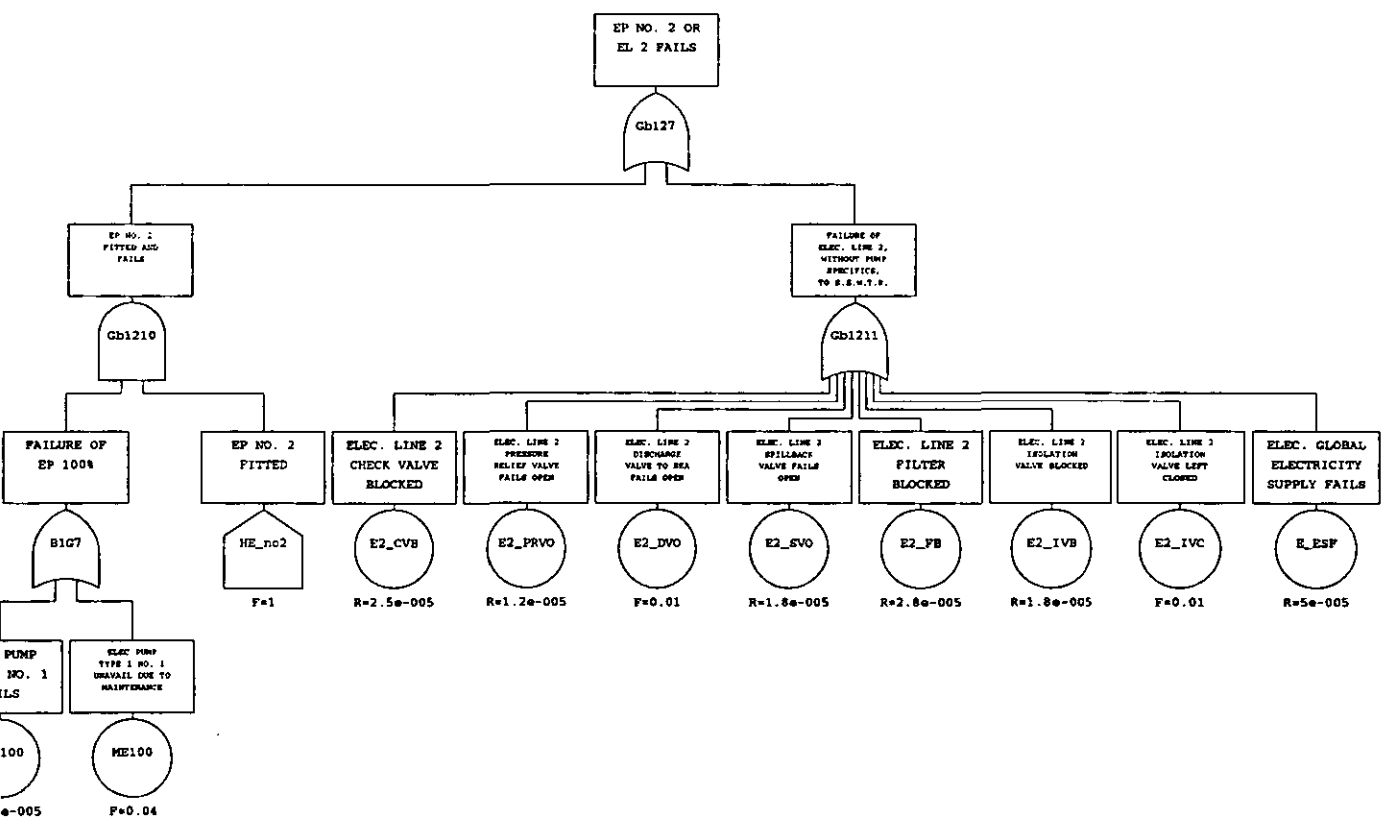


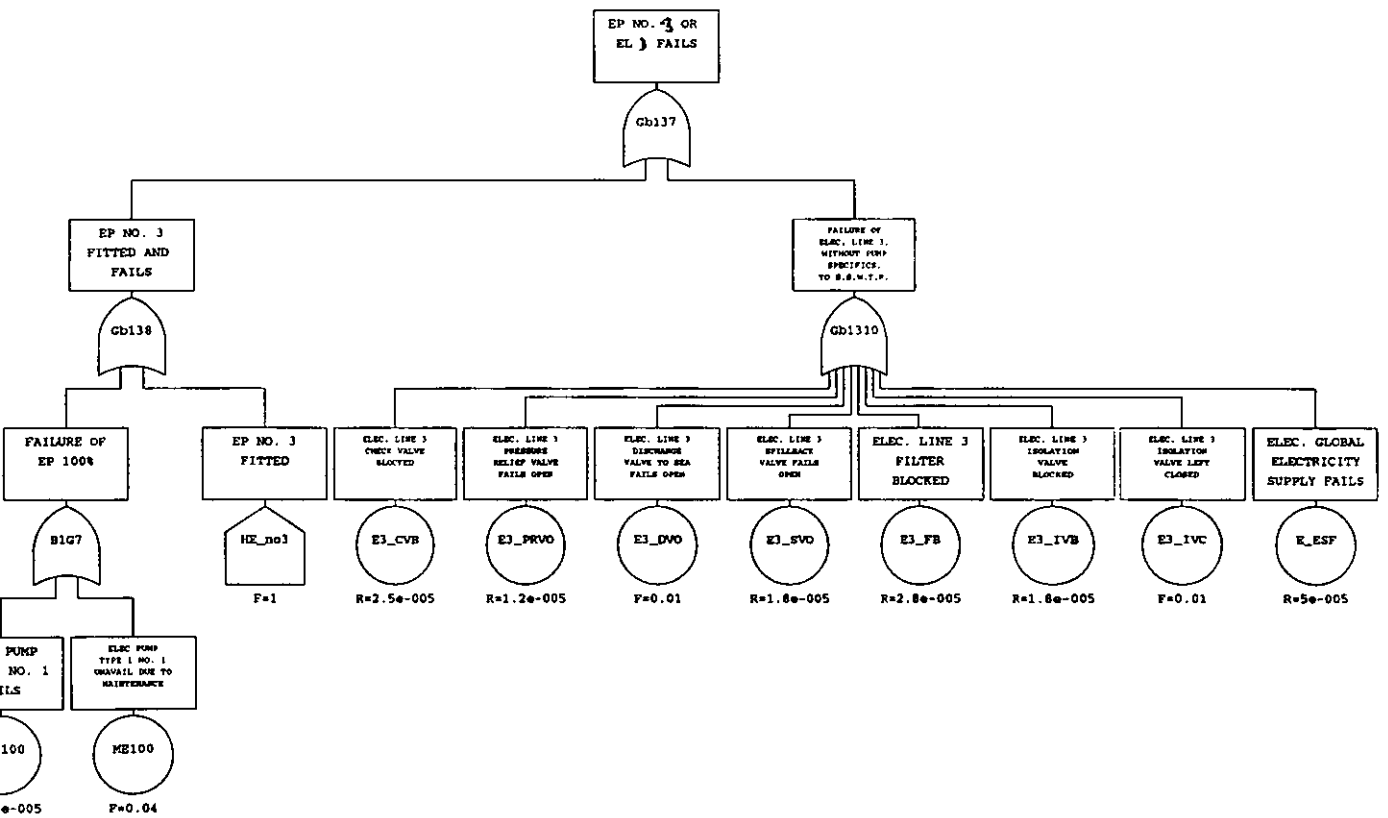


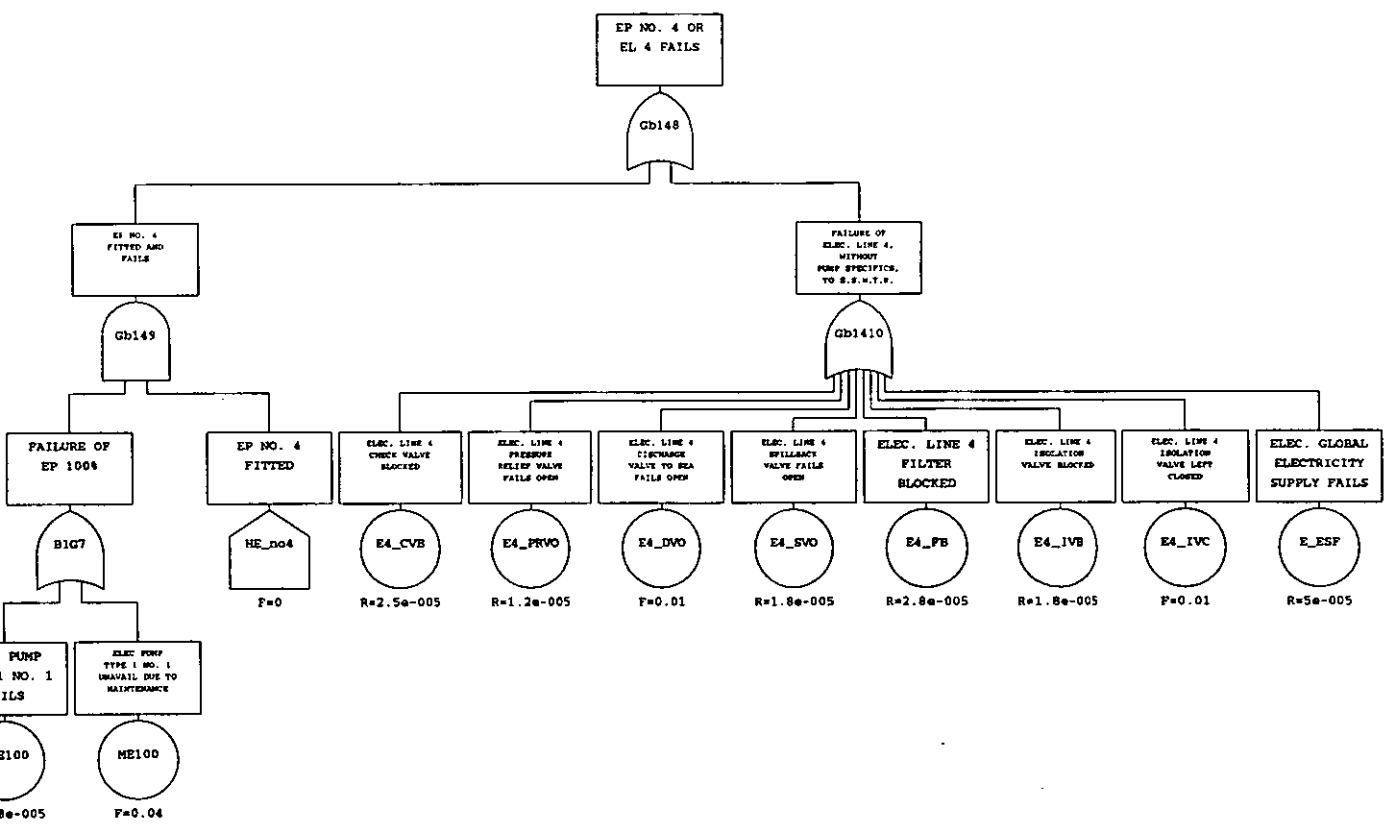


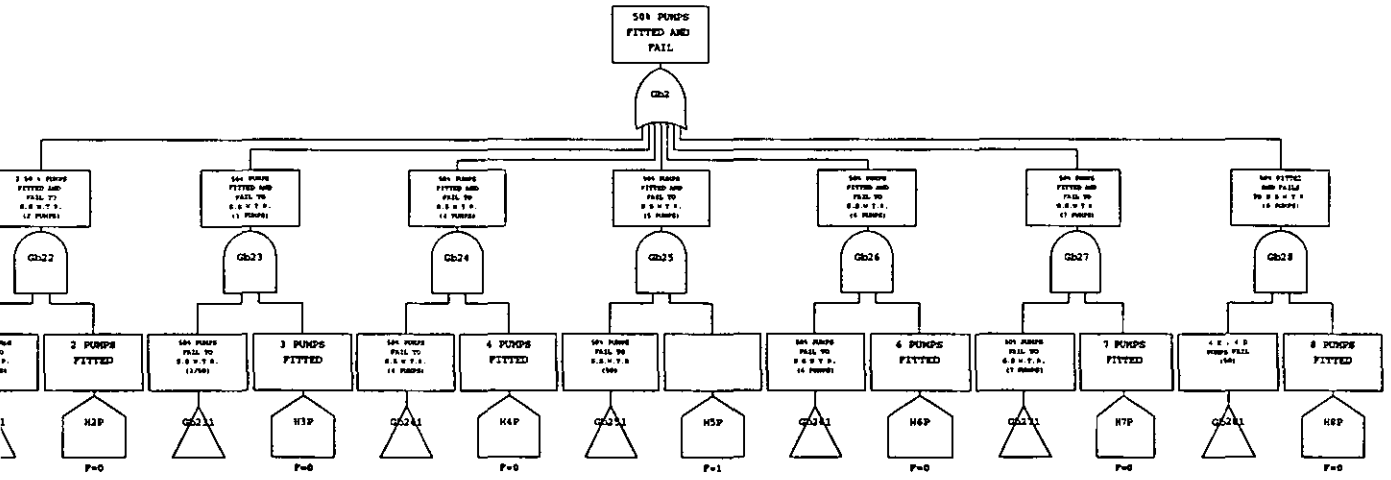


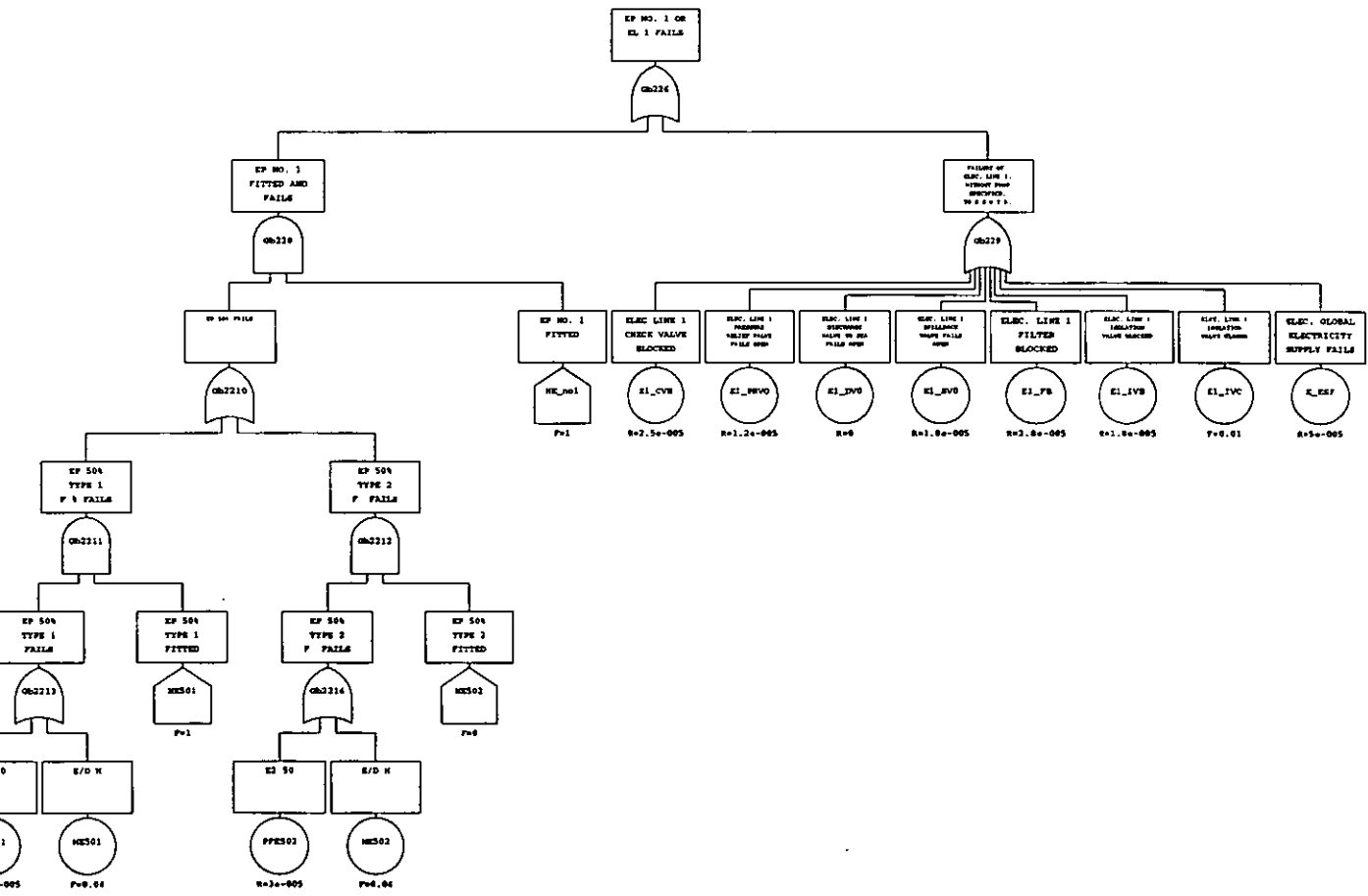


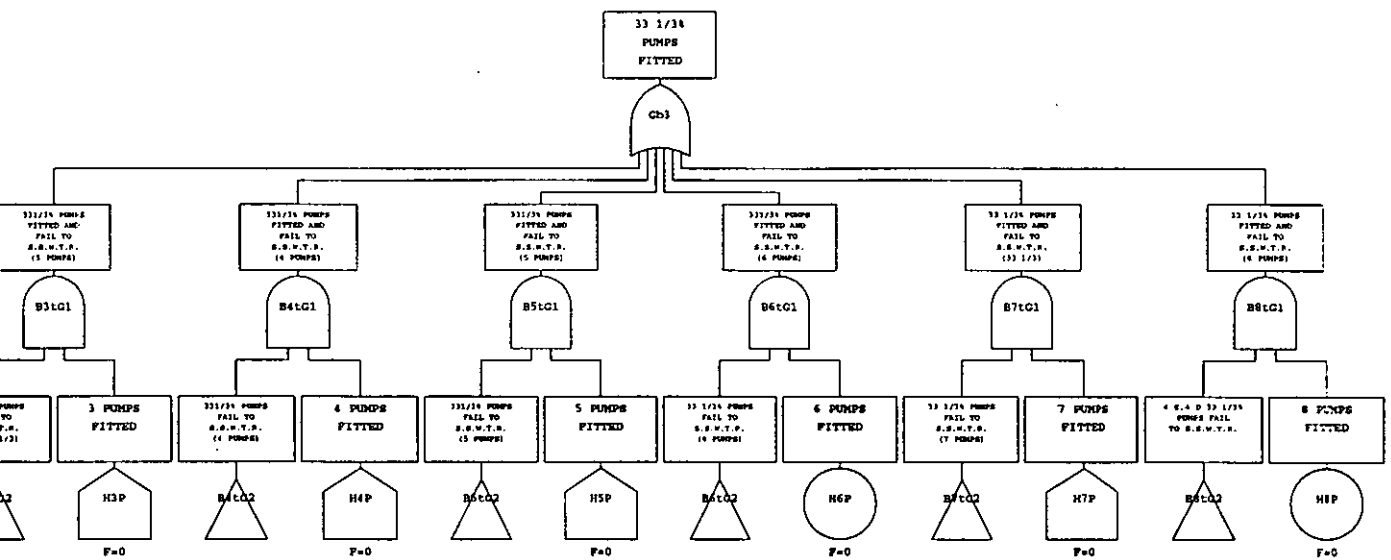


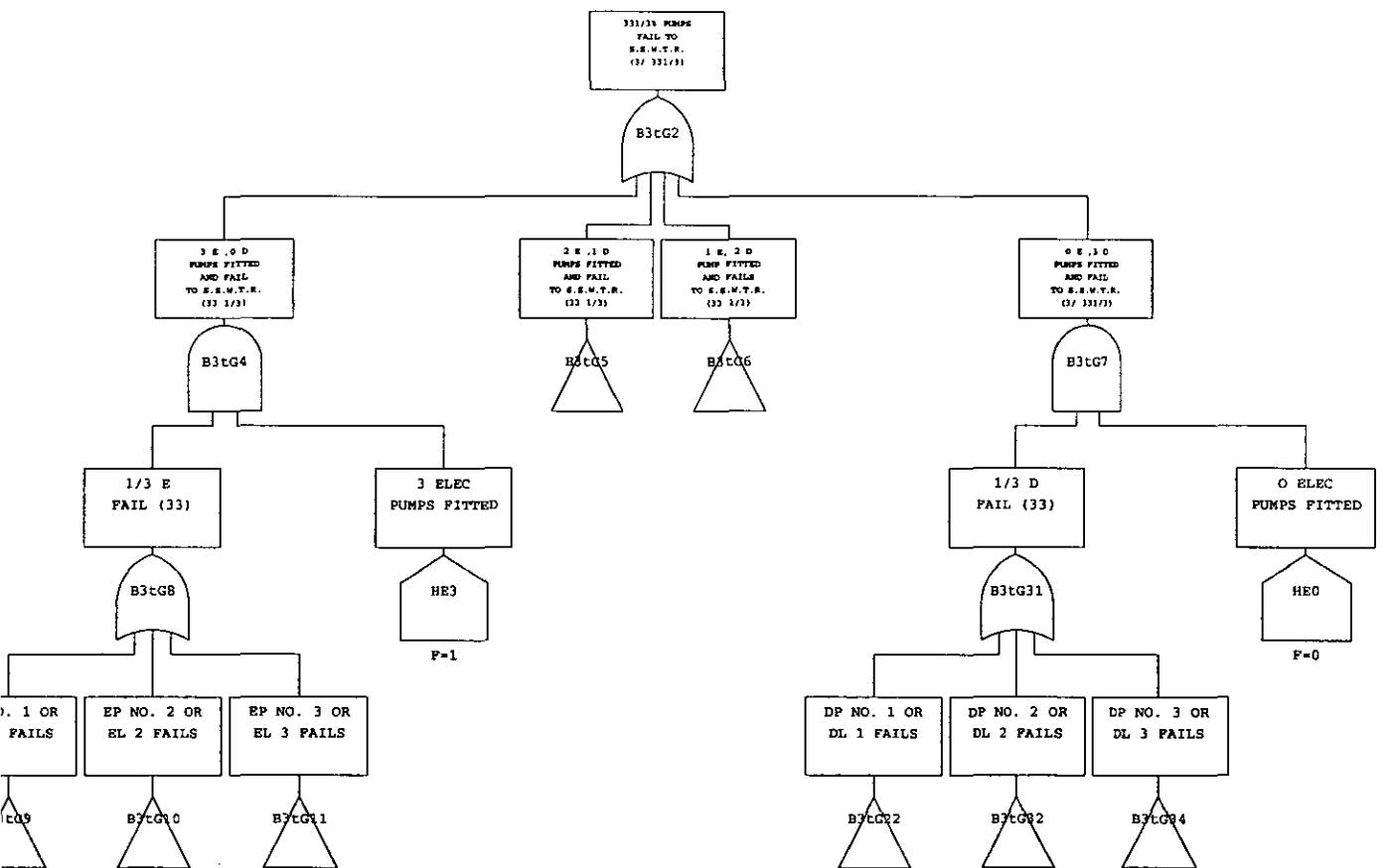


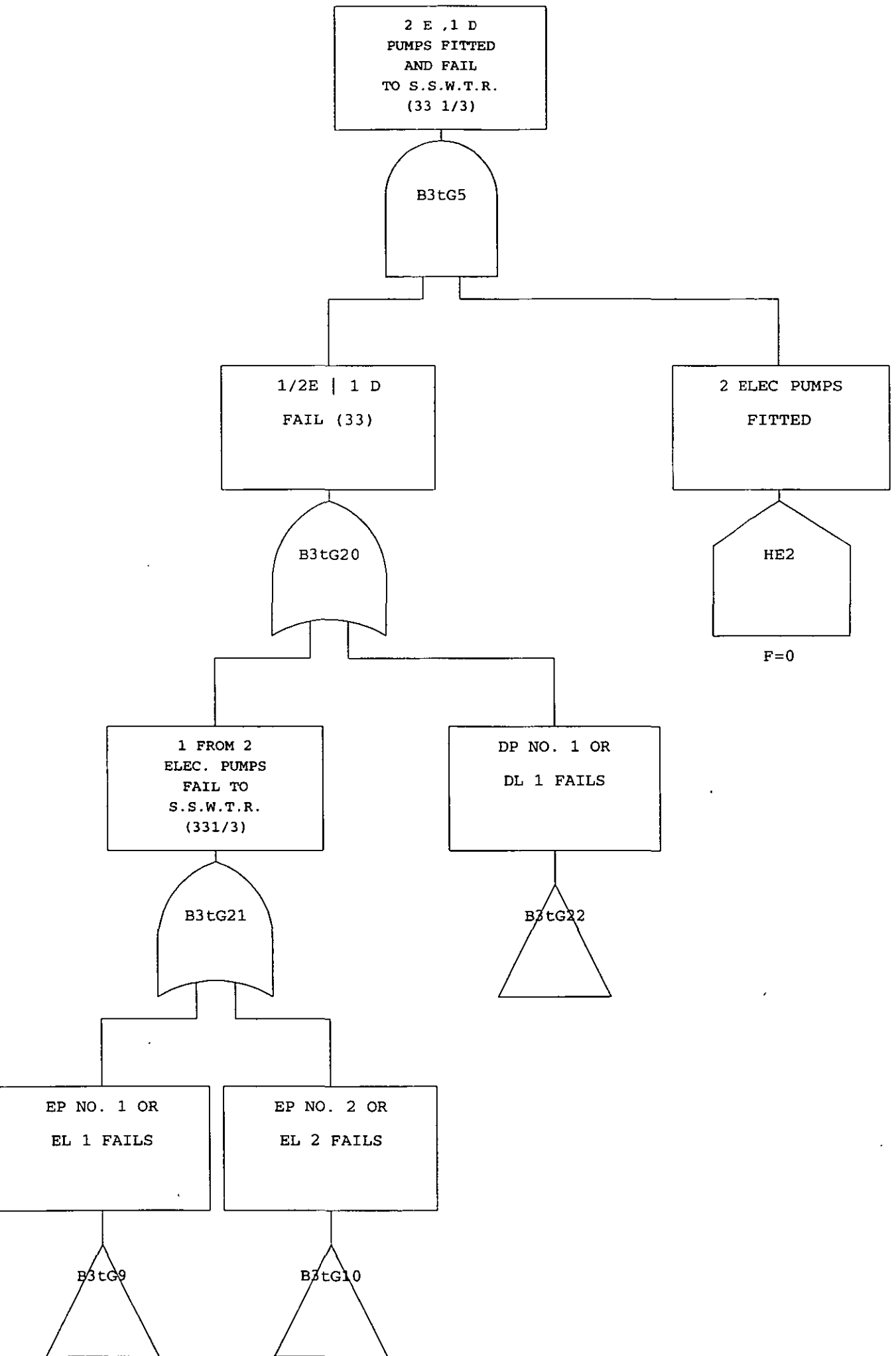












2 E, 1 D
PUMPS FITTED
AND FAIL
TO S.S.W.T.R.
(33 1/3)

B3tG5

1/2E | 1 D
FAIL (33)

2 ELEC PUMPS
FITTED

B3tG20

HE2

F=0

1 FROM 2
ELEC. PUMPS
FAIL TO
S.S.W.T.R.
(331/3)

DP NO. 1 OR
DL 1 FAILS

B3tG21

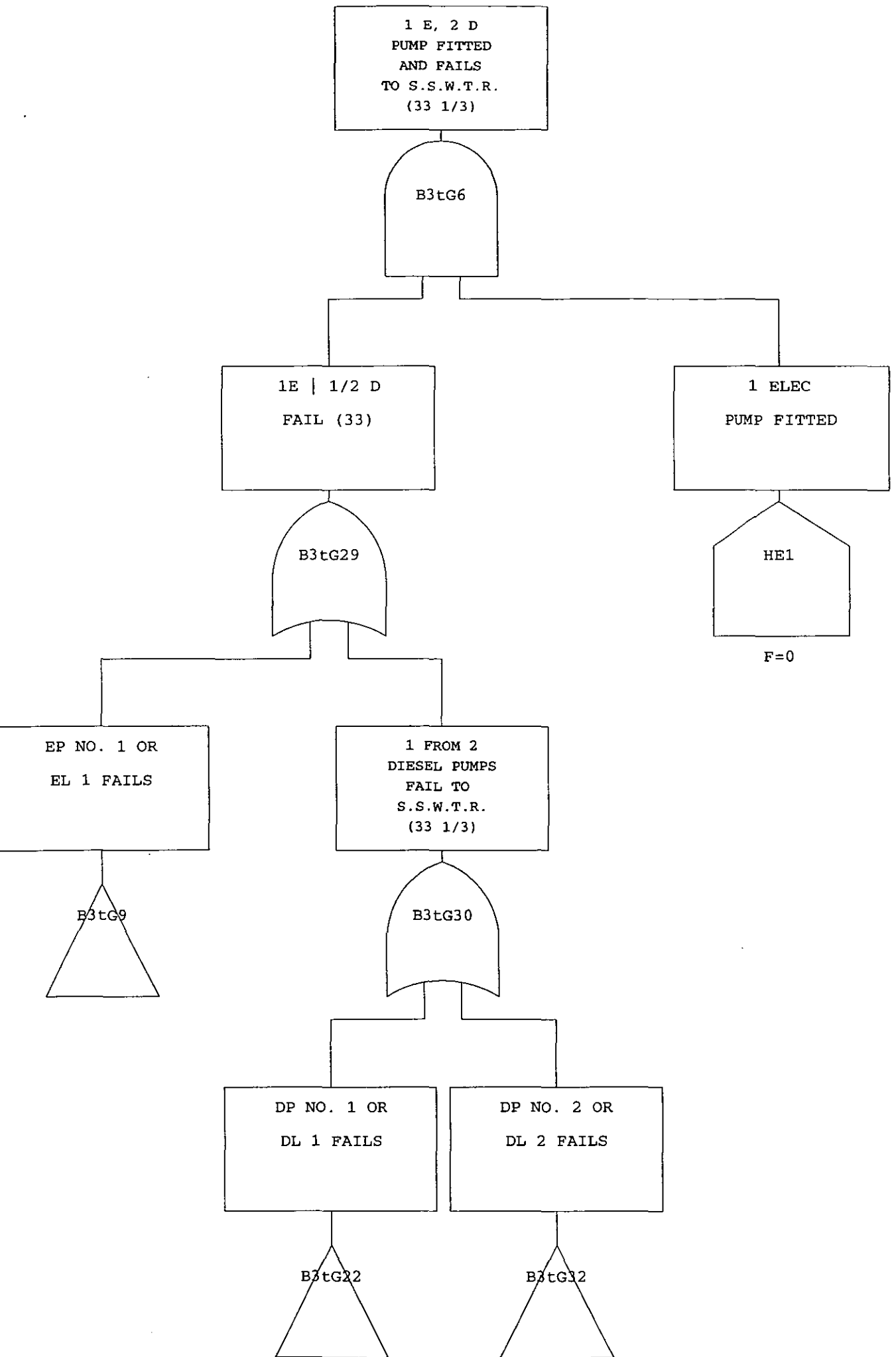
B3tG22

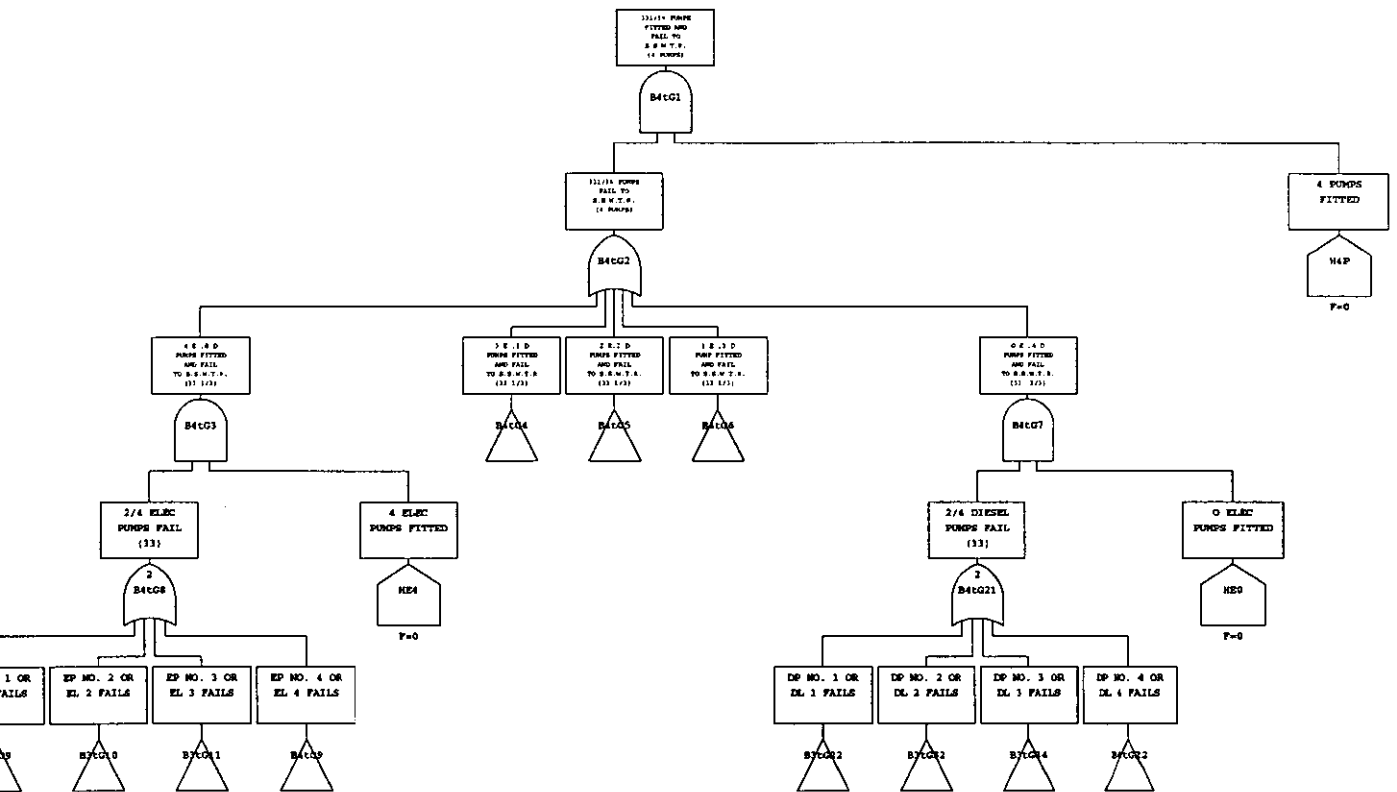
EP NO. 1 OR
EL 1 FAILS

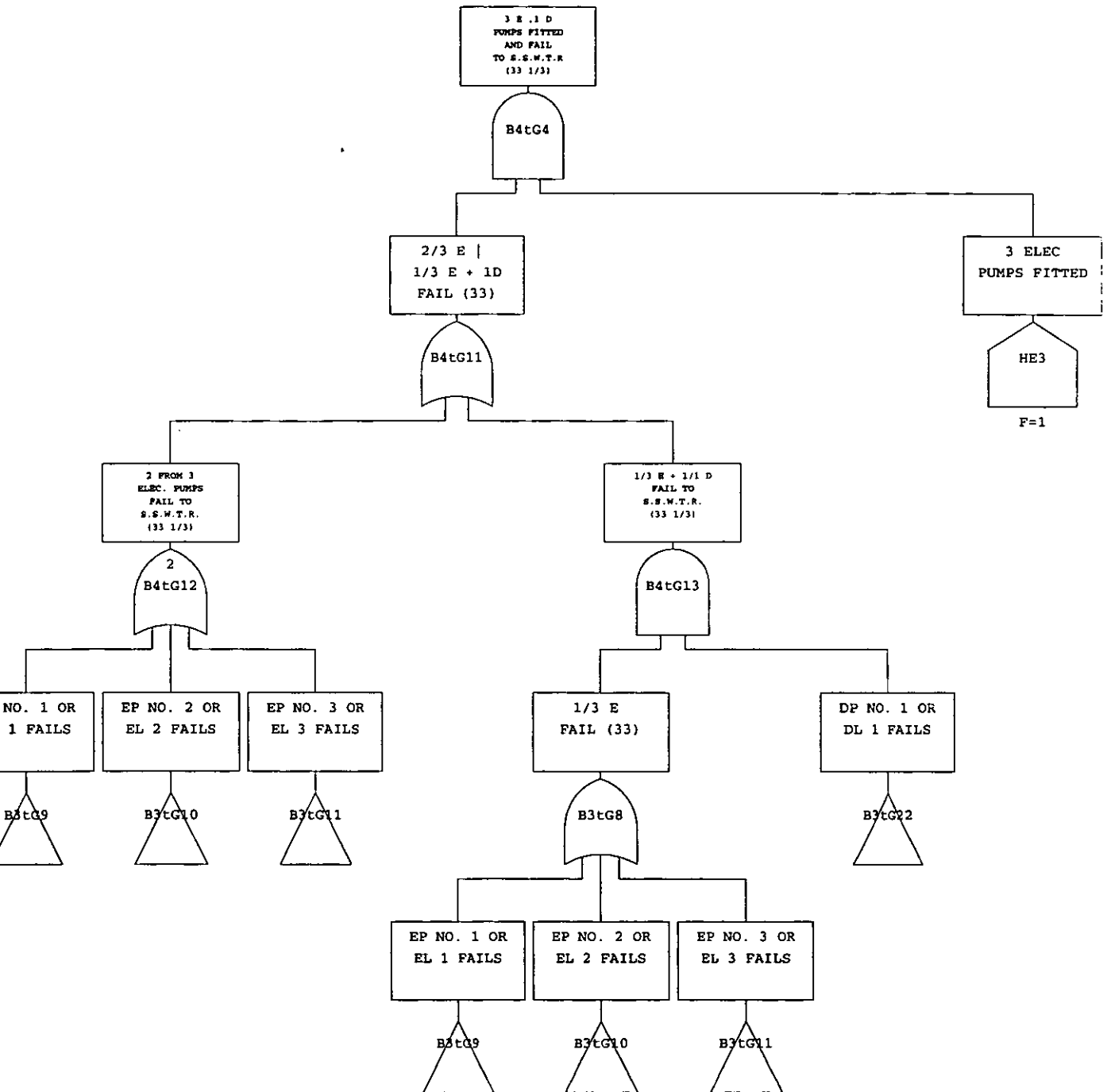
EP NO. 2 OR
EL 2 FAILS

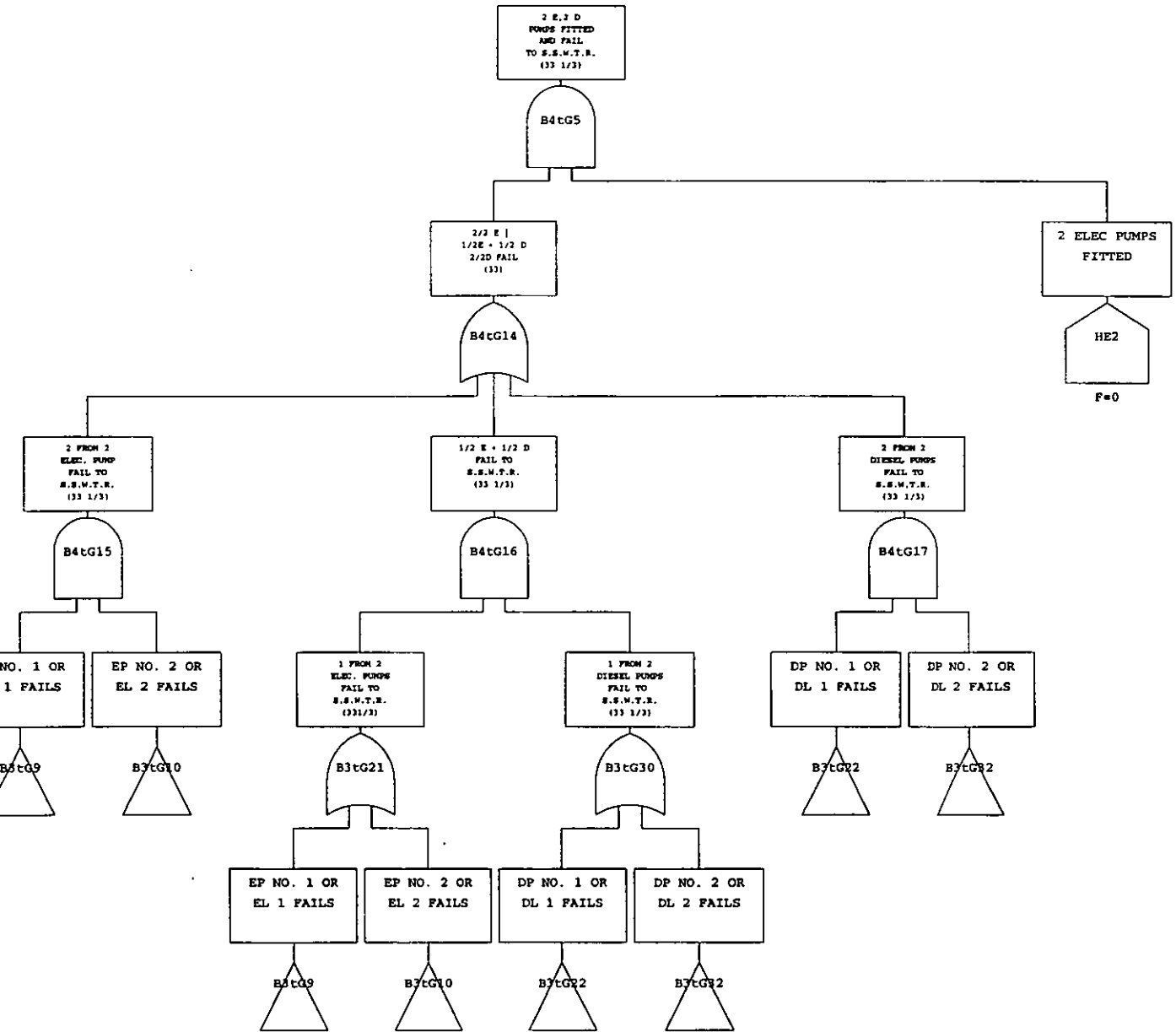
B3tG9

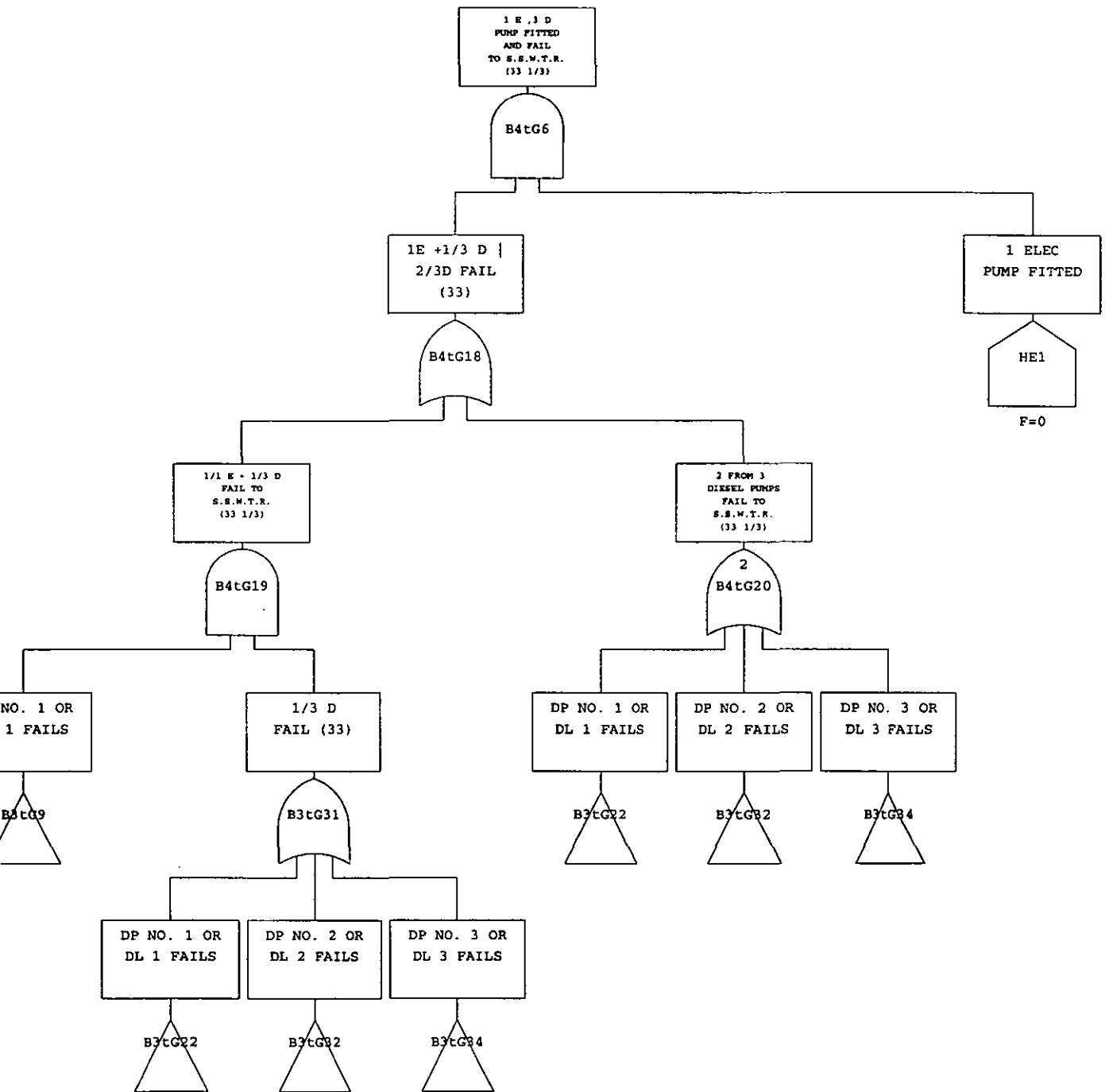
B3tG10

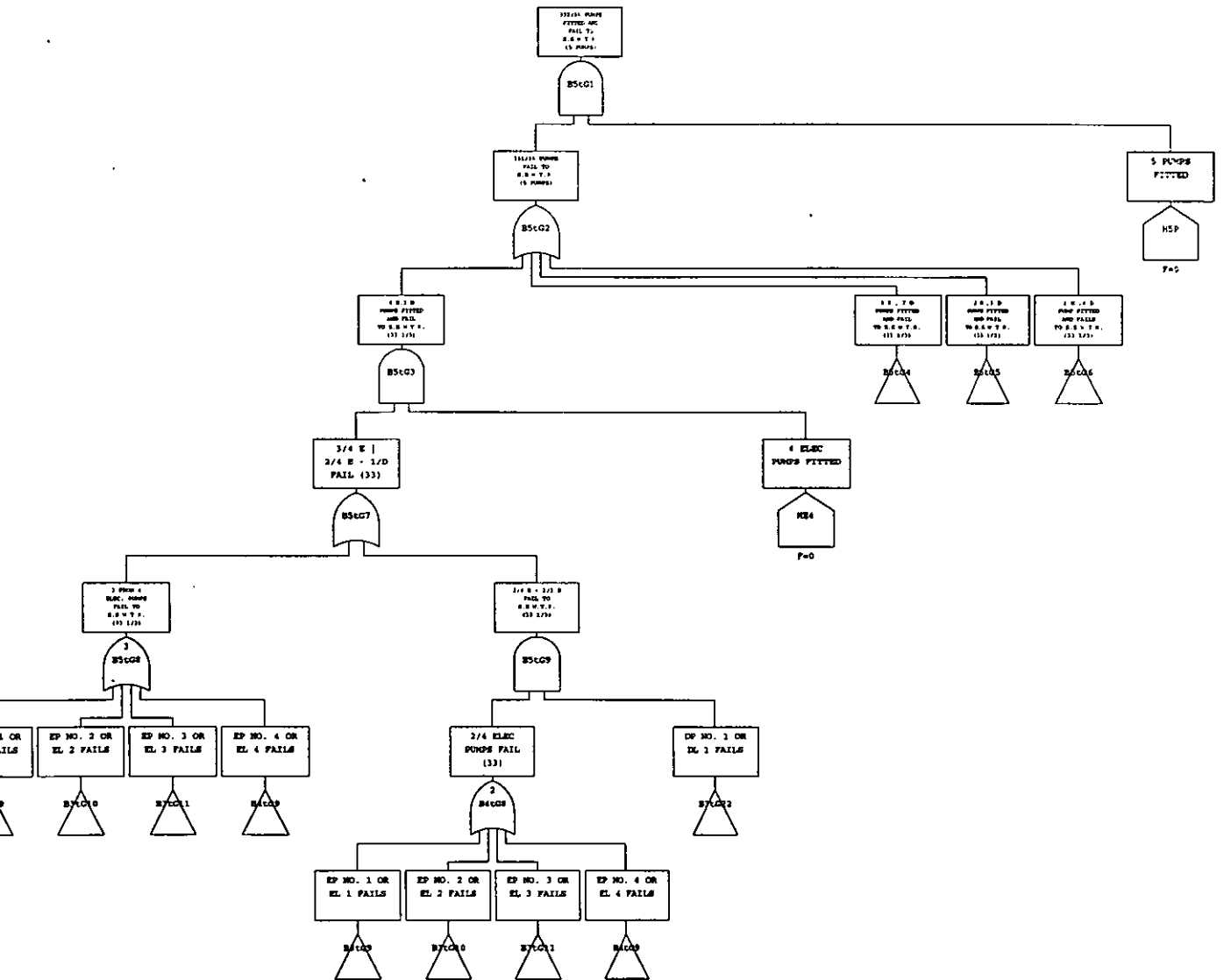


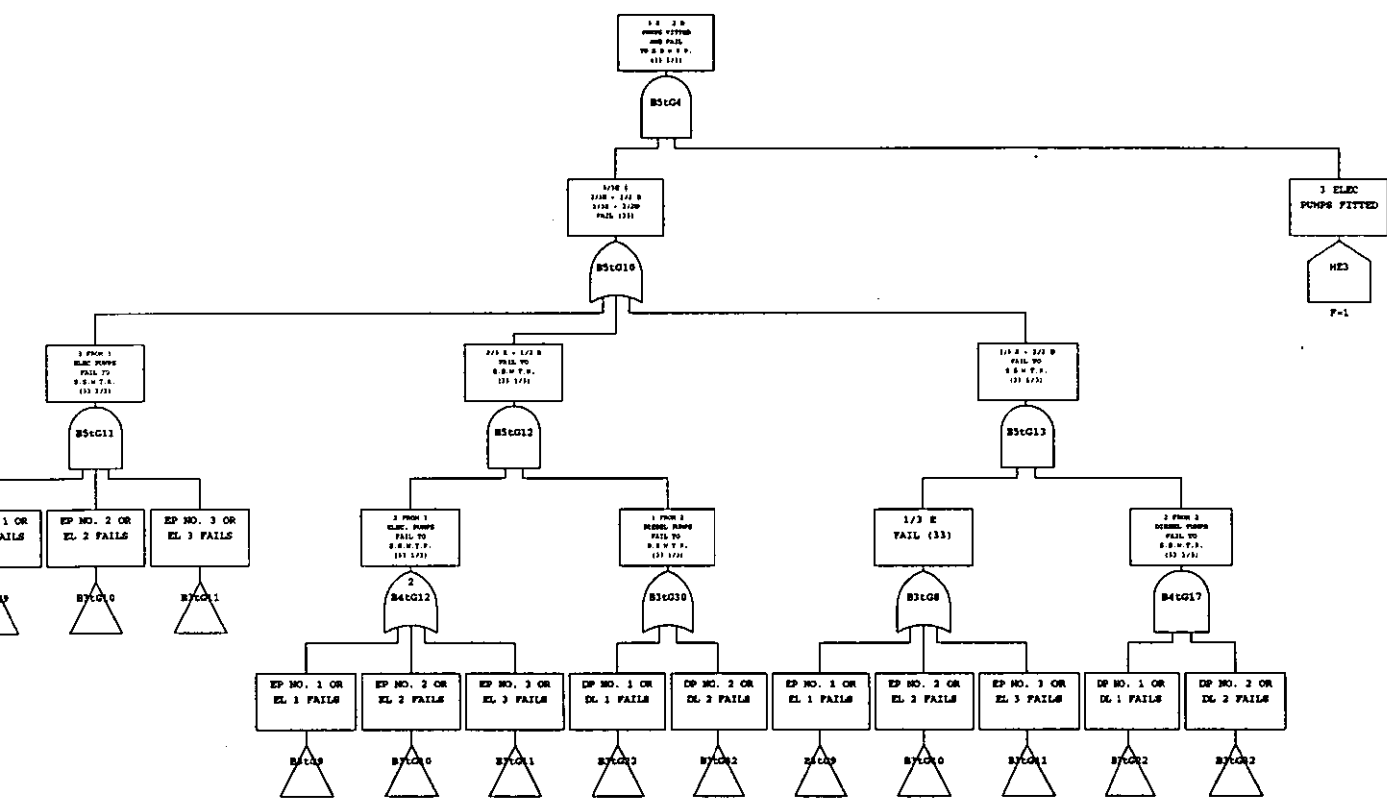


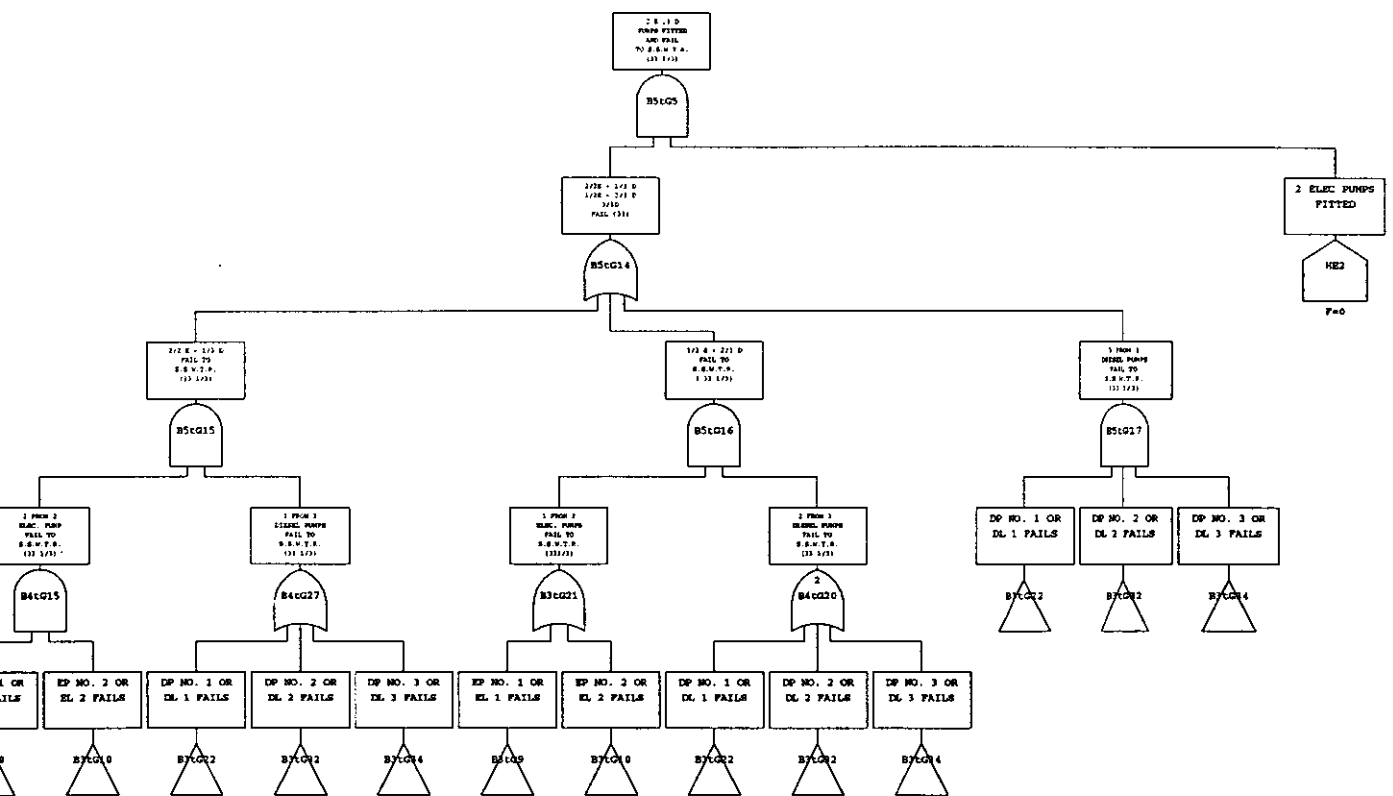


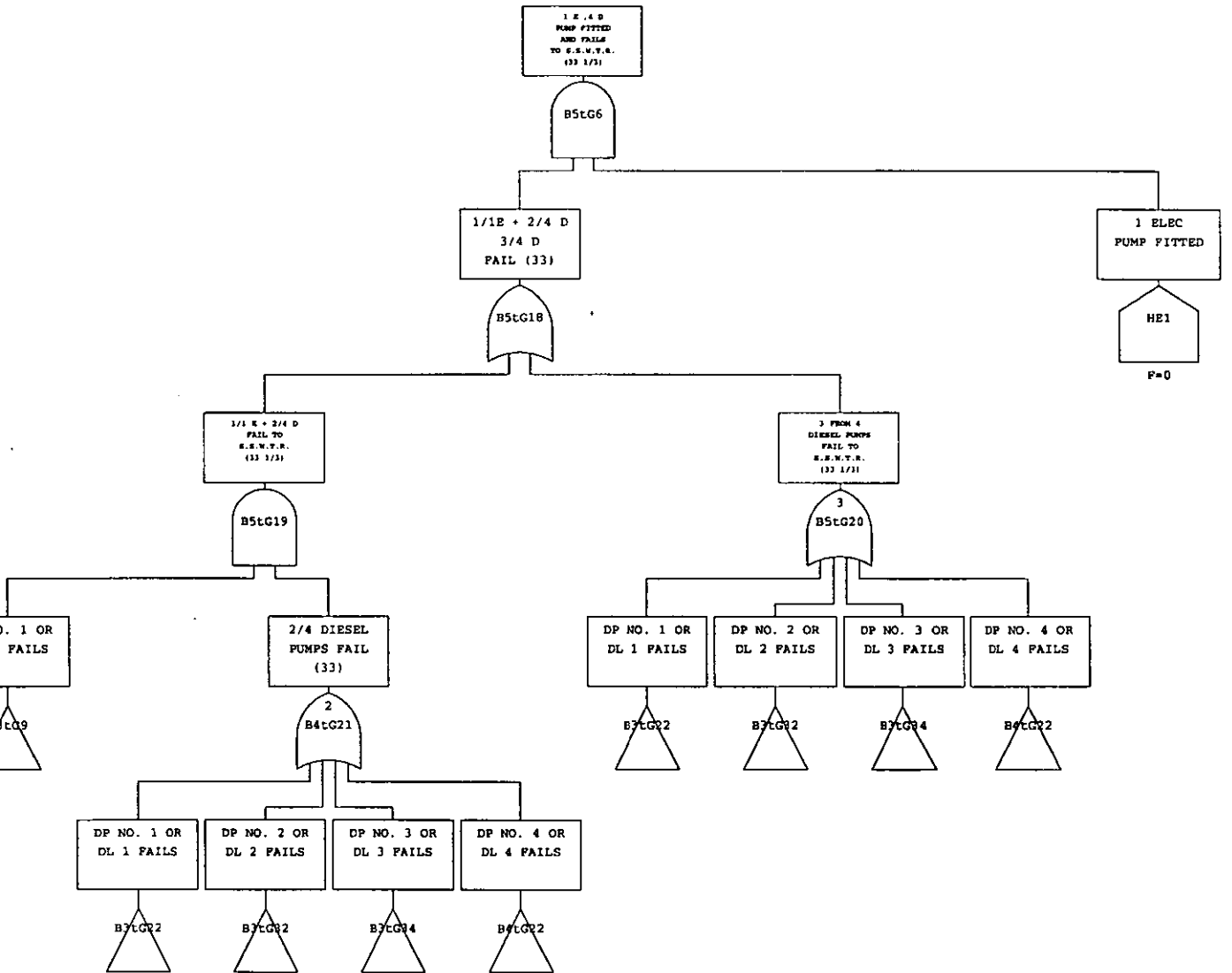


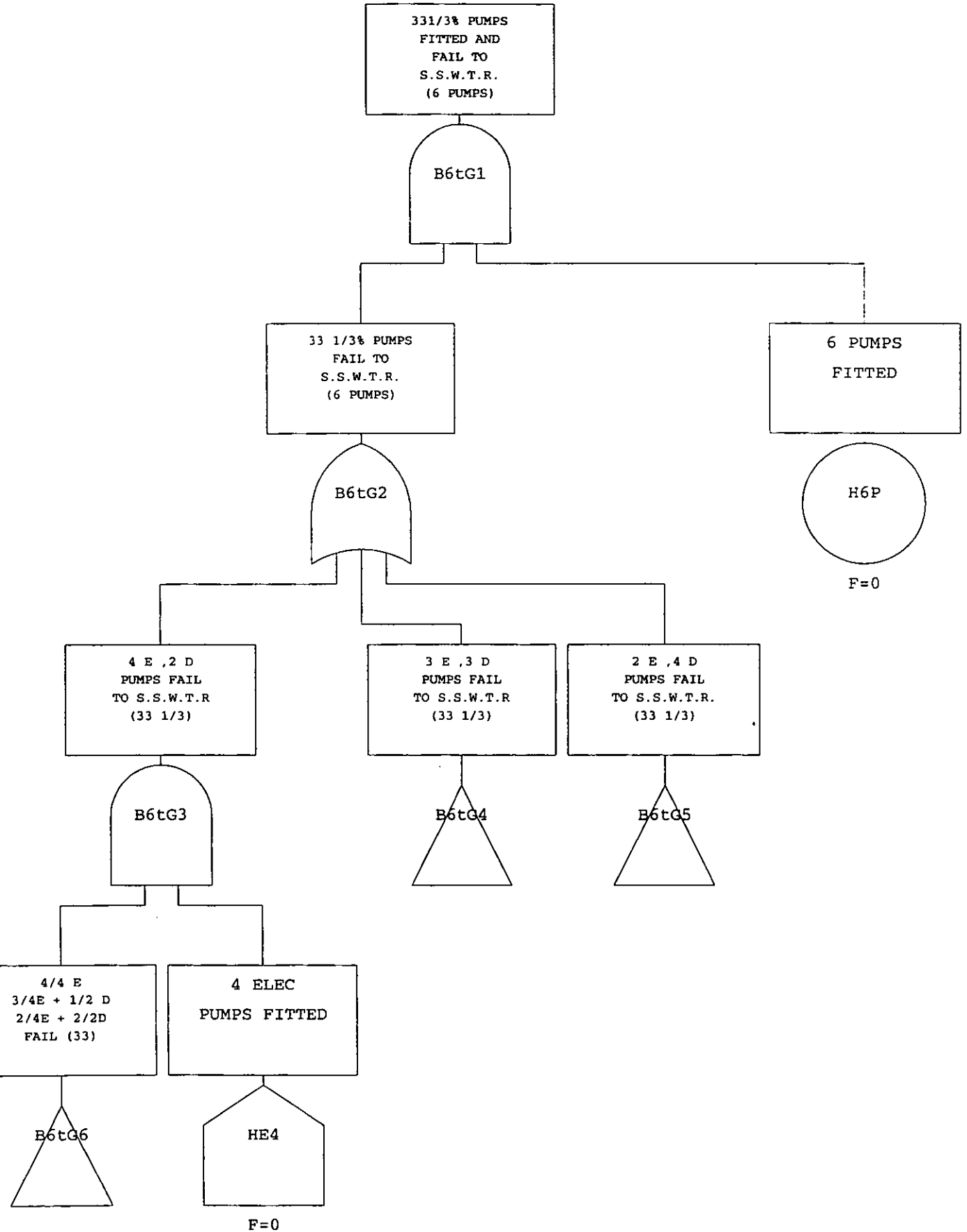


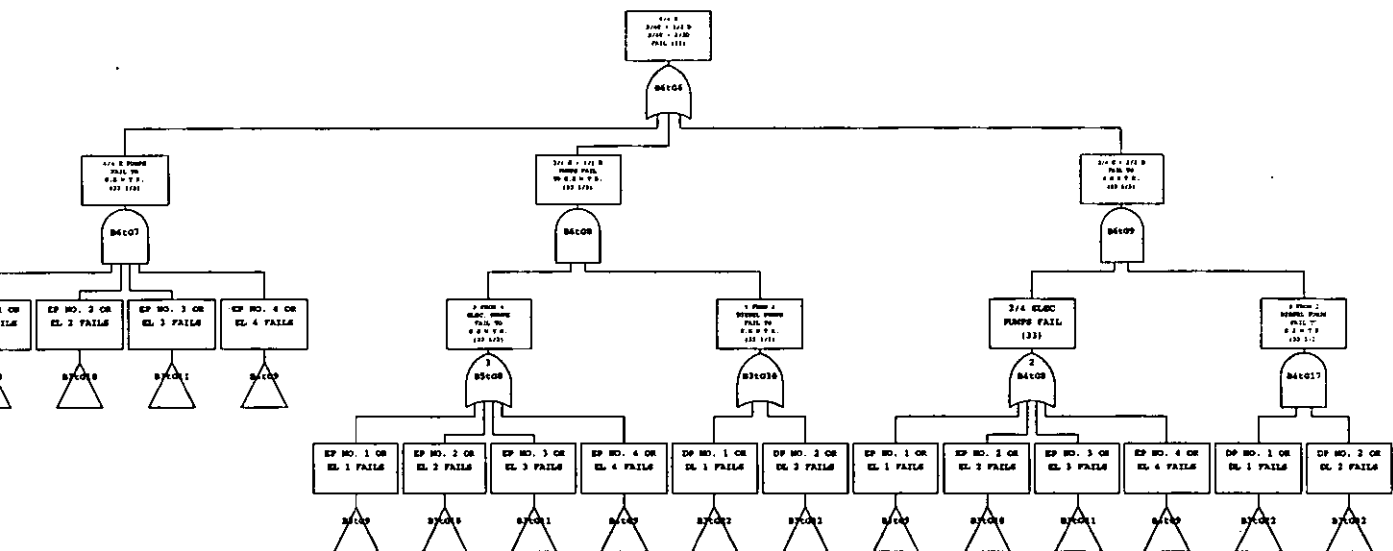


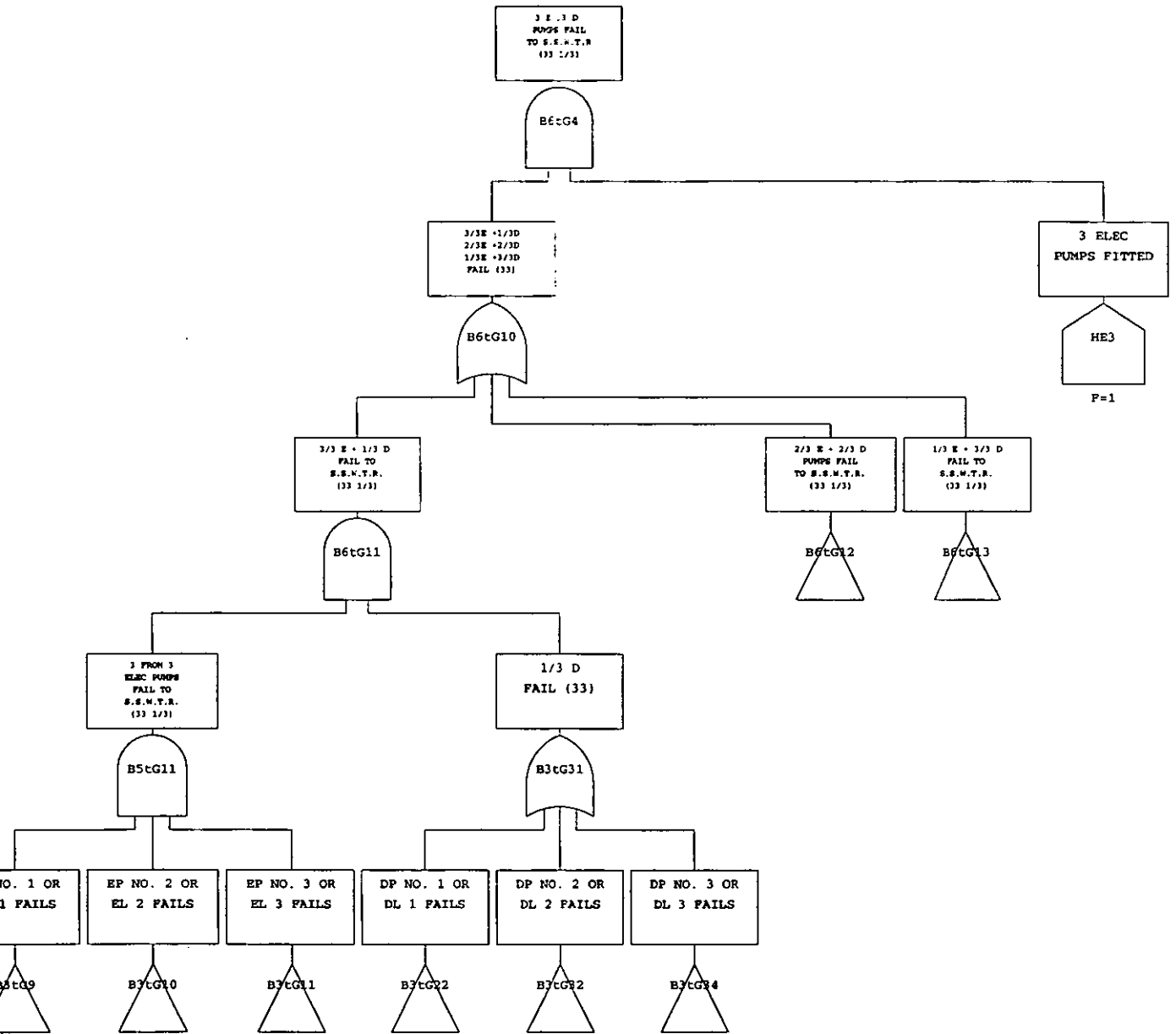


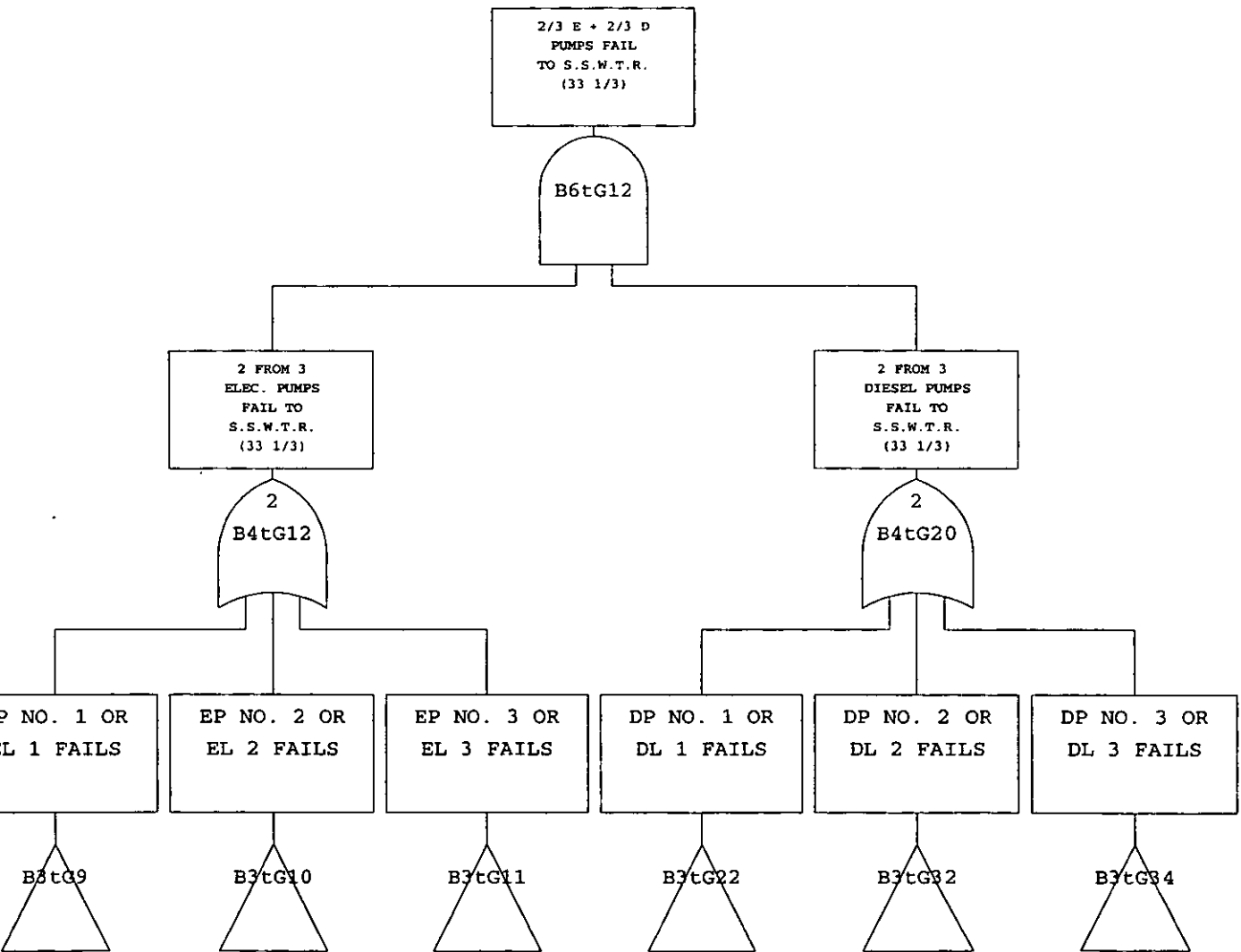


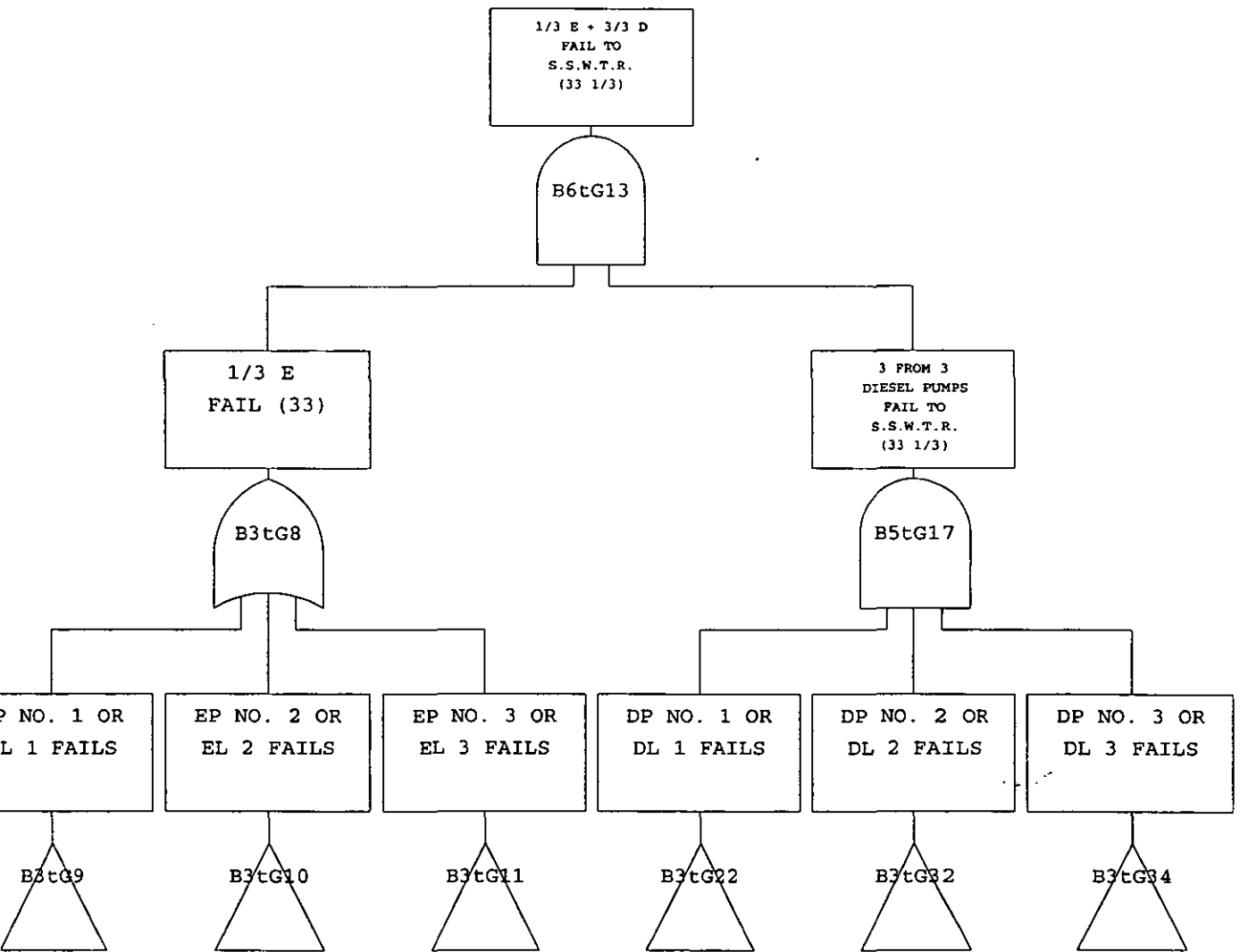


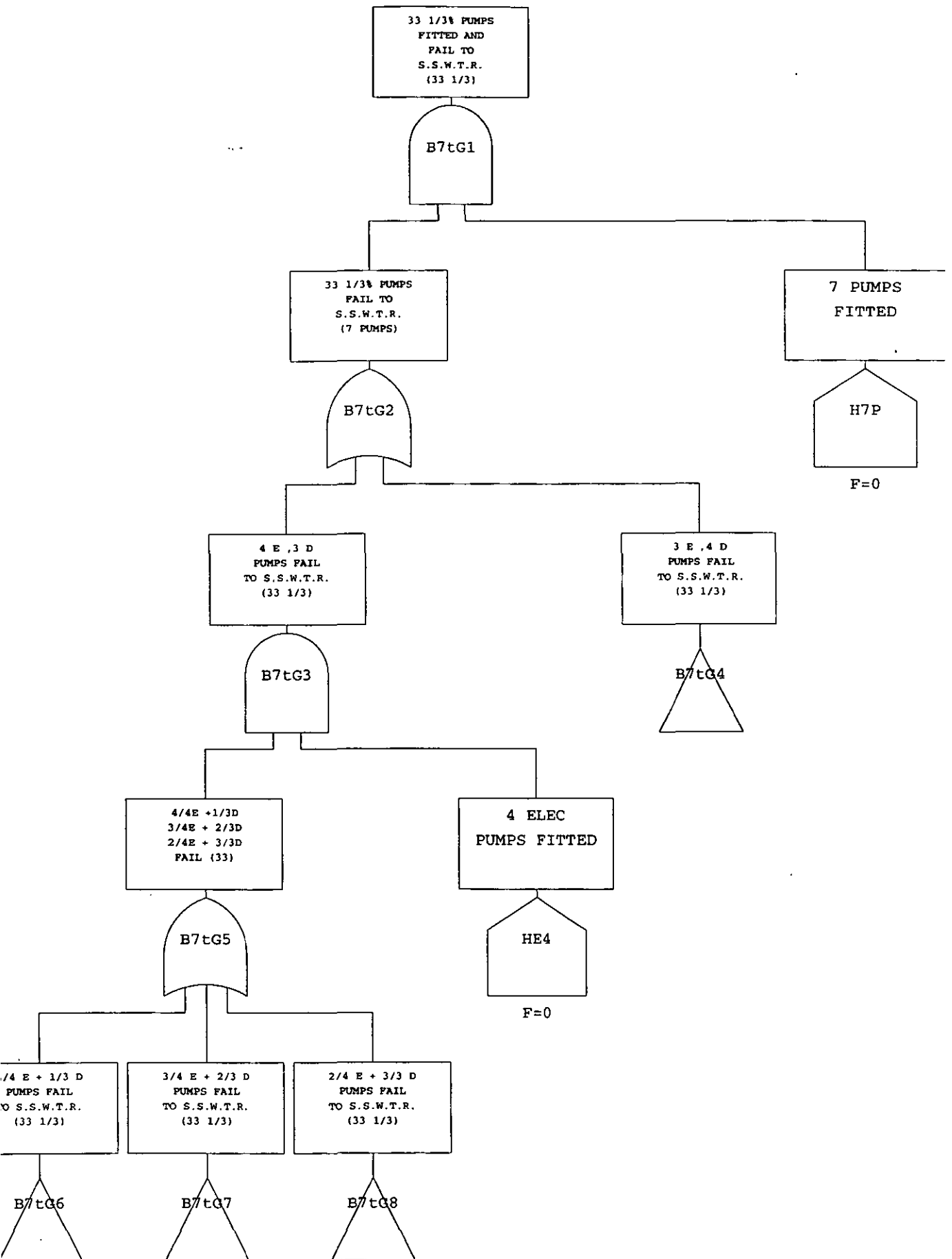


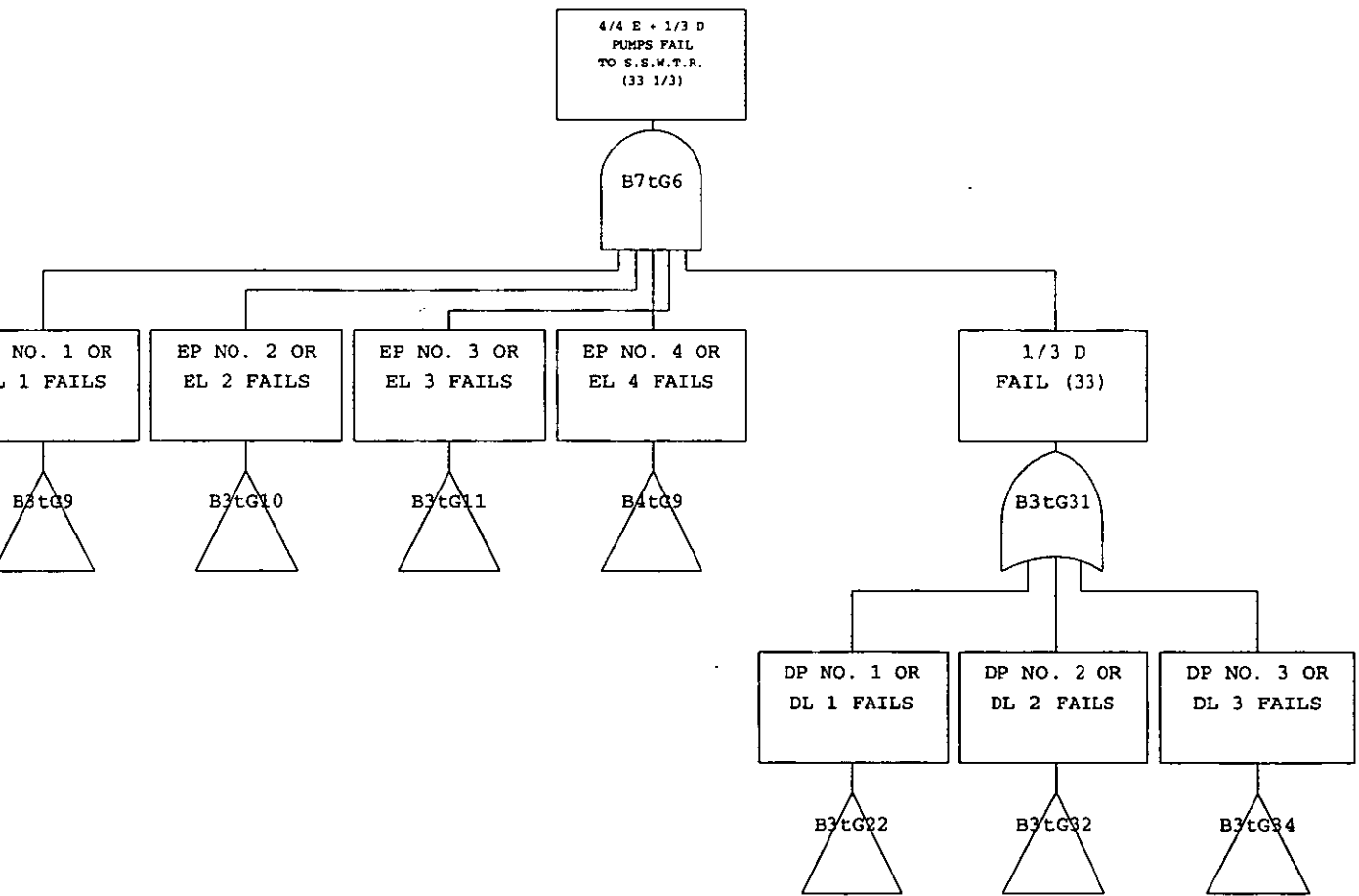


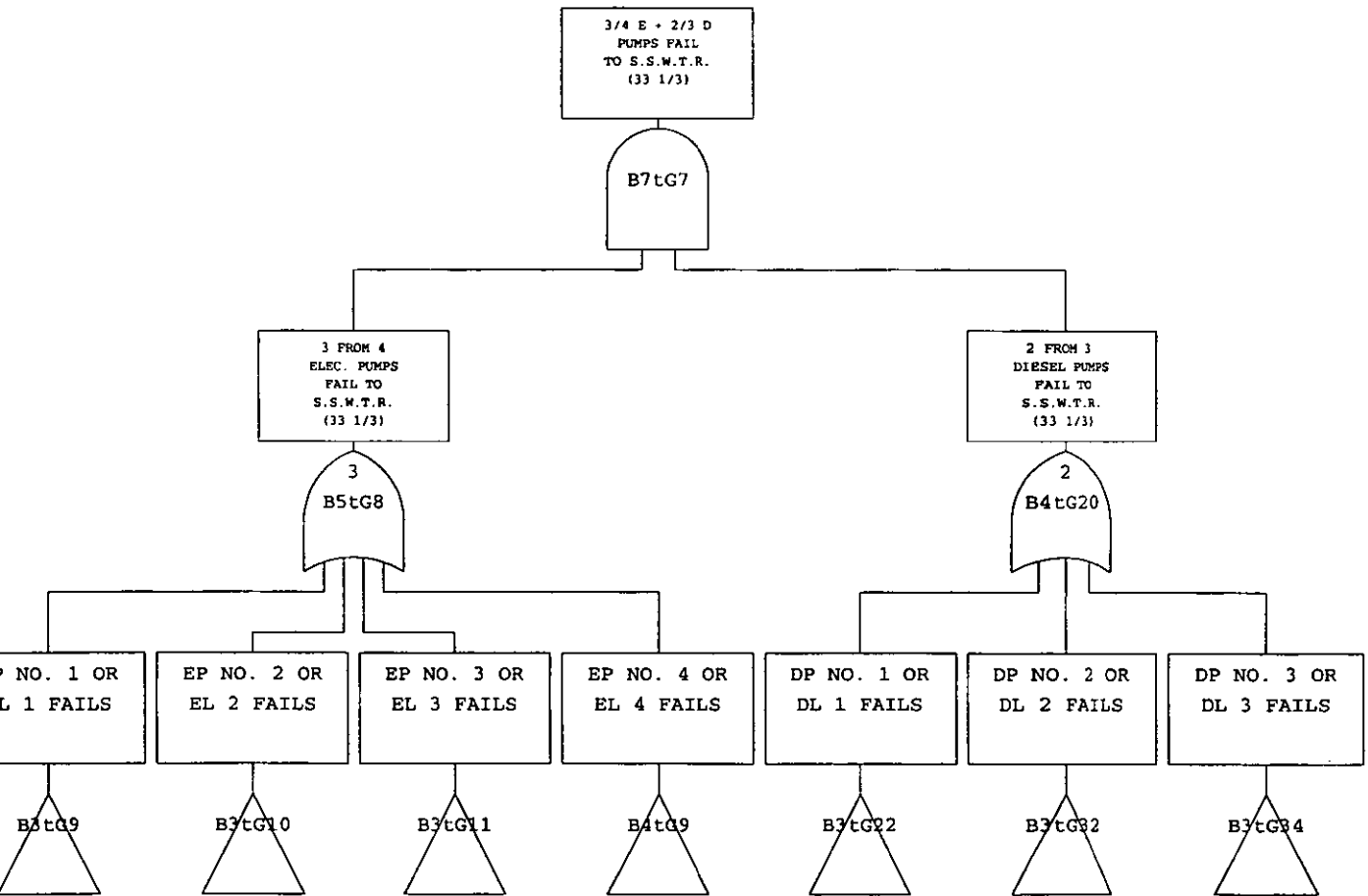


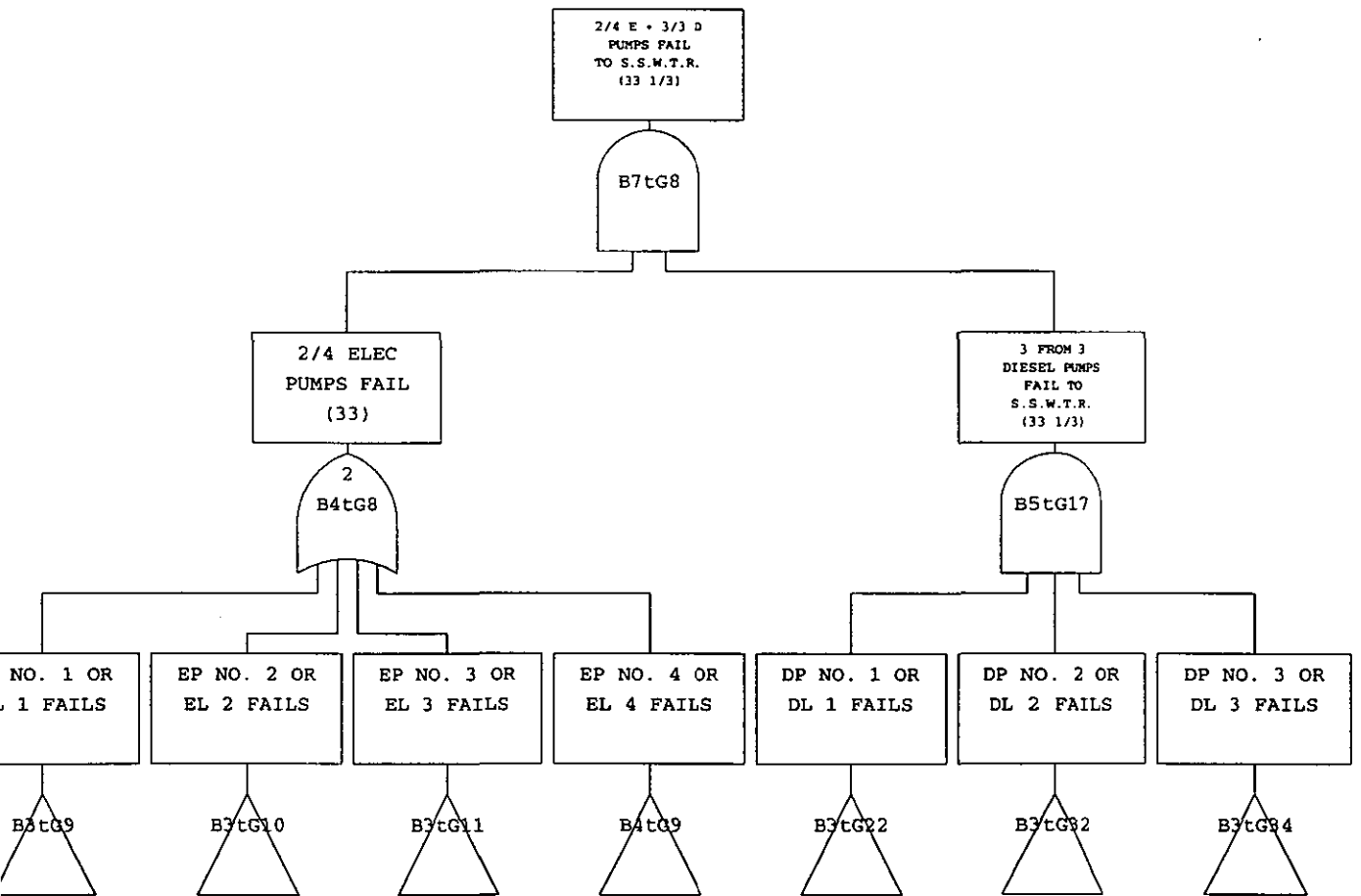


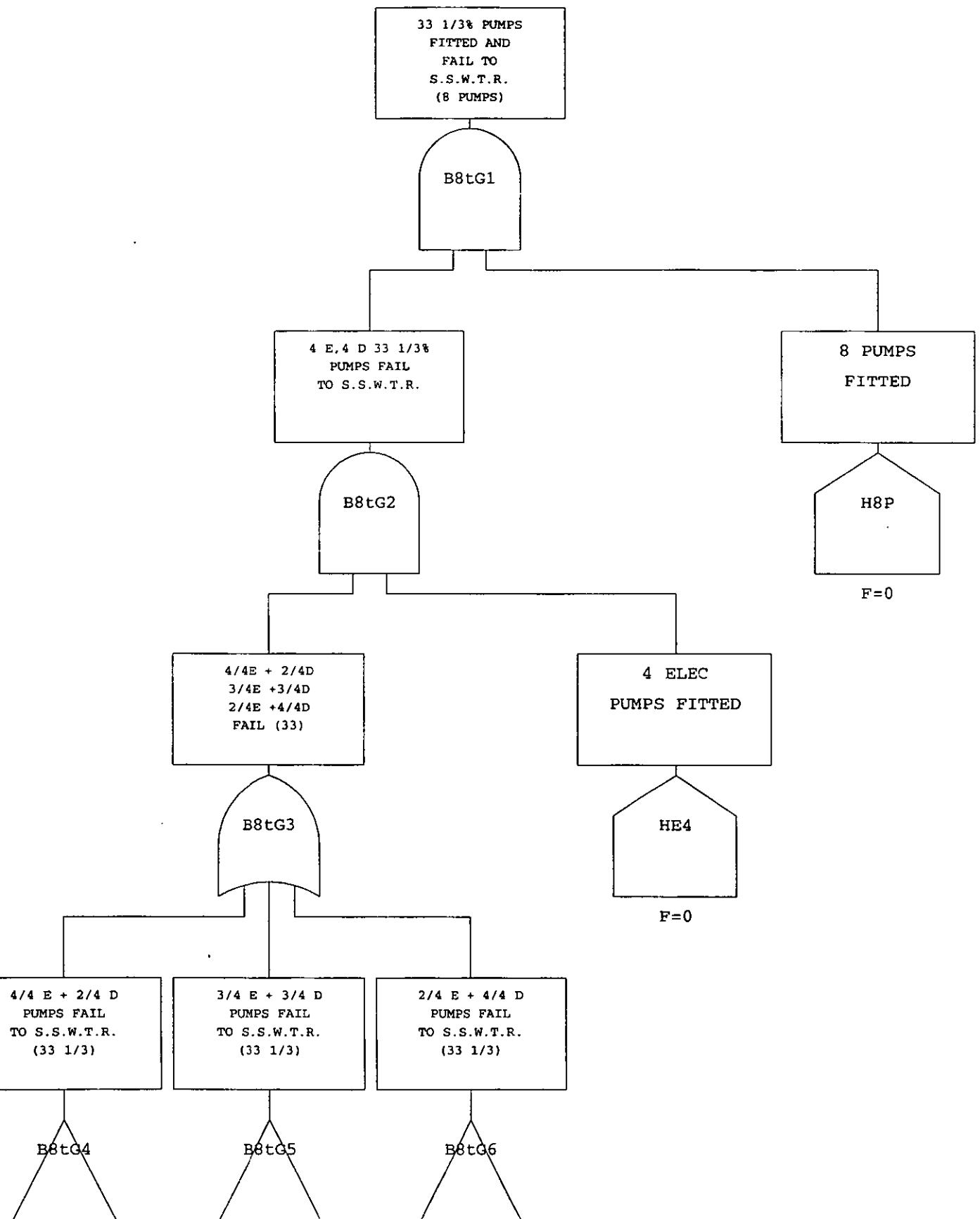


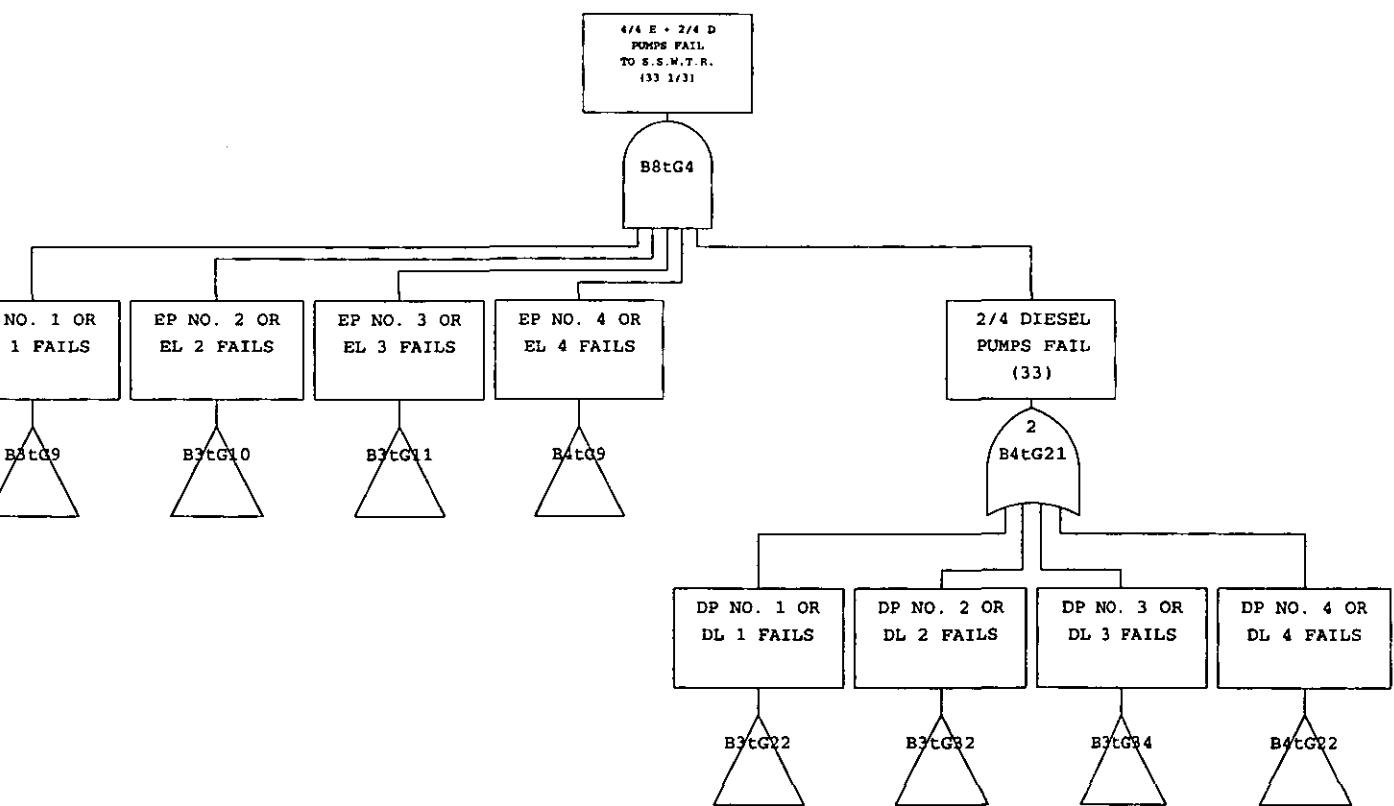


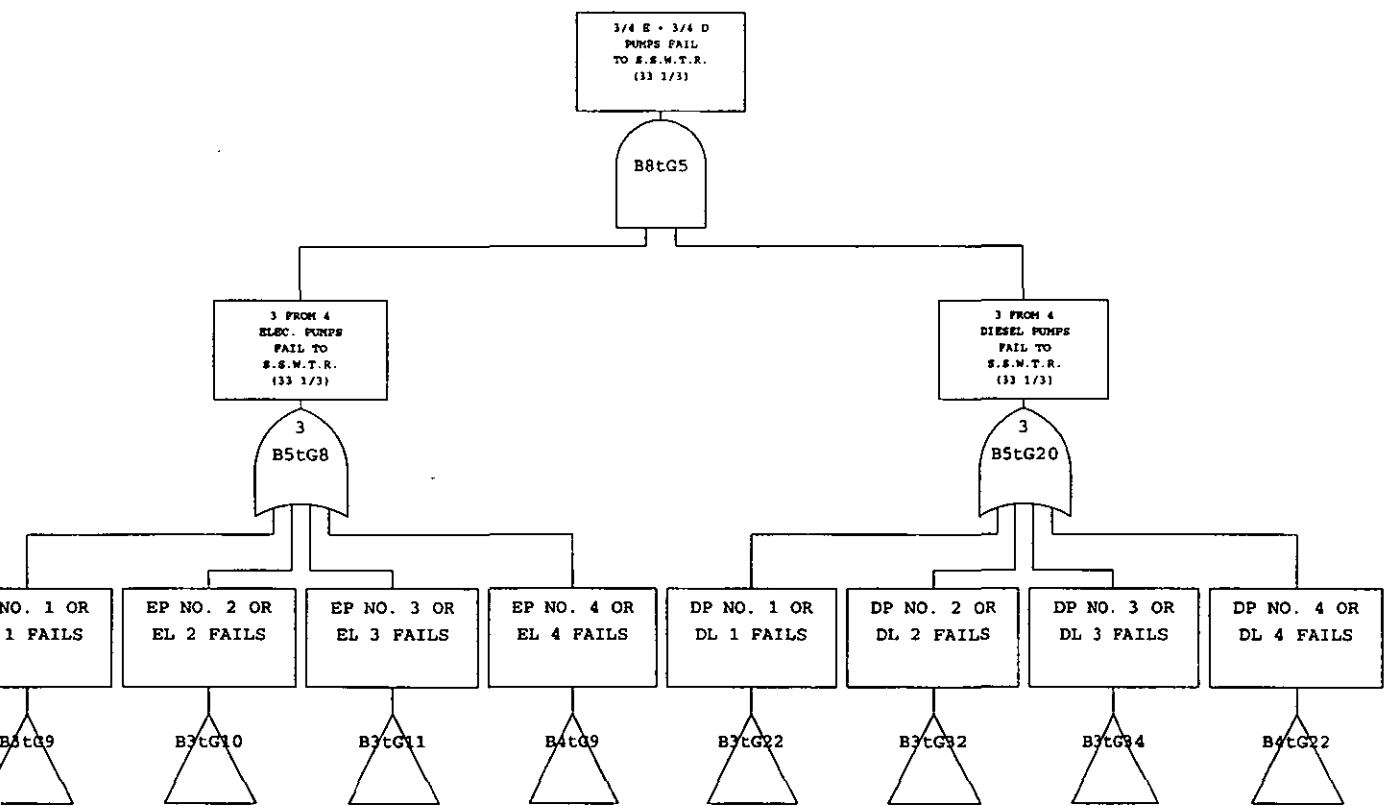


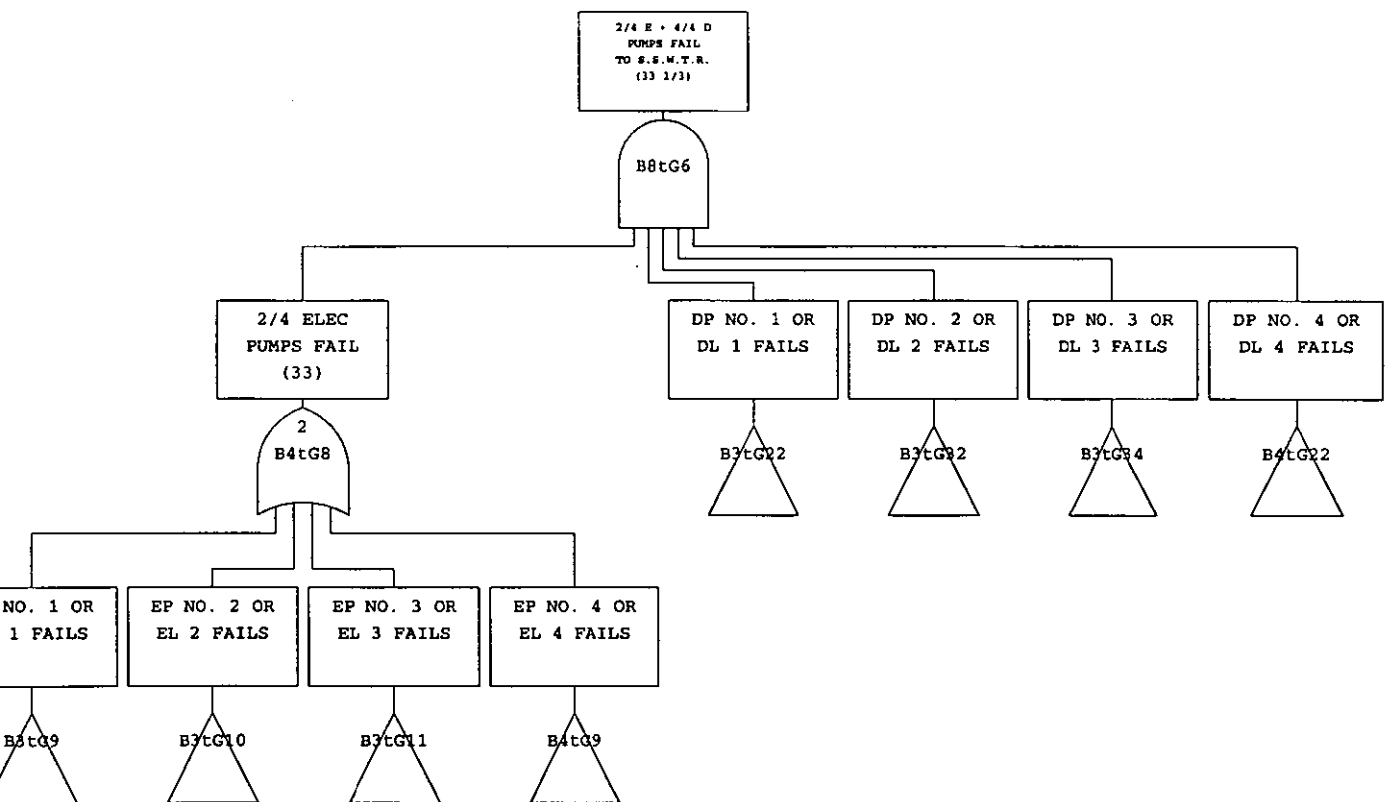


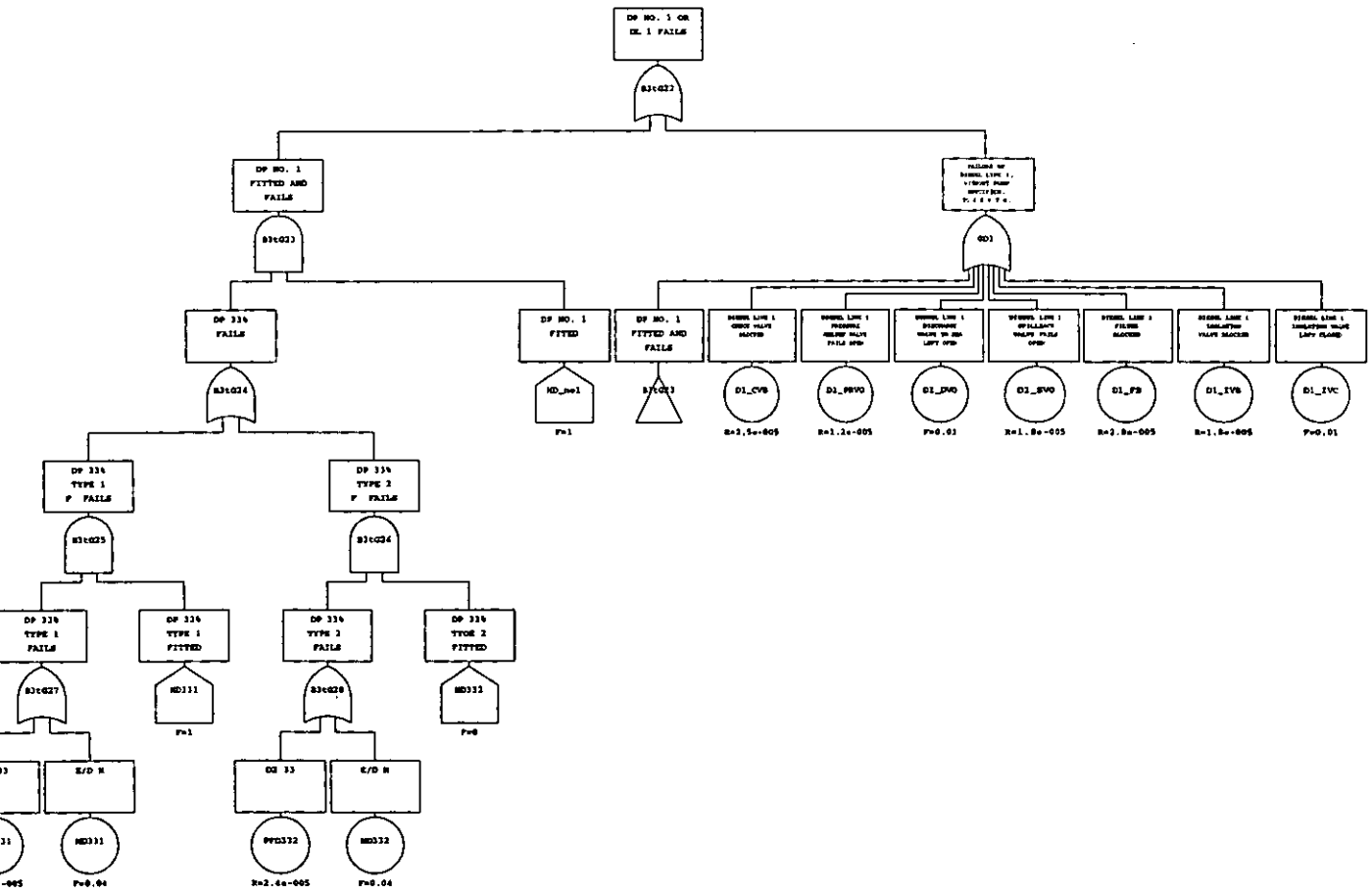


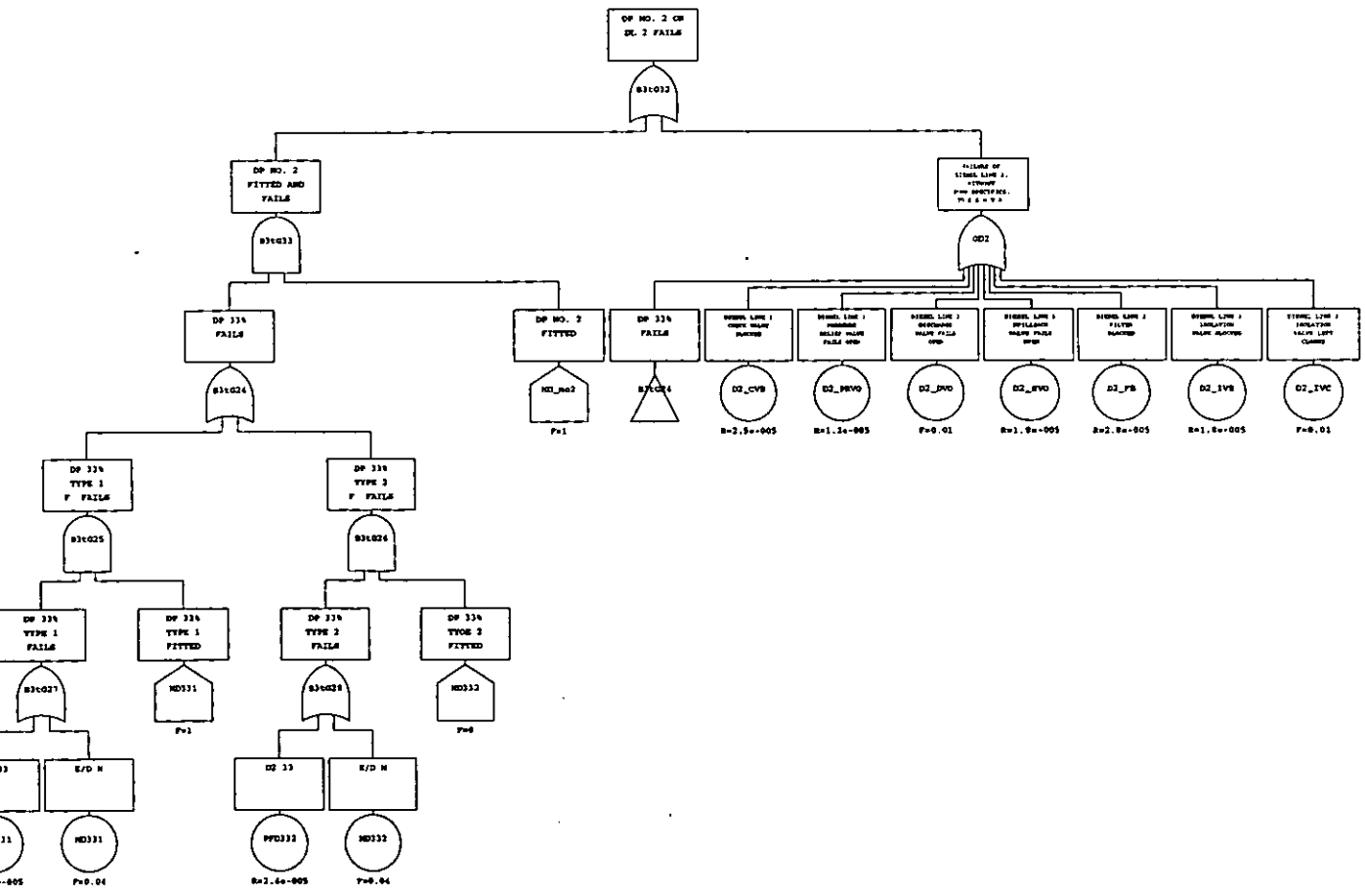


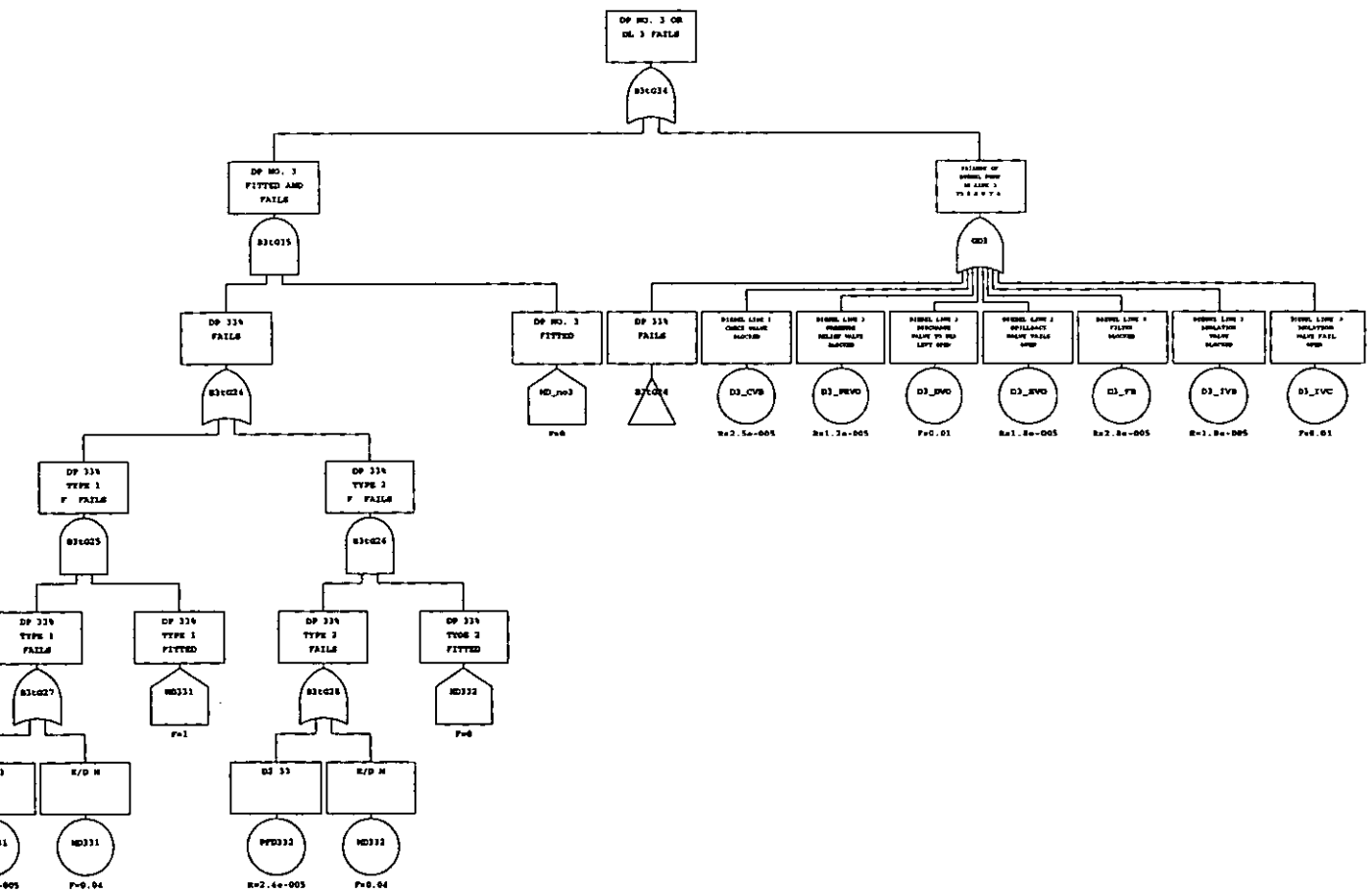


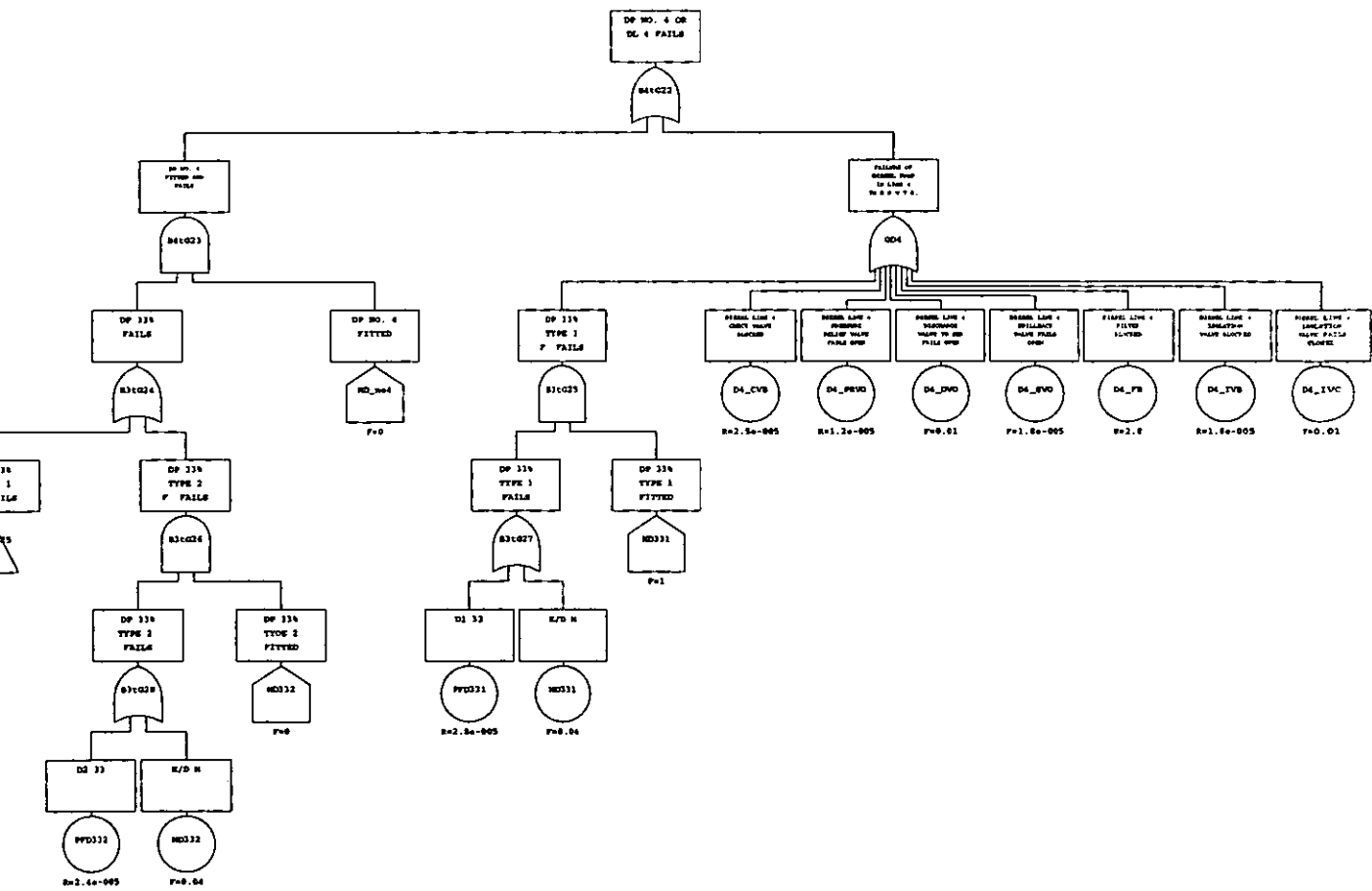


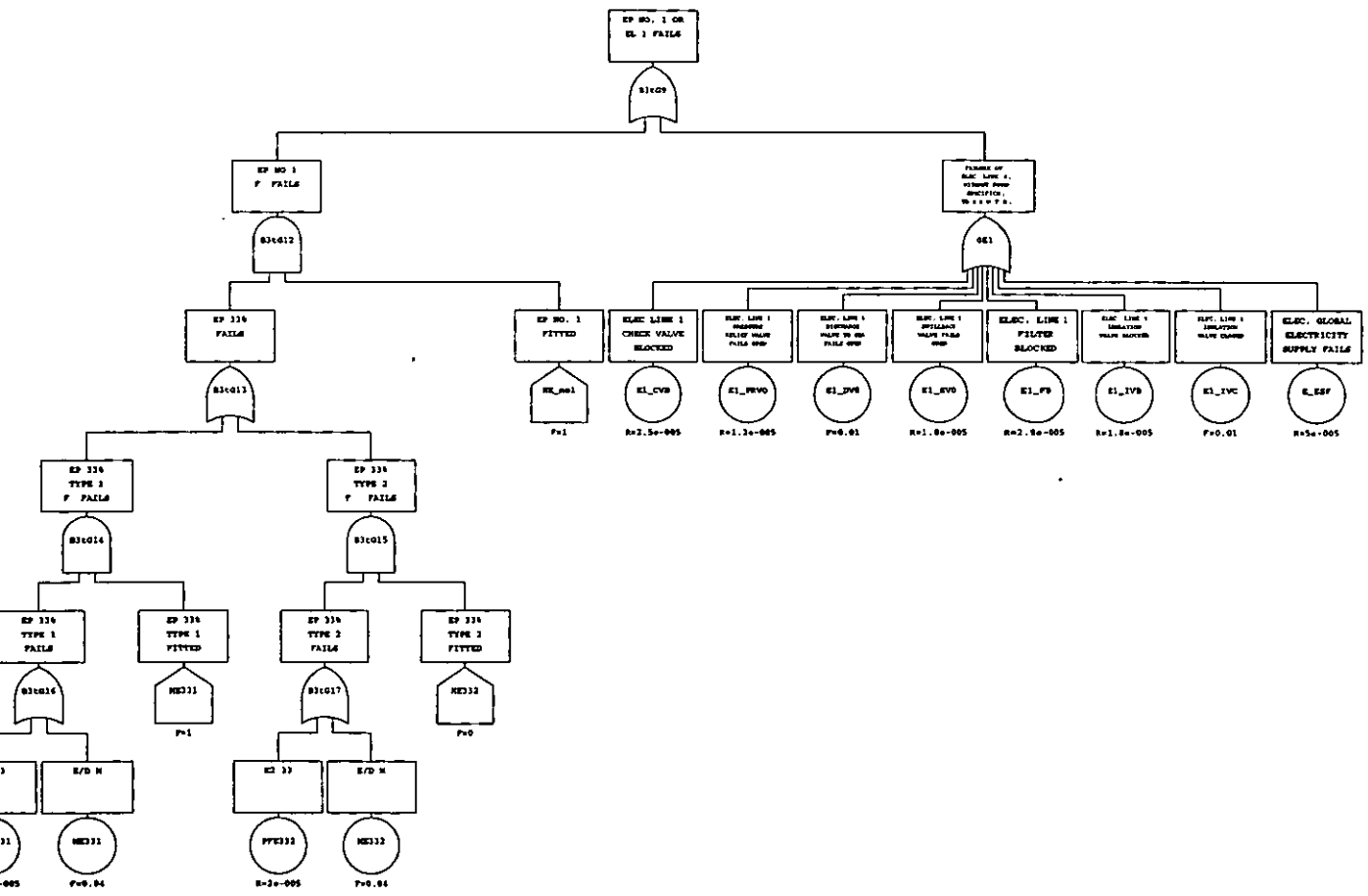


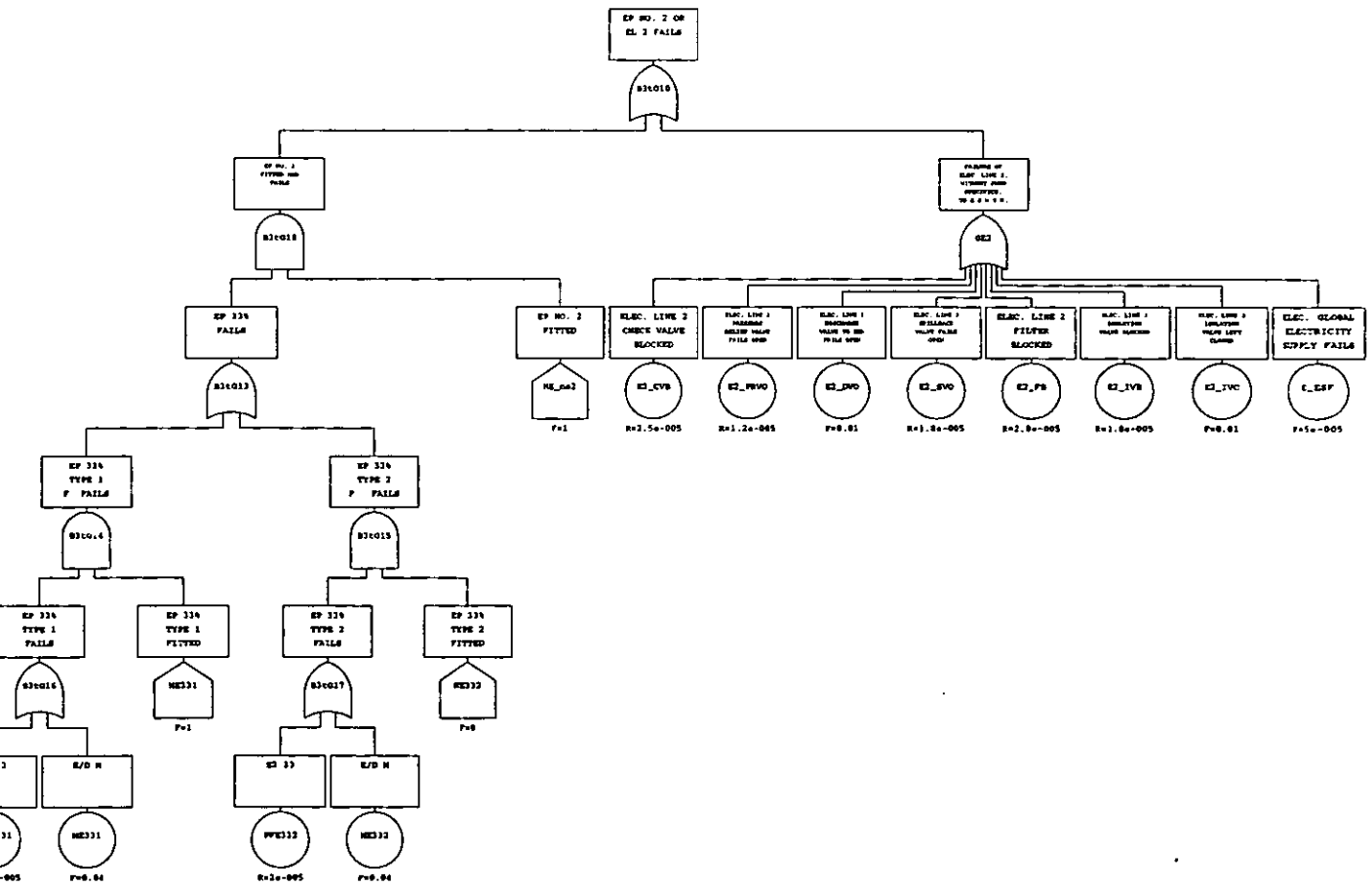


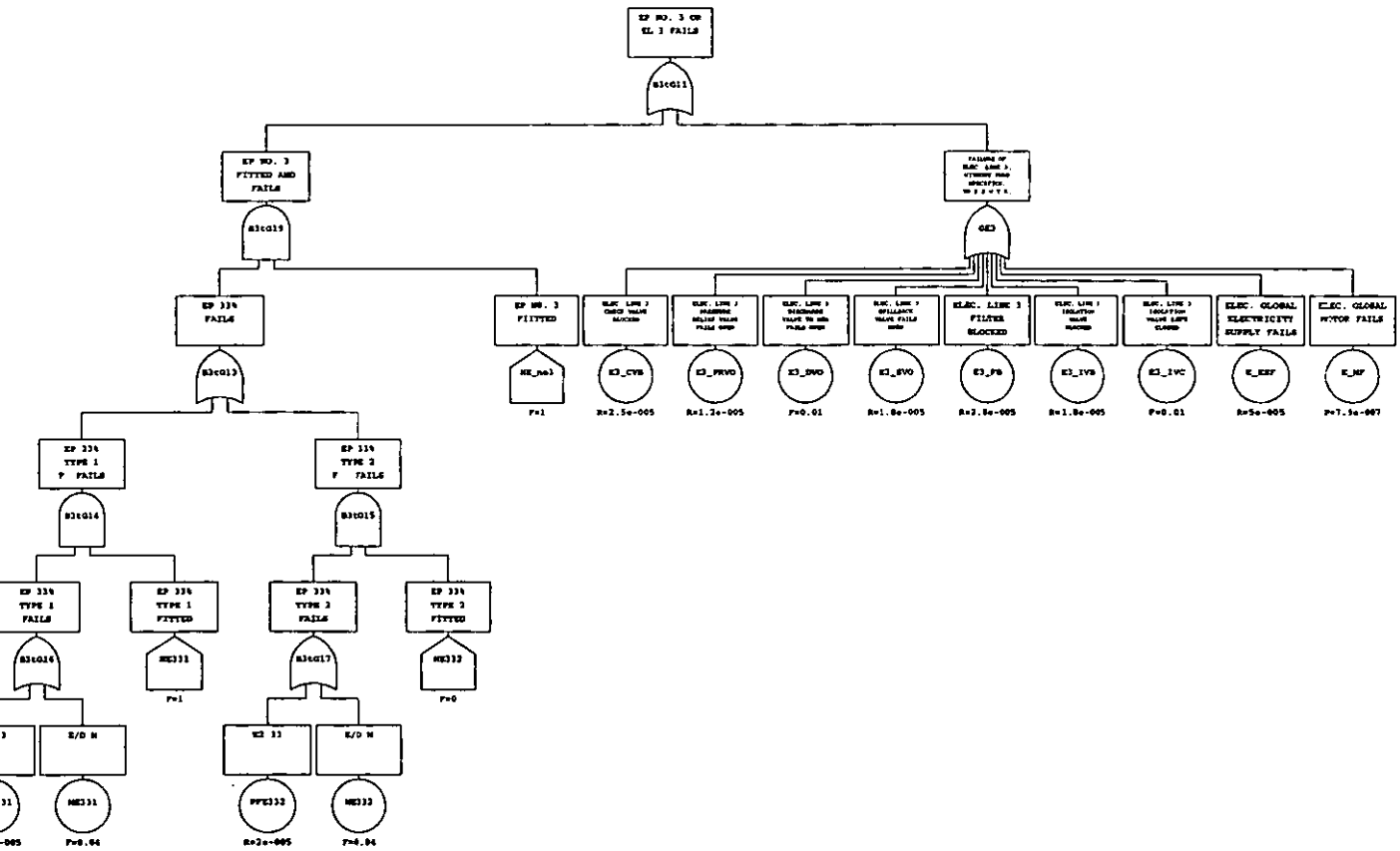


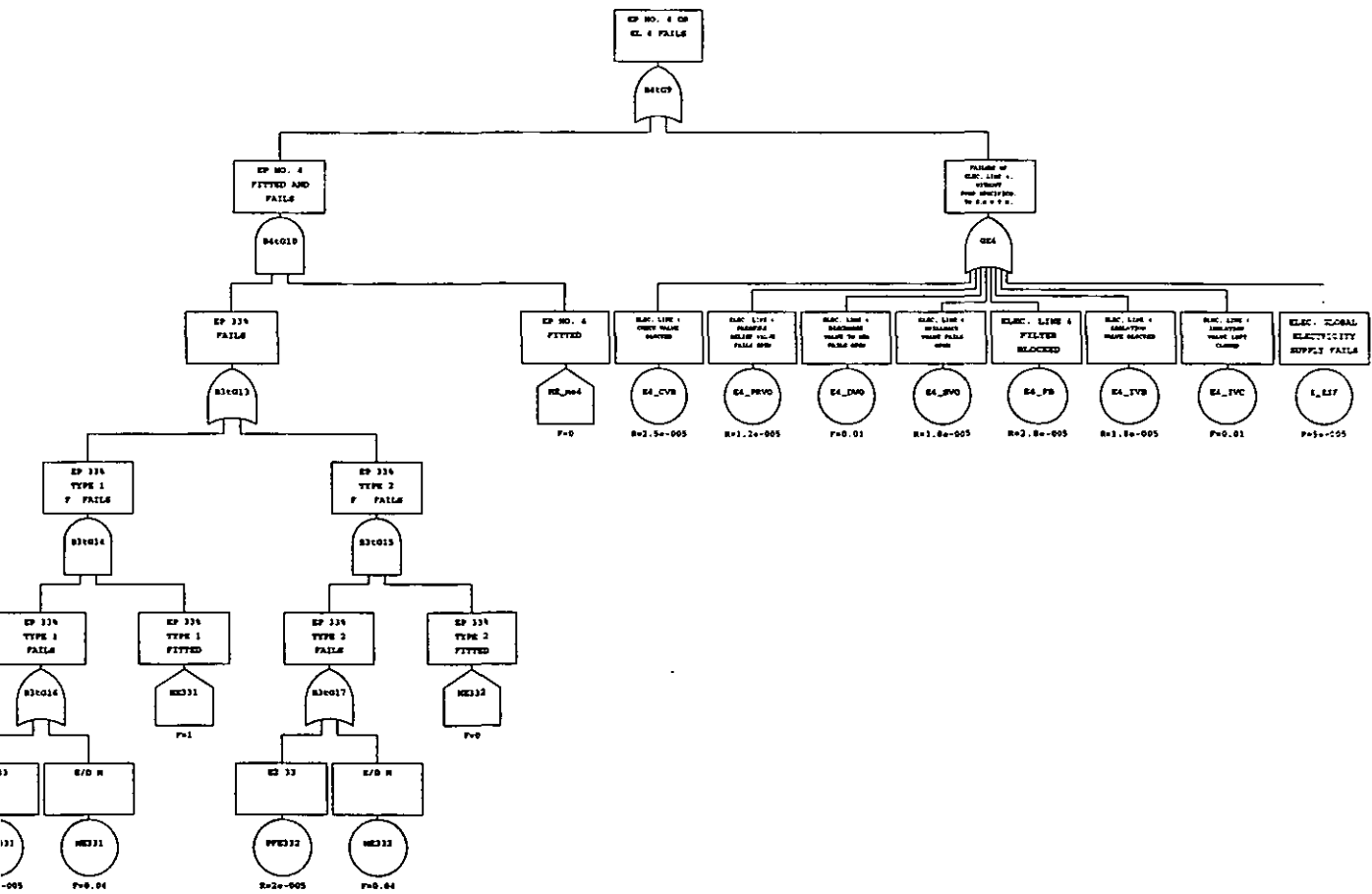


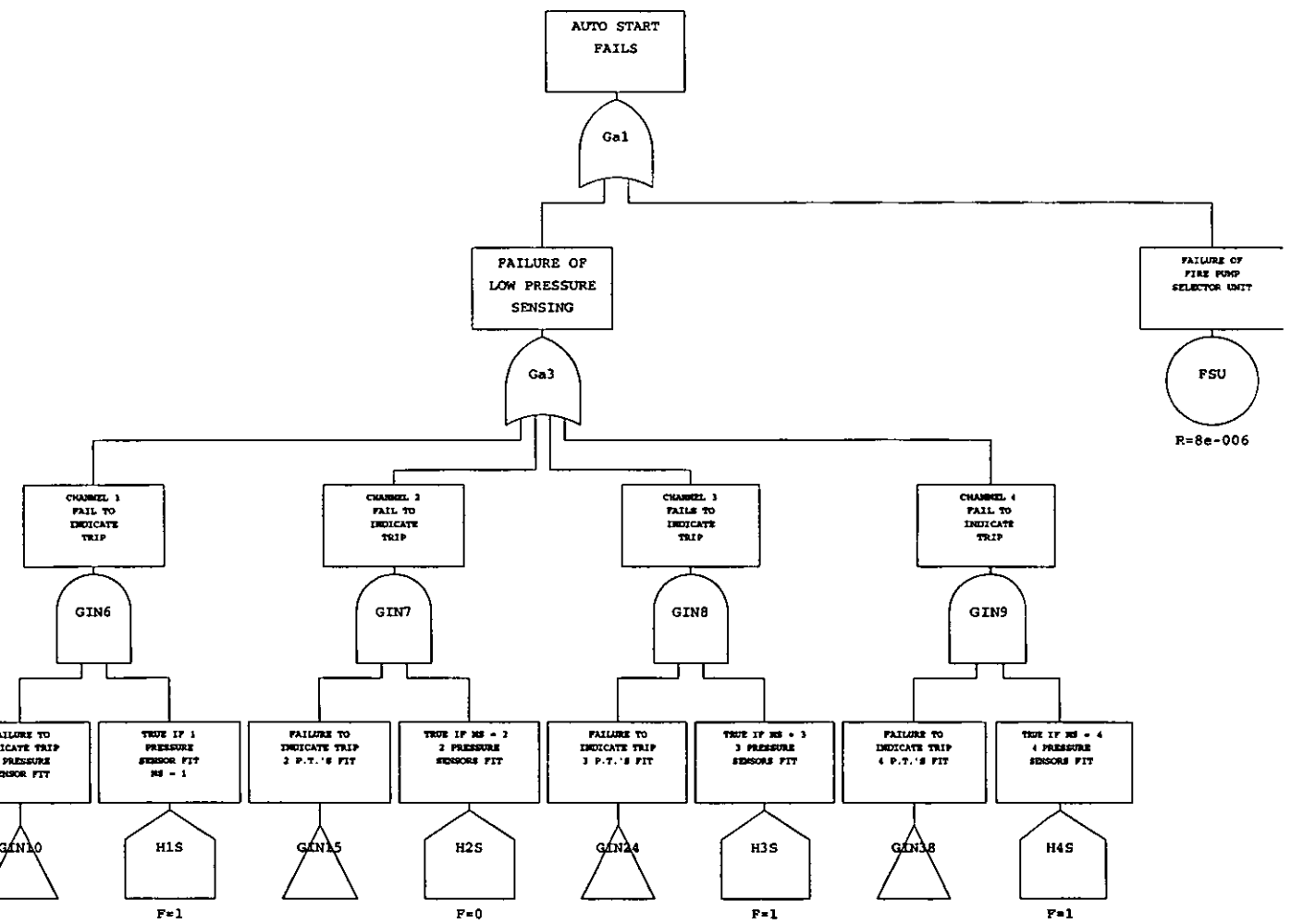


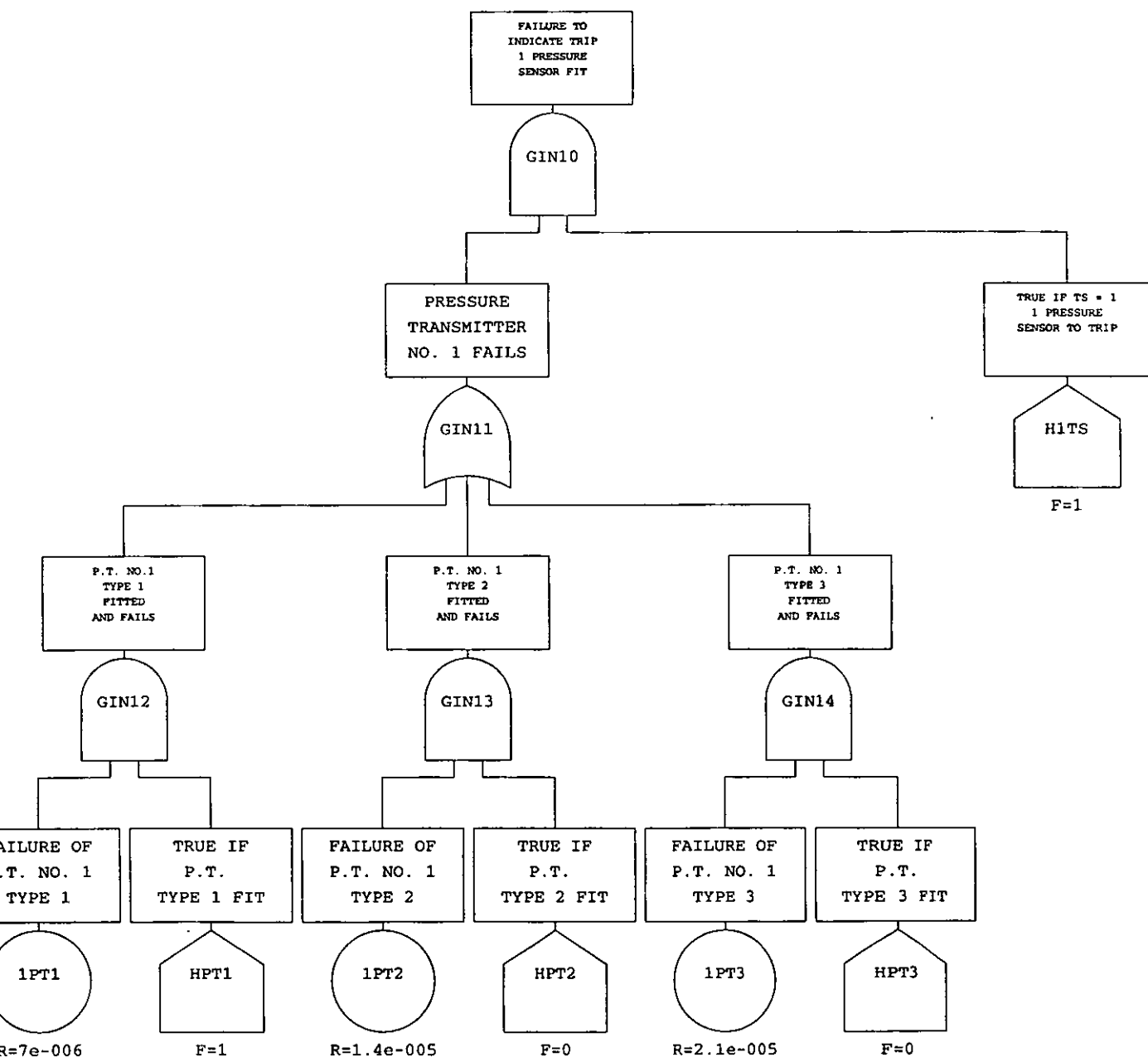


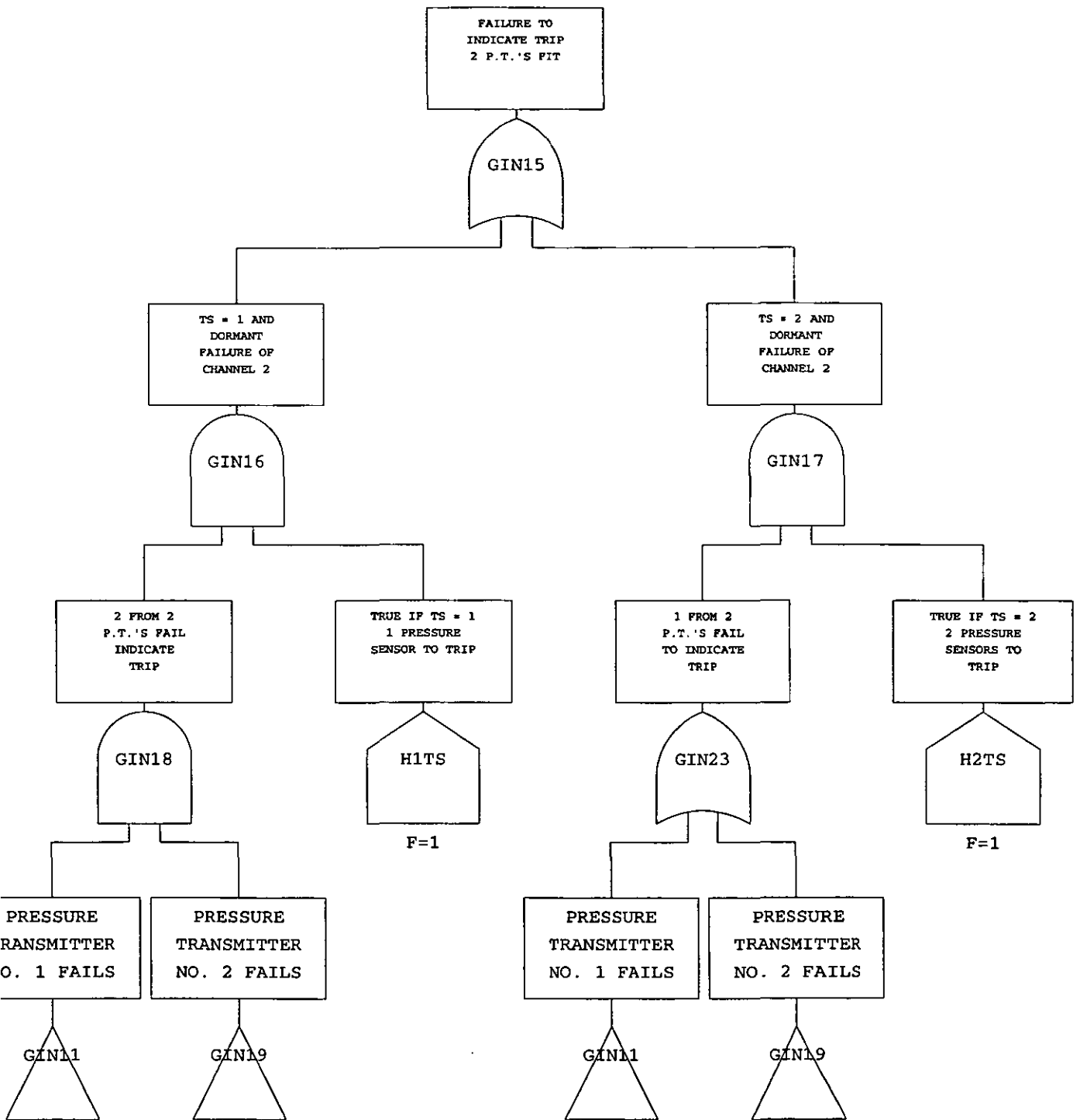


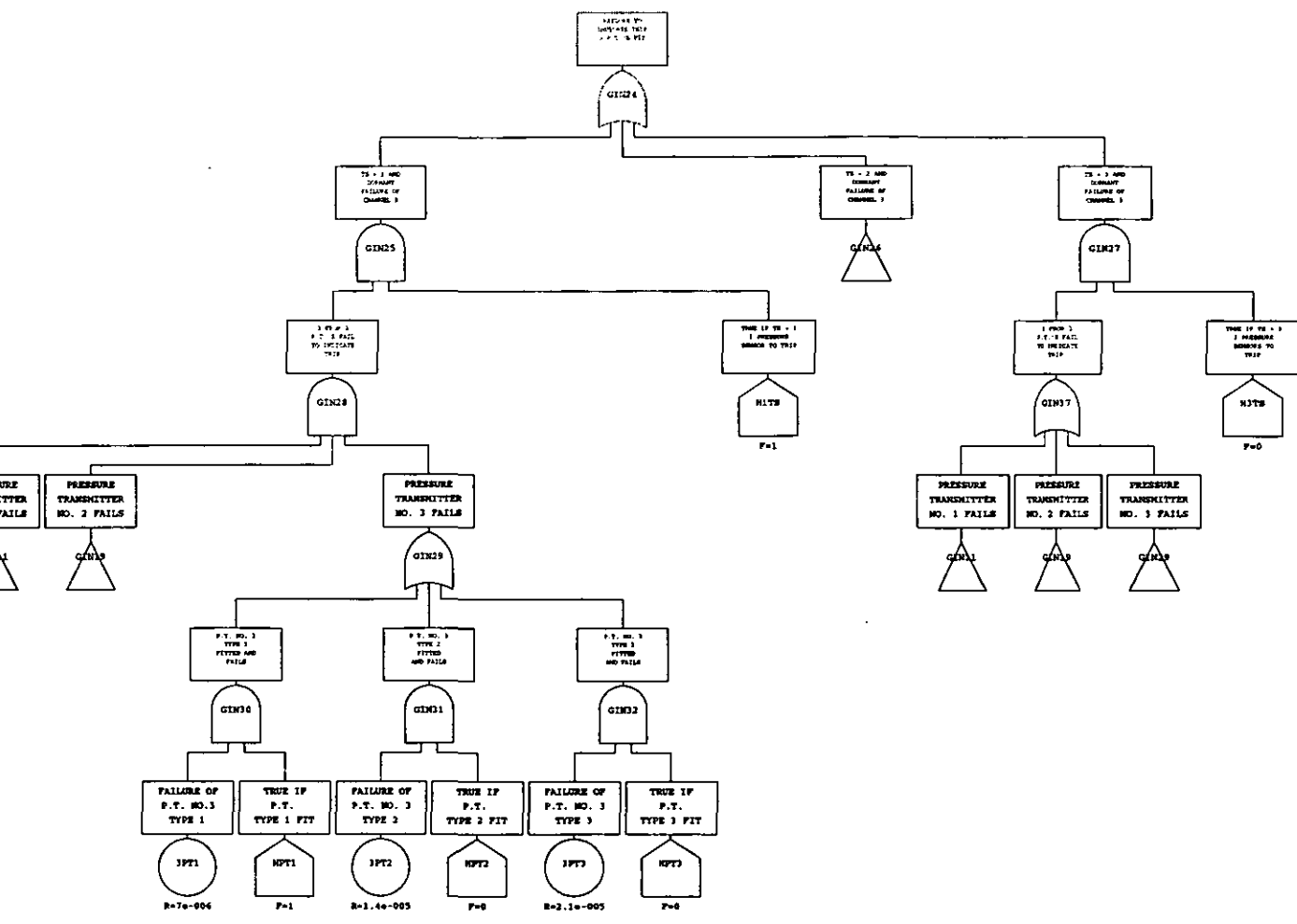


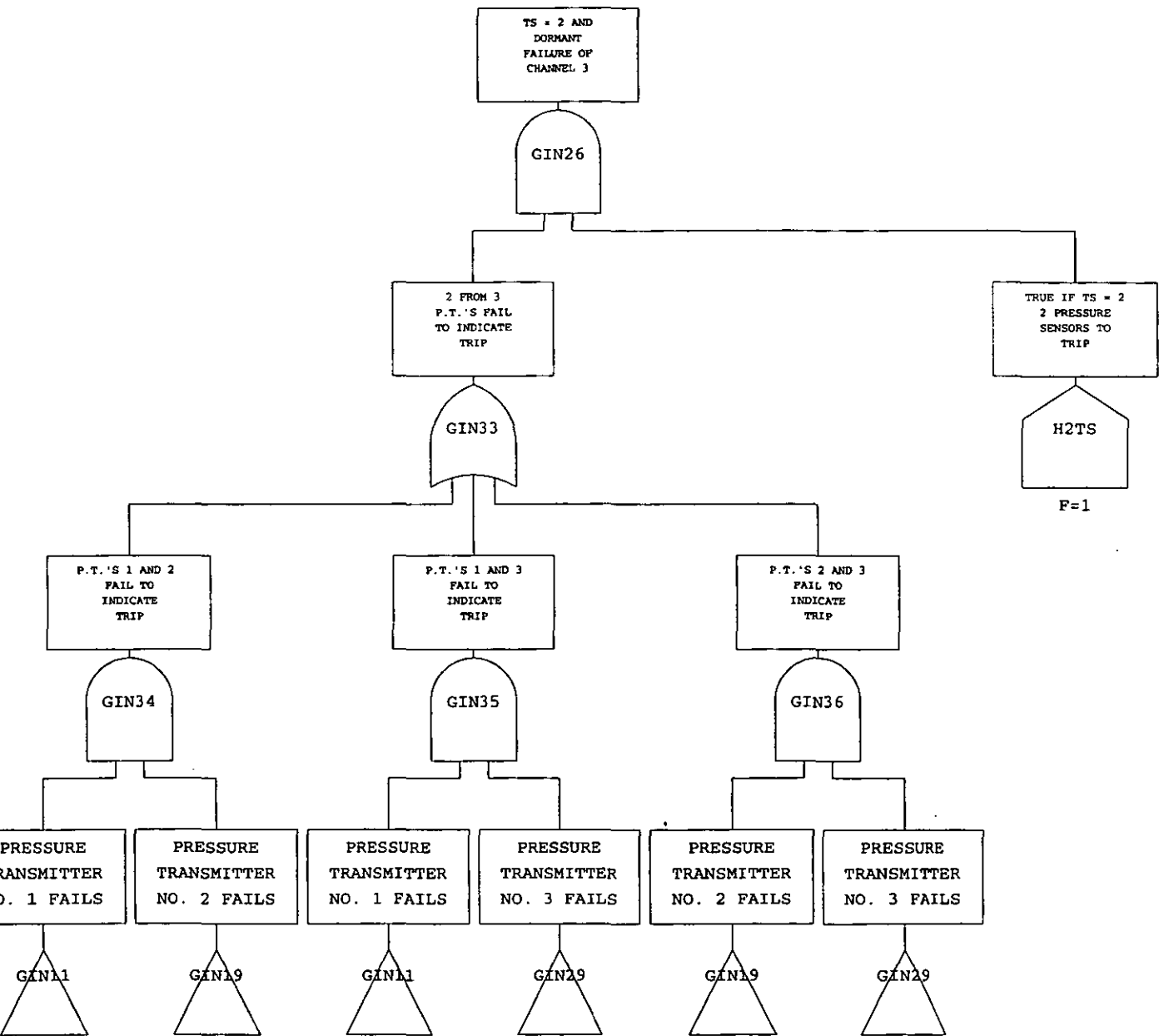


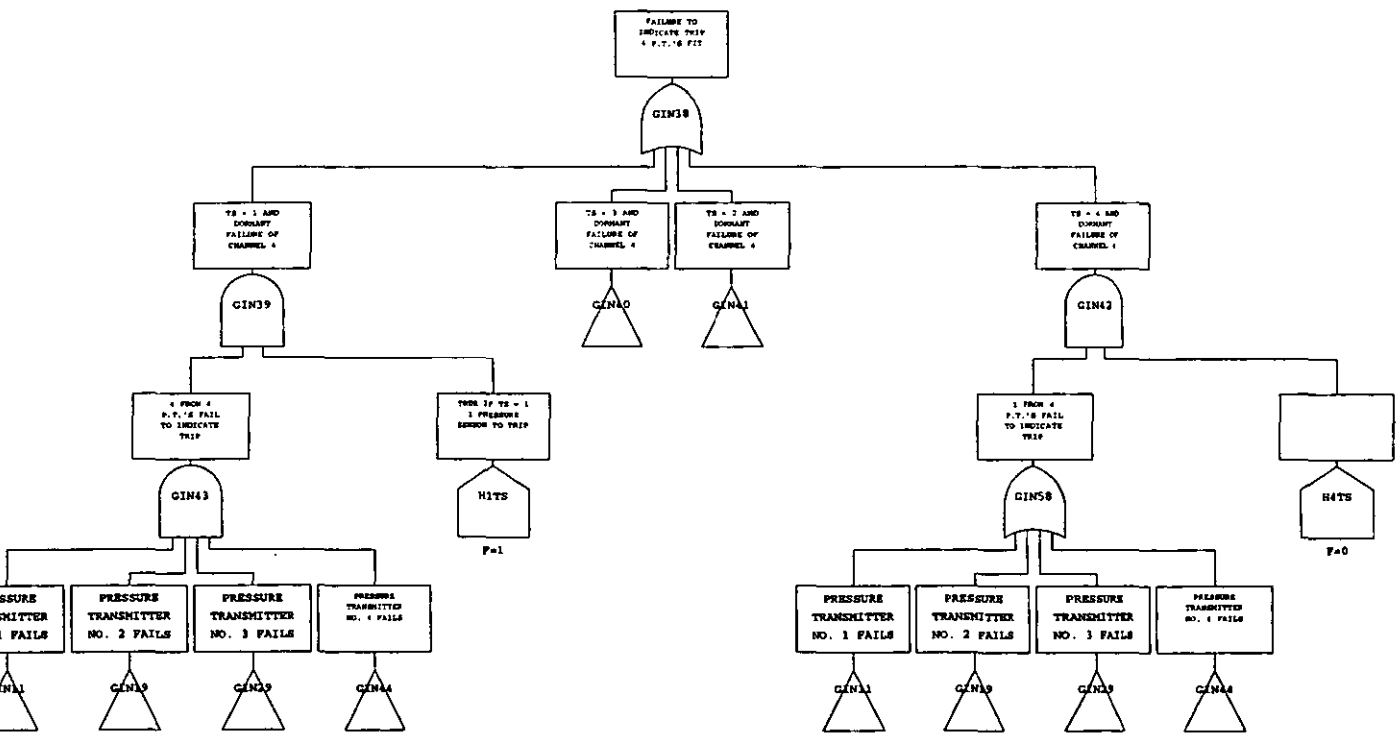


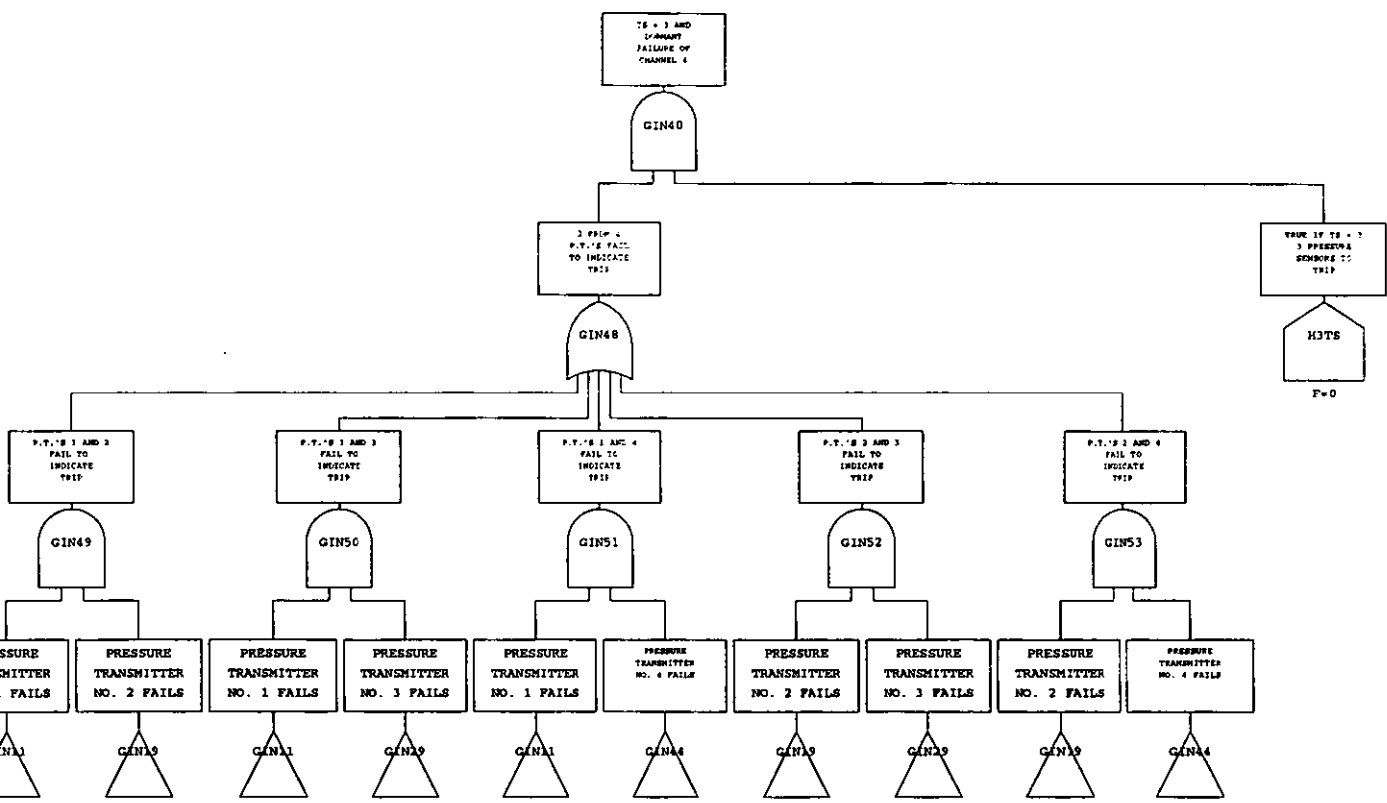


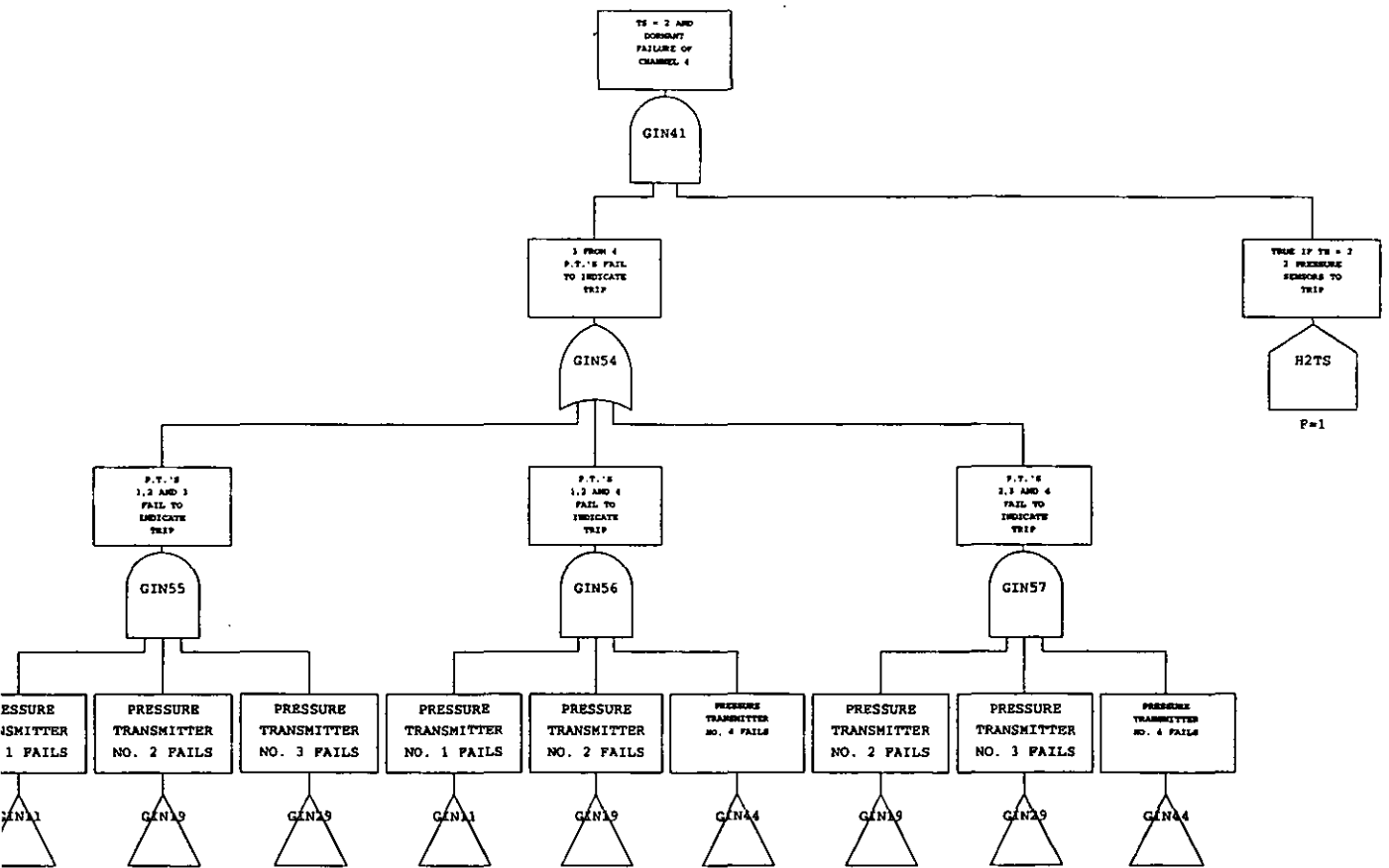


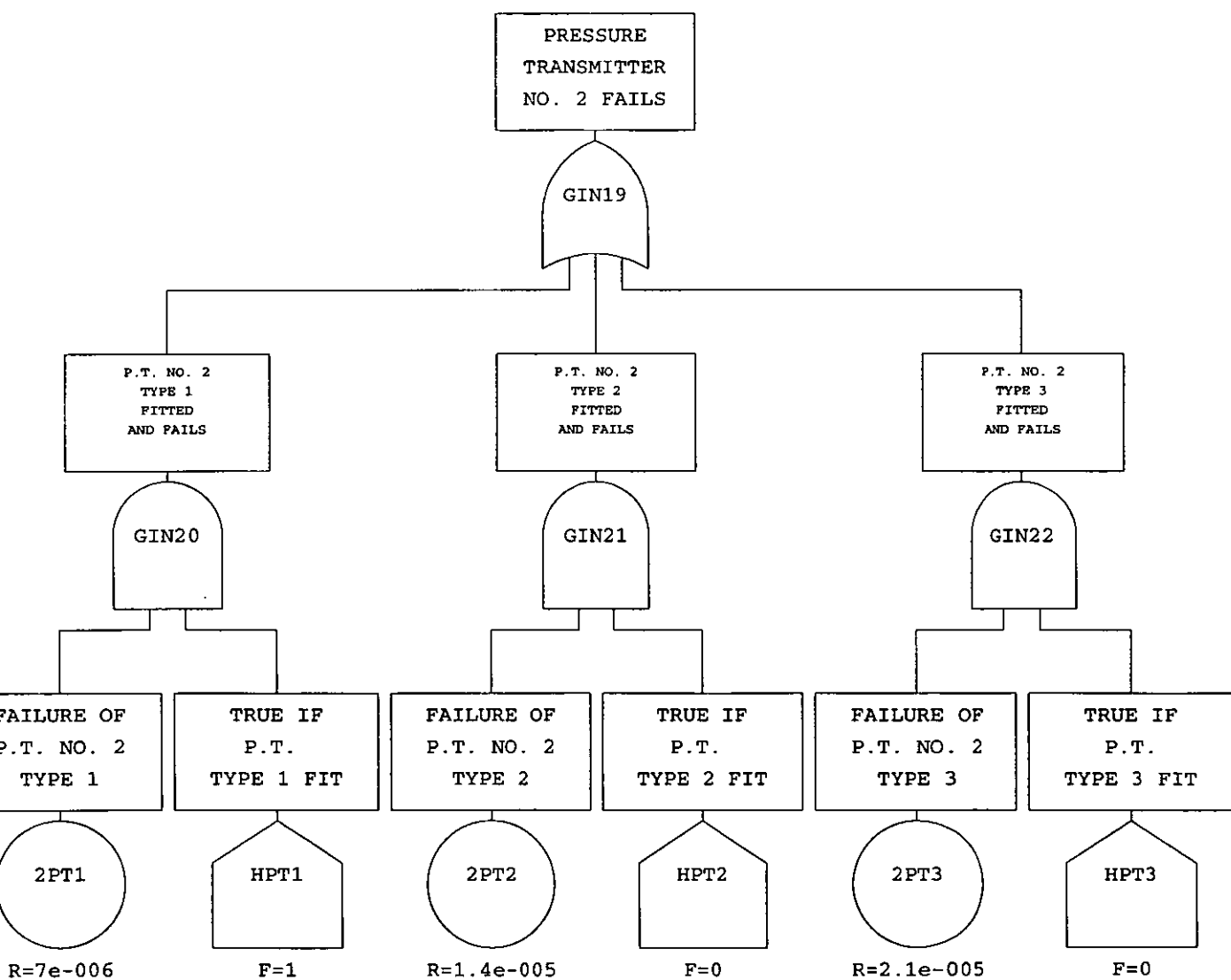












R=7e-006

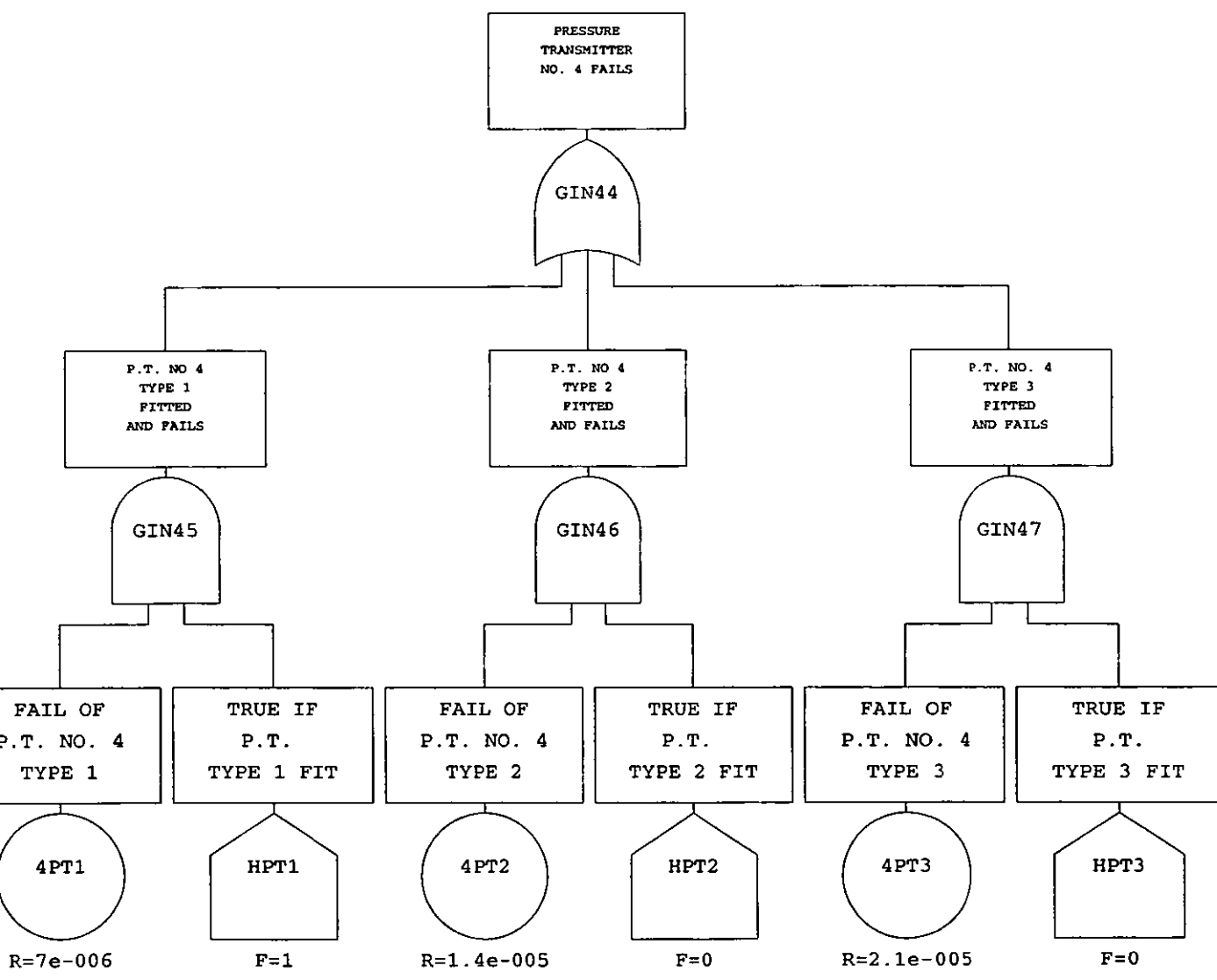
F=1

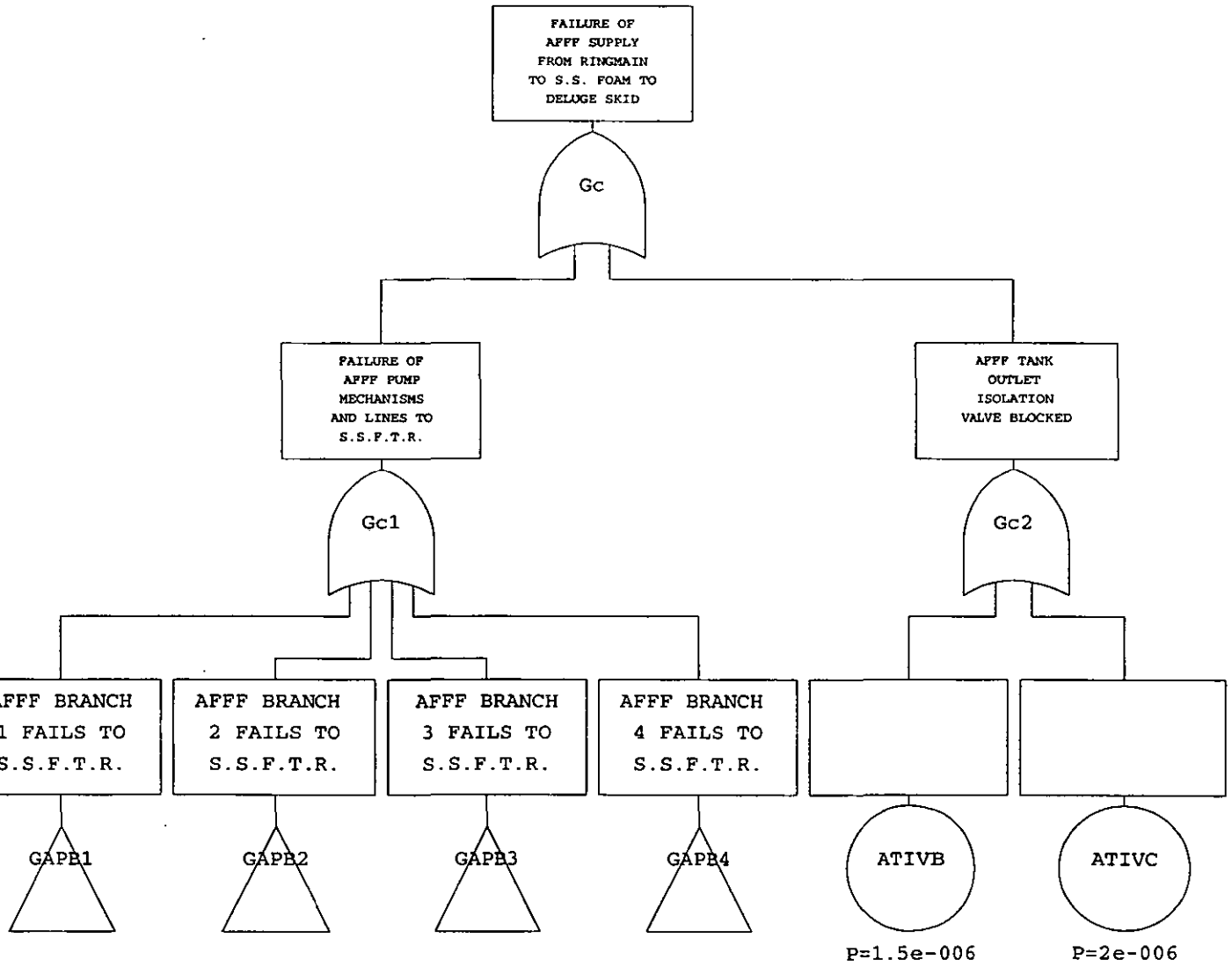
R=1.4e-005

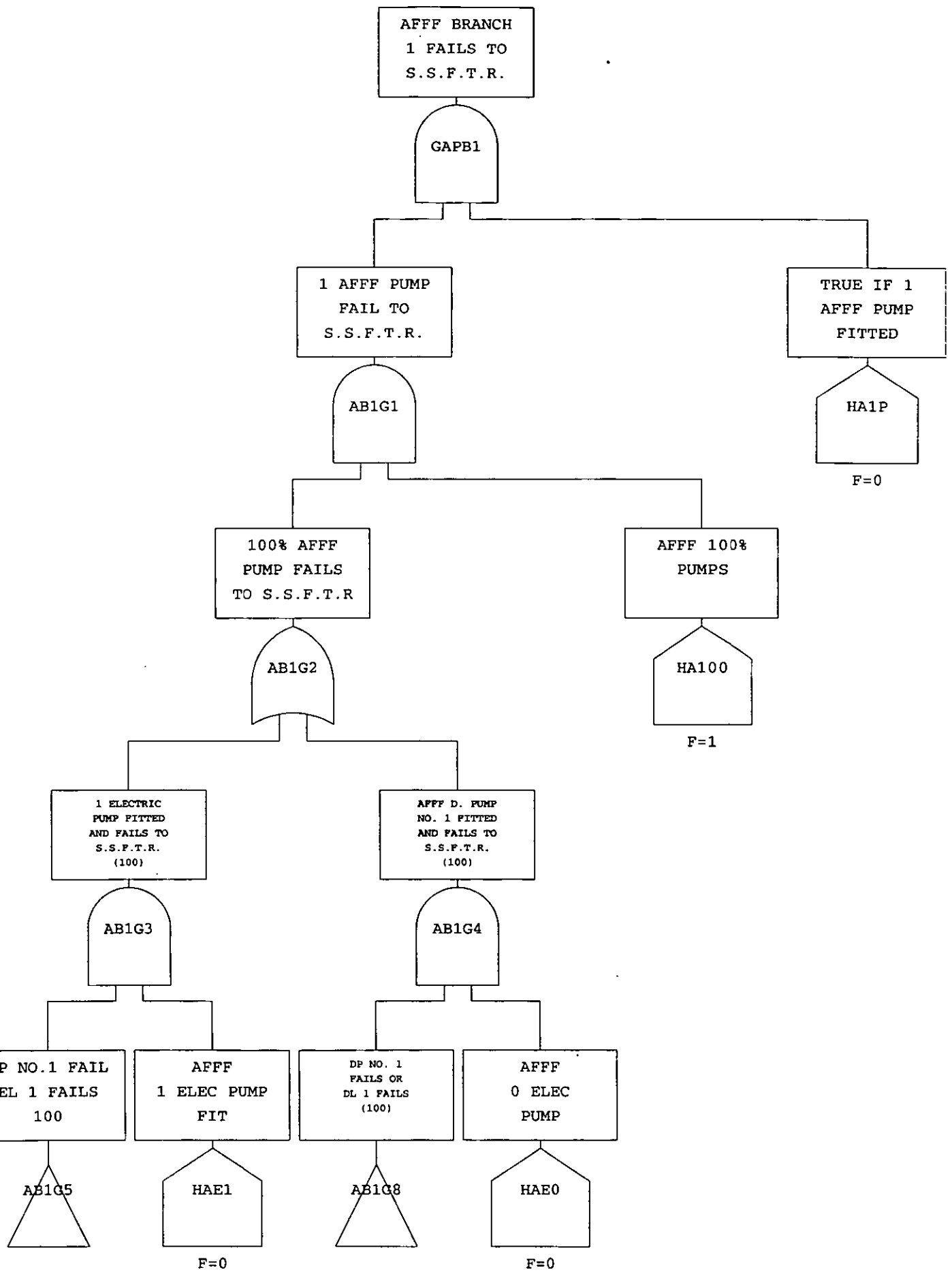
F=0

R=2.1e-005

F=0





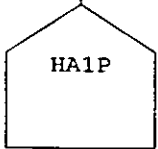
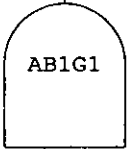


AFFF BRANCH
1 FAILS TO
S.S.F.T.R.



1 AFFF PUMP
FAIL TO
S.S.F.T.R.

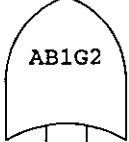
TRUE IF 1
AFFF PUMP
FITTED



F=0

100% AFFF
PUMP FAILS
TO S.S.F.T.R.

AFFF 100%
PUMPS



F=1

1 ELECTRIC
PUMP FITTED
AND FAILS TO
S.S.F.T.R.
(100)

AFFF D. PUMP
NO. 1 FITTED
AND FAILS TO
S.S.F.T.R.
(100)

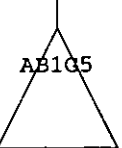


P NO.1 FAIL
EL 1 FAILS
100

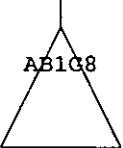
AFFF
1 ELEC PUMP
FIT

DP NO. 1
FAILS OR
DL 1 FAILS
(100)

AFFF
0 ELEC
PUMP



F=0



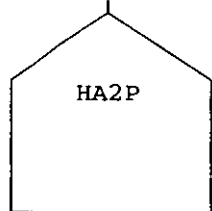
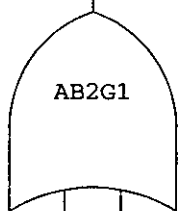
F=0

AFFF BRANCH
2 FAILS TO
S.S.F.T.R.



2 AFFF PUMPS
FAIL TO
S.S.F.T.R.

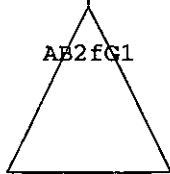
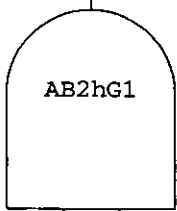
TRUE IF 2
AFFF PUMPS
FITTED



F=0

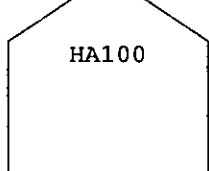
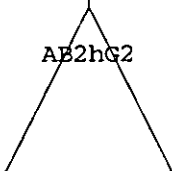
AFFF 100%
PUMPS FITTED
AND FAIL TO
S.S.F.T.R.
(2 PUMPS)

AFFF 50%
PUMPS FITTED
AND FAIL TO
S.S.F.T.R.
(2 PUMPS)

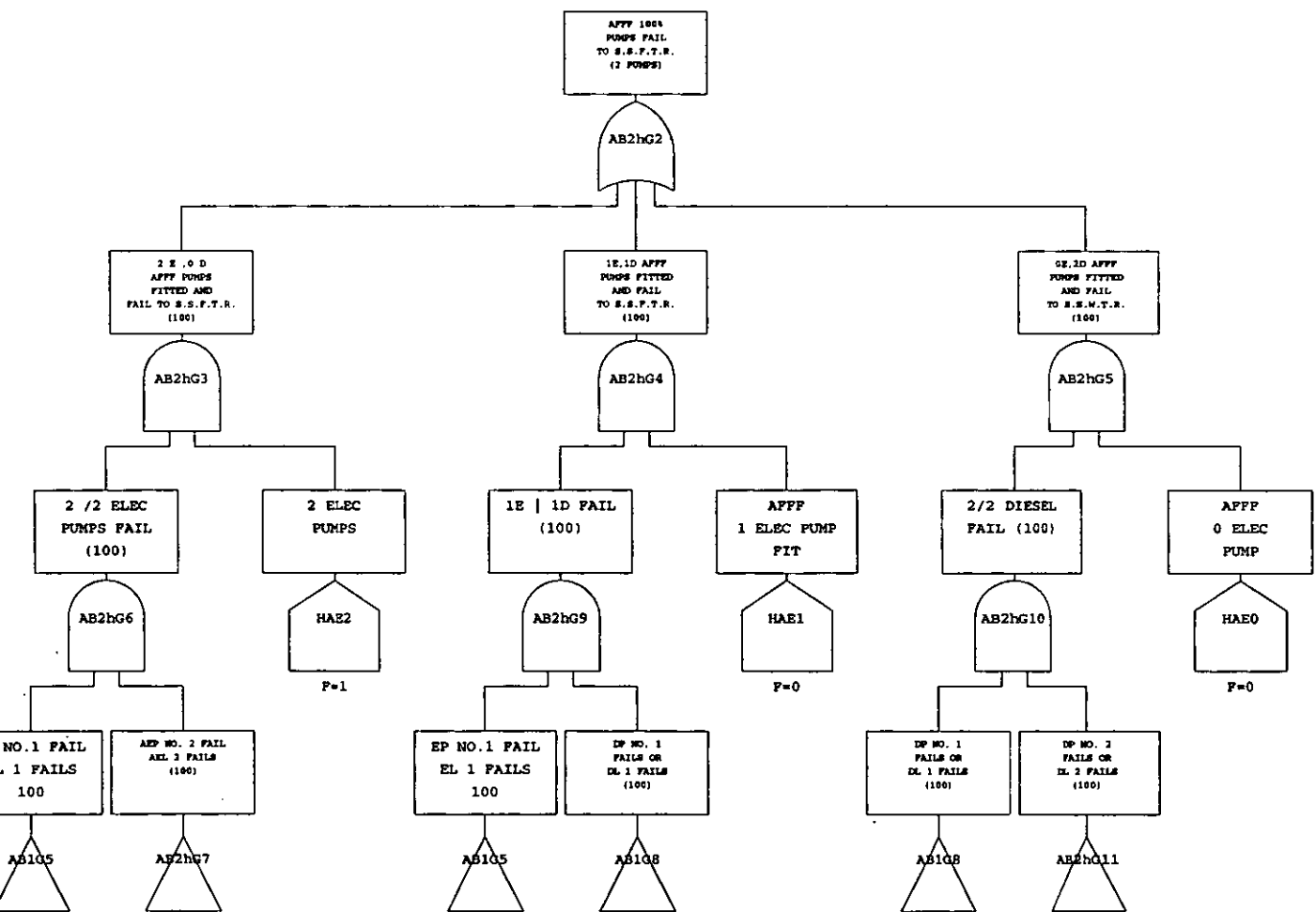


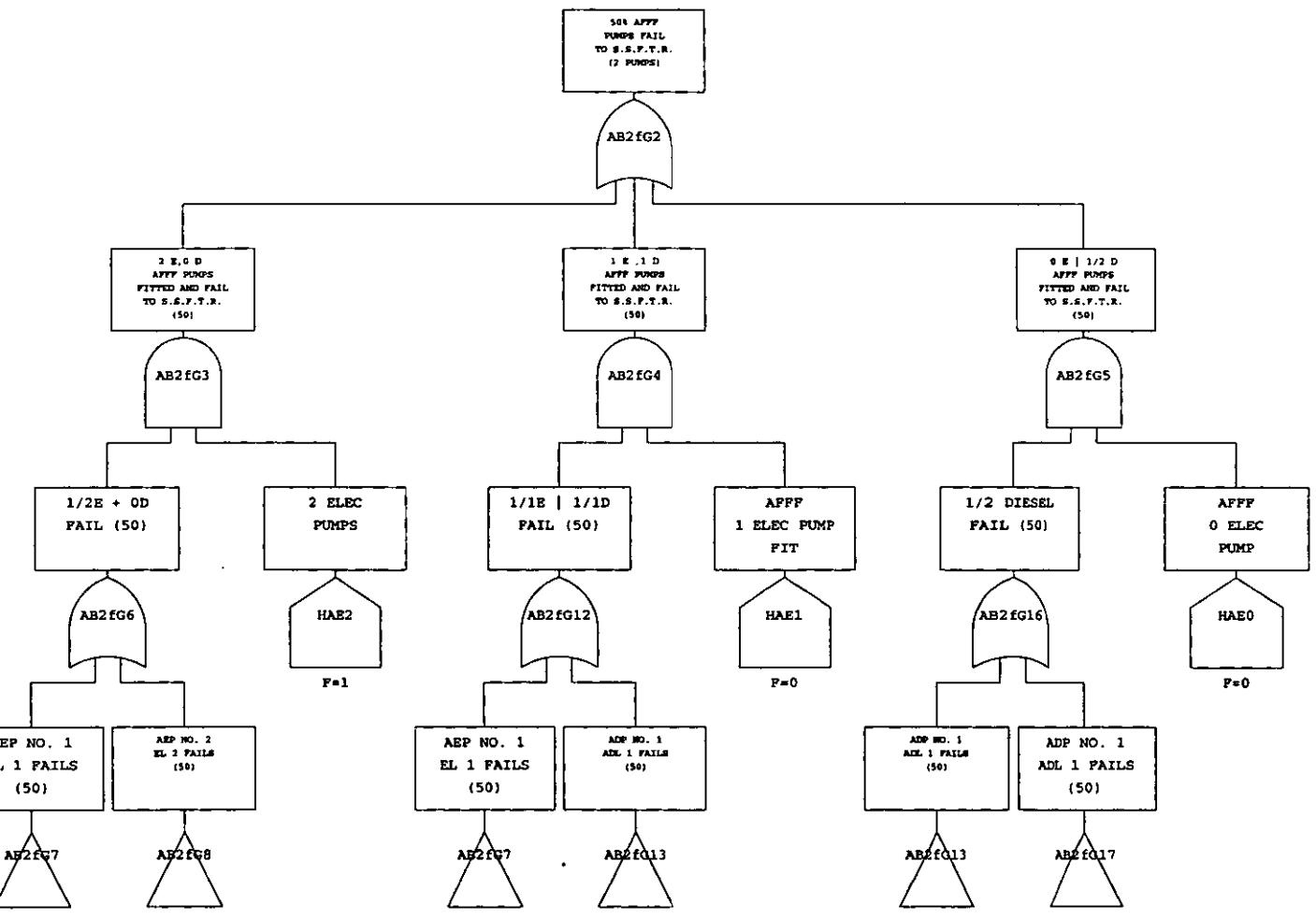
AFFF 100%
PUMPS FAIL
TO S.S.F.T.R.
(2 PUMPS)

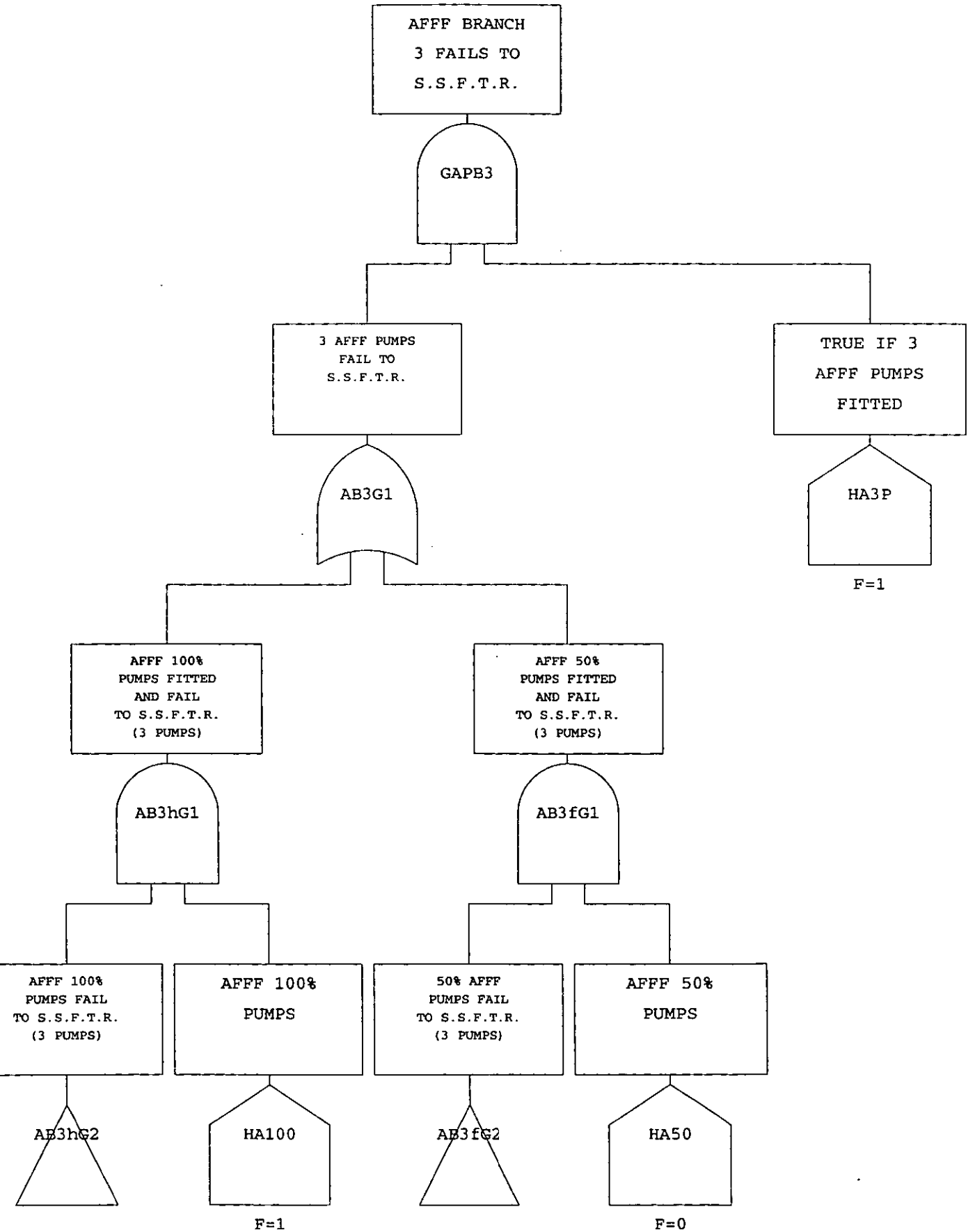
AFFF 100%
PUMPS

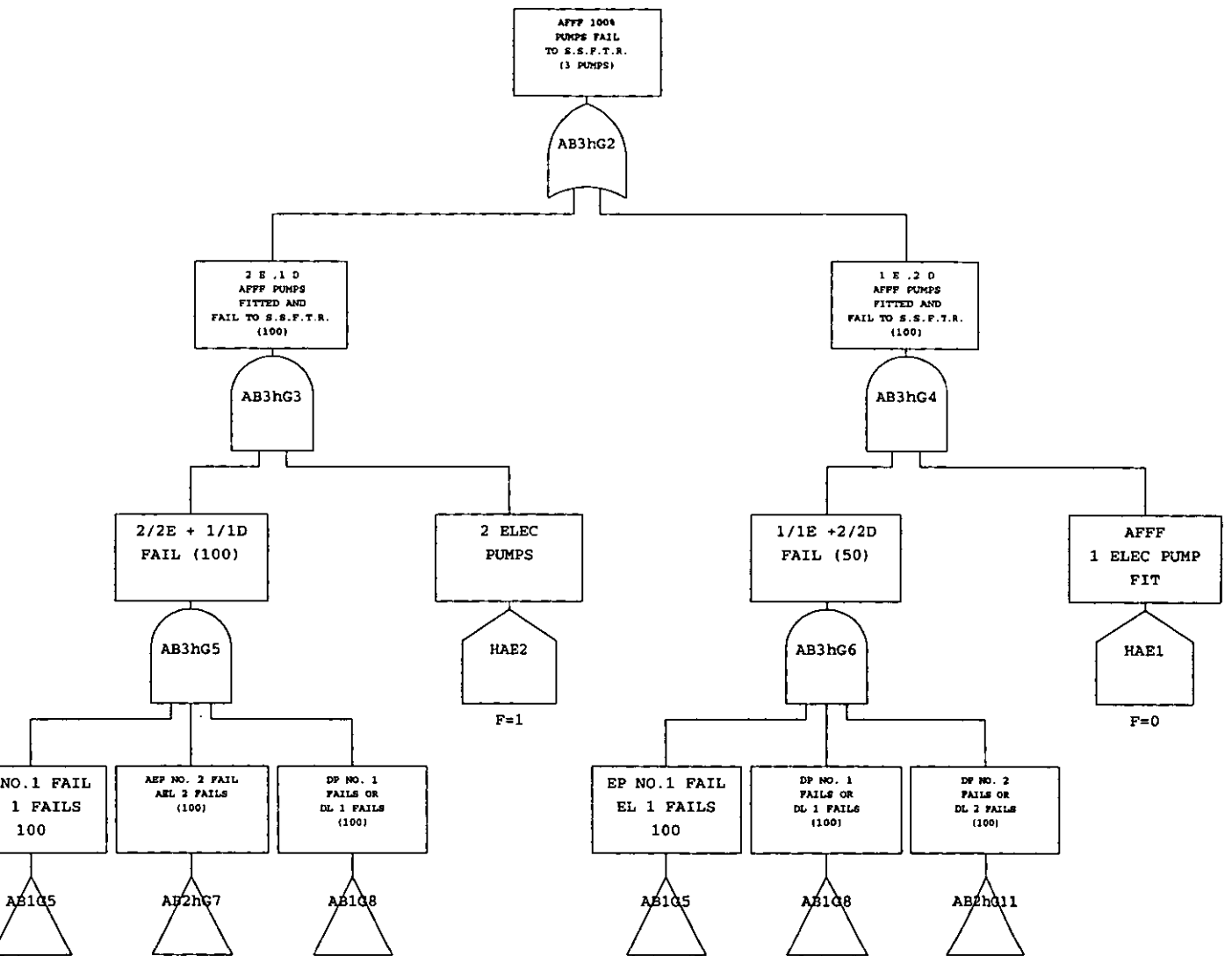


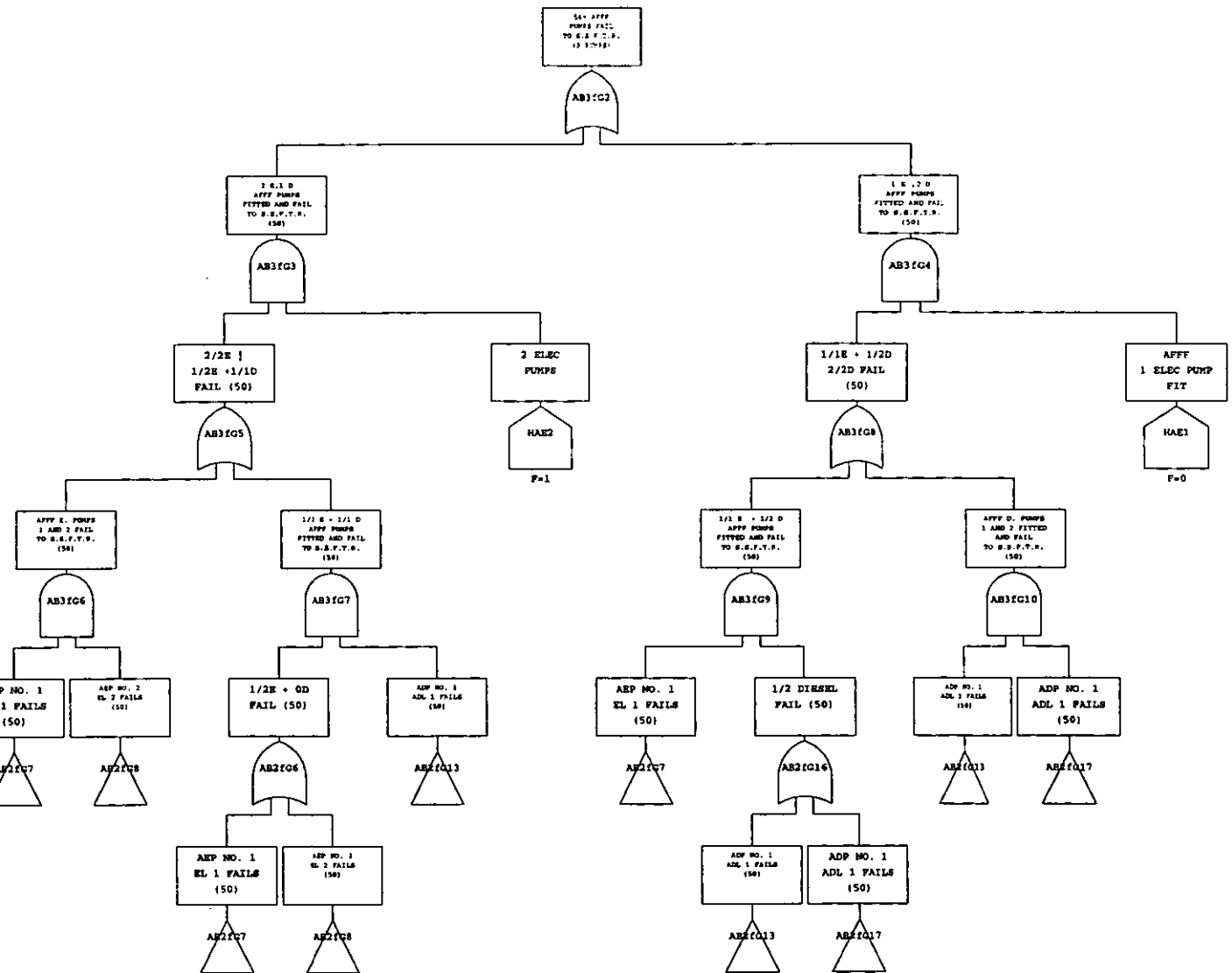
F=1

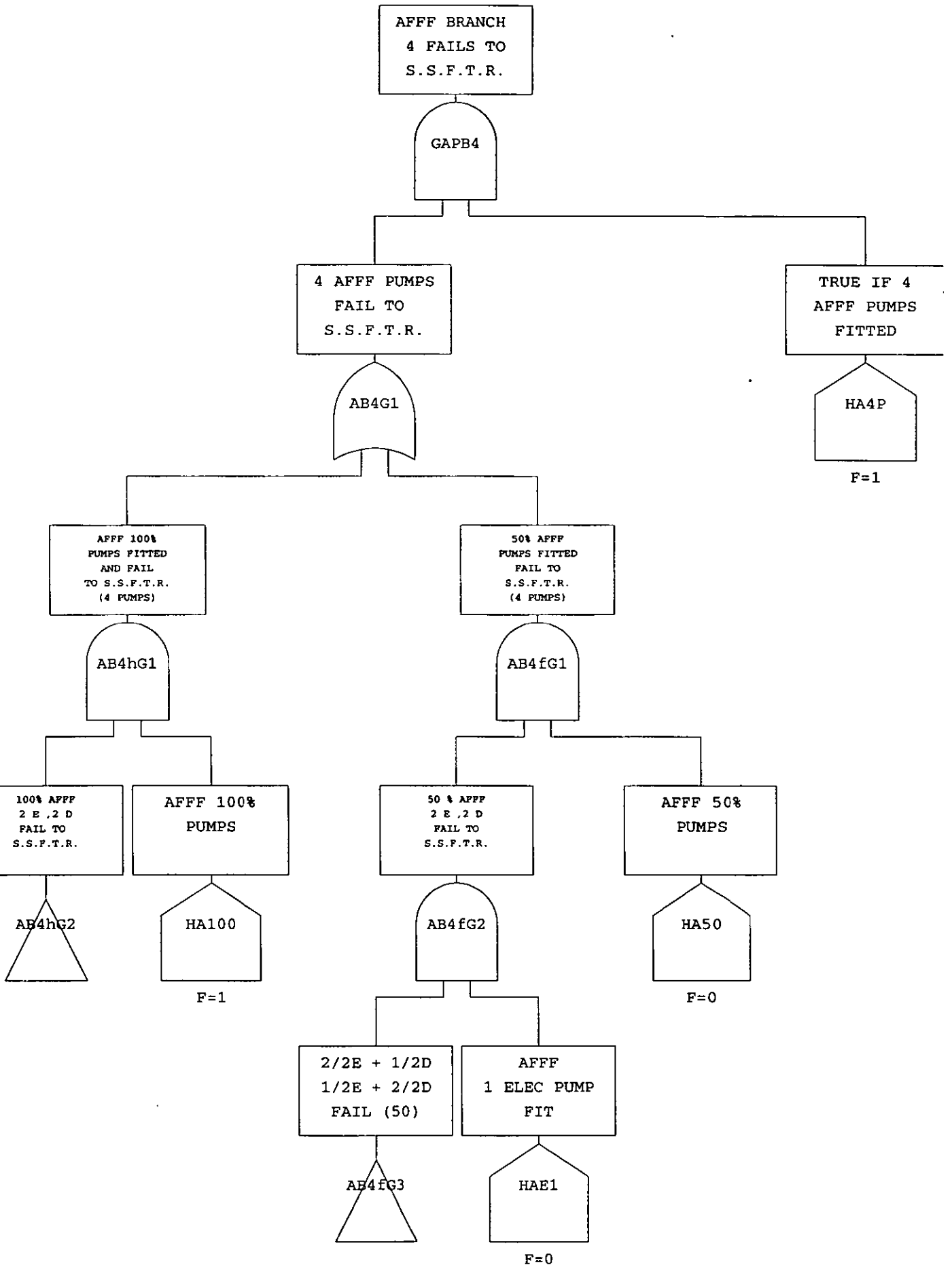


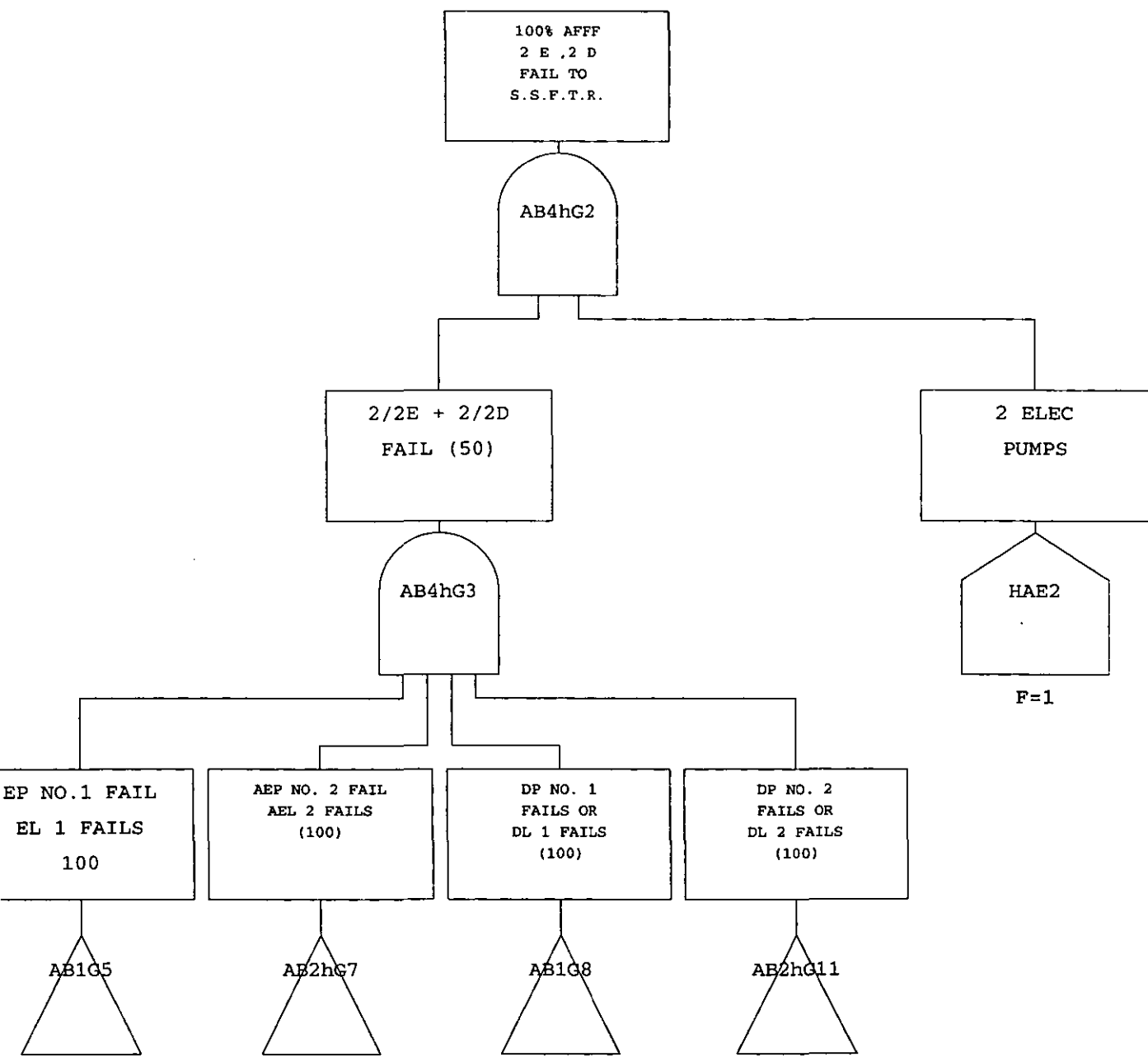


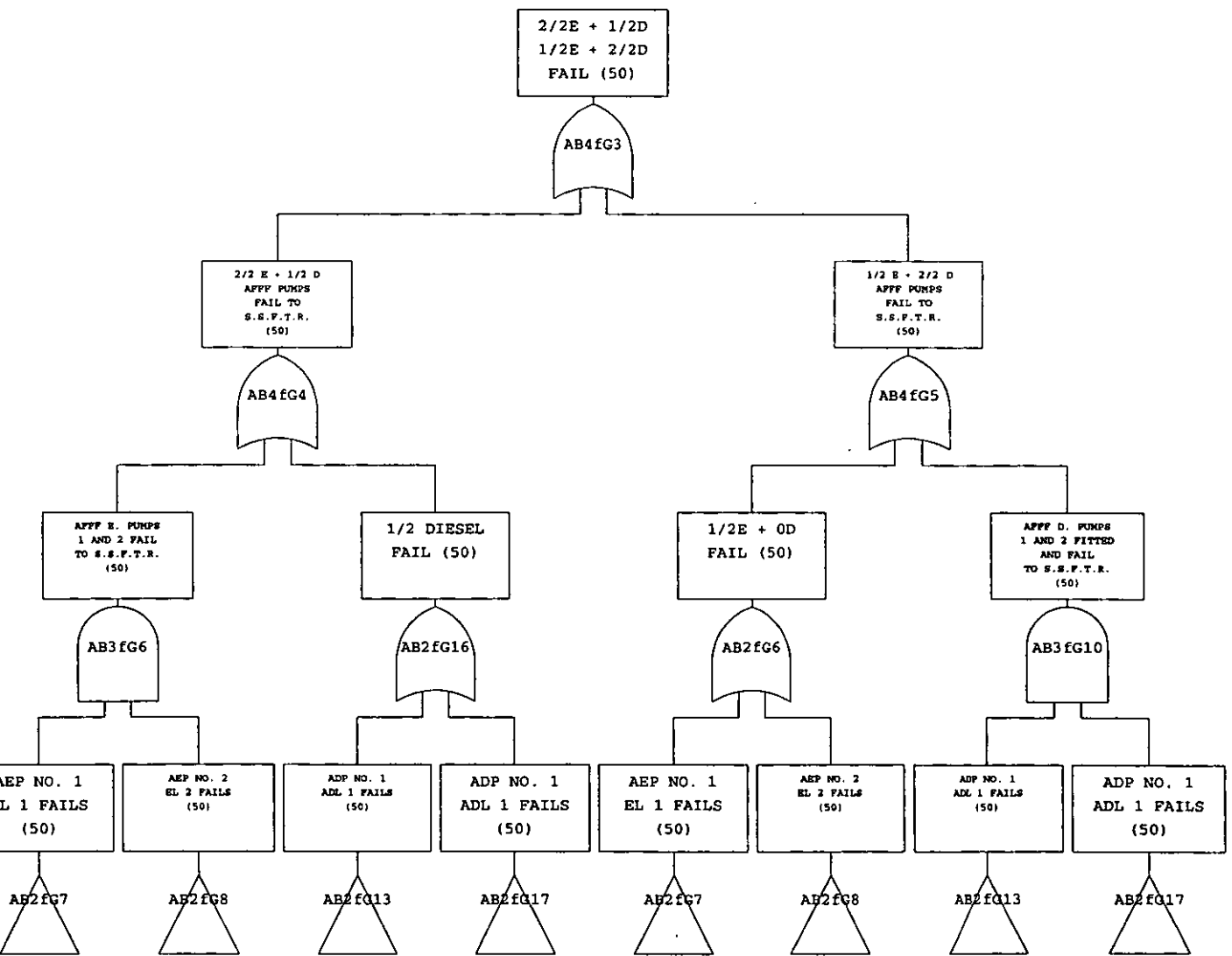


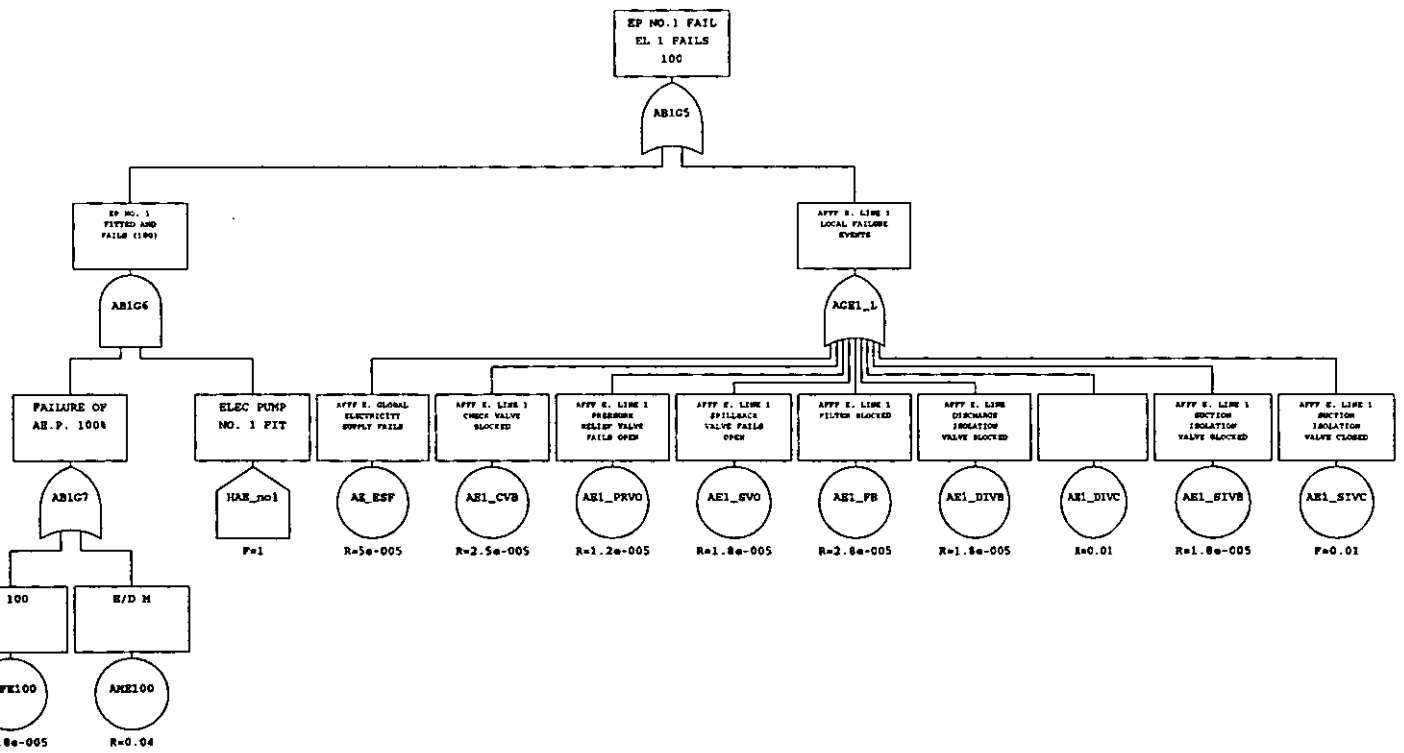


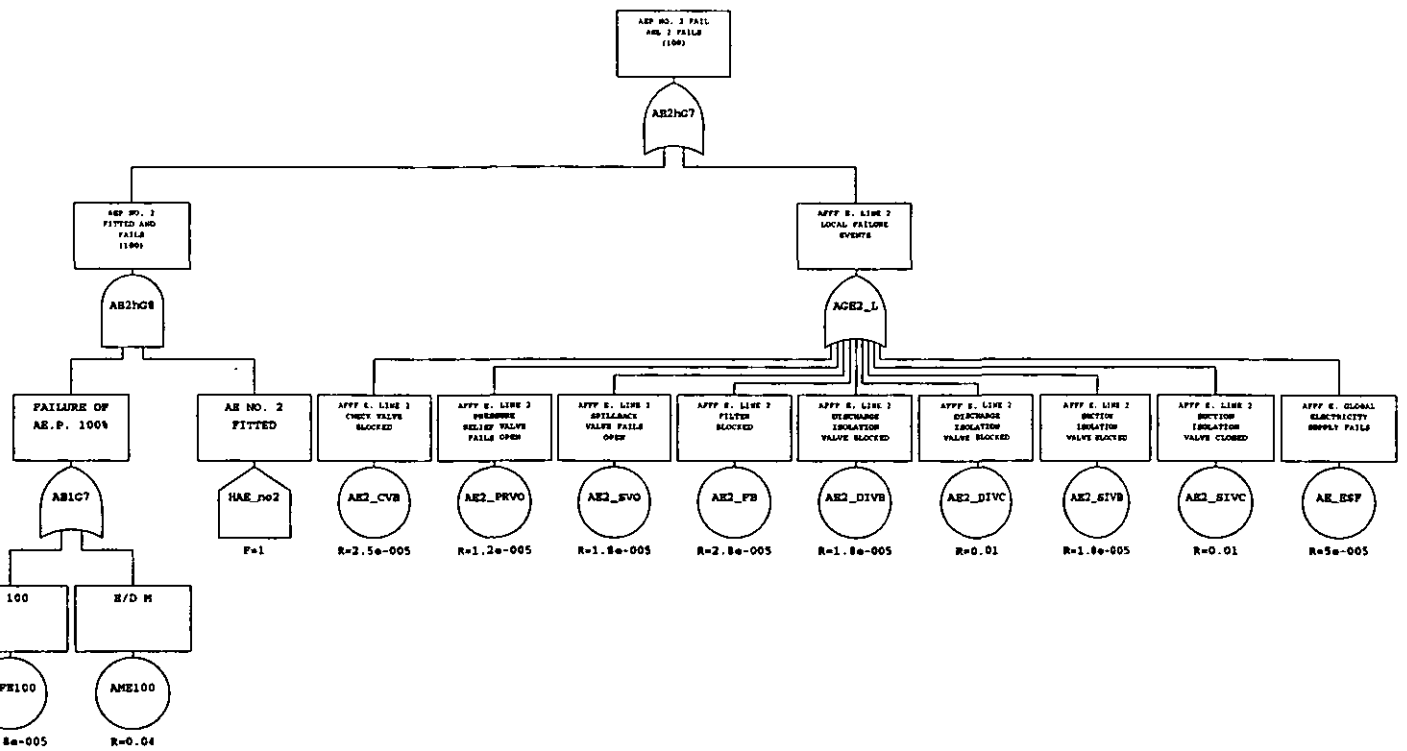


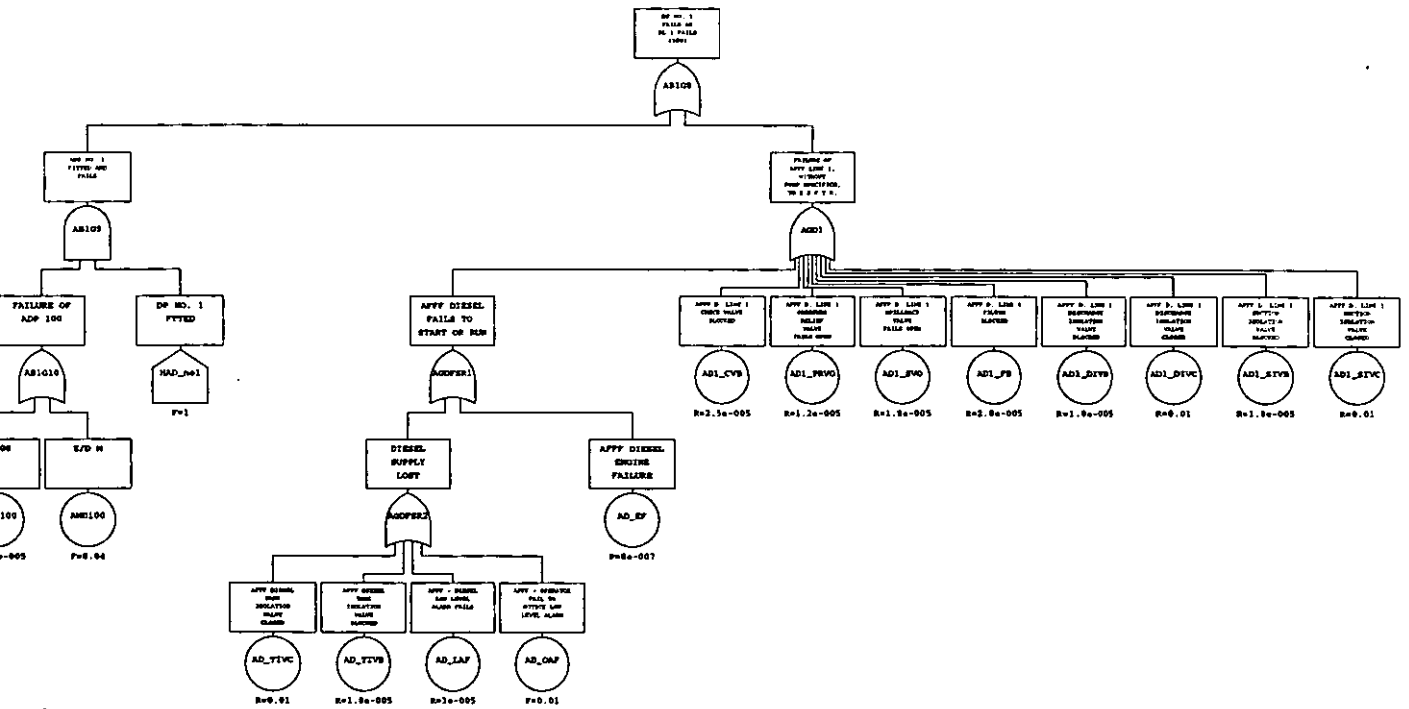


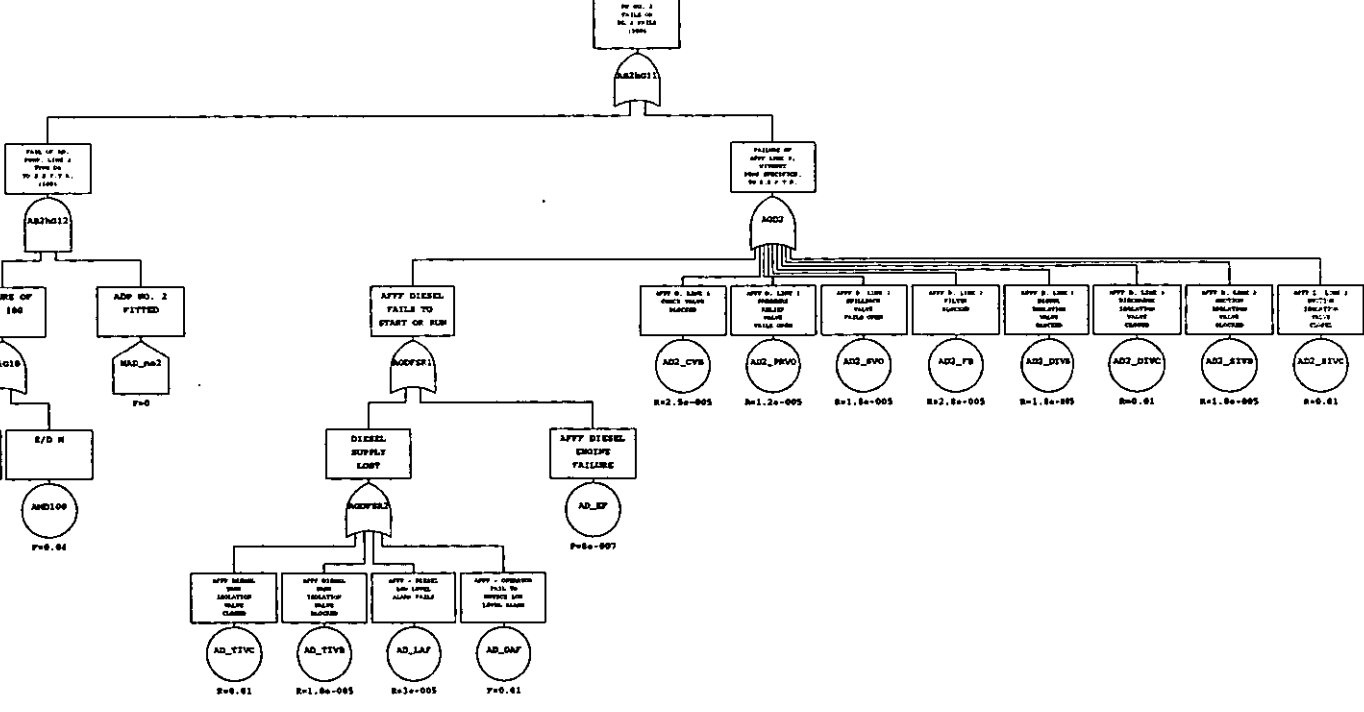


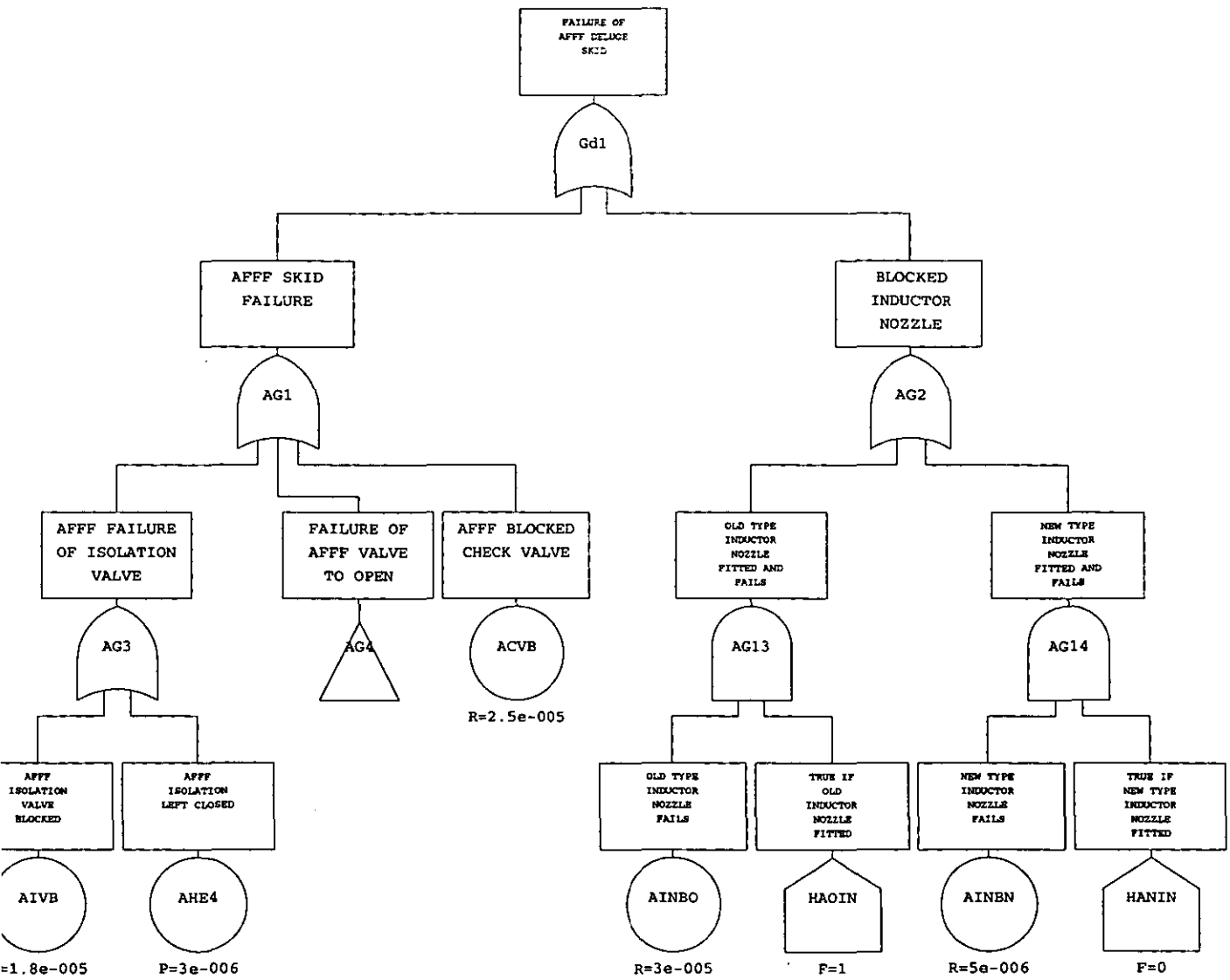


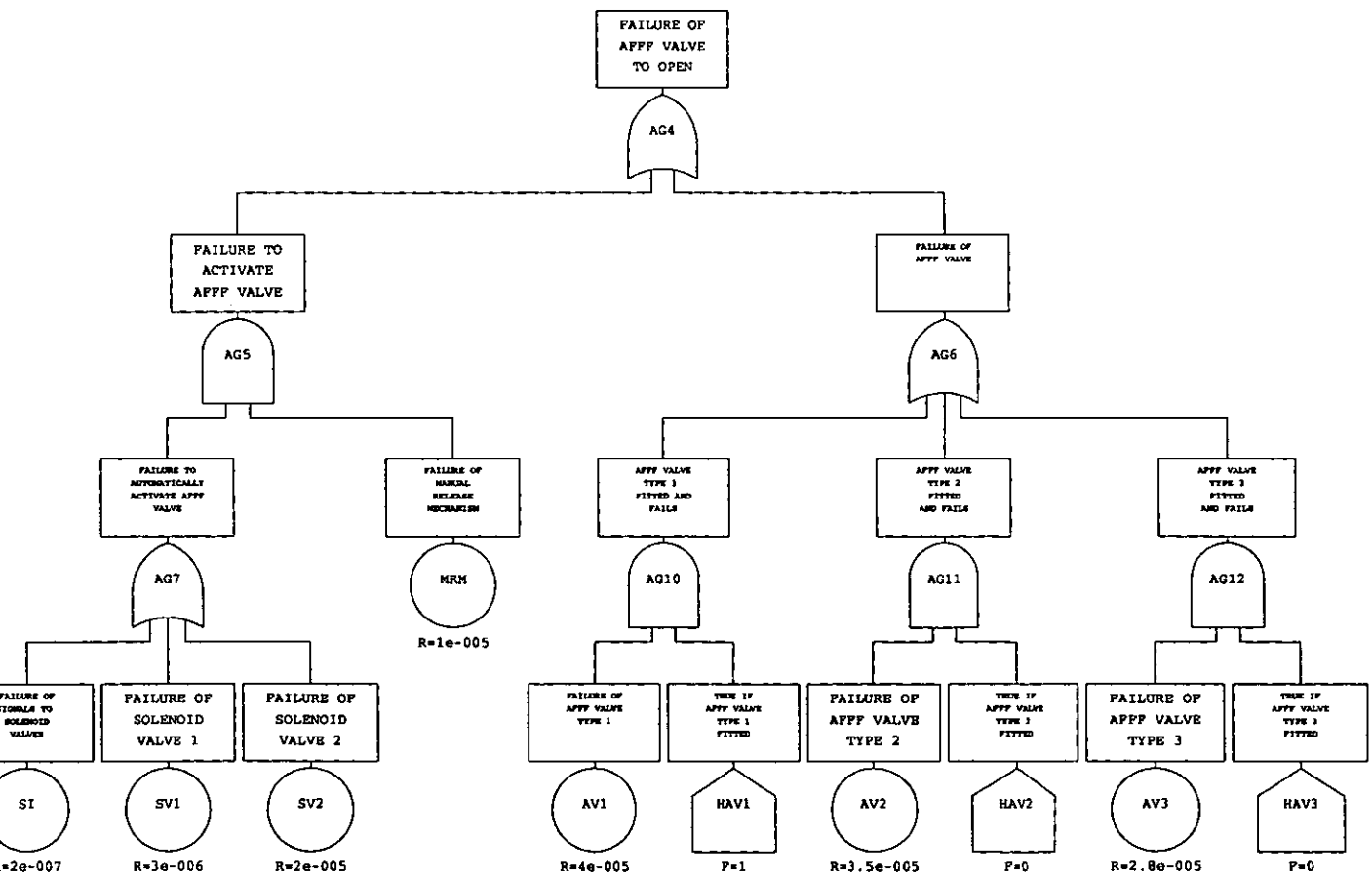


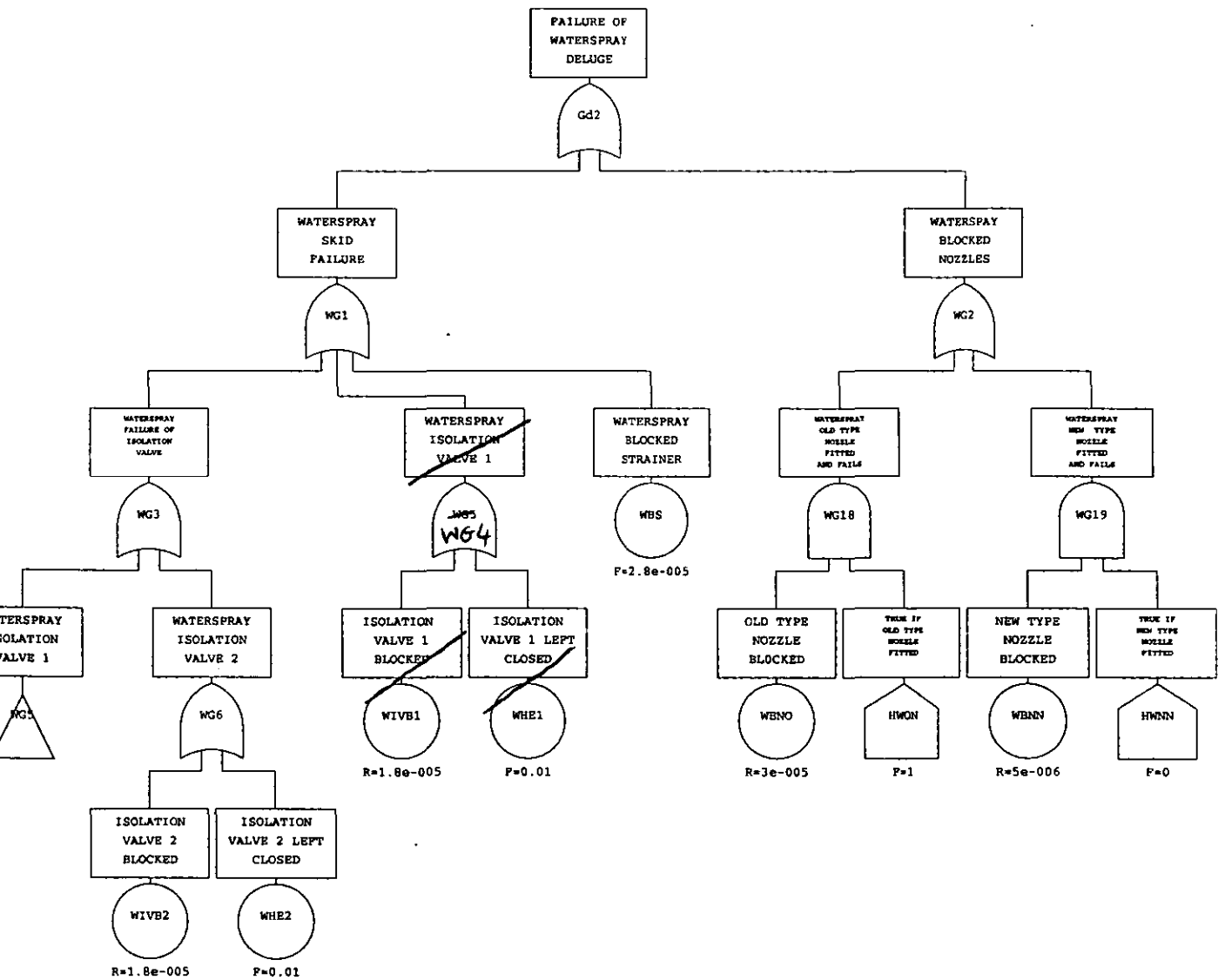


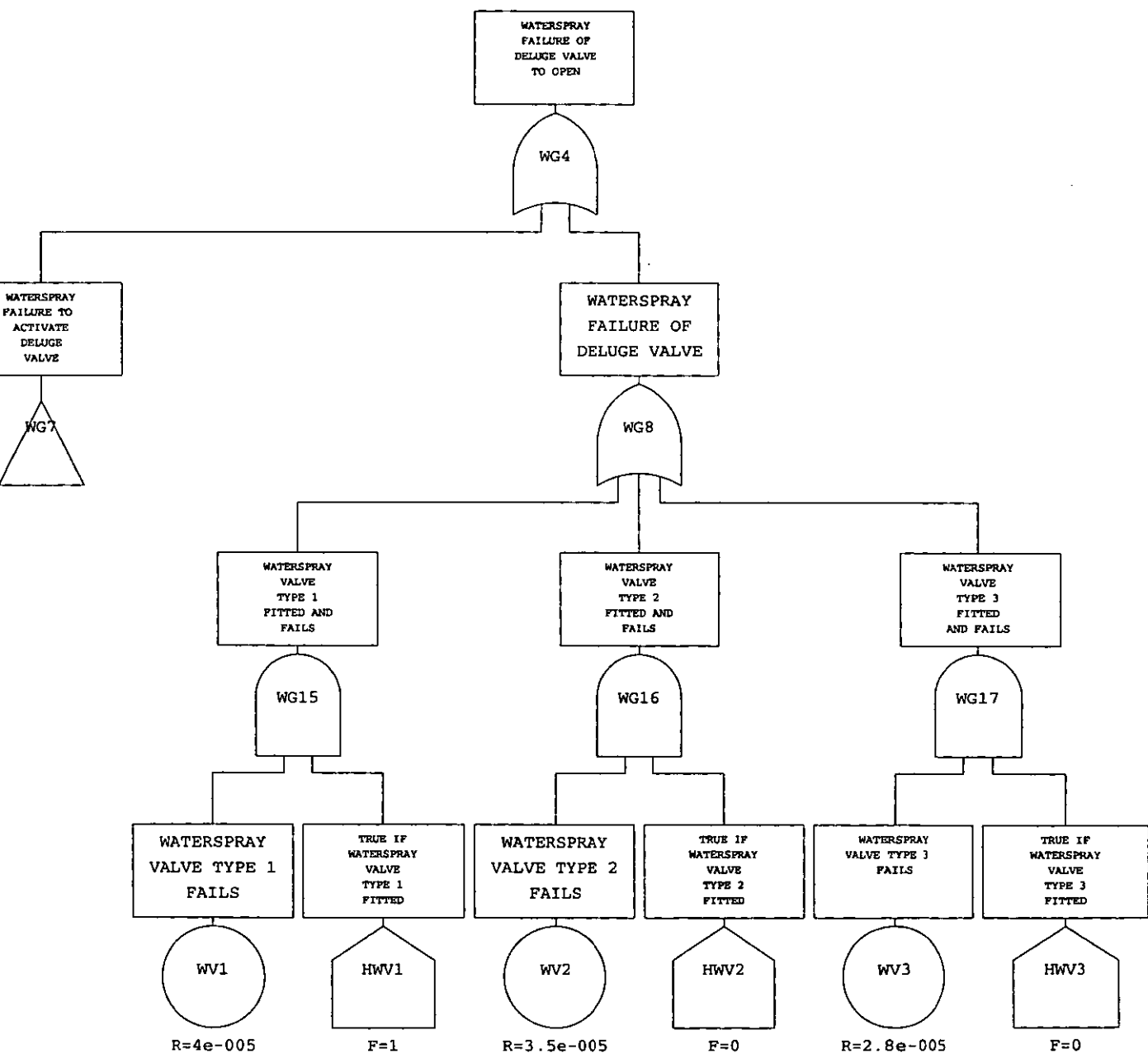


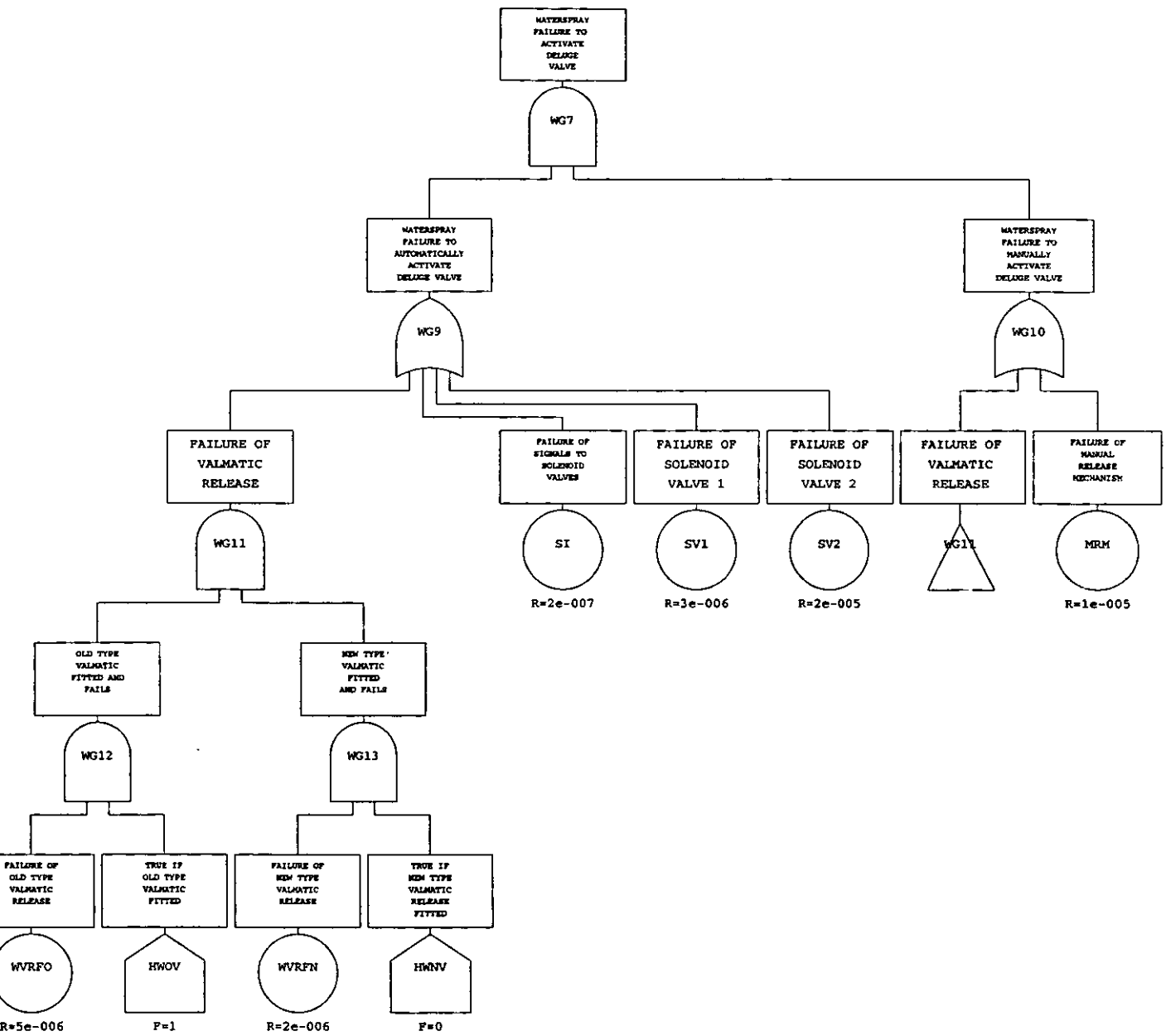






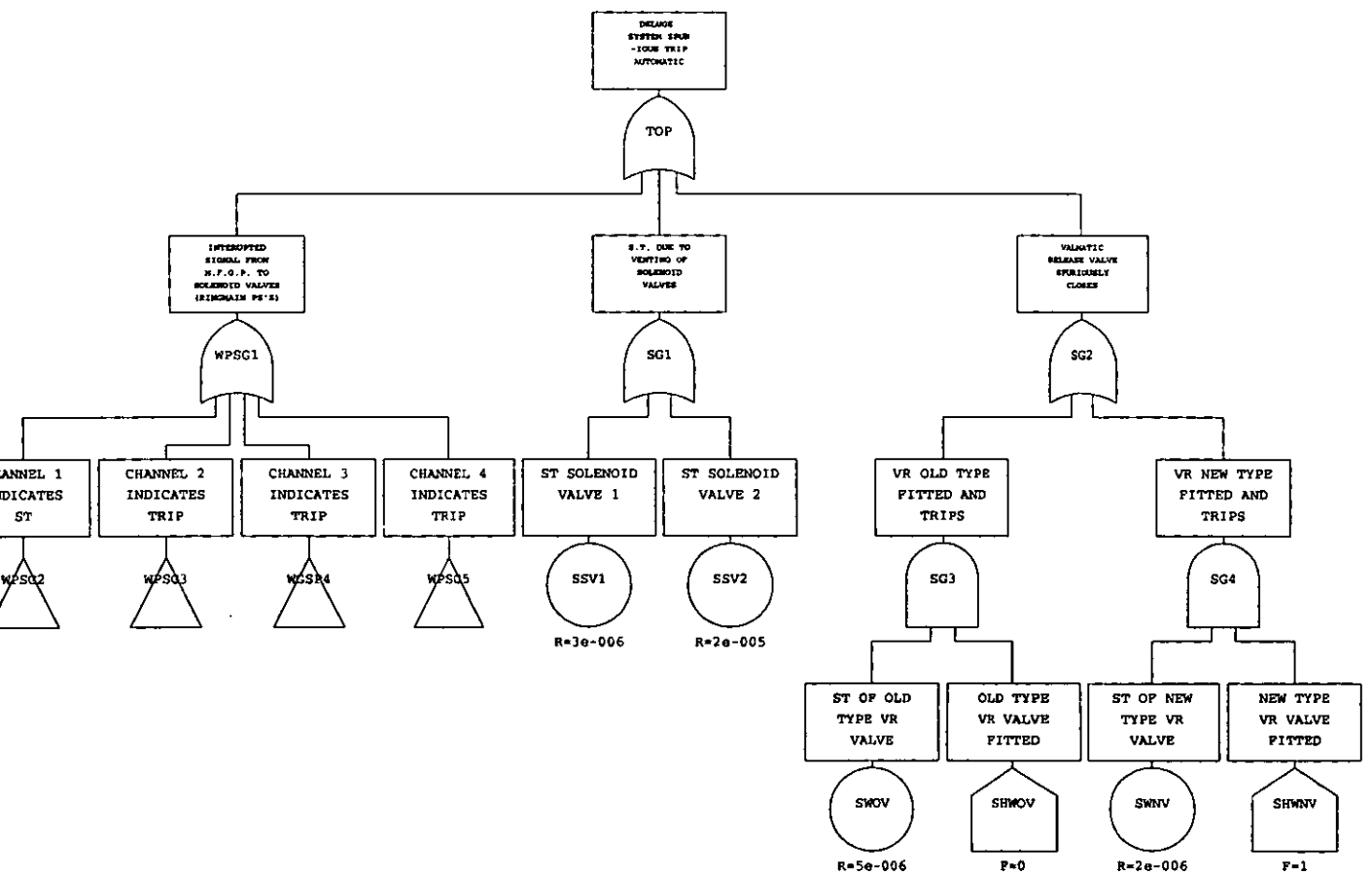


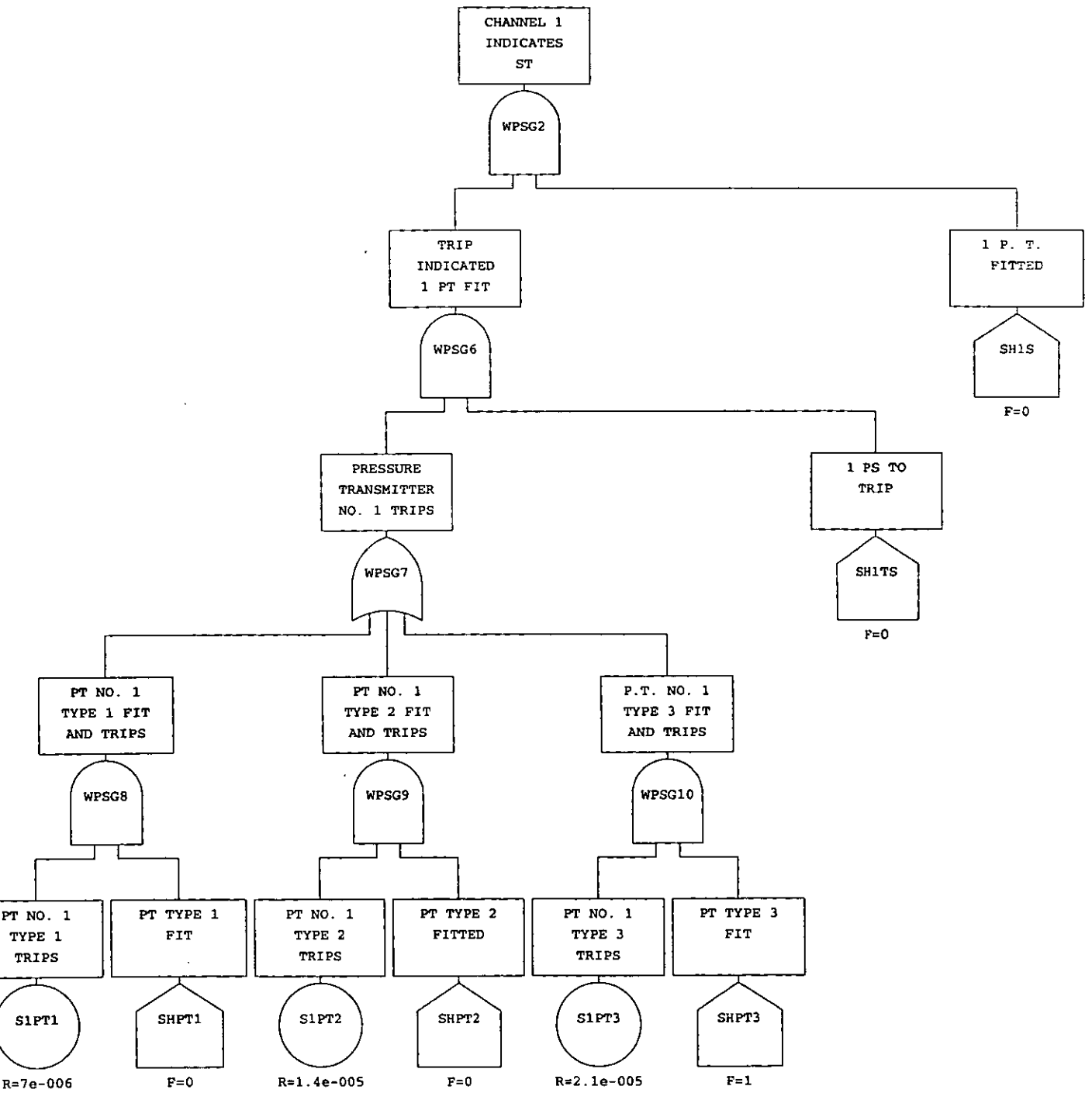


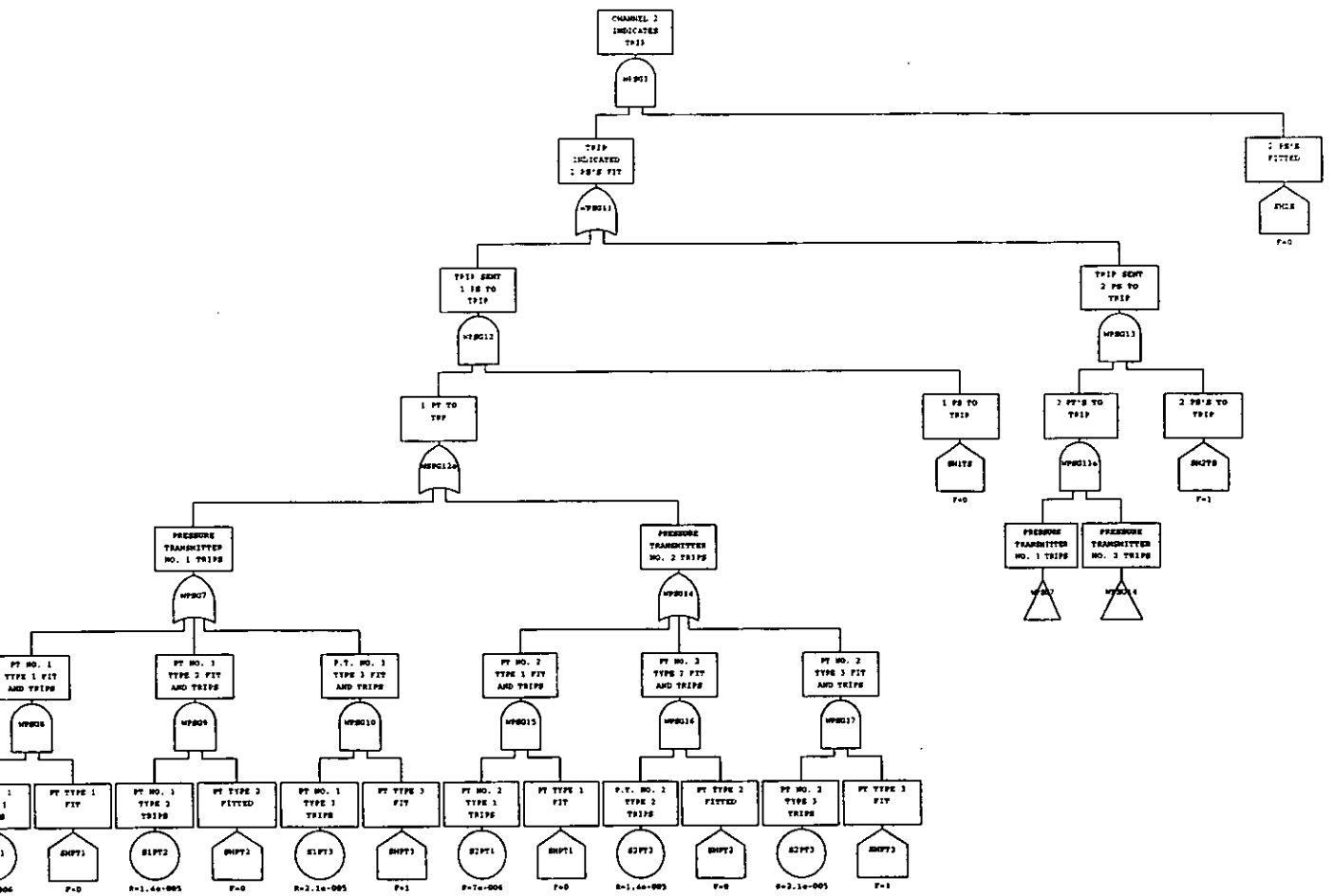


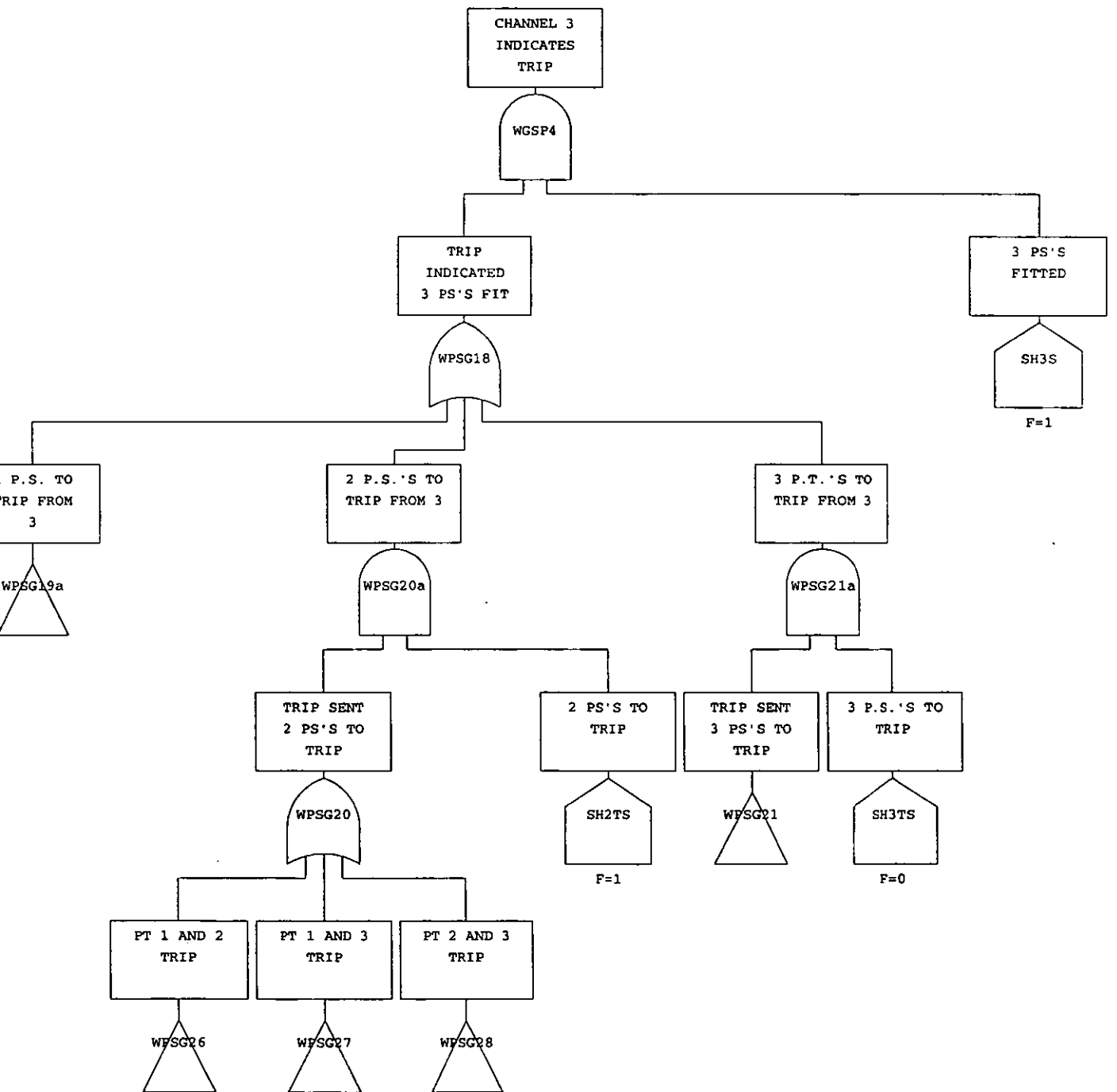
APPENDIX VI

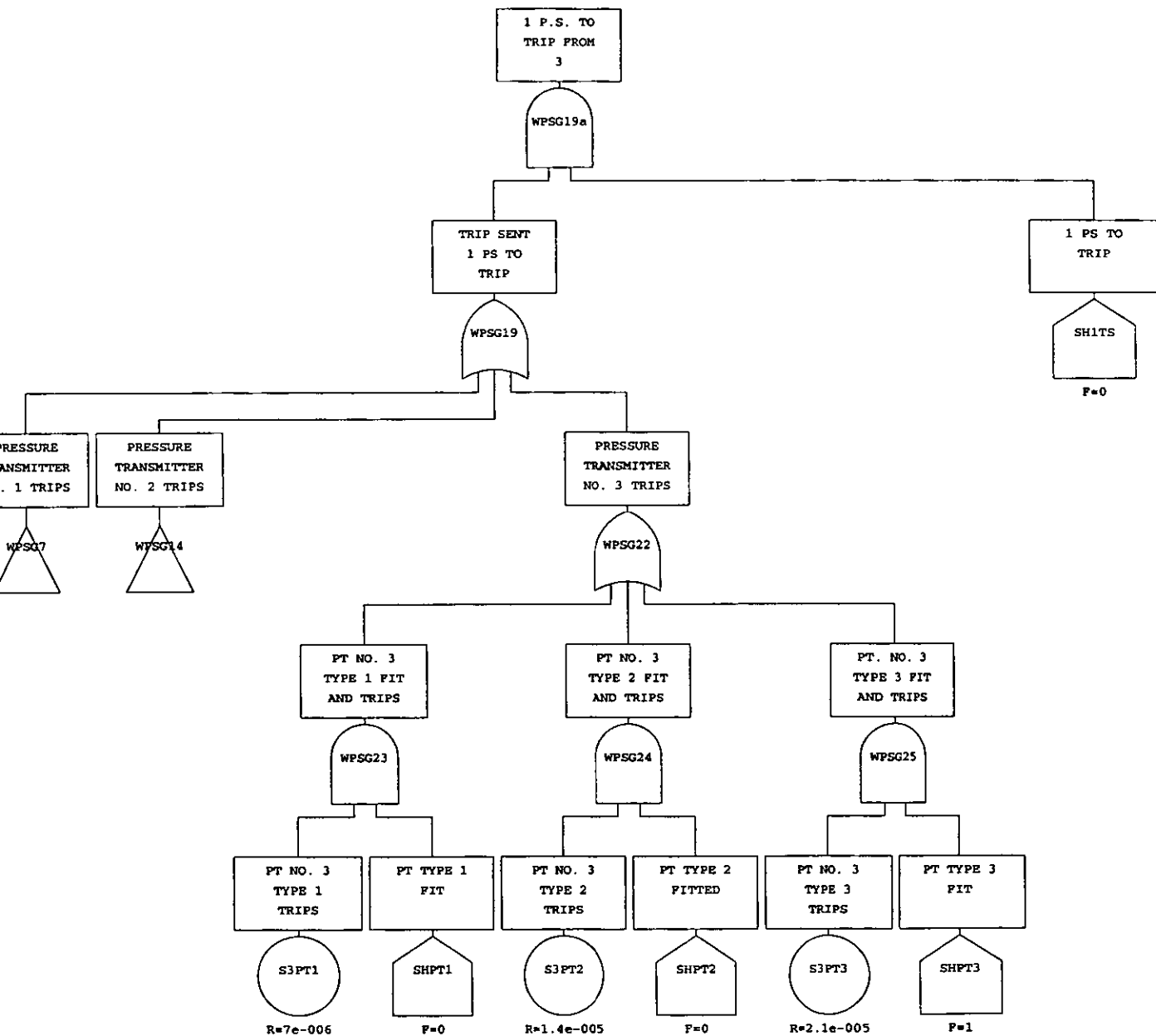
**FAULT TREE STRUCTURE FOR THE SPURIOUS TRIP FAILURE MODE
OF THE FIREWATER DELUGE SYSTEM**

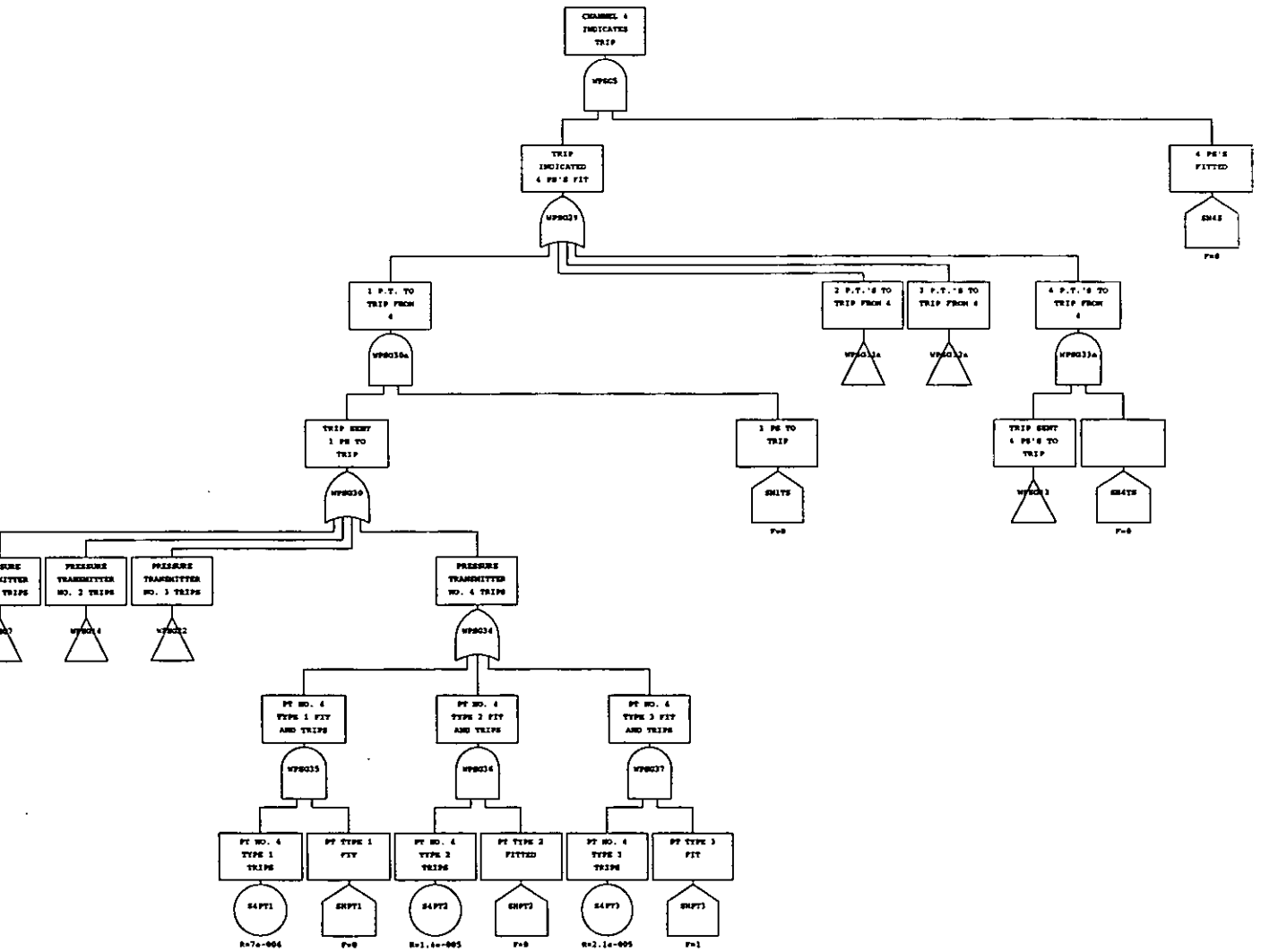


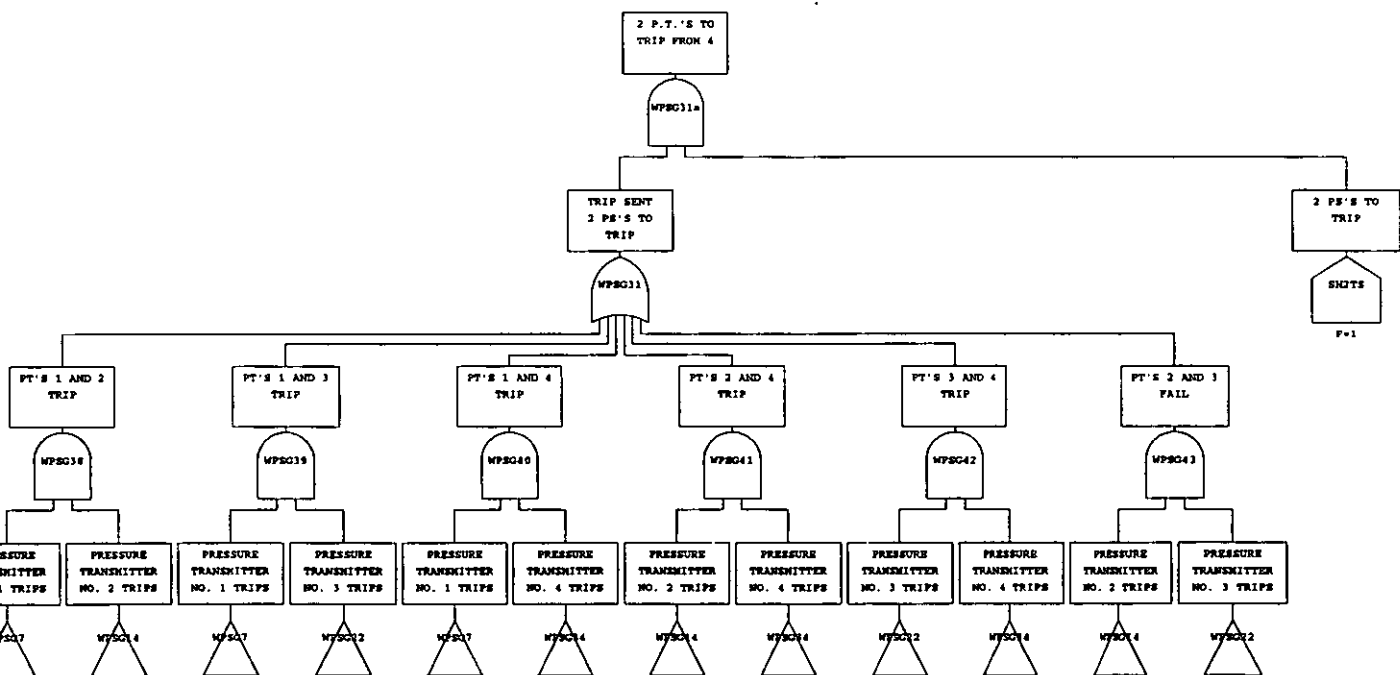


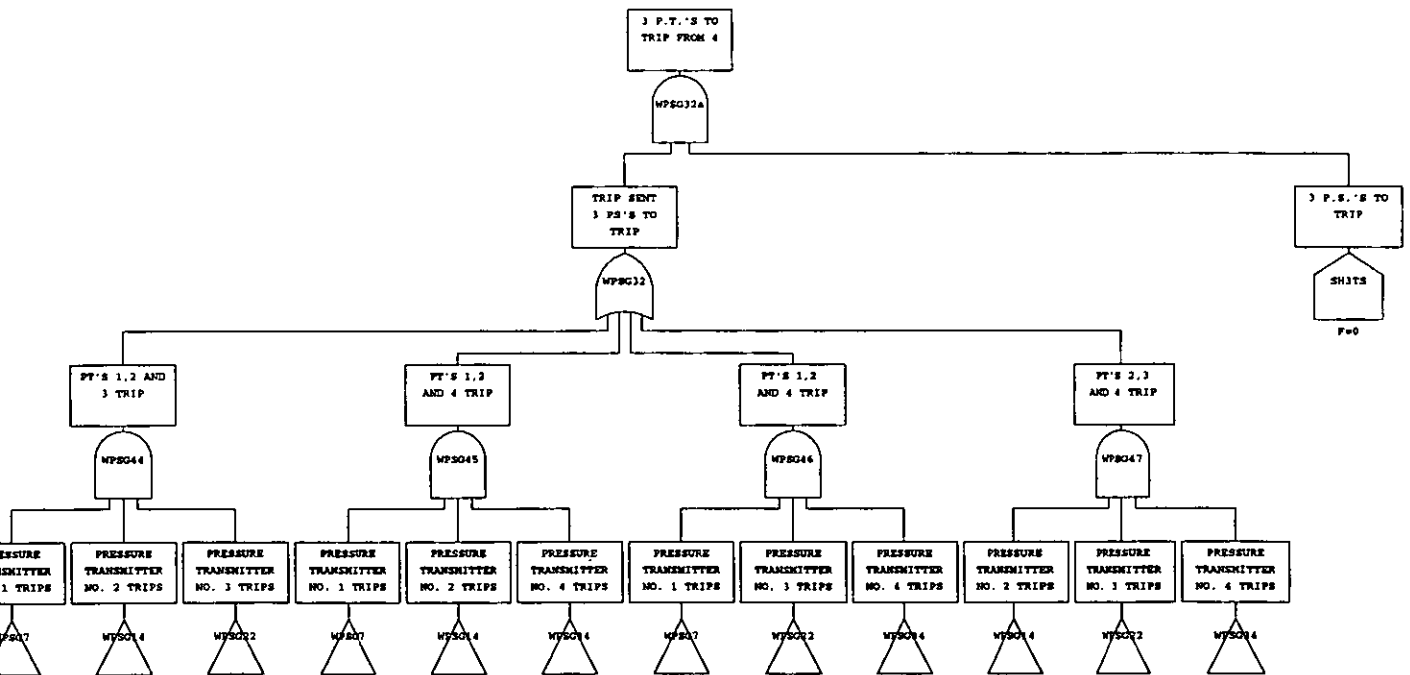












APPENDIX VII – $L_{27}(3^{13})$ Standard Orthogonal Array

Expt. No.	Column												
	1	2	3	4	5	6	7	8	9	10	11	12	13
1	1	1	1	1	1	1	1	1	1	1	1	1	1
2	1	1	1	1	2	2	2	2	2	2	2	2	2
3	1	1	1	1	3	3	3	3	3	3	3	3	3
4	1	2	2	2	1	1	1	2	2	2	3	3	3
5	1	2	2	2	2	2	2	3	3	3	1	1	1
6	1	2	2	2	3	3	3	1	1	1	2	2	2
7	1	3	3	3	1	1	1	3	3	3	2	2	2
8	1	3	3	3	2	2	2	1	1	1	3	3	3
9	1	3	3	3	3	3	3	2	2	2	1	1	1
10	2	1	2	3	1	2	3	1	2	3	1	2	3
11	2	1	2	3	2	3	1	2	3	1	2	3	1
12	2	1	2	3	3	1	2	3	1	2	3	1	2
13	2	2	3	1	1	2	3	2	3	1	3	1	2
14	2	2	3	1	2	3	1	3	1	2	1	2	3
15	2	2	3	1	3	1	2	1	2	3	2	3	1
16	2	3	1	2	1	2	3	3	1	2	2	3	1
17	2	3	1	2	2	3	1	1	2	3	3	1	2
18	2	3	1	2	3	1	2	2	3	1	1	2	3
19	3	1	3	2	1	3	2	1	2	3	1	3	2
20	3	1	3	2	2	1	3	2	3	1	2	1	3
21	3	1	3	2	3	2	1	3	1	2	3	2	1
22	3	2	1	3	1	3	2	2	3	1	3	2	1
23	3	2	1	3	2	1	3	3	1	2	1	3	2
24	3	2	1	3	3	2	1	1	2	3	2	1	3
25	3	3	2	1	1	3	2	3	1	2	2	1	3
26	3	3	2	1	2	1	3	1	2	3	3	2	1
27	3	3	2	1	3	2	1	2	3	1	1	3	2

