This item was submitted to Loughborough's Research Repository by the author.
Items in Figshare are protected by copyright, with all rights reserved, unless otherwise indicated.

## Variable ordering heuristics for binary decision diagrams

## PLEASE CITE THE PUBLISHED VERSION

## PUBLISHER

© Lisa Marie Bartlett

LICENCE

CC BY-NC-ND 4.0

REPOSITORY RECORD

Bartlett, L.M.. 2019. "Variable Ordering Heuristics for Binary Decision Diagrams". figshare. https://hdl.handle.net/2134/10362.

## This item was submitted to Loughborough University as a PhD thesis by the author and is made available in the Institutional Repository (https://dspace.Iboro.ac.uk/) under the following Creative Commons Licence conditions.

## (c) creative

C O M M O N S D E E D

Attribution-NonCommercial-NoDerivs 2.5

You are free:

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

Under the following conditions:

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


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

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

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

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

This is a human-readable summary of the Leqal Code (the full license).
Disclaimer $\left.{ }^{[ }\right]$

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

BLDSC no:- DX 229243
$\qquad$
Author/Filing Title . $\qquad$ BARTLETT
$\qquad$
Vol. No. $\qquad$ Class Mark $\qquad$

Please note that fines are charged on ALL overdue items.


$$
\begin{array}{c:cc} 
& \\
& \cdots & \\
\cdots & \cdots & \\
\hdashline & \cdots
\end{array}
$$

# Variable Ordering Heuristics For Binary Decision Diagrams 

By<br>Lisa Marie Bartlett

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

March 2000



#### Abstract

Fault tree analysis, FTA, is one of the most commonly used techniques for safety system assessment. Over the past five years the Binary Decision Diagram (BDD) methodology has been introduced which significantly aids the analysis of the fault tree diagram. The approach has been shown to improve both the efficiency of determining the minimal cut sets of the fault tree, and also the accuracy of the calculation procedure used to quantify the top event parameters. To utilise the BDD technique the fault tree structure needs to be converted into the BDD format. Converting the fault tree is relatively straightforward but requires the basic events of the tree to be placed in an ordering. The ordering of the basic events is-critical to the resulting size of the BDD, and ultimately affects the performance and benefits of this technique. There are a number of variable ordering heuristics in the literature, however the performance of each depends on the tree structure being analysed. These heuristic approaches do not always yield a minimal BDD structure for all trees, some approaches generate orderings that are better for some trees but worse for others. Within this thesis three pattern recognition approaches, that of machine learning classifier systems, multi-layer perceptron networks and radial basis function neural networks, have been investigated to try and select a variable ordering heuristic for a given fault tree from a set of alternatives. In addition a completely new heuristic based on component structural importance measures has been suggested with significant improvement in producing the smallest BDD over those methods currently in the literature.


## Acknowledgements

I would like to give my sincere thanks to Dr John Andrews, my supervisor, for his invaluable help, guidance and friendship during the course of my PhD work. Thanks to my boyfriend, Tom, for his indelible support throughout and his patience in explaining a number of computer issues. Thanks are given to my parents, sister, and friends, Annette and Rosie for their continual encouragement and friendship. Special thanks go Linda and Sandra for proof reading. Also thanks go to my colleagues in the mathematics department for their friendship and good humour.

## Contents

1 Introduction ..... 1
1.1 Introduction to Risk and Reliability ..... 1
1.2 System Reliability Modelling Techniques ..... 3
1.2.1 Introduction ..... 3
1.2.2 Fault Tree Analysis ..... 3
1.2.3 Event Tree Analysis ..... 5
1.3 Selecting an Analysis Method. ..... 7
1.3.1 Reasons For Fault Tree Popularity ..... 7
1.3.2 Qualitative and Quantitative Analysis in Brief ..... 8
1.4 The Binary Decision Diagram Approach ..... 10
1.4.1 Advantages ..... 10
1.4.2 Advances to The Binary Decision Diagram Technique ..... 11
1.5 Summary ..... 11
1.6 Scope of Research. ..... 12
2 Fault Tree Analysis. ..... 13
2.1 Introduction ..... 13
2.2 Background ..... 14
2.3 Description of The Fault Tree Construction Procedure ..... 15
2.3.1 Fault Tree Symbols ..... 15
2.3.2 Construction Methodology ..... 17
2.4 Qualitative Analysis of The Fault Tree ..... 19
2.4.1 Definition of The Two Types of Fault Tree ..... 19
2.4.2 The Analysis Procedure ..... 19
2.4.3 Obtaining Minimal Cut Sets For Coherent Trees ..... 21
2.4.4 Obtaining Prime Implicants of Non-Coherent Trees ..... 24
2.5 Quantitative Analysis ..... 28
2.5.1 Background of The Analysis Procedure ..... 28
2.5.2 Component and System Quantification Measures ..... 28
2.5.3 Defining The Fault Tree Structure - Structure Functions ..... 30
2.5.4 Shannon's Theorem ..... 32
2.5.5 General Approach to Calculate The Top Event Probability ..... 34
2.5.6 Approximation Methods ..... 37
2.5.7 Unconditional Failure Intensity ..... 38
2.6 Importance Measures ..... 40
2.6.1 Introduction ..... 40
2.6.2 Deterministic Measures ..... 41
2.6.3 Probabilistic Measures For Assessing System Unavailability. ..... 42
2.6.4 Probabilistic Measures For System Unreliability ..... 45
2.7 Advantages and Disadvantages of The Fault Tree Methodology ..... 47
2.8 Summary ..... 48
3 Binary Decision Diagrams ..... 49
3.1 Introduction ..... 49
3.2 Background ..... 49
3.3 BDD Architecture ..... 50
3.4 Constructing The BDD ..... 51
3.4.1 Brief Introduction ..... 51
3.4.2 Constructing The BDD Using The Top Event Logic Function ..... 52
3.4.3 Constructing The BDD Using The If-Then-Else Method ..... 55
3.4.4 Reducing The Size of The BDD ..... 60
3.5 Top Event Quantification ..... 68
3.5.1 System Failure Probability. ..... 68
3.5.2 System Failure Intensity ..... 69
3.6 Applications of BDDs To Event Trees ..... 75
3.6.1 Overview of Event Trees ..... 75
3.6.2 Traditional Solution to Dependencies Within The Event Tree. ..... 76
3.6.3 How The BDD Has Enhanced The Event Tree Technique ..... 78
3.7 Variable Ordering ..... 78
3.7.1 The Problem ..... 78
3.8 Summary ..... 80
4 Variable Ordering Heuristics ..... 82
4.1 Introduction ..... 82
4.2 The Most Commonly Used Heuristic ..... 82
4.3 Modifications to The Top-down, Left-right Approach ..... 85
4.3.1 Using Repeated Events. ..... 85
4.3.2 Using Subtrees (Depth-First and Priority Depth-First Heuristics) ..... 87
4.3.3 Variable Ordering Using Repeated Basic Events and Subtree Levels ..... 92
4.3.4 Applying Weights to The Depth-First Approach ..... 96
4.3.5 Using The Number of Leaves in Conjunction With The Depth-first Approach. ..... 99
4.3.6 Depth-First With Number of Fanouts Considered. ..... 101
4.4 Comparative Studies of Heuristics ..... 105
4.5 Alternative Heuristics ..... 110
4.5.1 Modules of a BDD ..... 110
4.6 Dynamic Ordering Methods. ..... 115
4.7 Performance of Heuristics ..... 116
4.8 Summary. ..... 116
5 Pattern Recognition Techniques - The Machine Learning Classifier System Incorporating Genetic Algorithms. ..... 118
5.1 Introduction to Pattern Recognition Techniques ..... 118
5.1.1 Summary of The General Approach ..... 118
5.1.2 Types of Problems Modelled by Pattern Recognition Techniques ..... 119
5.1.3 Considerations With All Pattern Recognition Techniques - The Curse of Dimensionality ..... 120
5.1.4 What Are Genetic Algorithms (G.A's)? ..... 120
5.1.5 Why Are They Used in The Classifier System Approach? ..... 121
5.2 Genetic Algorithm Principles ..... 121
5.3 A Simple Genetic Algorithm ..... 122
5.3.1 The Principles ..... 122
5.3.2 Reproduction Operator ..... 123
5.3.3 Crossover Operator ..... 124
5.3.4 Mutation Operator ..... 125
5.3.5 The Operation of a Simple Genetic ..... 126
5.4 Implementation of a Genetic Algorithm Within The Classifier System ..... 128
5.5 Background to Machine Learning Classifier Systems ..... 128
5.5.1 Introduction ..... 128
5.6 Machine Learning Systems - The Classifier Model ..... 130
5.6.1 General Overview of Model ..... 130
5.6.2 The Rule and Message System. ..... 131
5.6.3 The Apportionment of Credit Algorithm (A.O.C) ..... 134
5.6.4 Rule and Message Generation System - Genetic Algorithm ..... 137
5.7 Application Of The Classifier Approach With a Genetic Algorithm to the Ordering Problem ..... 138
5.7.1 Introduction ..... 138
5.7.2 Input Parameters Chosen To Represent The Problem ..... 139
5.7.3 Output Parameters Chosen To Represent The Problem ..... 141
5.7.4 Coding of The Fault Tree Parameters. ..... 142
5.7.5 Coding The Scheme Preferences ..... 144
5.7.6 Data Sets ..... 145
5.7.7 The Programmed Model ..... 146
5.8 Generating a Classifier Model ..... 148
5.8.1 Using Initial Four Fault Tree Characteristics ..... 148
5.8.2 Reviewing Relationship Between Characteristics and Scheme Choices. ..... 152
5.8.3 Changing The Characteristics ..... 154
5.8.4 Changing The Proportion of Classifiers Acted Upon By G.A ..... 160
5.8.5 Adding More Characteristics as Inputs ..... 162
5.8.6 Training Models For Each Scheme Option ..... 167
5.9 Results and Conclusions ..... 168
5.10 Deficiencies of The Classifier Method ..... 169
5.11 Summary. ..... 170
6 Additional Pattern Recognition Techniques - Application of Neural Networks. ..... 172
6.1 Neural Networks - General Overview ..... 172
6.1.1 Introduction - What is a Neural Network Model? ..... 172
6.1.2 Basis of Neural Network Principle - Polynomial Curve Fitting ..... 172
6.1.3 General Learning Techniques ..... 173
6.1.4 Types of Network ..... 174
6.2 Single Layer Neural Networks ..... 174
6.2.1 Introduction ..... 174
6.3 Multi-Layer Perceptrons (MLPs) ..... 176
6.3.1 Introduction ..... 176
6.4 Application to The Ordering Problem ..... 178
6.5 Inputs and Outputs For MLP Neural Network Model. ..... 179
6.5.1 Preferences For The Variable Ordering Schemes ..... 179
6.5.2 Fault Tree Characteristics and Scheme Preference Coding ..... 179
6.6 Training and Test Sets ..... 180
6.7 Creating A Neural Network Program. ..... 181
6.7.1 General Description of Program ..... 181
6.7.2 Details of The Multi-layer Perceptron Approach ..... 182
6.8 Training Procedure 1 - First Model ..... 193
6.8.1 End Aim of Trained Model. ..... 193
6.8.2 Data Set Up ..... 194
6.8.3 Setting Up The Network Architecture For The Problem ..... 195
6.8.4 MLP Architecture For Best Results. ..... 196
6.8.5 Altering The Precision of The Inputs and Outputs ..... 196
6.9 Optimisation Algorithm Changes - Enhanced Gradient Descent. ..... 197
6.10 Training Procedure 2 - Using The Enhanced Gradient Descent Technique ..... 198
6.10.1 Using Eleven Input Characteristics With New Optimisation Algorithm ..... 198
6.10.2 Best Network Architecture ..... 198
6.11 Training Procedure 3-Reviewing The Inputs and Outputs ..... 200
6.11.1 Input Characteristics Altered ..... 200
6.11.2 Reviewing Outputs ..... 202
6.12 Training Procedure 4 -New Data Set ..... 202
6.12.1 New Data Set - Removal of Redundant Trees ..... 202
6.13 Training Procedure 5-Re-testing Five Characteristics ..... 204
6.14 Establishing If Over -Training Is Occurring ..... 205
6.14.1 Consideration in Training - Length of Training Period ..... 205
6.15 Conclusions of Best Results ..... 208
6.15.1 Performance Based Assumptions ..... 208
6.15.2 Best Architecture ..... 208
6.15.3 Future Considerations. ..... 210
6.16 Summary ..... 210
7 Radial Basis Function Neural Networks ..... 212
7.1 Introduction ..... 212
7.2 Comparison With Multi-layer Perceptron Approach ..... 213
7.3 Advantages and Disadvantages of Radial Basis Functions Compared to Multi-layer Perceptrons ..... 214
7.3.1 $\quad$ The Advantages of The RBF ..... 214
7.3.2 The Disavantages of The RBF ..... 215
7.4 Dynamics of a Radial Basis Function Neural Network ..... 215
7.4.1 RBF Theory ..... 215
7.4.2 Types of Radial Basis Function ..... 218
7.4.3 Two Phases of Training - Optimisation of Parameters ..... 219
7.5 Programming A Radial Basis Function Network. ..... 223
7.5.1 The Main Program ..... 223
7.6 The Programming Routines Required ..... 224
7.6.1 Overview of Routines. ..... 224
7.6.2 Initialisation of The Network ..... 224
7.6.3 Programming Stage 1 of Training ..... 225
7.6.4 Programming The Second Phase of Training ..... 229
7.6.5 Programming The Predictive Phase ..... 230
7.7 Generating the Desired Network Architecture ..... 230
7.8 Results Using Four Characteristics As Inputs to Problem ..... 232
7.8.1 Using Three Radial Basis Function Centres ..... 232
7.8.2 Using Four Radial Basis Function Centres ..... 232
7.8.3 Using Five Radial Basis Function Centres ..... 233
7.8.4 Increasing Number of Centres to Six ..... 233
7.8.5 Using Seven Centres ..... 234
7.8.6 Thirty Centres ..... 235
7.8.7 Review of Four Characteristic Results ..... 235
7.9 Results and Findings Using Eleven Characteristics As Inputs to Problem ..... 236
7.9.1 Simple Architecture. ..... 236
7.9.2 Using Two and Three Radial Basis Function Centres in The Model ..... 236
7.9.3 Adding More Centres To The RBF Model ..... 237
7.10 Assessing The Accuracy of The Best Networks ..... 238
7.11 Next Research Steps ..... 241
7.11.1 Reducing Width Parameter ..... 241
7.11.2 Using Only Four Outputs to The Model ..... 243
7.11.3 Importance of Characteristics ..... 245
7.12 Summary ..... 246
8 Using Structural Importance Measures For BDD Variable Ordering. ..... 248
8.1 Introduction ..... 248
8.2 Problems With Re-writings of The Fault Tree Structure ..... 249
8.3 Definition of Importance Measures ..... 250
8.3.1 Understanding Importance Measures ..... 250
8.3.2 Deterministic Measures of Importance ..... 251
8.4 Calculation Method For Deterministic Structural Importance Measure ..... 251
8.5 Alternative Methods Of Calculating The Structural Importance Measure. ..... 255
8.5.1 Problems With The Hand Calculated Approach ..... 255
8.5.2 Using Birnbaums's Structural Importance Measure. ..... 255
8.6 Application of Numerical Structural Importance to The Ordering Problem ..... 257
8.6.1 Programming The Calculation Procedure. ..... 257
8.6.2 Results of Comparison ..... 258
8.6.3 Re-Ordering Components With Matched Structural Importance Measures ..... 260
8.7 Using the Approach as it Stands ..... 262
8.8 Alternative Methods For Calculating the Structural Importance Measure ..... 264
8.8.1 Highlighting Problems in Coding The Mathematical Measure From The Fault Tree ..... 264
8.9 Approximation Methods To Calculate the Structural Importance Measure ..... 265
8.9.1 Introduction to Procedures ..... 265
8.10 Approximation Method 1 - Pattern Identification ..... 265
8.11 Approximation Method 2 -Simpler Weighting Methods ..... 270
8.12 Approximation Method 3 - Applying Birnbaum's Importance Measure To The
Tree ..... 278
8.13 Summary of Structural Importance Measure Approach. ..... 282
9 Conclusions and Future Work. ..... 284
9.1 Summary of Work. ..... 284
9.2 Conclusions. ..... 287
9.3 Future Work. ..... 289
9.3.1 Examine The Input Characteristics of The Fault Tree. ..... 289
9.3.2 Applying Modularisation Techniques to The Fault Tree ..... 289
9.3.3 Continue Research With Neural Network Pattern Recognition Approaches ..... 289
9.3.4 Improve Approximated Structural Importance Measures ..... 290
Appendices. ..... 291
Appendix I - Characteristics of 51 Benchmark Fault Trees Used in Sinnamon Study [Sin96] ..... 291
Appendix II - Characteristics of Benchmark Fault Trees Used in Thesis. ..... 293
Appendix III - Number of BDD Nodes For Each Ordering Heuristic For All Benchmark Fault Trees Used in Thesis Studies ..... 298
Appendix IV - Remainder of Eleven Characteristics Used to Describe Fault Trees. ..... 325
Appendix V - Network Architectures Predicting 14/20 Correct Responses, With Eleven Input Nodes and Six Output Nodes ..... 330
Appendix VI - Results of Structural Importance Measure Variable Ordering Heuristic ..... 336
Appendix VII - Difference in BDDs Affected by The Structural Importance Measure With Additional Subroutine of Ordering Matched Components by The Number of Repetitions. ..... 341
Appendix VIII - Comparison of Results for BDD Size Using The Best of Six Ordering Heuristics and Simpler Weighting Methods For The Structural Importance Measures. 345
References ..... 350

## Publications

The research carried out as part of this thesis has led to the following publications and conference presentations:

## Journal Publications:

- L. M. Bartlett and J. D. Andrews. "Efficient Basic Event Ordering Schemes for Fault Tree Analysis". Quality and Reliability Engineering International, vol. 15, 1999, p95-101.
- L. M. Bartlett and J. D. Andrews. "An Ordering Heuristic To Develop The Binary Decision Diagram Based on Structural Importance". Submitted to Reliability Engineering and System Safety, 2000.
- L. M. Bartlett and J. D. Andrews. "Selecting An Ordering Heuristic For The Fault Tree to Binary Decision Diagram Conversion Process Using Neural Networks". Submitted to IEEE Transactions on Reliability, 2000.
- L. M. Bartlett and J. D. Andrews. "Comparison of Two New Approaches to Variable Ordering for Binary Decision Diagrams". Submitted to Quality and Reliability Engineering International, 2000.


## Conference Presentations:

- J. D. Andrews and L. M. Bartlett. "Efficient Basic Event Orderings For Binary Decision Diagrams". Proceedings of the Annual Reliability and Maintainability Symposium, Anaheim, Los Angeles, 1998, p61-68.
- L. M. Bartlett and J. D. Andrews . "Efficient Basic Event Ordering Schemes for Fault Tree Analysis". Advances in Reliability Techniques Symposium (ARTS), Manchester, 1998.
- L. M. Bartlett and J. D. Andrews. "Comparison of Variable Ordering Heuristics/Algorithms for Binary Decision Diagrams". Proceedings of the Safety and Reliability Society Symposium '99, Advances in Safety and Reliability, 12, June 1999, pl-15.
- L. M. Bartlett \& J. D. Andrews "Using Neural Networks To Establish an Efficient Binary Decision Diagram Ordering". Invited Presentation at Young OR 11 Conference, Cambridge, March 2000.
- J. D. Andrews and L. M. Bartlett. "Binary Decision Diagram Ordering Heuristics Based On Structural Importance". Invited Paper at MMR (International Mathematical Models in Reliability) Conference, Bordeaux, July 2000.
- L. M. Bartlett and J. D. Andrews. "Comparison of Two New Approaches to Variable Ordering for Binary Decision Diagrams". Submitted to ARTS, Manchester, November 2000.


## 1 Introduction

### 1.1 Introduction to Risk and Reliability

Everyday industrial processes can carry some form of risk, sometimes financial and sometimes to our surroundings or us. Failure of some industrial systems can be catastrophic, resulting in loss of life, for example, the local Kegworth aeroplane disaster in January 1989. This resulted from an engine fire on the 737-400 British Midlands plane and caused 47 deaths. More well known industrial disasters include the explosion on the Piper Alpha oil platform ${ }^{[C u 190]}$, the explosion at the U.K Based Flixborough chemical plant ${ }^{[L e e 80]}$, and the release of chemicals from a reactor at the Icmesa Chemical Company in Seveso, Italy ${ }^{[\text {Lee80] } . ~ R e l i a b i l i t y ~ a n d ~ r i s k ~ a s s e s s m e n t ~ m e t h o d s ~ a r e ~ u s e d ~ i n ~ m a n y ~}$ industrial applications to assess the safety of systems. Decisions resulting from the application of these methods can prevent the occurrence of hazardous incidents or at least reduce the probability below a suitable limit.

Risk and reliability assessment techniques have developed since the time of the Second World War. Contributions from advances in nuclear, defence and aircraft safety have led to the degree of sophistication seen in the reliability techniques used today. Reliability and risk assessment methods are both used in safety studies to establish the various combinations of faults that can lead to a potential hazard.

Reliability can be defined in a number of ways, a broad definition of reliability is:
> "Reliability is the science aimed at predicting, analysing, preventing and mitigating failures over time" ${ }^{[\mathrm{He} 998]}$.

More specifically, reliability can be defined as:
> "The probability that a device will operate successfully for a specified period of time and under specified conditions when used in the manner and for the purpose intended" ${ }^{[H e a 98]}$.

Traditionally reliability was concerned with answering the question "How long will an object continue to function before it fails?". Answering this question however, can only be achieved in probabilistic terms and nowadays knowing how to reduce the associated risk is also required.

In terms of analysing risks, no matter how much money is spent on trying to reduce the risk of a hazardous or fatal incident occurring it is impossible to avoid risks entirely. In the end society has to judge how much money it is worth spending to save each additional life. Risk or "expected loss" $(R)$ can be defined quantitatively as the product of the consequence of a specific incident $(C)$ and the probability or frequency of occurrence ( $P$ ):

$$
R=C * P .
$$

Hence, the reduction of any risk can be achieved by reducing the consequence of the incident ( $C$ ), or by reducing the probability of its occurrence $(P)$. The societal judgement or economic decision concerning a risk generates what is known as criteria of acceptability. The H.S.E (Health and Safety Executive) set these acceptance criteria based on what they have termed an 'ALARP' principle. Low risks are permissible, high risks are not accepted and events with a reasonable risk have to satisfy the ALARP principle. This means that risks must be As Low As Reasonably Practicable. Therefore, if to reduce the current risk by only a small fraction would cost an excessive sum of money then this may be regarded as not practicable, and hence the current risk would satisfy the ALARP principle.

Quantitative risk assessment involves four basic stages:

1. The identification of the potential safety hazards.
2. The estimation of the consequences of each hazard.
3. The estimation of the probability of occurrence of each hazard.
4. A comparison of the results of the analysis against the acceptability criteria.

The main concern of this chapter is to describe some of the more commonly used reliability assessment techniques. That is the methods that can be used to predict the reliability performance of a system in terms of the reliability performance of the components with which the system is constructed.

### 1.2 System Reliability Modelling Techniques

### 1.2.1 Introduction

There are several techniques that can be used in developing a system reliability model. Some techniques use a qualitative approach and others a quantitative approach to modelling the system's reliability. The most popular qualitative approach is Failure Mode and Effects Analysis (FMEA) ${ }^{\text {[AMO93] }}$. The information gathered during the execution of this technique can often provide inputs to some of the quantitative methods. These quantitative techniques include Reliability Block Diagrams ${ }^{[A M 093]}$, Fault Tree Analysis ${ }^{[\text {AM093] }}$, Event Tree Analysis ${ }^{[\text {AM093] }}$, Markov models ${ }^{[\mathrm{HKu} 81]}$, and Simulation approaches ${ }^{[\mathrm{HKu} 81]}$. Fault tree analysis plays a prominent role in many industrial assessments and event tree analysis is becoming more widely used. These two methods will be discussed in brief.

### 1.2.2 Fault Tree Analysis

H. A. Watson ${ }^{[\text {Wat61] }}$ first introduced the fault tree analysis technique in the 1960 's. Since this time fault tree analysis has been extensively used in safety and reliability studies to
identify the combinations of events which could cause the failure of the system, and ultimately result in a possible hazardous event.

Construction of the fault tree defines how the components within a system interact to yield the system failure event. A fault tree diagram can graphically represent and document the failure of a system in terms of the failure of its components. The tree is developed from the start point of the undesired event or failure mode. Branches lead down from the top event to other sub-events that show its possible causes. These subevents are continually redefined until the branches are terminated with component failures, termed basic events. These are the events for which failure/repair data is required. Each fault tree is constructed to form a series of gates and events, the gates link the events together depending on their causal relationships. A number of fault trees may need to be drawn to cover all the possible failure modes of a system. Each tree will represent a distinct failure mode. A simple tree structure is shown in figure 1.1.


Figure 1.1: Simple Fault Tree Diagram

The fault tree shown in figure 1.1 develops the top event failure mode of "Total Failure of Safety System". The immediate, necessary and sufficient causes of this are "Failure of

Sub System 1" OR "Failure of Sub System 2". The failure of subsystem 1 is caused by the simultaneous occurrence of the "Failure of Pressure Transmitter 1" AND the "Failure of Pressure Transmitter 2". As data (i.e. failure/repair probabilities) are available for these events the fault tree can be terminated at this stage. If data were not available the branching procedure would continue until events occurred where data was available. The failure of the second subsystem again has available data and the branch is terminated by the basic event, termed $A$.

Both qualitative and quantitative evaluations can be performed on the tree structure. The qualitative analysis involves establishing the causal relationships between the components of the system, hence outlining the groups of components which when they fail together cause the system to fail. The quantitative analysis procedure involves calculating the risk of failure of the system, i.e. the top-event or system failure mode probability. The quantitative analysis provides the top event reliability parameters necessary to determine whether the risk of the undesired event is sufficiently small and hence the safety of the system is acceptable. The method is well proven and computer programs are available to calculate the top event probability for a constructed fault tree.

The fault tree approach is the main topic of this thesis and is described in more detail in chapter 2.

### 1.2.3 Event Tree Analysis

Like the FMEA approach, event tree analysis is an inductive technique whereby the process begins by identifying a particular hazardous trigger or initiating event. From this event paths are traced to identify all the possible resulting consequences. The tree structure comprises a number of branch points, which usually represent the success or failure of different subsystems that can respond to the initiating event. Fault tree analysis is used in conjunction with this technique to identify the causes of the subsystem failures or branch events, and hence quantify the failure. If the subsystem responding to the
initiating event functions then a top branch represents this (labelled W). The lower branch indicates subsystem failure (labelled F).

The analysis procedure looks at each subsystem in succession, represented by a path, which leads to the final consequence to be determined. A simple example event tree is shown in figure 1.2 for a nuclear reactor power loss. The first safety feature of the nuclear reactor in the event of loss of an off-site electrical power supply is the shutdown of the reactor. This shutdown of the reactor is shown as the second event in the sequence. If this occurs, it has a defined probability $R_{r t}$, located on the working (W) branch of the diagram, however, if the shutdown fails, the consequences would be sufficiently severe that the later safety features would be irrelevant. The process of evaluating the outcome of each successive subsystem is continued until all subsystems have been considered. To calculate the probability of occurrence of each path is simply a matter of multiplying the initiating event frequency by the probabilities of each of the branches, assuming that the events are independent.


Figure 1.2: Event Tree Analysis of a Nuclear Reactor Power Loss

For example, in figure 1.2, the frequency of path $P 1$ is:

$$
\begin{aligned}
F_{P I}= & \text { Frequency of the Initiating Event } \\
& * \operatorname{Prob}[\text { reactor shutdown works] } \\
& * \operatorname{Prob}[\mathrm{emergency} \text { electrical supply works] } \\
& * \operatorname{Prob}[\text { emergency core cooling works] } \\
& * \operatorname{Prob}[\text { primary containment integrity works] }
\end{aligned}
$$

$$
F_{P I}=F_{i n i t} * R_{r t} * R_{e s} * R_{e c} * R_{p c}
$$

Each of the outcomes in figure 1.2 can have different consequences, therefore to evaluate the risk of each outcome the frequency of the outcome must be multiplied by the consequence. To evaluate the risk caused by the initiating event (i.e. in this example loss of off-site electricity) the sum of the path frequency multiplied by the consequence for each outcome needs to be calculated. The risk is expressed by equation 1.1, where $i$ refers to each outcome, $F$ is the frequency of the outcome and $C$ is the consequence of the outcome.

$$
R I S K=\sum_{i} F_{i} C_{i}
$$

### 1.3 Selecting an Analysis Method

### 1.3.1 Reasons For Fault Tree Popularity

Fault tree analysis is the most popular reliability technique that is used in industry to assess the safety and reliability of a system. Although the technique has some drawbacks (discussed in section 1.4.1) overall it has a methodology which is appropriate to apply for an efficient analysis of a broad range of systems. The method provides a systematic procedure for identifying failure causes, which is easy to follow and carry out. The analysis can identify the system failure modes and indicate which are the most likely to occur. This provides information to support system redesign to reduce the number of
these failure modes. The method requires manual construction of the tree, which requires the analyst to understand the system thoroughly.

### 1.3.2 Qualitative and Quantitative Analysis in Brief

Both a qualitative and quantitative analysis of the fault tree structure can be carried out. The qualitative analysis of the fault tree involves identifying the causes of system failure. Once a fault tree has been constructed for a specific system failure event a qualitative assessment that produces the minimal cut sets can be performed. A minimal cut set can be defined as:
"The smallest combination of component failures, which if they all occur will cause the top event to occur".

A mathematical logic expression for the top event in terms of its minimal cut sets can be written as:

$$
T O P=C_{1}+C_{2}+C_{3}+\cdots+C_{n}
$$

where $C_{i}, i=1, \ldots, n$, are the minimal cut sets, and ' + ' represents the logical OR operator. Each minimal cut set consists of a combination of basic events, which occur simultaneously and can be expressed as:

$$
C_{i}=X_{1} \cdot X_{2} \cdot \cdots \cdot X_{m}
$$

where $X_{i}, i=1, \ldots, m$ are basic event failures, and the symbol ' $\because$ ' represents the logical AND operator. For example, if

$$
T O P=A+B \cdot C+D \cdot E
$$

there are three minimal cut sets (one single order, and two second order), A, B.C and $D . E$. The order of the minimal cut set is the number of events it contains.

The methods to obtain the minimal cut sets of the fault tree are discussed in detail in chapter 2.

Once the minimal cut sets of the fault tree have been generated then quantification can begin. In the quantification process the following parameters can be calculated:

1. The probability of the top event occurrence.
2. The top event failure rate.
3. The expected number of occurrences of the top event in a specified time period.
4. The importance measures indicating the component's contribution to the top event.

The number of terms in the formula to calculate the exact top event probability is dependent upon the number of minimal cut sets in the fault tree. For large trees, with high numbers of minimal cut sets, it is not possible to fully evaluate these expressions using conventional techniques and approximations are required.

The importance measures calculate the contribution of each basic event to the top event occurrence. There are a number of importance measures that can be calculated, each providing different information on the relative event contribution. Four possible measures are:

1. The Structural Measure of Importance, which assesses the importance of a component to the system operation considering only the components place within the system structure.
2. Birnbaum's Measure of Importance, which can be defined as the probability that the system is in a critical system state for component $i$ i.e. it is the probability that the system fails only if component $i$ fails.
3. The criticality measure of importance, which is defined as the probability that the system is in a critical state for component $i$ and $i$ has failed given that the system has failed.
4. The Fussell-Vesely Measure of Importance, which is defined as the probability of the union of the minimal cut sets containing $i$ given that the system has failed.

The mathematical derivations of all of these quantification measures are explained in detail in chapter 2.

### 1.4 The Binary Decision Diagram Approach

### 1.4.1 Advantages

From an analysis perspective, if the fault tree is very complex then finding the minimal cut sets can require extensive computer processing capability. Another limitation is that when the tree is complex approximation methods need to be used to find the top event probability. Research to further enhance the efficiency of the fault tree methodology has proved difficult, as the current techniques that analyse the fault tree structure based on the Kinetic Tree Theory of Vesely ${ }^{[\text {[Ves } 70]}$ are already so well developed. In recent years improvements have been made in the qualitative and quantitative analysis procedures by using a completely new approach, that of Binary Decision Diagrams. The Binary Decision Diagram approach has been shown to improve both the efficiency of determining the minimal cut sets of the fault tree and also the accuracy of the calculation procedure used to determine the top event parameters. The qualitative and quantitative procedures employed in this technique are discussed in chapter 3.

### 1.4.2 Advances to The Binary Decision Diagram Technique

The fault tree diagram must be converted to an equivalent Binary Decision Diagram (BDD). This conversion process involves selecting all of the basic events (terminal events) of the fault tree and placing them in an order. From this ordering each basic event is selected in turn to successively construct each branch of the BDD diagram. Research in this area has found that the ordering of the basic events is crucial to the size of the resulting diagram. Using an inefficient ordering scheme will produce a nonminimal BDD structure. Alternative ordering schemes will produce BDDs of different sizes, the smaller the BDD the more optimal the diagram.

Therefore, a current problem in the BDD methodology is that of ordering the variables when the fault tree is converted to the BDD. This is critical for the analysis procedure to be efficient. The objective would be to produce an ordering scheme that achieves the 'best' BDD for all fault tree structures, hence providing the basis for an efficient analysis of the tree structure and corresponding failure mode. It is this ordering problem that is the focus of the research presented within this thesis.

### 1.5 Summary

- Fault tree analysis is the most commonly applied reliability modelling technique that is used in industrial applications.
- The fault tree provides a clear diagrammatic representation of the failure modes of the system under analysis.
- The fault tree analysis technique encompasses both a qualitative and quantitative procedure, which is well developed and computer packages are available to carry out the procedures.
- The only limitations of the analysis are in efficiency and accuracy when dealing with large fault tree structures, with hundreds and possibly thousands of basic events.
- These limitations with the fault tree method promote the use of Binary Decision Diagrams.
- Binary Decision Diagrams are qualitatively more efficient and quantitatively more exact than the predictions made by the fault tree methodology.
- BDDs could be very useful when combined with the event tree analysis methodology.
- The BDD approach still requires some development in terms of efficient strategies for the ordering of the components from the fault tree diagram.


### 1.6 Scope of Research

Using the advantages of the fault tree technique and replacing its inefficiency and inaccuracy of the analysis procedure with one that is better can only lead to an overall improvement in the modelling technique. Therefore, with the Binary Decision Diagram currently appearing to be the best way forward to analysing the fault tree diagram in a more efficient and accurate manner, this approach needs to be fully developed. Understanding and producing an optimum means of ordering the fault tree variables to produce an efficient BDD would further enhance the BDD methodology. It is this variable ordering dilemma which is the topic of study in this thesis.

Despite the BDD methodology being a relatively recent development, a number of research studies have focussed on the variable ordering dilemma. The current variable ordering heuristics are discussed in this thesis (chapter 4) as well as new ideas to solve the problem (chapters 5-8). The overall aim of this thesis is to produce a means of identifying the best ordering permutation for any given fault tree structure, whether this be by selecting from a set of current alternatives or generating a new and better ordering heuristic. This would then promote the efficiency of the BDD technique and enable a commercially available package to be produced. Can just one scheme be highlighted to predict the optimal BDD for any fault tree? Or perhaps a technique produced that can choose the best scheme from a set of alternatives.

## 2 Fault Tree Analysis

### 2.1 Introduction

Many industrial processes carry the risk of an undesirable system failure. Reliability studies are undertaken to quantify risks within a system and aid in the process of judging their acceptability. Fault tree analysis, FTA, is one of the most commonly used techniques for safety system analysis. This technique is a deductive, or backward analysis procedure whereby a "what can cause this" approach is taken.

This analysis procedure yields a complete description of the various causes of system failure. A system is made up of a number of smaller subsystems, which are in turn, comprised of basic elements or components. These components are linked together in a specific structure to facilitate the system function. A fault tree diagram can diagrammatically represent the failure of a system due to contributions of failures of the components. The system failure mode, or undesired event, is termed the 'top event' and is the event at the top of the fault tree diagram. Branches lead down from the top event to other intermediate events that show its possible causes. A system may have a number of failure modes and therefore a system safety assessment may require many fault tree diagrams.

Two main objectives of the analysis process are:

1. To determine the causal relationship between the components of a system.
2. To determine whether the risk of failure of the system is sufficiently small and hence the safety of the system is acceptably high.

The evaluation of the fault tree can be qualitative, as in objective 1 , or quantitative, as in objective 2. Both qualitative and quantitative evaluations can be performed on the tree structure depending on the type of analysis being carried out.

This chapter explains the fault tree analysis technique in detail. Each stage of the process is examined from the construction of the fault tree diagram to the qualitative and quantitative analysis procedures involved. Finally, the advantages and disadvantages of the method are discussed and possible means to improve upon the efficiency of the fault tree methodology are highlighted.

### 2.2 Background

Numerous notorious accidents over the years have heightened awareness to the fact that failure of industrial systems pose major risks to the public and workforce. To reduce these risks certain safety standards for individual industrial processes have been put in place. With the emergence of acceptability criteria for risks many industrial processes require the possible hazards within the system to be assessed, and hence, the use of risk assessment techniques, such as fault tree analysis, is now common place.

Reliability first became a concern of the aircraft industry after the First World War. Initial efforts were merely based on a trial and error approach however, with the slow development of failure data came the techniques used in reliability studies today.
H. A. Watson (1961/2) first introduced fault tree analysis during a study of the launch control system of the Minuteman intercontinental ballistic missile ${ }^{[\text {Wat6 } 1]}$. Reliability techniques have found applications in such fields as the process plant, automotive, engineering, aeronautic and military establishments.

### 2.3 Description of The Fault Tree Construction Procedure

### 2.3.1 Fault Tree Symbols

A fault tree diagram contains two basic elements, 'gates' and 'events'. There are two main types of event: an intermediate event; and a basic event (as shown in table 2.1). The term event represents elements in a system that can change state. In terms of a basic event this change of state is referring to the transference from a working to a failed state, or vice versa. For intermediate events the possible changes are between occurrence and non-occurrence of the event. Gates represent the causal relationship between these events, which lead to the top event.

| Event Symbol: | Meaning |
| :---: | :---: | :---: |
| 1 | Intermediate Event |
| (further developed by a gate) |  |

Table 2.1: Main Event Symbols Contained in a Fault Tree

The rectangle is the symbol used to define an intermediate event. This type of event is the output representing the relationship between the gate and the gates' inputs that lie below it in the tree (as shown in figure 2.1). The event is termed 'intermediate' because it can be further developed in terms of either other intermediate events or basic events. The circle defines a basic event, the smallest resolution item in the system for which data
is available. Data is required for the basic events in the quantitative analysis of the fault tree.

There are other event symbols, for example, house events, undeveloped events and conditional events. These are all discussed in detail in Andrews and Moss ${ }^{[\text {AMo93] }}$, but are not considered in this thesis.


Figure 2.1: Relationship Between Events and Gates

There are three main mathematical logic operators, each have their own gate symbol, namely an AND, OR and NOT gate (shown in table 2.2). These combine events in the same way as the Boolean operators of 'union', 'intersection', and 'complementation'. There is therefore a one-to-one correspondence between Boolean algebraic expressions and the fault tree structure.

Other gate symbols include, for example, vote gate, exclusive OR gate, and a priority AND gate to name a few. Details are given in Andrews and Moss ${ }^{[\text {AMo93] }}$.

|  | Gate Symbol | Gate Type | Causal Relation |
| :--- | :---: | :---: | :---: |

Table 2.2: Three Main Types of Gate Used in The Fault Tree

### 2.3.2 Construction Methodology

The construction procedure requires expert knowledge of all the components of the system and how they function. The structure of the whole fault tree is illustrated by a simple example as shown in figure 2.2. The structure is represented by a combination of events that are logically related to the 'top event' by OR gates. The combinations of events that lead to the 'top event' are all shown below the 'top event'. The intermediate events are represented by the rectangles, and the tree is terminated when component failure events or basic events are encountered.


Figure 2.2: An Example Fault Tree to Illustrate The Basic Events of The Top Event: 'Failure of Water from Hose'

The fault tree (shown in figure 2.2) illustrates the causes of the failure mode "Failure of Water from Hose". The two possible immediate causes of this event are either "No Water supply to Nozzle" or "No Water From Nozzle". In the latter instance, the failure corresponds to that of a nozzle and data is available for this event. Hence this is represented by a circle, which is termed a basic event. The other event needs further development. This can be caused by "No Water to Hose" or "Hose Blocked". If either the "Tap is Blocked" or there is "No Water" this will cause the intermediate event of "No Water to Hose". This is the causal relationship between the basic events $T$ and $W$, as represented by the OR gate in the diagram.

### 2.4 Qualitative Analysis Of The Fault Tree

### 2.4.1 Definition of The Two Types of Fault Tree

Fault tree structures can be categorised in two ways: coherent fault trees; and noncoherent fault trees. The simple definition of a coherent fault tree structure is:

1. The top event fails if all the components fail.
2. The top event works if all the components work.
3. The structure function (or logic function representation of the fault tree structure, discussed in section 2.5.3) must be monotonically increasing.

For example, the minimal cut set $A B \bar{C}$ can not come from a coherent structure, because the functionality of $C$ contributes to the system working. Coherent structures consist of only AND and OR gates and do not contain NOT logic.

The combinations of events that cause the top event in a non-coherent fault tree leads to the definition of an implicant set. An implicant set is a combination of basic events (successes or failures) that produces the top event. Analogous to the coherent alternative, the collection of basic events (successes or failures) which are both necessary and sufficient to cause the top event are termed the Prime Implicant Set. Conventional approaches to fault tree reduction do not deliver all prime implicants for every noncoherent tree hence methods to obtain the prime implicant sets are discussed in section 2.4.4.

### 2.4.2 The Analysis Procedure

The qualitative analysis of the fault tree involves identifying the possible causes of system failure. To establish these causes, the specific failure modes (types of failure) of the system must be identified. Once a fault tree has been constructed for a specific top
event a qualitative assessment that produces the minimal cut sets can be performed. The failure modes of a fault tree can be defined by the concept of a cut set.

A cut set is a collection of basic events such that if they all occur the top event also occurs.

For large fault tree structures there may be hundreds or even thousands of cut sets produced, however not all of these may be minimal. Minimal means that they are necessary and sufficient to produce system failure. For example, given that $A B$ is a cut set such that if both $A$ and $B$ occur the top event will occur. If however, an additional cut set $A$ exists, then the failure of $A$ alone will cause the top event to occur, then the state of $B$ is irrelevant and the top event will occur regardless of whether $B$ has failed or not. This introduces the concept of a minimal cut set.

## A minimal cut set is the smallest combination of component

 failures, which if they all occur will cause the top event to occur.The size (or order) of a minimal cut set is the number of components within the set. Lower order minimal cut sets, in general, contribute more to the failure of the top event. One component minimal cut sets (or first-order minimal cut sets), if they occur, represent single failures that cause the top event. Two component minimal cut sets (or second order) represent double failures such that if both components fail this will cause the top event.

The expression for the top event in terms of its minimal cut sets can be written as:

$$
T O P=C_{1}+C_{2}+C_{3}+\cdots+C_{n}
$$

where $C_{i}, i=1, \ldots, n$, are the minimal cut sets, and ' + ' represents the logical OR operator. Each cut set can consist of one or more basic events, where the '.' represents the logical AND operator. For example, if the expression of the top event is:

$$
T O P=A+B \cdot C+D \cdot E \cdot F
$$

then there are three minimal cut sets, one first order, $A$, one second order, B.C, and one third order, D.E.F .

Quantification of the fault tree can occur once the minimal cut sets have been calculated.

### 2.4.3 Obtaining Minimal Cut Sets For Coherent Trees

### 2.4.3.1 Boolean Algebraic Laws

The conventional approach to calculate the minimal cut sets of the fault tree involves taking the logic expression for the top event and transforming it into a disjunctive normal form (minimal sum-of-products form). Either a 'top-down' approach or 'bottom-up' approach can be used, with the difference residing in the starting location to begin the expansion process, i.e. the top gate or the bottom gate of the tree (the top-down method is discussed in section 2.4.3.2). Boolean algebra laws given below are used to remove redundancies in the expressions.

1. Commutative laws:

$$
\begin{aligned}
& A+B=B+A, \\
& A . B=B . A
\end{aligned}
$$

2. Associative laws:

$$
\begin{aligned}
& (A+B)+C=A+(B+C), \\
& (A \cdot B) \cdot C=A \cdot(B \cdot C)
\end{aligned}
$$

3. Distributive laws:

$$
\begin{aligned}
& A+(B \cdot C)=(A+B) \cdot(A+C), \\
& A \cdot(B+C)=A \cdot B+A \cdot C
\end{aligned}
$$

4. Identities:

$$
\begin{array}{ll}
A+0=A, & A+1=1 \\
A .0=0, & A .1=A
\end{array}
$$

5. Idempotent law:

$$
\begin{aligned}
& A+A=A, \\
& A . A=A
\end{aligned}
$$

6. Absorption Rule:

$$
\begin{aligned}
& A+A \cdot B=A, \\
& A \cdot(A+B)=A
\end{aligned}
$$

7. Complementation:

$$
\begin{aligned}
& A+\bar{A}=1, \\
& A \cdot \bar{A}=0, \\
& \overline{(\bar{A})}=A
\end{aligned}
$$

8. De Morgan's laws:

$$
\begin{aligned}
& \overline{(A+B)}=\bar{A} \cdot \bar{B}, \\
& \overline{(A \cdot B)}=\bar{A}+\bar{B}
\end{aligned}
$$

### 2.4.3.2 The 'Top Down' Approach

The common approach to obtain the minimal cut sets is to calculate a Boolean logic expression of the top event in a 'top-down' manner. Each gate is represented as a logic expression of its inputs, where the dot or product is representative of the AND gate and the sum is used to represent the OR gate. The top-down approach starts with the top gate and expands the gates with inputs that lie below it in the tree. Using the Boolean laws of
algebra the minimal cuts sets of the fault tree given in figure 2.3 can be calculated as follows:

Starting from the TOP gate, this is an AND gate with two inputs $G 1$ and $G 2$, this can be expressed as:

$$
T O P=G 1 . G 2
$$

Looking at G1 and G2, both are OR gates, these can be expressed as:

$$
\begin{gathered}
G 1=B+G 3 \\
G 2=A+B+C
\end{gathered}
$$

TOP now becomes:

$$
T O P=(B+G 3) \cdot(A+B+C)
$$

Expanding and simplifying before the next substitution gives:

$$
\begin{aligned}
& T O P=B \cdot A+B \cdot B+B . C+G 3 . A+G 3 \cdot B+G 3 . C . \\
& T O P=B+G 3 . A+G 3 . C \quad(\text { Since } B \cdot B=B \text { and } A \cdot B+B=B)
\end{aligned}
$$

Expanding G3: $G 3=$ D.E. Substituting this into the TOP expression becomes:

$$
T O P=B+(D \cdot E) \cdot A+(D \cdot E) \cdot C
$$

Hence, the minimal disjunctive normal form (sum-of-products form) of the logic expression of the top event is:

$$
T O P=B+D \cdot E \cdot A+D \cdot E \cdot C
$$

In this minimal form the minimal cut sets can be extracted. The minimal cut sets for this example are: $\{B\}$ - first order, $\{D, E, A\}$-third order, and $\{D, E, C\}$ - third order.


Figure 2.3: Example Fault Tree to Illustrate Calculation of The Minimal Cut Sets

Other methods to extract the minimal cut sets, for example the bottom-up approach and various computer algorithms, are discussed in Andrews and Moss ${ }^{\text {[AM093] }}$.

### 2.4.4 Obtaining Prime Implicants of Non-Coherent Trees

Obtaining the prime implicants of a non-coherent fault tree requires some additional work than that of obtaining the cut sets of coherent trees. Consider the non-coherent fault tree drawn in figure 2.4, which can be restructured to an equivalent fault tree using De Morgans Laws (shown in figure 2.5). The NOT gate is gradually pushed down the fault tree until the negation applies to the basic events, such that:
$\begin{array}{lll}\text { Complementing } & G 3=A+C: & \overline{A+C}=\bar{A} \cdot \bar{C} \\ \text { Complementing } & G 4=D+E: & \overline{D+E}=\bar{D} \cdot \bar{E}\end{array}$


Figure 2.4: Example Non-Coherent Fault Tree

By applying the top down approach (as used with coherent fault trees) to generate the logic expression of the top event of the fault tree shown in figure 2.5, we find:

$$
\begin{aligned}
& T O P=G 1+G 2 \\
& T O P=G 3 \cdot G 4+A \cdot B \\
& T O P=(\bar{A} \cdot \bar{C}) \cdot(\bar{D} \cdot \bar{E})+A \cdot B \\
& \therefore \quad T O P=\bar{A} \cdot \bar{C} \cdot \bar{D} \cdot \bar{E}+A \cdot B
\end{aligned}
$$



Figure 2.5: Equivalent Fault Tree to Figure 2.4

Unlike coherent fault trees this Boolean reduction may not yield a complete list of failure modes and additional work is required to find all the prime implicants. Within the last expression there may be hidden prime implicants until such revealing techniques as the consensus theorem are applied ${ }^{[\text {Men70] }}$.

The consensus theorem can be explained as follows:

Given two fundamental conjunctions (products) $\psi 1$ and $\psi 2$. If there is precisely one literal $p$ which occurs negated in one of $\psi 1$ and $\psi 2$ and un-negated in the other, the fundamental conjunction obtained from $\psi 1 . \psi 2$ by deleting $p$ and $\bar{p}$ and omitting repetitions of any literal is called the consensus of $\psi 1$ and $\psi 2$. For example, the consensus of $A \cdot \bar{B}$ and B.C.E is A.C.E.

Applying this theorem to the last expression gives:

The consensus of $\bar{A} \cdot \bar{C} \cdot \bar{D} \cdot \bar{E}+A \cdot B$ is $\bar{C} \cdot \bar{D} \cdot \bar{E} \cdot B$

Therefore the 'full' Boolean expression of the TOP event is:

$$
T O P=\bar{A} \cdot \bar{C} \cdot \bar{D} \cdot \bar{E}+A \cdot B+\bar{C} \cdot \bar{D} \cdot \bar{E} \cdot B
$$

This generates the three prime implicants of the non-coherent fault tree in figure 2.5.

Another commonly used algorithm to find all the prime implicants is that of Nelson's algorithm ${ }^{[\text {Nel54] }}$. This method involves taking the 'dual' of the Boolean expression. The 'dual' is the complementation of the whole top event expression. Nelson's algorithm was developed to manipulate logic functions to obtain all the prime implicants. This was then applied to non-coherent fault trees by performing the operation $d(d(F)$ ) where $d(F)$ corresponds to the dual of the Boolean expression $F$, for the top event of the fault tree. The method can be explained in two steps:

Step 1: Complement $F$ to give $\bar{F}$, expand $\bar{F}$ into disjunctive normal form (sum-of-products), drop zero products ( $p \cdot \bar{p}=0$ ), repeated literals ( $p \cdot p=p$ ) and subsumming products ( $p+p \cdot q=p$ ) and call the result $\bar{\phi}$.

Step 2: Complement $\bar{\phi}$ to give $\phi$, expand to disjunctive normal form, remove zero products, repeated literals and subsumming products and call the result $\Theta$.

Nelson proves that $\Theta$ is the sum of all, and only, the prime implicants of $F$. To illustrate consider the logic expression of the fault tree shown in figure 2.5:

$$
T O P=\bar{A} \cdot \bar{C} \cdot \bar{D} \cdot \bar{E}+A \cdot B
$$

Step 1: Complement:

$$
\begin{aligned}
\overline{T O P} & =\overline{(\bar{A} \cdot \bar{C} \cdot \bar{D} \cdot \bar{E}+A \cdot B)} \\
& =(A+C+D+E) \cdot(\bar{A}+\bar{B})
\end{aligned}
$$

Expand:

$$
\overline{T O P}=A \cdot \bar{A}+A \cdot \bar{B}+C \cdot \bar{A}+C \cdot \bar{B}+D \cdot \bar{A}+D \cdot \bar{B}+E \cdot \bar{A}+E \cdot \bar{B}
$$

Removal:

$$
\bar{\phi}=A \cdot \bar{B}+C \cdot \bar{A}+C \cdot \bar{B}+D \cdot \bar{A}+D \cdot \bar{B}+E \cdot \bar{A}+E \cdot \bar{B}
$$

Step 2: Complement: $\phi=\overline{(A \cdot \bar{B}+C \cdot \bar{A}+C \cdot \bar{B}+D \cdot \bar{A}+D \cdot \bar{B}+E \cdot \bar{A}+E \cdot \bar{B})}$

$$
=(\bar{A}+B) \cdot(\bar{C}+A) \cdot(\bar{C}+B) \cdot(\bar{D}+A) \cdot(\bar{D}+B) \cdot(\bar{E}+A) \cdot(\bar{E}+B)
$$

Expanding and removing:

$$
\Theta=\bar{A} \cdot \bar{C} \cdot \bar{D} \cdot \bar{E}+B \cdot \bar{C} \cdot \bar{D} \cdot \bar{E}+A \cdot B
$$

This is the same result as with applying the consensus theorem.

### 2.5 Quantitative Analysis

### 2.5.1 Background of The Analysis Procedure

The basis of the fault tree methodology is developed from Kinetic Tree Theory ${ }^{[V e s 70]}$. When initially developed Kinetic Tree Theory was a major advancement in the field of reliability and safety system analysis. Basically, the methodology permits the timedependent analysis of the reliability characteristics of a system, or more precisely, the evaluation of the fault tree of the system. The methodology forms the fundamental part of the majority of commercial fault tree packages. Implementation of the method does have its limitations, whereby approximations need to be used when quantifying the top event failure characteristics.

### 2.5.2 Component and System Quantification Measures

In the quantification process there are a number of mathematical parameters that can be predicted. To quantify the top event performance the numerical quantification of all the components effecting the top event need to be evaluated. The numerical parameters relating to component performance are expressed in terms of the component's failure probabilities.

Important parameters, which represent component performance for the repairable component, are the unconditional failure and repair intensities, and the conditional failure and repair intensities of a component.

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

Therefore, within a set of components which all function at $t=0$, at time $t=4$, say, the unconditional failure intensity is calculated as the number of components which
within the unit of time from $t=4$ to $t=5$, change from a working state to a failed state, divided by the total number of components in the set.
2. The unconditional repair intensity, $v(t)$, is defined as the probability that a failed component is repaired per unit time at $t$ given that it worked a $t=0$.

In a set of components that are working at time $t=0$, the numerator of this parameter is calculated as the number of components that change from a failed state to a working state in the unit of time, specified by $t$. The denominator, like in the unconditional failure intensity, is simply the total number of components in the set.
3. The conditional failure intensity, $\lambda(t)$, is defined as the probability that a component fails per unit time at $t$ given that it was working at time $t$ and at time zero.

The difference with this and the unconditional intensity is that this parameter only considers those components within the set that are actually working at time $t$. So whereas in a set of 10 components, the denominator for the unconditional failure intensity would be ten, if there were only 5 components working in the specified time interval then the denominator for the conditional failure intensity would be 5.
4. The conditional repair intensity, $\mu(t)$, is defined as the probability that a component is repaired per unit time at $t$ given that it failed at time $t$ and was working at time zero.

This parameter only considers those components that are actually failed at time $t$ and not all of the components within the set. All of these measures allow for failures or repairs prior to time $t$.

Integral equations utilising the failure and repair density functions, can be solved to yield the unconditional failure intensity, $w(t)$, and the unconditional repair intensity, $v(t)$. The derivation of these measures can be seen in Andrews and Moss ${ }^{[A M 093]}$, yielding the intensity formulas:

$$
\begin{align*}
& w(t)=f(t)+\int_{0}^{t} f(t-u) v(u) d u \\
& v(t)=\int_{0}^{t} g(t-u) w(u) d u
\end{align*}
$$

Once the unconditional failure and repair intensities have been established then the component unavailability, $Q(t)$, can be derived from the values of $v(t)$ and $w(t)$, to give:

$$
Q(t)=\int_{0}^{1}[w(u)-v(u)] d u
$$

For example, given a component with failure and repair intensities of $f(t)=\lambda e^{-\lambda t}$ and $g(t)=\mu e^{-\mu t}$, then the unavailability of the component can be calculated using Laplace transforms and gives:

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

### 2.5.3 Defining The Fault Tree Structure - Structure Functions

A binary system must by definition exist in either a working state or a failed state. Similarly, every component that is part of the system must also exist in one of these two states, so the state of the system will be a function of the state of its components. Once the component states are determined this will fix the system state. For a component $i$, let $x_{i}$ be the indicator variable so,

$$
\begin{array}{ll}
x_{i}=1 & \text { component fails } \\
x_{i}=0 & \text { component works }
\end{array}
$$

As the system can be defined in terms of the state of its components, the system state can be expressed as a function $\phi(\underline{x})$ such that:

$$
\begin{aligned}
& \phi(\underline{x})=1, \text { if the system is failed } \\
& \phi(\underline{x})=0, \text { if the system is working }
\end{aligned}
$$

where $\underline{x}=\left(x_{l}, \ldots, x_{n}\right)$ is the set of all $n$ component indicator variables and $\phi(\underline{x})$ is known as the structure function.

The structure function $\phi(\underline{x})$ for the top event of a fault tree can be defined in terms of an indicator function $P_{i}$ for the minimal cut sets $C_{i}$, where $i=1, \ldots, n$ is given by the formula:

$$
\phi(\underline{x})=1-\prod_{i=1}^{n}\left(1-P_{i}\right)
$$

The structure function for a tree with minimal cut sets $\mathrm{AC}, \mathrm{AD}, \mathrm{BC}, \mathrm{BD}$ is:

$$
\phi(\underline{x})=1-\left(1-x_{A} \cdot x_{C}\right)\left(1-x_{A} \cdot x_{D}\right)\left(1-x_{B} \cdot x_{C}\right)\left(1-x_{B} \cdot x_{D}\right)
$$

The probability of the top event equals the expectation of the structure function and in the case when each $C_{i}$ is statistically independent then the expectation of the structure function is the structure function of expectations:

$$
P(T O P)=E[\phi(\underline{x})]=\phi[E(\underline{x})]
$$

When there is not independence, $E[\phi(\underline{x})] \neq \phi[E(\underline{x})]$, and in this case the structure function must first be fully expanded and reduced using the rule shown below (i.e. to remove powers of the indicator variables) prior to taking the expectation:

$$
x_{A} \cdot x_{A}=x_{A}
$$

To illustrate finding the top event probability, then consider for example,

$$
\begin{aligned}
& \phi(\underline{x})=1-\left(1-x_{1} x_{2}\right)\left(1-x_{2} x_{3}\right)\left(1-x_{3} x_{1}\right) \\
& \phi(\underline{x})=x_{1} x_{2}+x_{2} x_{3}+x_{3} x_{1}-2 x_{1} x_{2} x_{3}
\end{aligned}
$$

If the probability of each component occurring is equal to 0.1 , i.e. $E\left[x_{i}\right]=0.1$ for $i=1,2$, 3 , then by taking the expectation of the structure function, following its full expansion and reduction, the top event probability of occurrence would be given as:

$$
\begin{aligned}
& P(T O P)=E[\phi(x)]=E\left[x_{1} x_{2}\right]+E\left[x_{2} x_{3}\right]+E\left[x_{3} x_{1}\right]-E\left[2 x_{1} x_{2} x_{3}\right] \\
& P(T O P)=E\left[x_{1}\right] \cdot E\left[x_{2}\right]+E\left[x_{2}\right] \cdot E\left[x_{3}\right]+E\left[x_{3}\right] \cdot E\left[x_{1}\right]-2 \cdot E\left[x_{1}\right] \cdot E\left[x_{2}\right] \cdot E\left[x_{3}\right] \\
& P(T O P)=0.01+0.01+0.01-2(0.001)=0.028
\end{aligned}
$$

An alternative method that is more efficient is Shannon's theorem. It is a decomposition technique that can be applied to the top event probability function.

### 2.5.4 Shannon's Theorem

Shannon's theorem ${ }^{[\text {[ch89] }}$ can be explained by considering a Boolean function $\phi(\underline{x})$ where $\underline{x}=\left(x_{1}, x_{2}, \ldots, x_{n}\right)$. This function can be written as:

$$
\phi(\underline{x})=x_{i} \phi\left(1_{i}, \underline{x}\right)+\bar{x}_{i} . \phi\left(0_{i}, \underline{x}\right)
$$

where,

$$
\begin{aligned}
& \bar{x}_{i}=1-x_{i} \\
& \phi\left(0_{i}, \underline{x}\right)=\phi\left(x_{1}, \ldots \ldots, x_{i-1}, 0, x_{i+1}, \ldots \ldots, x_{n}\right) \\
& \phi\left(1_{i}, \underline{x}\right)=\phi\left(x_{1}, \ldots ., x_{i-1}, 1, x_{i+1}, \ldots \ldots, x_{n}\right)
\end{aligned}
$$

$\phi\left(1_{i}, \underline{x}\right)$ and $\phi\left(0_{i}, \underline{x}\right)$ are called the residues of $\phi(\underline{x})$ with respect to $x_{i}$.

Therefore taking the expectation of equation 2.7 and letting $P_{i}=E\left(x_{i}\right)$, the probability function $Q(P)$ can be rewritten in the form:

$$
Q(\underline{P})=P_{i} \cdot E\left\{\phi\left(1_{i}, \underline{P}\right)\right\}+\left(1-P_{i}\right) \cdot E\left\{\phi\left(0_{i}, P\right)\right\}
$$

The structure function must first be expanded or pivoted with respect to the most repeated event using Shannon's expansion. This is continued until no powers of indicator variables exist in the residues. The expectation can then be taken to give the top event probability.

To illustrate consider again the structure function for the top event, TOP, given by equation 2.6.

$$
\phi(\underline{x})=1-\left(1-x_{A} \cdot x_{C}\right)\left(1-x_{A} \cdot x_{D}\right)\left(1-x_{B} \cdot x_{C}\right)\left(1-x_{B} \cdot x_{D}\right)
$$

The expectation can not be taken without first expanding because there are repeated events. Therefore, pivoting about event $x_{A}$ gives:

$$
\phi(\underline{x})=x_{A}\left[1-\left(1-x_{C}\right)\left(1-x_{D}\right)\left(1-x_{B} \cdot x_{C}\right)\left(1-x_{B} \cdot x_{D}\right)\right]+\left(1-x_{A}\right)\left[1-\left(1-x_{B} \cdot x_{D}\right)\left(1-x_{B} \cdot x_{C}\right)\right]
$$

Again there is still one repeated event, $x_{B}$, so pivot now about $x_{B}$ :

$$
\begin{aligned}
\phi(\underline{x})=x_{B}\left\{x_{A}\left[1-\left(1-x_{C}\right)\left(1-x_{D}\right)\left(1-x_{C}\right)\left(1-x_{D}\right)\right]+\left(1-x_{A}\right)\right. & {\left.\left[1-\left(1-x_{D}\right)\left(1-x_{C}\right)\right]\right\} } \\
+ & \left(1-x_{B}\right)\left\{x_{A}\left[1-\left(1-x_{C}\right)\left(1-x_{D}\right)\right]\right\}
\end{aligned}
$$

Note that the expression $\left(1-x_{C}\right)\left(1-x_{D}\right)\left(1-x_{C}\right)\left(1-x_{D}\right)$ reduces to just (1-x $)\left(1-x_{D}\right)$ using the Boolean Laws of algebra, therefore, the final expression is:

$$
\phi(\underline{x})=x_{B}\left\{x_{A}\left[1-\left(1-x_{C}\right)\left(1-x_{D}\right)\right]+\left(1-x_{A}\right)\left[1-\left(1-x_{D}\right)\left(1-x_{C}\right)\right]\right\}+\left(1-x_{B}\right)\left\{x_{A}\left[1-\left(1-x_{C}\right)\left(1-x_{D}\right)\right]\right\}
$$

There are now no repeated events so taking the expectation gives:

$$
\begin{aligned}
& P(T O P)=E[\phi(\underline{x})]=E\left[x_{B}\right]\left\{E\left[x_{A}\right]\left[1-\left(1-E\left[x_{C}\right]\right)\left(1-E\left[x_{D}\right]\right)\right]+\right. \\
&\left.\left(1-E\left[x_{A}\right]\right)\left[1-\left(1-E\left[x_{D}\right]\right)\left(1-E\left[x_{C}\right]\right)\right]\right\} \\
&+\left(1-E\left[x_{B}\right]\right)\left\{E\left[x_{A}\right]\left[1-\left(1-E\left[x_{C}\right]\right)\left(1-E\left[x_{D}\right]\right)\right]\right\}
\end{aligned}
$$

$$
\begin{aligned}
\therefore P(T O P)=P_{B}\left\{P_{A}\left[1-\left(1-P_{C}\right)\left(1-P_{D}\right)\right]+\left(1-P_{A}\right)\right. & {\left.\left[1-\left(1-P_{D}\right)\left(1-P_{C}\right)\right]\right\} } \\
& +\left(1-P_{B}\right)\left\{P_{A}\left[1-\left(1-P_{C}\right)\left(1-P_{D}\right)\right]\right\}
\end{aligned}
$$

Finally, the process just remains for the probabilities to be substituted in for each event, and the top event probability can be calculated.

### 2.5.5 General Approach to Calculate The Top Event Probability

The most commonly used method to calculate the top event probability is based on the minimal cut sets previously determined in the qualitative analysis. Using the minimal cut sets is a good general approach, which will yield the correct result for trees with repeated events providing the assumption that the basic events are independent is appropriate.

If a fault tree has $n$ minimal cut sets, $C_{\mathrm{i}}, i=1, \ldots, n$ then the top event exists if at least one minimal cut set exist, i.e.

$$
\begin{align*}
T O P & =C_{1}+C_{2}+\cdots+C_{n} \\
& =\bigcup_{i=1}^{n} C_{i}
\end{align*}
$$

The probability of $T O P$ is given by:

$$
P(T O P)=P\left(\bigcup_{i=1}^{n} C_{i}\right)=P\left(C_{1}+C_{2}+C_{3}+\cdots+C_{n}\right)
$$

Expanding this gives:

$$
P(T O P)=\sum_{i=1}^{n} P\left(C_{i}\right)-\sum_{i=2}^{n} \sum_{j=1}^{i-1} P\left(C_{i} \cap C_{j}\right)+\cdots(-1)^{n-1} P\left(C_{1} \cap C_{2} \cap \cdots \cap C_{n}\right)
$$

This expansion is known as the inclusion-exclusion expansion. Using this generates the exact probability of the top event occurrence.

To illustrate the usage of this expansion method, consider the fault tree whose minimal cut sets are:

$$
\begin{aligned}
& \{A C\} \\
& \{A D\} \\
& \{B C\} \\
& \{B D\}
\end{aligned}
$$

So the top event can be expressed as:

$$
T O P=A \cdot C+A \cdot D+B \cdot C+B \cdot D
$$

The unavailability or probability of the top event, $Q_{S r s}(t)$, is:

$$
Q_{S Y S}(t)=P(T O P)=P(A \cdot C+A \cdot D+B \cdot C+B \cdot D)
$$

Expanding this using the complete inclusion-exclusion expansion method gives:

```
\(Q_{S r S}(t)=[\underbrace{P(A C)+P(A D)+P(B C)+P(B D)}_{\text {firstrerm }}]\)
\(-[\underbrace{P(A C D)+P(B C A)+P(B D A C)+P(B C A D)+P(B D A)+P(B D C)}_{\text {sec onderm }}]\)
\(+[\underbrace{P(A C D B)+P(A C B D)+P(A C B D)+P(A D B C)}_{\text {thirdterm }}]\)
\(-[\underbrace{P(A C B D)}_{\text {fourtherm }}]\)
```

If all basic events have probabilities of occurrence of 0.1 then the top event probability would be:

$$
\begin{aligned}
Q(t) & =[0.01+0.01+0.01+0.01] \\
& -[0.001+0.001+0.0001+0.0001+0.001+0.001] \\
& +[0.0001+0.0001+0.0001+0.0001] \\
& -[0.0001] \\
& =[0.04]-[0.0042]+[0.0004]-[0.0001] \\
& =0.0361
\end{aligned}
$$

The full evaluation of each term in the inclusion-exclusion expansion can be extremely calculation intensive for trees with a large number of minimal cut sets. This is frequently beyond the capability of fast digital computers to fully evaluate. Therefore, approximations that produce acceptably accurate results are required. Looking at the formula, the inclusion-exclusion method adds successive odd numbered terms and subtracts successive even numbered terms. Each term in the expansion is successively less significant than the term preceding it. Truncating the expansion on an odd numbered term will provide an upper bound for the top event probability and truncating the expansion on an even numbered term will provide a lower bound.

### 2.5.6 Approximation Methods

There are three main approximation methods to calculate the top event probability, which are: the rare event approximation (upper bound); the lower bound approximation and the minimal cut set upper bound. Considering these three methods in turn, each are discussed in the following sections.

### 2.5.6.1 Upper Bounds For System Unavailability - Rare Event Approximation

As an increasing number of terms in the inclusion-exclusion formula are evaluated the probability produced gets closer and closer to the exact probability of the top event. Therefore, by just considering the first term in the inclusion-exclusion expansion, $\sum_{i=1}^{n} P\left(C_{i}\right)$, this is an upper bound for the top event probability, expressed as:

$$
\underbrace{Q_{S Y s}(t)}_{\text {exact }} \leq \underbrace{\sum_{i=1}^{n} P\left(C_{i}\right)}_{\text {upper bound }}
$$

The upper bound is known as the Rare Event Approximation, since it is accurate if the component failure events are rare (occur with small probability).

### 2.5.6.2 Lower Bounds For System Unavailability

A lower bound to the top event probability is the combination of the first two terms of the inclusion-exclusion expansion, to give:

$$
\underbrace{\sum_{i=1}^{n} P\left(C_{i}\right)-\sum_{i=2}^{n} \sum_{j=1}^{i-1} P\left(C_{i} \cap C_{j}\right)}_{\text {lower bound }} \leq \underbrace{Q_{S Y S}(t)}_{\text {exact }}
$$

### 2.5.6.3 Minimal Cut Set Upper Bound

A more accurate upper bound than the rare event approximation is the minimal cut set upper bound, which is formulated as follows:

$$
\begin{aligned}
P(\text { system failure }) & =P(\text { at least one minimal cut set occurs }) \\
& =1-P(\text { no minimal cut set occurs })
\end{aligned}
$$

Since

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

(equality being when no event appears in more than one minimal cut set). Thus,

$$
\begin{aligned}
& P(\text { system failure }) \leq 1-\prod_{i=1}^{n} P(\text { minimal cut set } i \text { does not occur }) \\
& \Rightarrow Q_{S Y S}(t) \leq 1-\prod_{i=1}^{n}\left[1-P\left(C_{i}\right)\right]
\end{aligned}
$$

It can be shown that:

$$
Q_{S Y S}(t) \leq 1-\prod_{i=1}^{n}\left[1-P\left(C_{i}\right)\right] \leq \sum_{i=1}^{n} P\left(C_{i}\right)
$$

$\Rightarrow$ Exact $\leq$ Minimal Cut Set Upper Bound $\leq$ Rare Event Approximation

The problem of using any of these approximations occurs when the basic event failures are not rare.

### 2.5.7 Unconditional Failure Intensity

One important, and often required, quantification measure is the unconditional failure intensity. The system unconditional failure intensity, $w_{S Y S}(t)$, is defined as the probability that the top event occurs at $t$ per unit time. This parameter is important for the
quantitative analysis of the fault tree as the 'expected number of top event occurrences' can be determined by integrating $w_{S Y S}(t)$ with respect to $t$. Therefore $w_{S Y S}(t) d t$ is the probability that the top event occurs in the time interval $[t, t+d t)$. For the top event to occur between $t$ and $t+d t$ all the minimal cut sets must not exist at $t$, then one or more occur during $t$ to $t+d t$. More than one minimal cut set can occur in a small time element $d t$ since component failure events can be common to more than one minimal cut set. Therefore:

$$
w_{S Y S}(t)=P\left[A \bigcup_{i=1}^{n} \theta_{i}\right]
$$

where:
A is the event that all minimal cut sets do not exist at time $t, \mathrm{~A}=\bigcap_{i=1}^{n} u_{i}$ $u_{i}$ denotes the $i^{\text {th }}$ minimal cut set does not exist at $t$.
$\bigcup_{i=1}^{n} \theta_{i}$ is the event that one or more minimal cut sets occur in time $t$ to $t+d t$

Since $P(A)=1-P(\bar{A})$, then $w_{S Y S}(t)$ can be written as:

$$
P\left[A \bigcup_{i=1}^{n} \theta_{i}\right]=P\left[\bigcup_{i=1}^{n} \theta_{i}\right]-P\left[\bar{A} \bigcup_{i=1}^{n} \theta_{i}\right]
$$

where,
$\bar{A}$ means at least one minimal cut set exists at $t$.

Hence,

$$
w_{S Y S}(t)=P\left[\bigcup_{i=1}^{n} \theta_{i}\right]-P\left[\bar{A} \bigcup_{i=1}^{n} \theta_{i}\right]
$$

The first term on the right hand side of the above expression is the contribution from the occurrence of at least one minimal cut set, and the second term is the correction contribution provided by minimal cut sets occurring while other minimal cut sets already exist (i.e. the system has already failed). These two terms can be denoted by $w_{S Y S}^{(1)}(t)$ and $w_{S Y S}^{(2)}(t)$ respectively, so:

$$
w_{S Y S}(t) d t=w_{S Y S}^{(1)}(t) d t-w_{S Y S}^{(2)}(t) d t
$$

Each of these terms can be expanded to generate the mathematical formulas required to calculate the desired unconditional failure intensities ${ }^{[A M 093]}$.

To calculate the expected number of system failures in time $t, W(0, t)$, means taking the integral of the system unconditional failure intensity, over the specified time interval $t$.

$$
W(0, t)=\int_{0}^{t} w_{S S S}(t) d t
$$

### 2.6 Importance Measures

### 2.6.1 Introduction

Components or basic events of a system have varying degrees of influence upon system failure. This is represented by their differing positions within the fault tree structure. An importance measure of each component dictates their contribution to system failure in an importance analysis. An importance analysis is a sensitivity analysis that identifies weak areas of the system. For each component its importance signifies the role that it plays in either causing or contributing to the occurrence of the top event. The numerical
value of the importance measure allows each basic event to be ranked according to the extent of its contribution to the occurrence of the top event.

There are two categories of importance measures:

1. Deterministic.
2. Probabilistic.

The second category (probabilistic measures) can be subdivided further into two classes, namely:

1. Those which are concerned with the top event probability.
2. Those that are concerned with the expected number of top event occurrences.

### 2.6.2 Deterministic Measures

Deterministic measures assess the importance of a component to the system operation without considering the component's probability of failure. One such measure is the Structural Measure of Importance (SMI), which is defined for a component $i$, as:

$$
S M I_{i}=\frac{\text { numberof critical states for component, } i}{\text { totalnumberof states for the }(n-1) \text { remaining components }}
$$

A critical state for component $i$, is a state for the remaining ( $n-1$ ) components such that a failure of component $i$ causes the system to go from a working state to a failed state. This deterministic measure is discussed in more detail in chapter 8.

### 2.6.3 Probabilistic Measures For Assessing System Unavailability

There are several probabilistic measures for assessing system unavailability or top event probability, four of which are addressed in this section.

### 2.6.3.1 Birnbaum's Measure of Importance

This measure of importance is also known as the criticality function. The criticality function for a component $i$ is denoted by $G_{i}(\underline{q})$. The criticality function can be defined as the probability that the system is in a critical system state for component $i$. Therefore, it is the probability that the system fails only if component $i$ fails.

There are two expressions for the criticality function as defined below:

1. $G_{i}(\underline{q})=Q\left(1_{i}, \underline{q}\right)-Q\left(0_{i}, \underline{q}\right)$ where:
$Q\left(1_{i}, \underline{q}\right)=Q\left(q_{1}, \cdots, q_{i-1}, 1, q_{i+1}, \cdots, q_{n}\right) ;$ the probability the system fails when $q_{i}=1$, and $Q\left(0_{i}, \underline{q}\right)=Q\left(q_{1}, \cdots, q_{i-1}, 0, q_{i+1}, \cdots, q_{n}\right) ;$ the probability the system fails when $q_{i}=0$.

To illustrate this, if the probability that a system fails is represented by the formula:

$$
Q(\underline{q})=q_{A} q_{B}+q_{A} q_{D}-q_{A} q_{B} q_{D}
$$

The criticality of component A, would be:

$$
\begin{aligned}
& Q\left(1_{A}, \underline{q}\right)=q_{B}+q_{D}-q_{B} q_{D} \\
& Q\left(0_{A}, \underline{q}\right)=0 \\
& \Rightarrow G_{A}(\underline{q})=q_{B}+q_{D}-q_{B} q_{D}
\end{aligned}
$$

The numerical value for the criticality is calculated by substituting in the unavailability data for each component.
2. $G_{i}(\underline{q})=\frac{\partial Q(\underline{q})}{\partial q_{i}}$

This is calculating the criticality function as a partial derivative. Using the same example as above given by equation 2.21 :

$$
G_{A}(\underline{q})=\frac{\partial Q(\underline{q})}{\partial q_{A}}=q_{B}+q_{D}-q_{B} q_{D}
$$

### 2.6.3.2 Criticality Measure of Importance

The criticality measure of importance, $C I$, is defined as the probability that the system is in a critical state for component $i$ and $i$ has failed given that the system has failed:

$$
C I_{i}=\frac{G_{i}(\underline{q}) q_{i}(t)}{Q_{s s s}(t)}
$$

To illustrate, using the same $Q_{s y s}(t)$ given in equation 2.21:

$$
C I_{A}=\frac{\left(q_{B}+q_{D}-q_{B} q_{D}\right) q_{A}}{q_{A} q_{B}+q_{A} q_{D}-q_{A} q_{B} q_{D}}
$$

### 2.6.3.3 Fussell-Vesely Measure of Importance

The Fussell-Vesely Measure of Importance, $F V I$, is defined as the probability of the union of the minimal cut sets containing $i$ given that the system has failed:

$$
F V I_{i}=\frac{P\left(\bigcup_{k i \in k} C_{k}\right)}{Q_{S r s}(t)}
$$

For example, using the equation 2.21 for the expression for the probability of top event failure (i.e. minimal cut sets $A B, A D$ ), the Fussell-Vesely Measure of Importance for component $B$ is:

$$
F V I_{B}=\frac{\left(q_{A} q_{B}\right)}{q_{A} q_{B}+q_{A} q_{D}-q_{A} q_{B} q_{D}}
$$

### 2.6.3.4 Fussell-Vesely Measure of Minimal Cut Set Importance

This measure ranks the minimal cut sets in order of the contribution to the top event, unlike those above which ranks each individual component. The importance measure, $F M C$, is defined as the probability of occurrence of minimal cut set $i$ given that the system has failed:

$$
F M C_{i}=\frac{P\left(C_{i}\right)}{Q_{S S S}(t)}
$$

### 2.6.4 Probabilistic Measures For System Unreliability

Three measures are highlighted in this section. These measures are used for systems where the interval reliability is to be assessed and the sequence in which components fail matters. Note, $W(0, t)$ is the expected number of system failures.

### 2.6.4.1 Definition of Initiator and Enabler Events

In some cases the order in which the events occur in any minimal cut set is important to the occurrence of the fault tree top event. For example, when analysing a safety protection system two possible outcomes could be, if the protection system fails and then a hazardous event occurs this could result in a dangerous system failure. However, if the failures occur in another sequence, i.e. the hazardous event occurs before the safety protection system fails, this will cause a system shutdown. This type of situation is modelled by considering the failures as either initiating or enabling events. Initiating and enabling events can be defined as follows:

## Initiating Events:

Perturb system variables and place a demand on control/protection systems to respond.

## Enabling Events:

Are inactive control/protection systems, which permit initiating events to cause the top event.

Importance measures can be found for each of these types of events.
2.6.4.2 Barlow-Proschan Measure of Initiator Importance

This Barlow-Proschan Measure, BPI, is the conditional probability that initiating event $i$ caused the failure, given that the system fails in the interval $[0, t)$.

$$
B P I_{i}=\int_{0}^{1} \frac{\left[Q\left(1_{i}, \underline{q}(t)\right)-Q\left(0_{i}, \underline{q}(t)\right)\right] w_{i}(t) d t}{W(0, t)}
$$

Hence, for the Barlow-Proschan Initiator Measure to be evaluated it requires the evaluation of the criticality function, $G_{i}(\underline{q}(t))$, represented in equation 2.26 as the difference between the two probability expressions, component unconditional failure intensity, $w_{i}(t)$, and the expected number of top event occurrences, $W(0, t)$.

### 2.6.4.3 Sequential Contributory Measure of Enabler Importance

This measure, $S Q I$, is the probability that an enabling event $i$ permits an initiating event, $j$, to cause system failure over the interval $[0, t)$. The failure of enabler $i$ is therefore only a factor when enabler $i$ and initiator $j$ both occur in the same minimal cut set, $C$.


### 2.6.4.4 Barlow-Proschan Measure of Minimal Cut Set Importance

This measure is the probability that a minimal cut set causes the system failure in the interval $[0, t)$ given that the system has failed:

$$
B P M I_{i}=\frac{\sum_{j \in i} \int_{0}^{t}\left[1-Q\left(0_{j}, 1^{i-(j)}, \underline{q}(t)\right)\right] \prod_{\substack{k \neq j \\ k \in i}} q_{k}(t) w_{j}(t) d t}{W(0, t)}
$$

( $j$ is initiating each event in the minimal cut set $\{i\}$.)

### 2.7 Advantages and Disadvantages of The Fault Tree Methodology

This method to qualitatively and quantitatively analyse the fault tree has its advantages and disadvantages. One benefit of the method is its systematic approach in taking the specific failure mode and in turn highlighting each of the intermediate causes. The resulting fault tree diagram yields a complete description of the system failure. Using the technique enables system performance measures to be obtained, i.e. the probability of the failure mode occurring and the frequency of failure in a specified time range. The analysis procedure highlights the causes of failure and also the contribution of each component to the system failure. The process of obtaining these system performance measures facilitates the identification of ways to modify the system if it does not meet the necessary requirements, be it safety or otherwise.

The disadvantages of the technique however are found within the analysis procedure itself. The technique can be inefficient in calculating the minimal cut sets of the fault tree, especially for large fault tree structures with many repeated events. In these circumstances culling methods may help resolve the problem. This culling method means that only cut sets of a specified order or probability are examined. Hence a complete list of the minimal cut sets may not be feasible for all trees. These problems are also extended to the quantification phase whereby the minimal cut sets are required for calculation purposes. Again, approximation methods are often employed to calculate the system performance measures.

To overcome these deficiencies one technique that has been established is the Binary Decision Diagram approach. This has been found to be more efficient in terms of finding the minimal cut sets of the fault tree and also permits the exact evaluation of the system performance measures.

### 2.8 Summary

To summarise the fault tree technique the following can be stated:

- The fault tree yields a descriptive diagrammatic representation of the failure modes of a system.
- A qualitative and quantitative analysis can be carried out.
- The qualitative analysis produces the combinations of components which when fail cause the top event to occur (i.e. failure); these are known as the minimal cut sets.
- The quantitative analysis procedure produces a number of system performance measures, i.e. top event probability and frequency of occurrence, as well as the importance measures of each component within the system.
- Finding the minimal cut sets of the fault tree using the techniques already mentioned can be inefficient, especially for large fault tree structures with hundreds and even thousands of components.
- The quantification procedure uses the minimal cut sets, and hence often approximations must be used to establish the system performance measures.
- A more efficient and exact analysis technique needs to be established to utilise the fault trees potential. One technique is the Binary Decision Diagram approach.


## 3 Binary Decision Diagrams

### 3.1 Introduction

Fault tree analysis is a commonly used means of assessing the system reliability performance in terms of its component's reliability characteristics. As indicated in the summary of chapter 2 , this analysis technique is not without its limitations especially when dealing with large fault tree structures. Analysis of the top event probability usually requires the use of approximations, as the exact method makes significant use of computer resources. Many of the techniques used to analyse the structure, i.e. the topdown and bottom-up approaches have been well researched. Any substantial reduction in computer utilisation is expected to be achieved only from a completely new method. Potentially the most successful recent development in the fault tree methodology is the Binary Decision Diagram (BDD) approach. The Binary Decision Diagram approach has been shown to improve both the efficiency of determining the minimal cut sets of the fault tree and also the accuracy of the calculation procedure used to determine the top event parameters.

To utilise the Binary Decision Diagram approach the fault tree structure is first converted to the BDD format. Implementing the conversion of the tree is relatively straightforward and efficient but requires the basic events of the tree to be placed in an ordering. The ordering scheme chosen is critical to the size of the BDD produced, and hence the efficiency of this technique.

### 3.2 Background

Over the past decade an alternative technique to Kinetic Tree Theory ${ }^{[V e s 70]}$, known as the Binary Decision Diagram method has been developed to analyse the fault tree.

Akers ${ }^{[\text {Ake78] }}$ was the first researcher to derive the basic methodology for the BDD approach. This work, involving digital functions, found there was a "description-gap" between the means of representing these functions. Previously a variety of languages were at hand to provide a functional description of the system or components, however, their conciseness led to difficulties in applying any type of analysis procedure. These languages included Boolean equations and truth tables. Hence, Aker's alternative was to represent the function in terms of a diagram, which defines how to determine the output value of a function by examining the values of the inputs. These diagrams were called Binary Decision Diagrams.

Since the introduction of the BDD analysis technique numerous researchers have further enhanced the method ${ }^{[\text {Ra496, SA196, Bry86, SA296, SA396] }}$, its efficiency and capabilities, to its current status. This method has proved to be more accurate and efficient than the conventional approaches. In calculating the system or top event parameters it does not need to first evaluate all the minimal cut sets, nor does it require the use of approximations, the exact calculations can be performed.

### 3.3 BDD Architecture

A BDD is a directed acyclic graph, as shown in figure 3.1. This means that all paths through the diagram go in one direction and there are no loops permitted, i.e. all the paths are directed in one straight route from the top of the diagram to the bottom. All paths through the BDD start at the root vertex and terminate in one of two states, either a 1 state, which corresponds to a system failure (or top event occurrence), or a 0 state, which corresponds to a system success (or top event non-occurrence). A BDD is composed of terminal and non-terminal vertices, which are connected by branches. Non-terminal vertices correspond to the basic events of the fault tree.


Figure 3.1: A Binary Decision Diagram

All the left branches leaving a vertex are the 1 branches (corresponding to the basic event failure or vertex occurrence) and all the right branches the 0 branches (corresponding to the basic event functioning or vertex non-occurrence). Only the vertices that lie on a 1 branch on the way to a terminal 1 vertex are included in the path. All the paths terminating in a 1 state give the cut sets of the fault tree. For example, the cut sets, that is the combinations of events that cause the top event failure, for the BDD shown in figure 3.1 are $\{X 1, X 2, X 3\}$ and $\{X 1, X 4\}$.

### 3.4 Constructing The BDD

### 3.4.1 Brief Introduction

To utilise the BDD approach the fault tree must first be converted to the appropriate diagram. There are two main methods used to convert the fault tree structure into a BDD, one involves using the top event logic function (discussed in section 3.4.2), the other using an If-Then-Else Method (discussed in section 3.4.3).

### 3.4.2 Constructing The BDD Using The Top Event Logic Function

The logic function representing the top event of the fault tree can be used to generate the Binary Decision Diagram. The process involves substituting in the value of 1 (component fails) and then 0 (component working) for each vertex (or node) in the BDD. To illustrate this, consider the fault tree in figure 3.2.


Figure 3.2: Example Fault Tree

The top event logic function (which can be given in any form) is represented by:

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

The minimal cut sets of the fault tree are:

$$
\{A, B\},\{B, C, D\}
$$

To start the conversion process the basic events need to be placed in an order. Using a top-down, left-right approach i.e. starting from the top gate and working downwards, at each level the tree is scanned from left to right, the ordering for the basic events would be:

$$
A<B<C<D
$$

This ordering means that first variable $A$ is considered in the conversion process, then variable $B$ and so on until all the variables in the list have been taken in turn. Now the assignment of values, corresponding to the component state of the Boolean variables selected from the list, can be made. The first variable is $A$ and assigning the value of 1 to this variable within the logic function produces the Boolean equation:

$$
T\left(1_{A}, \underline{x}\right)=B+B . C . D
$$

This forms the function at the end of the left-hand branch of the vertex. This function will be further evaluated with the assignment of values for the next Boolean variable in the ordered list. Assigning the value of 0 to the variable $A$ in the logic function produces the right-hand branch of the vertex, with the resulting Boolean equation:

$$
T\left(0_{A}, \underline{x}\right)=B . C . D
$$



Figure 3.3: BDD For Structure Function $T=A \cdot B+B \cdot C . D$

Continuing this substitution technique for each of the Boolean variables in turn (as determined by the ordering permutation) will produce the BDD as shown in figure 3.3. The Boolean equations involved in the conversion process are represented on the diagram.

The calculations for each of the intermediate logic functions have been removed from figure 3.3 to create the simplified version of the BDD shown in figure 3.4.


Figure 3.4: Simplified Version of The BDD Shown in Figure 3.3

The cut sets of this BDD can be obtained by tracing the paths from the root vertex along the 1 branches to a terminal 1 state, shown in figure 3.4 with arrows. Hence the cut sets are:

$$
\{A B\},\{B C D\} .
$$

In this instance, the diagram is in its minimal form and hence the cut sets are minimal. Sometimes the BDD produced is not minimal and to produce the minimal cut sets a minimisation procedure needs to be applied. This is described in section 3.4.4.

This conversion method requires a considerable amount of simplification by applying Boolean reduction laws after each function evaluation at the end of a branch. This is particularly evident when there are a number of repeated events within the logic function. One method to alleviate these problems is to use the If-Then-Else approach.

### 3.4.3 Constructing The BDD Using The If-Then-Else Method

Rauzy ${ }^{[\text {Rau93] }}$ first used the If-Then-Else (ite) structure method when converting the fault tree to the BDD. One important feature of the BDD method is that the ite structure is derived from Shannon's formula (chapter 2). To reiterate, if $f(\underline{x})$ is the Boolean structure function for the top event of the fault tree then by pivoting about any variable, $X_{i}$ say, the Shannon formula may be written as:

$$
f(x)=X_{i} . f 1+\overline{X_{i}} . f 2
$$

where $f 1$ and $f 2$ are Boolean functions with $X_{i}=1$ and $X_{i}=0$ respectively, and are of order one less than $f$. The corresponding ite structure is ite $\left(X_{i}, f 1, f 2\right)$, where $X_{i}$ is the Boolean variable and $f 1$ and $f 2$ are logic functions. This means if $X_{i}$ fails then consider function $f 1$ else consider $f 2$. Therefore, in the BDD $f 1$ represents the structure lying below the 1 branch of $X_{i}$, and $f 2$ represents the structure lying below the 0 branch. This ite structure is illustrated in diagrammatic form in figure 3.5.


Figure 3.5: ite representation of ite $\left(X_{i}, f 1, f 2\right)$

Following the ordering of the basic events the ite procedure can be applied to construct the BDD. Note, that in the following procedures <op> corresponds to the Boolean operation of the logic gates of the fault tree, so if the gate is an AND gate <op> will be the dot or product symbol (.), and if the gate is an OR gate <op> will be the sum symbol (+).

The procedure to apply the ite approach is outlined below:

1. Assign each basic event, $X_{i}$ in the fault tree the ite structure ite $\left(X_{i}, 1,0\right)$, ( $X_{i}$ can either fail - 1 branch, or work - 0 branch).
2. Convert the fault tree structure to one where each gate has only two inputs.
3. Consider each gate in the fault tree structure in a bottom-up approach.
4. If gate inputs are $J$ and $H$, where:

$$
J=\operatorname{ite}(x, F 1, F 2) \text { and } H=\operatorname{ite}(y, G 1, G 2)
$$

then apply the following rules:

- If $x<y$ :
$J<\mathrm{op}>H=\operatorname{ite}(x, F 1<\mathrm{op}>H, F 2<\mathrm{op}>H)$
- If $x=y: \quad J<\mathrm{op}>H=\operatorname{ite}(x, F 1<\mathrm{op}>G 1, F 2<\mathrm{op}>G 2)$
- Consider Vote gates individually.

If $F$ is a $k / n$ vote gate with inputs $F_{1}, \ldots, F_{n}$, i.e. $F=a t-$ least $\left(k, F_{1}, \ldots, F_{n}\right)$, then $F$ is written implicitly as ( $F_{1} \cap$ at-least $(k$ $\left.\left.1, F_{2}, \ldots, F_{n}\right)\right) \cup$ at-least $\left(k, F_{2}, \ldots, F_{n}\right)$.

These are used in conjunction with the following identities to produce the simplest ite structure for each gate:
$1<\mathrm{op}>H=1$, if $<\mathrm{op}>$ is an OR gate
$0<\mathrm{op}>H=H$, if $<\mathrm{op}>$ is an OR gate
$1<\mathrm{op}>H=H$, if $<\mathrm{op}>$ is an AND gate
$0<\mathrm{op}>H=0$, if $<\mathrm{op}>$ is an AND gate

An illustration of the procedure employed for a vote gate is if $k=3$ and $n=4$, then $F$ would have inputs $F 1, F 2, F 3$ and $F 4$. Thus, $F$ can be expressed as:

$$
\left(F_{1} \cap a t-l e a s t\left(2, F_{2}, F_{3}, F_{4}\right)\right) \cup a t-\text { least }\left(3, F_{2}, F_{3}, F_{4}\right) .
$$

Simplifying $F_{1} \cap\left(2, F_{2}, F_{3}, F_{4}\right)$ becomes:

$$
F_{1} \cap\left[\left(F_{2} \cap a t-\text { least }\left(1, F_{3}, F_{4}\right)\right) \cup a t-\text { least }\left(2, F_{3}, F_{4}\right)\right]
$$

Hence, the whole expression is:

$$
\left(F_{1} \cap\left[\left(F_{2} \cap a t-l e a s t\left(1, F_{3}, F_{4}\right)\right) \cup a t-l e a s t\left(2, F_{3}, F_{4}\right)\right]\right) \cup a t-l e a s t\left(3, F_{2}, F_{3}, F_{4}\right)
$$

Therefore, $F$ can be written as: $F_{1} \cdot F_{2} \cdot F_{3}+F_{1} \cdot F_{2} \cdot F_{4}+F_{1} \cdot F_{3} \cdot F_{4}+F_{2} \cdot F_{3} \cdot F_{4}$, and the ite formulation can now be applied.

To illustrate the ite procedure to obtain the top event ite structure, consider the simple fault tree in figure 3.6.


Figure 3.6: Simple Fault Tree Structure

Using an ordering of $A<B<C$ for the fault tree shown in figure 3.6, the ite structure for the top event is obtained as follows:

Each event is given an ite structure:

$$
\begin{aligned}
& A=\operatorname{ite}(A, 1,0) \\
& B=\operatorname{ite}(B, 1,0) \\
& C=\operatorname{ite}(C, 1,0)
\end{aligned}
$$

Working from the bottom of the tree to the top and applying the procedure as explained the result is:

$$
\begin{aligned}
\text { Gate1 } & =B . C \\
& =\operatorname{ite}(B, 1,0) . \operatorname{ite}(C, 1,0) \\
& =\operatorname{ite}(B, 1 . \operatorname{ite}(C, 1,0), 0 . \operatorname{ite}(C, 1,0)) \\
& =\operatorname{ite}(B, \operatorname{ite}(C, 1,0), 0) \\
\text { TOP } & =A+\text { Gatel }
\end{aligned}
$$

$$
\begin{aligned}
& =\operatorname{ite}(A, 1,0)+\operatorname{ite}(B, \operatorname{ite}(C, 1,0), 0) \\
& =\operatorname{ite}(A, 1+\operatorname{ite}(B, \operatorname{ite}(\mathrm{C}, 1,0), 0), 0+\operatorname{ite}(B, \operatorname{ite}(\mathrm{C}, 1,0), 0)) \\
& =\operatorname{ite}(A, 1, \operatorname{ite}(B, \operatorname{ite}(C, 1,0), 0))
\end{aligned}
$$

To construct the BDD the ite structure is successively broken down into its left and right branches. The root vertex is the node $A$, and the 1 branch of $A$ will terminate in a 1 terminal vertex, and the ite structure ite $(B$, ite $(C, 1,0), 0)$ will lie below the 0 branch of $A$ (shown in figure 3.7a).


Figure 3.7a: Creating The BDD by Expanding The ite Structure

Next node $B$ is developed. The ite structure ite $(C, 1,0)$ will lie below the 1 branch of node $B$, and the 0 branch will terminate in a 0 end vertex. The ite structure representative for node B is drawn in figure 3.7 b .


Figure 3.7b: Ite Structure Representing Expansion For Node $B$

The resulting BDD is shown in figure 3.7c.


Figure 3.7c: Resulting BDD From The Fault Tree Shown in Figure 3.6

The paths are then obtained as before, these being $\{A\},\{B, C\}$.

### 3.4.4 Reducing The Size of The BDD

Converting the fault tree into a BDD produces a problem if the minimal cut sets are required and the resulting BDD is not minimal. In such a situation a conversion process is needed to produce a minimal form of the BDD that encodes only the minimal cut sets.

One simple approach that may help the situation where nodes are repeated is the method of Friedman and Supowit ${ }^{[\mathrm{FS} 490]}$. It is stated that a tree representing a function of $n$ variables can have a maximum of $2^{n+1}-1$ nodes, however it was recognised that a BDD can be reduced in size by two 'collapsing' operations. These operations are:

1. If the two sons of a node $A$ are equivalent, then delete node $A$ and direct all of its incoming edges to its left son. (N.B. A 'son' of a node is simply the node attached to either its 1 or 0 branch).

To illustrate this operation consider the BDD shown on the left in figure 3.8, where node $B$ has two equivalent sons (node $C$ ). Deleting node $B$ and directing all of its incoming edges (the 1 branch from node $A$ ) to its left son (node $C$ ), produces the new BDD structure as shown by the BDD on the right in figure 3.8.


Figure 3.8: BDD to Illustrate Operation 1 of Friedman and Supowit's Approach
2. If nodes $A$ and $B$ are equivalent, then delete node $B$ and direct all of its incoming edges to $A$.

This reducing operation is shown in figure 3.9. The BDD on the left has two nodes, node 1 and 2 represented by the vertex label $C$, which are equivalent. To simplify the diagram node 2 can be removed and its incoming edge from the node labelled vertex $B$ can be directed to node 1 , as shown in the diagram on the right hand side of figure 3.9.


Figure 3.9: BDD to Illustrate Operation 2 of Friedman and Supowit's Approach

Sometimes the logic function is more complicated and despite applying the 'collapsing' algorithms above the BDD can still produce non-minimal cut sets, hence, another minimisation process is required. The fault tree in figure 3.10 , with resulting unminimised BDD (figure 3.11) will be used to illustrate this alternative minimisation algorithm.


Figure 3.10: Fault Tree to Illustrate Minimisation Algorithm of BDD

Constructing the BDD using the ite method with an ordering $X 1<X 2<X 3<X 4$ for the variables, the results for the intermediate gates are:

$$
\begin{aligned}
G 2 \quad & =X 3 \cdot X 4 \\
& =\operatorname{ite}(X 3,1,0) \cdot \operatorname{ite}(X 4,1,0) \\
& =\operatorname{ite}(X 3,1 \cdot i \operatorname{ite}(X 4,1,0), 0 . \operatorname{ite}(X 4,1,0)) \\
& =\operatorname{ite}(X 3, \operatorname{ite}(X 4,1,0), 0) \\
G 1 \quad & =X 1 \cdot X 2 . X 3 \\
& =\operatorname{ite}(X 1,1,0) \cdot \operatorname{ite}(X 2,1,0) \cdot \operatorname{ite}(X 3,1,0) \\
& =\operatorname{ite}(X 1,1 . \operatorname{ite}(X 2,1,0), 0 . i t e(X 2,1,0)) \cdot \operatorname{ite}(X 3,1,0) \\
& =\operatorname{ite}(X 1, \operatorname{ite}(X 2,1,0), 0) \cdot \operatorname{ite}(X 3,1,0) \\
& =\operatorname{ite}(X 1, \operatorname{ite}(X 2,1,0) \cdot \operatorname{ite}(X 3,1,0), 0 . \operatorname{ite}(X 3,1,0)) \\
& =\operatorname{ite}(X 1, \operatorname{ite}(X 2,1 . \operatorname{ite}(X 3,1,0), 0 . \operatorname{ite}(X 3,1,0)), 0) \\
& =\operatorname{ite}(X 1, \operatorname{ite}(X 2, \operatorname{ite}(X 3,1,0), 0), 0)
\end{aligned}
$$

Now both intermediate gates have been expanded the top gate can be evaluated.

$$
\begin{aligned}
& T O P= G 1+G 2 \\
&= \operatorname{ite}(X 1, \operatorname{ite}(X 2, \operatorname{ite}(X 3,1,0), 0), 0)+\operatorname{ite}(X 3, \operatorname{ite}(X 4,1,0), 0) \\
&= \operatorname{ite}(X 1, \operatorname{ite}(X 2, \operatorname{ite}(X 3,1,0), 0)+\operatorname{ite}(X 3, \operatorname{ite}(X 4,1,0), 0), \\
&0+\operatorname{ite}(X 3, \operatorname{ite}(X 4,1,0), 0)) \\
&= \operatorname{ite}(X 1, \operatorname{ite}(X 2, \operatorname{ite}(X 3,1,0)+\operatorname{ite}(X 3, \operatorname{ite}(X 4,1,0), 0), \\
&\quad 0+\operatorname{ite}(X 3, \operatorname{ite}(X 4,1,0), 0)), \operatorname{ite}(X 3, \operatorname{ite}(X 4,1,0), 0)) \\
&= \operatorname{ite}(X 1, \operatorname{ite}(X 2, \operatorname{ite}(X 3,1+\operatorname{ite}(X 4,1,0), 0+0), \operatorname{ite}(X 3, \operatorname{ite}(X 4,1,0), 0)), \\
&=\operatorname{ite}(X 3, \operatorname{ite}(X 4,1,0), 0)) \\
& \operatorname{ite}(X 2, \operatorname{ite}(X 3,1,0), \operatorname{ite}(X 3, \operatorname{ite}(X 4,1,0), 0)), \operatorname{ite}(X 3, \operatorname{ite}(X 4,1,0),
\end{aligned}
$$

$$
0) \text { ) }
$$

This TOP event ite structure is representative of the BDD shown in figure 3.11. The 'collapsing' rules defined above have been applied yet the cut sets that are produced are:

$$
\{X 1, X 2, X 3\},\{X 1, X 3, X 4\},\{X 3, X 4\}
$$

These are not minimal cut sets (the second set is non-minimal). Boolean Reduction Laws could be applied to produce the minimal cut sets, however this process increases computation time and memory requirements which destroys the aim of the BDD technique.


Figure 3.11: BDD Resulting From Fault Tree in Figure 3.10

Rauzy ${ }^{[\text {Rau93] }}$ has created an algorithm that generates a BDD defining exactly the minimal cut sets of the fault tree. The algorithm uses the ite operation procedure directly. To explain, consider a general node $x$ in a BDD. The algorithm states that:

If the output of a node is represented by the function $F$ where $F=$ ite $(x, G, H)$, let $\delta$ be a minimal solution of $G$ which is not a minimal solution of $H$, then the intersection of $\delta$ and $x$ will be a minimal solution of $F$, given by $F_{\min }=\{\delta\} \cap x$.

More clearly, tracing the paths to the terminal 1 vertices from the BDD vertices representing the functions $G$ and $H$ will give the sets of solutions that can cause these two events. Removing redundant combinations from these sets will yield the minimal solutions. The solutions of $G$ augmented with $x$ will cause $F$ as will the solution of $H$. To ensure that this combined set is a minimal solution of $F$ any minimal solutions of $H$ are removed from the minimal solutions of $G$.

The algorithm is completed as follows:

The set of all minimal solutions of $F$, sol $l_{\min }(F)$, will include the minimal solutions of $H\left(\operatorname{sol}_{\min }(H)\right)$ so:

$$
s o l_{\min } F=[\{\delta\} \cap x] \cup\left[s o l_{\min }(H)\right]
$$

That is, the minimal solutions of $F$ include both those minimal solutions of $G$ which are not contained within $H$, ( $\delta$, and the minimal solutions of $H$.

To illustrate with an example, consider the BDD shown in figure 3.12. The solutions of $G$ are found to be $\{A\},\{B C\},\{D\}$. The solution of $H$ is $\{\mathrm{D}\}$. To find $\delta$ the solutions that also occur in $H$ must be removed i.e. D. If retained on the 1 branch it will yield a solution of $x D$, which is not minimal. Hence, the minimal solutions of $G$ which were not minimal solutions of $H$, are $A$ and B.C. Therefore, including the minimal solutions of $H$, namely $D$, can complete the minimal solutions of $F$. Thus, the minimal solutions of $F$ are:

$$
\operatorname{sol}_{\min } F=[x . A, x . B . C, D]
$$



Figure 3.12: Illustrating Minimal Solution of BDDs

Rauzy has defined a 'without' operator, without( $G_{\text {min }}, H$ ) which removes from $G_{\text {min }}$ all the paths included in a path of $H$ i.e. establishes $\delta$. To demonstrate this algorithm it is applied to the BDD shown in figure 3.11. Considering the nodes in a top-down order, for the root vertex node $F 1$, the 0 branch leads to the node $F 4$, which corresponds to $H$. The solution of $H$ is $X 3 X 4$. The 1 branch of $F 1$ leads to node $F 2$ which corresponds to $G$. The solutions of $G$ are $X 2 X 3, X 3 X 4$. Hence solution $X 3 X 4$ is also included in the path from the 0 branch of F1. To establish the minimal solutions of $F 1$ we need to formulate $\mathrm{G}_{\min }$, i.e. the minimal solutions of $F 2$. In this case, the solutions of $F 2$ are minimal, therefore we remove from $F 2$ all the paths that include the solution $X 3 X 4$. This is performed by removing $F 4$ (vertex labelled $X 3$ ) from the 0 branch of $F 2$ and replacing it with a terminal 0 vertex. This application of minimising is carried out for all nodes within the BDD. In this example, this results in no further alterations. The BDD shown in figure 3.13 is therefore in its minimised form. The minimal cut sets can now be read directly from the BDD.


Figure 3.13: Minimised BDD of Figure 3.11

Sinnamon ${ }^{[S i n 96]}$ noticed that Rauzy failed to recognise two further results which would increase the efficiency of the minimisation procedure, these being:

1. without $(F, F)=\{ \}$

This statement is saying to remove from $F$ all the paths included in a path of $F$. As these are the same, this results in no paths (the empty set).
2. $\operatorname{ite}(x, F, F)=F$

The node with label $x$ can be removed by the collapsing rule.

These two factors are included in the computer code, produced by Sinnamon ${ }^{[\text {Sin96] }}$ to generate the BDD from the fault tree diagram.

### 3.5 Top Event Quantification

### 3.5.1 System Failure Probability

The BDD can be used to calculate the Top Event Probability and numerous other measures of quantification, including importance measures ${ }^{[S i n 96]}$. Since the BDD method converts the fault tree diagram into a format that encodes Shannon's decomposition it allows the exact failure probability to be determined in a very efficient calculation procedure.

Given a structure function $F(\underline{x})$ for the top event, the probability is obtained by taking the expectation:

$$
\begin{aligned}
E[F(\underline{x})] & =\sum_{x_{i}} x_{i} P\left(F(\underline{x})=x_{i}\right)=1 \cdot P(F(\underline{x})=1)+0 \cdot P(F(\underline{x})=0) \\
& =P(F(\underline{x})=1) \\
& =Q_{S Y S}
\end{aligned}
$$

Shannon's representation of the probability of the top event is given by:

$$
F(\underline{x})=x_{i} \cdot F_{1}\left(x_{1}, x_{2}, \cdots, x_{i-1}, 1, x_{x+1}, \cdots, x_{n}\right)+\bar{x}_{i} \cdot F_{2}\left(x_{1}, x_{2}, \cdots, x_{i-1}, 0, x_{x+1}, \cdots, x_{n}\right)
$$

Taking the expectation of this representation gives:

$$
\begin{align*}
& E[F(\underline{x})]=Q_{S Y S}(\underline{q}) \\
& E\left[F_{1}(\underline{x})\right]=Q_{S Y}\left(q_{1}, \cdots, q_{i-1}, 1, q_{i+1}, \cdots, q_{n}\right) \\
& E\left[F_{2}(\underline{x})\right]=Q_{S Y S}\left(q_{1}, \cdots, q_{i-1}, 0, q_{i+1}, \cdots, q\right) \\
& E[F(\underline{x})]=q_{i} E\left[F_{1}(\underline{x})\right]+\left(1-q_{i}\right) \cdot E\left[F_{2}(\underline{x})\right]
\end{align*}
$$

where $q_{i}=E\left[x_{i}\right]$, the probability that event $i$ occurred. Hence, the top event probability can be found by evaluating equation 3.3. The BDD encodes a Shannon form of the structure function.

Each path through the BDD to a terminal 1 vertex is mutually exclusive or disjoint, therefore to obtain the probability of occurrence of the top event the sum of the probabilities of the disjoint paths through the BDD is calculated ${ }^{[S A 196]}$. The probability of the disjoint paths is taken from the unminimised BDD structure because the minimised BDD changes the logic function to encode only the minimal cut sets. The disjoint paths are all the paths starting at the root vertex that terminate at a 1 vertex, including both the nodes on the 1 and 0 branches. The disjoint paths of the BDD shown in figure 3.11 are:

$$
\{X 1, X 2, X 3\},\{\overline{X 1}, X 3, X 4\},\{X 1, \overline{X 2}, X 3, X 4\}
$$

Before the calculation can begin, the basic events need to be assigned probabilities. Calculation of the top event now involves summing the probabilities of these disjoint contributions.

$$
\begin{aligned}
Q_{S Y S} & =P(X 1 \cdot X 2 \cdot X 3+\overline{X 1} \cdot X 3 \cdot X 4+X 1 \cdot \overline{X 2} \cdot X 3 \cdot X 4) \\
& =q_{X 1} \cdot q_{X 2} \cdot q_{X 3}+\left(1-q_{X 1}\right) \cdot q_{X 3} \cdot q_{X 4}+q_{X 1} \cdot\left(1-q_{X 2}\right) \cdot q_{X 3} \cdot q_{X 4}
\end{aligned}
$$

where $q_{i}$ is the unavailability of component $i$.

### 3.5.2 System Failure Intensity

One other key quantification measure is the top event unconditional failure intensity $w_{S Y S}(t)$, that is the probability per unit time that the system fails at time $t$. This can be determined from:

$$
w_{S Y S}(t)=\sum_{i=1}^{n} G_{i}(\underline{q}) \cdot w_{i}(t)
$$

where $G_{i}(\underline{q})$ is the criticality function (see section 2.6.3.1) and $w_{i}$ is the failure intensity for component $i$. To reiterate, the criticality function is defined as the probability that the system is in a critical state with respect to component $i$ and that the failure of component $i$ will then cause the system to go from the working to the failed state. Therefore,

$$
G_{i}(\underline{q})=Q\left(1_{i}, \underline{q}\right)-Q\left(0_{i}, \underline{q}\right)
$$

The two terms on the right hand side of equation 3.5 can be calculated using the BDD approach, which only requires one pass of the BDD structure for each component, unlike the fault tree approach which requires two passes. The process requires the evaluation of the probability of the path arriving at each node of the BDD and also the probability of the path going from this node to a terminal 1 vertex on each of its two output branches.

The formulae used when applying the BDD approach are:

$$
\begin{align*}
& Q\left(1_{i}, \underline{q}\right)=\sum_{i=1}^{n}\left(p r_{x i}(\underline{q}) \cdot p o_{x i}^{1}(\underline{q})\right)+Z(\underline{q}) \\
& Q\left(0_{i}, \underline{q}\right)=\sum_{i=1}^{n}\left(p r_{x i}(\underline{q}) \cdot p o_{x i}^{0}(\underline{q})\right)+Z(\underline{q})
\end{align*}
$$

where:
$p r_{x i}(\underline{q})$ is the probability of the path section from the root node to node $x_{i}$.
$p o_{x i}^{1}(\underline{q})$ is the probability of the path section from the 1 branch of node $x_{i}$ to a terminal 1 node (excluding probability of $x_{i}$ ).
$p o_{x i}^{0}(\underline{q})$ is the probability of the path section from the 0 branch of node $x_{i}$ to a terminal 1 node (excluding probability of $x_{i}$ ).
$Z(\underline{q})$ is the probability of the paths from the root node to the terminal 1 node not passing through the node for variable $x_{i}$.

Therefore, in terms of these equations the criticality function can be rewritten:

$$
G_{i}(\underline{q})=\sum_{i=1}^{n} p r_{x i}(\underline{q}) \cdot\left[p o_{x i}^{1}(\underline{q})-p o_{x i}^{0}(\underline{q})\right]
$$

Using this formula (equation 3.8) for the criticality function provides an efficient means of calculating the unconditional failure intensity of the system. To illustrate, consider the BDD shown in figure 3.14, the tables 3.1-3.4 demonstrate the mathematical calculation procedure.


Figure 3.14: Simple BDD For Calculating Quantification Measures

The first stage of the procedure requires knowledge of the connections between the nodes, i.e. the ite table for the BDD, as shown in table 3.1.

| Node Label | Variable | 1 branch pointer | 0 branch pointer |
| :---: | :---: | :---: | :---: |
| F1 | X1 | F2 | 0 |
| F2 | X2 | F3 | F4 |
| F3 | X3 | 1 | F5 |
| F4 | X3 | 1 | 0 |
| F5 | X4 | 1 | 0 |

Table 3.1: Ite Table For BDD Shown in Figure 3.11.

Next the calculation of each of the terms in equation 3.8 needs to be performed. The first probability, the probability of the path from the root vertex to node $x_{i}, p r_{x i}(\underline{q})$ is evaluated by summing the probabilities along the relevant path. The calculation steps can be seen in table 3.2 where the path taken is summarised in the "Explanation" column.

| Node Label |  |  |
| :---: | :---: | :---: |
| Probability of |  |  |
| path, $p r_{x i}(\underline{q})$ | Explanation <br> F1 | 1 |

Table 3.2: Calculation Procedure For $p r_{x i}(\underline{q})$.

Table 3.3 summarises the steps required to calculate the probability of $p o_{x i}^{1}(q)$, the path from the selected node along the 1 branch to any terminal 1 vertex, excluding the probability of the selected node. For example, from node F1 of figure 3.14 , the paths to a terminal 1 vertex along the 1 branch of the selected node are:

- Path 1 - along the 1 branch of F1 to node F2, along the 1 branch of F2 leading to node F3 and then to the terminal 1 vertex.
- Path 2 - along the 1 branch of F1 to node F2, along the 0 branch of F2 leading to node F4 and then to the terminal vertex at the end of the 1 branch of node F4.
- Path 3 - along the 1 branch of F1 to node F2, along the 1 branch of F2 leading to node F3, then along the 0 branch of F3 leading to node F5 and then to the terminal vertex at the end of the 1 branch of node F5.

In table 3.3, the column referring to "INT" is the intermediate calculation of the path and includes the probability of the selected component. The paths for each of the other nodes are explained in brief in the final column of table 3.3.

| Node <br> Label | INT | $p o_{x i}^{1}(q)$ | Explanation of Route |
| :---: | :---: | :---: | :---: |
| F1 | $\begin{aligned} & q_{1} q_{2} q_{3}+q_{1} q_{2}\left(1-q_{3}\right) q_{4} \\ & +q_{1}\left(1-q_{2}\right) q_{3} \end{aligned}$ | $\begin{aligned} & q_{2} q_{3}+q_{2}\left(1-q_{3}\right) q_{4} \\ & +\left(1-q_{2}\right) q_{3} \end{aligned}$ | $\begin{gathered} 3 \text { routes - } 1 \text { branch of } \\ \text { F1, } 1 \text { of F2, } 1 \text { F3, or } 1 \\ \text { F1, } 1 \mathrm{~F} 2,0 \mathrm{~F} 3,1 \mathrm{~F} 5 \text {, or } \\ 1 \mathrm{~F} 1,0 \mathrm{~F} 2,1 \mathrm{~F} 4 \end{gathered}$ |
| F2 | $q_{2} q_{3}+q_{2}\left(1-q_{3}\right) q_{4}$ | $q_{3}+\left(1-q_{3}\right) q_{4}$ | $\begin{gathered} 1 \text { F2, } 1 \text { F3 or } 1 \text { F2, } 0 \\ \text { F3, } 1 \text { F5 } \end{gathered}$ |
| F3 | $q_{3}$ | 1 | 1branch to terminal 1 node |
| F4 | $q_{3}$ | 1 | 1 branch |
| F5 | $q_{4}$ | 1 | 1 branch |

Table 3.3: Calculation of $p o_{x i}^{1}(\underline{q})$

The calculation of the probability of going along the 0 branch to a terminal 1 node is explained in table 3.4. Again "INT" refers to the whole path, including the probability of the selected node. Taking node F2 as an example, the paths to the terminal 1 vertex starting by going along the 0 branch of node F2 are:

- Path 1 - along the 0 branch of F2 leading to node F4 and then the 1 branch of F4 leading to terminal 1 vertex.


Table 3.4: Calculation of $p o_{x i}^{0}(\underline{q})$ For BDD Shown in Figure 3.11

Hence, ultimately $G_{i}(\underline{q})$ can be calculated for each variable, by adding together contributions from any nodes of the same variable, thus:

$$
\begin{aligned}
& G_{X 1}(\underline{q})=q_{2} q_{3}+q_{2}\left(1-q_{3}\right) q_{4}+\left(1-q_{2}\right) q_{3} \\
& G_{X 2}(\underline{q})=q_{1}\left(1-q_{3}\right) q_{4} \\
& G_{X 3}(\underline{q})=q_{1} q_{2}\left(1-q_{4}\right)+q_{1}\left(1-q_{2}\right) \\
& G_{X 4}(\underline{q})=q_{1} q_{2}\left(1-q_{3}\right)
\end{aligned}
$$

Having obtained $G_{i}(\underline{q})$ this is substituted into equation 3.8 to give $w_{S Y S}(t)$.

Being able to calculate the system unconditional failure intensity efficiently means that the calculation of the expected number of top event occurrences can also be efficiently calculated if required. The calculation of the expected number of failures of the top event means taking the integral of equation 3.8 (as detailed in section 2.5.7).

These quantification measures and others can be calculated using a computer program which has been produced by Sinnamon ${ }^{[\operatorname{Sin} 96]}$. The formulas for the remaining quantification measures can be found in Sinnamon ${ }^{[\text {Sin96] }}$.

### 3.6 Applications of BDDs To Event Trees

### 3.6.1 Overview of Event Trees

The event tree analysis technique is used to identify the consequences following the occurrence of a hazardous initiating event. Separate subsystems are examined in turn and their responses, either functioning and responding to the initiating event or failing and not responding, are identified, creating branch points of a tree structure. The end branch points of the tree correspond to different consequences resulting from the starting initiating event. Fault trees are developed to identify causes of each of the subsystem failures. One minus the failure probability (generated by evaluating the fault tree structure) gives the likelihood of passing along the success (subsystem working) branch of the event tree.

The problems of using the event tree approach occur when there are dependencies within the tree, that is, when component failures appear in more than one of the fault trees representing the causes of the branches. The underlying problem concerns the ability of the fault tree quantification process in dealing with working and failed components, i.e. quantification of non-coherent fault trees. These quantification problems are summarised in the following section.

### 3.6.2 Traditional Solution to Dependencies Within The Event Tree

When dependencies arise within the system, the fault trees incorporating the dependencies need to be considered as inputs to an AND gate, whose output now determines the causes of a higher level 'complex' event. For example, consider the event tree drawn in figure 3.15 which has dependencies in system 1 and system 2. The fault trees representing the causes of subsystem failure are given in figure 3.16. The dependencies between the systems are the repetition of the basic events labelled $A$ and $D$.


Figure 3.15: Event Tree With Dependencies Between System 1 and System 2

The fault tree for system 1 failure is drawn in figure 3.16a and that for system 2 in figure 3.16b.


Figure 3.16a: Fault Tree For System 1
Failure


Figure 3.16b: Fault Tree For System 2
Failure

There are four outcomes of the event tree, and considering just one, the outcome of S1 works and S2 fails, the combined fault tree is represented in figure 3.16c.


Figure 3.16c: Fault Tree Representing Outcome S1 Works and S2 Fails

Boolean reduction laws that are applied to this new fault tree result in combinations of successes and failures as the NOT logic is developed through the tree, hence the fault tree is non-coherent. Analysis of non-coherent fault trees relies heavily on approximation methods and is both inaccurate and inefficient. The quantification process involves evaluating the inclusion-exclusion expansion of the probability of the top event (equation 2.12). To enable this quantification, the prime implicants are reduced to their coherent approximations by assuming working states for components in the expression are set to TRUE, on the assumption that $\operatorname{Prob}$ (component works) $\approx 1$. Hence, coherent approximation methods can be applied. These approximation methods involve truncating the expansion after only a few terms, and to be valid for non-coherent trees many terms may need to be evaluated to gain the required accuracy.

### 3.6.3 How The BDD Has Enhanced The Event Tree Technique

Andrews and Dunnett ${ }^{[A D V 99]}$ have shown that the application of the BDD technique in quantifying the dependencies within the event tree is more efficient and accurate than using the conventional fault tree approach. The BDD methodology encodes Shannons formula therefore quantification can be evaluated from the diagram directly. The dual BDD or BDD representing system functioning is easily constructed by changing all the terminal 1 vertices to 0 and vice versa. To use the BDD technique for event tree analysis purposes involves combining the BDDs for each system (for example using the ite formulation) to produce the outcomes with dependencies. To quantify the BDD involves calculating the sum of the disjoint paths (successes and failures) leading to a terminal 1 vertex. This is the same as applying the full inclusion-exclusion formula. Thus, this method will produce exact quantification measures to use for the branches of the event tree, and also is more efficient than the fault tree analysis technique currently in use. An algorithm has been written to quantify a general event tree structure with dependencies using this BDD method.

### 3.7 Variable Ordering

### 3.7.1 The Problem

In constructing the BDD the ordering of the basic events is crucial to the size of the resulting diagram. Using an inefficient ordering scheme will produce a non-minimal BDD structure. Different ordering schemes will produce BDDs of different sizes, the smaller the BDD the more optimal the diagram. To illustrate this fact, consider the simple fault tree shown in figure 3.17. The tree has four basic events, where X 2 is repeated.


Figure 3.17: A Simple Fault Tree

If the basic event ordering permutation of $X 1<X 2<X 3<X 4$ is taken, the resulting BDD is shown in figure 3.18. This structure consists of only four nodes, it is a minimal structure and hence produces only minimal cut sets.


Figure 3.18: Resulting BDD From Ordering $X 1<X 2<X 3<X 4$.

However, if the alternative ordering permutation of $X 4<X 3<X 2<X 1$ is taken the resulting BDD consists of seven nodes, it is non-minimal and yields non-minimal cut sets
(shown in figure 3.19). From this result it can be shown that for larger fault tree structures the resulting BDD would be much larger, and in the worst case of using a poor ordering permutation, the diagram may be unsolvable.


Figure 3.19: Resulting BDD From Ordering $X 4<X 3<X 2<X 1$.

The objective would be to produce an ordering scheme that achieves the 'best' BDD for any fault tree. Numerous studies have investigated the effects of variable ordering schemes on the BDD size and it has been shown that there is no universal scheme that will always guarantee the 'best' BDD formation, and the most appropriate scheme must be selected depending on the characteristics of the fault tree. This area of work is described in detail in chapter 4.

### 3.8 Summary

- The best development in analysing the fault tree is to use the Binary Decision Diagram approach.
- The process requires the fault tree to be converted into an alternative structure known as a Binary Decision Diagram.
- The method produces the minimal cut sets of the fault tree more efficiently and can produce exact quantification measures.
- The advantages of the BDD technique can be applied to event tree analysis, as fault tree techniques are used in establishing the failure probabilities within the event tree. The benefits gained in the fault tree analysis are even more significant when applied to non-coherent systems. Incorporating the BDD technique improves the accuracy and efficiency of the event tree analysis.
- A potential difficulty of the technique stems from the conversion process of the fault tree to the BDD representation. In the conversion process the basic events (or components) of the fault tree need to be ordered, and it is this ordering that can cause complications.


## 4 Variable Ordering Heuristics

### 4.1 Introduction

It is possible that using an ordering heuristic for a given fault tree will cause the BDD structure to explode exponentially. The problem with this is an inefficient analysis or in the worst case it is not possible to develop the BDD. If a non-minimum BDD results from the conversion process it must undergo a minimising procedure to obtain the minimal cut sets, which can cause an undesirable increase in computer time. Also using a larger than necessary BDD is inefficient when calculating the top event probability and frequency of occurrence. Of course the degree of inefficiency depends on how much larger than the minimum the BDD is. It is therefore beneficial and desirable to achieve an ordering which is optimal in terms of the resulting size of the BDD, or in the extreme an ordering which will produce a BDD. Finding a solution to the ordering dilemma will enable the resulting BDD in the fault tree to BDD conversion process to be minimal or at least near minimal, and ultimately promote an efficient analysis.

Although the Binary Decision Diagram approach is a relatively new technique researchers have identified the variable ordering problem, and a number of ordering heuristics have been suggested. The techniques available and applicable to fault tree analysis are explained below and their characteristics and merits discussed.

### 4.2 The Most Commonly Used Heuristic

The most common heuristic for ordering is produced by listing the variables in a topdown, left-right basis from the original fault tree structure. To illustrate, consider the fault tree shown in figure 4.1 represented as an alternating AND/OR gate sequence. The fault
tree structure has four distinct levels separated by gates. The process of ordering begins at the top most level, and continues downwards until the last level is reached. At each level, the ordering commences in a left to right path and basic events that are encountered (as shown by the circles) are added to the ordering list. If a basic event is encountered which is already placed higher up the tree and has therefore been incorporated in the list, then it is ignored.


Figure 4.1: Fault Tree to Illustrate Ordering Heuristics

Applying this method, the ordering would progress in levels as shown:

- Level 1: Only one basic event encountered therefore first variable in ordering list.

$$
\{A\}
$$

- Level 2: Two basic events located, providing next two inputs to ordering list. $\{B, C\}$
- Level 3: Going from left to right along this level, produces three new inputs. Note variable $A$ has already occurred higher up the tree, and is therefore ignored.

$$
\{H, E, D\}
$$

- Level 4: Finally the variables in the last level are ordered, producing the final variable ordering for the fault tree shown in figure 4.1 as:

$$
A<B<C<H<E<D<K<G<F
$$

Therefore, when producing the BDD , first variable $A$ would be considered in the conversion process, then $B$ and so on until all the variables had been used.

Despite being the most popular heuristic it's performance is variable depending on the fault tree structure being converted. Often the BDDs produced using this simple ordering are non-minimal. This is ever prevalent as the complexity and size of the fault tree increases.


Figure 4.2: Same Fault Tree as Shown in Figure 4.1 With Variable Inputs Altered

One flaw with the methodology of the top-down, left-right heuristic is that this ordering alone does not produce a unique BDD structure for this example fault tree or any other tree. The positioning or drawing of the inputs of each gate causes the problem. If the events of the tree are drawn in another order as shown in figure 4.2, then the resulting ordering list generated by applying the same heuristic to the same tree is different and will ultimately yield a different BDD.

In figure 4.2 the inputs of Gate1 are re-ordered as $C$ first then event $B$ then Gate 3, and similarly the Top Event inputs are re-ordered producing the following alternative ordering using the same technique for the same top event:

$$
A<C<B<E<D<H<F<G<K
$$

### 4.3 Modifications to The Top-down, Left-right Approach

### 4.3.1 Using Repeated Events

Sinnamon and Andrews ${ }^{[S A 196]}$ present an alternative ordering to the common top-down approach. The scheme focuses on those basic events that are repeated in the fault tree structure. It is the repeated events that cause the problem of non-minimal cut sets and ultimately non-minimal BDDs. By considering these events first their aim was to simplify the resulting BDD structure and thus make it more optimal.

The alternative ordering scheme still uses the top-down algorithm, however, as each level is scanned from left to right, if any basic event which is repeated in the tree is encountered then it is placed ahead of any other variable along the same level. If there is more than one repeated basic event in any level, then the ordering is determined by the number of occurrences of each event (with most repeated being placed first). If there is a tie, it is broken by just applying the left-right strategy to each as they are positioned in the level.

Applying this heuristic to the tree in figure 4.1, the only alteration to the resulting ordering concerns the repeated variable $G$. In level $4, G$ is a repeated variable whereas $K$ and $F$ are not, hence, $G$ is placed in the ordering before $K$ and $F$. Therefore the ordering list using this heuristic is:

$$
A<B<C<H<E<D<G<K<F
$$

| Tree | Number of Gates | No. of <br> Basic <br> Events | Number of Repeated Events | Number of Cut Sets <br> Using Bottom Up <br> Approach | Number of Minimal cut sets from BDD |
| :---: | :---: | :---: | :---: | :---: | :---: |
| 1 | 6 | 8 | 1 | 5 | 2 |
| 2 | 3 | 7 | 1 | 3 | 1 |
| 3 | 3 | 7 | 2 | 5 | 3 |
| 4 | 4 | 8 | 2 | 4 | 2 |
| 5 | 3 | 4 | 1 | 2 | 2 |
| 6 | 5 | 6 | 1 | 4 | 3 |
| 7 | 4 | 8 | 2 | 4 | 3 |
| 8 | 6 | 10 | 2 | 14 | 6 |
| 9 | 5 | 9 | 1 | 19 | 15 |
| 10 | 4 | 5 | 1 | 4 | 2 |
| 11 | 3 | 4 | 1 | 4 | 2 |
| 12 | 6 | 9 | 2 | 6 | 3 |
| 13 | 3 | 4 | 1 | 2 | 1 |
| 14 | 4 | 7 | 1 | 3 | 3 |
| 15 | 9 | 17 | 3 | 17 | 8 |

Table 4.1: Example Fault Trees Used in Study ${ }^{[S A 196]}$

Within their study, the ordering heuristic was tested on fifteen example fault trees. A summary of the trial fault tree characteristics is given in table 4.1. Relatively small trees
were chosen to emphasise that even for small tree structures improvements could be made.

Only trees 14 and 15 of the examples used did not produce absolute minimal BDDs, however, this new repeated event based ordering did produce the more minimal BDD when compared to the common top-down approach of ordering. The overall results from Sinnamon and Andrews research suggested that the new ordering giving priority to repeated events appeared to produce more optimal BDDs compared to the top-down ordering alone.

This heuristic, like the top down approach, still has the problem that the events of each gate can be listed in any order and hence a unique BDD is not found with this technique. Also this heuristic can only attempt to improve the size of the BDD when repeated events are present within the tree structure. There can be no change if there are no repeated events (however then the fault tree analysis is trivial) and the heuristic only has more potential to change the ordering and resulting size of the BDD as the number of repeated events increases.

### 4.3.2 Using Subtrees (Depth-First and Priority Depth-First Heuristics)

Two additional heuristics that still use the top-down principle as their basis involve looking at subtrees of the whole tree. Branching from the top event are a number of inputs and branching from each of these inputs are smaller tree structures called subtrees. The depth-first and priority depth-first techniques involve looking at each subtree in turn from left to right along level 1 , and for each subtree applying the top-down, left-right principle.

The first heuristic to be examined is the Depth-First approach, this involves breaking the whole tree structure into smaller trees and looking at the optimal ordering of these subtrees. From figure 4.1, there are three subtrees, headed by the names $A$, Gate 1 and Gate 2. As $A$ is a basic event there is no more variables to add, so the next subtree is
examined. Looking at the inputs of Gate 1, by applying the top-down, left-right principle the order of variables would be $\{B, C, H, K, G\}$. The next step is to examine the final subtree, Gate 2. Applying the same principle as for Gate 1, and ignoring any basic events that may have already been ordered, the ordering would be: $\{E, D, F\}$. Therefore, the final ordering to use in the conversion process would be:

$$
A<B<C<H<K<G<E<D<F
$$

A slight alternative to the ordering is to give priority to the repeated events. In this example fault tree this only affects the variable $G$, to yield the different ordering of:

$$
A<B<C<H<G<K<E<D<F
$$

Research highlighting the benefits of the depth-first subtree approach was produced by Rauzy ${ }^{[\text {Rau93] }}$. Considering the computation time necessary for constructing a BDD and finding the minimal cut sets Rauzy compared two different ordering heuristics. The first ordering (which shall be referred to as ordering 1) was simply the variable ordering as written in the formula governing the tree, for example, if the formula $F$ was:

$$
F=(a \vee c) \wedge(b \vee c)
$$

Then the ordering assumed would be $a<c<b$. The alternative ordering (ordering 2) was the depth-first approach.

The results were compared on a set of thirteen industrial fault trees, nine were provided by a French Aviation company (Dassault Aviation) and the remaining four from Professor Y. Dutuit. Table 4.2 summaries some of the characteristics of the test set of fault trees. The number of events in the tree, number of gates in the tree and the number of cut sets of the tree are all given.

| Benchmark Source | $\begin{aligned} & \text { Tree } \\ & \text { Label } \end{aligned}$ | Number of Events | Number of Gates (connectives) | Number of Cut Sets (size) |
| :---: | :---: | :---: | :---: | :---: |
| Dassault | 1 | 103 | 145 | 248 |
| Dassault | 2 | 122 | 82 | 204 |
| Dassault | 3 | 51 | 30 | 81 |
| Dassault | 4 | 53 | 30 | 83 |
| Dassault | 5 | 52 | 20 | 71 |
| Dassault | 6 | 121 | 112 | 233 |
| Dassault | 7 | 276 | 324 | 600 |
| Dassault | 8 | 109 | 73 | 182 |
| Dassault | 9 | 49 | 36 | 85 |
| Dutuit - Chinese | 10 | 25 | 36 | 61 |
| Dutuit - European 1 | 11 | 61 | 84 | 145 |
| Dutuit - European 2 | 12 | 32 | 40 | 72 |
| Dutuit - European 3 | 13 | 80 | 107 | 187 |

Table 4.2: Summary Characteristics of Benchmark Fault Trees Used in Study ${ }^{\text {[Rau93] }}$

Using the BDD approach and ordering 1 the memory of the computer was reported as insufficient to analyse three of the nine Dassault trees. However, the results with the depth-first approach did not yield the same problem. Comparisons were made on computer time (given in seconds) which included the time to choose an index for each variable, the computation associated with the tree and the computation of the tree encoding the minimal cut sets. Their findings showed that for all thirteen trees the computation times were markedly reduced using the alternative ordering (ordering 2 ). The actual computation results were not given in the paper for ordering 1, but the computation times for ordering 2 were compared against results for calculating the BDD using the classical De Morgans algorithm (using the top event logic function to generate the BDD and applying Boolean laws of algebra where necessary). The algorithm was only used on the Dassault trees and for two of these trees results could not be found in
reasonable time. The actual length of time this refers to is not given. It can be observed from the results in table 4.3 that the depth-first ordering is considerably quicker.

| Tree Label | Time for Ordering 2 (in seconds) | Time using DeMorgans algorithm (in seconds) |
| :---: | :---: | :---: |
| 1 | 3.13 | 10577 |
| 2 | 0.56 | 2470 |
| 3 | 0.03 | 147 |
| 4 | 0.03 | 130 |
| 5 | 0.01 | 74 |
| 6 | 0.65 | 43279 |
| 7 | 4.43 | 84383 |
| 8 | 0.05 | Not time |
| 9 | 0.05 | Not time |
| 10 | 0.5 | Analysis not performed |
| 11 | 4.36 | Analysis not performed |
| 12 | 0.31 | Analysis not performed |
| 13 | 6.81 | Analysis not performed |

Table 4.3: Computation Times For Two Variable Ordering Heuristics ${ }^{\text {[Rau93] }}$

Extending the Depth-First approach, the Priority Depth-First heuristic has been suggested. This approach still orders each subtree individually, but if there exists in a level any basic events that are the only inputs to a gate these are ordered first. Using the tree in figure 4.1 as an example, looking at Gate 2, at level 3 there are 3 basic event inputs, $E, A$, and $D$. Using the Depth-First approach these are just ordered from left to right, therefore as written. However, using the Priority Depth-First approach, as variables $A$ and $D$ are the only basic events inputs to Gate 5 , unlike Gate 4 which has gates and basic events, then these are ordered first to produce the ordering $A, D$ then $E$. The resulting ordering using the Priority Depth-First approach would be:

$$
A<B<C<H<K<G<D<E<F
$$

Similarly, as with the Depth-First approach, prioritising the repeated events can be applied, resulting in the ordering:

$$
A<B<C<H<G<K<D<E<F
$$

In each of these heuristics a slightly different ordering results, which can effect the size of the BDD. Research by Sinnamon and Andrews ${ }^{[5 A 296]}$ investigated the effects that different ordering schemes produce on the resulting size of the BDD. The six heuristics mentioned so far were the orderings that were investigated, namely, the top-down, leftright approach, the depth-first approach, the priority depth-first approach, and repeated event versions of each. The results showed that there were vast differences in the number of computations required to construct the BDD when each of the different orderings were used for the basic events. Hence, great savings can be made in terms of computation time and memory requirements when an efficient ordering of the basic events can be established. However, the research showed each tree has an individual variable ordering that will optimise its size and there is not a general ordering scheme that will be 'best' for all trees.

As with all the approaches discussed so far, these four closely related subtree based heuristics are also affected by the writing of the variables in the fault tree. This has the most effect when the ordering of the top event inputs are altered, hence affecting which subtree and associated variables are ordered first. Again if there are no repeated events in the tree, then the modified versions (or repeated event versions) of the depth-first and priority depth-first approaches are useless. Also the priority depth-first approach only leads to changes if gates exist with basic event inputs only. For more complicated trees this is not often the case and it seems that such gates are usually found lower down the tree and questions then need to be asked as to their relevance on the system occurrence.

### 4.3.3 Variable Ordering Using Repeated Basic Events and Subtree Levels

Sinnamon ${ }^{[S i n 96]}$ provides an ordering heuristic which combines using information about the repeated events and number of levels within a tree, with the depth-first approach. The depth-first approach is chosen as a basis for the heuristic as additional work has shown that this ordering option is generally reasonably good (section 4.3.2). The reasoning behind the inclusion of repeated events is that it is these events in the fault tree that have significant influence on the size of the $\mathrm{BDD}^{[\mathrm{SAl96]}}$. Therefore combining these attributes should lend itself to a reasonable ordering heuristic. The heuristic is named REBESUL, which considers REpeated Basic Events and SUbtree Levels. The algorithm for REBESUL is based on six steps.

Step 1: $\quad$ Create a list of the repeated events in the fault tree, those with the highest number of occurrences are listed first. Repeated events that have an equal number of occurrences are placed in rows between the next highest and lowest.

Step 2: $\quad$ For each repeated event in step 1, create a list of the subtrees (first sons of the top gate) that contain this repeated event. List the subtrees in order of the highest number of different repeated event occurrences within each subtree to the lowest.

- If two or more subtrees share the same number of repetitions for an event, the subtree with the greatest number of levels takes precedence over how many repetitions there are in a subtree.

Step 3: Create a list of the levels in each subtree at which the repeated events in step 2 occur.

Step 4: Order the gates (depth-first) starting with the gate that 'contains' the lowest level occurrence (obtained in step 3 ) of a repeated event, followed
by the other gates which 'contain' the next level of occurrence of a repeated event. Note that the term 'contains' does not necessarily mean that a repeated event is a direct input to the gate, it may be an input a few levels down. List the repeated events first when ordering the inputs of each gate.

Step 5: If all the repeated events have been dealt with in this subtree order any remaining events to gates in the subtree depth-first and go to step 6. Otherwise go to step 3 for the next repeated event obtained in step 1.

Step 6: If all subtrees containing repeated events have been dealt with order any remaining subtrees depth-first. Otherwise order the next subtree containing repeated events, i.e. go to step 2.

To illustrate the heuristic, consider the fault tree in figure 4.3 (taken from research by Sinnamon ${ }^{[\operatorname{Sin} 96]}$ ). It contains eight basic events with four of these being repeated. The steps of the algorithm will produce the following results:

Step 1: $\quad$ List of repeated events $-A$ occurs three times, $B$ occurs twice, $E$ occurs twice and $G$ occurs twice.

Step 2: $\quad$ Finding subtrees in order of most different repeated events: Subtree 2 (G3) has the highest number of repetitions (four different repeated events), therefore is ordered first. Subtree 1 (G2) has three different repeated events.

Step 3: Find highest level of occurrence of a repeated event. Event $A$ occurs at level 3 and 5 of G3.

Step 4: $\quad G 6$ contains the lowest level occurrence of $A$ and $G 5$ contains the next level of $A$ (here $A$ is an input to $G 10$ which in turn is an input to $G 8$ which
in turn is an input to G5), therefore take the order of gates as $G 6, G 5, G 8$, G10, G9 which provides the basic event ordering:

$$
A<D<F<B<G<H<E
$$

Step 5: $\quad$ Go to step 6.

Step 6: Order subtree 1. This provides the last basic event $C$, so the final ordering produced is:

$$
A<D<F<B<G<H<E<C
$$



Figure 4.3: Fault Tree Used to Demonstrate REBESUL Ordering ${ }^{[\text {[Sin96] }}$

Sinnamon compared the results of this ordering against that for the depth-first approach. The depth-first approach was used as within her doctoral study this ordering produced the most number of optimal BDDs when using a test set of fifty one trees. This analysis was carried out on the same fifty one fault trees, whose characteristics are summarised in Appendix I. The comparative results for the number of nodes in the BDD are shown in table 4.4.

| Tree | Depth first | REBES UL | Tree | Depthfirst | $\begin{gathered} \text { REBES } \\ \mathrm{UL} \end{gathered}$ | Tree | Depthfirst | $\begin{gathered} \text { REBES } \\ \text { UL } \end{gathered}$ |
| :---: | :---: | :---: | :---: | :---: | :---: | :---: | :---: | :---: |
| 1 | 308 | 281 | 18 | 394 | 252 | 35 | 4 | 4 |
| 2 | 8 | 7 | 19 | 11 | 11 | 36 | - | 8762 |
| 3 | 4 | 4 | 20 | 12 | 10 | 37 | 6 | 6 |
| 4 | 26 | 26 | 21 | 104 | 122 | 38 | 413 | 501 |
| 5 | 20 | 20 | 22 | 59 | 52 | 39 | 4 | 4 |
| 6 | 41 | 38 | 23 | 162 | 179 | 40 | 4 | 4 |
| 7 | 63 | 63 | 24 | 42 | 42 | 41 | 8 | 8 |
| 8 | 61 | 61 | 25 | 475 | 550 | 42 | 5 | 5 |
| 9 | 60 | 60 | 26 | 7 | 7 | 43 | 2 | 2 |
| 10 | 40 | 40 | 27 | 7 | 7 | 44 | 4 | 4 |
| 11 | 5 | 4 | 28 | 21 | 21 | 45 | 14 | 16 |
| 12 | 61 | 62 | 29 | 19 | 19 | 46 | 390 | 382 |
| 13 | 20 | 20 | 30 | 21 | 21 | 47 | 6 | 4 |
| 14 | 22 | 22 | 31 | 366 | 491 | 48 | 8 | 7 |
| 15 | 33 | 33 | 32 | 39 | 60 | 49 | 6 | 6 |
| 16 | 335 | 201 | 33 | 38 | 46 | 50 | 7 | 7 |
| 17 | 647 | 506 | 34 | 7 | 6 | 51 | 12 | 13 |

Table 4.4: Comparing Number of Nodes in BDD For Depth-First Ordering and REBESUL ${ }^{[\mathrm{Sin} 96]}$

The lowest number of nodes for each tree has been highlighted. For 26 trees there is a match between the lowest number of BDD nodes produced by each ordering heuristic. In total the depth-first ordering produces the lowest number of nodes on 37 occasions, whereas the REBESUL ordering generated the lowest for 41 out of the 51 fault trees. Thus, this REBESUL ordering proves to be slightly more efficient than the depth-first approach. The study also looked at the ite computations required to generate the BDD before and after minimisation, and with the REBESUL technique 19 out of the 51 trees produced a minimal BDD directly.

One point with this heuristic is that it is not so drastically affected by the re-writing of the tree when repeated events are contained within it, as it is the repeated events which govern which subtree is selected first. For non-repeated events however these can still be affected but then the significance of such events may be less than for repeated events.

### 4.3.4 Applying Weights to The Depth-First Approach

Minato et al., ${ }^{[\mathrm{MrY91]}}$ generated an ordering heuristic based on the depth-first approach. The study uses circuit diagrams although the approach can be applied to fault trees. The method is termed 'dynamic weight assignment method'. The fundamental reasons given for the development of the heuristic are the following properties (as related to circuit diagrams):

1. The inputs that greatly affect the output functions should be high in the order.
2. The inputs whose connections are topologically close to one another in the circuit should be near in the order.

Applied to fault trees these properties imply that those inputs (basic events) which have most affect on the top event should be ordered high in the list, and those basic event inputs that lie close to each other in the tree should be ordered together.

The methodology begins by assigning the top event with the value of 1 . This weight is propagated down through the tree such that at each gate the weight is equally distributed between its inputs. Having executed this weighting mechanism the highest order is given to the basic event with largest weighting. The theorem is that the weight reflects the contribution to the top event in a topological sense and thus the basic event with the highest weight should contribute most to the top event. Given the fault tree in figure 4.4, the weights generated for the basic events are:

- $\mathrm{B}=\mathrm{C}=\mathrm{D}=1 / 6$
- $\mathrm{A}=1 / 3+1 / 6=1 / 2$


Figure 4.4: Fault Tree Used to Illustrate Minato Heuristic

Thus, the first basic event in the ordering list would be $A$. In order to choose the next primary input, the part of the tree that leads to the chosen basic event is deleted, and weights are reassigned from the beginning. Therefore, using the fault tree in figure 4.4, the branches leading to $A$ would be deleted resulting in the fault tree given in figure 4.5 .


Figure 4.5: Fault Tree on Second Stage of Applying Weights

Although strictly not a viable fault tree structure it is used only to generate an ordering. Again the weights are assigned from the top of the tree, and passed down through to the basic events. The highest weighting is then added to the list, hence variable $D$ will be next in the ordering. Deleting part of the diagram means that the last assignment is distributed to the neighbouring inputs so that their weights are increased. Thus, the neighbouring inputs are given near positions in the ordering list. This process is continued until all the basic events have been ordered.

Minato tested the ordering on 12 circuit diagrams. The dynamic weight allocation ordering was compared to three others: the original circuit diagram ordering (as written), the reverse original ordering and a random ordering. Both the number of nodes in the resulting BDD and the time in seconds to obtain the result are included in table 4.5. Column A refers to the dynamic weight allocation ordering results, column $B$ the original ordering results, column C the reverse original ordering results and column D the random ordering results. The number of nodes in the resulting BDD for half of the circuits are less than when using any other ordering.

The method does consider the influence of repeated events, by adding their weighted values, which has in previous studies proven to be beneficial in terms of the ordering generated. Although the performance is good for circuit diagrams the ordering methodology has not been applied to fault trees directly. Even without this, it is clear that
the ordering is not going to produce an optimal BDD for all circuits, and the same result is hypothesised for fault trees.

| Circuit | A-DWAO <br> No.nodes/time | B - Original No.nodes/time | C-Reverse No.nodes/time | D - Random No.nodes/time |
| :---: | :---: | :---: | :---: | :---: |
| Dec8 | 40 / 0.3 | 41/0.3 | 390/0.4 | 57/0.4 |
| Enc8 | $33 / 0.3$ | 31/0.3 | 30/0.3 | 37/0.4 |
| Add8 | 49/0.4 | 120/0.4 | 452 / 0.4 | 1183 / 0.6 |
| Add16 | $97 / 0.7$ | $248 / 0.5$ | $1700 / 0.9$ | 94814 / 24.1 |
| Mult4 | $330 / 0.5$ | 358/0.4 | 304/0.5 | 394 / 0.5 |
| Mult8 | 46594 / 18.3 | 38187/14.5 | 31026/14.0 | 77517 / 26.1 |
| C432 | 89338 / 34.1 | 11348 / 7.4 | 6205 / 5.6 | 479711 / 278.6 |
| C499 | 36862 / 21.5 | 68816/39.1 | 32577 / 21.0 | 112815 / 78.0 |
| C880 | 30548 / 11.5 | $>500000$ | $>500000$ | $>500000$ |
| C1355 | 119201 / 51.4 | 246937/102.9 | 103301/46.9 | 373974/179.0 |
| C1908 | 39373 / 22.5 | 47990 / 22.7 | 65895 / 63.3 | 91082 / 47.4 |
| C 5315 | 40306 / 29.8 | 105200 / 32.5 | $>500000$ | $>500000$ |

Table 4.5: Results of Minato's Study on BDDs Generated For Circuit Diagrams ${ }^{[\text {MIY91] }}$

### 4.3.5 Using The Number of Leaves in Conjunction With The Depth-first Approach

In the paper by Bouissou et al., ${ }^{[8 B R 97]}$ one ordering heuristic suggested was by Rauzy, but was unpublished. The ordering works by considering the number of connectives or leaves of each gate, whilst applying the depth-first approach to the fault tree. Gates with a lower number of leaves not already processed or ordered are considered first.

The leaves or connectives of a gate can be considered as the number of inputs and outputs to the gate. This is illustrated in figure 4.6, whereby it is shown that Gate 1 has 3 leaves i.e. one to Gate 3, one to variable $A$ and one to the Top event, and Gate 2 has 4 leaves
(one to each of the variables $D$ and $E$ and one to each of Gate 3 and Top). The same principle can be applied to all gates in the tree.


Figure 4.6: Illustration of Leaves of a Gate

Applying the ordering heuristic to the tree in figure 4.6, if the depth-first approach was being applied the gates would be considered in the order as shown, Gate 1 then Gate 2. As Gate 1 has fewer leaves than Gate 2 (three opposed to four) then the Gate 1 subtree is processed first. $A$ has fewer leaves than Gate 3, so $A$ becomes the first variable in the ordering list. Then the variables $B$ and $C$ of Gate 3 are ordered. So the partial order of $A$ $<B<C$ is gained. Considering Gate 2, as Gate 3 has already been processed, hence has the minimum number of unprocessed leaves, it is processed next, generating ordering $B<$ $C$ (as $B$ and C are already in the ordering list no variables are added). The remaining variables to order are then $D$ and $E$, to produce the final ordering $A<B<C<D<E$.

The effectiveness of this ordering heuristic has been researched in a comparative study by Bouissou et al., ${ }^{[B B R 97]}$ and it was found that this leaf method is good at producing a quick

BDD but not necessarily of smallest dimension. The inclusion of the output leaf is not relevant and can be discarded as each gate and event has one output leaf. The reasoning behind this method of ordering appears to be that in selecting those inputs which are associated with the least number of variables the size of the cut set is smaller, thus in theory having a larger influence on system failure. It can be seen from the example that the ordering is not always influential and the ordering list produced is the same as using the depth-first approach alone. This heuristic in the study did not seem to be affected by the re-writings of the variable inputs of each gate, however it seems that for all trees the results for generating an ordering which produces a good BDD are like most heuristics, variable.

### 4.3.6 Depth-First With Number of Fanouts Considered

Fujita et al., ${ }^{[7 F K 88]}$ using circuit diagrams derived an algorithm which applied the depthfirst approach and considered the number of fanouts of each gate. In a circuit diagram a number of wires enter into each gate and a number of wires go out of each gate. The formers are called fanins and the latter fanouts. The algorithm is applicable to a fault tree diagram as the circuit diagram can be viewed as a fault tree rotated through 90 degrees (i.e. both represent logic functions). In terms of fault trees the number of fanouts is therefore the number of occurrences of a gate in the tree, for example, consider the fault tree drawn in figure 4.6.

The number of fanins and fanouts of each of the events (both gates and basic events) are expressed in table 4.6. The final column refers to the number of fanouts of each event and it can be seen to refer to the number of occurrences of the event. In this example, all the events have one fanout apart from Gate 3, which has two.

| Event | Fanins | Fanouts | Number of Fanouts |
| :---: | :---: | :---: | :---: |
| Top | G1, G2 | None | 1 |
| G1 | A, G3 | Top | 1 |
| G2 | D, G3, E | Top | 1 |
| G3 | B, C | G1, G2 | 2 |
| A | None | G1 | 1 |
| B | None | G3 | 1 |
| C | None | G3 | 1 |
| D | None | G2 | 1 |
| E | None | G2 | 1 |

Table 4.6: Number of Fanins/Fanouts For Each Event in Figure 4.6

The aim of the research was to find an algorithm to minimise the 'number of crosspoints of nets', when drawing a circuit diagram. In relation to the fault tree approach this can be viewed as trying to minimise the mean number of fanouts within the tree. This viewpoint stems from the intuitive fact that in a fully modularised tree (which is easy to assess) each gate has only one fanout. Therefore, minimising the mean number should help in the assessment of the tree or BDD production. The research compared four variable orderings. These being:

1. Original ordering, as the circuit is written within the data file.
2. Manual ordering, an expert draws the diagram from its description and the ordering from the resulting diagram is taken.
3. Random ordering as generated by a computer.
4. New algorithm ordering (discussed below).

Two strategies are suggested in the research, which mean that the number of fanouts is reduced. These are:

Strategy 1: If the number of fanouts of all the gates are 1 then the best procedure is to traverse the diagram in a depth-first manner and order events as they are encountered.

Strategy 2: When the diagram has only one gate with more than one fanout the best procedure to order the events is to apply the depth-first procedure but to tackle the gate with more than one fanout first.

Incorporating these two strategies, the algorithm suggested involves traversing the diagram in a depth-first manner (according to strategy 1) and inputs which have more than one fanout are ordered first (as strategy 2).

| Name | Inputs | Outputs | Levels | Gates | Max gates for <br> single output | Max inputs for <br> single output |
| :---: | :---: | :---: | :---: | :---: | :---: | :---: |
|  |  |  |  |  |  |  |
| 1 | 14 | 8 | 9 | 34 | 55 | 14 |
| 2 | 36 | 7 | 7 | 203 | 145 | 36 |
| 3 | 41 | 32 | 11 | 275 | 102 | 41 |
| 4 | 60 | 26 | 24 | 469 | 130 | 45 |
| 5 | 41 | 32 | 24 | 619 | 322 | 41 |
| 6 | 33 | 25 | 40 | 938 | 557 | 33 |
| 7 | 233 | 140 | 32 | 1566 | 828 | 122 |
| 8 | 50 | 22 | 47 | 1741 | 1433 | 50 |
| 9 | 178 | 123 | 49 | 2608 | 937 | 67 |
| 10 | 32 | 32 | 124 | 2480 | 2327 | 32 |
| 11 | 207 | 108 | 43 | 3827 | 1096 | 194 |

Table 4.7: Characteristics of Circuits Used In Ordering Study ${ }^{[F F K 88]}$

Using figure 4.6 , Gate 3 is the only gate with more than one fanout therefore when carrying out the depth-first approach the inputs of Gate 1 would be considered as Gate 3 first then variable $A$. Hence the ordering using this new algorithm would be $B<C<A<$
$D<E$, whereas using the depth-first approach alone would have generated the ordering list of $A<B<C<D<E$.

The circuits used in the study have the characteristics given in table 4.7. The results from Fujita's study comparing the four ordering schemes tested are summarised in table 4.8. The new algorithm, incorporating the fanout notion, is proven to generate a BDD for 8 out of the 11 test circuits within the study in fewer computer-processing minutes than any of the other heuristics (these results are highlighted in table 4.8). On some occasions the algorithm produces a worse result (see circuit number 2 for example), thus even this algorithm does not yield the best solution for all circuit diagrams.

|  | Original Order | Manual Order |  | Random Order |  | New Algorithm |  |  |
| :---: | :---: | :---: | :---: | :---: | :---: | :---: | :---: | :---: |
|  | Time <br> $(\mathrm{sec})$ | Max <br> Nodes | Time <br> $(\mathrm{sec})$ | Max <br> Nodes | Time <br> $(\mathrm{sec})$ | Max <br> Nodes | Time <br> $(\mathrm{sec})$ | Max <br> Nodes |
| 1 | 8 | 339 | 7 | 197 | 8 | 578 | 6 | 213 |
| 2 | 109 | 1146 | 53 | 442 | $>50000$ | $>100000$ | 1423 | 6196 |
| 3 | 928 | 9020 | 673 | 4661 | $>50000$ | $>100000$ | 673 | 4661 |
| 4 | $>50000$ | $>100000$ | 85 | 3421 | $>50000$ | $>100000$ | 55 | 3359 |
| 5 | 675 | 9020 |  |  | $>50000$ | $>100000$ | 1009 | 4661 |
| 6 | 965 | 2912 | 1800 | 4505 |  |  | 778 | 3076 |
| 7 | $>50000$ | $>10000$ |  |  | $>50000$ | $>100000$ | 338 | 14763 |
| 8 | $>50000$ | $>100000$ |  |  | $>50000$ | $>100000$ | 12620 | 53460 |
| 9 | 2140 | 11807 |  |  | $>50000$ | $>100000$ | 498 | 3441 |
| 10 | $>50000$ | $>100000$ |  |  | $>50000$ | $>100000$ | $>50000$ | $>100000$ |
| 11 | $>50000$ | $>100000$ |  |  | $>50000$ | $>100000$ | 383 | 2096 |

NB. Blank cells refer to untested circuits
Table 4.8: Comparison of Four Ordering Schemes Used ${ }^{\text {[FFK88] }}$

One of the problems of using this heuristic is shown in the Fujita study, in that the heuristic does not produce a best solution for all circuit diagrams, and this point can assumedly be extended to fault trees. Using a fault tree example, if this heuristic is
applied to the fault tree drawn in figure 4.1, as there are no gates with more than 1 fanout then the theorem applies the standard depth-first approach hence the ordering will be no different to that of the depth-first approach. Also the algorithm does not explain what to do if more than 1 gate has more than 1 fanout. Questions like 'do we apply the algorithm per subtree', i.e. if the tree has two gates with more than one fanout but with one in each subtree can the algorithm be carried out as usual? Or alternatively, should all gates with more than 1 fanout be ordered first when applying the depth-first approach and should the gates with the most fanouts be ordered first? These types of questions are not addressed in the paper and the answers are thus not clear.

### 4.4 Comparative Studies of Heuristics

Bouissou et al., ${ }^{[B B R 97]}$ compared and evaluated six different variable ordering heuristics. Some of these heuristics were already discussed in the literature and in addition the authors proposed new ones. The heuristics discussed were:

1. The formula or fault tree is explored in a depth-first manner, and the variables put in an ordering as soon as they are encountered.
2. The inputs are ordered depending on the number of fanouts of each gate in conjunction with applying the depth-first traversal (this heuristic was proposed by Fujita et al., ${ }^{[\text {FFK88] }}$, section 4.3.6).
3. The variables ordered in a depth-first manner again, with inputs with fewer leaves (or branches) being placed first in the ordering (section 4.3.5).
4. Weights with a value of 1 are assigned to each basic event of the fault tree. The weight of each gate is obtained by adding the weights of its inputs. When all the weights are known a depth-first traversing of the tree is made, choosing at each level the variables of a gate by order of increasing weights.
5. This heuristic combined heuristics 2 and 4 above. Applying the weight heuristic following the re-writing in terms of the heuristic using the number of fanouts, generates method 5 of ordering.
6. This heuristic combined heuristics 2 and 3 above. A similar approach is taken here as in method 5 , the tree structure is altered using the heuristic in method 3 relating to the number of leaves, and then the weight heuristic is applied to get the final ordering.

Before the research is discussed clarification is made regarding the orderings of 4,5 and 6. The weighting ordering of method number 4 is similar to a heuristic proposed by Minato ${ }^{[\text {MrY91] }}$. The heuristic involves three steps and the fault tree shown in figure 4.7 is used to illustrate its effects.


Figure 4.7: Fault Tree Used To Illustrate Weightings Ordering

Step 1: Each basic event is assigned a weight of value 1. Each intermediate event is assigned the weight of the value of the sum of its inputs. For example in figure 4.7, the weights of all the gates and events are:

$$
w(A)=w(B)=w(C)=w(D)=w(E)=w(F)=w(G)=w(H)=1
$$

$$
\begin{gathered}
w(\text { Gate } 7)=w(E)+w(G)+w(H)=3 \\
w(\text { Gate } \sigma)=w(A)+w(B)=2 \\
w(\text { Gate } 5)=w(E)+w(F)+w(G)=3 \\
w(\text { Gate } 4)=w(F)+w(\text { Gate } 7)=4 \\
w(\text { Gate } 3)=w(B)+w(C)=2 \\
w(\text { Gate } 2)=w(\text { Gate } 5)+w(\text { Gate } 0)=5 \\
w(\text { Gate } 1)=w(D)+w(\text { Gate } 4)=5 \\
w(\text { Top })=w(\text { Gate } 1)+w(\text { Gate } 2)+w(\text { Gate } 3)=12
\end{gathered}
$$

Step 2: The inputs of each intermediate event are arranged in ascending order of weight values, hence the inputs of gate Top are ordered as Gate $3<$ Gate $1<$ Gate 2 with values ranging from 2 to 5 . The inputs of Gate 3 remain as shown, i.e. $B<$ $C$, and Gate 1 also remains unchanged. Gate 2's inputs are reversed with Gate 6 now before Gate 5. Gate 4's inputs are altered to become $F$ then Gate 7. Gate 5, 6 and 7 remain the same as these gates just contain basic event inputs.

Step 3: A depth-first traversal is made of the tree, and the resulting ordering produced:

Hence, the ordering is generated as follows:

$$
\begin{gathered}
\text { Gate } 3<\text { Gate } 1<\text { Gate } 2 \\
\{B<C\}<\{D<\text { Gate } 4\}<\{\text { Gate } 6<\text { Gate } 5\} \\
B<C<D<F<\text { Gate } 7<A<B<E<F<G
\end{gathered}
$$

Therefore, removing variables that have already occurred higher up the list, the ordering is:

$$
B<C<D<F<E<G<H<A
$$

Ordering number 5 is a combination of the fanout heuristic with the weighting algorithm applied to the result. Using ordering 6 to demonstrate the combinatorial technique will
illustrate the point, as finding a case when the two heuristics of ordering 5 together make a difference to the variable list is difficult, and the question needs to be asked of its usefulness as a heuristic. It may be that its effect is greater on more complicated trees and not the reasonably simple trees used to illustrate the principles of each technique.

Ordering number 6 takes a similar approach to 5 , and combines the 'number of leaves' heuristic with the weighting algorithm. Using the fault tree in figure 4.1 to demonstrate the effects, initially the tree is scanned and each gate is analysed for the number of leaves it has. The following results are:

```
Gate 1-4 leaves (B, C, Gate 3, TOP)
Gate 2-3 leaves (Gate 4, Gate 5, TOP)
Gate 3-3 leaves ( \(H\), Gate 6, Gate 3)
Gate 4-3 leaves ( \(E\), Gate 7, Gate 2)
Gate 5-3 leaves ( \(A, D\), Gate 2)
Gate 6-3 leaves ( \(K, G\), Gate 3 )
Gate 7-3 leaves ( \(F, G\), Gate 4)
```

The ordering generated by applying the leave heuristic, alters the Gate 1 and Gate 2 inputs around and would be:

$$
A<E<F<G<D<B<C<H<K
$$

Applying the weightings now to the tree, the values for each gate will be:

- Gate $1=5 ;$ Gate $2=5 ;$ Gate $3=3$; Gate $4=3$; Gate $5=2 ;$ Gate $6=2 ;$ Gate $7=2$.

The gates are then scanned from lowest to highest weightings as applying the depth-first approach, thus Gate 4 and Gate 5 exchange positions as Gate 5 has a lower weight value than Gate 4. This change means that the variables $A$ and $D$ are ordered before $E, F$ and $G$, and the resultant ordering is:

$$
A<D<E<F<G<B<C<H<K
$$

The research was carried out on 13 real life coherent trees from a varied industrial background. The characteristics of these benchmark trees are summarised in table 4.9.

| $\mathrm{No}$ | No. of events | No, of gates | No. of <br> Minimal <br> Cut Sets | Fail <br> (\%) |  | Max <br> Nodes | Mean <br> Nodes | Std. Dev <br> Nodes |
| :---: | :---: | :---: | :---: | :---: | :---: | :---: | :---: | :---: |
| 1 | 61 | 81 | 46188 | 0 | 1817 | 30121 | 5750.7 | 2212.37 |
| 2 | 32 | 40 | 4805 | 0 | 145 | 1943 | 457.952 | 142.404 |
| 3 | 80 | 107 | 24386 | 0 | 3305 | 27349 | 9926.87 | 3229.01 |
| 4 | 53 | 30 | 16701 | 0 | 53 | 153 | 79.4012 | 15.9311 |
| 5 | 103 | 145 | 8060 | 0 | 2614 | 20614 | 4833.58 | 1332.81 |
| 6 | 458 | 434 | 7520142 | 27.08 | 551 | 289421 | 9437.65 | 22412.5 |
| 7 | 278 | 251 | 94016 | 0 | 7562 | 128847 | 53581.8 | 25209.2 |
| 8 | 196 | 142 | 497 | 0 | 255 | 4727 | 1094.16 | 774.067 |
| 9 | 548 | 484 | 5604253 | 12.85 | 8705 | 691766 | 185871 | 118286 |
| 10 | 120 | 178 | 529984 | 5.5 | 145 | 34557 | 1325.65 | 2517.62 |
| 11 | 116 | 122 | 5197617 | 0 | 708 | 2993 | 1629.68 | 334.158 |
| 12 | 91 | 95 | 3431 | 0 | 703 | 10950 | 4203.71 | 1783.94 |
| 13 | 71 | 65 | 150436 | 0 | 149 | 805 | 409.923 | 64.4269 |

Table 4.9: Benchmark Trees Used in Comparative Study ${ }^{[B B R 97]}$

The first column indexes each of the thirteen trees. The next three columns identify some characteristics of the trees, the number of variables within the structure (i.e. the number of basic events), the number of gates, and the number of minimal cut sets respectively. To evaluate the effects of changing the variable inputs of a gate, the research involved random re-writings (changing the order of gate inputs) of each tree. In total these 6 heuristics were tested on 500 random re-writings of each tree. The fifth column labelled
'fail (\%)' refers to the percentage of times within all the trials that the BDD could not be produced for that tree with less than $10^{6}$ nodes. The four remaining columns show the variations in the size of the BDDs produced with the different re-writings. The minimum number of nodes within any of the BDDs produced for re-writings, the maximum number of nodes, the mean number of nodes for all the trials and the standard deviation are all summarised.

The results indicated that there were two classes of heuristics. One class, containing heuristics $3,4,5$ and 6 gave low standard deviations on the size of the BDD produced for all the trials, indicating that the heuristics are not very sensitive to the rewritings of the fault tree variables. The other class, containing the remaining two heuristics (1 and 2 in the list) showed high variations in the size of the BDDs produced. This indicates that these ordering heuristics are very sensitive to the position that the variables are placed in for each gate. The research showed that the heuristic using the number of fanouts produced better results than when using the depth-first approach. However, the problem with these heuristics was that both can lead to extremely large BDDs.

The research indicated that the first class of heuristics produced 'neither excellent' nor 'not bad' BDDs, but rarely the best BDD. The second class, on the other hand, had the greatest exploration potential to yield smaller BDDs yet extremely large cases could result.

### 4.5 Alternative Heuristics

### 4.5.1 Modules of a $B D D$

Bouissou ${ }^{[B o u 96]}$ reasoned that many heuristics have been proposed in order to find an acceptable ordering with low computer requirements and highlighted that the possible flaw in achieving this with all of the heuristics suggested is due to their lack of theoretical
foundation. The research points out that many heuristics are sensitive to the writings of the fault tree, i.e. shown previously in this chapter. Most heuristics are based on intuitions, i.e. variables with "bigger" influence should be placed higher up in the ordering. However, their experiments have shown that the positioning of cut sets of order one hardly has any effect.

The paper highlights a new theoretical relationship between BDD size and modules of a fault tree. A module of a fault tree can be explained simply as an independent subtree of the whole tree. In more complex terms, it can be defined as a subtree composed of at least two events which have no inputs from the rest of the tree and no outputs to the rest of the tree except from its output event. A pseudo-module is a subtree that has two outputs to the rest of the tree.

A number of methods have been proposed to detect modules or independent subtrees in a fault tree. The most efficient and simplest algorithm has been produced by Dutuit and Rauzy ${ }^{[\mathrm{DRa} 96]}$. The modular approach uses Rauzy's linear time algorithm to detect independent subtrees. The basic principle of the algorithm can be stated as follows:

Let $v$ be an internal event (gate) and $t_{1}$ and $t_{2}$ respectively the first and second dates of visits of $v$ in a depth-first left most traversal of the fault tree. Then $v$ is a module iff none of its descendants is visited before $t_{1}$ and after $t_{2}$ during the traversal. ${ }^{\text {[DRa96] }}$

By definition Top and each basic event are modules. Using this algorithm the tree is traversed twice, both in a depth-first manner. In the first traversal, counters are set to record the number of visits to a node (gate/event). In the first visit to a node the first_visit counter is set and in the second visit the second_visit counter is set. Further visits to a node increments a counter last_visit. This last visit is identifying the same node outside of the one already found, hence any repetitions. If the counter of a node is set then the inputs to that node are not visited, and also for basic events the first and second visits are identical.

To illustrate the algorithm consider the fault tree shown in figure 4.8. The numbers positioned by the side of each node indicate the counter, starting at 1 for the top event and ending when the top event is visited for the second time. Table 4.10 indicates the progression of visits made by the counter, which reflects the numbers shown on the fault tree.


Figure 4.8: Fault Tree Used to Illustrate Module Finding Algorithm

| Counter | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |
| :---: | :---: | :---: | :---: | :---: | :---: | :---: | :---: | :---: | :---: | :---: |
| 'v' | TOP | Gatel | A | Gate 3 | D | E | Gate 3 | Gate 1 | Gate 2 | Gate 4 |
| Counter | 11 | 12 | 13 | 14. | 15 | 16 | , | 18 | 19 |  |
| 'v' | D | C | Gate 4 | Gate 5 | F | G | Gate 5 | Gate 2 | TOP |  |

Table 4.10: Progression of Traversal Through Tree and Associated Counter Values

Using table 4.10, tabulation of the first, second and last visits to a node can be calculated.
The visits for each node are summarised in table 4.11.

| Visit | TOP / | Gate | Gate | Gate 3 | Gate 4 | Gate 5 | A | $D$ | $\mathbf{C}$ | $\mathrm{E}$ | Fs | G |
| :---: | :---: | :---: | :---: | :---: | :---: | :---: | :---: | :---: | :---: | :---: | :---: | :---: |
| $1^{\text {st }}$ | 1 | 2 | 9 | 4 | 10 | 14 | 3 | 5 | 12 | 6 | 15 | 16 |
| $2^{\text {nd }}$ | 19 | 8 | 18 | 7 | 13 | 17 | 3 | 5 | 12 | 6 | 15 | 16 |
| Last | 19 | 8 | 18 | 7 | 13 | 17 | 3 | 11 | 12 | 6 | 15 | 16 |

Table 4.11: First, Second and Last Visit Counters

Having identified the counters for all the nodes in the fault tree the second traversal of the tree can now occur. Again, a depth-first traversal is made but this time the minimum of the first_visit counter and the maximum of the last_visit counter for each of the gates inputs are established. The inputs to the node are examined, as it is possible that repeated events lie lower down and are not the immediate inputs of the node. On generating this information all the data is available to establish whether the gate is a module or not. To establish this, the definition states that: the gate ' $v$ ' is a module iff the collected minimum is greater than the first visit of $v$ and the collected maximum is less than the second date of $v$. The minimum of the first visit must be greater than the first_visit counter for the gate otherwise it means that the inputs have occurred before this gate. If the maximum of the last_visit counter is greater this indicates that the input has occurred outside of the gate or node (after this gate has been visited), hence it is also positioned elsewhere in the tree. In both of these instances one or more of the inputs of the gate are not independent of the node. The minimum, maximum and Module identification information is contained in table 4.12.

| GATE | TOP | GATE | GATE 2 | GATE 3 | GATE 4 | GATE 5 |
| :---: | :---: | :---: | :---: | :---: | :---: | :---: |
| Minimum | 2 | 3 | 10 | 5 | 5 | 15 |
| Maximum | 18 | 7 | 17 | 11 | 12 | 16 |
| Module | YES | YES | YES | NO | NO | YES |

Table 4.12: Second Traversal Results

From table 4.12 the linear time algorithm indicates that the modules of the fault tree are gates TOP, Gate 1, Gate 2 and Gate 5. This result is what is expected as Gates 3 and 4 both contain the repeated variable $D$, rendering Gate 3 and 4 dependent.

| Tree | Variables <br> /Gates | Min Cut <br> Set | Aralia BDD size min/max | Aralia - <br> BDD CPU <br> (s) min/max | Metaprime <br> - BDD size min/max | Metaprime - <br> BDD CPU <br> (s) $\min / \mathrm{max}$ |
| :---: | :---: | :---: | :---: | :---: | :---: | :---: |
| 1 | 196/142 | 497 | 766/ | 0.65/2.01 | 1567/ | 1.18/1.9 |
|  |  |  | 2882 |  | 2283 |  |
| $\mathrm{Op}(1)$ | 196/65 | 497 | 287/ | 0.03/0.33 | 352/ | 0.12/0.26 |
|  |  |  | 43060 |  | 627 |  |
| 2 | 282/244 | 115042 | 8232/ | 2.98/19.08 | 27292/ | 16.97/26.16 |
|  |  |  | 45135 |  | 32249 |  |
| $\mathrm{Op}(2)$ | 282/129 | 115042 | 7419/ | 2.0/13.2 | 6540/ | 1.72/10.3 |
|  |  |  | 40222 |  | 30873 |  |
| 3 | 306/337 | 7520142 | 28447/ | 18.8/363.8 | 24021/ | 32.6/158.6 |
|  |  |  | 156118 |  | 92549 |  |
| Op(3) | 306/259 | 7520142 | 8093/ | 6.0/40.6 | 15159/ | 19.0/33.8 |
|  |  |  | 29132 |  | 28723 |  |
| 4 | 311/289 | 105955422 | 75253/ | 12.6/229.2 | 64793/ | 16.0/77.8 |
|  |  |  | 435903 |  | 302489 |  |
| Op(4) | 311/193 | 105955422 | 55723/ | 10.7/ | 66486/ | 19.1/70.9 |
|  |  |  | 349780 | 1197.7 | 346741 |  |
| 5 | 548/484 | 5604253 | 5488/ | 0.7/66.5 | 10822/ | 4.0/149.5 |
|  |  |  | 260925 |  | 154452 |  |
| Op(5) | 548/166 | 5604253 | 3668/ | 0.3/22.9 | 2788/ | 0.4/26.9 |
|  |  |  | 227953 |  | 158434 |  |

Table 4.13: Results of Module Optimisation on BDD Construction ${ }^{[B o u 96]}$

The research produced a program to 'optimise' the tree, basically convert the tree to its modularised form. The results presented from the research ${ }^{[B 0 u 96]}$ compared the results of applying tools such as ARALIA and METAPRIME to the optimised and normal fault trees. The ordering used in the ARALIA and METAPRIME tools was not stated, however, the results given in the research were for a select few of a set of twenty trees. Table 4.13 indicates that for all the optimised versions of the trees (Op) the minimum number of nodes was less. The trees were tested on 100 re-writings of the variables, to test the re-writing theory, hence producing a minimum and maximum number of nodes for the BDD and minimum and maximum computation times.

The research showed that restructuring the fault tree, in order to create as many modules and pseudo-modules as possible, is an efficient pre-processing method applied before conventional ordering techniques to help alleviate some of the ordering problems. One possible constraint for any ordering heuristic using this method is that it should group the variables of a module. Using this modular property, the problem of determining a global optimal ordering can be split into smaller problems: determining optimal orderings for each module.

The technique is good in theory, but it does not state how much time is spent on preprocessing, or how exactly the modules can be extracted. The programming tool produced with the research would need to be used to benefit from this approach.

### 4.6 Dynamic Ordering Methods

Reported within the literature is the topic of dynamic variable orderings ${ }^{[\mathrm{Nik} 99]}$. The research focuses on circuit diagrams, which deal with multiple functions. These dynamic ordering approaches involve the swapping of variables. Although the research is centred around circuit diagrams the principle can be applied to the BDD approach whereby an initial BDD is produced and then variables within the BDD are exchanged resulting in a new BDD. This swapping or exchanging of variables continues until a smaller BDD results. Once the BDD is constructed its analysis is a linear function of the number of
nodes within it, therefore it is more efficient to carry out the quantitative analysis using the non-minimal BDD than to continually swap variables until the minimalist BDD is found and then to carry out the desired analysis. Thus the usefulness of these dynamic variable orderings in improving the resulting size of the BDD seems non-existent and the approaches will not be discussed further.

### 4.7 Performance of Heuristics

In summary, despite there being a considerable set of possible heuristics and a considerable amount of research carried out to investigate their usefulness, it is not possible to determine which would yield the best results for a given fault tree structure. Due to the radically different nature between some fault trees it seems unlikely that a single simple heuristic will be enough to cope with producing the best ordering for all possible trees. The latter point has been observed with the varying performance of heuristics on different fault trees.

By reviewing this previous research it is apparent that no one rule based approach has been identified for all fault trees. Similarly, no single heuristic has been identified which out performs any other. To make the BDD approach an efficient analysis procedure for all fault trees a 'super' ordering heuristic needs to be generated, one that will produce a minimal BDD for all trees. Or alternatively, the best scheme option needs to be selected from a set of alternatives to guarantee at least a near minimal BDD. The research in this thesis focuses on generating a rule based approach using pattern recognition techniques and in addition looks at a completely new heuristic to try and solve the ordering problem.

### 4.8 Summary

- No one single heuristic has been identified as 'good' for all fault tree structures.
- Many alternative ordering heuristics have been proposed and investigated with few strong conclusions.
- Investigations into this ordering problem has been conducted by two research communities, electronic circuits and fault trees. The diverse nature of the logic functions used in these applications make the validity of conclusions drawn from the circuits questionable in terms of appropriateness for fault trees.
- The variable ordering approaches can be separated into two approaches i) an ordered traversal of the fault tree giving a high priority to the neighbourhood so variables are placed in the ordering close to their neighbours; ii) weighting methods where adjacent variables in the ordering can be found from different sections of the tree structure.
- The majority of research and investigation has been performed on the ordered traversal approach.
- Relatively little research has been conducted on weighting methods.
- The performance of each heuristic is variable depending on the fault tree structure being analysed.
- The repeated event approach seems to perform well on a number of examples, however is not the solution to all tree structures.
- Most methods are dependent on the ordering of the inputs to the gates - if the fault tree is represented by changing the order of inputs the same ordering heuristic can result in a very different sized BDD.
- To promote an efficient BDD analysis and to take full advantage of this efficient and accurate fault tree analysis technique research needs to be carried out to determine an approach to select a good ordering heuristic for all trees.
- Rule based pattern recognition type approaches may be the way ahead in terms of finding a reasonable solution to the problem.


## 5 Pattern Recognition Techniques - The Machine Learning Classifier System Incorporating Genetic Algorithms

### 5.1 Introduction to Pattern Recognition Techniques

### 5.1.1 Summary of The General Approach

All "pattern recognition" approaches provide a means of identifying patterns or rules within a population encompassing a number of data examples. The patterns may be relatively simple i.e. classifying a population into separate girl and boy categories given a few distinguishing characteristics, or more complex with a series of inputs and possible outcomes. There are many different pattern recognition models that can be generated for a given problem domain, these include machine learning classifier systems ${ }^{[G 0197]}$, neural networks ${ }^{[B i s 95]}$, expert systems ${ }^{[\text {Yaz89] }}$, Bayesian methods and fuzzy logic ${ }^{[\text {[ESD96] }}$. The pattern recognition potential of these approaches can be used to solve a wide range of information processing problems. Some example problems are: speech recognition; the classification of hand written characters; and medical diagnosis.

The variable ordering problem can be specified in terms of finding a relationship between the fault tree structure and the best ordering scheme option that will produce an optimal BDD in the conversion process. Thus, finding the pattern between the input (the fault tree structure) and the required output (the best ordering scheme option) is characteristic of a pattern recognition problem. None of the approaches have rules indicating which type of problems they are best suited to solve, hence, there is no way to determine which of these would be the most effective to apply to the ordering scheme problem.

In this chapter the pattern recognition technique using the machine learning classifier system which incorporates a genetic algorithm is discussed. Initially the genetic algorithm concept is reviewed as it forms the corner stone of the functioning and ultimate success of the classifier system. A classifier system has been formed to model the ordering problem with limited success. Correct predictions of scheme choice for small fault trees are more common than for larger tree structures. The research steps, results, and problems encountered are all explained. Four papers were produced following this research ${ }^{[B A n 98, ~ A B a 98, ~ B A n 99, ~ B A n 991] ~}$.

### 5.1.2 Types of Problems Modelled by Pattern Recognition Techniques

Pattern recognition techniques encompass two main types of problem namely, classification and regression problems. Classification problems involve outcomes that can be grouped into distinct classes. For example, consider the problem of assigning a group of individuals into two classes one representing girls (class $C_{l}$ ) and the other boys (class $C_{2}$ ). The outcome can be viewed numerically as a 1 if the individual belongs to class $C_{l}$ and 0 if it belongs to $C_{2}$. Therefore, in general, the classification problem can be viewed as a mapping from a set of input variables, $x_{l}, \ldots, x_{d}$, representing each individual, to an output variable, $y$, representing the class label. For more complex tasks there may be a number of output classes.

The regression category, on the other hand, refers to problems in which the outputs represent values over a continuous range. For example, the prediction of the average height of an individual in the year 2000, given a large data set of average heights for the last 10 years.

The variable ordering problem can be modelled within the classification category. Each of the six scheme alternatives used in the study would form a separate class, and take the value of 1 if the input vector belonged to that class and 0 if not.

### 5.1.3 Considerations With All Pattern Recognition Techniques - The Curse of Dimensionality

When applying a pattern recognition approach to a problem a model incorporating the principles of the technique is generated to establish the pattern in the data. One consideration to be made when using any pattern recognition approach is the phenomenon known as 'the curse of dimensionality' ${ }^{[B 0082]}$. This phenomenon is concerned with the amount of information that the model is given. In theory, it would be sensible to suggest that the more information the model is given (in terms of the inputs of the problem not the number of examples within the data set) the easier its task of finding the pattern. However in reality, if the model is given too many parameters it becomes overloaded and performance deteriorates. In practice, beyond a certain point, adding new input features can actually lead to a reduction in performance of the pattern recognition model generated. Unfortunately this 'certain point' is not known and will vary depending on the problem.

### 5.1.4 What Are Genetic Algorithms (G.A's)?

John Holland and colleagues at the University of Michigan ${ }^{[H 0171]}$ first introduced genetic algorithms in the early 1970's. Genetic algorithms are stochastic search algorithms based upon the mechanics of natural genetics. The genetic algorithm operates upon populations of character strings in a similar manner to the genetic action on chromosomes in a population of organisms. In simple terms, genetic algorithms enforce the Darwinian survival of the fittest principle upon an artificial population of creatures (defined as strings). Each generation of a new strings is created using selected material from the fittest of the old generation. Genetic algorithms efficiently exploit past information to explore new regions in what is referred to as 'the search space' with a high probability of finding improved performance. The search space contains all the possible inputs for the problem.

### 5.1.5 Why Are They Used in The Classifier System Approach?

The classifier system approach to pattern recognition generates a model which contains a set of rules. These rules map to the patterns in the problem domain to guide the search using new inputs toward the correct response. These rules are learnt through a training phase where the model generated is subjected to a set of training examples, which is a large collection of inputs and outputs taken from the problem domain. To search for the appropriate rules or generate new rules that may more accurately reflect the pattern in the data, a rule generation system is required. This rule generation system often takes the form of a genetic algorithm. As the rules of the classifier system are one of its fundamental elements, this generation tool is discussed in detail to reflect its importance to the functioning of this type of pattern recognition approach.

### 5.2 Genetic Algorithm Principles

Genetic algorithms apply slightly different techniques to those of calculus based or enumerative search methods. One difference is that they work with a coding of the parameter set, usually in terms of a binary alphabet, whereas alternative approaches work directly with the parameter set.

To increase the diversity of possible areas to search G.A's utilise a database of points simultaneously, in contrast to many optimisation methods that search from a single point. This is like climbing peaks in parallel thereby reducing the likelihood of finding false peaks in the solution. The G.A starts with a population of randomly generated strings and thereafter generates successive populations of strings. The population's poor strings are weeded out and replaced with successive populations generated by the genetic algorithm, hence, the population should continually increase in strength.

The general operation of a G.A uses no auxiliary information just the values of the objective function associated with individual strings. Conventional approaches however require the use of auxiliary information, for instance, gradient based approaches require
the knowledge of the gradient at a specific point and this may not always be possible if the function is not continuous.

To perform an effective search G.A's use probabilistic transition rules. Unlike random walk type approaches the random rules are used as a guideline for searching. They use this random choice as a tool to guide a search toward regions of the search space with likely improvement.

### 5.3 A Simple Genetic Algorithm

### 5.3.1 The Principles

The basic components of the genetic algorithm are called strings. These are usually coded in binary and represent the input parameters of the objective function. For example, given the objective function $f(x)=x^{2}$ for $x$ in the range $(0,31)$, then the string representing $x=15$, could be: 01111. If more than one variable is involved then these are combined to make a continuous string. For example, if trying to find the optimal number of pumps and valves in series, with a maximum of 3 pumps and 2 valves, the binary representation of both would be two bits each. To model the problem these bits would be combined to form a single string, i.e. if there are two pumps [10] and one valve [01], the string would take the form: 1001.

The strings are grouped into sets called populations. The aim of the genetic algorithm is to generate successive populations (known as generations) based on the information contained in the previous population. To start the genetic algorithm the population is generated randomly. From this initial population strings are selected in a probabilistic manner to enter a mating pool for the next generation. Within this mating pool a series of genetic operations occur to inject new bits, possibly better bits, into some strings. Following this genetic operation the newly produced generation is decoded and the list of possible results of the problem assessed. The generation procedure is repeated until an
adequate solution is found, i.e. if there are any constraints then if these are met then a solution is obtained. The algorithm can be summarised as follows:

- Generate initial random population, $\mathrm{P}(0)$;
- Repeat next steps until solution found
- Generate $\mathrm{P}(t)$ using $\mathrm{P}(t-1)$ by applying genetic operators
- Evaluate P(t);

The main components for the successful execution of the genetic algorithm are the genetic operators. These help to maintain the strong sections of already good strings to produce new strings with possible improvement, and to maintain diversity. A simple G.A is comprised of three main operators:

- Reproduction.
- Crossover.
- Mutation.


### 5.3.2 Reproduction Operator

The process of reproduction in a genetic algorithm is a means to pass information from the current population into the next generation. Just as in human biology where genes are passed from the 'parent' to the 'child'. Within the genetic algorithm reproduction is a process in which individual strings are copied. The probability of strings being reproduced from the current population and passed into the mating pool for passage into the next generation is based on the objective function values (or fitness values) of each string. Copying strings according to their fitness means that strings with a higher value have a higher probability of contributing one or more offspring in the next generation and passing on important characteristics.

The implementation of the reproduction operator can occur in numerous ways, the simplest being to create a biased roulette wheel. Roulette wheel selection can be viewed as allocating pie-shaped slices of a circular shape to population members, with each slice proportional to its fitness, shown diagrammatically in figure 5.1. The overall effect of this selection process is to return a randomly selected parent. This parental selection technique has the advantage that it directly promotes reproduction of the fittest population members by biasing each member's chances of selection in accordance with its fitness evaluation. Once a string has been selected for reproduction, the string is copied. This string is then entered into what is termed a mating pool, the next successive population, for further genetic operator action. The members of the mating pool selected in this way should comprise of a set of strong members whose characteristics provide them with high fitness. Due to the stochastic nature of the selection process some less fit members may progress to the mating pool for the next generation. In this way the population diversity is maintained.


Figure 5.1: Roulette Wheel For Parent Selection. (Classifier C has the highest fitness value occupying $38 \%$ of the wheel)

### 5.3.3 Crossover Operator

Following the execution of reproduction, the genetic operator of crossover is implemented. Simple crossover may proceed in two steps. First, the newly reproduced strings in the mating pool are mated at random. Second, each pair of strings exchange information as follows: an integer position $k$ along the string is selected at random between 1 and the string length less one $[1, n-1]$. Two new strings are created by
swapping all characters between positions $k+1$ and $n$ inclusively. For example, consider strings A1 and A2:

$$
\begin{aligned}
& \mathrm{A} 1=110110 \mid 101 \\
& \mathrm{~A} 2=111100 \mid 010
\end{aligned}
$$

Suppose in choosing a random number between 1 and $8, k=6$ is obtained (as indicated by the separator symbol $\mid$ ). The latter sections of each string are then crossed resulting in two new strings, A1' and A2':

$$
\begin{aligned}
& \mathrm{A} 1^{\prime}=110110010 \\
& \mathrm{~A} 2^{\prime}=111100101
\end{aligned}
$$

This process explores the possibility of exchanging good features of the two 'parent' strings to create an even better 'child'.

### 5.3.4 Mutation Operator

Mutation is needed because the reproduction and crossover operators, despite searching the space effectively and identifying strong areas, occasionally may loose some potentially useful genetic material. In the simple genetic algorithm mutation is implemented with small probability, and creates a random alteration of the value of a bit in a binary string. It randomly produces a new point in the search space and is one of the mechanisms that helps explore the whole search space in an attempt to produce global rather than local optima.

### 5.3.5 The Operation of a Simple Genetic Algorithm

To illustrate the workings of a genetic algorithm, a simple optimisation problem to maximise the function $f(x)=x^{2}$, if $x$ varies over the range 0 to 31 is used as an example.

The first step in the procedure is to code the input parameter $x$ into a finite binary string. This would involve using a five bit binary code, whereby $\{00000\}$ represents the minimum value of $x$ over the range, and $\{11111\}$ represents the maximum, 31 .

To start the genetic algorithm process an initial population is needed. This is randomly generated. Table 5.1 shows an assumed initial population of 4 strings with associated objective function values.

| Binary coded version of $x$ | Integer value of $x$ | Objective Function value $f(x)$ | Probability of being reproduced |
| :---: | :---: | :---: | :---: |
| 01101 | 13 | $169\left(=13^{2}\right)$ | 169/1170 $=0.144$ |
| 11000 | 24 | 576 | $576 / 1170=0.492$ |
| 01000 | 8 | 64 | $64 / 1170=0.055$ |
| 10011 | 19 | 361 | $361 / 1170=0.309$ |
| Objective function sum Objective function average |  | $\begin{gathered} 1170 \\ 292 \end{gathered}$ |  |

Table 5.1: Input Population For Genetic Algorithm

The first stage of the genetic algorithm begins with reproduction. A mating pool of the next generation can be selected by spinning a weighted roulette wheel, where the segments are proportional to the objective function value. Therefore dividing the string objective function value by the total objective function value for the whole population generates the proportions. In this example, the objective function sum totals 1170. Therefore, for this example population string number 2 will occupy the largest proportion of the wheel and string number 3 the smallest. Spinning the wheel four times will
generate the next successive four strings. Thus, for each string a random number $x$ would be produced between 0 and 1 , such that if:

$$
\begin{array}{ll}
0.000 \leq x \leq 0.144 & \text { String } 1 \text { would be selected } \\
0.144<x \leq 0.636 & \text { String } 2 \text { would be selected } \\
0.636<x \leq 0.691 & \text { String } 3 \text { would be selected } \\
0.691<x \leq 1.000 & \text { String } 4 \text { would be selected }
\end{array}
$$

For this example it is assumed that the following strings are reproduced:

- String 1 is copied once (string 1 in mating pool).
- String 2 is copied twice (strings 2 and 3 in mating pool).
- String 3 is not selected.
- String 4 is copied once (string 4 in mating pool).

Simple crossover is the next genetic operator to alter the population. Strings are mated and the crossing sites selected both randomly. Table 5.2 shows the effect of mating strings 1 and 2 , and 3 and 4 of the new population together with crossover points 4 and 2 respectively. The last operator mutation is performed on a bit by bit basis. If a mutation rate of 0.001 is assumed then with 20 bit positions to possibly change, it is expected that $20^{*} 0.001$ changes would result. That is, 0.02 changes. Therefore, there are no mutation bit changes i.e. 0 's changed to 1 's and vice versa, in this example.

The new population is now ready to be tested. The population is decoded and the associated objective function values calculated. These results are shown in table 5.2. From the comparison of table 5.1 and 5.2 it is clear that the average objective function value and the sum of the objective function values have increased. The process would be repeated over many generations and the best (optimal) strings selected.

|  | Mating Pool | Mate | Crossover Site | New population | $x$ value | Objective Function value |
| :---: | :---: | :---: | :---: | :---: | :---: | :---: |
| 1 | 0110\|1 | 2 | 4 | 01100 | 12 | 144 |
| 2 | 1100\|0 | 1 | 4 | 11001 | 25 | 625 |
| 3 | 11\|000 | 4 | 2 | 11011 | 27 | 729 |
| 4 | 10\|011 | 3 | 2 | 10000 | 16 | 256 |
| Objective Function SumObjective Function Average |  |  |  |  |  | 1754 |
|  |  |  |  |  |  | 438 |

Table 5.2: Genetic Operator Applied to G.A Forming New Population

### 5.4 Implementation of The Genetic Algorithm Within The Classifier System

The genetic algorithm used in the machine learning classifier system is like the simple genetic algorithm explained in section 5.3. The algorithm uses the same tripartite processes of reproduction, crossover and mutation. However, the action of this genetic algorithm is to produce only a selected proportion of the next generation. Good performers in the previous generation are allowed to survive without modification. New randomly generated elements are also added to make up the final segment of the new generation in an attempt to generate better classifiers.

### 5.5 Background to Machine Learning Classifier Systems

### 5.5.1 Introduction

Early work in machine learning focused on tasks like learning from examples and language acquisition, employing methods that relied on significant amounts of data and
ultimately considerable search. Since the first practical implementation of a genetics based machine learning system a number of researchers have extended and applied the work to a variety of fields. In biological and medicinal areas a number of systems have been developed for medical diagnosis, for example for the detection of cancerous cells. In engineering fields genetic systems have been applied to gas pipeline optimisation and control tasks. There have also been applications to computer science, social science and even military uses.

The overall principle of the classifier system is to generate a model of the problem that incorporates a learning mechanism, whereby through a training process the rules that govern the problem are learned for later predictive purposes. Due to this learning mechanism contained within a system framework the classifier system is referred to as a "machine learner". Classifier systems that use a genetic algorithm are often called genetics based machine learning systems (GBML).

To generate the classifier system to model the patterns of the problem a large amount of training needs to be undertaken. This involves starting with a classifier system containing random rules and by subjecting the model to lots of sample population data (termed the training data set) rules are generated within the system that reflect the patterns (or relationship between the inputs and the outputs) within the problem.

Following the training phase and having established that the classifier system developed adequately models the pattern contained within the problem the system can be used to predict the outcome for new unseen inputs (using the test data set). The rules learnt in training provide the mechanism to direct the search operation toward the desired output response.

### 5.6 Machine Learning Systems - The Classifier Model

### 5.6.1 General Overview of Model

Classifier systems are a rule-based machine learning system, with general mechanisms for processing rules in parallel, for testing the effectiveness of existing rules and for adaptive generation of new rules. These rules provide the predictive or pattern recognition potential of the classifier system. The system can be described in simple terms of taking a number of inputs of a problem and classifying them into specific outputs depending on the learned rules that are contained within it. A classifier system, depicted schematically in figure 5.2, consists of three main components:

1. Rule and message system.
2. Apportionment of credit system.
3. Rule and message generation system.


Figure 5.2: Schematic Representation of a Classifier System.

The functioning of the three main components of the system summarise all of the classifier systems actions. The rule and message system provides the mechanism for
processing rules, for example, taking the inputs of the problem and matching the appropriate rules. The Apportionment of Credit System tests for the effectiveness of the existing rules as each rule is given a strength depending on its appropriateness in modelling the pattern of the problem. For adaptive generation of new rules, a rule and message generation system is used, whereby new rules are injected into the system.

The predictive operation of a classifier system begins with the receipt of an input message that represents the characteristics of the problem to be solved. This input is sent to the internal message list (a store for current messages). A check is made to see which rules (initially randomly generated) are activated. The rules are defined as classifiers, and match inputs to a generated output or action. Matching classifiers bid to place their messages on the message list. In the case of more than one match, a bidding process in the Apportionment of Credit System decides the winning classifier (the most successful to meet the known output determined by the training data set). Once a winning classifier has been found, this then posts its message to the message list, and the process is repeated until an eventual winner is found. The winner is established when there are no more matches within the classifier store or after a predetermined number of matches occur (defined within the program). This winner then outputs its result, the solution to the input problem.

In the training phase, the only difference is that after all the training data set has been entered or passed through the classifier system new rules are injected into the system. This requires a rule generation system, often in the form of a genetic algorithm. Following this, training is repeated on the same data set, until the desired set of rules is produced which model the problem pattern.

### 5.6.2 The Rule and Message System

The rule and message system forms the computational backbone of the machine learner. This part of the system contains the pattern recognition potential. This potential is unleashed via the rules and the messages that control the path selecting the appropriate
response for a given input. The fundamental elements of this system are called 'classifiers', that is the rules. Initially a random set of classifiers (usually in the form of binary strings) are generated and the new classifiers are learnt and adapted during training on a large random sample data set from the population.

Each classifier consists of two parts: a condition; and a message. The roles of the two parts are to allow for complicated patterns to be modelled. It is impossible to have within the finite rule set all the rules that could map any input to the correct output. If the classifier has just one section the process would be: if the input message is matched by a rule then this is the required response. This would generate a simple mapping, however, in practical situations the pattern to map is usually far more complicated and also the number of input variations is extensive. Therefore the pattern potential is increased by each classifier having an identification part and a 'next step' part, i.e. the condition part identifies the match and the message sends a new bit of information, searching for the next step in the pattern. The training process establishes a link or relationship between the condition and its message. It is like a jigsaw puzzle, whereby each piece is a part of the pattern and each new message sent is a new bit of the puzzle. When all the messages have been sent and there are no more matches then the jigsaw is complete and a solution is found.

The condition is usually coded in the form of a binary alphabet, which models the input characteristics of the problem. This condition part is a simple pattern recognition device where a wild card character (\#) is added to the underlying alphabet. The wild card character is added to cater for a range of input possibilities. As stated above only a finite number of rules can be modelled within the system and hence similar inputs can be mapped on to the same rule. An example expression for the condition may be:

$$
\text { condition } \Rightarrow\{0,1, \#\}^{n}
$$

i.e. condition $=01 \# 01 \# \# 111$, where $n=10$. Thus, a condition matches a message if at every position a ' 0 ' in the condition matches a ' 0 ' in the message, a ' 1 ' matches a ' 1 ',
and a '\#' matches either. Hence, a number of inputs may be matched by this rule for example, all three of the classifiers listed below would be matched by the example condition.

- 0100111111
- 0110111111
- 0110100111

The message part within the classifier is a means of information exchange, and is simply a finite length string over some finite alphabet. In terms of a binary alphabet, the message can be defined as:

$$
\text { message } \Rightarrow\{0,1\}^{n}
$$

i.e. message $\Rightarrow 0110100111$, where $n=10$. Thus, the message is defined as a concatenation of 0 's and 1 's of length $n$. The message of each classifier is the search mechanism for the next part of the pattern and is sent to look for the next rule that matches it. For some patterns it is not always one rule that will lead to a prediction, a number of rules mapping out the complicated pattern may be required and this is the principle behind continually posting a new message.

Both the condition and the message parts of a classifier contain coded input characteristics of the problem and coded outputs of the problem. An example condition or message part of a classifier is:


### 5.6.3 The Apportionment of Credit Algorithm (A.O.C)

The A.O.C is the competition phase of the classifier system, implemented in terms of a bidding process. This is where any matched classifiers bid to post the next message into the system. This bidding process is referred to as the bucket brigade algorithm. In the training phase of the learning classifier system a random rule set is generated with each classifier (or rule) given an equal strength value, indicating that each rule is as good as each other in terms of predicting the correct outcome. Each time a match occurs the matched classifier, say classifier $i$, makes a bid, $B_{i}$ proportional to its strength value $S_{i}$ :

$$
B_{i}=C_{b i d} * S_{i}
$$

where $C_{b i d}$ is equal to the bid coefficient representing the proportion of the strength value to be used in the bid, eg. 0.01 , of the strength of classifier $i$. The bid coefficient is assumed to be the same for the whole of the bidding process.

Having matched and then bid, the strength of the classifier is reduced. The classifier is rewarded if it wins the bidding process by gaining the bid of the next classifier that was activated by its own message. Hence, if a classifier bids and is unsuccessful its strength remains reduced. This process hopes to weed out those rules that are not very good in predicting the required pattern, i.e. their strength becomes less and less.

More precisely, in the next step the matched and activated classifier (the one with the highest bid) gives its bid to the classifiers responsible for posting the previous message that matched the winning bidding classifier's condition. Then this increases the strength of the winning classifier hence increasing the likelihood that if matched again its bid will be higher. This process starts to identify the strong rules in the system.

In this bidding process, some classifiers are not matched by the inputs of the system, this could be due to the finite size of the training data set or that the classifier is not applicable to the problem. Either way the classifier works by biasing performance toward those
classifiers that are productive, hence classifiers that exist at each generation are taxed for their existence. This is to prevent freeloading. A tax, $T_{i}$, proportional to the classifier's strength is deducted:

$$
T_{i}=C_{t a x} * S_{i}
$$

where $C_{t a x}$ is the tax coefficient, eg. 0.001 . Together these relationships, of bidding and taxing define the apportionment of credit algorithm. Hence, to evaluate classifier $i$ 's strength value at time $t+1$ given the strength values at time $t$, the following difference equation is given:

$$
S_{i}(t+1)=S_{i}(t)-C_{b i d} \cdot S_{i}(t)-C_{t a x} \cdot S_{i}(t)+R_{i}(t)
$$

where $R_{i}(t)$ is the reward given to the classifier either from another classifier for its message sending activity or from the environment for correctly predicting the right outcome.

To illustrate with a simple example the workings of the A.O.C system consider five classifiers, shown in table 5.3. Each classifier has a condition part and a message part, each with string length of four binary bits. Assuming initial strength values of 500 for all five classifiers, and providing an initial input (environmental) message of 0101 , where the first two digits $\{01\}$ represents the characteristics of the problem and the last two digits $\{01\}$ represents the output of the problem, the workings of the algorithm can be considered. A bid coefficient of 0.1 is assumed and the bid is taken as the product of this bid coefficient and the classifier's strength. To simplify this example, the tax coefficient is omitted.

In the initial step $(t=0)$, classifiers 1,4 and 5 match the input and each bid 50 units. Since they all have the same bids the tie is broken by random selection, and the winning classifier sends it's message during the next time step. Classifier 4 is randomly selected to win and pays it's bid to the party responsible for it's activation; in this case, the input
(environment) strength is increased by 50 units as the environmental message was responsible for activating classifier 4 . In subsequent time steps, activated classifiers make their payment to previously active classifiers, as can be seen in the next time step ( $t$ $=1$ ).

| $\cdots$ | Classifier ${ }^{\text {a }}$, | Strength: | Message ${ }^{\text {a }}$ | M' | Bid | Strength: | Message, | M | Bid, - C |
| :---: | :---: | :---: | :---: | :---: | :---: | :---: | :---: | :---: | :---: |
| 1 | 01\#\#: 0000 | 500 |  | e | 50 | 450 |  | 4 | 45 |
| 2 | 00\#0:0011 | 500 |  |  |  | 500 |  |  |  |
| 3 | 11\#\#: 1001 | 500 |  |  |  | 500 |  |  |  |
| 4 | 0\#01: 0111 | 500 |  | e | 50 | 450 | 0111* |  |  |
| 5 | \#101:1100 | 500 |  | e | 50 | 450 |  |  |  |
|  | Environment(e) | 0 | 0101 |  |  | 50 |  |  |  |


| - | Classifier, | Strength | Message. | M | Bid | Strength | Message. | Payoff | Strength |
| :---: | :---: | :---: | :---: | :---: | :---: | :---: | :---: | :---: | :---: |
| 1 | 01\#\#: 0000 | 405 | 0000 |  |  | 455 |  |  | 455 |
| 2 | 00\#0:0011 | 500 |  | 1 | 50 | 450 | 0011 | 0 | 450 |
| 3 | 11\#\#: 1001 | 500 |  |  |  | 500 |  |  | 500 |
| 4 | 0\#01:0111 | 495 |  |  |  | 495 |  |  | 495 |
| 5 | \#101: 1100 | 450 |  |  |  | 450 |  |  | 450 |
|  | Environment | 50 |  |  |  | 50 |  |  | 50 |

*when two bids are equal, tie broken by random selection.
Table 5.3: Apportionment of Credit System.

Classifier 4's message to be matched is $\{0111\}$. The only match is from the condition part of classifier 1 and it makes a bid of 45 units. As it is the only match, this classifier wins and posts its message into the next time step $(t=2)$. At the same time, a payment of 45 units is made to classifier 4 , which was responsible for causing its activation. The strength of classifier 4 is then increased to 495 units. The process is again repeated with the new message of $\{0000\}$. In time step 2 , the match is from classifier 2 . This bids 50 units and again is the only bidder, hence is the winner. It posts its message $\{0011\}$ and pays classifier 150 units for causing its activation. This then increases the strength of classifier 1.

Finally, at time step 3 there are no further matches of the classifiers and hence classifier 2 is deemed the winner. As this is the final stage in the classifier systems prediction, this
winning classifier contains the predicted response. A reward is given to the winning classifier if all the matching and bidding has resulted in the correct prediction, hence indicating that this series of rules has detected an appropriate pattern. The process of adding a reward biases the bidding process and also the generation of new rules for future inputs towards stronger rules which have shown in the past to predict the required patterns. In this example, no reward is given, as the output of classifier $2,\{11\}$, did not match that of the input message, $\{01\}$. This means that the rules depicted in classifier 4 then classifier 1 then classifier 2 did not find the required pattern between the inputs and the output. Therefore this rule is not desired, which is reflected in the classifiers involved now having a reduced strength value. Note here that the tax has been removed and the strength of classifier 3 which has not been activated in the whole process would have decreased and not remained with the highest strength value.

In general, the iterative process of matching and sending messages continues until there are no more matches to be made with the classifiers contained in the classifier store. However, an upper limit can be set on the number of iterations carried out. If this is done caution must be taken in setting a limit which does not restrict the pattern finding potential of the classifier system. However, if an appropriate model can be achieved by limiting the number of iterations then valuable computation time will have been saved.

### 5.6.4 Rule and Message Generation System - Genetic Algorithm

The ability of the model to predict the correct outcome is dependent on the rules within. It is impossible to generate all the rules required to solve the problem for any input, and hence initially a random population of rules is generated. To construct an appropriate model a number of examples from the problem domain are put through the model in what is called a training phase. By using a large training set of examples the model learns the trends or patterns within the data which hopefully will generalise well to unseen data when the model is used in a predictive capacity. Inputting all the training examples through the model and carrying out the matching and bidding process which finally leads to an output, constitutes what is known as a training cycle. Following one training cycle
some rules will increase in strength whereas others will decrease. Without inputting more rules into the system and completing the training cycle for each training example in the set will only lead to adaptation of those rules that are already in the rule set. As the population is randomly generated to start with all possibilities for the rules are not included and hence more rules need to be generated. To inject new, possibly better rules, into the system requires the use of a rule and message generation system, one possible system is to use a genetic algorithm. The genetic algorithm principle and mechanics have already been discussed earlier in this chapter. Within the classifier system the genetic algorithm generates part of the new set of rules which are created using bits and pieces of the fittest of the old generation. They exploit the benefits of the information contained within the previous generation of rules to seek new rules with above average performance. The strong strings in the rule set remain the same and the genetic algorithm alters the rest. The part of the population that is kept can be set to any percentage of the total population. This percentage is one variable parameter that is determined by the problem.

### 5.7 Application of The Classifier Approach With a Genetic Algorithm to The Ordering Problem

### 5.7.1 Introduction

The basic principle of the machine learner is to take an input and depending on the characteristic properties of the input, and applying the rules learnt during training procedures, provide a response. Ultimately, the aim of the variable ordering problem is to take some characteristics of the fault tree structure and then to find a relationship between these characteristics and the scheme option that predicts the best ordering for minimal BDD generation. Therefore, the classifier approach has been adopted as a possible solution to the problem.

Discussed in the following sections are the necessary inputs and outputs of the genetics based machine learning model and how they are coded. The steps to generating the
desired model and the training and test performances of the intermediate and final genetics based machine learning classifier system that has been developed for the ordering problem are explained. The computer code for the genetics based machine learner ( $\mathrm{gbml} . \mathrm{c}$ ), produced during the research, is written in the C programming language.

### 5.7.2 Input Parameters Chosen To Represent The Problem

The requirement of the basic event ordering problem is that given a fault tree structure some methodology needs to be applied to generate an ordering of the variables to use in the conversion process to a BDD. This ordering needs to generate a minimal or at least near minimal BDD for an efficient analysis. Therefore, the inputs to the problem are the fault tree characteristics and the output is an ordering scheme selected from a list of alternatives. Several characteristics can be used to represent the significant features of the trees. It is not clear which features will be important in identifying an appropriate ordering heuristic.

Figure 5.3 indicates a simple fault tree structure with common distinguishing elements labelled. By examining the combinations of these factors a fault tree structure can be described. The list of possible characteristics is endless. Some possibilities include the number of gates; number of events; number of repeated events; number of levels, etc. It is clearly impractical to code all possible characteristics, hence a select few need to be extracted which represent the most significant tree features with relevance to the BDD construction. In view of overloading the genetics based classifier model with too much information, the fear of the curse of dimensionality, the set of characteristics initially chosen totalled four. These being:

- Percentage of AND gates:

As the number of AND gates increase the possible size of each cut set increases, hence there is more chance of non-minimal cut sets being formed.

- Percentage of different events repeated:

This characteristic gives an indication of the proportion of repeated events and non-repeated events within the tree. The higher the proportion of repeated events the higher the influence of some scheme choices.

- Percentage of total events repeated:

Considering all the events in the tree (all repetitions included), with a higher percentage of repeated events there is an increased likelihood of cut sets containing a repetition. As repeated events often lead to problems with minimisation then this characteristic is thought possibly influential.

- Top gate type:

There are two possible gates that are considered in the research as the starting gate of the alternating sequence of gates within the tree structure, namely AND and OR. This start point may influence the scheme choice, or at least link with the other three characteristics to affect the scheme option best suited to the tree.


Figure 5.3: A Simple Fault Tree with Labelled Elements

Percentages of characteristics were chosen rather than the numerical values to try and reduce the number of bits required for each string, thus simplifying the classifier systems task.

These characteristics have been selected because they are considered to be potentially important and the result gained through training and testing the classifier system using these characteristics will provide a benchmark from which future classifier designs can be assessed.

### 5.7.3 Output Parameters Chosen To Represent The Problem

The object is to produce an optimal or at least near optimal BDD representation of the fault tree. The ordering placed on the basic events of a fault tree will determine the size of the resulting BDD , and hence the number of cut sets. The smaller the number of cut sets the less processing involved to convert the BDD to one which encodes the minimal cut sets alone. Also the smaller the original BDD the more efficient (less paths to a terminal ' 1 ' vertex) the top event quantification process. It is beneficial to achieve an ordering which is optimal in terms of the resulting size of the BDD.

It is clear that numerous methods of variable ordering are available. In previous research ${ }^{[\operatorname{Sin} 96]} 6$ different ordering schemes have been identified which have produced reasonable orderings depending on the fault tree structure. These are:

- Top-down, left-right approach:
- is produced by listing the variables in a top-down, left-right manner from the original fault tree structure.
- Modified top-down, left-right approach:
- as the top-down, left-right approach but repeated events along each level are considered first.
- Depth-first approach:
- involves breaking the whole tree structure into smaller trees (subtrees) and looking at the optimal ordering of these subtrees. The depth-first ordering scheme gives each subtree a top-down, left-right ordering, working from the first gate inputs of the top event.
- Modified depth-first approach:
- As unmodified version but with repeated events considered first.
- Priority depth-first approach:
- takes the depth-first approach one step further and considers subtrees with only basic event inputs first.
- Modified priority depth-first approach:
- consider repeated events first in the ordering.

For each of the modified versions, if the gate or level has more than one repeated event as an input then the most repeated event is placed first, if they occur the same number of times then the events are taken in gate list order to break the tie.

There are many other schemes that could have been included. However, this short list has been adopted to investigate the potential of the classifier approach to select the best from among them.

### 5.7.4 Coding of The Fault Tree Parameters

The classifier system being generated has rules that are coded in terms of a binary alphabet. Each classifier message and condition comprises 19 bits, which can be broken down into two main sections, namely: the characteristics coding; and the scheme coding. The characteristics coding comprises 13 bits and is broken down to:

- 4 bits representing the percentage of AND gates;
- 4 bits representing the percentage of different events repeated;
- 4 bits representing the percentage of total events repeated;
- 1 bit for top gate type.

The binary coding for each of the characteristics was generated as follows:

1) Characteristic 1: Percentage of AND gates - to reduce the length of the binary string the possible percentages were grouped into subclasses, with each subclass representing the binary number from 0 to 15 (binary string length of 4 bits) as indicated in table 5.4.

| Percentage <br> Groupings | Binary Coding | Integer <br> Value. |
| :---: | :---: | :---: |
| $0-6.25 \%$ | 0000 | 0 |
| $6.25-12.5 \%$ | 0001 | 1 |
| $12.5 \%-18.75 \%$ | 0010 | 2 |
| $18.75-25 \%$ | 0011 | 3 |
| $25-31.25 \%$ | 0100 | 4 |
| $31.25-37.5 \%$ | 0101 | 5 |
| $37.5-43.75 \%$ | 0110 | 6 |
| $43.75-50 \%$ | 0111 | 7 |
| $50-56.25 \%$ | 1000 | 8 |
| $56.25-62.5 \%$ | 1001 | 9 |
| $62.5-68.75 \%$ | 1010 | 10 |
| $68.75-75 \%$ | 1011 | 11 |
| $75-81.25 \%$ | 1100 | 12 |
| $81.25-87.5 \%$ | 1101 | 13 |
| $87.5-93.75 \%$ | 1110 | 14 |
| $93.75-100 \%$ | 1111 | 15 |

Table 5.4: Coding For The Characteristic of "Percentage of AND gates"
2) Characteristics 2 and 3: Percentage of Different and Total Events Repeated - these two characteristics were coded in the same manner as the percentage of AND gates, i.e. split into 16 categories hence requiring four bits each.
3) Characteristic 4: Top Gate Type - as only two types of gate were considered in this problem the coding process only required the use of one binary bit, a 1 represented an AND gate, and a 0 an OR gate.

The numerical characteristic values associated with each fault tree used in the classifier system study are given in Appendix II. The program used to generate these characteristics was written in Fortran and named character.f.

### 5.7.5 Coding The Scheme Preferences

The scheme coding comprises of 1 bit for each scheme option, being either a 0 or a 1 . As stated earlier, the number of nodes in the resulting BDD diagram after the conversion process from the fault tree determined the best or optimal scheme option. This optimal option (the one with fewest nodes) was labelled a 'good' scheme and hence its binary bit coding will be a 1 . To illustrate how the conversion to the binary coding is achieved, two trees with resulting sized BDDs for each scheme option are highlighted in table 5.5, the equivalent binary coding has also been illustrated alongside.

| Scheme | Scheme 2 | Scheme 3 | Scheme 4 | Scheme 5 | Scheme 6 |  |
| :---: | :---: | :---: | :---: | :---: | :---: | :---: |
| Tree 1 - nodes | 37 | 36 | 31 | 57 | 32 | 31 |
| Tree 1 - binary | 0 | 0 | 1 | 0 | 0 | 1 |
| Tree 2 - nodes | 513 | 453 | 476 | 569 | 45141 | 38105 |
| Tree 2 - binary | 0 | 1 | 0 | 0 | 0 | 0 |

Table 5.5: Coding Conversion Method For The Scheme Preferences

The total length of the string required for the scheme coding is 6 bits, one for each scheme option. Using a bit for each scheme option allows for two equal schemes to both be expressed. Using only three binary bits, representing the numerical values of 1 to 6 , corresponding to each of the alternative schemes, would not allow for matched schemes to be accounted for. Therefore, overall each classifier message and condition comprises of 19 bits.

### 5.7.6 Data Sets

Once the classifier system is designed the model needs to be trained and tested using input data. Fault tree structures used in the training and test phases were obtained from industry and also by random production using a computer program (randtree.c). Each tree structure was analysed for the chosen characteristics, and these characteristics were converted to the appropriate binary representation.

Each tree was analysed prior to training for the best ordering scheme for the most efficient BDD representation. The best scheme was identified by the minimum number of nodes in the BDD structure before minimisation (removal of redundant nodes). This will measure the relative efficiency of the conversion process rather than the analysis i.e. the optimum / minimum BDD. The resulting BDD properties for each fault tree when analysed using each of the six variable ordering schemes is given in Appendix III. The results were gained using code already available, schemes. $f$, which was programmed in Fortran.

The typical BDD information produced for a given fault tree for each scheme option is shown in table 5.6. The first column refers to the six different scheme options. Columns 2 and 3 deal with the number of ite computations that are required to generate the BDD before and after minimisation. Column 4 is the difference between the number of ite computations before and after minimisation. The information used for this research to establish the best scheme is written in column 5 , the minimum number of nodes out of
the six scheme options prior to minimisation. Column 6 is concerned with the number of non-repeated nodes in the BDD before the minimisation process occurs. The number of nodes in the BDD after minimisation has occurred and the number of non-repeated nodes in the diagram form column 7 and 8 respectively.

| Scheme | $\begin{aligned} & \text { Ite } \\ & \text { before } \end{aligned}$ | $\begin{aligned} & \text { Ite } \\ & \text { after } \end{aligned}$ | Diff | Nodes before min | Non-rep nodes | Nodes <br> After | Non-rep after |
| :---: | :---: | :---: | :---: | :---: | :---: | :---: | :---: |
| Top-Down | 513 | 661 | 148 | 2471 | 94 | 447 | 90 |
| Mod TD | 523 | 660 | 235 | 2341 | 93 | 436 | 88 |
| Depth-First | 587 | 755 | 168 | 4874 | 81 | 357 | 81 |
| Mod D-F | 598 | 756 | 158 | 4374 | 80 | 354 | 79 |
| Priority D-F | 650 | 866 | 216 | 437189 | 151 | 1007 | 97 |
| Mod P D-F | 616 | 794 | 178 | 286685 | 143 | 1001 | 93 |

Table 5.6: Information Provided on BDDs For Each Scheme Option

To evaluate the performance of the learning classifier system a test set of data was produced with different tree structures and known best ordering schemes. The schemes for prediction purposes are set to wildcard characters or unknowns to be determined by the classifier system. The performance is evaluated by comparing the number of correct scheme outputs predicted.

### 5.7.7 The Programmed Model

The genetics based machine learning program ( $g b m l . c$ ) developed during the research was based on the simple framework of the classifier approach. The program has two modes of execution, one to train the classifier system based on the input training data set and one for prediction of the best scheme for a new tree structure. The outline of the program is given in figure 5.4:

## IF TRAINING MODE $\{$ main0 $\}$

- Generate random initial population of classifiers
- Assign strength values to each newly generated classifier
- Select number of iterations process to be trained for
- Repeat next steps until number of iterations exceeded
- For each environmental training pattern do the following
- Calculate matches \{matchclassifiers0\}
- If there is a match then begin the next stage of classifier system
- Apply bidding process in apportionment of credit system \{aoc:0\}
- Hold auction to determine winning classifier \{auction0\}
- Repeat matching process with new winner \{newaoc0\}
- Get output of winning classifier (scheme choice) \{effector0\}
- Does the scheme choice match the environmental message?
- Compare classifier outputs \{criterion0\}
- If outputs match then rewards given \{reinforcement(), payreward0\}
- At end of training patterns, cycle completed so use genetic algorithm \{ga0\}
- When iterations exceeded print final results to gmbl.out

ELSE IF PREDICTIVE MODE

- Identify number of test trees
- Define all parameters i.e. bids and strengths
- Read in each environmental test message in turn
- Find any matches \{matchclassifiers0\}
- Apply bidding process \{aoc0\}
- Find out if winner matches known scheme choice \{winnout(0\}
- Results written to file predict.out

Figure 5.4: Program Functioning of Classifier System

In the training phase, firstly classifier 1 (this refers to the coded data for the first fault tree structure) is passed to the classifier system. This string is compared with the rules within the classifier store (the randomly generated rules). If there is a match, the matching classifier posts its message back to the classifier store for possible further matches. This matching process is set to continue for up to three iterations. This number can be altered if desired, more complicated patterns may require further iterations of possible matches. Setting it at three will provide a benchmark for future results. When there are no more matches, the winning rule from the classifier store is output. If it matches the desired outcome, i.e. predicts the correct scheme option as given by the input classifier then a reward is given. Throughout the matching process a bidding process is also occurring to separate the bids of more than one matched rule. Following a response by the classifier
system for the first classifier in the training set, the second is then entered, and the same process occurs. When all the classifiers have been entered into the classifier store for matching, the rules are then reviewed using the genetic algorithm. Changes resulting from the genetic algorithm could provide better rules which more accurately relate to the training data set, hence the cycle is repeated. This cycle repetition is set for 1000 iterations. More training may be required, there is no way to determine how many iterations will be needed apart from evaluating the predictive scores produced.

### 5.8 Generating a Classifier Model

### 5.8.1 Using Initial Four Fault Tree Characteristics

The first trial to construct a machine learning classifier system to model the ordering problem had a training data set of 181 trees, and a test set of 20 fault trees. All of the trees had been scanned for the appropriate four characteristics (as discussed in section 5.7.2). Also, each tree has been run through the program schemes.f to establish the BDD for each of the six scheme alternatives (as discussed in section 5.7.3). Using this characteristic and scheme choice information the input classifiers for the training phase of the model and the classifiers for the prediction phase of the model can be formed.

To use the classifier approach an initial random set of rules (or classifiers) is required. The classifier store (or rule set) was initially set to fifty. These fifty classifiers were randomly selected using a random number generator (rand) embedded within the classifier program code. As each classifier is comprised of a condition section and a message section, each had to be randomised separately, hence the routine required three seed inputs to function (i.e. $23 / 3 / 3$ ), one for the rand function to perform, one to randomise the selection of the conditions and the other for the messages.

The parameters of the first classifier system model created were set to:

- $\quad$ Number of iterations before training process terminated $=1000$.
- Crossover operator set to 0.75
- Mutation operator set to 0.002
- Proportion of rule set selected to be operated on by genetic algorithm = 0.4
- Reward $=500$ units
- Strength of initial classifiers $=500$ units
- Results of the predictive phase are output to a file predict.out as a list of six digit alternatives ( 1 's and 0 's) i.e.
$011011=$ Schemes 2, 3, 5 or 6
$100000=$ Scheme 1
$101000=$ Scheme 1 or 3
With each row corresponding to an output of each test tree.

The test set of fault trees used in the whole of the classifier system research is kept constant to allow for comparisons to be made. The characteristics of the trees are summarised in table 5.7, along with the known best scheme options. These are given in the final column of table 5.7. The aim of the classifier system generated is to correctly predict these scheme options given the input characteristics of each tree only.

Following the execution of the training phase of the model, a summary of input parameters, i.e. the genetic algorithm operators and bidding coefficients are output to a file gbml.out. In addition the rules within the classifier stores and the corresponding strengths are output to a file to be used in the predictive phase. Execution of the training model using twenty different sets of three input seeds, i.e. twenty attempts at randomly choosing the initial population of rules, proved insufficient. This was apparent as the number of matches made with the test trees and the rule set was negligible. As the purpose of the model is to take any unseen fault tree and produce the best ordering heuristic for the BDD conversion, this aspect of the model needs substantial improvement. To try and increase the likelihood of a match between the test set and the classifier store the random set of classifiers was increased to 100 . Again the same
number generator was used to select the initial rule set. In the output file results for all test trees were not produced, i.e. no suggestions were made for a number of the twenty trees, indicating that there were no comparative trees within the rule set. Therefore, the predictions were poor, as the number of matches was poor. It was clear at this stage that the model was not set up to recognise the correct pattern contained within the problem domain.

| Tree | $\begin{gathered} \% \text { AND } \\ \text { gate } \end{gathered}$ | $\%$ Total events repeated | \% Different events repeated | Top gate type | Output <br> Required |
| :---: | :---: | :---: | :---: | :---: | :---: |
| 1 | 60 | 29 | 50 | 0 (OR) | 111011 |
| 2 | 43 | 32 | 55 | 0 | 010000 |
| 3 | 33 | 9 | 2 | 0 | 111111 |
| 4 | 42 | 30 | 60 | 1 (AND) | 001000 |
| 5 | 38 | 30 | 63 | 1 | 010000 |
| 6 | 65 | 25 | 16 | 1 | 110000 |
| 7 | 38 | 31 | 51 | 0 | 010000 |
| 8 | 45 | 3 | 1 | 1 | 000100 |
| 9 | 50 | 19 | 7 | 0 | 010100 |
| 10 | 39 | 32 | 60 | 0 | 010100 |
| 11 | 33 | 10 | 4 | 0 | 001000 |
| 12 | 41 | 9 | 8 | 1 | 001000 |
| 13 | 38 | 32 | 38 | 1 | 010000 |
| 14 | 50 | 13 | 5 | 1 | 001000 |
| 15 | 50 | 33 | 1 | 1 | 111111 |
| 16 | 44 | 28 | 17 | 0 | 111000 |
| 17 | 45 | 23 | 9 | 1 | 111111 |
| 18 | 60 | 21 | 14 | 1 | 010000 |
| 19 | 70 | 21 | 7 | 1 | 001000 |
| 20 | 31 | 28 | 48 | 1 | 001000 |

Table 5.7: Characteristics of Test Set of Trees With Known Required Outputs

There were two factors that became evident from this initial research. The first is the problem of matching. The number of wildcard characters within the condition part of the classifier is set randomly. With fewer wild card characters the number of rules required to map to all possible inputs needs to be increased. Currently the research has used a maximum data store of 100 , therefore, the current task is to try and map 20 new trees (of 19 bits) on to one hundred, when the number of possibilities is hugely more than that. This indicates that in future tests the random rule set should be much larger to try and encourage more matches.

A number of trials were then carried out with successive increases in the number of rules in the classifier store. The rules were incremented in the following steps: $100 ; 125 ; 150$; $200 ; 250 ; 300$ and then 500 . Ten trials were carried out for each rule set configuration. On each of the ten trials, the number of test trees for which a rule within the classifier store was found to match that of the input tree was noted. This match determines the number of possible predictions that could have been made. The average number of matches and the average predictive outcomes for each rule set configuration are summarised in table 5.8.

| Number in Rule <br> Set | Average Number of <br> Matches | Average Predictive <br> Score (20) |
| :---: | :---: | :---: |
| 100 | 1.2 | $0.5(0)$ |
| 125 | 1.3 | $0.4(0)$ |
| 150 | 1.1 | $0.4(0)$ |
| 200 | 2 | $0.3(0)$ |
| 250 | 1.8 | $0.6(1)$ |
| 300 | 2.7 | $0.7(1)$ |
| 500 | 4.4 | $1.7(2)$ |

Table 5.8: Outcomes With Varying Rule Set Sizes

From table 5.8 it can be seen that as the number of random rules is increased the number of matches tends to increase. This increase has direct effects on the potential of the classifier system to make correct predictions. Even with more matches the predictive performance of the system is very poor. This is partially due to the fact the system may predict more than one scheme as the best. Included within this selection could be the correct scheme, but as other schemes are also predicted which are not the best, this result is incorrect. For example, the classifier system may make the prediction $\{010011\}$ and the answer required may be $\{010000\}$, so despite the system predicting scheme 2 which is correct, by predicting schemes 5 and 6 as well makes the solution incorrect. One possibility for improvements is to increase the rule set further. Another factor to be investigated is the four characteristics that describe the fault trees. The question needs to be asked as to whether these four characteristics which have been chosen to represent the fault tree structure provide adequate information from which the desired pattern can be formulated. It is felt that the analysis of these characteristics is the next step forward.

### 5.8.2 Reviewing Relationship Between Characteristics and Scheme Choices

The variable ordering problem is quite complex and it is not obvious from just looking at a fault tree what the best ordering scheme should be. If this were the case then the variable ordering dilemma would not exist. By examining the characteristics themselves and their influence individually on the scheme preference it was made apparent that a number of inter-linked factors must contribute to the ordering problem. As can be seen with all the scatter plots (shown in figures 5.5-5.7) there is not a clear set of numbers for each characteristic which link it directly to the scheme option.

The scatter plot comparing the top gate type to the scheme option was trivial and therefore was not drawn, such that a tree starting with either an AND or OR gate could have the same best scheme option. Thus, it must be the interaction of a number of factors that ultimately will affect which scheme is best. Therefore, to mimic the complexity of the problem, more characteristics are added to the input list.


Figure 5.5: Comparison of Percentage of AND gates and Scheme Option


Figure 5.6: Comparison of Percentage of Different Events Repeated and Scheme Option


Figure 5.7: Comparison of Percentage of Total Events Repeated and Scheme Option

### 5.8.3 Changing The Characteristics

Both the points of the previous research are taken onboard and utilised in the future stages of research. Firstly two additional characteristics are added to the list of four fault tree characteristics previously being used. These are: the number of levels in the tree, which helps to model the depth of the tree; and the number of inputs to the top gate, which helps to model the breadth of the tree. Both of these characteristics show the size and magnitude of the tree because as the number of levels and inputs from the top gate increase the tree is ultimately growing in size, thus complexity. On review of the training data the number of levels were grouped accordingly and the coding for the number of levels in the tree involved 2 binary bits:

- $\quad 1-3$ levels $=00$
- 4-6 levels $=01$
- 7-9 levels $=10$
- $>9$ levels $=11$

For the number of inputs to the top gate, again only two bits were required. The number of events from the top gate must be greater than two otherwise the tree has not been constructed properly. On review most of the trees within the sample set had a number of inputs that were reasonably low. Thus, only four subclasses were chosen. The fault trees within the population as a whole may have a greater number of inputs to the top gate than four, which is taken care of by the last category in the coding. Hence, the coding took the following format:

- 2 events $=00$
- 3 events $=01$
- 4 events $=10$
- $>4$ events $=11$

On further examination of the fault trees within the training and test data sets it was apparent that the subgroups of coding for the 'percentage of different events that were repeated' and for the 'percentage of total events repeated' was not an accurate representation of the data. Most of the percentages were below thirty percent so the coding was split up differently. Those values below thirty were mapped directly on to their binary equivalent, and those greater than thirty percent were mapped on to the binary coding 11111. The reasoning behind this was to separate the clustered values which may help to differentiate between the scheme choices. The new coding for the two characteristics is summarised as

$$
\begin{aligned}
& \text { - } \quad 0-30 \%=00000-11110 . \\
& \text { - }>30 \%=11111 .
\end{aligned}
$$

Now each classifier representing the fault tree characteristics comprises of 19 bits and six bits for the scheme preferences, a total of 25 bits.

Before increasing the number of random rules generated the new characteristics are tested on the previous model set up. This should provide confirmation as to the importance of a large initial population of rules. Therefore, the genetic algorithm parameters remain the same. Still the number of matches produced was poor, which is to be expected when now the length of the string to be matched is longer than in the first trials. To reduce the number of predictive matches attempting to be made the test data set was reduced to ten, however this had no effect.

Having confirmed that the number of rules needs to be increased, the determining factor is by how much. The minimum possible is desired to make the system as efficient as possible. Thus, successive increments are made until a feasible set up is achieved. The next trial using the new set of six characteristics, involved increasing the random rules initially selected to 100 , with again ten in the test set. Still the number of matches was poor, thus it is concluded that the number of rules in the set to start with is influential in the matching process.

Incrementing the rule set to 500 , and again testing with ten fault trees, the results predicted by the classifier system on five different trials can be seen in table 5.9.

| Trial_, Matches/10 | Seedictions/10 | Mat |  |
| :---: | :---: | :---: | :---: |
| 1 | $10 / 2 / 3$ | 4 | 8 |
| 2 | $20 / 4 / 9$ | 1 | 5 |
| 3 | $476 / 5 / 3$ | 1 | 7 |
| 4 | $34 / 7 / 2$ | 1 | 2 |
| 5 | $26 / 5 / 8$ | 1 | 5 |

Table 5.9: Results of Five Trials of Classifier System, With 500 Initial Random Rules

The number of matches had increased, although the performance was poor. The number of matches still needs to be increased. Without getting all twenty trees matched the performance is useless for a new input. However, firstly the classifier system is tested on the original twenty fault tree structures. Only three trials were carried out (shown in table 5.10) as it was clear that performance was the same. Currently the model is not trained well enough to make predictions on previously unseen data.

| Trial | Seed | Predictions $/ 20$ | Matches $/ 20$, |
| :---: | :---: | :---: | :---: |
| 1 | $204 / 5 / 5$ | 2 | 8 |
| 2 | $25 / 3 / 3$ | 2 | 6 |
| 3 | $28 / 6 / 7$ | 2 | 5 |

Table 5.10: Results From Testing Same Classifier System as Above on Twenty Test Trees

To increase the pattern recognition potential the rule set needed to be increased further. The random rule set was increased to 750 . Initial results are shown in table 5.11.

| Trial | Seed | Predictions $/ 20$ | Matches $/ 20$ |
| :---: | :---: | :---: | :---: |
| 1 | $267 / 4 / 3$ | 5 | 8 |
| 2 | $33 / 3 / 3$ | 3 | 7 |

Table 5.11: Classifier System Results Using 750 Initial Random Rules

The performance has increased slightly as indicated by the two trials carried out, although different seeds were entered which may have been more suited to the random rule set, for example, the results may have been similar for this seed combination when tested on 500 initial random rules. However, the predictions are not the primary concern at this stage, but the number of matches. These matching figures have not increased significantly.

The program currently loops round for up to three matches, i.e. if a match occurs the message can be posted and the matching process begin again for up to three times. At the end of three rotations the process is terminated and the winning classifier at that stage is termed the winner. To see the effect of the matching and bidding process the view was to alter the number of loops performed to see the affect on performance. It is possible that the number of loops currently set is limiting performance, i.e. the correct rules to be found, alternatively the number of loops could be in excess of what is required, making the process inefficient.

The first trial had the following characteristics:

- 1 loop of matching process before winning classifier found
- Reward of 850 units given to classifiers if correct classifier selected
- Life tax for existing classifiers in A.O.C system set to 0
- . 1000 iterations before training stopped
- G.A parameters set as in first trial
- Predictions made on twenty trees.
- 200 random rules in initial rule set

The results obtained are summarised in table 5.12. The predictive performance is poor, and also the number of matches is poor. It is thought that having only one loop is limiting the selection of the rules in training, and ultimately affecting the predictive performance.

| Trial | Seed | Predictions $/ 20$ | Matches $/ 20$ |
| :---: | :---: | :---: | :---: |
| 1 | $23 / 3 / 3$ | 0 | 4 |
| 2 | $36 / 3 / 3$ | 1 | 1 |
| 3 | $101 / 3 / 3$ | 1 | 3 |
| 4 | $265 / 3 / 3$ | 1 | 7 |
| 5 | $492 / 3 / 4$ | 0 | 2 |

Table 5.12: Results Using One Loop in Apportionment of Credit System

The next stage is to increase the number of matching loops and observe the effects. Table 5.13 below compares the results (predictions and matches) for loop iterations of 2, 3 and 5 respectively.

| Trial | Seed | Preds: <br> 2 loops | Matches: 2 loops | Preds: <br> 3 loops | Matches: 3 loops | Preds: | Matches: 5 loops |
| :---: | :---: | :---: | :---: | :---: | :---: | :---: | :---: |
| 1 | 23/3/3 | 0 | 2 | 0 | 2 | 0 | 1 |
| 2 | 36/3/3 | 2 | 2 | 1 | 1 | 1 | 1 |
| 3 | 101/3/3 | 0 | 1 | 0 | 1 | 0 | 1 |
| 4 | 265/3/3 | 1 | 6 | 1 | 3 | 1 | 4 |
| 5 | 492/3/4 | 1 | 1 | 0 | 4 | 0 | 3 |

Table 5.13: Comparative Results Using 2, 3 or 5 Loops in Matching Process

From table 5.13 it is evident that performance does not seem to be improving despite increasing the number of loops. If within the classifier set there is no potential for
matching then increasing the number of loops is not going to be of any benefit, hence the number of random classifiers is increased again.

Using 500 random classifiers in the initial classifier store, and a reward of 850, a life tax of zero and five levels in the matching routine the system was retested. The results are given in table 5.14.

| Trial | Seed | Predictions $/ 20$ | Matches/20 |
| :---: | :---: | :---: | :---: |
| 1 | $23 / 3 / 3$ | 2 | 7 |
| 2 | $36 / 3 / 3$ | 3 | 11 |
| 3 | $101 / 3 / 3$ | 4 | 11 |
| 4 | $265 / 3 / 3$ | 0 | 5 |
| 5 | $492 / 3 / 4$ | 1 | 6 |

Table 5.14: Results When Random Classifiers Increased to 500

The number of matches has increased, and the performance has increased slightly, which probably is a reflection on the number of matches. Before, the research continues utilising five loops for matching it is important to determine if this many is necessary. Hence, for the same seeds the number of iterations is reduced to 3 . The results can be observed in table 5.15.

| Trial | Seed | Predictions $/ 20$ | Matches /20 |
| :---: | :---: | :---: | :---: |
| 1 | $23 / 3 / 3$ | 2 | 7 |
| 2 | $36 / 3 / 3$ | 2 | 10 |
| 3 | $101 / 3 / 3$ | 5 | 10 |
| 4 | $265 / 3 / 3$ | 1 | 2 |
| 5 | $492 / 3 / 4$ | 1 | 6 |

Table 5.15: Results For Three Loops and 500 Classifiers

The performance between these two trials is very similar. Using three iterative loops for the matching and bidding process as opposed to five seems as feasible. It is obvious that performance can only improve if the number of matches is improved. Currently the genetics based machine learner is not fully trained before being used to predict the population trends. The first place to start is to increase the number of classifiers within the classifier store. This is increased to 1000 . Using the same seeds as before to continue the comparisons, the results are given in table 5.16.

| Trial | Seed | Predictions $/ 20$ | Matches $/ 20$ |
| :---: | :---: | :---: | :---: |
| 1 | $23 / 3 / 3$ | 5 | 16 |
| 2 | $36 / 3 / 3$ | 3 | 10 |
| 3 | $101 / 3 / 3$ | 6 | 12 |
| 4 | $265 / 3 / 3$ | 3 | 17 |
| 5 | $492 / 3 / 4$ | 1 | 11 |

Table 5.16: Results Using 1000 Initial Random Rules

There is a great improvement in the number of matches. The performance has improved although by no means to a desirable level.

### 5.8.4 Changing The Proportion of Classifiers Acted Upon By G.A

Currently the proportion of the random rules which are selected to be manipulated by the genetic algorithm is set to 0.4 . Therefore in the last example, 600 of the classifiers remain unaltered and the genetic operators within the genetic algorithm act upon 400. If however, the classifiers which remain unaltered do not reflect the rules required then increasing the proportion selected may help to improve the rules and help with matches. Thus, the next stage of research focussed on altering the proportion of the rule set selected for genetic action. The selection process is carried out by a biased roulette wheel approach, hence the likelihood of classifiers being selected is based on their strength values.

For all the trials, the number of initial rules was set to 1000 . The number of loops for matching was set to 3 . The initial probability for the proportion of classifiers to be selected to go into the genetic algorithm was 0.9 , i.e. 900 selected for genetic algorithm action and 100 kept. Results using the first seed, $23 / 3 / 3$, produced only 1 correct prediction, with only ten matches. This is worse than the result in table 5.16 hence it was immediately suggested to reduce the proportion selected. Using 0.75 as the proportion to be selected, the following results (given in table 5.17) were found:

| Trial | Seed | Predictions/20 | Matches/20\% |
| :---: | :---: | :---: | :---: |
| 1 | $23 / 3 / 3$ | 2 | 12 |
| 2 | $36 / 3 / 3$ | 4 | 15 |

Table 5.17: Predictive Results Using 0.9 as Proportion of Classifiers Selected

The results are no better for the initial trials, so the proportion is reduced again. Using the proportion selected with probability of 0.6 the results improved slightly, although both the number of matches and the predictions are not up to the required standard to say that a good model has been generated. The results are shown in table 5.18.

| Trial | Seed $/$ Predictions/20 | Matches/20 |  |
| :---: | :---: | :---: | :---: |
|  | $23 / 3 / 3$ | 5 | 14 |
|  | $36 / 3 / 3$ | 5 | 14 |
| 3 | $101 / 3 / 3$ | 2 | 15 |
| 4 | $265 / 3 / 3$ | 5 | 13 |
| 5 | $492 / 3 / 4$ | 2 | 17 |

Table 5.18: Proportion Selected Set to 0.6

Using this proportion has achieved an overall correct number of predictions of 19 for the five trials combined, whereas using the 0.4 probability produced 18 correct responses. In
both instances these figures are very poor. Still the number of matches needs to be increased so increment initial population of rules to 10,000 .

Using this gained 20 matches for the first two trials, with predictions of 3 and 5 out of 20 respectively, using a proportion to select of 0.9 . Although the performance is poor, it is better than when 0.9 was used in previous trials (using 1000 starting population). So now the correct number of matches is being established but the selected classifiers are incorrect.

It is clear that the rules govern the number of matches found with the predictive trees and ultimately the possible number of correct predictions. The approach now taken in the research is the alteration in how the random rule set is altered after each training cycle. The new code is called gbml2.c, which has only minor modifications to that of the first code. The philosophy here it to keep 1000 of the best classifiers, generate 1000 from these using the genetic algorithm and randomly generate another 8000 . It is hoped with the additional random regeneration that a larger proportion of possible rules will be tested.

Initial results using the $23 / 3 / 3$ seed found 19 matches and 5 correct predictions. Using the $36 / 3 / 3$ seed again found 19 matches with the classifier store, but again a low predictive performance of 3 . Still the accuracy of the predictions are poor. The selection and regeneration method was altered to keep 500 of the best, generate 500 from these using the genetic algorithm and to randomly regenerate the remaining 9000. However, results still produced predictions of between 4 and 5 out of 20 for five trials.

### 5.8.5 Adding More Characteristics As Inputs

As the number of matches associated with using 10000 initial random rules has increased but the performance has not, questions need to be asked as to whether the pattern is too complex to find. One aspect that will influence how clear the pattern is, is whether the characteristics chosen to represent the fault tree are important factors in distinguishing
between the best scheme for that tree and other schemes. If they are not finding the pattern is going to be an impossible mission. As it is not known which characteristics may separate trees into scheme categories it is difficult to know which characteristics to use. To increase the likelihood of including some of the distinguishing characteristics within the input classifier, the number of characteristics used is increased to incorporate five more. The coding for characteristic five, the number of levels in the tree has been altered and now is represented as:

1) Characteristic 5: Number of Levels in The Tree - the coding was extended to incorporate groupings for a larger number of levels. The groupings are shown in table 5.19.

| $0-4$ levels | 00 |
| :---: | :---: |
| $5-8$ levels | 01 |
| $9-12$ levels | 10 |
| $>12$ levels | 11 |

Table 5.19: Number of Levels in Tree and Corresponding Binary Coding

The additional characteristics and their coding are:
2) Characteristic 7: Number of Basic Events - the number of basic events in the tree was one of the most variable characteristics and possibly one of the most influential in the ordering process. A larger number of groupings were incorporated involving the use of 3 binary bits as shown in table 5.20.

| Number of Basic <br> Events | Binary Coding |
| :---: | :---: |
| $2-5$ | 000 |
| $6-10$ | 001 |
| $11-15$ | 010 |
| $16-20$ | 011 |
| $21-25$ | 100 |
| $26-30$ | 101 |
| $31-35$ | 110 |
| $>35$ | 111 |

Table 5.20: Basic Event Binary Coding
3) Characteristic 8: Maximum Number of Gates in Any Level - the number of gates were usually below seven and thus it was felt that these needed to be individually identified. The number of gates increased the number of levels within the tree and hence the possible effect on the ordering especially when subtrees were used. So using three bits the coding was: 1 gate $=000 ; 2$ gates $=001 ; 3$ gates $=010 ; 4$ gates $=$ $011 ; 5$ gates $=100 ; 6$ gates $=101 ; 7$ gates $=110 ;$ and $>7$ gates were represented by the coding 111.
4) Characteristics 9 and 10: Number of Gates with Just Event or Gate Only Inputs both of these characteristics were coded in the same manner. The coding is shown in table 5.21.

| Number of Gates | Binary Coding |
| :---: | :---: |
| 0 | 000 |
| 1 | 001 |
| 2 | 010 |
| 3 | 011 |
| 4 | 100 |
| 5 | 101 |
| 6 | 110 |
| $>6$ | 111 |

Table 5.21: Binary Coding For Number of Gates
5) Characteristic 11: Highest Number of Repeated Events - as the majority of the repeated events were only repeated a few times the coding required just two binary bits, whereby 0 repeats (hence not a repeated event) was represented by $00 ; 2$ repeats equalled 01 ; three repeats $=10$; and an event that was repeated more than three times was given the coding 11 .

Therefore, the characteristic coding can be summarised as shown in figure 5.5:


Figure 5.5: Overall Characteristic Coding For Classifier

The breakdown of the additional characteristics for each fault tree is given in Appendix IV. With the increase in characteristics as inputs, the size of each classifier string has
lengthened. With this increase also comes the added difficulty of finding the necessary matches within the classifier store.

To begin the search for an appropriate classifier system using the eleven characteristics the initial random classifier store was set to 1000 . Results using an initial strength value for each rule of 500 units, 850 units as reward, selecting 0.4 as the proportion of the rules to be acted upon by the genetic algorithm, a crossover probability of 1.0, and a mutation probability of 0.0002 , were poor. Again arose the problem of matching. For ten trials the average number of matches was 2.4 . With the extra difficulty in matching thirty nine bits of information it was expected that a considerably larger initial rule set would be required to attempt to match all possible population data. The same G.A parameters, reward values and classifier strengths were used, but this time the classifier system had 10,000 random rules. The training period was extended to 10,000 iterations. The results on ten random starts are given in table 5.22.

| Trial Predictions/20 | Seed | Matches/20 |  |
| :---: | :---: | :---: | :---: |
| 1 | $23 / 3 / 3$ | 5 | 14 |
| 2 | $36 / 3 / 3$ | 7 | 19 |
| 3 | $101 / 3 / 3$ | 3 | 15 |
| 4 | $265 / 3 / 3$ | 8 | 20 |
| 5 | $492 / 3 / 4$ | 6 | 17 |
| 6 | $52 / 2 / 7$ | 6 | 14 |
| 7 | $101 / 5 / 2$ | 5 | 15 |
| 8 | $35 / 4 / 6$ | 9 | 17 |
| 9 | $61 / 8 / 4$ | 8 | 19 |
| 10 | $120 / 30 / 5$ | 6 | 17 |

Table 5.22: Results Using 10000 Random Rules and 10000 Iterations For Training

The increase in the random population of rules has lead to an increase in the number of matches between the test data and the rule set. However, there is not sufficient
consistency in generating twenty matches, which without will limit the number of possible correct predictions of best scheme choice. A prevalent fact is still the poor predictive capacity, which remains at a best of 9/20.

As the possibility of finding a classifier system with the appropriate set up to model this ordering problem with desired accuracy looks unlikely a slightly new approach was considered, i.e. considering each scheme option separately.

### 5.8.6 Training Models For Each Scheme Option

The next progression to finding a classifier system to model the variable ordering problem focussed on training a classifier model for each scheme option, rather than using all six schemes as outputs to one model. Therefore, six individual models were generated, one for each scheme option. Starting with scheme one, if the model didn't predict scheme one as the chosen scheme preference then the fault tree input data was tried against the scheme two model etc. On finding a correct solution the process would stop. In this manner only one response (scheme choice) would be found, hopefully alleviating the problems with the single model where often the correct option was combined with incorrect ones. The classifier models were tested in ascending order of complexity, thus the first correct choice made would involve the simplest method of ordering. Scheme one was used as the default scheme option if no schemes were chosen.

Using this methodology meant that each classifier had a string length of thirty four bits. For each model, the same data set was used, but with scheme outputs altered. The test data set was the same as in previous tests. All models were initialised with 10,000 random rules, and the training was carried out for 10,000 iterations. The genetic algorithm was used to change 0.4 of the rule set, with crossover probability of 1 and mutation probability of 0.0002 . The rules contained in each system were given a strength value of 500 units, as in previous tests and a reward of 850 was used when a correct prediction was made during the training phase.

Still the problem of matching existed, and the predictions on all twenty trials carried out ranged between 4 and 8 . Unfortunately these results are no better than that obtained with the classifier system modelling all six scheme models.

With the added complexity of generating six individual systems which hold the correct pattern within each, and the poor predictive results that were found, it is thought more beneficial for further tests to be carried out using the original single model than try to generate six separate models.

### 5.9 Results and Conclusions

The results gained using the classifier model approach have been variable. The results for the smaller fault tree structures have been more convincing but this is believed to be due to a number of scheme options producing equally good BDDs. Hence, the output required is less specific than for larger structures. Complications arise with the larger fault trees and predictive power lessens. However, it is expected that as the process can be applied to small trees then a change in the characteristics chosen as inputs to take better account of larger structures would promote the same pattern recognition potential to be unleashed for these more complicated fault trees.

As the aim of the research is to produce a model that will take any given fault tree structure and generate the desired or most optimal ordering heuristic the performance of any of the classifier models generated does not produce the require accuracy. The best classifier system produced has a predictive capability of 9 out of 20 correct predictions.

The classifier system had the following set parameters:

- Classifier of 39 bits in length.
- Initial strength value for each rule of 500 units.
- 850 units given as reward.
- 0.4 proportion of the rules selected to be acted upon by the genetic algorithm.
- Crossover probability of 1.0 .
- Mutation probability of 0.0002 .
- 10,000 random rules in initial population.
- 10,000 iterations for training.
- Random Seed 35/4/6.

Prediction results though not brilliant do illustrate that the classifier approach has the potential to be trained to predict the best of alternative ordering schemes to yield an efficient BDD representation as shown with smaller fault trees. This also applies to the model taking each scheme option in turn. However, finding the correct model structure is the main concern. Further improvements are expected by continuing the trial and error type approach to change the infinite number of variable parameters contained within the system. This process would be very time consuming and the end result is expected to be some what sub-optimal with a predictive capability which does not match that required to use this approach in a commercial package. It is felt that the most influential parameter to first change is the fault tree characteristics themselves. It is clear that the factors chosen to represent the fault tree structure need to be scrutinised for their importance in describing the tree.

### 5.10 Deficiencies of the Classifier Method

- Before using the method the best inputs to use to find the pattern between a fault tree and its optimal ordering heuristic is not known, hence the pattern may not even be evident in the data used for training.
- Also the preferred coding for the imputs and outputs is not known. Using numerical values increases the length of each string required and complicates the task, however coding in subgroups may be confusing the pattern if the subgroups are incorrect.
- In conjunction with the unknown inputs and best coding configuration is the large number of variable parameters to chose within the classifier system itself, namely:
- Number of training iterations.
- Number of matches in classifier store before winner found.
- Number of initial classifiers in store (rules).
- How much of population to select to apply genetic algorithm too.
- Optimal settings for genetic algorithm operators.
- As the predictions are made in terms of 1's or 0's, a number of schemes can be found as the best. In some cases the answer is contained within a set of solutions and thus is incorrect. However, this is a reflection of incorrect rules.
- The method by which the schemes are coded does not provide the system with a great deal of information. A scheme is either classified as good or bad, when in reality the difference between two schemes may be slight i.e. if one scheme produced 31 BDD nodes and another 32 , the first would receive a value of 1 the latter 0 . In reality, it makes no difference if the second scheme is chosen rather than the first.


### 5.11 Summary

- The classifier system which incorporates a genetic algorithm does appear to have some potential to select an ordering scheme option depending on the characteristics of a given fault tree.
- The results for the best model architecture are variable, being more promising for smaller fault tree structures.
- Best result $=9$ out of 20 correct predictions, using 39 bit model.
- Results using models for separate scheme choices yielded no better results and were more time consuming to construct and use than the single model incorporating all six scheme options.
- As the larger fault tree structures are more complex and with performance being worse on these types of structures it is felt that the characteristics used to model
the fault tree do not adequately reflect those required for the relationship between structure and scheme.
- Further investigation into the characteristics used to describe the fault tree structure and how they relate to the best ordering for that tree needs to be undertaken.
- Adoption of another, perhaps, better suited pattern recognition approach to the problem may produce significantly better results.
- Whilst showing some promise the results using the classifier method are not totally convincing and so it was decided to investigate the capability of alternative approaches to predict an ordering scheme.


## 6 Additional Pattern Recognition Techniques Application of Neural Networks

### 6.1 Neural Networks - General Overview

### 6.1.1 Introduction - What is a Neural Network Model?

The neural network is another method of identifying patterns. It can be regarded as a particular choice for a set of functions that map a set of input variables to a set of output variables. The importance of neural networks is that they offer a very powerful and very general framework for representing non-linear mappings from several input variables to several output variables, where the form of the mapping is governed by a number of adjustable parameters. These adjustable parameters help to guide the mapping to generate the correct pattern and ultimate response. The process of determining the values for these parameters on the basis of the data set is called learning or training, and for this reason the data set of examples is generally referred to as a training set. The neural network model once trained on a set of examples can then be used to predict the output of a set of novel inputs using the parameters generated in training.

### 6.1.2 Basis of Neural Network Principle - Polynomial Curve Fitting

Many of the important issues concerning the application of neural networks can be introduced in the simpler context of polynomial curve fitting. This can be illustrated by the problem of fitting a polynomial to a set of data points by the technique of minimising an error function. For example, consider the $M^{\text {th }}$ order polynomial given by:

$$
\begin{align*}
& y=w_{0}+w_{1} \cdot x+w_{2} x^{2}+\cdots+w_{M} x^{M} \\
& \text { i.e. } y=\sum_{i=0}^{M} w_{i} x^{i}
\end{align*}
$$

This can be regarded as a non-linear mapping which takes $x$ as an input and produces $y$ as an output. The coefficients, $\underline{w}$, applied to each input can be viewed as one of the adjustable parameters of the network mapping.

In order to find suitable values for the coefficients in the polynomial, it is convenient to consider the error for each data point $x^{i}$ (where $i=1, \ldots, M$ ) between the desired output $t^{t}$ for a particular input $x^{i}$, and the corresponding value predicted by the polynomial function given by $y\left(x^{i} ; \underline{w}\right)$. Standard curve-fitting procedures involve minimising the square of this error, summed over all the data points, given by

$$
E=\frac{1}{2} \sum_{i=1}^{M}\left\{y\left(x^{i} ; \underline{w}\right)-t^{i}\right\}^{2}
$$

$E$ is regarded as being a function of the coefficient vector, $\underline{w}$, so the polynomial can be fitted to the data by choosing a value for $\underline{w}$, denoted by $\underline{w}^{*}$, which minimises $E$.

### 6.1.3 General Learning Techniques

There are various learning techniques that can be used to train the neural network. The minimisation of an error function (equation 6.2), which involves target values for the network outputs is called supervised learning since for each input pattern the value of the desired output is specified. Usually the target response is known, like in the variable ordering problem where the ordering scheme choice for a particular tree is predetermined and the training performance is based on this target. A second form of learning in neural networks, called unsupervised learning, does not involve the use of target data. Instead of learning an input-output mapping the goal may be to model the probability distribution
of the input data, or to discover clusters or other structures in the data. There is a third form of learning, called reinforcement learning, where again actual desired values are not given but information is supplied as to whether the network outputs are good or bad. Within this research supervised learning techniques have been employed.

### 6.1.4 Types of Network

There are a number of different types of neural network, all with the same principle of generating a mapping between a set of inputs and a set of outputs, but each have a slightly different approach to tackling the problem. The simplest type of neural network architecture is the single layer neural network. Networks with more complicated architectures which are used to deal with more complex problems are the multi-layer perceptron and radial basis function. The single layer and multi-layer perceptron neural network models are discussed in this chapter, with the latter being applied to the ordering problem.

### 6.2 Single Layer Neural Networks

### 6.2.1 Introduction

The simplest form of a neural network is one that has just a layer of inputs and a layer of outputs. Considering the case where the input layer consists of a number of nodes all of which are related to just one output node, then this can be modelled as shown in figure 6.1. The inputs $x_{1}, \ldots . . . ., x_{d}$ are shown as circles, which are connected by weights, $w_{l}, \ldots .$, $w_{d}$, to the output $Y 1$. The weights govern the functional mapping of the network. A bias node is added and is represented as a weight from an extra input $x_{0}$ which is permanently set to 1 . This node acts like adding a constant to the equation.


Figure 6.1: A Single Layer Neural Network With One Output Unit Y1

Extending the theory to the more complicated task of problems with several categories of classification, this can be expressed in terms of the neural network diagram shown in figure 6.2.

OUTPUTS


Figure 6.2: Single Layer Network With Many Outputs.

Each output $y_{k}$ is associated with a weight $w_{k i}$, connecting input node $i$ to output node $k$, and a bias $w_{k 0}$ connecting to each output node. The network's outputs can be expressed as a simple linear combination of the inputs ( $d$ in total), with the bias included forming the single summation written as:

$$
y_{k}=\sum_{i=0}^{d} w_{k i} \cdot x_{i}
$$

Usually, a non-linear activation function is applied to the linear combination as the pattern is more complex than being represented by a linear function.

Single layer networks correspond to a very narrow class of possible functional mappings, and in many practical situations may not represent the optimal choice. Single layer networks provide many useful insights into the properties of more complex multi-layer networks.

### 6.3 Multi-Layer Perceptrons (MLPs)

### 6.3.1 Introduction

The more complicated extension of the single layer network is the multi-layer feedforward network or multi-layer perceptron. The network is made up of a series of layers with connections running from every unit in one layer to every unit in the next layer. These connections are known as the weights (as in the single layer network) and they control the influence each node has on propagating the intermediate outcome to the output nodes. Typically the network consists of a set of input nodes that constitute the input layer, one or more hidden layers of nodes, and an output layer of nodes (as shown in figure 6.3).

The network has a larger number of connections than the single layer network allowing more diversity in the non-linear mapping it can model. Thus, it can tackle a much wider range of problems.


Figure 6.3: Diagram Representing a Multi-layer Perceptron

The fundamental operation of the network is to take the inputs and by a training process determine weight values for the connections between the nodes. In the prediction phase these weight values then determine the path through the network and ultimately determine the output response for the given input problem.

During the training phase multi-layer perceptrons commonly use an algorithm known as the error back-propagation algorithm. The algorithmic process consists of two possible passes through the different layers of the network: a forward pass and a backward pass. In the forward pass, an input vector is applied to the input nodes of the network, and subsequent outcomes are evaluated layer by layer. Finally, a set of outputs is produced as the actual response of the network. During the forward pass the weights of the network are fixed. During the backward pass, on the other hand, the weights are adjusted in accordance with an error-correction rule or delta rule. Specifically, the actual response of the network is subtracted from a desired (target) response to produce an error signal.

This error signal is then propagated backward through the network, against the direction of weight connections, hence the name "error back-propagation". The weights are adjusted so as to make the actual response of the network move closer to the desired response.

During the training phase the weight values of the connections between the nodes are established. The training phase involves a number of cycles whereby on each cycle the search for better weights is directed to a new area as defined by a specified search parameter. When the error has been reduced sufficiently it is these weights that are used as fixed values in the predictive phase. How well the network has been trained and models the problem will be reflected in the prediction of new input data. If the network has been trained well it will generalise well to new data and a correct response should be predicted.

### 6.4 Application to The Ordering Problem

In recent years neural computing has emerged as a practical technology ${ }^{[B i s 95]}$ with successful applications in many fields. The majority of these applications are concerned with pattern recognition problems and make use of feedforward network architectures. The task of identifying a suitable basic event ordering scheme for a given fault tree structure can be formulated as a pattern recognition problem. Whilst orderings derived by the classifier approach produced limited success, it is felt that a neural network with a more solid theoretical basis may have more to offer. This chapter investigates the potential to obtain an efficient BDD ordering by using a multi-layer feedforward neural network.

The work presented in this chapter uses the pattern recognition approach of neural networks to select the 'optimal' ordering scheme from a set of alternatives. A multi-layer feedforward network is used with an error back-propagation method to adjust the network weights. A network has been trained on a specially constructed set of features representing the fault tree structures and tested using different data to measure the
success of the approach. A set of six alternative ordering schemes has been identified (as described in section 6.5). The objective, as when applying the previous machine learning technique, is to predict the best of the six alternative ordering permutations which will yield the most efficient BDD for the given fault tree structure. The research in this area is published in two papers ${ }^{[B A n 001, ~ B A n 002]}$.

### 6.5 Inputs and Outputs For MLP Neural Network Model

### 6.5.1 Preferences For The Variable Ordering Schemes

The objective of the initial neural network model will be to establish the capability of the network to select the best ordering heuristic from a restricted group of alternatives for a given fault tree. In this study the 6 different potential structured ordering schemes known as:

- Top-down, left-right approach;
- Modified top-down, left-right approach;
- Depth-first approach;
- Modified depth-first approach;
- Priority depth-first approach;
- Modified priority depth-first approach.
will again be used, as the information necessary for the training process is available from the work performed using the machine learning approach.


### 6.5.2 Fault Tree Characteristics and Scheme Preference Coding

The difficulty in the neural network approach is in correctly modelling the problem. Some fault tree attributes have been selected to characterise the structure as in previous work. The input layer of the neural network represents the eleven characteristics which
were selected to represent the fault tree structure, and the output layer of nodes within the network are used to model the six scheme preferences.

To summarise, the characteristics that have been chosen to represent the fault tree structure are:

- Percentage of AND gates;
- Percentage of different events repeated;
- Percentage of total events repeated;
- Top gate type;
- Number of inputs to top gate;
- Number of levels in tree;
- Number of basic events;
- Maximum number of gates in any level;
- Number of gates with gate inputs only;
- Number of gates with event inputs only;
- and highest multiple of a repeated event.

With the neural network approach the input and output variables do not need to be categorised in binary, as in the machine learning approach, instead numerical values can be applied. Using numerical values for the output values, rather than binary good or bad, will give an indication of how good a scheme is in relation to the best. This should give the neural network approach a better chance of locating the domain boundaries for each output than the classifier approach.

### 6.6 Training and Test Sets

Fault tree structures utilised for the training and testing of the multi-layer perceptron were the same as those used in the classifier approach. To promote an efficient functioning of the network it is best to scale the outputs in accordance with the non-linear activation function used. The format to specify the inputs and outputs is discussed in later sections.

To evaluate the performance of the neural network a test set of data was produced with different tree structures and known best ordering schemes. The performance of the network was evaluated by the number of correct scheme preferences predicted by the network. The same test set of trees was used in the initial MLP studies so performance could be compared with the classifier approach.

### 6.7 Creating a Neural Network Program

### 6.7.1 General Description of Program

During the research the neural network program net_mlp.c was created using the C programming language. The breakdown of the program is shown in figure 6.4. The following model characteristics are implemented:

1. Multi-layer perceptron approach.
2. Batch method of training.
3. Gradient descent optimisation technique.
4. Minimum error and maximum number of iterations for stopping criterion.
5. Forward and Backward propagation techniques.
6. Mean sum-of-squared errors term.
7. Sigmoidal activation function.

The program has two phases of execution, one for training purposes and one for predictive purposes. This is controlled within the main subroutine of the program.


Figure 6.4: Neural Network Program Outline

### 6.7.2 Details of The Multi-layer Perceptron Approach

The network constructed to model the problem is such that it has a number of layers, each with a number of individual nodes. The input layer contains eleven nodes and the output six. The layers and nodes in between are to be determined through a training procedure. As known target schemes are used a form of supervised learning is carried out. The program applies the principles of back propagation within the perceptron approach.

Each subroutine of the program is explained as follows:

## INITIALISATION:

This reads in the input data files. If the training mode is chosen then the training data set of examples are read in consisting of 181 fault trees. Otherwise in the predictive phase the test set of twenty trees is read in.

## EXTRACTDATA:

This routine creates the input and target data arrays. In the training phase the inputs are propagated through the network to gain the appropriate weights and the target vector is used to establish the error-correction rule. In the predictive phase the input data array contains the new information for previously unseen trees to be given a scheme choice for ordering and the target array is used to compare the results generated by the model.

## CREATE_NET:

This creates the basic architecture of the network. Three parameters need to be found, being the weights, the number of hidden layers, and associated nodes.

## 1. Weight Initialisation

Before any training of a network can begin the weights (adjustable parameters) of the network need to be initialised. The customary practice is to set all the weights of the network to random numbers that are uniformly distributed inside a small range of values. The wrong choice of initial weights can lead to a phenomenon known as premature saturation ${ }^{[L O K 91]}$. This phenomenon refers to a situation where the sum of squared errors remains almost constant for some period of time during the learning process, that is there is no change in the predictions made by the network. Incorrectly chosen weights limit the pattern recognition potential of the network and without changes the performance is very poor. In this research the weights are set randomly between the values of $+/-0.5$.

## 2. Hidden Layers and Associated Nodes

For each network used to solve a particular problem there is an optimal structure. Often for multi-layer networks the number of hidden nodes within the structure can mean the difference between success and failure. Finding the optimum number of hidden nodes and layers is solely dependent on the problem being modelled. In general, the smallest number of layers and nodes in each layer is the best solution. When using more than one hidden layer training often slows dramatically. This is due to the fact that the number of false minima increases. Using too few nodes within the whole network will starve the network of the resources it needs to solve the problem. Using too many will increase the training time, perhaps so much that it becomes impossible to train it adequately in a reasonable period of time. One rough guideline for choosing the number of hidden nodes in many problems is the geometric pyramid rule ${ }^{[\text {Mas93] }}$ in which for a three layer network with $n$ input nodes and $m$ output nodes, the hidden layer would have $\sqrt{m . n}$ nodes. Even so, this rule may over or under estimate the number required depending on the network architecture. The best approach to finding the optimal number of hidden nodes is time-consuming and starts with "too small" a guess, for example two nodes. The network is trained and tested, and the performance recorded. The number of nodes is then increased and the model retrained and tested. The training and test procedure is repeated until the error is acceptably small, or no significant improvement is noted. This method is adopted within the net_mlp.c program that has been constructed.

NET_TRAIN:
This routine directs the training procedure. There are two common training methods, namely: pattern and batch modes of training. In the pattern mode of training weight updating is performed after the presentation of each training example. Consider a training data set consisting of $N$ training examples, the first example in the data set is presented to the network, and the sequence of forward and backward computations is performed, resulting in certain adjustments to the network weights. Then, the second example in the data set is presented, and the sequence of forward and backward
computations is repeated, resulting in further adjustments to the network weights. This process is continued until the $N^{\text {th }}$ example in the data set is accounted for.

Using the batch mode of back propagation means that weight updating is performed after the presentation of all training examples within the data set. Hence the forward pass is made for all training patterns and the sum of squared error function for the whole data set is established. Following this the backward phase is executed and the weight adjustment made. There is no way to know which method may be most appropriate so initially the batch method of learning is tried.

## FORWARD AND BACKWARD PHASES (NET_TRAIN1):

As a batch method of training is used, the routine net_trainl() executes the forward phase (routine net forward()) for each training pattern before the backward phase (routine net_backward()) is carried out and any weight changes made (net_weight()).

## 1. NET_FORWARD:

The first step in the learning process is a feed forward operation, which calculates the value of the output nodes from the input layer through the hidden layers to the output layer. If a model is used with $d$ input nodes, one hidden layer of $m$ nodes and an output layer of $k$ nodes then the following calculations would result for each training pattern $n$ :

1. Hidden layer node values, $v_{j}(n)$ :

The linear combination is established between the hidden node $j$ and weight connections $w_{j i}$ to each input $x_{i}(n)$. A non-linear activation function, $g()$, is applied to the linear combination that is calculated.

$$
v_{j}(n)=g\left(\sum_{i=0}^{d} w_{j i} x_{i}(n)\right)
$$

2. Output layer node values, $y_{k}(n)$ :

The linear combination is calculated between output node $k$ and weight connection $w_{k j}$ to each hidden node $v_{j}(n)$. An activation function $\mathrm{g}^{*}()$ is then applied to the result, this may be the same function as used in the hidden layer or different.

$$
y_{k}(n)=g^{*}\left(\sum_{j=0}^{m} w_{k j} \cdot v_{j}(n)\right)
$$

## 2. SIGMOID:

For most problems, the mapping between the inputs and the outputs is non-linear, except in simple cases. To introduce non-linearities within the function mapping the input variables to the output variables an activation function is applied. These activation functions are implemented across all the nodes at each hidden and output layer within the network. One activation function, represented by $g()$, is the sigmoidal function given by:

$$
g(a)=1 /(1+\exp (-a))
$$

where $a$ represents the output which is to be activated, usually the linear combination of the weights and connecting nodes. The output of the activation function is in the range $\{0,1\}$ as shown in figure 6.5. Hence, it is convenient for the outputs of the problem to be scaled in a similar range. This activation function is used within the program.


Figure 6.5: Sigmoidal Activation Function

## 3. NET_BACKWARD:

A. Calculating The Errors: The first step of the backward phase involves determining the error at the output nodes. The training data examples are propagated through the network, initially using the random weights to produce a result for each of the output nodes. Each of these outputs is then compared with the target responses to produce an error.

The error $e_{j}(n)$ is calculated for each output node $j$ for all $n$ training patterns, by taking the target response of each node $t_{\mathcal{J}}(n)$, and subtracting from it the response generated for that node by the network $y_{j}(n)$.

$$
e_{j}(n)=t_{j}(n)-y_{j}(n)
$$

As a batch mode of training is used, the squared error term is calculated for each training pattern, summed for all patterns, then divided by the total number of training patterns to give the error, $E$, for that cycle. This then determines whether any changes in the weight connections are required to move the response of the network toward the desired outcome. The error function used in the program thus takes the form:

$$
E=\frac{1}{2 N} \sum_{j=1}^{N} \sum_{i=1}^{M} e_{i}^{2}(j)
$$

where $M$ is the total number of output nodes, and $N$ is the total number of training patterns.
B. Propagating Errors Back Through Network: If the error is greater than a certain predetermined value then the weights would need to be altered using an optimisation technique. Different optimisation algorithms will perform best on different problems and it is therefore not possible to recommend a single universal optimisation algorithm. The aim of all the algorithms though is to optimise the search direction to locate the best
weight values to govern the connections in the network and ultimately reduce the error of the network to a minimum.

One of the simplest, most practical algorithms is that of gradient descent. To start the algorithm, some initial guess for the weights is chosen. Then the weights are iteratively updated such that the search is moved a short distance in the direction of the greatest rate of decrease of the error, i.e. in the direction of the negative gradient. Thus, to use the gradient descent algorithm the derivatives of the error function with respect to the weights need to be propagated back through the network.

Considering a general node $j$ in a feedforward network, as shown in figure 6.6.


Figure 6.6: General Node in Feedforward Network

The output of node $j$, for training pattern $n, a_{j}(n)$, is equivalent to the linear combination of the weights $w_{j i}(n)$ and the inputs $z_{i}(n)$, represented by equation 6.9. The outcome $z_{j}(n)$ of the activation of this linear combination, $a_{j}(n)$, is represented by equation 6.10 .

$$
a_{j}(n)=\sum_{i} w_{j i}(n) \cdot z_{i}(n)
$$

$$
z_{j}(n)=g\left(a_{j}(n)\right)
$$

If the error is calculated as the sum over all training patterns, $E(n)$, then it can be expressed as:

$$
E=\sum_{n} E(n)
$$

The goal is to evaluate the derivatives of the error function with respect to the weights contained within the network. These derivatives can be expressed as sums of each pattern derivative over the whole training set. Therefore, considering a single pattern, the derivative of $E(n)$ with respect to some weight $w_{j i}(n)$ depends only on that weight via the summed input $a_{j}(n)$ to unit $j$. Thus, the chain rule for partial derivatives can be applied to give equation 6.11.

$$
\frac{\partial E(n)}{\partial w_{j i}(n)}=\frac{\partial E(n)}{\partial a_{j}(n)} \cdot \frac{\partial a_{j}(n)}{\partial w_{j i}(n)}
$$

Letting the first term on the right hand side of equation 6.11 be denoted by $\delta_{j}$, where the $\delta^{\prime} s$ are often referred to as errors, and using equation 6.9 , we can write

$$
\frac{\partial a_{j}(n)}{\partial w_{j i}(n)}=z_{i}(n)
$$

Substituting this into equation 6.11 the derivative of the error can be expressed as:

$$
\frac{\partial E(n)}{\partial w_{j i}(n)}=\delta_{j}(n) \cdot z_{i}(n)
$$

Equation 6.13 states that the required derivative can be obtained by multiplying the value of $\delta$ for the unit at the output end of the weight by the value of $z$ for the unit at the input end of the weight. Thus, in order to evaluate the derivatives it is only necessary to
calculate the value of $\delta_{j}$ for each hidden and output unit in the network and then apply equation 6.13.

For output units, as shown in figure 6.7, equation 6.13 can be applied directly, where equation 6.10 has been used with $z_{j}(n)$ replaced by $y_{k}(n)$, such that:

$$
\delta_{k}(n)=\frac{\partial E(n)}{\partial a_{k}(n)}=\frac{\partial E(n)}{\partial y_{k}(n)} \cdot \frac{\partial y_{k}(n)}{\partial a_{k}(n)}=g^{\prime}\left(a_{k}(n)\right) \cdot \frac{\partial E(n)}{\partial y_{k}(n)}
$$



Figure 6.7: Representation of General Output Node

In order to evaluate equation 6.14 appropriate expressions for each of the terms on the right hand side of the equation are substituted in. For example, if the sum of squared error function and the sigmoidal activation function (equation 6.6 ) were used, the differential of the activation function can be expressed as:

$$
\begin{gathered}
g^{\prime}\left(a_{k}(n)\right)=\left[1-g\left(a_{k}(n)\right)\right] \cdot g\left(a_{k}(n)\right) \\
=\left[1-y_{k}(n)\right] \cdot y_{k}(n)
\end{gathered}
$$

Thus, $\delta_{k}(n)$ can be calculated using:

$$
\delta_{k}(n)=\left[t_{k}(n)-y_{k}(n)\right] \cdot y_{k}(n) \cdot\left[1-y_{k}(n)\right]
$$

For a hidden unit, as shown in figure 6.6, the chain rule is applied again (equation 6.15) where the sum runs over all units $k$ to which unit $j$ sends connections.

$$
\begin{align*}
& \delta_{j}(n)=\frac{\partial E(n)}{\partial a_{j}(n)}=\sum_{k} \frac{\partial E(n)}{\partial a_{k}(n)} \cdot \frac{\partial a_{k}(n)}{\partial a_{j}(n)} \\
& a_{k}(n)=\sum_{k} w_{k j}(n) \cdot g\left(a_{j}(n)\right) \\
& \therefore \frac{\partial a_{k}(n)}{\partial a_{j}(n)}=g^{\prime}\left(a_{j}(n)\right) \cdot \sum_{k} w_{k j}(n)
\end{align*}
$$

Applying the definition of $\delta$ to equation 6.15 and making use of equation 6.16, the following back propagation formula is obtained:

$$
\delta_{j}(n)=g^{\prime}\left(a_{j}(n)\right) \sum_{k} w_{k j}(n) \delta_{k}(n)
$$

This states that the value of $\delta$ for a hidden unit can be obtained by propagating the $\delta$ 's backwards from units higher up in the network. Using a sigmoidal activation function, $\boldsymbol{\delta}_{j}$ for training pattern $n$, for a node in the hidden layer, the errors can be expressed as:

$$
\delta_{j}(n)=y_{j}(n) \cdot\left[1-y_{j}(n)\right] \cdot\left\{\sum_{k} \delta_{k}(n) \cdot w_{k j}(n)\right\}
$$

Thus, the back-propagation procedure for evaluating the derivatives of the error $E(n)$ with respect to the weights can be summarised in four steps:

1. Apply an input vector $x$ to the network and forward propagate through the network using equation 6.9 and 6.10 to find the activations of all the hidden and output units.
2. Evaluate $\delta_{k}$ for all output units using equation 6.14.
3. Back-propagate the $\delta$ 's using equation 6.17 to obtain $\delta_{j}$ for each hidden unit in the network.
4. Use equation 6.13 to evaluate the required derivatives.

Repeating the above steps for each pattern in the training set, and then summing over all patterns obtains the derivative of the total error $E$ :

$$
\frac{\partial E}{\partial w_{j i}}=\sum_{n} \frac{\partial E(n)}{\partial w_{j i}(n)}
$$

C. Weight Change: As the batch mode of training is used the weight change if desired occurs after all training patterns have been passed through the network. If a weight change $\left(\Delta w_{j i}\right)$ is to occur then the search moves a distance in the direction of the greatest change in the error with respect to the weights $\left(\partial E / \partial w_{j i}\right)$. The gradient descent algorithm uses a parameter $\eta$ called the learning rate parameter, which governs the step size (distance moved) in the search space. In practice, a constant value of $\eta$ is often used as this generally leads to better results even though the guarantee of convergence is lost. Thus, the weight change formula applied using the batch method of training can be expressed as:

$$
\Delta w_{j i}=-\sum_{n} \frac{\partial E(n)}{\partial w_{j i}(n)}=-\frac{\eta}{N} \sum_{n} \delta_{j}(n) y_{i}(n)
$$

where $N$ is the total number of training patterns.

If the learning rate parameter is too large the weight change will result in oscillatory performance, however, if the learning rate parameter is too small then the change in weights would be very small and the search for an optimum would be slow. One way to overcome this is to use a momentum term, $\alpha$. This new weight change parameter with momentum included is used in the program net_mlp.c, and takes the form:

$$
\Delta w_{j i}{ }^{(T)}=\alpha \Delta w_{j i}{ }^{(T-1)}-\frac{\eta}{N} \sum_{N} \delta_{j}(n) y_{i}(n)
$$

Or similarly:

$$
\Delta w_{j i}{ }^{(T+1)}=\alpha\left[\Delta w_{j i}{ }^{(T)}-w_{j i}{ }^{(r-1)}\right]-\frac{\eta}{N} \sum_{N} \delta_{j}(n) y_{i}(n)
$$

where $T$ refers to the cycle number.

## PROGRAM TERMINATION:

To stop the training process of the network then some form of stopping criterion needs to be enforced. Within the program two stopping criteria are used. If any one of them is reached then the training process ceases. The criteria used are: reaching a acceptable error, which is set at a level of 0.001 ; and also in conjunction with this termination when the maximum number of iterations, set at 10000 cycles, is exceeded.

### 6.8 Training Procedure 1 - First Model

### 6.8.1 End Aim of Trained Model

The overall aim of the network is to be able to select the best scheme option for a new fault tree structure, following training of the network using a large data set. The initial data set comprised 181 trees, with an additional twenty different test trees. The test set of trees comprised a variety of structures, some very basic and others with a much larger sized resulting BDD and hence more complexity. A range was chosen to see whether the network could make correct predictions for all tree types. The target predictions required by the trained network are given in table 6.1.

| Tree _1 | 1, | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |
| :--- | :---: | :---: | :---: | :---: | :---: | :---: | :---: | :---: | :---: | :---: |
| Best Scheme(s) | $1-6$ | 1 | $1-6$ | 3 | 2 | 1,2 | 2 | 4 | 1,3 | 2 |
| Tree | 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 | 19 | 20 |
| Best Schemes | $1,2,4$ | 3 | 2 | 3 | $1-6$ | $1-3$ | $1-6$ | 2 | 3 | 3 |

Table 6.1: Target Predictive Schemes For Test Data Set

### 6.8.2 Data Set Up

The first program used was net_mlp.c, which incorporated the factors described in section 6.7.2. The scheme options were coded whereby the best scheme option (the lowest number of nodes in the BDD) was given the value 0.999 , and the worst scheme option (highest number of nodes in the BDD) the value 0.001 . Each remaining scheme option was given a value in the range relative to its position between the highest and lowest number of BDD nodes (see examples in table 6.2). The outputs were scaled in this manner to match the outputs of the activation function, therefore making comparison easy. This scaling promotes errors within a suitable range because comparing a value between 0 and 1 for an activation output with a value between 0 and 400,000 as the target output would produce ridiculously larger error values.

| No | Options | Schene 1 | Scheme 2 | Scheme 3. | Scheme 4 | Scheme 5, | Scheme 6 |
| :---: | :---: | :---: | :---: | :---: | :---: | :---: | :---: |
| 1 | Old (Nodes) | 112937 | 112553 | 110987 | 110698 | 167835 | 166390 |
|  | New Values | 0.960 | 0.967 | 0.994 | 0.999 | 0.001 | 0.026 |
| 2 | Old (Nodes) | 4 | 3 | 4 | 3 | 4 | 3 |
|  | New Values | 0.001 | 0.999 | 0.001 | 0.999 | 0.001 | 0.999 |

Table 6.2: Coding For Scheme Options

Similar to the scheme outputs, the input characteristics were also scaled between the values of 0 and 1 . The reason behind this was due to the large range of possible input values, for example the values for the top gate type could be either 1 or 0 , whereas for the number of basic events the value could be up to 364 . To produce weights in the
initialisation to cater for differences in these values would not be possible unless either separate weights were chosen for each column or all the inputs were scaled. Thus, for each characteristic the largest value within the training data set was given the value of 0.950 and the smallest value 0.050 and the other inputs were scaled in this range relative to the largest value. Values above the largest or below the smallest characteristic values, which may be found in new trees, were allocated the values of 1 and 0 respectively. The only exceptions to this were with the top gate type, whereby an OR gate was given the value of 0 and an AND gate 1 .

### 6.8.3 Setting Up The Network Architecture For The Problem

Having generated the training and test data sets, determined the number of characteristics to represent the problem and the number of scheme preferences as outputs the initial network architecture could be constructed. To review, the base network structure is given by the network architecture described below:

Network Architecture: This comprised of eleven fault tree characteristics, each of which was represented by one node in the input layer. The output layer comprised of six nodes, one for each of the scheme preferences. The data set comprised 181 training fault tree structures of varying sizes and complexity, with 20 test trees of different configurations. When using the network for prediction purposes the best scheme preference was established as the node value that lies nearest to the largest value (used as the most optimal scheme preference in training).

Throughout the research the network model architecture centred around altering the number of hidden layers and the number of nodes in each hidden layer. The alterations occurred in a systematic fashion whereby first one hidden layer was tried and two hidden nodes. Then three nodes, four, etc up to ten were all used to train the model. After this two hidden layers were tried, starting with two nodes in each layer and successively increasing both values. The maximum number of nodes used in either layer for the two
hidden layers was six, as it was felt that the amount of data available was not sufficient to make use of more than this number of nodes. At the same time, the learning and momentum parameters used in the optimisation algorithm were continually altered. Altogether 186 trials were carried out and the predictive results are summarised in table 6.3. The majority of trial predictions fell below eight correct schemes chosen out of twenty. However, the aim is to find a single network architecture, which produces the most correct predictions, ideally twenty. Using the current set up the most correct predictions totalled 12.

| Score | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 |
| :---: | :---: | :---: | :---: | :---: | :---: | :---: | :---: | :---: | :---: | :---: | :---: | :---: | :---: |
| No, | 10 | 8 | 12 | 16 | 27 | 19 | 44 | 15 | 21 | 4 | 6 | 3 | 1 |

Table 6.3: Predictive Scores Obtained in Training

### 6.8.4 MLP Architecture For Best Results

The best network set up found using the net_mlp.c program was trained with 6 output nodes, 11 input nodes, and 2 hidden layers each with 4 nodes in them. Looking at the error every one hundred cycles over the 10000 cycles that occurred during training, there was oscillation. To try to alleviate this the learning rate parameter within the program was reduced, until the oscillation no longer existed. The learning rate parameter was reduced from 0.09 to 0.02 . At this value however the reduction in the error term from cycle to cycle was very small. The momentum term was set to 0.005 . At this stage the best results obtained was an error of 0.036464 , and from this it predicted $12 / 20$ correct. This result is already better than anything produced with the machine learning classifier approach and shows significant promise with more study.

### 6.8.5 Altering The Precision of The Inputs and Outputs

One alteration that was made and executed using the same net_mlp.c program was the precision of the inputs and outputs. With a value near to the sigmoid functions range the
harder it is to achieve, so instead of having the outputs in four decimal places, this was reduced to two. Using the same architecture as above there was an improvement in performance ( $17 / 20$ correctly predicted). However this improvement is due to the reduction in precision, namely if two outputs previously were coded as 0.010 , and 0.013 , when re-scaled, these would come out as 0.01 and 0.01 , hence this time it could correctly predict the coded scheme if previously it predicted the wrong one, although in terms of the actual number of nodes the prediction would be incorrect. One point to consider here is that if two schemes are so closely related, for example if there are only a few nodes difference in the resulting diagram does it really matter which scheme is selected as the optimal one?

### 6.9 Optimisation Algorithm Changes - Enhanced Gradient Descent

After the numerous trials that were executed using the standard gradient descent optimisation procedure, further developments were only expected from a change in the algorithm. One problem with simple gradient descent, with momentum term included, is that it contains two parameters $\alpha$ and $\eta$ whose values have to be selected by trial and error. The optimum values for these parameters will depend on the particular problem, and will typically vary during training. It would be more advantageous if a procedure was introduced which would automatically set these parameters during the training process. These automated processes of setting the variable parameters are introduced in the enhanced gradient descent approach. One such approach is the bold driver technique ${ }^{[\mathrm{VMR} 88]}$ whereby depending on the change in the error the learning rate parameter is altered to maximise the search.

This technique includes a multiplicative factor for $\eta$ the learning rate parameter, and alters it depending on whether the error has increased or decreased. If the error function has decreased after one step of the gradient descent then the new values of the weight change are accepted. However, the learning rate may have been too small so its value is increased. Alternatively, if the error has increased after a given step in the gradient descent then the algorithm must have overshot the minimum, and so the learning rate
parameter must have been too large, hence, the weight change is undone and the learning rate is decreased. Hence, updating the new learning rate parameter follows these rules:

$$
\begin{array}{ll}
\eta_{\text {new }}= & \beta^{*} \eta_{\text {old }} \text { if } \Delta \mathrm{E}<0 \\
& \varphi^{*} \eta_{\text {old }} \text { if } \Delta \mathrm{E}>0
\end{array}
$$

The parameter $\beta$ to increase the learning rate parameter is chosen to be slightly greater than 1 . The parameter $\varphi$ to reduce the learning rate parameter value is chosen to be significantly less than 1 , e.g. 0.5 .

### 6.10 Training Procedure 2 - Using The Enhanced Gradient Descent Technique

### 6.10.1 Using Eleven Input Characteristics With New Optimisation Algorithm

The training and test data set remain the same and the input and output layers of the neural network both remain unaltered. The systematic alteration of the number of hidden nodes and number of layers, with associated changes in the starting values for the learning rate parameter, momentum term and now the increasing and decreasing factors ( $\beta$ and $\varphi$ respectively) required by the enhanced gradient descent technique is undertaken. The program used to generate the models with this new optimisation algorithm is enhanced.c.

### 6.10.2 Best Network Architecture

Two hundred different trials were carried out, varying the network parameters. The range of predictive values varied from 3 to 13 . The distribution of these results can be seen in figure 6.8. The best network architecture was found with the characteristics shown in table 6.4, and the network parameters were set to those shown in table 6.5.

| Structure Considerations | , |
| :--- | :--- |
| Number of Input Nodes | 11 |
| Number of Output Nodes | 6 |
| Number of Hidden Layers | 1 |
| Number of Nodes in Hidden layer(s) | 5 |

Table 6.4: Best Network Architecture

| Parameter Considerations |  |
| :---: | :---: |
| Learning Rate Parameter, $\eta$ | 0.01 |
| Momentum parameter, $\alpha$ | 0.005 |
| Scaling Parameters for $\boldsymbol{\eta}$ - Increasing factor | 1.04 |
| Scaling Parameters for $\eta$ - decreasing factor | 0.6 |

Table 6.5: Parameter Values Associated With Best Network Architecture


Figure 6.8: Distribution of Predictive Scores for 200 Models Trained and Tested

Using the enhanced gradient descent technique has lead to slight improvements in the predictive potential of the neural network compared to the standard gradient descent optimisation technique. The network predicted 13 out of the 20 test trees with correct scheme preferences. The error at the end of training was 0.341937 . Despite this value
being the best achieved so far, improvements can still be made, with currently six incorrect schemes being chosen. The inputs and outputs are in turn examined.

### 6.11 Training Procedure 3 - Reviewing The Inputs and Outputs

### 6.11.1 Input Characteristics Altered

It appears that the choice of the input characteristics is the key to the success of the network, as it is from these characteristics that an output is predicted. As it is not known which characteristics are important, it is possible that contained within the eleven currently in use are irrelevant characteristics. Having such a large amount of information (eleven input nodes) with possible irrelevancies may cause the network difficulties in learning. By looking at the scheme preferences themselves and observing the rules that the heuristics use to construct the ordering for the tree, the network was modelled on five key characteristics (determined by observation). These five input characteristics being:

1. Number of basic events.
2. Number of levels in the tree.
3. Percentage of different events that are repeated.
4. Number of inputs to the top gate.
5. Number of gates with just event inputs.

The reasoning behind this choice of characteristics is:

1. If the number of basic events in the tree is large then the possible variations in the ordering list are larger as compared to an original list with a smaller number of basic events.
2. The number of levels can influence the variation in the ordering permutation, as a larger number of levels will affect the resultant ordering when using the subtree type approaches compared to the top-down approach.
3. As some of the ordering schemes consider repeated events, the greater the number of repeated events the greater the possible variation. If there are no repeated events the ordering will be unaltered.
4. The number of inputs to the top gate influences the number of subtrees within the main tree and these are used in some of the ordering permutations hence possibly altering the ordering.
5. As the priority depth-first approach looks at gates with just basic event inputs then the more of these that exist the greater the possible change in ordering as compared to the other heuristics.

This reduced set of characteristics are all scaled in the range 0 to 1 . The model is tested using the enhanced gradient optimisation technique. One hundred and twenty trials were carried out. The results have been similar to those already obtained using the eleven characteristics, with variable predictive scores depending on the network architecture and parameters. Unfortunately, no further improvements in the predictive score have been found, with the best architecture being summarised in table 6.6. It seems evident from the results that the characteristics of the trees need to be examined further, with these five not providing the necessary information to separate the scheme choices sufficiently.

| Parameters Values | 5 |
| :---: | :---: |
| Input Nodes | 6 |
| Output Nodes | $1 / 5$ |
| Hidden Layer(s) / Nodes | $0.04 / 0.001$ |
| Learning Parameter / Momentum | $1.05 / 0.6$ |
| Increasing / Decreasing Factor | $12 / 20$ |
| Predictions |  |

Table 6.6: Best Network Architecture Using 5 Inputs and 6 Outputs

### 6.11.2 Reviewing Outputs

By changing the inputs there was no improvements in the predictions made. Another possible problem could be the outputs themselves. In this attempt to find the optimal network solution the number of scheme alternatives were reduced. In examining the best scheme options for all the fault trees it was found that on very few instances schemes five and six led to a best ordering. Therefore, it was decided to concentrate on the scheme options that were more influential, resulting in only four output nodes. The training data set of fault tree structures was altered to comprise only trees with greater than ten minimal cut sets, hence eradicating the inclusion of very small trees where the ordering is not important, and now totalled 146. The set also comprised equal groupings of the four scheme preferences, as multi-layer feedforward networks can not adjust for unbalanced training sets ${ }^{\text {[Mas93] }}$. If a set is disportionately represented, the network will strive to optimise its performance when presented with members of that subclass, at the expense of members of the other subclasses. Eleven characteristics were again used as the inputs following no improvements with five. The prediction process was the same as for all previous training methods tried.

Eighty trials were carried out with all results less than or equal to ten correct predictions. This is less than using the original six schemes. This result could be influenced by the change in training data set, and it is expected that a greater number of training trees needs to be included to provide the necessary information for these more complex tree structures.

### 6.12 Training Procedure 4 - New Data Set

### 6.12.1 New Data Set - Removal of Redundant Trees

On reviewing the data set it was thought necessary to remove trees with redundant information, i.e. those trees which resulted in a same sized BDD regardless of the scheme option. Following the results of the four scheme output problem more training data was
added, to help with possible lacking information. The training set was increased to 198 trees using a program to generate them randomly. The predictions required by the network also changed as some of the trees within the test set were altered. The scheme choices required by the network are shown in table 6.7, where the number of nodes for the resulting BDDs for each scheme option is given, and the target response is highlighed.

| Tree | Scheme 1 | Scheme 2 | Scheme 3 | Scheme 4 | Scheme 5 | Scheme 6 |
| :---: | :---: | :---: | :---: | :---: | :---: | :---: |
| 1 | $\mathbf{3 4}$ | 478 | 1657 | 1117 | 6640 | 3477 |
| 2 | 54 | 44 | 39 | 113 | 771 | 771 |
| 3 | 79169 | 55680 | 137991 | 65900 | 507669 | 507544 |
| 4 | 253 | 253 | 284 | 284 | 1830 | 1706 |
| 5 | 6836 | 5197 | 10152 | 10018 | 999999 | 999999 |
| 6 | 526 | 499 | 330 | 303 | 330 | 39677 |
| 7 | 157 | 139 | 157 | 139 | 1393 | 916 |
| 8 | 3504 | 2548 | 119593 | 59599 | 396283 | 214895 |
| 9 | 40 | 40 | $\mathbf{3 5}$ | 40 | 4337 | 4337 |
| 10 | $\mathbf{8}$ | $\mathbf{8}$ | $\mathbf{8}$ | $\mathbf{8}$ | 47 | 40 |
| 11 | 8453 | 487 | 218284 | 15334 | 999999 | 999999 |
| 12 | 253 | 298 | 148 | 194 | 5326 | 5326 |
| 13 | 960 | 905 | 3901 | 2240 | 5802 | 4438 |
| 14 | 47 | 44 | 43 | 49 | 441 | 362 |
| 15 | 39 | 39 | 30 | 60 | 1921 | 141 |
| 16 | 12368 | 9959 | 58847 | 16818 | 999999 | 999999 |
| 17 | 4953 | $\mathbf{3 0 5 2}$ | 7986 | 3868 | 842393 | 473405 |
| 18 | 31276 | 9172 | 11236 | 3716 | 147594 | 99047 |
| 19 | 26233 | 17901 | 42677 | 39592 | 241117 | 217713 |
| 20 | 386 | 366 | 264 | 202 | 701 | 649 |

Table 6.7: Test Tree Target Predictions (highlighted scheme options)

Ninety five different models were created with varying degrees of success. The worst predictions made were four out of twenty correct choices, however, the best equalled 14 . The network design that produced the best result is summarised below:

- Number of Input Nodes: 11
- Number of Output Nodes: 6
- Number of Hidden Layers: 1
- Number of Nodes in Hidden Layer: 5
- Learning Rate Parameter: 0.01
- Momentum term: 0.001
- Increasing Factor ( $\beta$ ): 1.04
- Decreasing Factor ( $\varphi$ ): 0.5
- Training Error: 0.236714
- Test Error: 0.899318

The predictions made for the test set of trees were:

Scheme: 2, 3, 2, 2, 2, 3, 2, 2, 3, 3, 2, 2, 2, 2, 2, 2, 2, 4, 2, 2

Again still further improvements could be made to the predictive capacity of the model. With the new data set the previous work carried out with the reduced number of characteristic inputs is re-tested.

### 6.13 Training Procedure 5 - Re-testing Five Characteristics

Using the new data set of 198 trees, the enhanced gradient optimisation algorithm was applied to the network with five input nodes and 6 output nodes. A number of network architectures were tested, 120 in total, involving both single and double layers of hidden nodes. The predictive scores produced ranged from 4 to 12 . The results predicting 12 correct responses were generated with a network comprising two hidden layers. The architecture and parameters used are summarised in table 6.8.

| Network Parameter | Value |
| :--- | :---: |
| Input Nodes | 5 |
| First Hidden Layer | 2 |
| Second Hidden Layer | 3 |
| Output Nodes | 6 |
| Learning Rate Parameter | 0.01 |
| Momentum | 0.1 |
| Increasing Factor | 1.09 |
| Decreasing Factor | 0.7 |

Table 6.8: Best Results Using Five Characteristics

The predictions made were:

Scheme 2, 2, 2, 2, 4, 4, 2, 4, 2, 2, 4, 2, 4, 2, 2, 2, 2, 2, 2

The results with this model are worse than the network that takes in eleven nodes. With these five inputs the number of correct scheme choices can still be improved, with still eight inaccuracies.

### 6.14 Establishing if Over-Training is Occurring

### 6.14.1 Considerations in Training - Length of Training Period

One point to consider is the theory of over training or over fitting. The theory states that there is an optimal amount of training, and that continuing past this point will improve performance of the training set, but degrade performance on the general population (shown in figure 6.9).


Figure 6.9: Over Training Phenomenon

To observe this phenomenon involves taking the training and test error after varying numbers of cycles. The training error should continually decrease, whereas there will come a point (denoted by A in figure 6.9) where the test error increases despite the decrease in training error. At this point, the model is said to be 'over fitted'.

To minimise the possibility of the occurrence of the phenomenon, the best solution is to use as few hidden nodes as possible. Also great care must be taken to ensure the training set adequately represents the population hence reducing the number of idiosyncrasies or rogue data patterns. If the training set does not resemble the population, the phenomenon of over fitting will almost always appear, regardless of the number of hidden nodes. Accurately training the network will be a hopeless task.

Currently the number of iterations set for the training phase is 10,000 . To establish if performance of any network has been affected by over training the task is to find out whether the phenomenon has occurred within these 10,000 iterations. To carry out this investigation, initially an error term was required for the prediction phase. This followed a similar format to that used in training. The averaged sum-of-squared error term was used. Re-running the best network architecture again ( 14 out of 20 correct using enhanced gradient descent technique), the test data set error after 10,000 iterations was calculated as 0.899318 .

To establish if the current termination point has incurred over training principles, the network was trained in cycles of multiples of 1000 iterations, the weights generated at each stage are used in the predictive network to calculate the error in the test data. The point at which the test error increases despite the training error decreasing indicates the point at which the network is beginning to be over trained.

| Number of Iterations | Training Error | Test Error | Predictions |
| :---: | :---: | :---: | :---: |
| 0 | 0.617862 | 0.637808 | 13 |
| 1000 | 0.352643 | 0.956745 | 13 |
| 2000 | 0.352638 | 0.956735 | 13 |
| 3000 | 0.352636 | 0.956728 | 14 |
| 4000 | 0.341189 | 0.899349 | 14 |
| 5000 | 0.341189 | 0.899345 | 14 |
| 6000 | 0.341189 | 0.899340 | 14 |
| 7000 | 0.341189 | 0.899334 | 14 |
| 10000 | 0.341187 | 0.899318 | 14 |
| 15000 | 0.341182 | 0.899300 | 14 |
| 20000 | 0.341178 | 0.899295 | 14 |
| 30000 | 0.341163 | 0.899289 | 14 |
| 40000 | 0.341152 | 0.899296 | 14 |
| 50000 | 0.341140 | 0.899307 | 14 |
| 80000 | 0.341119 | 0.899347 | 14 |

Table 6.9: Testing For Over Training

The network architecture which produced the fourteen out of twenty correct predictive scores was used to test the over training principle (section 6.10.2). The training and test errors and resultant predictive scores after varying number of iterations are summarised in table 6.9. It is shown that even after 40,000 iterations both the training and test error were decreasing, albeit slowly. It is felt that the current setting for the program
termination with regard to the number of iterations is not sufficient to have an over training effect on the network.

### 6.15 Conclusions of Best Results

### 6.15.1 Performance Based Assumptions

The multi-layer neural network method for pattern recognition shows an improvement in the number of correct scheme preferences selected compared to the results presented using the classifier system application. The best result produced for predicting the correct scheme heuristic is 14 out of 20 achieved using eleven input characteristics. With this network the training data set consisted of only trees with greater than ten minimal cut sets. This result reflects the possibility of using the MLP technique to select an appropriate ordering heuristic, as this method provides a 70 percent chance of getting an optimal or near optimal BDD which is considerably better than using any of the heuristics individually mentioned in the literature.

### 6.15.2 Best Architecture

The best predictive performance for the ordering problem has been achieved using the basic network architecture of eleven characteristic inputs, six outputs, and one hidden layer comprising of five hidden nodes. The optimisation technique used to minimise the error by altering the weights was the enhanced gradient descent technique. The learning rate parameter was set to 0.01 , the momentum term to help prevent oscillation was set to 0.001 . The increasing and decreasing factors applied to the learning rate parameter to alter it depending on the effect upon the error were set to 1.04 and 0.5 respectively. Through the training process the weights of the network were determined. The first layer weights from the input layer to the hidden layer are shown in table 6.10.

|  | Hidden 1 | Hidden 2 | Hidden 3 | Hidden 4 | Hidden 5 |
| :--- | :--- | :--- | :--- | :--- | :--- |
| Bias node | 0.147330 | 0.243064 | 0.539561 | 0.673099 | -0.331514 |
| Input 1 | 0.641682 | 0.183194 | -0.924516 | -0.665176 | 0.708712 |
| Input 2 | 0.796946 | 0.886754 | 0.186239 | -0.306955 | 0.086139 |
| Input 3 | 0.842000 | 0.883227 | 0.623115 | -1.054132 | -0.037604 |
| Input 4 | 0.382882 | 0.365247 | -0.433369 | -0.995933 | 0.156702 |
| Input 5 | 0.506398 | 0.347366 | -0.710282 | -0.981350 | -0.983196 |
| Input 6 | 0.642397 | 0.202210 | 1.478797 | 1.438517 | 0.256880 |
| Input 7 | -0.382380 | 0.972270 | 0.371361 | -0.064075 | -0.772999 |
| Input 8 | -0.963998 | 0.648210 | 0.894712 | 0.339276 | -0.428676 |
| Input 9 | 0.329570 | 0.264644 | 0.067410 | -0.007906 | 0.446248 |
| Input 10 | -1.004186 | 0.236299 | 0.296611 | 0.159278 | 0.035732 |
| Input 11 | 0.603500 | 0.120445 | -0.361202 | -0.926677 | 0.304544 |

Table 6.10: First Layer Weights of Best Network

The second layer weights, from the hidden layer to the output layer are given in table 6.11 .

|  | Output 1 | Output 2 | Output 3 | Output 4 | Output 5 | Output 6 |
| :--- | :--- | :--- | :--- | :--- | :--- | :--- |
| Bias node | 0.435191 | 1.562687 | 0.732578 | 1.316444 | -0.566968 | 0.199515 |
| Hidden 1 | 0.034015 | 0.156753 | 0.507769 | -0.003383 | 0.253515 | 0.194763 |
| Hidden 2 | 0.902266 | 0.383328 | -0.219266 | -0.184228 | 0.102770 | -0.108666 |
| Hidden 3 | 0.184042 | 0.803264 | 0.543061 | -0.025922 | -1.121804 | -0.156623 |
| Hidden 4 | 0.291198 | -0.271374 | 0.895298 | 0.851694 | -1.121293 | -0.405690 |
| Hidden 5 | -0.378771 | 0.430375 | 0.526418 | 0.116150 | 0.104950 | -0.617658 |

Table 6.11: Second Layer Weights of Best Network

### 6.15.3 Future Considerations

Obviously to improve the method further a larger set of training data would be necessary. This would accommodate for more fault tree structures and their relation to the 'optimal' scheme. Also changing the parameter optimisation technique could alter the network performance. From the results gained so far it does appear that this method could be used to predict the most 'optimal' ordering permutation of the basic events in a fault tree from a set of alternative schemes, to be used to generate an optimal or near optimal BDD. Of most influence are the inputs themselves, and before more complex methods of optimisation are tried it is these characteristics which need to be examined. If the input characteristics of the data do not reflect those needed to determine the best scheme permutation then no neural network architecture will produce the desired result. It does appear that some pattern recognition approaches are more suited to the ordering problem. With the improvement seen in the multi-layer neural network approach, compared to the genetics based machine learning system, possible improvements are expected by trying an alternative network technique, i.e. radial basis function neural networks.

### 6.16 Summary

- The multi-layer perceptron approach has been adapted to model the variable ordering problem.
- The best model was found using an enhanced gradient descent optimisation algorithm.
- Numerous numbers of hidden layers and nodes were experimented with but the best architecture involved a single layer with five hidden nodes in it.
- The best predictive results gained were 14 out of 20 correct, using eleven input nodes.
- Additional points to test are to incorporate a more complicated optimisation algorithm, and increase the training data set size, hopefully to model more of the population domain.
- Analysis of the inputs of the network model needs to be undertaken to see whether they relate to the best ordering scheme heuristic.
- The model has proven overall to be a reasonable predictor of the scheme preference and could be used as a possible solution to this problem.
- The final pattern recognition approach of radial basis functions should be tested, as it may be more suited to the task.


## 7 Radial Basis Function Neural Networks

### 7.1 Introduction

An alternative form of neural network to use in the ordering dilemma problem is that of a Radial Basis Function (RBF). Like the multi-layer perceptron approach the aim of the technique is to solve a non-linear mapping between a set of inputs and their desired outputs. However, this approach has some fundamental differences compared to the multi-layer perceptron approach, which may be of benefit to this problem. The radial basis function method again involves a network with an input layer, output layer and hidden units. In this method the hidden units are dependent upon the distance between the input vector and a prototype vector.

Several radial basis function neural networks have been designed to try and model the mapping between a fault trees characteristics' and the optimal ordering scheme to use in the BDD conversion process. Three models were initially generated, the first using four characteristics as the inputs and six scheme preferences as the outputs. The second model used eleven characteristic inputs to try and distinguish more convincingly between the scheme choices for each tree. The final model generated used just four scheme preferences as outputs rather than six as under observation two of the schemes were on very few occasions the best alternatives. The favoured results have been produced using the latter two models, namely the eleven characteristics and six scheme output model and the eleven characteristic and four output model, both with a correct predictive probability of 70 percent. Also, results with the test set of twenty trees indicated that the incorrect predictions were on the majority of occasions the second best alternative with very little difference in the size of the BDD produced. Thus, the research has shown that the radial basis function neural network approach is the best solution to date to finding an optimal or near optimal BDD for most fault tree structures. Further improvements in this method
are expected by investigation of the relevance of the characteristic inputs in differentiating between the scheme outputs. The research steps and conclusions reached are reviewed in this chapter. One publication ${ }^{[B a n 002]}$ has been produced concerning the use of the radial basis neural network approach to the BDD ordering problem.

### 7.2 Comparison With Multi-layer Perceptron Approach

Both Multi-layer Perceptron and RBF network approaches provide the same role in approximating arbitrary non-linear functional mappings. However, there are slight differences between each of the approaches, these are summarised below.

## 1. The hidden units:

Within a multi-layer perceptron the hidden units are generated by computing a linear sum of the data inputs and the weights, which are transformed by a non-linear activation function. The hidden nodes in the radial basis function are generated depending on the distance between the input and a prototype vector and transformed using a localised function.

## 2. Contribution of hidden units to the output values:

In a multi-layer perceptron, for a given input vector many hidden units may contribute to the determination of the output value. The contribution of the hidden units to generate the correct outputs for a range of possible input values leads to problems in training such as local minima. This problem can lead to slow convergence of the network. In comparison the radial basis function network deals with localised basis functions which form a representation in the space which is local to the input values. This localisation means that typically only a few hidden units will have significant activations that will influence the output values.

## 3. Network Architecture:

The radial basis function architecture is very simple with just two layers of weights, in which the first layer contains the parameters of the basis function and the second forms the linear combinations of the activations of the basis functions to generate the outputs. In contrast, the multi-layer perceptron has a very complex architecture with lots of connectivity. There may be numerous layers of weights and a number of different activation functions.

## 4. Training Phases:

In a multi-layer perceptron supervised training occurs whereby all the parameters in the network are determined at the same time. However, in a radial basis function network there are two stages of training. Initially the network parameters are established using only the input data, in an unsupervised method of training. The second stage involves calculating the weights in a fast linear supervised training manner.

### 7.3 Advantages and Disadvantages of Radial Basis Functions Compared to Multi-layer Perceptrons

### 7.3.1 The Advantages of The RBF

Considering the differences between the RBF neural network approach and the multilayer perceptron approach, there are some potential benefits and some disadvantages of the radial basis function network. The advantages of the radial basis function approach are:

1. During training the radial basis function neural network does not get stuck in any local minima, because there are none.
2. The time taken for training is much shorter.
3. Examples that are far from decision boundaries have little influence on the radial basis function (while in multi-layer perceptions they have more influence).

### 7.3.2 The Disadvantages of The RBF

Despite the advantages, there are some factors that limit the radial basis function performance as compared to that of the multi-layer perceptron. These are:

1. More computer memory is required due to the large number of hidden nodes (radial basis function centres) that are used.
2. Some radial basis function networks require a second algorithm to be programmed to perform clustering to find the centres of the hidden nodes.
3. The general speed of operation of a RBF neural network is slightly slower.
4. Due to the unsupervised method of learning the network parameters (centres and width or spread of the RBF) are not optimal.

As the RBF model has only to deal with up to eleven input nodes and six output nodes, the problems associated with memory and speed are not going to be critical for this size of problem.

### 7.4 Dynamics of a Radial Basis Function Neural Network

### 7.4.1 RBF Theory

The theoretical origins of radial basis functions are founded in the techniques for performing exact interpolation of a set of data points in a multi-dimensional space ${ }^{[B i s 95]}$. The exact interpolation problem requires every input vector to be mapped onto the corresponding target vector. The radial basis function approach uses a modified version of the exact interpolation technique. The approach introduces a set of basis functions,
whereby the number of these functions is determined by the complexity of the problem. Each radial basis function takes the form:

$$
\Phi_{j}\left(\left\|\underline{x}-\underline{\mu}_{j}\right\|\right)
$$

where $\Phi_{( }($.$) is some non-linear function (discussed in section 7.4.2). Thus the j^{\text {th }}$ radial basis function depends on the Euclidean distance $\left\|\underline{x}-\mu_{j}\right\|$ between $\underline{x}$ (the set of input vectors) and $\mu_{j}$ (the $j^{\text {th }}$ radial basis function centre vector).

Like multi-layer perceptrons the network has a number of input nodes, representing each feature component of the problem, and a number of output nodes, or targets. The radial basis function network typically has only one layer of hidden nodes, where each node has an activation function centred on a chosen radial basis function, denoted by $\Phi_{j}($.$) .$ Diagrammatically the radial basis function neural network (as shown in figure 7.1) looks similar to the perceptron model.


Figure 7.1: Radial Basis Function Neural Network Architecture

The connections between the input layer and the hidden layer of radial basis functions represent the vectors determining the centres of the radial basis functions. For example, if the vector $\mu_{l}$ (with elements $\mu_{l j}$ connecting radial basis function 1 and input node $j$ ) determining the centre of radial basis function $\Phi_{l}$ was $\{1.3,1.6,0.4,0.7\}$ then the connection between input node $X 1$ and hidden node $\Phi_{I}$, represented by $\mu_{I I}$ would be equal to 1.3. The connection $\mu_{12}$ would equal $1.6, \mu_{13}$ equalling 0.4 and the connection $\mu_{14}$ (with $n=4$ in figure 7.1) would equal 0.7. The first stage of training identifies these parameters, denoted by $\mu_{\mathrm{xx}}$ in the diagram in figure 7.1.

The connections between the basis functions and the output layer represent the weights of the network, and these are determined during the training phase of the neural network. The connection between radial basis function $\Phi_{I}$ and output node 1 is $w_{11}$, between $\Phi_{2}$ and output node 3 is $w_{32}$ and so on.

The output of the $k^{\text {th }}$ node, $y_{k}(\underline{x})$, representing the mapping of the radial basis function neural network is then taken to be a linear combination of the basis functions and the weight vectors (the connections between the output layer nodes $k$ and the basis functions $j$ ) associated with all paths to the desired output node, written in the form:

$$
y_{k}(\underline{x})=\sum_{j=1}^{m} w_{k j} \Phi_{j}\left(\left\|\underline{x}-\underline{\mu}_{j}\right\|\right)+w_{k 0}
$$

The right most term in expression 7.2, $w_{k 0}$, is representative of the bias. This can be absorbed into the summation by including an extra basis function $\Phi_{0}$, whose activation is set to 1 . The purpose of the bias is to add a constant to the equation.

In the ordering problem, the input and output layer of nodes are fixed. The input nodes, as in the multi-layer perceptron represent the chosen characteristics of the fault tree structure. The output nodes similarly represent the six variable ordering options hence
there are six for the first two models generated and four for the latter. The hidden layer of nodes is variable, and is the key to finding a network architecture that will model and solve this problem sufficiently and accurately. The methodologies for selecting the radial basis function centres and the techniques for establishing the weights of the network are explained in detail in the following sections.

### 7.4.2 Types of Radial Basis Function

There are several non-linear basis functions that could be considered, the most common is the Gaussian function of the form:

$$
\Phi_{j}(x)=\exp \left(-\frac{\left\|\underline{x}-\underline{\mu}_{j}\right\|^{2}}{2 \sigma_{j}^{2}}\right)
$$

where $\underline{x}$ is the $d$-dimensional input vector with elements $x_{i}$, and $\mu_{j}$ is the vector determining the centre of the basis function $\Phi_{j}$ and has elements $\mu_{j \text {. }}$. The parameter $\sigma$, or width parameter, controls the smoothness of the interpolating function. The Gaussian function is a localised basis function with the property that $\Phi \rightarrow 0$ as $|x| \rightarrow \infty$.

Another choice of basis function with the same properties is the function

$$
\Phi_{j}(x)=\left(x^{2}+\sigma^{2}\right)^{-\alpha} \quad \alpha>0
$$

The radial basis function does not need to be a localised function, thus another choice is the thin-plate spline function

$$
\Phi(x)=x^{2} \ln (x)
$$

Other alternatives to the spline function are:
$\begin{aligned} & \text { i. } \Phi(x)=\left(x^{2}+\sigma^{2}\right)^{\beta} \quad 0<\beta<1 \text {, which for } \beta=1 / 2 \text { is known as the multi- } \\ & \text { quadratic function } \\ & \text { ii. The cubic function: } \\ & \\ & \text { iii. The linear function: }\end{aligned} \quad \Phi(x)=x^{3} \quad 7.6$

These functions all have the property that $\Phi \rightarrow \infty$ as $|x| \rightarrow \infty$.

### 7.4.3 Two Phases of Training - Optimisation of Parameters

### 7.4.3.1 Brief Description of Phases

Training of the radial basis function neural network can be considerably quicker than training a multi-layer perceptron as only one cycle is performed. As mentioned before there are two phases of training, and there is a clear distinction between the roles of the first and second phases. In the first phase, the input data set alone is used to determine the network parameters, namely the basis function centres and width or spread parameters. In the second phase the network parameters remain fixed while the second layer weights are established.

### 7.4.3.2 Phase 1 of Training: Determining The Network Parameters

There are two parameters during this stage that are determined:

1. The Radial Basis Function Centres.
2. The Width Parameters.

There are a variety of techniques that could be used to select the radial basis function centres, and the problem itself will determine the complexity of the selection technique required. One of the simplest procedures for selecting the centres is to set them equal to a random subset of the input vectors from the training data set. Although this may not be an optimal approach in that it may require a large number of radial basis function centres in order to achieve adequate performance on the training data set, it is often used as a starting point upon which other techniques can be compared.

In an alternative approach all the training data points are selected as radial basis function centres and then selectively centres are removed so as to minimise the disruption to the performance of the network.

More principled approaches to selecting the centres of the network are related to the techniques of orthogonal least squares and clustering algorithms. In addition Kohonen topographic feature maps, Gaussian mixture models or non-linear optimisation algorithms can be applied. All of these methods are discussed in Bishop ${ }^{[B i s 95]}$.

The procedures highlighted above select only the centres. A separate procedure is required to determine the width parameters of the radial basis functions. In determining the width parameters one consideration is whether each radial basis function has its own specific width parameter or if all functions have the same parameter. A common heuristic is to choose all the width parameters to be the same and equal to a multiple of the average distance between the basis function centres. This ensures that the basis functions overlap to some degree and hence give a relatively smooth representation of the distribution of the data set.

All the approaches mentioned above to select the network parameters (centres and widths) are very fast and allow a radial basis function network to be generated very quickly, although the network may produce sub-optimal performance.

### 7.4.3.3 Phase 2 of Training: Optimising The Weights

The second phase of training involves a more complex mathematical procedure. In this phase the aim is to establish the weight values of the connections between the RBF centres and the output nodes. One consideration in this phase of training is that if there are fewer basis functions than there are training data patterns (which is usually the case) then in general, it will not be possible to find a set of weight values which will map the inputs to the outputs exactly. Having fewer radial basis functions means that generalisation to new data is much better. Hence, selecting the number of basis functions to use in the network is a balance between network performance on the training set (producing a good model for problem) and generalisation of the trends and patterns.

Considering the formula representing the radial basis function mapping producing an output given the inputs for training pattern $p$ :

$$
y_{k}\left(x^{p}\right)=\sum_{j=0}^{N} w_{k j} \Phi_{j}\left(x^{p}\right)
$$

where $N$ is the total number of radial basis function centres, $k$ represents each output node and there are $S$ training patterns in total.

This can be written in matrix notation for all outputs ( $M$ in total), such that:

$$
\begin{align*}
& \underline{Y}(\underline{x})=\underline{W} \Phi(\underline{x})
\end{align*}
$$

where $\underline{W}$ represents the weights $w_{k j}$ connecting each output node $k$ to hidden node $j$ thus has dimension $M x N$, and $\Phi$ represents each activation $\Phi_{/}\left(x^{k}\right)$ by calculating the distance between input vector for training pattern $p$ and centre vector $j$, which has dimension $S x$ $N$.

The weights can be optimised by minimising a suitable error function. The most commonly used function is the sum-of-squares error function, given by:

$$
E(W)=\frac{1}{2} \sum_{p=1}^{S} \sum_{k=1}^{M}\left\{y_{k}\left(x^{p}\right)-t_{k}^{p}\right\}^{2}
$$

where S is the total number of training patterns, $t_{k}{ }^{p}$ is the target value for output unit $k$ when the network is presented with input vector $\underline{x}^{p}$. Using equation 7.9, equation 7.11 can be rewritten in terms of the radial basis function mapping as:

$$
E(W)=\frac{1}{2} \sum_{p=1}^{S} \sum_{k=1}^{M}\left\{\sum_{j=0}^{N} w_{k} \Phi_{j}\left(x^{p}\right)-t_{k}^{p}\right\}^{2}
$$

Differentiating equation 7.12 with respect to the weights, $w_{k j}$, and minimising (equating the derivative to zero) the result is:

$$
\frac{\partial E(W)}{\partial w_{k j}}=\sum_{p=1}^{S}\left\{\sum_{j=0}^{N} w_{k j} \Phi_{j}\left(x^{p}\right)-t_{k}^{p}\right\} \Phi_{j}^{p}\left(x^{p}\right)=0
$$

Therefore in matrix form:

$$
\boldsymbol{\Phi}^{T} \boldsymbol{\Phi} \mathbf{W}^{T}=\boldsymbol{\Phi}^{T} \mathbf{T}
$$

where $\Phi$ is a $N \times S$ matrix with element $\Phi_{\mathrm{J}}^{\mathrm{p}}$ in the $j^{\text {th }}$ column and $p^{\text {th }}$ row.
W is a $M x N$ matrix with element $w_{k j}$ in the $k^{\text {th }}$ column and $j^{\text {th }}$ row.
T is a $N \times \mathrm{S}$ matrix with elements $t_{k}^{p}$ in the $k^{\text {th }}$ column and $p^{\text {th }}$ row.

Thus the formal solution for the weights is governed by equation 7.15.

$$
\mathbf{W}^{T}=\left[\Phi^{\mathrm{T}} \boldsymbol{\Phi}\right]^{-1} \boldsymbol{\Phi}^{T} \mathbf{T}
$$

The matrix multiplication can be simplified by calculating the psuedo-inverse of $\Phi$. If $\Phi^{\mathbf{T}} \Phi$ is non-singular (i.e. the determinant is not equal to zero) then let $\Phi^{*}=\left[\Phi^{\mathbf{T}} \Phi\right]^{-1} \Phi^{\mathbf{T}}$, then $\Phi^{*}$ is an $M x N$ matrix known as the psuedo-matrix of $\Phi$, so

$$
\mathbf{W}^{T}=\Phi^{*} \mathbf{T}
$$

Thus, the weights can be found using fast matrix inversion techniques.

### 7.5 Programming a Radial Basis Function Network

### 7.5.1 The Main Program

The main credentials of the radial basis function approach are the two stages of training. The program (rbf.c) constructed during the research of this technique is written in the C programming language and the algorithm shown in figure 7.2 explains the main loop of the program.

Incorporated within the algorithm are the following tasks:

1. The random selection of the centres from the training data set.
2. The width parameters of the radial basis functions are determined as the largest distance between the centres.
3. The radial basis function used is the Gaussian function.
4. The weights are established in training by calculating the series of matrix multiplications (equation 7.15).

## Main()

- Initialisation of all the variables and read in training / test data.
- The program then takes two directions, either training purposes or testing purposes. If the network is in training then:
- The number of radial basis functions required is entered
- Stage 1 of training commences
- Stage 2 of training commences
- Repeat until appropriate network architecture found
- If predictive mode selected, then:
- Activations of radial basis function with new inputs are calculated
- Linear sums are calculated for each output node/scheme option
- Errors calculated

Figure 7.2: Fundamentals of Operation For Radial Basis Function Code

### 7.6 The Programming Routines Required

### 7.6.1 Overview of Routines

Each of the routines and techniques used within the radial basis function network constructed are discussed in the following sections. The methodology adopted to initially set up the basic network architecture and then to calculate the radial basis function centres, width parameters, stages of training and predictive routines are explained in detail.

### 7.6.2 Initialisation of The Network

The first function within the program is to initialise the network architecture (initialisation routine). The number of input nodes and output nodes are fixed, however the user needs to define the desired number of radial basis function centres being tried.

Within the initialisation subroutine during the training phase, the training data set of fault tree structures and known best ordering schemes for the BDD are read into appropriate arrays. The scheme outputs are scaled in the range 0.001 to 0.999 , where 0.999 represents the number of nodes on the smallest BDD and 0.001 the maximum number of BDD nodes, and the rest are scaled relative to the best and worst schemes. The characteristic inputs are all scaled in the range zero to one in a manner depending on the specific characteristic (this is summarised in section 5.7.7/5.8.5). Also the user is instructed to enter a seed for the random number generator, for selection of the radial basis function centres. In the predictive phase the test data set is read into the computer program.

### 7.6.3 Programming Stage 1 of Training

The Stage_1 subroutine applies the first stage of training techniques. The routine determines the centres, the width parameters of the radial basis function and the activation's of the radial basis functions. The stage uses an unsupervised learning technique whereby only the knowledge of the input data is used.

Within this program the radial basis function centres are initially selected by choosing a random subset of the training data population. The user defines the number of radial basis function centres to be used. A random number generator incorporated into the C programming libraries was utilised and an additional routine was used to select this random number for each centre between 1 and the total number of training patterns.

When selecting the training patterns to be used as the centres it was necessary to ensure that distinct patterns were selected and not repeated patterns as this would cause problems inverting the matrix in equation 7.15.

To illustrate how the centres may have been allocated, consider the training data set of four patterns, each with 2 inputs:

$$
\begin{aligned}
& x_{1}=\{0.9,1.2\} \\
& x_{2}=\{0.8,0.9\} \\
& x_{3}=\{1.4,1.6\} \\
& x_{4}=\{3.0,0.6\}
\end{aligned}
$$

If two radial basis function centres were to be used in the network and the two training patterns randomly selected were pattern 1 and 4 then the two centres would be:

$$
\begin{aligned}
& \mu_{1}=\{0.9,1.2\} \text { where } \mu_{1 I}=0.9 \text { and } \mu_{12}=1.2 \\
& \mu_{2}=\{3.0,0.6) \text { where } \mu_{2 I}=3.0 \text { and } \mu_{22}=0.6
\end{aligned}
$$

Each centre vector $\mu_{j}$, has elements $\mu_{j i}$ which represent the connection between radial basis function centre $j$ and input node $i$. In general this relationship is represented by the network connections shown in figure 7.3.


Figure 7.3: The Radial Basis Function Connections Within The Network

Following the selection of the radial basis function centres comes the allocation of the common width parameter. There are various techniques to select the width but the initial method used was to take the largest distance between the centres. This meant that the
distance between each of the different pairings of the centres needed to be evaluated and the largest one was used as the width parameter. Using the largest width ensured that there were overlaps between the functions.

The distance calculated was that of the Euclidean squared distance. The Euclidean distance between two vectors $\underline{u}$ and $\underline{v}$ is given as:

$$
\|\underline{u}-\underline{v}\|=\sqrt{\left(u_{1}-v_{1}\right)^{2}+\left(u_{2}-v_{2}\right)^{2}+\cdots+\left(u_{n}-v_{n}\right)^{2}}
$$

Using the Gaussian basis function means that the squared width parameter is required. Therefore, given two centres $\mu_{l}$ and $\mu_{2}$ the squared distance between them can be calculated in the following manner:

$$
\underline{u}=\underline{\mu}_{1}=\{0.9,1.2\} \text { and } \underline{v}=\mu_{2}=\{3.0,0.6\}
$$

The Euclidean squared distance is:

$$
\begin{aligned}
\|\underline{u}-\underline{v}\|^{2} & =\left(u_{1}-v_{1}\right)^{2}+\left(u_{2}-v_{2}\right)^{2} \\
& =(0.9-3.0)^{2}+(1.2-0.6)^{2} \\
& =4.77
\end{aligned}
$$

This process is repeated for each pairing of the total number of centres and the largest value assigned to be the squared width parameter of each radial basis function.

Following the selection of the network parameters the activation's of the radial basis function centres at the hidden layer of the network need to be established $\left(\Phi_{j}(x)\right.$ ). This was achieved using the subroutine activation(). As the network was constructed using the Gaussian basis function $\Phi_{j}(x)=\exp \left(-\frac{\left\|\underline{x}-\underline{\mu}_{j}\right\|^{2}}{2 \sigma_{j}^{2}}\right)$ the process of calculating the activation's has three main steps:

1. Calculating the squared norm matrix $\left(\left\|\underline{x}-\mu_{j}\right\|^{2}\right)$
2. Dividing each input by 2 times the width parameter (which is already squared).
3. Exponentiating each negated value to get the activations.

The squared norm matrix refers to the calculation of the distance between the input vector(s) and the centre vector(s). This distance is calculated by using the Euclidean distance formula (equation 7.17 squared), used for the width parameters. In the resulting norm matrix the $i^{t h} j^{\text {th }}$ element is the squared distance between the $i^{\text {th }}$ row of the input data matrix and the $j^{\text {th }}$ row of the centre matrix.

For example, if the training data matrix, $T r$, has 6 training patterns with 5 inputs each, and the centre matrix, $C$, has 2 centres each with 5 inputs each. The element in the $4^{\text {th }}$ row, $2^{\text {nd }}$ column of the resulting squared norm matrix, $S N$, would be the distance between row 4 of $\operatorname{Tr}$ and row 2 of $C$. Calculated as:

If

$$
\operatorname{Tr}=\left[\begin{array}{ccccc}
0.5 & 0.7 & 0.3 & 0.6 & 0.8 \\
1.1 & 0.8 & 0.9 & 0.6 & 0.1 \\
0.7 & 0.3 & 0.7 & 0.7 & 0.2 \\
0.9 & 0.6 & 0.5 & 0.4 & 0.5 \\
1.2 & 0.4 & 0.6 & 0.9 & 0.9 \\
0.8 & 0.5 & 0.5 & 0.1 & 0.7
\end{array}\right] \quad C=\left[\begin{array}{ccccc}
0.4 & 0.3 & 0.7 & 0.8 & 0.8 \\
0.9 & 0.3 & 0.3 & 0.4 & 0.7
\end{array}\right]
$$

Then,

$$
\begin{aligned}
S N_{42} & =(0.9-0.9)^{2}+(0.6-0.3)^{2}+(0.5-0.3)^{2}+(0.4-0.4)^{2}+(0.5-0.7)^{2} \\
& =1.57
\end{aligned}
$$

After establishing the norm matrix, each element needs to be divided by 2 times the squared width parameter, and then the exponential of the negative of the resulting value
is taken to get a matrix of radial basis function activation's. If $\operatorname{Tr}$ (the matrix of training data) has dimension $M \times k$ where $k$ is the total number of inputs, and $C$ (the matrix of centres) has dimension $N \mathrm{x} k$, then the resulting matrix of activation will have dimension $M \times N$.

### 7.6.4 Programming The Second Phase of Training

The second phase of training is programmed in subroutine Stage_2. The second stage establishes the weight values between the hidden layer and the output layer. The program performs a series of matrix multiplications to calculate equation 7.15. Several separate functions are called to perform the multiplications. Each of the routines gradually breaks down the weight formula (equation 7.15) and therefore stage 2 involves taking the transpose of a matrix, the inverse of a matrix and three matrix multiplication operations.

Diagrammatically the weights in the matrix correspond to the following connections within the network architecture (figure 7.4):


Figure 7.4: Network Representation of The Weights

The performance of the trained network on the predictive test set is used to determine the adequacy of the network. If the network does not seem to perform well on the test set then a new set of radial basis function centres are selected, or the number of radial basis function centres is altered until an appropriate network architecture is found.

### 7.6.5 Programming The Predictive Phase

During this phase the initial steps are to read in the data relating to the test set of fault tree structures. This data along with the radial basis function centres, determined in the training phase, are used in conjunction to generate the activations at the hidden layer. The value of the output nodes are then calculated by generating a linear sum of the weight values (established in training) of the appropriate path from the activation at the hidden node for all the path combinations to the selected output node.

### 7.7 Generating The Desired Network Architecture

There is no simple or quick means of determining what the desired network architecture should be. The process is monotonous and involves a trial and error type approach. As the training of a network is very short a large number of network architectures can be tested in the equivalent time of only testing a few with the multi-layer perceptron approach.

In the initial experiments a network was generated to model the problem using just four characteristics to represent the fault tree structure, as used in the previous machine learning pattern recognition approach. These characteristics are:

1. Percentage of AND gates.
2. Percentage of the total number of events that were repeated.
3. The percentage of the different basic events that were repeated.

## 4. The type of top gate (AND/OR).

Further research also investigated trying to establish a model for the problem using eleven characteristics to define the fault tree structure. The additional characteristics are:
5. Number of Levels in the Tree.
6. Number of Events off Top Gate (Top Gate Inputs).
7. Number of Basic Events.
8. Maximum Number of Gates in Any Level.
9. Number of Gates with Just Event Inputs Only.
10. Number of Gates with Just Gate Inputs Only.
11. Highest number of repeated events.

As the number of inputs and number of outputs remain the same during the investigations of each set up, the only parameter to change is the number of radial basis functions to use and the random selection procedure used to allocate the function centres.

The steps to finding a model started from a simple network architecture of three radial basis function centres. This number was increased on failure of the network to perform over a large number of random selection procedures.

The training phase was carried out with the same data set as was used in the latter experiments using the multi-layer perceptron approach, and again the same set of twenty test trees was used. The correct predictions required for the twenty test trees are:

Scheme: 1, 3, 2, 1\&2, 2, 4, 2\&4, 2, 3, 1-4, 2, 3, 2, 3, 3, 2, 2, 4, 2, 4

The performance of the network was evaluated in terms of the number of times the correct response (scheme option) was selected by the network.

### 7.8 Results Using Four Characteristics as Inputs to Problem

### 7.8.1 Using Three Radial Basis Function Centres

The number of inputs for the first network was set to four, one representing each of the four fault tree characteristics used in the genetics based machine learning approach. The experiments using three centres totalled one hundred. The predictive results using the one hundred different random seeds to initialise the centres varied depending on the centres chosen. The worst performance was a prediction of five out of twenty correct, and the best thirteen out of twenty. A summary of the predictive scores is given in table 7.1, which compares models with three to seven centres (section 7.8.5).

It can be seen from table 7.1 that performance is variable across the predictive scores. The most correct predictions were made at eleven out of twenty, although a score of seven closely followed this. As the aim of the research is to identify one radial basis function network which is suitable for the problem the best result obtained is the key. Here the best predictions were achieved at thirteen out of twenty. In comparison with the genetic based machine learning approach this result is considerably better and is slightly less than the results gained using the multi-layer perceptron. From these initial predictions it is identified as a feasible method, however the level of improvement is yet to be decided. Ideally, the model would be best if it could predict all test trees correctly, hence the next goal is to pursue better performance.

### 7.8.2 Using Four Radial Basis Function Centres

To try and improve the performance of the radial basis function approach the number of radial basis function centres was increased to four. Still the number of input and output nodes are set to four and six respectively. One hundred different combinations of four centres were tested. Although the results obtained for each random selection of centres was variable, ranging from three to thirteen out of twenty correct, a larger proportion
using four centres were found at between ten and thirteen correct predictions than when using just three centres. The results are summarised in table 7.1.

Still the best performance of any network produced did not exceed the thirteen out of twenty predicted with the three centres, however the result was equalled. It is thought that by increasing the number of centres further the outputs may be distinguished between with more ease. Five radial basis function centres will be tried next.

### 7.8.3 Using Five Radial Basis Function Centres

Again, one hundred trials were carried out using five radial basis function centres. The overall performance is shown in table 7.1, where the predictive scores produced for each model are reviewed.

The best predictions made with any network using five radial basis function centres are thirteen correct schemes chosen. For 72 of the trials the performance was greater or equal to $50 \%$ of correct selections, however the number of lower predictions were spread across a greater range than when using fewer than five centres. It appears that using five radial basis functions has the potential to find a better solution and at the same time increased likelihood of producing poor models (i.e. low predictive scores). If this is the case then increasing the number of centres further should promote increased variation in the results.

### 7.8.4 Increasing Number of Centres to Six

Using six centres has produced the results shown in table 7.1. The best performance still has not exceeded thirteen correct predictions and as expected the performance is variable, with over a fifth of the results produced being less than five correct. At the same time a large number of models have produced reasonable results (ten or more). Before
considering any other possible changes to the architecture the number of radial basis function centres randomly chosen is increased to seven.

### 7.8.5 Using Seven Centres

Using seven centres has indicated further undesirable poor results. The number of trials was only 66 as up to this point it was felt that no further improvements looked likely. The number of scores predicted at ten and above has also decreased.

| Predictive Score | 3 centres | 4 centres | 5 centres | 6 centres | 7 centres |
| :---: | :---: | :---: | :---: | :---: | :---: |
| 0 |  |  |  |  | 1 |
| 1 |  |  |  |  | 1 |
| 2 |  |  |  | 2 |  |
| 3 |  | 2 | 4 | 11 | 3 |
| 4 |  |  | 3 | 1 | 3 |
| 5 | 2 |  | 4 | 10 | 10 |
| 6 |  | 1 | 4 | 5 | 3 |
| 7 | 21 | 14 | 5 |  | 2 |
| 8 | 8 | 12 | 4 | 2 | 4 |
| 9 | 11 | 6 | 5 | 5 | 3 |
| 10 | 16 | 12 | 14 | 7 | 1 |
| 11 | 32 | 39 | 32 | 24 | 23 |
| 12 | 8 | 12 | 20 | 25 | 10 |
| 13 | 2 | 1 | 6 | 8 | 2 |

Table 7.1: Predictive Results of RBF Model With Three - Seven Centres

It appears that when using seven centres the performance covers both ends of the scale, however, still the performance can not be improved beyond thirteen out of twenty correct
predictions, or a $65 \%$ chance of when subjecting the model to a new input it correctly predicting the best scheme option from the set of six alternatives.

### 7.8.6 Thirty Centres

Before dismissing the performance potential of the radial basis function model in its current set up, a large number of centres were used. The number is chosen as thirty, which is felt should be able to distinguish between the outputs if the information given by the inputs is sufficient. Fifty trials were carried out, after which it was reasoned that a good architecture might not be possible or at least particularly hard to find. Of the fifty trials the best result achieved was eleven out of twenty correct predictions. On twentynine trials the predictions made equalled zero and the remainder of trials had predictive results between three and seven. These results confirm the fact that the performance with higher numbers of centres covers both ends of the predictive spectrum, with the better predictions being harder and harder to find and the poor predictions more common.

### 7.8.7 Review of Four Characteristic Results

With the experimental models generated so far in the research it is apparent that by increasing the number of centres that are randomly chosen the performance has more potential to find a better set up at the same time with increased likelihood of poor performance. As the aim of the research is to identify a set up that models the problem of ordering the variables of a fault tree it is only one model that is required. To date the best results that have been obtained is a predictive potential of $13 / 20$, that is a $65 \%$ chance of correctly predicting which scheme option of the six alternatives is the best for a new input fault tree. This is far better than anything achieved with the genetics based machine learner and slightly better than the multi-layer perceptron approach when using only five characteristics. It is felt that by further increasing the number of centres of the model no additional performance is to be gained as can be seen from the results using thirty centres. There are alternative methods which could be adopted to chose the centres, however it is
felt at this stage that despite the selection procedure, the centres are still chosen from the data set, so the procedure is not going to improve the performance of the model if the training data set does not accurately reflect the population. Before discarding the data set as useless, it may also be a point that the information provided as inputs of the data set do not accurately reflect the pattern to be modelled. Therefore, further research is carried out using the eleven characteristic problem, which may or may not help to distinguish the differences between how to select the alternative scheme choices.

### 7.9 Results and Findings Using Eleven Characteristics as Inputs to Problem

### 7.9.1 Simple Architecture

With the number of inputs changing the simple set up of the model changes. Now there are eleven input nodes, and six output nodes. There still exists one hidden layer of nodes, which represents the radial basis function centres. The same procedure is adopted to randomly generate the centres, and the research initially focuses on trying to identify how many centres are needed to find the optimal network architecture.

### 7.9.2 Using Two and Three Radial Basis Function Centres in The Model

Initial research using the eleven input nodes for the radial basis function model focused on using two randomly chosen centres to provide a benchmark of results to compare future models. Twenty two trials were run, and all the predictive results for the twenty test trees were in the range of nine to twelve correct outcomes out of twenty. Many of the results were predicting the same scheme, scheme 2 for all trees. This indicates that there is insufficient means to differentiate between the schemes, i.e. that the two centres chosen are not enough to deal with the complexity of the problem.

To overcome this and be able to direct routes to all schemes, an additional centre was added. The results gained using three radial basis function centres are summarised in
table 7.2. The best result obtained is thirteen out of twenty correct predictions, which is the same as predicted by the best models using four characteristics only. Using this model with eleven characteristics has produced results which exceed eight correct predictions with every set of random centres produced, which seems to indicate that the model can make better predictions with these additional characteristics added.

These initial results indicate that using the eleven characteristics to dissociate the scheme choices depending on the inputs has improved success. To increase the potential of the model additional centres are added.

### 7.9.3 Adding More Centres to The RBF Model

Future research focussed on increasing the number of centres used in the model. Successive increases were made from four to ten centres. The results are compared for each different number of centres in table 7.2. Initial research involved one hundred trials for each different centre model. However, using eight radial basis functions onwards the results began to become more variable and rather than continue increasing the number of centres further, tests were carried out on the lesser number of centres, hence the difference in tallies of total trials for those models with eight or more centres (given in table 7.2).

The best result to date is now fourteen out of twenty correct predictions, hence a $70 \%$ chance of selecting the correct scheme option for a previously unseen tree. This is considerably better than using any single heuristic alone. The predictive potential has been achieved with a varied number of RBF centres. The performance is not perfect i.e. gaining a correct prediction 99 or 100 percent of the time, but it seems unlikely in the current set up that the performance is going to improve significantly. To understand the full potential of the models the accuracy of the incorrectly predicted schemes needs to be established, i.e. whether using the predictions given by the network would have drastic effects on the resulting size of the BDD produced.

| Predictive Score | $\begin{gathered} 3 \\ \text { centres } \end{gathered}$ | centres | $\begin{gathered} 5 \text { centres } \end{gathered}$ | centres | $\begin{gathered} 7 \\ \text { centres } \end{gathered}$ | centres | 9 centres | $\begin{gathered} 10 \\ \text { centres } \end{gathered}$ |
| :---: | :---: | :---: | :---: | :---: | :---: | :---: | :---: | :---: |
| 1 |  |  |  |  | 1 |  |  | 1 |
| 3 |  |  |  |  | 2 |  |  | 1 |
| 4 |  |  |  |  |  |  |  | 2 |
| 5 |  |  |  |  | 4 |  | 5 | 4 |
| 6 |  |  |  | 2 | 1 | 1 | 1 | 2 |
| 7 |  |  | 1 | 3 | 1 | 1 | 3 | 4 |
| 8 | 1 | 2 | 1 |  | 3 | 1 | 1 | 5 |
| 9 | 2 | 2 | 4 | 6 | 8 | 8 | 2 | 7 |
| 10 | 4 | 21 | 21 | 25 | 24 | 6 | 15 | 21 |
| 11 | 32 | 73 | 51 | 61 | 72 | 25 | 31 | 28 |
| 12 | 55 | 204 | 206 | 167 | 151 | 42 | 35 | 22 |
| 13 | 6 | 77 | 49 | 71 | 67 | 16 | 7 | 3 |
| 14 |  | 5 | 2 |  |  |  | 1 |  |
| Total trials | 100 | 335 | 335 | 335 | 335 | 100 | 100 | 100 |

Table 7.2: Comparative Predictions For Models With 4-10 Centres

### 7.10 Assessing The Accuracy of The Best Networks

A number of network architectures were generated, trained and tested. The best performance produced a score of fourteen out of the twenty test trees whose schemes were selected correctly by the network. This level of performance was achieved on eight different network architectures as shown in table 7.3. All the networks had eleven input nodes and six output nodes, with the number of centres being the variable parameter. The exact network details for all eight architectures are given in Appendix V.

In terms of performance any one of the network architectures shown in table 7.3 could be used to generate the $14 / 20$ correct responses. In terms of efficiency the network with the
smallest number of radial basis function centres should be used, as computer requirements are reduced. Hence, any one of the five networks using four radial basis function centres would be the most efficient solution.


Table 7.3: Predictions Made by Best Network Architectures

As the networks do not have a 100 percent success rate at producing the best scheme, the predictions for the incorrect schemes need to be examined.

| Tree | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 0 | 9 |
| :---: | :---: | :---: | :---: | :---: | :---: | :---: | :---: | :---: | :---: | :---: |
| Best | 1 | 3 | 2 | $1 / 2$ | 2 | 4 | $2 / 4$ | 2 | 3 | $1-4$ |
| $2^{\text {nd }}$ | 2 | 2 | 4 | $3 / 4$ | 1 | 3 | $1 / 3$ | 1 | $1 / 2 / 4$ | $5 / 6$ |
| $3^{\text {rd }}$ | 4 | 1 | 1 | 6 | $3 / 4$ | 2 | 6 | 4 | $5 / 6$ |  |
| Tree | 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 | 19 | 20 |
| Best | 2 | 3 | 2 | 3 | 3 | 2 | 2 | 4 | 2 | 4 |
| $2^{\text {nd }}$ | 1 | 4 | 1 | 2 | $1 / 2$ | 1 | 4 | 2 | 1 | 3 |
| $3^{\text {rd }}$ | 4 | 1 | 4 | 1 | 4 | 4 | 1 | 3 | 4 | 2 |

Table 7.4: First to Third Best Choices of Scheme Option For Test Trees

The predictions for each of the networks on the same set of twenty fault tree structures are given in table 7.3. The incorrect responses are highlighted. Table 7.4 shows the best, second best and third best scheme options for each of the test trees. In the majority of instances of the six schemes that are incorrectly predicted (using the best architecture that predicts fourteen out of the twenty trees tested correctly) five out of the six ordering heuristics that are selected using this network approach are the second best option.

For example, using the network architecture generated using the random number of 638 to select radial basis function centres, and having four chosen centres the predictions for the twenty test trees are shown in figure 7.5. It is clear that five out of the six incorrect predictions are near to best scheme coding, and testing on the other network architectures (mentioned in section 7.10) the same result is evident. Looking at the values of the second best option in comparison to the best, there is little difference in the coded values, indicating that the second best option is not dramatically different, or the resulting BDD structure is not considerably larger than the optimal structure. Table 7.5 shows the coded comparative results for the second and third best scheme option. The first options have values of 0.999

| Tree | 1 | 2 | 3 | 4 | 5 | 6 | 7 |  | 9 | 10 |
| :---: | :---: | :---: | :---: | :---: | :---: | :---: | :---: | :---: | :---: | :---: |
| $2^{\text {nd }}$ | 0.932 | 0.992 | 0.976 | 0.979 | 0.997 | 0.998 | 0.985 | 0.998 | 0.998 | 0.001 |
| $3^{\text {rd }}$ | 0.835 | 0.979 | 0.947 | 0.079 | 0.994 | 0.994 | 0.381 | 0.951 | 0.001 |  |
| Tree | 11. | 12 | 13 | 14 | 15. | 16 | 17. | 18 | 19 | 20 |
| $2^{\text {nd }}$ | 0.962 | 0.990 | 0.988 | 0.996 | 0.994 | 0.997 | 0.998 | 0.961 | 0.962 | 0.875 |
| $3^{\text {rd }}$ | 0.889 | 0.979 | 0.727 | 0.989 | 0.983 | 0.992 | 0.997 | 0.947 | 0.902 | 0.671 |

Table 7.5: Coded Comparison of First, Second and Third Best Scheme Options

Hence, it is clear that this neural network technique could be used to solve the ordering problem, with optimal BDDs being produced approximately $70 \%$ of the time and for the remainder 30\% a near optimal BDD structure should be produced, hence facilitating an efficient analysis.

## $\sqrt{ }$ correct prediction



Best Scheme Actually: 1 Answer Given: 2nd Best ding for Scheme $1=0.999$ ding for Scheme $2=0.932$

Best Scheme Actually: 3
Answer Given: 2nd Best
Coding for Scheme $3=0.999$
Coding for Scheme $2=0.992$

Best Scheme Actually: 3 Answer Given: 2nd Best
Coding for Scheme $3=0.999$
Coding for Scheme $2=0.996$
Best Scheme Actually: 3 Answer Given: 2nd Best Coding for Scheme $3=0.999$ Coding for Scheme $1=0.994$

Best Scheme Actually: 4 Answer Given: 3rd Best Coding for Scheme $4=0.999$
Coding for Scheme $2=0.671$
Best Scheme Actually: 4
Answer Given: 2nd Best
Coding for Scheme $4=0.999$
Coding for Scheme 2 $=0.961$

Figure 7.5: Findings of The Incorrect Predictions

### 7.11 Next Research Steps

### 7.11.1 Reducing Width Parameter

Further improvements in the predictive potential of the RBF model still need to be sought. An avenue of possible change is the network parameters themselves. One parameter to alter is the width or spread of the radial basis function. Currently this is set to be the largest distance between all the centres. At this setting there will be a considerable amount of overlapping of the search space between the radial basis functions. Ideally, given a new input the output would be influenced by a single radial basis function activation, however, when the RBFs have a large overlap this influence is spread between a number of RBF activations. The whole search space needs to be covered, but not with extreme over lapping and at the same time with sufficient spread to cover most of the area (which is not possible with too small a spread parameter and too few functions). There needs to be a balance between the number of activations and the spread of each.

Hence, the next research steps looked at the influence of reducing the width parameter. The following trials were executed using a width parameter of half the largest distance between the centres. The network architectures examined involved eleven inputs, six outputs and between four and seven radial basis functions. The results are summarised in table 7.6. For each specified number of centres one hundred trials were carried out. The results are similar to those using the 'largest distance between centres' as the width parameter. The best result of fourteen out of twenty correct predictions was again achieved, using a network with five centres. No further improvements were made upon this accuracy. A larger number of centres were tried to check that no further improvements could be made. As shown in table 7.6 using twelve centres produced a more varied response with no overall improvements.

| Predictive | 4 centres | 5 centres | 6 centres | 7 centres | 12 centres |
| :---: | :---: | :---: | :---: | :---: | :---: |
| Score |  |  |  |  |  |
| 4 |  |  |  |  | 3 |
| 5 |  |  |  |  | 1 |
| 6 |  |  |  |  | 2 |
| 7 |  |  |  | 1 | 1 |
| 8 | 1 | 1 | 1 | 1 | 3 |
| 9 | 6 | 2 | 1 | 3 | 16 |
| 10 | 12 | 11 | 18 | 13 | 22 |
| 11 | 27 | 26 | 19 | 21 | 31 |
| 12 | 51 | 48 | 41 | 43 | 15 |
| 13 | 3 | 11 | 20 | 19 | 5 |
| 14 |  | 1 |  |  |  |
| Total trials | 100 | 100 | 100 | 100 | 100 |

Table 7.6: Comparative Predictions For Models With 4-7 Centres

The best network architecture using this reduced width parameter has the following characteristics:

## Random Number seed: 42

Number of Centres: 5

Centres were randomly selected as:
$\underline{\mu}_{1}=\{0.604,0.088,0.667,0.001,0.154,0.769,0.220,0.197,0.103,0.471,0.263\}$
$\underline{\mu}_{2}=\{0.386,0.088,0.667,0.999,0.385,0.462,0.209,0.197,0.069,0.441,0.211\}$
$\mu_{3}=\{0.663,0.088,0.756,0.999,0.385,0.385,0.203,0.213,0.001,0.382,0.263\}$
$\mu_{4}=\{0.396,0.882,0.600,0.999,0.231,0.308,0.030,0.033,0.001,0.059,0.158\}$
$\underline{\mu}_{5}=\{0.297,0.001,0.001,0.999,0.385,0.308,0.660,0.660,0.834,0.147,0.001\}$

## Width Parameter: 1.075507

Weights from output node $k$ to each centre (including bias centre node):

$$
\left.\begin{array}{l}
k=1: \quad\{0.005360,0.641249,0.884705,0.319664,-1.008781,0.156937\} \\
k=2: \quad\{0.230803,0.595655,0.866074,-0.630348,0.315994,-0.050278\} \\
k=3: \\
k=4: \\
k=50.054871,0.730951,-1.826630,1.288010,0.553288,0.700058\} \\
k=5: \quad\{0.161022,0.615295,0.431763,-0.848396,0.253029,0.697006\} \\
k=6:
\end{array}\{-0.008962,-0.1434475,-2.014668,0.659339,1.806165,-0.183221\}\right\}
$$

Predictions: 2, 2, 2, 2, 2, 4, 2, 2, 4, 2, 2, 3, 2, 3, 1, 2, 2, 2, 2, $2=14 / 20$

For this problem the reduction in the spread of the RBF did not lead to an increase in predictive potential of the model. Either the complexity of the problem warrants a more complicated centre selection procedure and width allocation method to find the pattern in the data or the data needs to be examined itself.

### 7.11.2 Using Only Four Outputs to The Model

There are infinitely many network architectures still to try but whether the results are important is debatable. There are two possible alternatives that can be seen as a way forward, one is relating to the inputs of the problem, the other to the outputs. The
predictive potential of the RBF network can only be gained if the training data set accurately reflects the population. On reviewing the data it was observed that schemes four and five, namely those using the priority depth-first and modified priority depth-first approach yielded best BDDs on only a minority of occasions. Hence, for continuing research these have been deleted, and the potential of the radial basis function model has been tested using eleven inputs and four outputs.

Within the training and test data set there are fault trees whose structure warrants the use of a number of ordering heuristics with the same sized BDD resulting. As the network is now trying to pick the best of four scheme preferences, any trees with equal BDDs for each of these four schemes on conversion have been removed, as no information is provided from this data. Thus, the training data set is reduced to 171 trees. One tree was removed from the test set and an additional one from the training set included, hence still twenty trees exist within the test set. Therefore the target predictions needed by the network are:

$$
1,3,2,1 / 2,2,4,2 / 4,2,3,2,3,2,3,3,2,2,4,2,4,3 .
$$

The RBF model is generated as in the original set up with the width parameter set to the largest distance between centres as using the alternative half largest distance proved to be no more effective. Four, five and six radial basis function centres were tried as the optimal network architecture. Again one hundred trials were carried out for each specified number of centres and the results are given in table 7.7.

The results are similar for all three network architectures, with the best prediction being fourteen out of twenty of the test trees being correctly chosen. It is not felt likely that future improvements would be made by increasing the number of centres further and instead examination of the inputs may provide a better way forward to solving this problem.

| Predictive <br> Score- <br> Scentres | 5 centres | 6 centres |  |
| :---: | :---: | :---: | :---: |
| 7 | 2 |  | 2 |
| 8 | 1 | 2 | 2 |
| 9 | 7 | 6 | 4 |
| 10 | 11 | 13 | 8 |
| 11 | 37 | 27 | 27 |
| 12 | 27 | 37 | 31 |
| 13 | 11 | 13 | 23 |
| 14 | 4 | 2 | 3 |
| Total trials | 100 | 100 | 100 |

Table 7.7: Results For The Four Output Problem

### 7.11.3 Importance of Characteristics

Rather than continuing to test the RBF models with different training procedures it is felt that the most benefit could be found in scrutinising the population inputs. What needs to be determined is whether the characteristics currently chosen to represent the fault tree structure are relevant or not to the ordering dilemma. If the inputs do not represent those factors which dissociate the fault tree from the different schemes then regardless of the training procedure employed no pattern will be able to be found. To examine the importance of the characteristics the radial basis function network approach (as well as the multi-layer perceptron approach) can be used by reviewing the weight values and activations connecting the input and output nodes. It is suggested that pursuing this research should initially centre on the importance of the input characteristics.

### 7.12 Summary

- Radial Basis Function Neural Networks have been shown to have the potential to model the variable ordering problem.
- The model which used eleven fault tree characteristics to dissociate the six scheme alternatives produced the best results.
- Eight network architectures have been generated which produce the best result so far of 14 out of 20 correct predictions.
- The most efficient and best RBF network is one with four RBF centres. Five architectures have been found. One possible set up to use when trying to find the best scheme for a new fault tree is:


## Input nodes: 11

Output nodes: 6
Centres (4):
$\underline{\mu}_{1}=\{0.644,0.235,0.733,0.001,0.538,0.462,0.489,0.459,0.138,0.971,0.263\}$
$\mu_{2}=\{0.624,0.059,0.644,0.001,0.308,0.462,0.154,0.115,0.034,0.206,0.158\}$
$\mu_{3}=\{0.416,0.059,0.711,0.001,0.462,0.385,0.129,0.131,0.103,0.294,0.263\}$
$\underline{\mu}_{4}=\{0.465,0.029,0.600,0.001,0.462,0.385,0.102,0.066,0.001,0.206,0.158\}$
Width Parameter: 1.023383
Weights from output node $k$ to each centre (including extra bias node):
$k=1:\{0.627533,0.040619,-0.944214,2.572754,-1.308350\}$
$k=2:\{0.728906,-0.285797,-0.712158,2.209473,-0.995605\}$
$k=3:\{0.645231,-0.158051,0.200012,-0.248779,0.456787\}$
$k=4:\{0.659595,0.432404,-0.780762,-2.036621,2.743286\}$
$k=5:\{0.261979,-0.376621,3.887093,-2.044250,-1.780518\}$
$k=6:\{0.167886,-0.868729,1.308273,1.308136,-1.978363\}$

- Of the 6 incorrect predictions produced using the optimal network architecture, in the majority of instances (five out of the six) the second best ordering option is predicted, with the difference in the size of the resulting BDD being small.
- The outputs of the model were scrutinised and a network produced using four scheme outputs. Again, the chance of selecting the correct scheme choice for a previously unseen tree was 70 percent.
- To further enhance the performance a more complicated method for selecting the centres and width parameters of the radial basis function could be considered, or alternatively a different type of radial basis function could be tested.
- Further improvements are thought more feasible by examining the characteristics used to model the fault tree in terms of their influence in selecting an ordering heuristic. This can be achieved using the radial basis function by examining the weight connections between the inputs. This topic is the suggested for future work in this area.


## 8 Using Structural Importance Measures For BDD Variable Ordering

### 8.1 Introduction

The procedures researched so far for selecting an ordering heuristic that produces an optimal or good variable ordering for the conversion of a fault tree to a Binary Decision Diagram concentrates on selecting from a set of alternatives. The pattern recognition approaches of the machine learning classifier system and neural networks have been used to identify the best of a set of six ordering alternatives for any given fault tree structure. Characteristics of the fault tree have been used as inputs to predict the best scheme.

The problem of using these pattern recognition approaches is the considerable amount of computation time required in initially finding the best classifiers or neural network structure. The selection of a specific ordering method is made from a specified list of heuristics. Each of these heuristics or rules were taken from approaches suggested in the literature and all provide a very structured approach to the ordering, i.e. components that are inputs to the same gate are ordered next to each other in the ordering list. There is no random selection of components, i.e. components are not selected from opposite sides of the tree and ordered next to each other in the ordering list. There is no indication that components need to be selected and grouped according to their neighbourhood. What appears to be important is the contribution any event makes to the minimal form of the fault tree logic expression. This contribution is represented by the structural importance measure ordering heuristic, to be discussed.

The ordering that is required is not related to any physical properties of the system, i.e. an ordering of $A<B$ does not mean that if component $A$ fails component $B$ must fail. The ordering is purely based on the position of the components within the tree structure and
accounts for the Boolean reduction process. An ordering is required to rank components in terms of their significance within the system, that is their contribution to the top event occurrence, and also their influence in relation to the other components also within the system. One measure that incorporates these qualities is the structural importance measure of a component.

In this chapter the use of the components structural importance within the tree structure has been used to create an ordering of the components or basic events ${ }^{[B A n 991]}$. This ordering has then been used to produce the BDD. The number of nodes in the BDD resulting from the conversion is used as a comparative measure. The best scheme option from the set of six alternatives used in the pattern recognition approaches is compared to the structural importance measure ordering. Results using the mathematical formula (implemented using the BDD itself) have been very convincing, with a large proportion of the trees producing a BDD with equal or less nodes. Applying the formula to the fault tree is computationally intensive and hence a simplified approximation has been generated. This new approximated structural method produces similar importance measures and related orderings and ultimately produces an optimal or near optimal BDD for the majority of fault tree structures ${ }^{[B A n 003]}$.

### 8.2 Problems With Re-writings of The Fault Tree Structure

All of the heuristics mentioned so far suffer from the problem that if the fault tree structure is re-written, drawn with the order of the gate inputs changed, then applying the same heuristic will result in different orderings and hence different BDDs. Applying the top-down, left-right approach to the two trees in figures 8.1 and 8.2, which represent the same top event logic function, the difference in variable orderings can be seen.

Thus, using any one of the heuristics mentioned so far will not produce a unique BDD for a given fault tree, the result will depend upon how the engineer has represented the function. Therefore, how the fault tree is drawn may influence the type of ordering heuristic that results in the smallest BDD. As the pattern recognition approaches are
using the best scheme option for a given fault tree as a guide to future predictions of the best scheme for new trees this variation gives 'noise' in the data. Thus, the pattern recognition approaches are then trying to establish trends from this noisy data. In terms of the neural network approach, it is very important that the network does not become over trained, thus learning these trends, hence reducing the generalisation performance of the network.


To irradicate these re-writing problems a heuristic needs to be used which is not affected by how the tree is represented. Using the structural importance measure means that regardless of how the tree is drawn, it is the logic function that is used to establish the weighting values for each component.

### 8.3 Definition of Importance Measures

### 8.3.1 Understanding Importance Measures

A very useful piece of information, which can be derived from a system reliability assessment, is the importance measure for each component or minimal cut set. For each
component its importance signifies the role that it plays in either causing or contributing to the occurrence of the top event. This role is given a rank in terms of a numerical value.

Importance measures can be categorised in two ways: (1) deterministic; and (2) probabilistic. The probabilistic measures can themselves be categorised into those which are appropriate for system availability assessment (top event probability) and those which are concerned with system reliability assessment (expected number of top event occurrences). These have all been discussed in chapter 2.

As the ordering does not depend on the probabilistic failure characteristics of the components just it's position in the tree, the deterministic structural importance measure is analysed and discussed as a potential ordering mechanism.

### 8.3.2 Deterministic Measures of Importance

Deterministic measures assess the importance of a component to the system operation without considering the component's probability of failure. One such measure is the structural measure of importance, SMI, which is defined for a component, $i$, as

$$
S M I_{i}=\frac{\text { number of critical system states for component } i}{\text { total number of states for the }(n-1) \text { remaining components }}
$$

A critical state for component $i$ is a state for the remaining ( $n-1$ ) components such that a failure of component $i$ causes the system to go from a working state to a failed state.

### 8.4 Calculation Method For Deterministic Structural Importance Measure

To illustrate this structural importance measure, consider the fault tree drawn in figure 8.3.

The logic expression for the top event is:

$$
T O P=A+B . C
$$



Figure 8.3: Simple Fault Tree Structure

Therefore, the top event will occur (or top event failure) if $A$ occurs or both $B$ and $C$ occur. To generate a variable ordering the structural importance measures of each component need to be calculated. The procedure to carry out this process can be simplified as follows:

For each component:

1. Find the possible states for the remaining components.
2. Test whether each of the remaining states are critical for the chosen component.

Taking component $A$ from figure 8.3 as an example, there are four states for the remaining components these are:

1. $B$ working and $C$ working $(\overline{B C})$
2. $B$ failed and $C$ working $(B \bar{C})$
3. $\quad B$ working and $C$ failed $(\vec{B} C)$
4. $\quad B$ and $C$ failed ( $B C$ )

To explain the theory behind a critical state, each of the four states for the remaining components needs to be examined. If component $A$ is working and given the states of the remaining components the system works (top event non-occurrence) then this reflects the possibility of a critical state. The determining factor is whether failure of component $A$ causes the system to fail. If it does then this is referred to as a critical state for component $A$.

Given one state for the remaining components, namely B working and C working $(\overline{B C})$, with component $A$ working $(\bar{A})$ the system would be working. If component $A$ then failed the system would fail and this can be defined as a critical state (see column 2, row 2 of table 8.1).

Therefore, the structural importance measure for component $A\left(S M I_{A}\right)$ is calculated as shown in table 8.1.

| States for other components | Critical State for $A$ |
| :---: | :---: |
| $\bar{B}, \bar{C}$ | Yes |
| $B, \bar{C}$ | Yes |
| $\bar{B}, C$ | Yes |
| $B, C$ | No |

(NB. the ${ }^{-}$means component works)
Table 8.1: Calculation of The Structural Importance Measure For Component $A$

There are four states for the remaining components, of which three of these are critical for component $A$, hence $S M I_{A}=3 / 4$.

The same process is repeated for all of the other components, in this example variable $B$ and $C$. Hence, the structural importance measure of component $B\left(S M I_{B}\right)$ is calculated as shown in table 8.2:

| States for other components | Critical State for $B$ |
| :---: | :---: |
| $\bar{A}, \bar{C}$ | No |
| $A, \bar{C}$ | No |
| $\bar{A}, C$ | Yes |
| $A, C$ | No |

Table 8.2: Calculation of the Structural Importance Measure for Component $B$

Hence, $S M I_{B}=1 / 4$.

The structural importance measure of the final component $C$, (SMIC) is equal to $1 / 4$, like $S M I_{B}$ (as shown in table 8.3).

| States for other components | Critical State for $C$ |
| :---: | :---: |
| $\bar{A}, \bar{B}$ | No |
| $\bar{A}, B$ | Yes |
| $A, \bar{B}$ | No |
| $A, B$ | No |

Table 8.3: Calculation of the Structural Importance Measure For Component $C$

On gaining each of the importance measures, the remaining factor is to order the components in descending order depending on the values calculated.

### 8.5 Alternative Methods of Calculating The Structural Importance Measure

### 8.5.1 Problems With The Hand Calculated Approach

To calculate the structural importance measure for all the components within a fault tree as illustrated previously even for small trees is relatively time consuming. This is ever more prevalent with large fault tree structures where the number of possible combinations of the working and failed states of the components is exponentially increasing with respect to the number of components.

Programming the method utilised in section 8.4 would not be very efficient, as the process would require the following:

1. The logic expression for top event.
2. For each component, the program would need to repeatedly substitute in values for the remaining components in the system, for every state combination.

Whilst simple in concept it is computationally time consuming to perform the procedure for each variable.

### 8.5.2 Using Birnbaums's Structural Importance Measure

The probabilistic importance measure of Birnbaum, namely Birnbaums Measure of Criticality has been mentioned in chapter 2. Lambert ${ }^{[\text {Lam } 75]}$ stated that this probabilistic measure could be used to evaluate the structural importance measure. However, this still requires the system probability function $Q(q)$ or an approximation of it.

Birnbaums measure of criticality $\left(G_{i}(q)\right)$ is defined as:

$$
G_{i}(q)=Q\left(1_{i}, q\right)-Q\left(0_{i}, q\right)
$$

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

$$
Q\left(1_{i}, q\right)=\left(q_{1}, q_{2}, \ldots . ., q_{i-l}, 1, q_{i+l}, \ldots \ldots . q_{n}\right)
$$

$Q\left(0_{i}, q\right)=\left(q_{1}, q_{2}, \ldots . ., q_{i-1}, 0, q_{i+1}, \ldots \ldots . q_{n}\right)$. 8.3

From Lamberts ${ }^{[\text {Lam75] }}$ paper it states that if the probability of failure of component $i, q_{i}(t)$, is set equal to $1 / 2$ for all $i \neq j$, then the number of states in which component $i$ is critical, denoted by $B_{i}$, is defined as:

$$
B_{i}=\left\{Q\left(1_{i}, \underline{1 / 2}\right)-Q\left(0_{i}, \underline{1 / 2}\right)\right\}
$$

Implementing this numerically for the tree in figure 8.3, the top event probability expression is given as:

$$
Q(q)=q_{A}+q_{B} q_{C}-q_{A} q_{B} q_{C}
$$

Calculating the structural importance measure for $A$;

$$
\begin{aligned}
& Q\left(1_{A}, q\right)=1 \\
& Q\left(0_{A}^{A}, g\right)=q_{B} q_{C}
\end{aligned}
$$

Therefore, $B_{A}(q)=1-q_{B} q_{C}$, and $B_{A}(\underline{1 / 2})=1-(1 / 2.1 / 2)=3 / 4$.

The same principle is applied to $B$ and $C$, resulting in $B_{B}(\underline{1 / 2})=B_{C}(\underline{1 / 2})=1 / 4$.

The Birnbaum measure of structural importance can also be found by calculating the differential of the top event probability expression with respect to the component. The importance value being calculated by equating all the remaining components with the probability of $1 / 2$.
i.e.

$$
Q(q)=q_{A}+q_{B} q_{C}-q_{A} q_{B} q_{C}
$$

$$
\Rightarrow \quad B_{A}(\underline{q})=\frac{\partial Q_{S Y S}(t)}{\partial q_{A}}=1-q_{B} q_{C}=1-\left(\frac{1}{2} \cdot \frac{1}{2}\right)=\frac{3}{4}
$$

This is the same as using equation 8.4.

### 8.6 Application of Numerical Structural Importance to The Ordering Problem

### 8.6.1 Programming The Calculation Procedure

To test the potential of the component structural importance method to the problem of generating a variable ordering heuristic to yield an optimal BDD the Birnbaum structural importance measure was used. Fortran code was available to produce the Birnbaum Measure of Criticality from the BDD. As initially it is the validity of the measure that is being established and not the efficiency of the technique then a BDD, which has been constructed using a different variable ordering, has been used to gain the importance measures.

The program strimpgsq.f was created which reads in the fault tree structure from a data file, and using Birnbaums structural importance measure yields the appropriate strengths for each component. All the components were then ranked in descending order (highest to lowest) in accordance with their structural importance value, hence producing an ordering for the variables of the fault tree. For components with the same resulting structural measure the tie was broken by ranking the component that lies higher up in the tree structure (using the top-down, left-right approach) first.

To establish the influence of the new ordering permutation a comparison was made with the best of the six previously identified alternative schemes (Andrews and Bartlett, $1998^{[\mathrm{ABa} 98]}$ ). Each ordering permutation was output to a file $*$.bsi, which was read into a
separate program which converts the fault tree structure into a BDD. The program calculates the number of nodes within the BDD. It is the number of nodes before minimisation of the BDD structure that are used in the comparison process. The reason for this was that the smaller the initial BDD, from which the quantification process can be carried out, the more efficient the quantification process, and also to determine the minimal cut set less minimisation needs to occur if the BDD is initially smaller.

### 8.6.2 Results of Comparison

In table 8.4 three groupings relating to the number of nodes in the BDD have been identified for the comparison. These are less than the previous best, equal to the previous best, and greater than the previous best. For this method to be successful then the majority or all of the trees when converted using this new ordering need to result in a BDD of the same or smaller dimension than the previous best of the set of six alternatives. The fault trees generated for use in the neural network study (both the training and test trees) and some additions have been combined to produce a data set for this ordering heuristic comparison. Thus, two hundred and twenty five trees were compared and the results are shown in table 8.4. The results for each fault tree are shown in Appendix VI.

| $\begin{aligned} & \text { Nodes in comparison } \\ & \text { (to best) } \end{aligned}$ | No. of trees | $\%$ of trees | $\text { Total }=K$ |
| :---: | :---: | :---: | :---: |
| = | 77 | 34.2 |  |
| $<$ | 96 | 42.7 | 76.9 \% |
| > | 52 | 23.1 |  |

Table 8.4: Results of Comparison of Structural Importance Ordering and Previous Best Ordering on BDD Size

From these results, it is concluded that approximately $77 \%$ of all the trees tested within the data set, using the structural importance ordering yields a BDD of equal or smaller dimension than the previous best scheme ordering.

From the set of six orderings the distribution of 'best schemes' is illustrated in table 8.5. The total number of trees that each scheme predicts the best result for is greater than 225 in total as some schemes produce a BDD of equal size to another scheme. The best scheme out of the six for overall performance is the modified top-down, left-right approach, although the winning margin is very small. This new structural importance ordering heuristic clearly outperforms all of these six schemes individually and more research is needed to unravel it's full potential and try to establish an efficient method of calculation.

| Ordering Heuristic | Number of Instances Minimal BDD Produced | Percentage of times best BDD produced |
| :---: | :---: | :---: |
| Top-down, left-right | 87 | 15.4 |
| Modified Top-down, left-right | 169 | 29.8 |
| Depth-First | 120 | 21.2 |
| Modified Depth-First | 117 | 20.6 |
| Priority Depth-First | 36 | 6.3 |
| Modified Priority Depth-First | 38 | 6.7 |

Table 8.5: Performance of Six Different Ordering Heuristics in Producing BDDs

There are two routes of progression with this new measure. One is to try and improve the percentage of trees whose BDD size are an improvement or equal to the best of the six alternatives and the second is to use the method to produce a defined ordering heuristic. The following section looks at improvements that could be made to the measure by altering the method used to order components with matched importance measures. Section 8.8 onwards focuses on methods to generate the same ordering by a computationally simpler procedure.

### 8.6.3 Re-Ordering Components With Matched Structural Importance Measures

Of the trees whose structural importance measure for the components yielded an ordering which resulted in a larger BDD structure than the previous best the number of nodes difference between the two values varied considerably. Some of the values were only a few nodes larger than the previous best whereas others where much larger (as can be seen in Appendix VI).

If the structural importance of a component is the same as another then the component which lies higher up the tree (nearer the top event) is ordered first. An alternative way which may improve upon the performance of the structural importance measure ordering heuristic is to look at the matching components and see if a better method can be found for determining the order of these components with equal structural importance.

The method investigated was to order the matched components based on the most repeated event being ordered first, and so on in descending order of repetition. If again there were still matches the tie would be broken as before. A program strrep.f was created, based on strimpgsq.f, which has an extra routine to calculate the number of times a component is repeated. Based on these values of repetition the matched components are separated. If the components are repeated the same number of times then the program switches to the previous routine of ordering the components in a top-down, left-right manner from the tree structure. The ordering was output to a file *.bri.

Each ordered data file *.bri of components was run through the program (schemes. $f$ ) which converts the fault tree to the BDD and calculates the number of nodes. Comparing the results to those of the original structural importance ordering, the findings are as follows:

1. The total number of fault tree structures and orderings tested was 225 .
2. Previous results (Birnbaums structural importance ordering) produced:
a. 52 trees with nodes greater than the six scheme previous best
b. 77 trees with nodes equal to the six scheme previous best.
c. Therefore, 129 trees with possible improvement to be made.
3. 152 trees out of the 225 tested had orderings that were changed due to this new repeated event routine.
4. Of the 52 trees that were greater than the previous best using original structural importance heuristic the ordering of 26 remained unchanged.
5. Of the 76 trees with equal BDD sizes, 46 of these had their ordering changed but only 2 resulted in changes in BDD size, both being larger than when using standard structural importance measure. The lack of change is possible due to many of these trees already being minimal hence further reductions in size are not possible.
6. Of the 152 trees whose ordering was changed, 15 trees resulted in differences in the number of nodes for the BDD. These can be broken down as follows:

- For 2 trees the number of BDD nodes increased. Previously these were greater than the best, hence there is no effect on the previous results.
- For 2 trees the number of BDD nodes increased, and these were previously equal to best. Therefore, this is a worse result for the performance of the repeated event version of the structural importance heuristic.
- For 2 trees the number of nodes increased. These were less than previous best using standard importance measure, and still less than previous best, therefore no change in result.
- For 4 trees the number of nodes decreased, but already less than previous best, so no alteration in result.
- For 2 trees the number of nodes decreased. Previously the result was greater than the best of the six alternatives and this result is still greater so again no change in performance.
- For 3 trees the number of nodes decreased. Previously the trees were greater than the best and now less than previous best, an improvement in performance.

The differences made to the 152 trees affected by the new repeated aspect of the structural importance measure are given in Appendix VII. The results gained using this additional routine to separate matched components are summarised in table 8.6.

| Nodes in comparison (to best) | No. of trees | \% of trees | $\text { Total }=1<$ |
| :---: | :---: | :---: | :---: |
| $=$ | 75 | 33.3 |  |
| < | 99 | 44.0 | 77.3 \% |
| > | 51 | 22.7 |  |

Table 8.6: Results of Comparison of Structural Importance Ordering and Previous Best Ordering on BDD Size With Repeated Routine

For the trees that were altered by the new routine the amount of change in the number of BDD nodes was very small. Despite the small improvement in overall performance of the routine ( $77.3 \%$ ), not all the orderings for the trees were altered by the repeated routine and only a very small proportion of the trees with a change resulted in differences in the size of the BDD produced. Therefore, although the process did have a slight improvement in performance, another method may be more beneficial in reducing the number of trees that currently have nodes greater than the best of the previous six alternative scheme approaches i.e. finding another mechanism for matching components or an alternative structural importance measure. Despite some of the BDDs being larger, the overall performance of the measure is better than a single heuristic used in the comparative research so far.

### 8.7 Using The Approach As it Stands

Having shown that the importance measure produces a good ordering heuristic, investigations now focus on trying to establish an efficient way to calculate it. Whilst the
performance of the measure was being evaluated the structural importance heuristic was found from the BDD. As the aim of the ordering heuristic is to find an ordering to generate an optimal BDD then this technique is not ideal. However, the results using this approach are better and more productive in the long run than current methodologies, as will be explained.

A research group in France, ARALIA ${ }^{[G A-55]}$, has constructed a BDD package that considers this ordering dilemma. Their response to the problem is to select an ordering heuristic from a set of alternatives and draw the resulting BDD, then select a different ordering heuristic and redraw the diagram. The process is carried out until the best alternative is found since the fault tree to BDD conversion process implemented within the program package is very fast. Therefore, this new approach of using the structural importance measure allows a good ordering to be generated which has been proven on a large number of trees to be better or equal to the best of a set of six alternatives. A reasonable solution is expected and the processing time to get this should on the majority of occasions be shorter than randomly selecting ordering heuristics.

The best way to minimise the size of the first BDD from which to derive the structural importance measures of the components in the tree is to use an ordering that is relatively efficient at producing near minimal BDD structures. Overall the best heuristic that produces the ordering to generate the smallest BDD is the modified top-down, left-right ordering (table 8.5). Hence, a solution to the problem would be to use the modified topdown, left-right approach to generate the original BDD, to then find the structural importance measures to produce the final BDD structure.

### 8.8 Alternative Methods For Calculating The Structural Importance Measure

### 8.8.1 Highlighting Problems in Coding The Mathematical Measure From The Fault Tree

Alternative ways of deriving the importance rankings have been investigated in an attempt to produce a more efficient process. The program used to establish the suitability of the structural importance measure ordering theory needs a BDD to produce the appropriate values. Following this an ordering is produced, which is used to redraw the BDD. Hence, for this ordering heuristic to be a practical proposition a program needs to be constructed to calculate the structural importance directly from the fault tree structure. There are however a number of problems with the mathematical procedure required to calculate the measures directly and these can be summarised as:

- An excessive amount of computer processing power would be required to search for all the possible combinations of states for the remaining components and the effect that failure of the selected component would then have on each of these states. For example, the number of combinations is calculated by the formula $2^{n-1}$, where $n$ is the total number of components. If the number of components is 12 then the number of combinations for the remaining $n-1$ components is $2^{11}=2048$ or for 63 components the combinations of the $n-1$ components is $2^{63}=$ $9.22337 \mathrm{E}^{18}$. Plus, this is just the number of combinations for one component and it needs to be done for each of the $n$ components, therefore the resulting number or combinations is $n 2^{n-1}$. A large amount of memory allocation is required for this method and is not a feasible solution.
- The system probability function can be used but requires a fault tree or BDD analysis to get it.

Therefore, some other programmable method needs to be formulated to calculate the structural importance measure. The only possibility seems to be an approximated
technique. This may be viable since the appropriate method is only required to produce a relative ordering of the component values to be successful.

### 8.9 Approximation Methods to Calculate The Structural Importance Measure

### 8.9.1 Introduction to Procedures

Instead of trying to program the structural importance measure exactly using the mathematical equations it seems evident that the processing time and efficiency of using this technique could be enhanced by using some approximation or simpler method to calculate a near-exact measure or at least a similar ordering.

There are three methods that have been used to generate approximations to the structural importance ordering:

1. To calculate the structural importance measures of the components in the tree and then to look for patterns within the tree relating to these measures. If established this would enable the ordering to be produced by inspection of the tree structure.
2. Generate alternative weightings similar to the importance ranking measures derived by a simpler method.
3. Apply the Birnbaum structural importance methodology directly to the tree.

### 8.10 Approximation Method 1 - Pattern Identification

One question to ask in trying to find a simple approach to establishing the structural importance measures of the components in the tree is, "Is there any pattern in the location of components with certain structural importance values?"

If patterns exist which relate to importance values of events within small fault tree modules, then it is possible that this modular approach can be repeatedly applied through the tree structure and higher level importance values ascertained. It would then remain to decide how to treat repeated events.

All fault trees used in this research incorporate an alternating gate sequence. The first theory to investigate when searching for a pattern is whether the start gate of the sequence affects the pattern. To begin the analysis, consider the fault tree in figure 8.4. The logic expression for the top event is given by:

$$
T O P=A+B D+B E F
$$

The probability expression of the top event is given by the equation:

$$
P(T O P)=q_{A}+q_{B} q_{D}+q_{B} q_{E} q_{F}-\left(q_{A} q_{B} q_{D}+q_{A} q_{B} q_{E} q_{F}+q_{B} q_{D} q_{E} q_{F}\right)+q_{A} q_{B} q_{D} q_{E} q_{F}
$$

Hence, calculating the structural importance measures, using equation 8.4, the importance measures are given in table 8.7.


Figure 8.4: Simple Tree to Illustrate Effects of Gate Sequence on Structural Importance Values

Now considering the same tree, but this time with the starting gate being an AND gate rather than an OR, the reduced top event expression with the logic reversed is:

$$
T O P=A B+A D E+A D F
$$

The probability of the top event is then:

$$
\begin{aligned}
Q_{S K S}=P(T O P)=q_{A} q_{B}+ & q_{A} q_{D} q_{E}+q_{A} q_{D} q_{F}- \\
& \left(q_{A} q_{B} q_{D} q_{E}+q_{A} q_{B} q_{D} q_{F}+q_{A} q_{D} q_{E} q_{F}\right)+q_{A} q_{B} q_{D} q_{E} q_{F}
\end{aligned}
$$

Calculating the structural importance values gives the figures shown in table 8.7:

As can be seen, the values are the same for both trees, which is true for trees of any structure. This finding identifies that the type of the gate at the top of the tree, namely whether the starting sequence is an AND gate or an OR gate, is not important when trying to find the structural importance of the components within the tree. This indicates that the importance relies on the structure of the tree and not it's logic.

| Component | Structural Importance for | Structural Importance |
| :---: | :---: | :---: |
| with reversed logic |  |  |
| $A$ | figure 8.4 | $11 / 16$ |
| $B$ | $11 / 16$ | $5 / 16$ |
| $D$ | $5 / 16$ | $3 / 16$ |
| $E$ | $3 / 16$ | $1 / 16$ |
| $F$ | $1 / 16$ | $1 / 16$ |

Table 8.7: Importance Values For Components in Figure 8.4 as Shown and With Reversed Logic

Having established that the starting gate in the alternating gate sequence is irrelevant, the next search is for patterns evident between the location of the component and it's
importance measure. To investigate the possibility of a pattern, trees with no repeated events were initially considered. The theory behind this is that finding a pattern for a simple tree should facilitate the emergence of a pattern for trees with repeated events. Initial steps involved drawing a progression of simple fault trees (as shown in figures 8.5-8.10). The structural importance of each component was gained using the computer code which incorporates Birnbaums measure, each is written on the fault tree for reference.


Figure 8.5: Example Tree 1

Figure 8.7: Example Tree 3


Figure 8.6: Example Tree 2


Figure 8.8: Example Tree 4



Figure 8.9: Example Tree 5


Figure 8.10: Example Tree 6

The first step in the analysis involved looking at the importance values of components that were inputs to the same gate. It was found that these basic events each had the same importance values, as can be seen within all the trees in figures 8.5-8.10. For gates with just basic event inputs the components of those gates with fewer inputs were found to have higher structural importance values, as can be seen with figure 8.10, where components $A$ and $B$ have higher structural values (35/128) than components $D, E$ and $F$ (15/128).

The pattern becomes more complex as the number of levels in the tree increases. If the tree has just one branching structure, as shown in figure 8.6, then the components higher up the tree have a larger structural importance value. If the same single branching structure occurs and the number of components on each branch is increased then the components of each branch assume the same structural importance values (shown in figure 8.7).

So far the patterns are clear, however, this fact no longer holds true when the complexity increases further with a number of branching structures. Adding another branch to the tree, as in figure 8.8 the ordering seems to suggest that the component connected to two others $(A)$ has a higher structural importance than the two components $B$ and $C$ which are linked to each other only. This leaves the components on the bottom level of the tree to
have the lowest structural importance. Hence, in this example there is a swapping between branches to get the desired ordering. However, in example tree 5 (figure 8.9), the order is generated by just taking a top-down, left-right approach and there is no swapping between branches. There does not seem an apparent rule to decide which order to take the components in, sometimes swapping between branches occurs and in other instances it does not. By simple inspection of the tree it does not seem evident that a pattern can be seen between the structural importance of the component and it's location in the tree structure.

There are a number of factors that obviously affect the influence a component has on the other components and ultimately the top event. Thus, finding all these factors by just viewing the tree structure does not seem possible to solve the mystery of how each component should be ranked in terms of it's structural importance.

### 8.11 Approximation Method 2 - Simpler Weighting Methods

Having not been able to spot a visible pattern to locate the order of the variables as if generated by the structural importance measure, the next idea is to look at whether an approximation can be made to finding the same orderings or similar importance measures.

The importance measures provide the basic events in the fault tree with a relative weighting. There are alternative approaches to produce relative weights for events, some of which are examined below:

- Approach 1 - Calculating The Structural Importance Measures by Dividing by the Number of Inputs

This method divides a gates' output value (weight) by the number of inputs since each contributed equally. This is a top-down approach and a value is allocated for the top event, which is then re-distributed to lower level events. In this case the values are
normalised relative to unity and so the top event is assigned the value of 1 . The value of lower level gates and events is then calculated by dividing by the number of inputs, so if there are three inputs then the value of each event or gate will be a third. Again if one of these components is a gate, the inputs to this gate are equal to a third divided by the number of inputs. To illustrate this, examine the tree in figure 8.6. $T O P$ would $=1, A=$ $1 / 2, B=C=1 / 6$, and $D=E=1 / 12$. Although the numerical values are not the same, the ordering produced is the same as that produced for the structural importance measure.

Testing on the tree in figure 8.8 gives the values shown below:

$$
\begin{aligned}
& B=A=C=1 / 4 \\
& D=E=1 / 8
\end{aligned}
$$

From this the ordering would be given as: $C<B<A<D<E$, applying a top-down ordering for matched measures. This however, is different than the ordering produced using the structural importance measure which would be $A<C<B<D<E$. This ordering is only different because of the way the tie between matched measures was broken, and breaking the tie in another way could have resulted in the same ordering. Still the actual values are different for each component using this method as compared to the structural importance measures. Just from using these two simple examples it is apparent that the method is not going to match exactly the ordering produced using the mathematical structural importance for every tree. However, despite the mismatch it is possible that this method of ordering would produce as good or better results. Hence, this possibility was investigated. The first problem however is to deal with repeated events that are common in the fault trees within the data set.

There is a dilemma with how to approach repeated events. Just adding their values will disproportionately increase their importance, because a component that occurs twice in the tree does not necessarily have twice the importance. The second occurrence may be redundant in the Boolean function. Any redundancy will mean that this type of approach will never be exact, but the aim is to produce a heuristic that works most of the time.

One solution could be to find the average of the values given at it's different appearances. This however, would probably underestimate the components importance within the fault tree because if an event has one repetition positioned high in the tree structure and another nearer the bottom then the average value will be lower than that of the repetition value at the top if on its own, which is likely to have more importance. Hence what is required is some scaled version of the total combination. Exactly what kind of scale that is required is unknown, but in this method the average value is multiplied by the square root of the total number of inputs. In essence, the weight $w_{i}$ for repeated event $i$, is calculated by summing the values of each repetition and multiplying by the square root of the total number of repeated components, as given in equation 8.5 , where $i$ refers to the component, and $j$ each of it's occurrences.

$$
w_{i}=\sqrt{n} \sum_{j} w_{i j}
$$

Using this scaling mechanism for repeated events, the values for the components shown in the tree in figure 8.4 are given in table 8.8 with structural importance measures shown for comparison.

| Component | Approach 1 values | Structural Importance Values |
| :---: | :---: | :---: |
| A | $\frac{1}{3}=0.3333$ | 0.6875 |
| B | $\left[\frac{1}{6}+\frac{1}{9}\right] / \sqrt{2}=0.1964$ | 0.3125 |
| D | $\frac{1}{6}=0.1667$ | 0.1875 |
| E | $\frac{1}{9}=0.1111$ | 0.0625 |
| F | $\frac{1}{9}=0.1111$ | 0.0625 |

Table 8.8: Approximated Structural Importance Measures Using Approach 1

The variable order generated from this would be $A<B<D<E<F$. The matched values are decided upon my employing the usual top-down, left-right ordering. This is the same
ordering that would be generated if the mathematical structural importance method was applied to the tree.

This method is much simpler to calculate and yields the same result for this example fault tree structure. Now the task is to compare to other fault tree structures. On further testing the variable orderings produced did not produce the exact order as compared to the mathematical structural importance method. This however, may not be a problem if the resulting BDD produced is still as optimal as the one produced using the exact structural measure or at least better than the best of the six alternatives.

A program sim_str. $f$ was produced which takes a fault tree data file and produces the ordering of the components based on the calculation of the simplified structural importance measure. This output file (*.sim) was then read into a program to generate the resulting BDD. The results of the simplified ordering scheme as compared to the best of the six ordering alternatives previously used are shown in table 8.9. The same 225 trees used to establish the structural importance potential have been re-tested. The full results for each tree are given in Appendix VIII.

|  | No. of trees | Sums | Percentages |
| :---: | :---: | :---: | :---: |
| $<$ | 40 |  |  |
| $=$ | 79 | $(</=) 119$ | $(</=) 52.9 \%$ |
| $>$ | 106 | $(>) 106$ | $(>) 47.1 \%$ |

Table 8.9: Comparison of BDD Size of Simplified Structural Importance Ordering and Best of Set of Six Alternatives

The proportion of trees whose ordering resulted in a BDD of equal or smaller dimension than the best from the set of six alternatives is considerably less than the results obtained when using the structural importance measure (76.9\%). Despite the fact that the percentage is reduced, this approximated technique has an increased potential over any single heuristic of the set of six at producing the smallest BDD size. The program runs
very quickly and the results predict over half of the data set of trees having a BDD of better or equal dimension than the chosen best from a set of six alternatives. As using the pattern recognition approaches is not always guaranteed to predict the best scheme this method is a feasible alternative. Also compared to the six heuristics individually, from the 225 trees the most frequent scheme to produce the best result was the modified topdown, left-right approach, but only on $29.8 \%$, so this is considerably better.

- Approach 2 - Dividing by Number of Critical States

From approximation approach 1, the input values of a gate are generated by just dividing the gate value by the number of inputs to it. If however, the number of critical states for components from a single gate are examined the output values are slightly different. For example, consider the gate with three inputs $A, B$, and $C$, shown in figure 8.11.


Figure 8.11: Gate With Three Inputs $-A, B$, and $C$

Results from section 8.10 have shown that the structural importance of the components is the same regardless of whether the top gate is AND or OR, therefore the gate type need not be considered. If the number of critical states for component $A$, for which there are two remaining components, is calculated, the result is as previously found in table 8.1. By examining the number of critical states the criticality of component $A$ is $1 / 4$.

Now considering a gate with four components $A, B, C$, and $D$. The criticality of component $A$ is $1 / 8$ as shown in table 8.10.

| Component States | Critical If And Gate |
| :---: | :---: |
| $B C D$ | Yes |
| $\bar{B} C D$ | No |
| $\bar{B} \bar{C} D$ | No |
| $\overline{B C D}$ | No |
| $B \bar{C} D$ | No |
| $B \bar{C} \bar{D}$ | No |
| $B C \bar{D}$ | No |
| $\bar{B} C \bar{D}$ | No |
|  | Criticality $=1 / 8$ |

Table 8.10: Importance Values For a Gate With Four Inputs

If the number of components is increased the following pattern arises with regard to the criticality of each component (shown in table 8.11):

| Number of inputs, | 2 | 3 | 4 | n |
| :---: | :---: | :---: | :---: | :---: |
| Importance Values - Approach 2 | 1/2 | $1 / 4$ | 1/8 | $1 / 2^{\mathrm{n}-1}$ |
| Importance Values - Approach 1 | 1/2 | 1/3 | 1/4 | 1/n |

Table 8.11: Pattern of Importance Measures

Therefore, for $n$ components the criticality is equal to $\frac{1}{2^{n-1}}$. The differences in the values between using approximation approach 1 and this new technique (approach 2) can be seen in the bottom row of the table 8.11.

Applying this numbering scheme the value of the components are evaluated by taking the value of the gate and dividing by $2^{n-1}$. For the tree in figure 8.4 , the new values of importance are shown in table 8.12.
\(\left.\begin{array}{|c|c|c|c|}\hline Component \& Importance Value \& No. of inputs \& Inputs of gates, <br>
\hline T O P \& 1 \& 3 from TOP \& A / 2 AND gate's (say G 1 <br>

and G2)\end{array}\right]\)| $G 1=B / D$ |
| :---: |
|  |
| $A=G 1=$ |
| $G 2$ |

Table 8.12: New Importance Values For Components in Figure 8.4

The values for each component using the repeated event rule (equation 8.5) equals:

$$
\begin{array}{ll}
A=1 / 4 & =0.25 \\
B=[1 / 8+1 / 16] / \sqrt{ } 2 & =0.1326 \\
D=1 / 8 & =0.125 \\
E=1 / 16 & =0.0625 \\
F=1 / 16 & =0.0625
\end{array}
$$

Again the use of this simplified version of calculating the components structural importance and the top-down, left-right ordering for matched components produces the correct ordering for this one tree.

The technique was programmed (sim_strl.f) and using the program schemes.f the resulting size of the BDD was compared with the best of the six orderings and the percentage predictive results for the exact structural measure. The outcome given is shown in table 8.13 for the same 225 trees.

| No. of trees | Sums, | Percentages |  |
| :---: | :---: | :---: | :---: |
| $<$ | 36 |  |  |
| $=$ | 77 | 113 | $50.2 \%$ |
| $>$ | 112 | 112 | $49.8 \%$ |

Table 8.13: Results of Resulting BDD Sizes Using Approach 2 Ordering

The result using this method produces worse results compared to the approximation approach 1 and the exact structural importance measure. Obviously this technique is not generating the correct ordering or values for the components. This may be due to the factor of repeated events. As this approach is no better than approach 1 no further research has been carried out. The BDD sizes resulting for each tree are given in Appendix VIII.

## - Approach 3-Altering The Repeated Event Multiplication Factor

A problem area appears to be in calculating the weights of repeated events. A new method is tried whereby the values of repeated events are added when they are encountered, the value of the second repeated event is divided by the square root of 2 and added to the first value. If another repeated event is encountered then the value is divided by square root of 3 and this added to the previous sum. For example, from figure 8.4 the first encounter of $B=1 / 6$. Then $B$ is found further along the level with a value of $1 / 9$. The sum would then be $B=1 / 6+(1 / 9) / \sqrt{2}$. If there was a third repetition of event $B$ with a value of $1 / 32$, then the sum would be $B=1 / 6+(1 / 9) / \sqrt{2}+(1 / 32) / \sqrt{ } 3$.

After looking at the result of the ordering produced by this third approximation approach, it was apparent that some tree orderings where affected and others not. The real outcome to be observed is the effect on the BDD size. Program sim_str2.f was generated to create the new order and schemes. $f$ was used to generate the BDD. For the same 225 trees as
used to test approaches 1 and 2 , the results for approach 3 are given in table 8.14, with the number of nodes in each BDD for each tree given in Appendix VIII.

|  | No. of trees | Sums | Percentages |
| :---: | :---: | :---: | :---: |
| $<$ | 65 |  |  |
| $=$ | 75 | $(</=) 140$ | $(</=) 62.2 \%$ |
| $>$ | 85 | 85 | $37.8 \%$ |

Table 8.14: Results of Approach 3 Ordering on BDD Size

These results are the best out of the three approximation approaches, although still slightly worse than the exact structural importance measure. These results have shown that the approximation technique is dependent on the way the repeated events within the fault tree are tackled, with regard to combining the values of each repetition. As this is the best result produced it is recommended that if an approximation technique is to be used then approach 3 should be adopted.

### 8.12 Approximation Method 3 - Applying Birnbaum's Structural Importance Method to The Tree

The principle of the Birnbaum structural importance measure can be applied directly to the tree if the basic events are independent. Using this technique, the selected component assumes the failure probability of 1 and 0 on two consecutive runs, the rest of the components are given failure probabilities equal to $1 / 2$ and the probability of occurrence of the top event is evaluated by working up through the tree structure. The BDD variable ordering is generated depending on the basic event that generates the largest probability value contribution for the top event. The difference with this and the exact version of Birnbaum's structural measure is the terms in the unavailability expression of the top event. When this approximation method is applied to a tree the redundancies have not been reduced by Boolean algebra and so the cut sets may not be minimal. For this reason
the structural importance values and resulting weights can not be expected to be exactly the same but it may still offer a relatively good ordering heuristic.

Using this method of applying the principle directly to the tree, the component which is selected and leads to the highest value contribution for the top event will be first in the ordering of the variables. The selected component that produces the smallest value for the probability of the top event occurrence will be positioned at the end of the variable ordering list.

To illustrate the application of this proposed Birnbaum method, consider the tree in figure 8.6. The data file representing the fault tree structure would be written in the following format, with the first gate in the tree given the name Gate 1 (AND gate) and the second gate (OR gate) the name Gate 2 for reference purposes:

| TOP | OR | 1 | 1 | Gate1 | A |  |
| :--- | :--- | :--- | :--- | :--- | :--- | :--- |
| Gate1 | AND | 1 | 2 | Gate2 | B | C |
| Gate2 | OR | 0 | 2 | D | E |  |

The code written to establish the structural importance value for each component within the data file follows these steps:

1. Make a list of all the components within the tree structure, easiest method using a topdown, left-right approach.
2. Repeat step 2 twice, first setting the selected component failure probability to 1 and the second time with the selected component failure probability set to 0 .
a. Start at the top of the data file.
b. Repeat the following steps:

- Work through the data file and find gates with only basic events, substitute in value for selected component and $1 / 2$ for the remaining component failure probabilities.
- Calculate intermediate values of importance - if the gate is an AND gate multiply values of inputs, if an OR gate then use $1-\prod_{i=1}^{n}\left(1-q_{i}\right)$.
- Substitute in data file value for gate just calculated.
- Continue through the data file, if at bottom, start process again at top searching for gates whose inputs all have a calculated value.
- Calculate new intermediate results.
- Continue until the Top Event gate has been given a value.
c. Record value.

3. Subtract the value gained from the second run from the first. The result is the approximation for the structural importance value of the component.

Therefore, the structural measure for each component in figure 8.6 is:

- Measure for $A=1-3 / 16=13 / 16$
- Measure for $B=11 / 16-1 / 2=3 / 16$
- Measure for $C=11 / 16-1 / 2=3 / 16$
- Measure for $D=5 / 8-9 / 16=1 / 16$
- Measure for $E=5 / 8-9 / 16=1 / 16$

From these values and using the top-down, left-right method of ordering for matched components, the ordering would be:

$$
A<B<C<D<E
$$

This is the same as that generated with the mathematical structural importance measure.

The approach was tested on a number of trees with non-repeated events, some produced the same order others did not. As no approach so far has given exact results as compared
to the mathematical structural importance measure, this approximated measure was evaluated further.

Computer code sim_str3.f was produced to generate the variable ordering list for two hundred and twenty five fault trees (used for other approximation methods) using the Birnbaum measure applied directly to the tree. The number of trees for which the BDD produced was equal to, greater than, or less than the size of the BDD derived from the best scheme of the six alternatives is given in table 8.15.

| Outcome | Percentage |
| :---: | :---: |
| < 77 |  |
| $=74$ | 67.1 |
| $>74$ | 32.9 |

Table 8.15: Results Using Approximation Method 3

From table 8.15 it can be seen that for $67.1 \%$ of the test set of trees the BDD produced was of equal or smaller dimension than the BDD produced using the best scheme option from a set of six. This result is the highest percentage of equal or smaller BDDs of the three approximation methods tested. In comparison to the mathematical structural importance measure, whose percentage for this category of BDDs was $76.9 \%$, this approximated figure is nearing the same accuracy.

It is felt that by altering the method for matched measures, other than the top-down, leftright approach, then further improvements may result.

The result produced using this approximated measure as a heuristic is approaching the seventy percent chance of gaining a minimal or near minimal BDD achieved with the neural network approaches. However, the simplicity of this heuristic compared to the necessary effort required to generate an adequate network using a form of neural network means that its potential is far greater. This approximated technique is recommended as
an alternative variable ordering heuristic, which is believed to date to be one of the best at gaining a minimal or near minimal BDD for any given fault tree structure.

### 8.13 Summary of Structural Importance Measure Approach

- Static variable orderings can be created in two ways:
- A structured traversal of the tree (preserving neighbourhoods)
- A method allocating weights to events (not necessarily preserving neighbourhoods)
- The result of both methods are dependent on the way the tree has been drawn, not the logic function it represents.
- A deterministic importance measure has been applied to generate an ordering of the variables of a fault tree, the structural measure, which is dependent upon the logic function and not the drawing of the fault tree.
- To assess the effectiveness of the structural measure BDDs using this ordering have been compared to BDDs generated by using other variable ordering heuristics applying a structured traversal. The results proved to be consistently good.
- The structural importance approach has proven to produce a BDD of equal or smaller dimension than the previous best result from an ordering selected from six structured traversal alternatives on $77 \%$ of occasions.
- Using the two pass approach, such that the structural importance is generated using the system probability function from the BDD generated using the best ordering over a selection of trees - modified top-down, left-right traversal. A second BDD is then produced using the importance ordering.
- To improve upon the efficiency of the two pass method, an approach to approximate the weights obtained by the structural importance measure has been researched with three alternatives.
- The first approximation method involved trying to locate patterns. Examining the location of components with certain structural importance measures a pattern
could not be established, and applying a simplified calculation procedure to the tree itself was recommended as the next step.
- The second approximation technique considered three approaches to finding simpler weighting methods. Using approach 1 , of generating the ordering by calculating the structural importance by dividing by the number of inputs, produces $56 \%$ of equal or better dimension BDDs to the previous best. This is better than using the single best heuristic of the modified top-down, left-right approach that only predicts the best BDD structure on $29.8 \%$ of occasions.
- Approximation approach 2 of dividing by $2^{n-1}$ produced equal or better results on $50.6 \%$ of occasions, although worse than approach 1 , it is still better than a single heuristic. Changing the approach to combining the repeated events (approach 3 ) produced a $56 \%$ chance of getting an equal or better result than the best of six alternatives.
- All of these simplified approaches, produce better results than using a single heuristic, and despite not producing the same effect as the mathematical structural importance measure are quick and efficient.
- The third method to find an alternative structural importance measure was to use the Birnbaum measure applied directly to the fault tree. Results using this technique produced equal or lesser sized BDDs on $67 \%$ of occasions.
- Of all the approximation methods tried the latter, using Birnbaums measure, is preferred. The measure almost achieves the same percentage of gaining the smallest BDD as using the neural network approaches, although this method is much easier and less time consuming to implement.
- It is felt that this approximated structural importance ordering heuristic is the best method to date in trying to achieve a minimal or near minimal BDD structure for any given fault tree.
- Future work in this area needs to focus on improving the simplified versions of the structural importance measure and finding an alternative method that approaches the 77 percent performance of the mathematical measure.


## 9 Conclusions and Future Work

### 9.1 Summary of Work

Qualitative and quantitative assessment of a fault tree using the Binary Decision Diagram approach has been found to be more efficient in terms of finding the minimal cuts and an exact probability of failure can be found. The limitation with the technique however is in the conversion process from a fault tree to the BDD where the variable ordering involved can influence the resulting size of the BDD. There are already several ordering heuristics in the literature but none of them will produce a minimal or near minimal BDD for all fault tree structures. The research of this thesis has focussed on trying to find a solution to the problem of variable ordering. To try and overcome this dilemma two main avenues of research were sought. The first area looked at trying to select the best heuristic for a given fault tree from a set of alternatives, thus allowing different ordering schemes to be selected for different trees, hence possibly over coming the limitations of a single heuristic. The second methodology looked at a completely new heuristic approach using the structural importance measure of the components within the tree to produce a ranked list.

To try and select an appropriate heuristic from a set, it was necessary to choose the alternatives. There are two types of ordering heuristic in the literature, namely neighbourhood and weighting methods. Neighbourhood methods produce an ordering by performing a systematic traversal of the tree adding the basic events to the ordered list as they are encountered. Weighting methods usually make two passes of the tree. On the first pass a weight is given to each basic event and gate within the tree, and the second pass then orders the basic events depending on the weights allocated in the first pass. For this research a set of six structured neighbourhood methods of ordering were used, as these were shown in previous research to perform well. These schemes were:

- Top-down, left-right.
- Depth-First.
- Priority Depth-First.
- Modified versions of each of the three above.

To be able to choose between the alternatives a set of rules or a pattern needed to be identified between the characteristics of a fault tree and the ordering heuristics. The features of the fault tree which help to distinguish the appropriate scheme to yield the smallest BDD structure were not known, and therefore a number of characteristics were chosen which were hoped provided the correct information. The best results were gained using eleven characteristics and these were:

1. Percentage of AND gates;
2. Percentage of the total number of events that were repeated;
3. The percentage of the different basic events that were repeated;
4. The type of top gate (AND/OR).
5. Number of Levels in the Tree;
6. Number of Events off The Top Gate (Top Gate Inputs);
7. Number of Basic Events;
8. Maximum Number of Gates in Any Level;
9. Number of Gates with Just Event Inputs Only;
10. Number of Gates with Just Gate Inputs Only;
11. Highest number of repeated events.

The first pattern recognition approach investigated was that of the machine learning classifier system. This approach incorporated a genetic algorithm. The basic principle of the classifier system was to generate a set of rules through a matching and bidding process of information contained within a training data set. The classifier system has a large number of variable parameters and although a considerable number of trials were carried out with varying parameters a system with even adequate performance at predicting the desired scheme choice could not be found.

Following on from this, two alternative pattern recognition approaches were tested. These involved two types of neural network: a multi-layer network and a radial basis function network. The basis of this method was to take the inputs of the problem and map them to the output by using a series of weights and non-linear functions.

On reviewing the heuristics currently in the literature certain problem areas were evident. One was that many of the heuristics were affected by how the fault tree was drawn, therefore for the same logic expression a number of different BDDs could result depending on how the fault tree was represented. Also many of the heuristics have a structured pattern and even some of the weighting methods incorporate a structured pattern. That is, the ordered list is generated by going from the top of the tree to the bottom, and it does not allow for components to be selected from different branches of the tree and lie next to each other in the ordering list. Another problem is how to deal with matched components in the weighting methods, using one method to separate the basic events with equal weights will result in a different ordering compared to when a different separation procedure is used, again two separate BDDs for the same tree.

From this the properties required in a good ordering heuristic seem to be:

- The contribution of an event to the system failure mode must be reflected in the ordering produced.
- The ordering must be robust i.e. the ordering must be dependent upon the logic function represented by the fault tree and not influenced by the way the fault tree has been drawn.
- To uniquely map the fault tree onto a single event ordering.

Considering these points the structural importance measure was investigated. This heuristic satisfies two out of the three points above. It does represent the contribution each component makes to the occurrence of the top event, and it is also unaffected by the way the tree is written or drawn. However, the ordering produced is not unique because
ties may result with some of the component measures and the means of breaking these ties will affect the ordering. Comparing the BDD sizes resulting from the best scheme option from the set of six used in the pattern recognition research and this new measure the results were favourable.

As the structural measure is calculated via the BDD , hence requiring two BDD formulations to produce the smallest end product, to improve upon this approximation methods were adopted to find the structural importance measure or at least the same ordering produced as with the structural importance measure. Three main areas were addressed, finding patterns in the tree, using simpler weighting methods and applying the Birnbaum measure directly to the fault tree.

### 9.2 Conclusions

1. The three pattern recognition approaches tried produced varying degrees of success at trying to predict from a set of six alternatives the best scheme option for a given fault tree. The worst approach was the machine learning classifier system. The results using this were inconsistent over a number of trials and no model could be found with an adequate level of performance to be used as a predictive mechanism.
2. Two neural network approaches were used to identify the pattern between the fault tree characteristics and the ordering heuristic preferred. The multi-layer perceptron produced a correct prediction for fourteen out of the twenty test fault trees used. Although not an exact prediction rate, this percentage for achieving a near minimal or minimal BDD is higher than using any of the six ordering heuristics on their own.
3. The radial basis function neural network was also investigated. The results produced were equal to those produced using the multi-layer perceptron. It was also found that the incorrect predictions for most trees were the second best choice, with very little difference in the desired coded values, hence reflecting that the size of the resulting BDD was not that much bigger than the best. As the ease of generating this network was much greater than using the MLP approach, this network technique is one method suggested to aid in the solution of the variable problem.
4. To change the direction of the research from choosing from a selection of ordering heuristics to finding a heuristic that singularly produced a minimal BDD the research focussed on the structural importance of each component. Using this method of ranking produced smaller or equal sized BDDs on 77 percent of test trials. The problem with the technique is that the BDD needs to be constructed to establish the structural values, which means generating the BDD twice. Despite this the technique is feasible and it is suggested that the modified top-down, left-right approach be adopted to generate the initial BDD from which the structural importance measures can be generated and then the next BDD generated using the structural importance measures, to hence obtain system parameters.
5. To make the process more efficient by irradiating the need to construct the BDD twice simpler approximation methods were researched to yield the same ordering permutations. The most successful of these involved applying the Birnbaums structural measure directly to the fault tree, instead of the probability expression of the top event. Doing this produced smaller or equal sized BDDs on 67 percent of the trial data set. Although this is not as high a percent as using the mathematical measure the result is significantly higher than using any of the other six ordering heuristics, hence it is suggested as the best ordering heuristic to date at producing the smallest BDD for any given fault tree.
6. Following the research, the alternative idea to solve the ordering problem is to use the neural network approach to select the best ordering heuristic from the set of six and to use this to generate the BDD. From this BDD the structural importance values of all the components of the fault tree can be produced and a corresponding ranking used to generate the smallest BDD , from which the qualitative and quantitative analysis can occur.

### 9.3 Future Work

### 9.3.1 Examine The Input Characteristics of The Fault Tree

Following the research using the pattern recognition approaches, the predictive potential of the best networks using the multi-layer perceptron and radial basis function still have room for improvement. A large number of networks were investigated and it was felt that further improvements were only possible by examination of the inputs and whether they provide the necessary pattern information from which the outputs can be established. As it was not known which characteristics of the fault tree were most important examining these characteristics and adding new possibly better characteristics and removing existing poor characteristics may help to yield better predictive results. It is thought that examining the weights of the multi-layer perceptron may yield information into the strengths of the connections and ultimate effect of the characteristic in deciding the path through the network.

### 9.3.2 Applying Modularisation Techniques To The Fault Tree

Another method to alter the inputs of the network relates to the characteristics of the fault tree. The problem may be simplified if the starting fault tree is in its simplified form. Applying a modularisation technique may help to identify key characteristics which when applied to the neural network approach may help the predictive capabilities.

### 9.3.3 Continue Research With Neural Network Pattern Recognition Approaches

Following examination of the characteristics, either by reviewing the weights or by modularisation techniques, the research can be continued with further testing of the neural network pattern recognition approaches. The input data set could be enhanced to accommodate more population fault trees with possibly a greater number of trees from industrial applications. Also the parameters of the network could be altered, for example by using more complex optimisation techniques.

### 9.3.4 Improve Approximated Structural Importance Measures

The research using the mathematical structural importance measure has shown positive results. The mathematical method could be enhanced by considering different ways of ordering matched measures. Following this the approximated techniques need to be further researched to produce the same predictive potential and ideally a percentage high in the nineties for the smallest BDD production for any given fault tree structure.

## Appendix I

Characteristics of 51 Benchmark Fault Trees Used in Sinnamon Study [Sin96]

| Fault Tree Number | Number of Gates | Number of Basic Events | Number of Repeated Basic Events | Number of Minimal Cut Sets |
| :---: | :---: | :---: | :---: | :---: |
| 1 | 79 | 103 | 39 | 3804 |
| 2 | 6 | 7 | 3 | 7 |
| 3 | 3 | 4 | 1 | 2 |
| 4 | 19 | 16 | 2 | 27 |
| 5 | 14 | 13 | 2 | 9 |
| 6 | 17 | 11 | 7 | 43 |
| 7 | 32 | 63 | 0 | 8716 |
| 8 | 29 | 61 | 0 | 7471 |
| 9 | 30 | 60 | 0 | 7056 |
| 10 | 21 | 40 | 0 | 416 |
| 11 | 3 | 4 | 1 | 3 |
| 12 | 21 | 40 | 4 | 84424 |
| 13 | 19 | 19 | 1 | 63 |
| 14 | 21 | 21 | 1 | 75 |
| 15 | 30 | 32 | 1 | 2100 |
| 16 | 42 | 41 | 21 | 11934 |
| 17 | 58 | 57 | 21 | 36990 |
| 18 | 60 | 57 | 41 | 11934 |
| 19 | 10 | 10 | 1 | 13 |
| 20 | 6 | 8 | 2 | 6 |
| 21 | 30 | 72 | 8 | 255 |
| 22 | 10 | 31 | 2 | 71 |
| 23 | 25 | 61 | 57 | 7777 |
| 24 | 12 | 30 | 4 | 61 |
| 25 | 81 | 199 | 68 | 8179 |
| 26 | 5 | 7 | 0 | 4 |
| 27 | 5 | 7 | 3 | 4 |
| 28 | 11 | 21 | 0 | 36 |
| 29 | 11 | 20 | 1 | 30 |
| 30 | 11 | 20 | 1 | 10 |
| 31 | 70 | 68 | 26 | 4892 |
| 32 | 30 | 34 | 28 | 35 |
| 33 | 26 | 16 | 11 | 20 |
| 34 | 5 | 7 | 1 | 3 |


| Fault Tree Number | Number of Gates | Number of Basic Events | Number of Repeated Basic Events | Number of Minimal Cut Sets |
| :---: | :---: | :---: | :---: | :---: |
| 35 | 4 | 5 | 1 | 2 |
| 36 | 122 | 61 | 60 | 46188 |
| 37 | 4 | 6 | 0 | 6 |
| 38 | 58 | 114 | 114 | 35300 |
| 39 | 4 | 5 | 1 | 3 |
| 40 | 5 | 6 | 1 | 3 |
| 41 | 8 | 8 | 1 | 6 |
| 42 | 5 | 5 | 3 | 4 |
| 43 | 7 | 6 | 3 | 2 |
| 44 | 7 | 6 | 3 | 4 |
| 45 | 10 | 10 | 2 | 8 |
| 46 | 153 | 74 | 46 | 340 |
| 47 | 3 | 4 | 1 | 2 |
| 48 | 4 | 6 | 1 | 3 |
| 49 | 3 | 4 | 2 | 4 |
| 50 | 4 | 5 | 3 | 5 |
| 51 | 10 | 8 | 4 | 10 |

## Appendix II

Characteristics of Benchmark Fault Trees Used in Thesis

| Tree Number | \% ANDS | \% Diff Rep | \% Total Rep | \% Top Gate |
| :---: | :---: | :---: | :---: | :---: |
| Aaaaaaa | 33.33 | 33.33 | 33.33 | 1 |
| Artqual | 40.00 | 42.86 | 27.27 | 1 |
| Artree | 66.67 | 25 | 20 | 0 |
| Astolfo | 36.84 | 12.5 | 9.09 | 0 |
| Bddtest | 66.67 | 15.38 | 13.33 | 0 |
| Benjiam | 33.33 | 63.64 | 31.82 | 1 |
| Bpfeg03 | 40 | 0 | 0 | 0 |
| Bpfen05 | 41.18 | 0 | 0 | 0 |
| Bpfig05 | 35.71 | 0 | 0 | 0 |
| Bpfin05 | 66.67 | 25 | 25 | 0 |
| Bpfpp02 | 33.33 | 10 | 9.09 | 1 |
| Bpfsw02 | 60 | 42.86 | 25 | 1 |
| Ch8tree | 25 | 5.26 | 5 | 0 |
| Dre1019 | 25 | 4.76 | 4.55 | 0 |
| Dre1032 | 28.57 | 3.12 | 3.03 | 1 |
| Dre1057 | 23.08 | 51.22 | 32.81 | 1 |
| Dre1058 | 29.41 | 36.84 | 26.25 | 1 |
| Dre1059 | 26.32 | 71.93 | 28.47 | 1 |
| Dresden | 25 | 10 | 9.09 | 0 |
| Emerh2o | 40 | 25 | 20 | 1 |
| Fatram2 | 31.58 | 11.11 | 10 | 0 |
| Hpisf02 | 28.57 | 6.45 | 6.06 | 0 |
| Hpisf03 | 33.33 | 93.44 | 27.4 | 0 |
| Hpisf21 | 37.5 | 13.33 | 11.76 | 0 |
| Hpisf36 | 36.73 | 34.17 | 18.73 | 0 |
| Jdtreel | 60 | 0 | 0 | 1 |
| Jdtree2 | 60 | 0 | 0 | 1 |
| Jdtree3 | 45.45 | 0 | 0 | 1 |
| Jdtree4 | 45.45 | 5 | 4.76 | 1 |
| Jdtree5 | 54.55 | 5 | 4.76 | 0 |
| Khictre | 52.38 | 100 | 18.64 | 1 |
| Modtree | 50 | 40 | 28.57 | 1 |
| Nakashi | 61.9 | 68.75 | 37.92 | 1 |
| Newtre2 | 40 | 14.29 | 11.11 | 0 |
| Newtre3 | 50 | 20 | 16.67 | 0 |
| Newtree | 50 | 16.67 | 14.29 | 0 |
| Relcour | 33.33 | 0 | 0 | 1 |
| Rstreel | 31.45 | 100 | 42.22 | 0 |
| Rstree2 | 50 | 20 | 16.67 | 0 |
| Rstree3 | 60 | 16.67 | 14.29 | 0 |
| Rstree4 | 50 | 12.5 | 10 | 0 |
| Rstree5 | 33.33 | 60 | 30 | 1 |
| Rstree6 | 66.67 | 50 | 33.33 | 0 |
| Rstree7 | 66.67 | 33.33 | 25 | 0 |



| Tree Number | \% ANDS | \% Diff Rep | \% Total Rep | Top Gate |
| :---: | :---: | :---: | :---: | :---: |
| Rando47 | 70 | 40.28 | 27.42 | 0 |
| Rando48 | 50 | 66.67 | 33.33 | 1 |
| Rando49 | 60 | 37.5 | 25 | 1 |
| Rando50 | 50 | 50 | 33.33 | 1 |
| Rando5 1 | 33.33 | 60 | 33.33 | 0 |
| Rando52 | 60.61 | 70.59 | 30 | 0 |
| Rando53 | 61.54 | 52.38 | 31.43 | 1 |
| Rando54 | 38.46 | 14.71 | 12.82 | 1 |
| Rando55 | 33.33 | 60.87 | 34.15 | 1 |
| Rando56 | 50 | 66.67 | 40 | 0 |
| Rando57 | 40 | 62.5 | 29.41 | 1 |
| Rando58 | 70 | 58.82 | 35.71 | 1 |
| Rando59 | 69.57 | 28.57 | 20 | 0 |
| Rando60 | 61.11 | 22.86 | 18.39 | 0 |
| Rando61 | 57.89 | 75 | 29.41 | 1 |
| Rando62 | 61.54 | 66.67 | 34.29 | 1 |
| Rando63 | 33.33 | 60.87 | 34.15 | 1 |
| Rando64 | 63.16 | 28.57 | 22.22 | 0 |
| Rando65 | 54.55 | 40 | 24 | 0 |
| Rando66 | 58.82 | 53.85 | 27.45 | 1 |
| Rando67 | 50 | 50 | 33.33 | 1 |
| Rando68 | 50 | 87.5 | 36.84 | 0 |
| Rando69 | 50 | 50 | 33.33 | 0 |
| Rando 70 | 50 | 16.67 | 14.29 | 1 |
| Rando71 | 50 | 50 | 30 | 1 |
| Rando72 | 60 | 83.33 | 35.71 | 0 |
| Rando73 | 63.64 | 64.71 | 33.85 | 1 |
| Rando74 | 33.33 | 33.33 | 25 | 0 |
| Rando75 | 50 | 47.06 | 28.57 | 1 |
| Rando 76 | 66.67 | 31.25 | 22.22 | 1 |
| Rando 77 | 58.06 | 64.86 | 30.38 | 0 |
| Rando78 | 58.82 | 23.33 | 18.42 | 1 |
| Rando 79 | 50 | 57.14 | 33.33 | 0 |
| Rando80 | 44.44 | 11.54 | 10.34 | 1 |
| Rando81 | 50 | 33.33 | 25 | 0 |
| Rando82 | 66.67 | 69.23 | 33.33 | 0 |
| Rando83 | 64.29 | 38.1 | 26.67 | 1 |
| Rando84 | 42.11 | 20.51 | 17.02 | 0 |
| Rando85 | 61.54 | 42.31 | 27.5 | 0 |
| Rando86 | 66.67 | 28.57 | 22.22 | 1 |
| Rando87 | 36.36 | 22.73 | 17.24 | 0 |
| Rando88 | 54.55 | 13.64 | 12 | 0 |
| Rando89 | 41.67 | 36.59 | 24.59 | 0 |
| Rando90 | 50 | 0 | 0 | 1 |
| Rando91 | 62.5 | 50 | 29.59 | 1 |
| Rando92 | 63.41 | 71.88 | 35.38 | 1 |
| Rando93 | 42.11 | 30 | 21.82 | 0 |
| Rando94 | 66.67 | 28.57 | 22.22 | 1 |
| Rando95 | 36.36 | 36.36 | 25.81 | 1 |
| Rando96 | 66.67 | 12.5 | 11.11 | 1 |
| Rando97 | 50 | 20 | 16.67 | 0 |
| Rando98 | 36.36 | 23.08 | 17.39 | 0 |


| Tree Number | \% ANDS | \% Diff Rep | \% Total Rep | , Top Gate |
| :---: | :---: | :---: | :---: | :---: |
| Rando99 | 65.38 | 65 | 33.77 | 1 |
| Rand100 | 68.42 | 55.56 | 28.30 | 1 |
| Rand101 | 66.67 | 14.29 | 12.5 | 1 |
| Rand102 | 66.67 | 28.57 | 22.22 | 1 |
| Rand103 | 38.46 | 21.74 | 17.24 | 0 |
| Rand104 | 62.5 | 59.09 | 31.71 | 0 |
| Rand105 | 46.67 | 12.12 | 10.81 | 0 |
| Rand106 | 38.71 | 62.16 | 30.26 | 1 |
| Rand107 | 50 | 12.5 | 11.11 | 0 |
| Rand108 | 37.5 | 74.29 | 34.21 | 1 |
| Rand109 | 55.56 | 19.64 | 16.18 | 0 |
| Rand110 | 41.67 | 60 | 29.51 | 0 |
| Rand111 | 42.11 | 68.18 | 31.91 | 0 |
| Rand112 | 50 | 40 | 28.57 | 1 |
| Rand113 | 50 | 72.73 | 28.57 | 1 |
| Rand114 | 50 | 33.33 | 25 | 1 |
| Rand115 | 38.1 | 44.83 | 28.26 | 0 |
| Rand116 | 58.33 | 69.7 | 33.82 | 1 |
| Randl17 | 50 | 29.41 | 21.74 | 0 |
| Rand118 | 42.11 | 20.51 | 17.02 | 0 |
| Rand119 | 57.14 | 20 | 16.22 | 1 |
| Rand120 | 60 | 17.95 | 14.89 | 1 |
| Rand121 | 38.89 | 35.14 | 26 | 1 |
| Rand122 | 50 | 20 | 16.67 | 0 |
| Rand123 | 55.56 | 23.53 | 17.39 | 1 |
| Rand124 | 58.33 | 20.83 | 16.67 | 0 |
| Rand125 | 50 | 35.71 | 26.32 | 0 |
| Rand 126 | 40 | 37.84 | 26.42 | 0 |
| Rand127 | 41.67 | 10.71 | 9.68 | 1 |
| Rand128 | 58.33 | 62.86 | 32.35 | 1 |
| Rand129 | 62.5 | 30 | 23.08 | 1 |
| Rand130 | 61.54 | 56.52 | 32.5 | 0 |
| Rand131 | 50 | 28.57 | 20 | 1 |
| Rand132 | 48.39 | 66.67 | 30.95 | 1 |
| Rand133 | 61.76 | 48.21 | 27.55 | 0 |
| Rand134 | 37.93 | 54.17 | 30.59 | 0 |
| Rand 135 | 37.5 | 69.7 | 35.94 | 1 |
| Rand 136 | 50 | 50 | 33.33 | 1 |
| Rand137 | 50 | 23.81 | 19.23 | 1 |
| Rand138 | 60 | 50 | 29.03 | 1 |
| Rand139 | 42.86 | 55.17 | 32 | 1 |
| Rand140 | 33.33 | 11.11 | 9.09 | 0 |
| Rand141 | 41.67 | 60 | 29.51 | 0 |
| Rand142 | 37.5 | 63.04 | 29.9 | 1 |
| Rand 143 | 64.71 | 35.71 | 25 | 0 |
| Rand144 | 37.93 | 54.17 | 30.59 | 0 |
| Rand145 | 45.45 | 3.03 | 2.94 | 0 |
| Rand146 | 50 | 23.81 | 19.23 | 1 |
| Rand147 | 38.89 | 67.44 | 32.22 | 1 |
| Rand148 | 33.33 | 11.11 | 9.68 | 0 |
| Rand149 | 40.91 | 10.53 | 9.38 | 0 |
| Rand150 | 37.93 | 54.55 | 32.43 | 0 |


| Tree Number | \%ANDS | \% Diff Rep | \% Total Rep | Top Gate |
| :---: | :---: | :---: | :---: | :---: |
| Rand151 | 50 | 14.29 | 12.5 | 0 |
| Rand152 | 50 | 50 | 33.33 | 1 |
| Rand153 | 43.75 | 61.9 | 27.66 | 1 |
| Rand154 | 45.45 | 33.33 | 23.33 | 0 |
| Rand155 | 60 | 30.30 | 21.28 | 1 |
| Rand156 | 70 | 27.27 | 21.43 | 1 |
| Rand158 | 30.61 | 47.89 | 27.64 | 1 |
| Lisaba9 | 29.41 | 12.20 | 10.87 | 0 |
| Lisab30 | 63.16 | 37.50 | 26.67 | 1 |
| Lisab60 | 57.14 | 31.25 | 21.74 | 1 |
| Lisab59 | 50 | 0 | 0 | 0 |
| Lisab31 | 64.52 | 68.09 | 34.04 | 1 |
| Lisaba4 | 57.69 | 34.09 | 23.81 | 1 |
| Lisab57 | 38.89 | 57.14 | 34.78 | 0 |
| Lisab28 | 33.33 | 0 | 0 | 0 |
| Lisab10 | 51.85 | 45.83 | 27.50 | 0 |
| Lisab35 | 36.84 | 30 | 21.05 | 0 |
| Lisab51 | 62.50 | 10.53 | 9.52 | 1 |
| Lisab44 | 30 | 55 | 33.33 | 0 |
| Lisab25 | 46.67 | 38.46 | 27.03 | 0 |
| Lisab54 | 33.33 | 26.67 | 21.05 | 0 |
| Lisab53 | 20 | 11.11 | 10 | 0 |
| Lisab52 | 64.52 | 84.21 | 50 | 1 |
| Lisab34 | 62.50 | 92.31 | 57.04 | 1 |
| Lisab36 | 34.78 | 9.52 | 8.70 | 1 |
| Lisab42 | 57.14 |  |  | 1 |

## Appendix III

# Number of BDD Nodes For Each Ordering Heuristic For All Benchmark Fault Trees Used in Thesis Studies 

Key To Table:

Scheme 1 = Top-down, left-right
Scheme 2 = Modified top-down, left-right
Scheme 3 = Depth-first
Scheme $4=$ Modified depth-first
Scheme 5 = Priority depth-first
Scheme $6=$ Modified priority depth-first
Column 1 = Tree name and in brackets number of minimal cuts sets in tree
Column 2 = Scheme Option
Column 3 = Number of ite calculations required to generated BDD before minimisation
(If equals 999999 then the calculation exceeded computer capacity)
Column $4=$ Number of ite calculations after minimisation
Column 5 = Difference Between ite calculations before and after minimisation
Column $6=$ Number of nodes in BDD before minimisation
Column $7=$ Number of Non-repeated nodes before minimisation
Column $8=$ Number of nodes in BDD after minimisation
Column $9=$ Number of non-repeated nodes after minimisation
Column $10=$ Difference in number of nodes before and after minimisation
999999 = Calculation could not be performed (in excess of limits)

| Tree | Scheme | Tte b4 | Ite after | Diff | Node 64 | Non-rep | Node af | Nonrep | $\text { Differ }^{2}$ |
| :---: | :---: | :---: | :---: | :---: | :---: | :---: | :---: | :---: | :---: |
| aaaaaaa(2) | 1 | 6 | 7 | 1 | 4 , | 4 | 4 | 4 | 0 |
|  | $\bigcirc 2$ | 7 | 7 | 0 | 3 | , 3 | 3 | 3 | 0 \% |
|  | 3 | 6 | 7 | 1 | 4 | 4 | 4 | 4 | 0 |
|  | 4 | \% | 7 | 0 | 3 | 3 | 3 | 3 | 0 |
|  | 5 | 6 | 7 | 1 | 4 | 4 | 4 | 4 | 0 |
|  | 6 | 4 | 7. | 0 | 3 | 3. | 3 | 3 | 0 |
| Artqual (7) | 1 | 21 | 21 | 0 | 14 | 14 | 11 | 8 | 3 |
|  | 2 | 21 | 21 | 0 | 11 | 8 | 11 | 8 | 0 |
|  | 3 | 22 | 22 | 0 | 11 | 8 | 11 | 8 | 0 |
|  | 4 | 21 | 21 | 0 | 11 | 8 | 11 | 8 | 0 |
|  | 5 | 24 | 24 | 0 | 17 | 8 | 17 | 8 | 0 |
|  | 6 | 24 | 24 | 0 | 17 | 8 | 17 | 8 | 0 |
| Arttree <br> (2) | 1. | 9 | 12 | 3 | \% 7 | W 5 | 5 | 5 , | , 2 , |
|  | 2. | 11 | 13 | 2 | \% ${ }^{5}$ | 4, 4. | $4{ }^{\text {a }}$ | 4 . | , 1 |
|  | 3 | - 9 | -12 | 3 | \% $\mathrm{K}^{7}$ | 5 | 5 | 5 \% | - 2 |
|  | 4 | 411 | 413 | 2 | 4, | W, 4. | + 4.4 | 4, , | -1 1 m |
|  | 5 | 5.10. | - 10 | - 0 | 4. | 3-4. ${ }^{2}$ | - 4 | 4. | 0 , |
|  | 6 | \% 10 | +10 | 0 \% | 4, | 4, 4, | -4 | 4 | - 0 , ${ }^{\text {a }}$ |



|  | 5 | 163 | 189 | 26 | 167835 | 60 | 115611 | 61 | 52224 |
| :---: | :---: | :---: | :---: | :---: | :---: | :---: | :---: | :---: | :---: |
|  | 6 | 161 | 190 | 26 | 166390 | 61 | 115322 | 62 | 51068 |
| Ch8tree (5) | 1 | 23 | 23 | 0 | 11 | 9 | 11 | 9 | 0 |
|  | 2 | 24 | 24 | 0 | 11 | 9 | 11 | 9 | 0 |
|  | 3 | 22 | 22 | 0 | 11 | 9 | 11 | 9 | 0 |
|  | 4 | 23 | 23 | 0 | 11 | 9 | 11 | 9 | 0 |
|  | 5 | 26 | 26 | 0 | 15 | 9 | 15 | 9 | 0 |
|  | 6 | 26 | 26 | 0 | 15 | 9 | 15 | 9 | 0 |
| $\begin{aligned} & \text { dre1019 } \\ & (63) \end{aligned}$ | 1 | 48 | 49 | 1 | 72 | 20 | 72 | 20 | 0 |
|  | 2 | 98 | 98 | 0 | 69 | 19 | 69 | 19 | 0 |
|  | 3 | 48 | 49 | 1 | 72 | 20 | 72 | 20 | 0 |
|  | 4 | 98 | 98 | 0 | 69 | $\therefore 19$ | $\therefore 69$ | 19 | 0 |
|  | 5 | 48 | 49 | 1 | 72 | 20 | 72 | 20 | 0 |
|  | 6 | 48 | 49 | 1 | 72 | 20 | 72 | 20 | 0 |
| $\begin{gathered} \text { dre1032 } \\ (75) \end{gathered}$ | 1 | 55 | 56 | 1 | 91 | 22 | 91 | 22 | 0 |
|  | 2 | 125 | 125 | 0 | 87 | 21 | 87 | 21 | 0 |
|  | 3 | 55 | 56 | 1 | 91 | 22 | 91 | 22 | 0 |
|  | 4 | 125 | 125 | 0 | 87 | 21 | 87 | 21 | 0 |
|  | 5 | 55 | 56 | 1 | 91 | 22 | 91 | 22 | 0 |
|  | 6 | 55 | 56 | 1 | 91 | 22 | 91 | 22 | 0 |
| $\begin{aligned} & \mathrm{dre} 1057 \\ & (2100) \end{aligned}$ | 1 | 92 | 93 | 1 | 2590 | 44 | 2590 | 44 | 0 |
|  | 2\% | 180 | 180 | 0 | 2478 | 43 | 2478 | 43 | 0 |
|  | 3 | 104 | 116 | 12 | 2856 | 33 | 2716 | 33 | 140 |
|  | 4 | 191 | 191 | 0 | 2712 | 32 | 2712 | 32 | 105 |
|  | 5 | 101, | 113 | 12 | 2619 | 33 | 2491 | 33 | 384 |
|  | 6 | 101 | 113 | 12 | 2619 | 33 | 2491 | 33 | 352 |
| $\begin{aligned} & \text { dre1058 } \\ & (11934) \end{aligned}$ | 1 | 265 | 464 | 199 | 84902 | 172 | 24938 | 183 | 265908 |
|  | 2 | 380 | 446 | 66 | 24764 | 153 | 24764 | 162 | 265908 |
|  | 3 | 494 | 698 | 204 | 57222 | 181 | 21787 | 245 | 93585 |
|  | 4 | 530 | 602 | 72 | 27090 | 126 | 20948 | 160 | 93585 |
|  | 5 | 459 | 778 | 319 | 111374 | 233 | 25080 | 323 | 150350 |
|  | 6 | 462 | 781 | 319 | 111374 | 233 | 25080 | 390 | 150350 |
| $\frac{\mathrm{dre} 1059}{}(36990$ | 1 . | 326 | 539 | 213 | 324532 | 223 | 61596 | 280 | 273668 |
|  | 2 | 476 | 566. | 90 | 324532 | 213 | 61476 | 258 | 273668 |
|  | 3 | 771 | 1263 | 492 | 193068 | 381 | 103650 | 549 | 89418 |
|  | 4 | 1095 | 1447 | 352 | 126718 | 361 | 102760 | 519 | 23958 |
|  | 5 | 999999 |  |  | प6\% |  | \% | \% | 4, |
|  | 6 | 999999 |  | \%ab | \% | \% | Ta | , | \% |
| $\begin{aligned} & \hline \text { dresden } \\ & (11934) \end{aligned}$ | 1 | 999999 |  |  |  |  |  |  |  |
|  | 2 | 999999 |  |  |  |  |  |  |  |
|  | 3 | 999999 |  |  |  |  |  |  |  |
|  | 4 | 999999 |  |  |  |  |  |  |  |
|  | 5 | 2582 | 3104 | 522 | 221217 | 310 | 27125 | 398 | 194092 |
|  | 6 | 2582 | 3105 | 522 | 221217 | 310 | 27125 | 398 | 194092 |
| Emerh20 <br> (13) | 1 | 25 | 25 | 0 | 16 | 10 | 16 | $\times 10$ | 0 |
|  | 2 | 28 | 28 , | 0 | - 16 | - 10 | 16 . | -10 | 0 |
|  | 3 | 25 | 25 , | 0 | $\cdots 16$ | 10 | 16 | 10 | - 0 |
|  | 4 | 28 | 28 | 0 \% | 16 | 10 | 16 | 10 | 0 |
|  | 5 | 29 | 35 | 6 | 25 | 10 | , 16 | +10 | 9 |
|  | 6 . | -29 | 35 | 6. | 25 \% | - 10 | 16 | -10 | 9 |
| Fatram2 <br> (6) | 1 | 25 | 27 | 2 | 12 | 12 | 12 | 12 | 0 |
|  | 2 | 26 | 26 | 0 | 10 | 10 | 10 | 10 | 0 |


|  | 3 | 25 | 27 | 2 | 12 | 12 | 12 | 12 | 0 |
| :---: | :---: | :---: | :---: | :---: | :---: | :---: | :---: | :---: | :---: |
|  | 4 | 26 | 26 | 0 | 10 | 10 | 10 | 10 | 0 |
|  | 5 | 29 | 32 | 3 | 16 | 14 | 14 | 14 | 2 |
|  | 6 | 29 | 31 | 2 | 14 | 13 | 13 | 13 | 1 |
| $\begin{aligned} & \text { Hpisf02 } \\ & \therefore(255) \end{aligned}$ | 1 | 387 | 707 | 320 | 214237 | 200 | 476 | 136 | 213761 |
|  | 2 | 494 | 814 | 320 | 214237 | 200 | 476 | 136 | 213761 |
|  | 3 | 336 | 456 | 120 | 164539 | 98 | 296 | 98 | 164243 |
|  | 4 | 444 | 564 | 120 | 164539 | 98 | 296 | 98 | 164243 |
|  | 5 | 283 | 425 | 142 | 290695 | 93 | 296 | 99 | 360681 |
|  | 6 | 283 | 425 | 142 | 290695 | 93 | 296 | 99 | 360681 |
| Hpisf03 <br> (71) | 1 | 108 | 139 | 31 | 246 | 43 | 86 | 43 | 160 |
|  | 2 | 185 | 200 | 15 | 202 | 42 | 82 | 42 | 120 |
|  | 3 | 108 | 139 | 31 | 246 | 43 | 86 | 43 | 289 |
|  | 4 | 185 | 200 | 15 | 202 | 42 | 82 | 42 | 225 |
|  | 5 | 152 | 233 | 81 | 431 | 59 | 102 | 59 | 329 |
|  | 6 | 153 | 234 | 81 | 409 | 59 | 101 | 59 | 308 |
| $\underset{(7777)}{\text { Hpisf21 }}$ | 1 | 503. | 702 | 199 | 18535 | 188 | 8772 | 188 | 54766 |
|  | 2 | 1203 | 1398 | 195 | 18451 | 186 | 8770 | 186 | 42572 |
|  | 3 | 507 | 595 | 88 | 10635 | 182 | 8619 | 162 | 11155 |
|  | 4 | 1207 | 1272 | 65 | 10593 | 180 | 8617 | 160 | 9101 |
|  | 5 | 524 | 612 | 88 | 10635 | 182 | 8619 | 162 | 2016 |
|  | 6 | 508 | 596 | 88 | 10635 | 182 | 8619 | 162 | 2016 |
| Jdtreel <br> (4) | 1 | 17 | 17 | 0 | 12 | 10 | 12 | 10 | 0 |
|  | 2 | 17 | 17 | 0 | 12 | 10 | 12 | 10 | 0 |
|  | 3 | 16 | 16 | 0 | 10 | 7 | 10 | 7 | 0 |
|  | 4 | 16 | 16 | 0 | 10 | 7 | 10 | 7 | 0 |
|  | 5 | 15 | 15 | 0 | 11 | 7 | 11 | 7 | 0 |
|  | 6 | 15 | 15 | 0 | 11 | 7 | 11 | 7 | 0 |
| Jdtree2 <br> (4) | 1 | 20 | 20 | 0 | 12 | 10 | 12 | 10 | 0 |
|  | 2 | 20 | $20 \%$ | 0 | 12 | 10 | 12 | 10 | 0 |
|  | 3 | 19 | 19 | 0 | 10 | $\square 7$ | 10 | 7 | 0 |
|  | 4 | 19 | 19 | 0 | 10 | 7 | 10 | 7 | 0 |
|  | 5 | 18 | 18 | 0 | 11 | 7 | 11 | 7 | 0 |
|  | 6 | 18 | 18 | 0 | 11 | 7 | 11 | 7 | 0 |
| Jdtree3 (36) | 1 | 67 | 67 | 0 | 79 | 37 | 79 | 37 | 0 |
|  | 2 | 67 | 67 | 0 | 79 | 37 | 79 | 37 | 0 |
|  | 3 | 59 | 59 | 0 | 71 | 21 | 71 | 21 | 0 |
|  | 4 | 59 | 59 | 0 | 71 | 21 | 71 | 21 | 0 |
|  | 5 | 58 | 58 | 0 | 76 | 21 | 76 | 21 | 0 |
|  | 6 | 58 | 58 | 0 | 76 | 21 | 76 | 21 | 0 |
| Jdtree4 (30) | 1 | 72 | 72 | 0 | . 67 | 31 | 67. | 31 | 0 |
|  | 2 | 72 | 72, | 0 | 67. | 31 | 67 | 31 | 0 |
|  | 3 | 66 | 66 | 0 | 59. | 19 | 59 | 19. | 0 |
|  | 4 | 66 | 66 | 0 | - 59 | 19 | 59 | 19 | 0 |
|  | 5 | 55 | 55 | 0 | 63. | 419 | 63 | 19 | 0 |
|  | 6 | 55 | 55 | 0 | 63. | 19 | 63 | 19 | 0 |
| Jdtree5 (10) | 1 | 66 | 113 | 47 | 76 | 35 | 21 | 21 | 55 |
|  | 2 | 66 | 113 | 47 | 76 | 35 | 21 | 21 | 55 |
|  | 3 | 58 | 76 | 18 | 70 | 20 | 20 | 20 | 50 |
|  | 4 | 61 | 80 | 19 | 92 | 22 | 22 | 22 | 70 |
|  | 5 | 57 | 77 | 20 | 72 | 21 | 21 | 21 | 51 |
|  | 6 | 57 | 77 | 20 | 72 | 21 | 21 | 21 | 51 |


| Khictre <br> (21) | 1 | 191 | 216 | 25 | 1244 | 28 | 32 | 27 | 1212 |
| :---: | :---: | :---: | :---: | :---: | :---: | :---: | :---: | :---: | :---: |
|  | 2 | 201 | 226 | 25 | 1244 | 28 | 32 | 27 | 1212 |
|  | 3 | 202 | 243 | 41 | 1182 | 36 | 36 | 31 | 1146 |
|  | 4 | 196 | 221 | 25 | 1134 | 28 | 32 | 27 | 1102 |
|  | 5 | 158 | 190 | 32 | 3377 | 28 | 32 | 27 | 3345 |
|  | -6 | 159 | 1918 | 32 | 3377 | 28 | 32 | 27 | 3345 |
| Modtree <br> (2) | 1 | 13 | 13 | 0 | 4 | 4 | 4 | 4 | 0 |
|  | 2 | 13 | 13 | 0 | 4 | 4 | 4 | 4 | 0 |
|  | 3 | 13 | 13 | 0 | 4 | 4 | 4 | 4 | 0 |
|  | 4 | 13 | 13 | 0 | 4 | 4 | 4 | 4 | 0 |
|  | 5 | 12 | 12 | 0 | 9 | 5 | 5 | 5 | 4 |
|  | 6 | 12 | 12 | 0 | 9 | 5 | 5 | 5 | 4 |
| Nakashi <br> (20) | 1 | 211 | 427. | 216 | 715 | 213 | 73 | 58 | 642 |
|  | 2 | 203 | 414 | 211 | 460 | 115 | 72 | 60 | 388 |
|  | - 3 | 156 | 281 | 125 | 1049 | 70 | 84 | 54 | 965 |
|  | 4 | 153 | 270 | 117 | 766 | 65 | 83 | 53 | 473 |
|  | 5 | 110 | 168 | 58 | 536 | 43 | 64 | 38 | 487 . |
|  | 6 | 109 | 164 | 55 | 550 | 42 | - 63 | 37 \% | 450 |
| Newtre2 <br> (3) | 1 | 17 | 22 | 5 | 11 | 7 | 7 | 7 | 4 |
|  | 2 | 19 | 23 | 4 | 9 | 6 | 6 | 6 | 3 |
|  | 3 | 17 | 22 | 5 | 11 | 7 | 7 | 7 | 4 |
|  | 4 | 19 | 23 | 4 | 9 | 6 | 6 | 6 | 3 |
|  | 5 | 17 | 22 | 5 | 11 | 7 | 7 | 7 | 4 |
|  | 6 | 18 | 23 | 5 | 10 | 7 | 7 | 7 | 3 |
| Newtre3 <br> (2) | 1 | 11 | 14 | 3 | 6 | 4 | 4 | 4 | 2 |
|  | 2 | 11 | 14 | 3 | 6 | 4 | 4 | 4 | 2 |
|  | 3 | 11 | 14 | 3 | 6 | 4 | 4 | 4 | 2 |
|  | 4 | 11 | 14 | 3 | 6 | 4 | 4 | 4 | 2 |
|  | 5 | 11 | 14 | 3 | 6 | 4 | 4 | 4 | - 2 |
|  | 6 | 11 | 14 | 3 | 6 | 4 | 4 2 | 4 \% | 2 |
| Newtree <br> (3) | 1 | 14 | 19 | 5 | 11 | 7 | 7 | 7 | 4 |
|  | 2 | 16 | 20 | 4 | 9 | 6 | 6 | 6 | 3 |
|  | 3 | 14 | 19 | 5 | 11 | 7 | 7 | 7 | 4 |
|  | 4 | 16 | 20 | 4 | 9 | 6 | 6 | 6 | 3 |
|  | 5 | 14 | 19 | 5 | 11 | 7 | 7 | 7 | 4 |
|  | 6 | 15 | 20 | 6 | 10 | 7 | 7 | 7 | 3 |
| Relcour <br> (6) | 1 | 13 | 13 | 0 - | 9 | -6 | 9 | 6 | 0 |
|  | - 2 2 | 14. | 14 \% | - 0 | 4, 9. | 6 | 9 | 6 | 0 , |
|  | - 3 \% | -13 | 13 | 0 | 9 . | 6 | 9 | 6 | 0 |
|  | -4. | . 14 | 14 | 0 | 9 | 6 | 9 | 6 | - 0 |
|  | 5 | 12 | 12 | 0 . | 10 | 6 | 10 , | \% 6 | 0 |
|  | 6 | 12 | 12 | 0 | 10 | 6 | 10 | 6 | 0 \% |
| Rstree 1 (3) | 1 | 15 | 15 | 0 | 4 | 4 | 4 | 4 | 0 |
|  | 2 | 15 | 15 | 0 | 4 | 4 | 4 | 4 | 0 |
|  | 3 | 15 | 15 | 0 | 4 | 4 | 4 | 4 | 0 |
|  | 4 | 14 | 14 | 0 | 4 | 4 | 4 | 4 | 0 |
|  | 5 | 15 | 15 | 0 | 4 | 4 | 4 | 4 | 0 |
|  | 6 | 14 | 14 | 0 | 4 | 4 | 4 | 4 | 0 |
| Rstree2(3) | $\underline{1}$ | $\times 17$ | 17 | - 0 | - 4 | - 4 - 4 | 4 | 4 | 0 |
|  | -2 | 17 | 17 | \% 0 \% | $\bigcirc 4$ | \% 4 . | 4 | 4 | 0 , |
|  | -3 | -17. | 17 | 0 | 4 | - 4 | 4 | 4 | - 0 , |
|  | 4 | -17\% | 17 | 0 | 4 . | 4 | 4. | 4 | 0 |


|  | 5 | 17 | 17 | 0 | 4 | 4 \% | "4 | 4 | 0 |
| :---: | :---: | :---: | :---: | :---: | :---: | :---: | :---: | :---: | :---: |
|  | 6 | 17 | 17 | 0 | 4 | 4 | 4 | 4 | 0 |
| Rstree3 <br> (6) | 1 | 21 | 23 | 2 | 13 | 11 | 13 | 11 | 0 |
|  | 2 | 21 | 23 | 2 | 13 | 11 | 13 | 11 | 0 |
|  | 3 | 23 | 23 | 0 | 11 | 8 | 11 | 8 | 0 |
|  | 4 | 23 | 23 | 0 | 11 | 8 | 11 | 8 | 0 |
|  | 5 | 23 | 23 | 0 | 11 | 8 | 11 | 8 | 0 |
|  | 6 | 23 | 23 | 0 | 11 | 8 | 11 | 8 | 0 |
| Rstree 4 <br> (4) | 1 | 15 | 15 | 0 | 5 | 5 | 5 | 5 | 0 |
|  | 2 | 15 | 15 | 0 | 5 | 5 | 5 | 5 | 0 |
|  | 3 | 15 | 15 | 0 | 5 | 5 | 5 | 5 | 0 |
|  | 4 | 15 | 15 | 0 | 5 | 5 | 5 | 5 | 0 |
|  | 5 | 15 | 15 | 0 | 5 | 5 | 5 | 5 | 0 |
|  | 6 | 15 | 15 | 0 | 5 | 5 | 5 | 5 | 0 |
| Rstree5 <br> (2) | 1 | 10 | 10 | 0 | 2 | 2 | 2 | 2 | 0 |
|  | 2 | 9 | 9 | 0 | 2 | 2 | 2 | 2 | 0 |
|  | 3 | 10 | 10 | 0 | 2 | 2 | 2 | 2 | 0 |
|  | 4 | 10 | 10 | 0 | 2 | 2 | 2 | 2 | 0 |
|  | 5 | 10 | 10 | 0 | 2 | 2 | 2 | 2 | 0 |
|  | 6 | 10 | 10 | 0 | 2 | 2 | 2 | 2 | 0 |
| Rstree6 <br> (4) | 1 | 16 | 16 | 0 | 4 | 4 | $\times 4$ | 4 | 0 |
|  | 2 | 17 | 17 | 0 | 4 | 4 | 4 | 4 | 0 |
|  | 3 | 16 | 16 | 0 | 4 | 4 | 4 | 4 | 0 |
|  | 4 | 14.3 | 14 | 0 | 4 | 4 | 4 | 4 | 0 |
|  | 5 | 14 | 14 | 0 | 4 | 4 | 4 | 4 | 0 |
|  | 6 | 14 | 14 | 0 | - 4 | - 4 | 4 | 4 | 0 |
| Rstree7 <br> (8) | 1 | 46 | 55 | 9 | 33 | 18 | 23 | 18 | 7 |
|  | 2 | 50 | 59 | 9 | 33 | 18 | 23 | 18 | 7 |
|  | 3 | 38 | 47 | 9 | 33 | 13 | 20 | 13 | 22 |
|  | 4 | 41 | 50 | 9 | 33 | 13 | 20 | 13 | 22 |
|  | 5 | 40 | 53 | 13 | 55 | 14 | 20 | 14 | 35 |
|  | 6 | 40 | 53 | 7 | 55 | 14 | 20 | 14 | 10 |
| Usatree <br> (2) | 1, | 9 | , 10 | 1 . | 5 \% | 5 | 5 | 5 | 0 |
|  | - 2 \% | +10 | - 10 | + 0 | 4 | 4 | 4 | 4 | O 0 |
|  | 3 | 9 | \% 10 | , 1 | 5 | , 5 | 5 | 5 | 0 \% |
|  | 4 4 | 10 | - 10 | 0 | \% 4 | 4, 4 | - 4 | 4 | 0 |
|  | 5 | 10 | \% 12 | - 2 | 7 | 6 | 6.. | 6 | - 1 |
|  | 6 | 10 | \% 10 | \% | 5, | 5 | 5 | 5 | 0 \% |
| Worrell (10) | 1 | 36 | 38 | 2 | 19 | 16 | 18 | 16 | 1 |
|  | 2 | 35 | 37 | 2 | 19 | 15 | 18 | 15 | 1 |
|  | 3 | 34 | 37 | 3 | 17 | 13 | 16 | 13 | 1 |
|  | 4 | 34 | 37 | 3 | 17 | 13 | 16 | 13 | 1 |
|  | 5 | 34 | 37 | 3 | 17 | 12 | 15 | 12 | 2 |
|  | 6 | 34 | 37 | 3 | 17 | 12 | 15 | 12 | 2 |
| Hpisf36 <br> (61) <br> a <br> , | - 1 | 134 | 177 | 43 | - 322 | $42^{\text {xa }}$ | -80 | 42 | 242, |
|  | 2 | 209 | 242 | 33 | 178 | 40 \% | -72 | 40 , | 106 |
|  | 3 | 142 | 178 | 36 | 220 | 42 | 80 | 42 | 140 |
|  | 4 | 217 | 239 | 22 | 132 | \% 40 和 | 72 | 40 | , 60 |
|  | 5 | 118 | 176 | 58 | 890 K | 46 . | -844.9 | 44 | 8806 |
|  | 6 | 118 | 176 | 58 | 890 | 46. | , 84\% | 44. | 806 |
| Trials1 <br> (45) | 1 | 297 | 479 | 182 | 1190 | 139 | 158 | 95 | 1032 |
|  | 2 | 298 | 471 | 173 | 913 | 138 | 157 | 94 | 756 |


|  | 3 | 271 | 446 | 175 | 2264 | 106 | 200 | 85 | 2064 |
| :---: | :---: | :---: | :---: | :---: | :---: | :---: | :---: | :---: | :---: |
|  | 4 | 269 | 435 | 166 | 1842 | 105 | 197 | 83 | 1645 |
|  | 5 | 289 | 470 | 181 | 1678 | 104 | 169 | 84 | 1509 |
|  | 6 | 277 | 463 | 186 | 1246 | 100 | 169 | 84 | 1077 |
| Trials2 | 1 | 77 | 87 | 10 | 37 \% | 17 | 15 | 14 | 22 |
| (6) | 2 | 78 | 89 | 11 | 37 | 17 | 15 | 14 | 22 |
|  | 3 | 78 | 88 | 10 | 29 | 17 | 16 | 15 | 13 |
|  | 4 | 78 | 89 | 11 | 37 | 17 | 15 | 14 | 22 |
|  | 5 | -89 | 101 | 12 | 53 | 18 | 14 | 14 | 39 |
|  | 6 | 93 | 105 | 12 | 43 | 18 | 15 | 15 | 28 |
| Trials 3 | 1 | 142 | 142 | 0 | 2 | 2 | 2 | 2 | 0 |
| (1) | 2 | 150 | 150 | 0 | 2 | 2 | 2 | 2 | 0 |
|  | 3 | 160 | 160 | 0 | 2 | 2 | 2 | 2 | 0 |
|  | 4 | 168 | 168 | 0 | 2 | 2 | 2 | 2 | 0 |
|  | 5 | 174 | 174 | 0 | 2 | 2 | 2 | 2 | 0 |
|  | 6 | 174 | 174 | 0 | 2 | 2 | 2 | 2 | 0 |
| Trials 4 | 1 | 414 | 554 | 140 | 312 | 115 | 162 | 102 | 150 |
| (49) | 2 | 415 | 555 | 140 | 302 | 115 | 163 | 103 | 139 |
|  | 3 | 506 | 679 | 173 | 688 | 138 | 167 | 116. | 521 |
|  | 4 , | 515 | 675 | 160 \% | 637 | 133 | 164 | 113 | 473 |
|  | 5 | 651 | 946 | 295 | 1940 | 183 | 199 | 133 | 1741 |
|  | 6 | 651 | 946 | 295 | 1940 | 183 | 199 | 133 | 1741 |
| Random | 1 | 23 | 23 | 0 | 7 | 7 | 7 | 7 | 0 |
| $1$ | 2 | 26 | 26 | 0 | 6 | 6 | 6 | 6 | 0 |
| (5) | 3 | 23 | 23 | 0 | 7 | 7 | 7 | 7 | 0 |
|  | 4 | 27 | 30 | 3 | 8 | 6 | 8 | 8 | 0 |
|  | 5 | 21 | 27 | 6 | 11 | 9 | 9 | 9 | 2 |
|  | 6 | 21 | 23 | 6 | 11 | 9 | 9 | 9 | 2 |
| Random | 1 | 14 | 14 | 0 | 2 | 2 | 2 | 2 | 0 |
| - 2 | 2. | 16 | 16 | 0 | 2 | 2 | 2 | 2 | - 0 |
| - (2) | 3 | 14 | 14 | 0 | 2 | 2 | 2 | 2 | 0 |
|  | 4 | 16 | 16 | 0 | 2 | 2 | 2 | 2 | $\because 0$ |
|  | 5 | 14 | 14 | 0 | 2 | 2 | 2 | 2 | 0 |
|  | 6 | 15 | 15 | 0 | 2 | - 2 | 2 | 2 | 0 |
| Random | 1 | 513 | 661 | 148 | 2471 | 94 | 447 | 90 | 2024 |
| 3 | 2 | 523 | 660 | 235 | 2341 | 93 | 436 | 88 | 1905 |
| (235) | 3 | 587 | 755 | 168 | 4874 | 81 | 357 | 81 | 4517 |
|  | 4 | 598 | 756 | 158 | 4374 | 80 | 354 | 79 | 4020 |
|  | 5 | 650 | 866 | 216 | 437189 | 151 | 1007 | 97 | 436182 |
|  | 6 | 616 | 794 | 178 | 286685 | 143 | 1001 | 93 | 285684 |
| Random | 1 1\% | 16 | 16 | 0 , | 5 , | 5 | 5. | 5 | 0 0, |
| + 4 | 2 | 20 . | 20 | 0 | 5 | 5 | 5 | 5 \% | 0 |
| \% (5) | 3 | 16 | 16 | 0 | 5 | 5 | 5 | 5 | - 0 |
|  | 4 | 20 | 20 | 0 | 5 , | 5 | 5 | 5 | 0 |
|  | 5\% | 18 | 18 | 0 | 5 , | - 5 | - 5 | 5. | - 0 , |
|  | \% 6 | 18 | 18 | - 0 | 5 5 | - 5 | 5 | 5. | 0 |
| Random | 1 | 1521 | 3313 | 1792 | 71851 | 917 | 321 | 190 | 71530 |
| $6$ | 2 | 1539 | 3314 | 1775 | 40210 | 924 | 319 | 187 | 39891 |
|  | 3 | 999999 |  |  |  |  |  |  |  |
|  | 4 | 999999 |  |  |  |  |  |  |  |
|  | 5 | 999999 |  |  |  |  |  |  |  |
|  | 6 | 999999 |  |  |  |  |  |  |  |


| Random 7 <br> (1) | 1 | 14 | 14 | 0 | 4 | 4 | 4 | 4 | 0 |
| :---: | :---: | :---: | :---: | :---: | :---: | :---: | :---: | :---: | :---: |
|  | 2 | 16 | 16 | 0 | 4 | 4 | 4 | 4 | 0 |
|  | 3 | 14 | 14 | 0 | 4 | 4 | 4 | 4 | 0 |
|  | 4 | 14 | 14 | 0 | 8 | 5 | 4 | 4 | 4 |
|  | 5 | 16 | 16 | 0 | 4 | 4 | 4 | 4 | 0 |
|  | 6 | 15 | 15 | 0 | 4 | 4 | 4 | 4 | 0 |
| $\begin{array}{\|c\|} \hline \text { Random } \\ 8 \\ (4) \end{array}$ | 1 | 66 | 80 | 14 | 37 | 19 | 16 | 16 | 21 |
|  | 2 | 73 | 86 | 13 | 36 | 18 | 15 | 15 | 21 |
|  | 3 | 66 | 77 | 11 | 31 | 15 | 15 | 15 | 16 |
|  | 4 | 78 | 94 | 16 | 57 | 18 | 23 | 18 | 34 |
|  | 5 | 65 | 77 | 12 | 32 | 16 | 16 | 16 | 32 |
|  | 6 | 69 | 80 | 11 | 31 | 15 | 15 | 15 | 16 |
| $\begin{aligned} & \text { Random } \\ & 9 \\ & \text { (2) } \end{aligned}$ | 1 \% | 36 | 36 | 0 | 6 | 6 | 6 | 6 | 0 |
|  | 2 | 37 | $\begin{array}{r}36 \\ \hline\end{array}$ | 0 | 6 | 6 | 6 | 6 | 0 |
|  | 3 | 36 | 36 | 0 | 6 | 6 | 6 | 6 | 0 |
|  | 4 | 36 | 36 | 0 | 10 | 10 | 10 | 10 | 0 |
|  | 5 | 34 | 34. | 0 | 10 | 6 | 10 | 6 | 0 |
|  | 6 | 34 | 34 | 0 | 10 | 6 | 10 | 6 | 0 |
| Rando10 <br> (4) | 1 | 15 | 15 | 0 | 4 | 4 | 4 | 4 | 0 |
|  | 2 | 17 | 17 | 0 | 4 | 4 | 4 | 4 | 0 |
|  | 3 | 15 | 15 | 0 | 4 | 4 | 4 | 4 | 0 |
|  | 4 | 15 | 15 | 0 | 4 | 4 | 4 | 4 | 0 |
|  | 5 | 17 | 17 | 0 | 4 | 4 | 4 | 4 | 0 |
|  | 6 | 16 | 16 | 0 | 4 | 4 | 4 | 4 | 0 |
| Rando12 | 1 | 674 | 1239 | 565 | 10259 | 333 | 252 | 155 | 10007 |
|  | 2 | 696 | 1252 | 556 | 9545 | 328 | 253 | 156 | 9292 |
|  | 3 | 698 | 1264 | 566 | 14922 | 323 | 341 | 156 | 14581 |
|  | 4 | 719 | 1283 | 564 | 12812 | 324 | 342 | 157. | 12470 |
|  | 5 | 999999 | $\bigcirc$ |  | , m. | , | , |  | \% |
|  | 6 | 999999 |  | \% | \% 6 | 1 | $2 \times 3$ | \% ${ }^{\text {a }}$ |  |
| Randol3 <br> (73) | 1 | 887 | 1066 | 179 | 1834 | 125 | 180 | 72 | 1654 |
|  | 2 | 884 | 1045 | 161 | 1394 | 117 | 174 | 67 | 1220 |
|  | 3 | 1063 | 1226 | 163 | 2524 | 101 | 166 | 64 | 2358 |
|  | 4 | 966 | 1127 | 161 | 2265 | 103 | 160 | 58 | 2105 |
|  | 5 | 1200 | 1480 | 280 | 340247 | 173 | 395 | 67 | 339852 |
|  | 6 | 1141 | 1412 | 271 | 2270727 | 165 | 395 | 67 | 2270332 |
| Randol4 <br> (1) | 1 \% | 16 | 16 | 0 | 2 | 2 | 2 | 2 | 0 |
|  | 2 2 | 22 | 22 | 0 | 2 | 2 | 2 | 2 | 0 |
|  | 3 | 16 | 16 | 0 | 2 | 2 | 2 | 2 | 0 |
|  | 4. | 22 | \% 22 | 0 | 2 | 2 | \% 2 | 2 , | 0 \% |
|  | 5 | 16 | -16. | 0 | 2 | 2, | 2 | - 2 | 0 |
|  | 6 | 16 | 16 | 0 | 2 , | 2 | 2 | 2 . | 0 |
| Rando15 <br> (5) | 1 | 29 | 29 | 0 | 5 | 5 | 5 | 5 | 0 |
|  | 2 | 32 | 32 | 0 | 5 | 5 | 5 | 5 | 0 |
|  | 3 | 29 | 29 | 0 | 5 | 5 | 5 | 5 | 0 |
|  | 4 | 31 | 31 | 0 | 5 | 5 | 5 | 5 | 0 |
|  | 5 | 30 | 30 | 0 | 5 | 5 | 5 | 5 | 0 |
|  | 6 | 31 | 31 | 0 | 5 | 5 | 5 | 5 | 0 |
| Rando16(76) | 1 | 1396 | 1541 | 145 | 513 | 117 | 164 | 102 | 349 |
|  | 2 | 1266 | 1391 | 125 | 453 | 111 | -159 | 97. | + 294 |
|  | , 3 | 1406 | \% 1492 | 86. | 476. | 66 | 139 | 57. | - 3 37\% |
|  | 4 | 1353 | 1475 | 122 | -569 | 117 | 224 | 106 | 345 |




| Rando34 <br> (49) | 1 | 236 | 324 | 88 | 345 | 73 | 128 | 74 | 217. |
| :---: | :---: | :---: | :---: | :---: | :---: | :---: | :---: | :---: | :---: |
|  | 2 | 236 | 292 | 56 | 212 | 71 | 125 | 72 | 87 |
|  | 3 | 239 | 300 | 61 | 583 | 68 | - 150 | 51 | 433 |
|  | 4 | 234 | 304 | 70 | 555 | 74 | 157 | 55 | 398 |
|  | 5 | 302 | 387 | 85 | 2183 | 76 | 133 | 52 | 2050 |
|  | 6 | 303 | 389 | 86 | 1825 | 77 | 133 | 53 | 1692 |
| Rando35 <br> (8) | 1 | 418 | 455 | 37 | 184 | 41 | 31 | 31 | 153 |
|  | 2 | 380 | 427 | 33 | 180 | 38 | 33 | 33 | 147 |
|  | 3 | 459 | 493 | 34 | 310 | 41 | 28 | 28 | 282 |
|  | 4 | 336 | 419 | 83 | 198 | 57 | 43 | 43 | 155 |
|  | 5 | 331 | 398 | 67 | 1439 | 47 | 44 | 37 | 1395 |
|  | 6 | 323 | 386 | 63 | 1583 | 48 | 43 | 36 | 1540 |
| Rando36 <br> (10) | 1 | 159 | 193 | 34 | 62 | 28 | 19 | 19 | 43 |
|  | 2 | 165 | 199 | 34 | 62 | 28 | 19 | 19 | 43 |
|  | 3 | 150 | 168 | 18 | 66 | 19 | 18 | 18 | 48 |
|  | 4 , | 157 | $\cdots 175$ | 18 | 66 . | 19 | 18 | 18 | 48 |
|  | 5 | 151 | 185 | 34 | 4910 | 38 | 43 | 23 | 4867 |
|  | 6 | 151 | 185 | 34 | 4910 | 38 | 43 | 23 | 4867 |
| Rando37 <br> (29) | 1 | 314 | 409 | 95 | 238 | 68 | 75 | 58 | 163 |
|  | 2 | 326 | 382 | 56 | 136 | 67 | 71 | 55 | 65 |
|  | 3 | 433 | 673 | 240 | 702 | 142 | 111 | 79 | 591 |
|  | 4 | 377 | 550 | 173 | 910 | 125 | 119 | 86 | 791 |
|  | 5 | 728 | 1024 | 296 | 7040 | 266 | 171 | 112 | 6869 |
|  | 6 | 698 | 998 | 300 | 6109 | 249 | 169 | 112 | 5940 |
| Rando38 <br> (9) | 1 | 96. | 113 | 17 | 66 | 17 | 15 | 15 | 51 |
|  | 2 | 100 | 117 | 17 | 66 | 17 | 15 | 15 | 51 |
|  | 3 | 99 | 114 | 15 | 50 | 14 | 14 | 14 | 36 |
|  | 4 | 104 | 119 | 15 | 197 | 17 | 14 | 14 | 183 |
|  | 5 | 99 | 121. | 22 | 154 | 24 | 15 | 15 | 139 |
|  | 6 | 93 | 114 | 21 | 114 | 23 | 15 | 15 | 99 |
| $\begin{gathered} \text { Rando39 } \\ (51) \end{gathered}$ | 1 | 343 | 476 | 133 | 658 | 115 | 157 | 99 | 501 |
|  | 2 | 364 | 500 | 136 | 626 | 118 | 154 | 99 | 472 |
|  | 3 | 302 | 387 | 85 | 1644 | 111 | 167 | 82 | 1477 |
|  | 4 | 354 | 520 | 166 | 877 | 136 | 187 | 106 | 690 |
|  | 5 | 366 | 438 | 72 | 2084 | 101 | 219 | 76 | 1865 |
|  | 6 | 368 | 440 | 72 | 2106 | 101 | 220 | 76 | 1886 |
| Rando40 <br> (9) | 1 | 59. | $\bigcirc 78$ | 19 | 52 | 22 | 25 | 19 | 27 |
|  | 2 | 68 | . 85 | 17. | 44 | \% 22 | 22 | 18 | 22 |
|  | 3 | 56 | 70 | 14 | 44. | 20 | 23 | \% 19 | 21 |
|  | 4 | 68 | 85 | 17 | 44 | 22 | 22 | 18 | 22 |
|  | 5 | 79 | 101 | 22 | 155. | 421 | . 25 | \% 19 | 130 |
|  | 6 | 79 | 101 | 22 | 155 | 21 | 25 | 19 | 130 |
| Rando41 <br> (I) | 1 | 22 | 22 | 0 | 5 | 5 | 5 | 5 | 0 |
|  | 2 | 28 | 28 | 0 | 5 | 5 | 5 | 5 | 0 |
|  | 3 | 22 | 22 | 0 | 5 | 5 | 5 | 5 | 0 |
|  | 4 | 28 | 28 | 0 | 5 | 5 | 5 | 5 | 0 |
|  | 5 | 26 | 26 | 0 | 5 | 5 | 5 | 5 | 0 |
|  | 6 | 25 | 25 | 0 | 5 | 5 | 5 | 5 | 0 |
| Rando42 <br> (2) | 1 | 60 | 60 | 0 | 5 | 5 | 5 | 5 | 0 |
|  | 2 | 65 | 65 | 0 | 5 | 5 | 5 | 5 | 0 |
|  | 3 3, | 60 | -60 | 0 | 5 . | - 5 | \% 5 | 5 | 0 |
|  | 4 , | 67 | 67 | 0 | 5 , | 5 | 5 | , 5 | 0 |





|  | 5 | 36 | 38 | 2 | 7 | 6 | 6 | 6 | 1 |
| :---: | :---: | :---: | :---: | :---: | :---: | :---: | :---: | :---: | :---: |
|  | 6 | 37 | 39 | 2 | 7 | 6 | 6 | 6 | 1 |
| Rando69 <br> (6) | 1 | 25 | 25 | 0 | 8 | 8 | 8 | 8 | 0 |
|  | 2 | 27 | 27 | 0 | 8 | 8 | 8 | 8 | 0 |
|  | 3 | 25 | 25 | 0 | 8 | 8 | 8 | 8 | 0 |
|  | 4 | 28 | 32 | 4 | 12 | 12 | 12 | 12 | 0 |
|  | 5 | 27 | 34 | 7 | 21 | 9 | 8 | 8 | 13 |
|  | 6 | 27 | 34 | 7 | 16 | 8 | 8 | 8 | 8 |
| Rando70 <br> (27) | 1 | 111 | 144 | 33 | 117 | 39 | 68 | 46 | 49 |
|  | 2 | 117 | 136 | 19 | 87. | 37. | 66 | 45 | 21 |
|  | 3 | 110 | 142 | 32 | 102 | 37 | 63 | 45 | 39 |
|  | 4 | 117 | 136 | 19 | 80 | 36 | 63 | 45 | 17 |
|  | 5 | 124 | 157 | 33 | 491 | 39 | 247 | 41 | 244 |
|  | 6 | 123 | 145 | 22 | 360 | 38 | 246 | 40 \% | 114 |
| Rando71 <br> (2) | 1 | 23 | 24 | 1 | 7 | 7 | 7 | 7 | 0 |
|  | 2 | 23 | 24 | 1 | 7 | 7 | 7 | 7 | 0 |
|  | 3 | 23 | 24 | 1 | 7 | 7 | 7 | 7 | 0 |
|  | 4 | 25 | 29 | 4 | 9 | 7 | 7 | 7 | 2 |
|  | 5 | 24 | 26 | 1 | 8 | 8 | 8 | 8 | 0 |
|  | 6 | 24 | 26 | 1 | 8 | 8 | 8 | 8 | 0 |
| Rando 72 <br> (2) | 1, | 20 | 20 | 0 | - 2 | 2 | 2 , | 2 | 0 |
|  | 2 | 21 | 21 | 0 | 2 | 2 | 2 | 2 | 0 \% |
|  | 3 | 20 | 20 | 0 | 2 | 2 | 2. | 2 | 0 |
|  | 4 | 22 | 22 | 0 | 2 | 2 | 2 | 2 | 0 |
|  | 5 | 23 | 23 | 0 | 11 | 4 | 3 | 3 | 8 |
|  | 6 | 22 | 22 | 0 | 11 | 4 | 3 | 3 | 8 |
| Rando73 <br> (80) | 1 | 257 | 286 | 29 | 614 | 81 | 212 | 61 | 402 |
|  | 2 | 263 | 292 | 29 | 763 | 79 | 218 | 61 | 545 |
|  | 3 | 235 | 287 | 52 | 2837 | 56 | 237 | 38 | 2600 |
|  | 4 | 248 | 293 | 45 | 1407 | 52 | 244 | 42 | 1163 |
|  | 5 | 277 | 324 | 47 | 19901 | 75 | 322 | 55 | 19579 |
|  | 6 | 283 | 330 | 47 | 23339 | 76 | 324 | 55 | 23015 |
| Rando74 <br> (2) | 1 | 15 | 15 | 0 | 2 | 2 | 2 | 2 | 0 \% |
|  | 2 | 17. | 17 | 0 | + 2 | - 2 | 2 , | 2 | 0 |
|  | - 3 | \% 15 | 15 | 0 | - C 2 | $4{ }^{2}$ | 2 2, | , 2 , | O $0^{2}$ |
|  | 4. | 17 | 17 | 0 | 2. | 2 | 2 , | 2 | 0 |
|  | 5 | 17 | 17 | 0 | 2 \% | 2 | $\cdots 2$ | 2 | 0 \% |
|  | 6 | 18 | 18 | 0 | \% 2 , | 2 | $\times 2$ | 2, | 0 , |
| Rando75 <br> (4) | 1 | 68 | 74 | 6 | 16 | 13 | 12 | 12 | 4 |
|  | 2 | 71 | 77 | 6 | 16 | 13 | 12 | 12 | 4 |
|  | 3 | 68 | 74 | 6 | 16 | 13 | 12 | 12 | 4 |
|  | 4 | 83 | 97 | 14 | 32 | 22 | 16 | 16 | 16 |
|  | 5 | 74 | 83 | 9 | 37 | 13 | 21 | 12 | 16 |
|  | 6 | 75 | 84 | 9 | 37 | 13 | 21 | 12 | 16 |
| Rando76 <br> (24) | 1. | 154 | 196. | 42 | \% 363 | 49 \% | 97 | 40 | 266\% ${ }^{\text {' }}$ |
|  | 2 | 165 | 212 | 47 | 299 | 48 , | 9 | 40 | 290 |
|  | 3 | $\underline{147}$ | 171 | 24 | 307 | 31 \% | 83. | 27. | 224 |
|  | - 4 | 166 | 191 | 25 | - 272 | 36 | \% 88 | 33. | 184 |
|  | - 5 | 165 | 199 | 34 | 2772 | 32 緒䜌 | - 170 | \%28 | 2602. |
|  | - 6 | 162 | 197 | 35 | 2260 | 32 , | - 170 | 29 | 2090 , |
| Rando77 <br> (27) | 1 | 371 | 408 | 37 | 244 | 51 | 92 | 48 | 152 |
|  | 2 | 373 | 404 | 31 | 230 | 49 | 90 | 47 | 140 |


|  | 3 | 407 | 448 | 41 | 373 | 56 | 95 | 46 | 278 |
| :---: | :---: | :---: | :---: | :---: | :---: | :---: | :---: | :---: | :---: |
|  | 4 | 385 | 435 | 50 | 1506 | 58 | 96 | 47 | 1410 |
|  | 5 | 547 | 660 | 113 | 20795 | 184 | 100 | 7 | 20695 |
|  | 6 | 530 | 637 | 107 | 18379 | 175 | 98 | 46 | 18281 |
| Rando78 <br> (2) | 1 | 133 | 133 | 0 | 5 | 5 | 5 | 5 | 0 |
|  | 2 | 138 | 138 | 0 | 5 | 5 | 5 | 5 | 0 |
|  | - 3 | 132 | 132 | 0 | 5 | 5 | 5 | 5 | 0 |
|  | 4 | 133 | 133 | 0 | 5 | 5 | 5 | 5 | 0 |
|  | 5 | 150 | 150 | 0 | 7 | $5 \cdots$ | 7 | 5 | 0 |
|  | 6 | 144 | 144 | $\because 0$ | 7 | 5 | 7 | 5 | 0 |
| Rando79 <br> (4) | 1 | 21 | 21 | 0 | 4 | 4 | 4 | 4 | 0 |
|  | 2 | 23 | 23 | 0 | 4 | 4 | 4 | 4 | 0 |
|  | 3 | 21 | 21 | 0 | 4 | 4 | 4 | 4 | 0 |
|  | 4 | 23 | 23 | 0 | 4 | 4 | 4 | 4 | 0 |
|  | 5 | 24 | 24 | 0 | 9 | 5 | 5 | 5 | 4 |
|  | 6 | 24 | 24 | 0 | 4 | 4 | 4 | 4 | 0 |
| $\begin{gathered} \text { Rando80 } \\ (22) \end{gathered}$ | <1\% | 77 | 99 | 22 | $\therefore 131$ | 27 | 49 | 27 | 82 |
|  | 2 | 92 | 112 | \% 20 | -119 | 26 | 45 | 26 | $74 \%$ |
|  | 3 | - 89 | +128 | 39 | 146 | 32 | -44 | 32 | 102 |
|  | 4 | 96 | 119 | 23 | 131 | 24 | 43 | 24 | 88 |
|  | 5 | 83 | ¢ 106 | 23 | -770 | 23 | 226 | -23 | \% 544 |
|  | 6 | -83 | - 106 | 23 | 770 | 23 | 226 | 23 | 544 |
| Rand081 <br> (4) | 1 | 15 | 15 | 0 | 4 | 4 | 4 | 4 | 0 |
|  | 2 | 17 | 17 | 0 | 4 | 4 | 4 | 4 | 0 |
|  | 3 | 15 | 15 | 0 | 4 | 4 | 4 | 4 | 0 |
|  | 4 | 17 | 17 | 0 | 4 | 4 | 4 | 4 | 0 |
|  | 5 | 16 | 16 | 0 | 4 | 4 | 4 | 4 | 0 |
|  | 6 | 16 | 16 | 0 | 4 | 4 | 4 | 4 | 0 |
| Rando82 <br> (5) | -1 | 75 | 75 | 0 | $\therefore 6$ | 6 | 6 | 6 | 0 |
|  | 2\% | 70 | 70 | $\therefore 0$ | 6 | 6 | 6 | 6 | 0 |
|  | 3 | -75 | 75 | 0 | 6 6 | 6 | 6 | 6 | 0 |
|  | 4 | \% 75 | 75 | 0 | 10 | 7 | 6 | 6 | 54 |
|  | 5 | 63 | 68 | 5 | 22 | 8 | 6 | 6 | 16 |
|  | - 6 | 63 | 68 | 5 | 22 | 8 | 6 | \% 6 | 16 |
| Rando83(39) | 1 | 105 | 163 | 58 | 396 | 57 | 153 | 57 | 243 |
|  | 2 | 108 | 132 | 24 | 219 | 52 | 138 | 52 | 81 |
|  | 3 | 121 | 192 | 71 | 372 | 53 | 152 | 53 | 220 |
|  | 4 | 120 | 178 | 58 | 301 | 47 | 141 | 48 | 160 |
|  | 5 | 149 | 286 | 137 | 1160 | 73 | 154 | 73 | 1006 |
|  | 6 | 148 | 283 | 135 | 953 | 71 | 149 | 72 | 804 |
| Rando84 <br> (52) | 1 1 | 272 | 503 | 231 | 2627 | 150 | 149 | 98 | 2478 |
|  | 22 | 277 | 494 | 217 | 1547 | 146 | -145 | 95 | \% 1402 |
|  | 3 | 209 | 366 | $157 \%$ | 1755 | 103 | 137. | + 91 | 1618. |
|  | 44 | 223 | 358 | 135 | $\because 703$ \% | 103 | 147 | -92 | 556 |
|  | 5 | 310 | - 503 | -193 | 76818 | -98\% | -153 | -91-90 | -76665 |
|  | \% 6 | 301 | 486 | 185 | 33010 | $94 \times$ | 153 | -90, | 32857 |
| Rando85 <br> (7) | 1 | 215 | 215 | 0 | 27 | 18 | 18 | 16 | 9 |
|  | 2 | 206 | 206 | 0 | 18 | 16 | 18 | 16 | 0 |
|  | 3 | 197 | 197 | 0 | 26 | 16 | 18 | 15 | 8 |
|  | 4 | 185 | 185 | 0 | 36 | 16 | 18 | 15 | 18 |
|  | 5 | 178 | 217 | 39 | 591 | 39 | 37 | 22 | 554 |
|  | 6 | 174 | 207 | 33 | 427 | 35 | 37 | 23 | 390 |


| Rando86 <br> (1) | 1 | 16 | 16 | 0 | 2 | 2 | 2 | 2 | 0 |
| :---: | :---: | :---: | :---: | :---: | :---: | :---: | :---: | :---: | :---: |
|  | 2 | 22 | 22 | 0 | 2 | 2 | 2 | 2 | 0 |
|  | 3 | - 16 | 16 | 0 | 2 | 2 | 2 | 2 | 0 |
|  | 4 | 22 | 22 | 0 | 2 | 2 | 2 | 2 | 0 |
|  | 5 | 16 | 16 | 0 | 2 | 2 | 2 | 2 | 0 |
|  | 6 | 16 | 16 | 0 | 2 | 2 | 2 | 2 | 0 |
| Rando87 <br> (15) | 1 | 75 | 75 | 0 | 22 | 19 | 22 | 19 | 0 |
|  | 2 | 81 | 81 | 0 | 22 | 19 | 22 | 19 | 0 |
|  | 3 | 75 | 75 | 0 | 21 | 15 | 21 | 15 | 0 |
|  | 4 | 80 | 80 | 0 | 21 | 15 | 21 | 15 | 0 |
|  | 5 | 105 | 128 | 23 | 193 | 23 | 31 | 22 | 162 |
|  | 6 | 103 | 126 | 23 | 98 | 22 | 31 | 22 | 67 |
| Rando88 <br> (29) | 1 | 103 | 190 | 87 | 812 | 56 | 68 | 34 | 744 |
|  | 2 | 105 | 190 | 85 | 812 | 56 | 68 | 34 | 744 |
|  | 3 , | 87 | 122 | 35 | 504 | 29 | 62 | 29 | 442 |
|  | 4 | 89 | 124 | $\square 35$ | 504 | 29 | 62 | 29 | 442 |
|  | 5 , | 91 | 154 | 63 | 3896 | 36 | 130 | - 36 | 3766 |
|  | 6 | 91 | 154 | 63 | 3896 | $36 \%$ | 130 | 36 | 3766 |
| Rando89 <br> (21) | 1 | 975 | 1007 | 32 | 238 | 30 | 39 | 29 | 199 |
|  | 2 | 1012 | 1044 | 32 | 238 | 30 | 39 | 29 | 199 |
|  | 3 | 776 | 832 | 56 | 282 | 39 | 45 | 32 | 237 |
|  | 4 | 1049 | 1130 | 81 | 680 | 53 | 54 | 41 | 626 |
|  | 5 | 569 | 666 | 97 | 4992 | 69 | 51 | 45 | 4941 |
|  | 6 | 554 | 651 | 97 | 3882 | 67 | 51 | 45 | 3831 |
| Rando 90(2) | 1 | 5 | 5 | 0 | 3 | 3 | 3 | 3 | 0 |
|  | 2 \% | 5 | 5 | 0 | 3 | 3 | 3 | 3 | 0 |
|  | 3 | 5 | 5 | 0 | 3 | $3:$ | 3 | 3 | 0 |
|  | 4 | 5 | 5 | 0 | 3 | 3 | 3 | 3 | 0 |
|  | 5 | 5 | 5 | 0 | 3 | 3 , | 3 | 3 | \% |
|  | $\therefore 6$ | 5 | 5 | 0 | 3 | 3 | 3 | 3 | 0 |
| Rando91 <br> (106) | 1 | 819 | 1611 | 792 | 65067 | 424 | 340 | 163 | 64727 |
|  | 2 | 813 | 1564 | 751 | 32108 | 419 | 331 | 161 | 31777 |
|  | 3 | 728 | 1225 | 497 | 644641 | 379 | 468 | 178 | 644173 |
|  | 4 | 761 | 1258 | 497 | 302151 | 398 | 468 | 182 | 301683 |
|  | 5 | 999999 |  |  |  |  |  |  |  |
|  | 6 | 999999 |  |  |  |  |  |  |  |
| Rando92 (58) | 1. | 7540 | 8176 | 636 | 25773 | 332 | 228 | 156 | 25545 |
|  | 2 . | 7562 | 8152 | 590 | 9526 | 341 | 226 | 157. | 9300 |
|  | 3 | 9404 | 10242 | 838 | 93103 | 430 | 304 | 185 | 92799 |
|  | 4 . | 7885 | 8489 | 604 | 37067 | 353 | 283 | 176 | 36784 |
|  | 5 | 999999, |  | \% | \%, | $\because 6$ |  | \% | \%\% |
|  | 6 | 999999 |  | , ${ }^{\text {a }}$ | , | , $\square^{4}$, |  |  | 4 |
| Rando93 <br> (16) | 1 | 403 | 432 | 29 | 104 | 42 | 34 | 31 | 70 |
|  | 2 | 403 | 433 | 30 | 61 | 40 | 36 | 32 | 25 |
|  | 3 | 365 | 406 | 41 | 122 | 44 | 55 | 37 | 67 |
|  | 4 | 433 | 522 | 89 | 205 | 74 | 73 | 54 | 132 |
|  | 5 | 25 | 323 | 68 | 5383 | 60 | 99 | 43 | 5284 |
|  | 6 | 257 | 332 | 75 | 3099 | 63 | 100 | 44 | 2999 |
| Rando94 <br> (1) | 1. | 16 | 16 | 0 | 2 | 2 | 2 | 2 | $\bigcirc$ |
|  | 2 \% | 22 | 22 | 0 | 2 | 2 , | 2 | 2 | \% 0 |
|  | 3 3 | , 16 | $\cdots$ | 0 0, | 2, | - 2 | -2, | -2 | \% c - 0 |
|  | 4 , | 22 | . 22 | 0 | 2 | 2. | 2 | 2 | \% 0 |


|  | 5 | 16 | 16 | 0 | 2 | 2 | 2 | 2 | 0 |
| :---: | :---: | :---: | :---: | :---: | :---: | :---: | :---: | :---: | :---: |
|  | 6 | 16 | 16 | 0 | 2 | 2 | 2 | 2 | 0 |
| $\begin{aligned} & \text { Rando95 } \\ & (31) \end{aligned}$ | 1 | 122 | 149 | 27 | 138 | 41 | 82 | 42 | 56 |
|  | 2 | 125 | 141 | 16 | 86 | 38 | 75 | 37 | 11 |
|  | 3 | 105 | 110 | 5 | 57 | 30 | 56 | 32 | 1 |
|  | 4 | 112 | 116 | 4 | 63 | 35 | 61 | 36 | 2 |
|  | 5 | 143 | 206 | 63 | 856 | 42 | 107 | 42 | 749 |
|  | 6 | 144 | 210 | 66 | 840 | 43 | 106 | 43 | 734 |
| Rando96 <br> (5) | 1 | 17 | 17 | 0 | 7 | 7 | 7 | 7 | 0 |
|  | 2 | 21 | 21 | 0 | 7 | 7 | 7 | 7 | 0 |
|  | 3 | 17 | 17 | 0 | 7 | 7 | 7 | 7 | 0 |
|  | 4 | 21 | 21 | 0 | 7 | 7 | 7 | 7 | 0 |
|  | 5 | 18 | 18 | 0 | 7 m | 7 | 7 | 7 | 0 |
|  | 6 | 18 | 18 | 0 | 7 | 7 | 7 | 7 | 0 |
| Rando97 <br> (2) | 1 | 14 | 14 | 0 | 3 | 3 | 3 | 3 | 0 |
|  | 2 | 14 | 14 | 0 | 3 | 3 | 3 | 3 | 0 |
|  | 3 | 14 | 14 | 0 | 3 | 3 | 3 | 3 | 0 |
|  | 4 | 16 | 16 | 0 | 3 | 3 | 3 | 3 | 0 |
|  | 5 | 14 | 14 | 0 | 3 | 3 | 3 | 3 | 0 |
|  | 6 | 16 | 16 | 0 | 3 | 3 | 3 | 3 | 0 |
| $\begin{aligned} & \text { Rando98 } \\ & (283) \end{aligned}$ | 1 | 679 | 1064 | 385 | 48808 | 237 | 632 | 178 | 48176 |
|  | 2 | 1038 | 350 | ; | 24088 | 688 | 622 | 170 | 23466 |
|  | 3 | 677 | 1099 | 422 | 35038 | 214 | 799 | 177 | 34239 |
|  | 4 | 640 | 1006 | 366 | 30348 | 194 | 783 | 161 | 29565 |
|  | 5 | 999999. |  | 36 | \% | - | 5 |  | \%\% |
|  | 6. | 999999 |  |  |  |  |  |  |  |
| Rando99 (28) | 1 | 438 | 568 | 130 | 737 | 112 | 76 | 53 | 661 |
|  | 2 | 442 | 577 | 135 | 776 | 113 | 76 | 53 | 700 |
|  | 3 | 661 | 875 | 214 | 3426 | 252 | 109 | 73 | 3317 |
|  | 4 | 687 | 10007 | 320 | 5535 | 358 | 123 | 88 | 5412 |
|  | 5 | 1045 | 1498 | 453 | 13902 | 359 | 188 | 95 | 13714 |
|  | 6 | 1053 | 1504 | 451 | 13998 | 365 | 188 | 95 | 13810 |
| Rand100 <br> (8) | 11. | 182 | 189 | 7 | 27 | 12 | 11 | 11 | 16 |
|  | 2 | 186 | 193 | 7 | 21 | 11 | 11 | 11 | 10 |
|  | 3 | 186 | 193. | - 7 | - 27 | 12 | 11 | 11 | 16. |
|  | 4 | 190 | 197 | 7 | 21 | 11 | 11 | 11 | 10 \% |
|  | 5 | 198 | 217 | 19 | 592 | 26 | - 13 | 13 | 579 |
|  | 6, | 194 | - 212 | 18 | 460 | 25 | 13 | 13. | $447 \times$ |
| Rand101 <br> (2) | 1 | 19 | 19 | 0 | 7 | 7 | 7 | 7 | 0 |
|  | 2 | 24 | 24 | 0 | 7 | 7 | 7 | 7 | 0 |
|  | 3 | 19 | 19 | 0 | 7 | 7 | 7 | 7 | 0 |
|  | 4 | 24 | 24 | 0 | 7 | 7 | 7 | 7 | 0 |
|  | 5 | 18 | 18 | 0 | 12 | 8 | 12 | 8 | 0 |
|  | 6 | 18 | 18 | 0 | 12 | 8 | 12 | 8 | 0 |
| Rand102 <br> (1) | 1 | 16 | 16 \% | 0 \% | 2 | 2 \% | + 2 | 2 | 0 O |
|  | 2 | 22 | 22. | 0 | 2 | 2 | 2 | 2 | 0 |
|  | 3 | 16 | 16 | 0 | 2 | 2 | 2 | 2 | 0 |
|  | , 4 , | 422 | , 22 | 0 | 2 , | +2 | 2 | 2 | 0 |
|  | 5 | 16 | , 16 | 0 | 2 | - 2 | , 2 | 2 | 0 |
|  | 6 | 16 | 16 . | $10^{0}$ | 2 \% | - 2 , | - 2 23 | 2 | 0 0 |
| Rand103 <br> (13) | 1 | 144 | 208 | 64 | 185 | 50 | 40 | 37 | 145 |
|  | 2 | 150 | 206 | 56 | 124 | 48 | 39 | 36 | 85 |


|  | 3 | 121 | 160 | 39 | 129 | 31 | 32 | 29 | 99 |
| :---: | :---: | :---: | :---: | :---: | :---: | :---: | :---: | :---: | :---: |
|  | 4 | 128 | 167 | 39 | 94 | 33 | 32 | 29 | 62 |
|  | 5 | 134 | 200 | 66 | 1239 | 50 | 89 | 43 | 1150 |
|  | 6 | 132 | 186 | 54 | 778 | 49 | 88 | 42 | 690 |
| Rand104 <br> (9) | 1 | 226 | 244 | 18 | 171 | 24 | 16 | 16 | 155 |
|  | 2 | 231 | 248 | 17 | 113 | 22 | 15 | 15 | 98 |
|  | 3 | 189 | 204 | 15 | 110 | 17 | 14 | 14 | 96 |
|  | 4 | 181 | 203 | 22 | 136 | 22 | 16 | 16 | 120 |
|  | 5 | 234 | 252 | 18. | 307 | 29 | 16 | 16 | 291 |
|  | 6 | 234 | 252 | 18 | 307 | 29 | -16 | 16 | 291 |
| Rand105 (96) | 1 | 167 | 268 | 101 | 1005 | 81 | 163 | 65 | 842 |
|  | 2 | 179 | 278 | 99 | 1001 | 80 | 162 | 64 | 839 |
|  | 3 | 176 | 266 | 90 | 953 | 60 | 146 | 54 | 807 |
|  | 4 | 210 | 319 | 109 | 1144 | 71 | 181 | 69 | 963 |
|  | 5 | 183 | 301 | 118 | 15642 | 71 | 423 | 63 | 15219 |
|  | 6 | 182 | 299 | 117 | 15642 | 71 | 423 | 63 | 15219 |
| Rand106 <br> (8) | 1 | 453 | 459 | 6 | 45 | 18 | 20 | 16. | 25 |
|  | 2 | 460 | 465 | 5 | 41 , | 17 | 18 | 15 | 23 |
|  | 3 | 432 | 486 | 54 | 3019 | 77 | 33 | 27 | 2985 |
|  | 4 | 469 | 516 | 47 | 3492 | 78 | 34 | 28 | 3458 |
|  | 5 | 577 | 608 | 31 | 10129 | 76 | 36 | 21 | 100093 |
|  | 6 | 577 | 608 | 31 | 10129 | 76 | 36 | 21 | 100093 |
| Rand107 <br> (5) | 1 | 17 | 17 | 0 | 5 | 5 | 5 | 5 | 0 |
|  | 2 | 23 | 23 | 0 | 5 | 5 | 5 | 5 | 0 |
|  | 3 | 17 | 17 | 0 | 5 | 5 | 5 | 5 | 0 |
|  | 4 | 23 | 23 | 0 | 5 | 5 | 5 | 5 | 0 |
|  | 5 | 18 | 18 | 0 | 11 | 6 | 6 | 6 | 5 |
|  | 6 | 18 | 18 | 0 | 11 | 6 | 6 | 6 | 5 |
| Rand108 <br> (35) | 1 | 346 | 385 | 39 | 162 | 81 | 124 | 79 | 38 |
|  | 2 | 348 | 387 | 39 | 162 | 81 | 124 | 79 | 38 |
|  | 3 | 457 | 588 | 131 | 1907 | 138 | 172 | 106 | 1735 |
|  | 4 | 423 | 530 | 107 | 1214 | 118 | 163 | 99 | 1051 |
|  | 5 | 540 | 788 | 248 | 8467 | 204 | - 212 | 135 | 8255 |
|  | 6 | 543 | 791 | 248 | 8605 | 206 | 212 | 133 | 8393. |
| $\begin{gathered} \text { Rand109 } \\ \text { (203) } \end{gathered}$ | 1 | 538 | 899 | 361 | 20884 | 215 | 486 | 145 | 20398 |
|  | 2 | 554 | 886 | 332 | 9197 | 209 | 473 | 140 | 8724 |
|  | 3 | 404 | 629 | 225 | 27508 | 140 | 439 | 108 | 27069 |
|  | 4 | 403 | 596 | 193 | 14208 | 123 | 437 | 106 | 13771 |
|  | 5 | 999999 |  |  |  |  |  |  |  |
|  | 6 | 999999 |  |  |  |  |  |  |  |
| Rand1 10 <br> (8) | 1 | 428 | 440 | 12 | 54 | 18 | 14 | 14 | 40 |
|  | 2 | 444 | 456 | 12 | 44 | 17 | - 14 | 14 | - 30 |
|  | 3 | 441. | 458 | 17 | 39 | 18 | 14 | 14 | 25 |
|  | 4 | 324 . | 340 | 16 | 113 | 20 | 22 | 16 | 2. 91 |
|  | 5 | 538 | 568 | 30 | 771 | 50 | $\bigcirc 13$ | 13 | . 758 |
|  | 6 | 536 | 566 | 30 | 683 | 48 | \% 13 | 13 | \% 670 |
| Rand 111 <br> (22) | 1 | 309 | 385 | 76 | 261 | 62 | 74 | 59 | 187 |
|  | 2 | 283 | 345 | 62 | 217 | 60 | 71 | 56 | 146 |
|  | 3 | 321 | 395 | 74 | 327 | 68 | 73 | 61 | 254 |
|  | 4 | 274 | 335 | 61 | 204 | 63 | 82 | 58 | 122 |
|  | 5 | 410 | 566 | 156 | 2011 | 104 | 84 | 71 | 1927 |
|  | 6 | 402 | 537 | 135 | 1139 | 91 | 80 | 69 | 1059 |


| Rand112 <br> (2) | 1 | 11 | 11 | 0 | 5 | 5 | 5 | 5 | 0 |
| :---: | :---: | :---: | :---: | :---: | :---: | :---: | :---: | :---: | :---: |
|  | 2 | 16 | 16 | 0 | 5 | 5 | 5 | 5 | 0 |
|  | 3 | 11. | 11 | 0 | 5 | 5 | 5 | 5 | 0 |
|  | 4 | 16 | 16 | 0 | 5 | 5 | 5 | 5 | 0 |
|  | 5 | 15 | 15 | 0 | 5 | 5 | 5 | 5 | 0 |
|  | 6 | 14 | 14 | 0 | 5 | 5 | 5 | 5 | 0 |
| Rand 113 <br> (2) | 1 | 74 | 74 | 0 | 6 | 6 | 6 | 6 | 0 |
|  | 2 | 70 | 70 | 0 | 6 | 6 | 6 | 6 | 0 |
|  | 3 | 74 | 74 | 0 | 6 | 6 | 6 | 6 | 0 |
|  | 4 | 64 | 64 | 0 | 6 | 6 | 6 | 6 | 0 |
|  | 5 | 73 | 73 | 0 | 21 | 8 | 6 | 6 | 0 |
|  | 6 | 73 | 73 | 0 | 21 | 8 | 6 | 6 | 0 |
| Rand114 <br> (2) | 1 | 28 | 28 | 0 | 7 | 7 | 7 | 7 | 0 |
|  | 2 | 34 | 34 | 0 | \% 7 , | 7. | 7 | \% 7 | 0 |
|  | 3. | 28 | 28 | 0 | 37 | W.7. ${ }^{\text {a }}$ | 7 | 7 | 0 |
|  | 4 | 34 | 34 | 0 | 7 7 , | 4 | 7 | 7 | 0 |
|  | 5 | 26 | 26 | 0 | 12 | - 7 | 12 | 7 | 0 |
|  | 6 \% | 26 | 26 | 0 | 12 | 7 | 12 | 4 | 0 |
| Rand115 <br> (46) | 1 | 331 | 558 | 227 | 851 | 159 | 135 | 93 | 716 |
|  | 2 | 333 | 598 | 265 | 1046 | 162 | 135 | 95 | 911 |
|  | 3 | 250 | 330 | 80 | 1131 | 73 | 101 | 53 | 1030 |
|  | 4 | 343 | 569 | 226 | 4144 | 174 | 195 | 94 | 3949 |
|  | 5 | 254 | 393 | 139 | 3834 | 87 | 170 | 59 | 3664 |
|  | 6 | 258 | 109 | 151 | 4200 | 89 | 175 | 62 | 4025 |
| Rand116$(15)$ | 1 | 457 | 586 | 129 | 12364 | 89 | 76 | 65 | 12288 |
|  | - 2 | 427 | 529 | 102 | 1181 | 84 | 74 | 61 | 1107 |
|  | 3 | 394 | 515 | 121 | 937 | 80 | 73 | 61 | 864 |
|  | 4 | 621 | 997 | 376 | 1628 | 229 | 120 | 75 | 1508 |
|  | 5 | 464 | 615 | 151 | 7654 | 125 | 85 | 83 | 7569 |
|  | 6 | 673 | 1041 | 368 | 8199 | 252 | 119 | 75 | 8080 |
| $\begin{gathered} \text { Rand117 } \\ \text { (11) } \end{gathered}$ | 1 | 70 | 86 | 16 | 40 | 24 | 29 | 24 | 11 |
|  | 2 | 77 | 93 | 16 | 40 | 24 | 29 | 24 | 11 |
|  | 3 | 74 | 80 | 6 | 32 | 29 | 30 | 29 | 2 |
|  | 4 | 81 | 87 | 6 | 32 | 29 | 30 | 29 | 2 |
|  | 5 | 78 | 89 | 11 | 153 | 3 | 43 | 26 | 110 |
|  | 6 | 78 | 89 | 11 | 167 | 31 | 42 | 25 | 125 |
| Rand118 <br> (52) | 1 , ${ }^{\text {c }}$ | 272. | 503 | 231 | 2627 | - ${ }^{3}$ | 149 | 98 | 2478 |
|  | 2 | 277 | 494 | 217 | 1547 | \% 146 | 145 | $\bigcirc 95$ | 1402 . |
|  | 3 | 209 | 366 | 157 | 1755. | - 103 | 137 | 91 | -1618 |
|  | 4 | 223 | 358 | 135 | , 703 | Y 103 | 147 | 92 | 556 |
|  | - 5 | 310 | 503 | 193 | 76818 | - 98 | 153 | 91 | 76665 |
|  | 6 | 301 | 486 | 185 | 33010 | $\bigcirc 94$ | 153. | 90 | 32857 |
| $\begin{gathered} \text { Rand119 } \\ (84) \end{gathered}$ | 1 | 141 | 169 | 28 | 309 | 43 | 169 | 43 | 140 |
|  | 2 | 147 | 175 | 28 | 309 | 43 | 169 | 43 | 140 |
|  | 3 | 154 | 178 | 24 | 381 | 30 | 161 | 30 | 220 |
|  | 4 | 165 | 208 | 43 | 418 | 55 | 180 | 55 | 238 |
|  | 5 | 161 | 216 | 55 | 7375 | 42 | 518 | 36 | 6857 |
|  | 6 | 156 | 209 | 53 | 6102 | 39 | 488 | 34 | 5614 |
| Rand120 | 1 | 463 | 1102: | 639 | 4496 | 349 | 201 | 146 | 4295 |
|  | 2 | 464 | 1103 | 639 | 3392, | 343 | 201 | 146 | 3191 |
|  | 3 | 321 | 618 | 297 | 12152 | $\bigcirc 152$ | 292 | \% 101 | 11860 |
|  | $\therefore 4$ | 323 | 580 | 257 | 9856 | 148 | 286 | 97. | 9570 |



|  | 3 | 64 | 64 | 0 | 4 | 4 | 4 | 4 | 0 |
| :---: | :---: | :---: | :---: | :---: | :---: | :---: | :---: | :---: | :---: |
|  | 4 | 79 | 79 | 0 | 4 | 4 | 4 | 4 | 0 |
|  | 5 | 81 | 81 | 0 | 9 | 5 | 5 | 5 | 4 |
|  | 6 | 77 | 77 | 0 | 4 | 4 | 4 | 4 | 0 |
| Rand 130 <br> (5) | 1 | 189 | 189 | 0 | 5 | 5 | 5 | 5 | 0 |
|  | 2 | 196 | 196 | 0 | - 5 | 5 | 5 | 5 | 0 |
|  | 3 | 161 | 161 | 0 \% | 5 | 5 | 5 | 5 | 0 |
|  | 4 | 169 | 169 | 0 | 20 | 8 | 5 | 5 | 15 |
|  | 5 | 142 | 156 | 14 | 305 | 30 | 7 | 7 | 298 |
|  | 6 | 143 | 157 | 14 | 385 | 29 | 7 | 7 | 278 |
| Rand131 <br> (2) | 1 | 21 | 22 | 1 | 8 | 8 | 8 | 8 | 0 |
|  | 2 | 22 | 23 | 1 | 8 | 8 | 8 | 8 | 0 |
|  | 3 | 21 | 22 | 1 | 8 | 8 | 8 | 8 | 0 |
|  | 4 | 22 | 23 | 1 | 8 | 8 | 8 | 8 | 0 |
|  | 5 | 23 | 25 | 2 | 9 | 9 | 9 | 9 | 0 |
|  | 6 | 23 | 25 | 2 | 9 | 9 | 9 | 9 | 0 |
| Rand132 <br> (67) | 1 | 539 | 1097 | 558 | 6953. | 325 | 251 | 154 | 6702 |
|  | 2 | 551 | 1120 | 569 | 5128 | 321 | 251 | 156 | 4877 |
|  | 3. | 735 | 1766 | 1031 | $\checkmark 17069$ | 473 | 326 | 219 | 16743 |
|  | 4. | 739 | 1698 | 959 | 13922 | 462 | 324 | 217 | 13598 |
|  | 5 | 853 | 2151 | 1298 | 88000 | 578 | 437 | 240 | 87563 |
|  | 6 | 863 | 2144 | 1281 | 91309 | 584 | 439 | 242 | 90870 |
| Rand133 <br> (4) | 1 | 16 | 16 | 0 | 5 | 5 | 5 | 5 | 0 |
|  | 2 | 19 | 19 | 0 | 5 | 5 | 5 | 5 | 0 |
|  | 3 | 16 | 16 | 0 | 5 | 5 | 5 | 5 | 0 |
|  | 4 | 21 | 21 | 0 | 6 | 6 | 6 | 6 | 0 |
|  | 5 | 20 | 20 | 0 | 12 | 6 | 8 | 5 | 4 |
|  | 6 | 20 | 20 | 0 | 12 | 6 | 8 | 5 | 4 |
| Rand 134 <br> (60) | 1 \% | 502 | 642 | 140 | 682 | 104 | 151 | 72 | 531 |
|  | 2 | 520 | 649 | 129 | 620 | 102 | 151 | 72 | 469 |
|  | 3 , | 620 | 872 | 252 | 6188 | 229 | 185 | 79 | 6003 |
|  | 4 | 630 | 849 | 219 | 4358 | 205 | 179 | 77 | 4179 |
|  | 5 | 999999 |  |  | \% |  | , | - ${ }^{\text {a }}$ | \% |
|  | 6 | 999999 |  |  | - | S |  | 2, 6 | \%r |
| Rand 135 <br> (24) | 1 | 258 | 314 | 56 | 450 | 52 | 94 | 47 | 356 |
|  | 2 | 264 | 319 | 55 | 382 | 52 | 94 | 47 | 288 |
|  | 3 | 348 | 502 | 154 | 1539 | 112 | 136 | 63 | 1403 |
|  | 4 | 333 | 476 | 143 | 3078 | 111 | 139 | 61 | 2939 |
|  | 5 | 757 | 1120 | 363 | 15347 | 303 | 177 | 96 | 15170 |
|  | 6 | 757 | 1120 | 363 | 15347 | 303 | 177 | 96 | 15170 |
| Rand 136 <br> (1) | 1 | 9 | 9 | 0 , | 4 | 4 | 4 | 4 | 0 |
|  | 2 | 11 | 11 | 0 | 4 | 4 | 4 | 4 | 0 |
|  | 3 | 9 | 9 | 0 | \% 4 | 4 | 4 | 4 | 0 |
|  | 4 | 11 | 11 | 0 | 4 | 4 | 4 | 4 | \% |
|  | 5 | 10 | 10 | 0 , | $\cdots$ | 4 \% | 4 | - 4. | - 0 |
|  | 6 | +11 | 11 | 0 . | - 4 | 4 | 4 | 4. | 0 |
| Rand137 <br> (15) | 1 | 97 | 131 | 34 | 157 | 34 | 85 | 36 | 72 |
|  | 2 | 104 | 136 | 32 | 139 | 32 | 85 | 36 | 54 |
|  | 3 | 97 | 131 | 34 | 157 | 34 | 85 | 36 | 72 |
|  | 4 | 104 | 136 | 32 | 139 | 32 | 85 | 36 | 54 |
|  | 5 | 109 | 152 | 43 | 1393 | 35 | 154 | 30 | 1239 |
|  | 6 | 105 | 148 | 43 | 916 | 33 | 154 | 30 | 762 |


| Rand138 <br> (2) | 1 | 81 | 81 | 0 | 2 | 2 | 2 | 2 | 0 |
| :---: | :---: | :---: | :---: | :---: | :---: | :---: | :---: | :---: | :---: |
|  | 2 | 90 | 90 | 0 | 2 | 2 | 2 | 2 | 0 |
|  | 3 | 82 | 82 | 0 | 2 | 2 | 2 | 2 | 0 |
|  | 4 | 85 | 85 | 0 | 5 | - 3 | 3 | $\therefore 3$ | 2 |
|  | 5 | 91 | 91 | 0 | 2 | \% 2 | 2 | 2 | 0 |
|  | 6 | 91 | 91 | 0 | 2 | 2 | 2 | 2 | 0 |
| Rand139 (49) | 1 | 229 | 318 | 89 | 734 | 88 | 142 | 82 | 592 |
|  | 2 | 229 | 305 | 76 | 478 | 83 | 138 | 78 | 340 |
|  | 3 | 239 | 293 | 54 | 1657 | 52 | 163 | 47 | 1494 |
|  | 4 | 242 | 293 | 51 | 1117 | 52 | 158 | 45 | 959 |
|  | 5 | 274 | 394 | 120 | 6640 | 75 | 158 | 66 | 6482 |
|  | 6 | 272 | 392 | 120 | 3477 | 73 | 153 | 65 | 3324 |
| Rand140 <br> (5) | 1 | 20 | 20 | 0 | 5 | 5 | 5 | 5 | 0 , |
|  | 2 | 24 | 24 | 0 | 5 | 5 | 5 | 5 | 0 |
|  | 3 | 20 | -20 | 0 | 5 | 5 | 5 | 5 | 0 |
|  | 4 | 24 | 24 | 0 | 5 | 5 | 5 | 5 | 0 |
|  | 5 | 21. | 21 | 0 | 5 | 5 | 5 | - 5 | 0 |
|  | 6 | 21 | 21 | 0 | 5 | 5 | 5 | 5 | 0 |
| Rand141 <br> (8) | 1 | 428 | 440 | 12 | 54 | 18 | 14 | 14 | 40 |
|  | 2 | 444 | 456 | 12 | 44 | 17 | 14 | 14 | 30 |
|  | 3 | 441 | 458 | 17 | 39 | 18 | 14 | 14 | 25 |
|  | 4 | 324 | 340 | 16 | 113 | 20 | 22 | 16 | 91 |
|  | 5 | 538 | 568 | 30 | 771 | 50 | 13 | 13 | 758 |
|  | 6 | 538 | 568 | 30 | 771 | 50 | 13 | 13 | 758 |
| Rand142 | 1 | 2394 | 6247 | 3853 | 79169 | 1744 | 1220 | 524 | 77949 |
|  | 2 | 2226 | 5639 | 3413 | 55680 | 1572 | 1151 | 487 | 54529 |
|  | 3 . | 2116 | 6307 | 4191 | 137991 | 1601 | 1574 | 642 | 136417 |
|  | 4 | 1978 | 5798 | 3820 | 65900 | 1464 | 1487 | 639 | 64413 |
|  | 5 | 2383 | 6798 | 4415 | 507669 | 1636 | 1673 | 625 | 505996 |
|  | 6 | 2380 | 6783 | 4403 | 507544 | 1630 | 1673 | 625 | 505871 |
| Rand143 <br> (8) | 1 | 178 | 215 | 37 | 253 | 27 | 22 | 22 | 231 |
|  | 2 | 181 | 218 | 37 | 253 | 27 | 22 | 22 | 231 |
|  | 3 | 185 | 268 | 83 | 284 | 52 | 30 | 30 | 254 |
|  | 4 | 185 | 268 | 83 | 284 | 52 | 30 | 30 | 254 |
|  | 5 | 149 | 227 | 78 | 1830 | 51 | 26 | 26 | 1804 |
|  | 6 | 149 | 227 | 78 | 1706 | 50 | 26 | 26 | 1680 |
| Rand144 <br> (41) | 1 | 456 | 723 | 267 | 6836 | 146 | 113 | 78 | 6723 \% |
|  | 2 | 456 | 731 | 275 | 5197 | 149 | 113 | -78 | 5084 |
|  | 3 , | 700 | 869 | 169 | 10152 | 163 | 93 | 75 | 10059 |
|  | 4 | 766 | 1478 | 712 | 10018 | , 341 | 185 | 113 | 9833. |
|  | 5 | 999999 | W, 积新 |  | + | , | \% 18 | , | , ${ }^{2}$ |
|  | 6 | 999999 | \% |  | 45 | , | \% ${ }^{4} \times$ | 3 | \% |
| Rand 145$(47)$ | 1 | 111 | 149 | 38 | 526 | 47 | 98 | 43 | 428 |
|  | 2 | 128 | 153 | 25 | 499 | 46 | 95 | 42 | 404 |
|  | 3 | 117 | 150 | 33 | 330 | 38 | 72 | 38 | 258 |
|  | 4 | 134 | 154 | 20 | 303 | 37 | 69 | 37 | 234 |
|  | 5 | 117 | 150 | 33 | 330 | 38 | 72 | 38 | 258 |
|  | 6 | 138 | 196 | 58 | 39677 | 36 | 103 | 36 | 39574 |
| Rand146 | 1 | 97 | 131 | 34 | 157 | - 34 | 85 | \% 36 | 72 |
|  | 2 | 104 | 136 | 32 | 139 | 32 | 85 | 36 | $\therefore 54$ |
|  | 3 | 97: | 131 | -34 | 157 | 34 | 85 | 36 | 72, |
|  | 4. 4 | 104 | 136 | 32 | 139 | , 32 | 85 | , $\times 36$ | 5 54 |


|  | $5 \cdots$ | 109 | 152 | 43 | 1393 | 35 | 15 | 30 | 1378 |
| :---: | :---: | :---: | :---: | :---: | :---: | :---: | :---: | :---: | :---: |
|  | 6 | 105 | 148 | 43 | 916 | 33 | 154 | 30 | 762 |
| $\begin{gathered} \text { Rand147 } \\ (30) \end{gathered}$ | 1 | 535 | 878 | 343 | 3504 | 198 | 104 | 85 | 3400 |
|  | 2 | 548 | 890 | 342 | 2548 | 198 | 104 | 85 | 2444 |
|  | 3 | 1896 | 3272 | 1376 | 195933 | 997 | 192 | 165 | 195741 |
|  | 4 | 1091 | 2020 | 929 | 59599 | 470 | 166 | 140 | 59433 |
|  | 5 | 2123 | 3327 | 1204 | 396283 | 1046 | 183 | 154 | 396100 |
|  | 6 | 2121 | 3399 | 1278 | 214895 | 1046 | 187 | 158 | 214708 |
| Rand 148 <br> (8) | 1 | 103 | 128 | 25 | 40 | 21 | 15 | 15 | 25 |
|  | 2 | 112 | 137 | 25 | 40 | 21 | 15 | 15 | 25 |
|  | 3 | 87 | 105 | 18 | 35 | 20 | 15 | 15 | 20 |
|  | 4 | 97 | 122 | 25 | 40 | 21 | 15 | 15 | 25 |
|  | 5 | 120 | 166 | 46 | 4337 | 37 | \% 32 | 20 | $\therefore 4302$ |
|  | 6 | 120 | 166 | 46 | 4337 | 37 | - 32 | 20 | 4302 |
| Rand 149 <br> (18) | 1 | 251 | 277 | 26 | 144 | 24 | 28 | 22 | 116 |
|  | 2 | 278 | 304 | 26 | 144 | 24 | 28 | 22 | 116 |
|  | 3 | 293 | 317 | 24 | 87 | 21 | 27 | 21 | 60 |
|  | 4 | 313 | 337 | 24 | 175 | 22 | 27 | 22 | 148 |
|  | 5 | 999999 |  |  |  |  |  |  |  |
|  | 6 | 999999 |  |  |  |  |  |  |  |
| $\begin{aligned} & \text { Rand } 150 \\ & (114) \end{aligned}$ | 1 | 1096 | 2615 | 1519 | 84053 | 667. | 457 | 223 | 83596 |
|  | 2 | 1101 | 2541 | 1440 | 48700 | 669 | 460 | 220 | 48240 |
|  | 3 | 1655 | 4142 | 2487 | 218284 | 1165 | 693 | 310 | 217591 |
|  | 4 | 1495 | 3563 | 2068 | 153304 | 1015 | 617 | 277 | 152687 |
|  | 5 | 999999 | \% | \% |  | - | S |  | 3-260 |
|  | 6 | 999999 | \%... | \% |  |  | M, \% | - | 4\% |
| $\begin{gathered} \text { Rand151 } \\ (36) \end{gathered}$ | 1 | 108 | 135 | 27 | 253 | 29 | 59 | 25 | 194 |
|  | 2 | 121 | 148 | 27 | 298 | 30 | 59 | 25 | 239 |
|  | 3 | 115 | 133 | 18 | 148 | 24 | 58 | 24 | 90 |
|  | 4 | 133 | 157 | 24 | 194 | 32 | 65 | 31 | 129 |
|  | 5 | 106 | 142 | 36 | 5326 | 29 | 157 | 27 | 5169 |
|  | 6 | 106 | 142 | 36 | 5326 | 29 | 157 | 27 | 5169 |
| Rand152 <br> (1) | 1 | 4 | 4 | 0 | 1 | 1 | 1 | - 1 | 0 |
|  | 2 | 4 | 4 \% | 0 | 2, 1 | 1 | 1 | - 1 | Q 0 |
|  | 3 | 4 | 4 | 0 | 1 | 1 | 1 | 1 m | 0 , |
|  | -4.4. | 4 | 4 4 | 0 , | \% 1 | 1 | 1 | - 1 , 1 , | 0 0, |
|  | \% 5 | 4 | 4 | 0 | 1 | 1 | 1 . | 12 | 0 0 |
|  | 6 | 4 | \% 4 | 0 | 1. | 419 | , 1 | , 10, | \% 0 , |
| Rand153 <br> (3) | 1 | 159 | 159 | 0 | 8 | 8 | 8 | 8 | 0 |
|  | 2 | 163 | 163 | 0 | 8 | 8 | 8 | 8 | 0 |
|  | 3 | 159 | 169 | 0 | 8 | 8 | 8 | 8 | 0 |
|  | 4 | 165 | 171 | 6 | 16 | 11 | 11 | 11 | 5 |
|  | 5 | 188 | 194 | 6 | 47 | 16 | 17 | 11 | 30 |
|  | 6 | 189 | 189 | 0 | 40 | 14 | 16 | 10 | 24 |
| Rand154 <br> (1) | 1 | 119 | 119 | 0 | 1 | 1 | 1. | 1 | - 0 |
|  | 2 . | 116 | 116 | 0 \% | - 1 | 1 | 1 | , 1, | 20 |
|  | 3 | 100 | 100 | 0 | 1 | 1 | 1 . | 1 | 0 |
|  | 4. | 101. |  | 0 | 1 | 1 | W. 1 | - 1 | 0 |
|  | 5 | 121 | 121 | 0 | 1 | 1 | 1 | 1 | . 0 |
|  | 6 | 124 | - 124 | 0 0 | . 114 | 1, | - 1 | - 1 1- | $00^{0}$ |
| Rand155 <br> $(52)$ | 1 | 291 | 597 | 306 | 960 | 182 | 171 | 93 | 789 |
|  | 2 | 291 | 574 | 283 | 905 | 179 | 168 | 90 | 737 |


|  | 3 | 305 | 583 | 278 | 3901 | 146 | 275 | 111 | 3626 |
| :---: | :---: | :---: | :---: | :---: | :---: | :---: | :---: | :---: | :---: |
|  | 4 | 297 | 515 | 218 | 2240 | 136 | 263 | 103 | 1977 |
|  | 5 | 303 | 521 | 218 | 5802 | 137 | 347 | 96 | 5455 |
|  | 6 | 312 | 512 | 200 | 4438 | 146 | 344 | 93 | 4094 |
| $\begin{aligned} & \text { Randl156 } \\ & (20) \end{aligned}$ | 1 | 188 | 188 | 0 | 47. | 29 | 47 | 29 | 0 |
|  | 2 | 197 | 197 | 0 | 44 | 28 | 44 | 28 | 0 |
|  | 3 | 149 | 149 | 0 | 43 | 23 | 43 | 23 | 0 |
|  | 4 | 124 | 129 | 5 | 49 | 31 | 49 | 31 | 0 |
|  | 5 | 111 | 126 | 15 | 441 | 29 | 147 | 29 | 294 |
|  | 6 | 109 | 124 | 15 | 362 | 27 | 143 | 27 | 219 |
| Rand158 <br> (9) | 1 | 8170 | 8192 | 22 | 39 | 27 | 21 | 21 | 18 |
|  | 2 | 7474 | 7496 | 22 | 39 | 27 | 21 | 21 | 18 |
|  | 3 | 6399 | 6407 | 8 | 30 | 18 | 18 | 18 | 12 |
|  | 4 | 6611 | 6619 | 8 | 60 | 19 | 18 | 18 | 42 |
|  | 5 | 2639 | 2662 | 23 | 1921 | 36 | 53 | 23 | 1868 |
|  | 6 | 2611 | 2633 | 22 | 1410 | 34 | 52 | 22 | 1358 |
| Lisaba9 (85) | 1 | 269 | 479 | 210 | 4067 | 136 | 204 | 58 | 3863 |
|  | 2 | 283 | 493 | 210 | 4067 | 136 | 204 | 58 | 3863 |
|  | 3 | 196 | 283 | 87 | 3571 | 56 | 187 | 61 | 3384\% |
|  | 4 | 211 | 298 | 87 | 3515 | 56 | $\cdots 185$ | 61 | 3330 |
|  | 5 | 248 | 357 | 109 | 227961 | 63 | 199 | 49 | 227762 |
|  | 6 | 248 | 357 | 109 | 227961 | 63 | 199 | 49 | 227762 |
| $\begin{gathered} \text { Lisab30 } \\ (17) \end{gathered}$ | 1 | 254 | 305 | 51 | 293 | 48 | 59 | 39 | 234 |
|  | 2 | 256 | 291 | 35 | 137 | 40 | 53 | 35 | 84 |
|  | 3 | 242 | 268 | 26 | 249 | 34 | 51 | 33 | 198 |
|  | 4 | 248 | 270 | 22 | 129 | 33 | 45 | 29 | 84 |
|  | 5 | 289 | 381 | 92 | 12822 | 56 | 52 | 43 | 12770 |
|  | 6 | 290 | 383 | 93 | 12978 | 56 | 53 | 43 | 12925 |
| Lisab60 <br> (19) | 1 | 60 | 79 | 19. | 123 | 26 | 51 | 21. | - 72 |
|  | 2 | 60 | 76 | 16 | 66 | 24 | \% 48 | 20 | - 18 |
|  | 43 | 62 | 85 | 23 | 134 | 26 | 60 | 23 | 74 |
|  | 4 | 69 . | 82 | 13 | 73 | 25 | 53 | 21 | 20 |
|  | + 5 | . 77 | 101 | 24 | 188 | 26 | 65 | 23 | 123 |
|  | 6 | 79\% | 103 | 24 | 148 | 26 | 64 | 23 | 84 |
| $\begin{gathered} \text { Lisab59 } \\ (3096) \end{gathered}$ | 1 | 231 | 378 | 147 | 77222 | 133 | 4670 | 117 | 72552 |
|  | 2 | 264 | 411 | 147 | 77222 | 133 | 4670 | 117 | 72552 |
|  | 3 | 174 | 220 | 46 | 43242 | 49 | 5381 | 49 | 37861 |
|  | 4 | 207 | 253 | 46 | 43242 | 49 | 5381 | 49 | 37861 |
|  | 5 | 999999 |  |  |  |  |  |  |  |
|  | 6 | 999999 |  |  |  |  |  |  |  |
| Lisab31(164) | 1\% | 613 | 899 | 286 | 5871 | 213 | 600 | 176 | 5271 |
|  | 2 | 621 | 914 | 293 | 4534 , | 217 | - 607 | -178 | 3 3927 |
|  | 2 3 | 1035 | 1495 | 460 | 41809 | 274 | 767 | 163 | 41042 |
|  | 4 | - 977 | 1651 | 674 | 44935 , | 392 | 747 | 183 | 44188 |
|  | 5 | 1246 | 1837 | 591 | 837509 | 494 | 1062 | 230 | 836447 |
|  | 6 | 1203 | 1770 | 567 | 607491 | 477 | 1035 | 226 | 606456 |
| Lisaba4 (827) | 1 | 645 | 1091 | 446 | 12368 | 346 | 2302 | 333 | 10066 |
|  | 2 | 643 | 1009 | 366 | 9959 | 335 | 2283 | 315 | 7676 |
|  | 3 | 799 | 1326 | 527 | 58847 | 257 | 2645 | 303 | 56202 |
|  | 4 | 542 | 1012 | 470 | 16818 | 258 | 2639 | 300 | 14179 |
|  | 5 | 999999 |  |  |  |  |  |  |  |
|  | 6 | 999999 |  |  |  |  |  |  |  |


| $\begin{gathered} \text { Lisab57 } \\ (170) \end{gathered}$ | 1 | 179 | 280 | 101 | 1291 | 100 | 417 \% | 81 | 874 |
| :---: | :---: | :---: | :---: | :---: | :---: | :---: | :---: | :---: | :---: |
|  | 2 | 185 | 284 | 99 | 1245 | 100 | 404 | 81 | 841 |
|  | 3 | 251 | 515 | 264 | 2869 | 164 | 511\% | 134 | 2358 |
|  | 4 | 259 | 491 | 232 | 2125 | 160 | 443 | 121. | 1682 |
|  | 5 | 228 | 475 | 247 | 3475 | 126 | 473 | 93 | 3002 |
|  | 6 | 226 | 473 | 247 | 2921 | 125 | 451 | 94 | 2470 |
| $\begin{gathered} \text { Lisab28 } \\ (66) \end{gathered}$ | 1 | 64 | 76 | 12 | 201 | 30 | 116 | 28 | 85 |
|  | 2 | 79 | 91 | 12 | 201 | 30 | 116 | 28 | 85 |
|  | 3 | 69 | 84 | 15 | 156 | 22 | 90 | 22 | 66 |
|  | 4 | 84 | 99 | 15 | 156 | 22 | 90 | 22 | 66 |
|  | 5 | 64 | 84 | 20 | 646 | 22 | 350 | 22 | 296 |
|  | 6 | 64 | 84 | 20 | 646 | 22 | 350 | 22 | 296 |
| $\begin{gathered} \text { Lisab10 } \\ (940) \end{gathered}$ | 1 | 743 | 1306 | 563 | 22074 | 414 | 1921 | 335 | 20153 |
|  | 2 | 740 | 1192 | 452 | 9593 | 399 | 1983. | 334 | 7610 |
|  | 3 | 702 | 1049 | 347 | 30126 | 224 | 2912 | 231 | 27214 |
|  | 4 | 809 | 1268 | 459 | 34660 | 282 | 2994. | 287 | 31666 |
|  | 5 | 796 | 1305 | 509 | 103892 | 339 | 2625 | 280 | 101267 |
|  | 6 | 797 | 1303 | 506 | 86036 | 333 | 2622 | 279 | 83414 |
| $\begin{gathered} \hline \text { Lisab35 } \\ (136) \end{gathered}$ | 1 | 513 | 1176 | 663 | 26233 | 386 | 435 | 192 | 25798 |
|  | 2 | 507 | 1149 | 642 | 17901 | 372 | 419 | 183 | 17482 |
|  | 3 | 604 | 1534 | 930 | 42677 | 474 | 557 | 175 | 42120 |
|  | 4 | 606 | 1533 | 927 | 39592 | 468 | 548 | 174 | 39044 |
|  | 5 | 589 | 1562 | 973 | 241117 | 460 | 810 | 200 | 240307 |
|  | 6 | 579 | 1534 | 955 | 217713 | 450 | 808 | 198 | 216905 |
| Lisab51 <br> (11) | 1 | 70 | 115 | 45 | 163 | 31 | 29 | 29 | 134 |
|  | 2 | 77 | 119 | 42 | 103 | 30 | 28 | 28 | 75 |
|  | 3 | 77 | 125 | 48 | 182 | 29 | 27 | 27 | 155 |
|  | 4 | 85 | 126 | 41 | 119 | 28 | 26 | 26 | 93 |
|  | 5 | 62 | 101 | 39 | 279 | 26 | 26 | 26 | 253 |
|  | 6 | 62 | 102 | 40 | 249 | 26 | 26 | 26 | 223 |
| Lisab44 <br> (12) | 1 | 115 | 138 | 23 | 176 | 29 | 30 | 28 | 146 |
|  | 2 | 124 | 145 | 21 | 164 | 29 | 29 | 27 | 135 |
|  | 3 | 107 | 128 | 21 | 158 | 24 | 33 | 23 | 125 |
|  | 4 | 111 | 142 | 31 | 248 | 35 | 47 | 29 | 201 |
|  | 5 | 112 | 166 | 54 | 524 | 35 | 41 | 34 | 483 |
|  | 6 | 113 | 171 | 58 | 440 | 36 | 40 | 35 | 400 |
| Lisab25 (35) | 1 | 202 | 287 | 85 | 254 | 61 | 84 | 51 | 170 |
|  | 2 | 218 | 291 | 73 | 168. | 63 | -86 | 53 | - 82 |
|  | - 3 | 204 | 302 | 98 | + 199 | 64 \% | 69 | 54 | 130 |
|  | 4 | 220 | 316 | 96 | 165 | 69 | 68 | 56 | 97 |
|  | 5 \% | 264 | 455 | 191 | 3487 | 94. 9 | 171 | -75 | 3316 |
|  | 6 | 268 | 456 | 188 | 3487 | 98 | 168 | 75 | 3319 |
| Lisab54 <br> (14) | 1 | 65 | 92 | 27 | 99 | 23 | 28 | 21 | 71 |
|  | 2 | 72 | 93 | 21 | 57 | 22 | 27 | 21 | 30 |
|  | 3 | 74 | 97 | 23 | 50 | 22 | 26 | 22 | 24 |
|  | 4 | 77 | 92 | 15 | 40 | 20 | 24 | 20 | 16 |
|  | 5 | 68 | 121 | 53 | 177 | 31 | 37 | 30 | 140 |
|  | 6 | 69 | 116 | 47 | 124 | 32 | 39 | 31 | 85 |
| $\begin{aligned} & \text { Lisab53 } \\ & (15) \end{aligned}$ | 1 | 23 | 26. | 3. | 31. | 12. | , 27. | 縎, 12 |  |
|  | 2 | 26. | 26 | 0 | 25 | 11 \% | 25 | 11 | 0 |
|  | 3 | 23. | 26 | 3 | 31 | 10 | 27 . | -10 | \% 4 |
|  | - 4 , | 26 | 26 | 0 | 25 \% | - 9 9, | . 25 \% | \% 9 | 0 \% |


|  | 5 | 23 | 26 | 3 | 31 | 12 | 27 | 12 | 4 |
| :---: | :---: | :---: | :---: | :---: | :---: | :---: | :---: | :---: | :---: |
|  | 6 | 23 | 26 | 3 | 31 | 12 | 27 | 12 | 4 |
| $\begin{gathered} \text { Lisab52 } \\ (139) \end{gathered}$ | 1 | 815 | 1907 | 1092 | 6370 | 531 | 492 | 307 | 5878 |
|  | 2 | 800 | 1848 | 1048 | 4981 | 505 | 489 | 301 | 4492 |
|  | 3 | 997 | 2280 | 1283 | 30114 | 629 | 775 | 358 | 29339 |
|  | 4 | 842 | 1817 | 975 | 23475 | 513 | 765 | 354 | 22710 |
|  | 5 | 1103 | 2790 | 1687 | 121612 | 650 | 1020 | 344 | 120592 |
|  | 6 | 1049 | 2612 | 1563 | 105773 | 603 | 996 | 335 | 104777 |
| Lisab34(14) | 1 | 70 | 81 | 11 | 35 | 22 | 30 | 23 | 5 |
|  | 2 | 75 | 82 | 7 | 33 | 23 | 29 | 23 | 4 |
|  | 3 | 69 | 76 | 7 | 36 | 22 | 28 | 22 | 8 |
|  | 4 | 73 | 84 | 11 | 45 , | 25 | 32 | 24 | 13 |
|  | 5 | 77 | 94 | 17 | 83 | 20 | 38 | 21 | 45 |
|  | 6 | 78 | 95 | 17 | 83 | 20 | 38 | 21 | 45 |
| Lisab36 <br> (52) | 1 | 1083 | 1271 | 188 | 1641 | 135 | 202 | 91 | 1439 |
|  | 2 | 1064 | 1267 | 203 | 2820 | 144 | 204 | 93 | 2616 |
|  | 3 | 1063 | 1196 | 133 | 480 | 101 | 161 | 79 | 319 |
|  | 4 | 1019 | 1204 | 185 | 3413 | 125 | 177 | 87 | 3236 |
|  | 5 | 1515 | 2007 | 492 | 27753 | 362 | 240 | 127 | 27513 |
|  | 6 | 1518 | 2051 | 533 | 27513 | 366 | 238 | 124 | 27275 |
| Lisab42 <br> (10) | 1 | 64 | 64 | 0 | 23 | 19 | 23 | 19 | 0 |
|  | 2 | 76 | 76 | 0 | 23 | 19 | 23 | 19 | 0 |
|  | 3 | 61 | 61 | 0 | ${ }^{17}$ | 17 | \% 17 | 17 | 0 |
|  | 4 | 73 | 73 | 0 | 17 , | 17 | 17 | 17 | 0 |
|  | 5 | 65 | 87 | 22 | 223 | 17 | 33 | 17 | 190 |
|  | -6 | 65 | 87 | 22 | 223. | 17 | 33 | 17 | 190 |

## Appendix IV

Remainder of Eleven Characteristics Used to Describe Fault Trees
Program Used To Produce Results - prodchar.f

| Tree | Number of Outputs From Top | Number of Levels | Total no. of Events | Max. Gates in Any Level | Gates with Gate Only Inputs. | Gates with Event only Inputs | Most Repeated Event |
| :---: | :---: | :---: | :---: | :---: | :---: | :---: | :---: |
| A ${ }^{\text {aaaaaa }}$ | 2 | 2 | 4 | 2 | 1. | 2 | 2 |
| Artqual | 3 | 4 | 11 | 2 | 0 | 2 | 3 |
| Arttree | 2 | 2 | 5 | 2 | 1 | 2 | 2 |
| Astolfo | 2 | 7 | 22 | 5 | 5 | 7 | 5 |
| Bddtest | 2 | 4 | 15 | 4 | 1 | 4 | 2 |
| Benjiam | 2 | 4 | 22 | 8 | 5 | 8 | 4 |
| Bpfeg03 | 2 | 5 | 63 | 8 | 4 | 14 | 0 |
| Bpfen05 | 2 | 5 | 61 | 6 | 3 | 12 | 0 |
| Bpfig05 | 2 | 5 | 60 | 6 | 3 | 12 | 0 |
| Bpfin05 | 2 | 5 | 40 | 6 | 3 | 10 | 0 |
| Bpfpp02 | 3 | 2 | 5 | 2 | 0 | 2 | 2 |
| Bpfsw02 | 2 | 6 | 44 | 8 | 7 | 8 | 2 |
| Ch8tree | 3 | 3 | 12 | 2 | 0 | 2 | 3 |
| Dre1019 | 3 | 3 | 20 | 2 | 1 | 2 | 2 |
| Drel032 | 3 | 3 | 22 | 2 | 1 | 2 | 2 |
| Dre1057 | 3 | 4 | 33 | 3 | 2 | 4 | 2 |
| Dre1058 | 6 | 4 | 64 | 6 | 3 | 8 | 4 |
| Dre1059 | 2 | 6 | 80 | 6 | 5 | 8 | 4 |
| Dresden | 8 | 6 | 144 | 10 | 3 | 0 | 8 |
| Emerh2o | 4 | 3 | 11 | 2 | 1 | 2 | 2 |
| Fatram2 | 3 | 4 | 10 | 2 | 0 | 2 | 2 |
| Hpisf02 | 11 | 5 | 80 | 10 | 6 | 11 | 2 |
| Hpisf03 | 13 | 3 | 33 | 4 | 2 | 4 | 2 |
| Hpisf21 | 4 | 5 | 208 | 9 | 6 | 3 | 4 |
| Hpisf36 | 15 | 3 | 34 | 4 | 2 | 5 | 2 |
| Jdtreel | 3 | 3 | 7 | 2 | 0 | 2 | 0 |
| Jdtree2 | 3 | 3 | 7 | 2 | 0 | 2 | 0 |
| Jdtree3 | 3 | 6 | 21 | 2 | 0 | 2 | 0 |
| Jdtree4 | 3 | 6 | 21 | 2 | 0 | 2 | 2 |
| Jdtree5 | 3 | 6 | 21 | 2 | 0 | 2 | 2 |
| Khictre | 2 | 4 | 118 | 34 | 1 | 1 | 16 |
| Modtree | 4 | 3 | 7 | 2 | 0 | 2 | 2 |
| Nakashi | 2 | 6 | 29 | 8 | 4 | 9 | 3 |
| Newtre2 | 2 | 3 | 9 | 2 | 1 | 3 | 3 |
| Newtre3 | 2 | 3 | 6 | 2 | 1 | 2 | 2 |
| Newtree | 2 | 3 | 7 | 2 | 1 | 2 | 2 |
| Relcour | 3 | 2 | 6 | 2 | 0 | 2 | 0 |
| Rstreel | 2 | 4 | 6 | 1 | 0 | 1 | 2 |
| Rstree2 | 2 | 5 | 7 | 1 | 0 | 1 | 2 |
| Rstree3 | 2 | 5 | 10 | 2 | 1 | 2 | 3 |
| Rstree4 | 4 | 3 | 10 | 3 | 1 | 2 | 4 |
| Rstree5 | 4 | 2 | 6 | 2 | 0 | 2 | 2 |
| Rstree6 | 6 | 2 | 8 | 2 | 0 | 2 | 2 |


| Tree | Outputs | Levels | Events | Max.Gates | Gate Only | Event only | Rep Event |
| :---: | :---: | :---: | :---: | :---: | :---: | :---: | :---: |
| Restree7 | 2 | 4 | 13 | 3 | 1 | 3 | 3 |
| Usatree | 3 | 2 | 5 | 2 | 0 | 2 | 2 |
| Worrell | 2 | 4 | 13 | 4 | 1 | 4 | 3 |
| Trials1 | 2 | 9 | 65 | 8 | 8 | 9 | 8 |
| Trials2 | 5 | 7 | 32 | 4 | 4 | 8 | 7 |
| Trials3 | 4 | 9 | 44 | 4 | 3 | 7 | 5 |
| Trials4 | 2 | 12 | 85 | 13 | 13 | 7 | 9 |
| Random1 | 5 | 3 | 12 | 3 | 0 | 3 | 3 |
| Random2 | 3 | 2 | 7 | 1 | 0 | 1 | 2 |
| Random3 | 5 | 7 | 61 | 8 | 0 | 12 | 2 |
| Random4 | 6 | 2 | 9 | 1 | 0 | 1 | 2 |
| Random5 | 3 | 8 | 134 | 18 | 5 | 23 | 5 |
| Random6 | 4 | 5 | 122 | 21 | 3 | 24 | 5 |
| Random7 | 5 | 3 | 8 | 1 | 0 | 1 | 2 |
| Random8 | 3 | 5 | 21 | 2 | 1 | 3 | 2 |
| Random9 | 5 | 4 | 17 | 2 | 0 | 2 | 3 |
| Randol0 | 5 | 2 | 8 | 1 | 0 | 1 | 2 |
| Randol1 | 7 | 5 | 143 | 23 | 0 | 27 | 3 |
| Randol2 | 2 | 5 | 98 | 13 | 3 | 17 | 4 |
| Randol3 | 4 | 5 | 140 | 24 | 0 | 26 | 6 |
| Randol4 | 3 | 3 | 9 | 1 | 0 | 1 | 2 |
| Rando15 | 6 | 3 | 18 | 3 | 0 | 3 | 4 |
| Randol6 | 2 | 7 | 84 | 12 | 1 | 17 | 6 |
| Randol7 | 6 | 2 | 7 | 1 | 0 | 1 | 2 |
| Rando18 | 7 | 6 | 178 | 28 | 4 | 33 | 5 |
| Rando19 | 3 | 5 | 133 | 22 | 3 | 27 | 6 |
| Rando20 | 4 | 7 | 143 | 17 | 4 | 24 | 6 |
| Rando21 | 2 | 4 | 11 | 2 | 1 | 2 | 0 |
| Rando22 | 4 | 6 | 128 | 22 | 1 | 23 | 5 |
| Rando23 | 4 | 6 | 56 | 7 | 1 | 7 | 3 |
| Rando24 | 5 | 2 | 8 | 1 | 0 | 1 | 2 |
| Rando25 | 5 | 4 | 33 | 6 | 1 | 8 | 5 |
| Rando26 | 3 | 4 | 15 | 3 | 0 | 3 | 3 |
| Rando27 | 5 | 7 | 115 | 19 | 2 | 22 | 5 |
| Rando28 | 2 | 8 | 50 | 3 | 1 | 7 | 3 |
| Rando29 | 4 | 6 | 67 | 11 | 3 | 12 | 4 |
| Rando30 | 3 | 5 | 45 | 6 | 2 | 7 | 2 |
| Rando31 | 2 | 8 | 120 | 16 | 2 | 25 | 7 |
| Rando32 | 6 | 3 | 15 | 2 | 0 | 2 | 3 |
| Rando33 | 2 | 4 | 63 | 9 | 1 | 10 | 5 |
| Rando34 | 5 | 7 | 61 | 8 | 0 | 12 | 5 |
| Rando35 | 4 | 5 | 51 | 8 | 1 | 10 | 5 |
| Rando36 | 6 | 5 | 37 | 4 | 0 | 7 | 3 |
| Rando37 | 5 | 5 | 74 | 13 | 0 | 13 | 5 |
| Rando38 | 5 | 5 | 26 | 3 | 0 | 6 | 2 |
| Rando39 | 3 | 6 | 66 | 10 | 3 | 12 | 4 |
| Rando40 | 4 | 4 | 22 | 4 | 1 | 4 | 2 |
| Rando41 | 6 | 4 | 12 | 1 | 0 | 1 | 2 |
| Rando42 | 5 | 4 | 24 | 3 | 0 | 4 | 2 |
| Rando43 | 7 | 4 | 31 | 4 | 0 | 5 | 3 |
| Rando44 | 4 | 6 | 68 | 8 | 2 | 12 | 2 |
| Rando45 | 2 | 5 | 60 | 12 | 1 | 12 | 5 |


| Tree | Outputs | Levels | Events. | Max.Gates | Gate Only | Event only | Rep Event |
| :---: | :---: | :---: | :---: | :---: | :---: | :---: | :---: |
| Rando46 | 6 | 5 | 69 | 10 | 0 | 11 | 4 |
| Rando47 | 4 | 4 | 62 | 11 | 0 | 11 | 3 |
| Rando48 | 6 | 4 | 42 | 7 | 1 | 9 | 4 |
| Rando49 | 7 | 5 | 24 | 3 | 1 | 4 | 4 |
| Rando50 | 6 | 4 | 12 | 1 | 0 | 1 | 2 |
| Rando51 | 4 | 3 | 9 | 1 | 0 | 1 | 3 |
| Rando52 | 2 | 10 | 80 | 12 | 3 | 16 | 5 |
| Rando53 | 5 | 5 | 35 | 4 | 0 | 6 | 3 |
| Rando54 | 5 | 8 | 39 | 3 | 1 | 6 | 2 |
| Rando55 | 2 | 6 | 41 | 6 | 0 | 6 | 4 |
| Rando56 | 2 | 4 | 15 | 2 | 0 | 3 | 2 |
| Rando57 | 5 | 4 | 17 | 2 | 0 | 2 | 4 |
| Rando58 | 5 | 5 | 28 | 5 | 0 | 5 | 3 |
| Rando59 | 5 | 4 | 60 | 4 | 3 | 15 | 4 |
| Rando60 | 7 | 6 | 87 | 14 | 3 | 18 | 3 |
| Rando61 | 3 | 6 | 51 | 6 | 0 | 8 | 5 |
| Rando62 | 5 | 5 | 35 | 4 | 0 | 6 | 4 |
| Rando63 | 2 | 6 | 41 | 6 | 0 | 6 | 4 |
| Rando64 | 4 | 6 | 45 | 8 | 2 | 10 | 2 |
| Rando65 | 4 | 4 | 25 | 4 | 0 | 6 | 4 |
| Rando66 | 6 | 7 | 51 | 4 | 2 | 6 | 4 |
| Rando67 | 5 | 2 | 6 | 1 | 0 | 1 | 2 |
| Rando68 | 6 | 4 | 19 | 2 | 0 | 3 | 5 |
| Rand069 | 6 | 3 | 12 | 2 | 0 | 2 | 2 |
| Rando70 | 7 | 6 | 28 | 3 | 1 | 4 | 2 |
| Rando71 | 4 | 3 | 10 | 2 | 0 | 2 | 3 |
| Rando72 | 5 | 3 | 14 | 3 | 0 | 3 | 3 |
| Rando 73 | 6 | 5 | 65 | 7 | 1 | 11 | 4 |
| Rando74 | 3 | 3 | 8 | 1 | 0 | 1 | 2 |
| Rando75 | 7 | 6 | 28 | 3 | 1 | 4 | 3 |
| Rando76 | 8 | 5 | 45 | 5 | 1 | 9 | 4 |
| Rando77 | 5 | 6 | 79 | 10 | 1 | 15 | 4 |
| Rando78 | 5 | 5 | 38 | 7 | 1 | 9 | 3 |
| Rando79 | 6 | 3 | 12 | 2 | 0 | 2 | 3 |
| Rando80 | 7 | 4 | 29 | 3 | 1 | 6 | 2 |
| Rando81 | 5 | 2 | 8 | 1 | 0 | 1 | 2 |
| Rando82 | 5 | 6 | 27 | 3 | 1 | 4 | 5 |
| Rando83 | 3 | 5 | 30 | 4 | 2 | 7 | 3 |
| Rando84 | 6 | 5 | 47 | 8 | 3 | 10 | 2 |
| Rando85 | 6 | 6 | 40 | 6 | 1 | 6 | 4 |
| Rando86 | 3 | 3 | 9 | 1 | 0 | 1 | 2 |
| Rando87 | 4 | 5 | 29 | 5 | 0 | 6 | 4 |
| Rando88 | 4 | 4 | 25 | 4 | 0 | 6 | 2 |
| Rando89 | 5 | 7 | 61 | 8 | 0 | 12 | 3 |
| Rando90 | 2 | 2 | 3 | 1 | 0 | 1 | 0 |
| Rando91 | 2 | 5 | 98 | 13 | 3 | 17 | 4 |
| Rando92 | 5 | 7 | 130 | 16 | 1 | 20 | 5 |
| Rando93 | 5 | 6 | 55 | 6 | 0 | 9 | 4 |
| Rando94 | 3 | 3 | 9 | 1 | 0 | 1 | 2 |
| Rando95 | 5 | 6 | 31 | 3 | 1 | 6 | 3 |
| Rando96 | 3 | 3 | 9 | 1 | 0 | 1 | 2 |
| Rando97 | 2 | 4 | 6 | 1 | 0 | 1 | 2 |


| Tree | Outputs | Levels | Events | Max.Gates | Gate Only | Event only | Rep Event |
| :---: | :---: | :---: | :---: | :---: | :---: | :---: | :---: |
| Rando98 | 6 | 5 | 69 | 10 | 0 | 11 | 4 |
| Rando99 | 5 | 5 | 77 | 12 | 1 | 14 | 4 |
| Rand100 | 5 | 6 | 53 | 9 | 0 | 10 | 4 |
| Rand101 | 6 | 3 | 8 | 1 | 0 | 1 | 2 |
| Rand102 | 3 | 3 | 9 | 1 | 0 | 1 | 2 |
| Rand103 | 2 | 5 | 29 | 5 | 2 | 7 | 3 |
| Randi04 | 5 | 6 | 41 | 5 | 0 | 9 | 4 |
| Rand105 | 6 | 5 | 37 | 4 | 0 | 7 | 2 |
| Randl06 | 5 | 6 | 76 | 12 | 2 | 15 | 4 |
| Rand107 | 6 | 2 | 9 | 1 | 0 | 1 | 2 |
| Rand108 | 4 | 6 | 76 | 12 | 1 | 16 | 5 |
| Rand109 | 4 | 6 | 68 | 8 | 2 | 12 | 3 |
| Rand110 | 5 | 7 | 61 | 8 | 0 | 12 | 5 |
| Rand111 | 6 | 5 | 47 | 8 | 3 | 10 | 5 |
| Rand112 | 6 | 2 | 7 | 1 | 0 | 1 | 2 |
| Rand113 | 7 | 6 | 28 | 3 | 1 | 4 | 4 |
| Rand114 | 6 | 4 | 12 | 1 | 0 | 1 | 2 |
| Randl 15 | 5 | 5 | 46 | 7 | 1 | 10 | 3 |
| Randl16 | 7 | 5 | 68 | 9 | 3 | 12 | 4 |
| Rand117 | 2 | 4 | 23 | 4 | 2 | 6 | 3 |
| Rand118 | 6 | 5 | 47 | 8 | 3 | 10 | 2 |
| Randl19 | 7 | 5 | 37 | 4 | 0 | 8 | 3 |
| Randl20 | 4 | 5 | 47 | 7 | 1 | 10 | 3 |
| Rand121 | 5 | 6 | 50 | 5 | 1 | 6 | 2 |
| Rand122 | 5 | 2 | 6 | 1 | 0 | 1 | 2 |
| Rand123 | 2 | 5 | 23 | 2 | 1 | 3 | 3 |
| Rand124 | 4 | 6 | 30 | 4 | 1 | 6 | 3 |
| Rand125 | 6 | 4 | 19 | 2 | 0 | 3 | 2 |
| Randl26 | 7 | 5 | 53 | 8 | 4 | 14 | 3 |
| Rand 127 | 2 | 6 | 31 | 4 | 1 | 4 | 2 |
| Rand128 | 7 | 5 | 68 | 9 | 3 | 12 | 4 |
| Rand129 | 5 | 5 | 26 | 3 | 0 | 3 | 2 |
| Rand130 | 6 | 6 | 40 | 6 | 1 | 6 | 3 |
| Rand131 | 4 | 3 | 10 | 2 | 0 | 2 | 3 |
| Rand132 | 2 | 6 | 84 | 9 | 3 | 14 | 5 |
| Rand133 | 3 | 2 | 9 | 2 | 0 | 2 | 3 |
| Randl34 | 5 | 6 | 98 | 14 | 1 | 15 | 4 |
| Rand135 | 3 | 6 | 64 | 10 | 1 | 13 | 4 |
| Rand136 | 5 | 2 | 6 | 1 | 0 | 1 | 2 |
| Rand137 | 5 | 6 | 26 | 3 | 1 | 4 | 2 |
| Rand138 | 3 | 6 | 31 | 3 | 0 | 5 | 3 |
| Rand139 | 3 | 6 | 50 | 7 | 3 | 10 | 3 |
| Randl40 | 6 | 3 | 11 | 1 | 0 | 1 | 3 |
| Rand141 | 5 | 7 | 61 | 8 | 0 | 12 | 5 |
| Rand142 | 2 | 6 | 97 | 12 | 4 | 15 | 6 |
| Rand143 | 4 | 6 | 40 | 5 | 0 | 9 | 3 |
| Rand144 | 7 | 5 | 85 | 13 | 2 | 16 | 5 |
| Rand145 | 6 | 5 | 34 | 4 | 0 | 4 | 2 |
| Rand146 | 5 | 6 | 26 | 3 | 1 | 4 | 2 |
| Rand147 | 3 | 6 | 90 | 16 | 4 | 19 | 5 |
| Rand148 | 3 | 5 | 31 | 4 | 0 | 6 | 3 |
| Rand149 | 4 | 5 | 64 | 9 | 1 | 12 | 3 |


| Tree, | Outputs | Levels | Events | Max. Gates | Gate Only | Event only | Rep Event |
| :---: | :---: | :---: | :---: | :---: | :---: | :---: | :---: |
| Rand150 | 3 | 5 | 74 | 11 | 3 | 17 | 4 |
| Rand151 | 7 | 5 | 32 | 3 | 1 | 6 | 2 |
| Rand152 | 2 | 2 | 3 | 1 | 0 | 1 | 2 |
| Rand153 | 5 | 6 | 47 | 6 | 1 | 8 | 5 |
| Rand154 | 2 | 6 | 30 | 3 | 0 | 5 | 3 |
| Rand155 | 4 | 5 | 47 | 7 | 1 | 10 | 4 |
| Rand156 | 5 | 5 | 28 | 5 | 0 | 5 | 2 |
| Rand158 | 5 | 6 | 123 | 28 | 6 | 29 | 5 |
| Lisaba9 | 4 | 5 | 46 | 9 | 1 | 10 | 2 |
| Lisab30 | 4 | 6 | 45 | 8 | 2 | 10 | 3 |
| Lisab60 | 4 | 3 | 23 | 3 | 0 | 4 | 3 |
| Lisab59 | 6 | 4 | 49 | 7 | 1 | 9 | 0 |
| Lisab31 | 4 | 5 | 94 | 14 | 2 | 16 | 6 |
| Lisaba4 | 2 | 3 | 63 | 7 | 3 | 14 | 4 |
| Lisab57 | 3 | 4 | 46 | 8 | 1 | 10 | 3 |
| Lisab28 | 2 | 5 | 22 | 3 | 0 | 4 | 0 |
| Lisab10 | 6 | 6 | 80 | 7 | 1 | 11 | 4 |
| Lisab35 | 2 | 4 | 57 | 10 | 2 | 11 | 4 |
| Lisab51 | 5 | 4 | 21 | 3 | 0 | 4 | 2 |
| Lisab44 | 7 | 3 | 33 | 6 | 0 | 7 | 3 |
| Lisab25 | 6 | 5 | 37 | 4 | 0 | 7 | 3 |
| Lisab54 | 4 | 3 | 19 | 3 | 0 | 4 | 2 |
| Lisab53 | 2 | 3 | 10 | 3 | 1 | 3 | 2 |
| Lisab52 | 4 | 5 | 94 | 14 | 2 | 16 | 8 |
| Lisab34 | 4 | 3 | 23 | 4 | 0 | 4 | 4 |
| Lisab36 | 7 | 5 | 130 | 21 | 1 | 26 | 8 |
| Lisab42 | 6 | 4 | 23 | 3 | 0 | 3 | 2 |

## Appendix V

## Network Architectures Predicting 14/20 Correct Responses, With Eleven Input Nodes and Six Output Nodes

## Network 1:

Random Number Seed $=2082$
Number of Centres $=4$
Centres were randomly chosen as:

$$
\begin{aligned}
\underline{\mu}_{1} & =\{0.644,0.235,0.733,0.001,0.538,0.462,0.489,0.459,0.138,0.971,0.263\} \\
\mu_{2} & =\{0.624,0.059,0.644,0.001,0.308,0.462,0.154,0.115,0.034,0.206,0.158\} \\
\mu_{3} & =\{0.416,0.059,0.711,0.001,0.462,0.385,0.129,0.131,0.103,0.294,0.263\} \\
\underline{\mu}_{4} & =\{0.465,0.029,0.600,0.001,0.462,0.385,0.102,0.066,0.001,0.206,0.158\}
\end{aligned}
$$

Width Parameter: 1.023383
Weights from output node $k$ to each centre (including extra bias node):

$$
\begin{aligned}
k & =1:\{0.627533,0.040619,-0.944214,2.572754,-1.308350\} \\
k & =2:\{0.728906,-0.285797,-0.712158,2.209473,-0.995605\} \\
k & =3:\{0.645231,-0.158051,0.200012,-0.248779,0.456787\} \\
k & =4:\{0.659595,0.432404,-0.780762,-2.036621,2.743286\} \\
k & =5:\{0.261979,-0.376621,3.887093,-2.044250,-1.780518\} \\
k & =6:\{0.167886,-0.868729,1.308273,1.308136,-1.978363\}
\end{aligned}
$$

Predictions: 2, 2, 2, 3, 2, 4, 2, 2, 3, 2, 2, 3, 2, 3, 1, 2, 2, 2, 2, 2.

## Network 2:

Random Number Seed $=638$
Number of Centres $=4$
Centres were randomly chosen as:

$$
\begin{aligned}
& \mu_{1}=\{0.347,0.147,0.622,0.001,0.538,0.385,0.357,0.344,0.064,0.765,0.421\} \\
& \mu_{2}=\{0.545,0.029,0.489,0.001,0.385,0.538,0.088,0.066,0.138,0.235,0.368\} \\
& \mu_{3}=\{0.465,0.029,0.600,0.001,0.462,0.385,0.102,0.066,0.001,0.206,0.158\} \\
& \underline{\mu}_{4}=\{0.624,0.088,0.489,0.999,0.154,0.385,0.269,0.213,0.103,0.500,0.211\}
\end{aligned}
$$

Width Parameter: 1.389346

Weights from output node $k$ to each centre (including extra bias node):

$$
\begin{aligned}
k & =1:\{-0.400345,0.879898,-1.373670,1.440329,0.788738\} \\
k & =2:\{0.044533,0.357495,-1.194422,1.515049,0.503082\} \\
k & =3:\{0.160515,-0.350220,-0.277689,1.152544,0.335304\} \\
k & =4:\{0.261047,-0.212697,0.719505,-0.013809,0.249889\} \\
k & =5:\{0.723707,-0.996623,2.494805,-2.038268,-0.336407\} \\
k & =6:\{0.038984,-0.743388,0.171157,0.542048,0.112501\}
\end{aligned}
$$

Predictions: 2, 2, 2, 2, 2, 4, 2, 2, 3, 2, 2, 3, 2, 2, 1, 2, 2, 2, 2,2

Network 3:

Random Number Seed $=14$
Number of Centres $=4$
Centres were randomly chosen as:

$$
\begin{aligned}
& \underline{\mu}_{1}=\{0.614,0.059,0.756,0.999,0.385,0.385,0.096,0.066,0.001,0.176,0.211\} \\
& \mu_{2}=\{0.574,0.088,0.756,0.999,0.538,0.385,0.187,0.148,0.103,0.353,0.211\} \\
& \mu_{3}=\{0.703,0.001,0.289,0.999,0.308,0.231,0.044,0.066,0.001,0.118,0.105\} \\
& \underline{\mu}_{4}=\{0.644,0.235,0.733,0.001,0.538,0.462,0.489,0.459,0.138,0.971,0.263\}
\end{aligned}
$$

## Width Parameter: <br> 2.486194

Weights from output node $k$ to each centre (including extra bias node):

$$
\begin{aligned}
k & =1:\{-0.539177,0.410477,0.621460,-0.250919,0.989006\} \\
k & =2:\{-0.023300,2.511902,-1.787384,-0.315865,0.807106\} \\
k & =3:\{-0.093994,3.335388,-3.664124,0.702023,0.865509\} \\
k & =4:\{-0.11501,-1.884018,-0.157532,2.505596,0.815941\} \\
k & =5:\{0.705609,4.511517,-4.432554,-0.518198,-0.315331\} \\
k & =6:\{0.131954,5.745249,-4.993504,-0.800303,0.079931\}
\end{aligned}
$$

Predictions: 2, 2, 2, 2, 2, 4, 3, 2, 3, 4, 2, 3, 2, 3, 1, 2, 2, 2, 2, 2

## Network 4:

Random Number Seed $=2104$
Number of Centres $=4$

Centres were randomly chosen as:

$$
\begin{aligned}
\underline{\mu}_{1} & =\{0.693,0.059,0.600,0.001,0.308,0.308,0.170,0.180,0.001,0.324,0.158\} \\
\underline{\mu}_{2} & =\{0.495,0.059,0.733,0.999,0.462,0.308,0.115,0.115,0.034,0.265,0.211\} \\
\mu_{3} & =\{0.663,0.001,0.844,0.999,0.385,0.231,0.022,0.016,0.001,0.029,0.105\} \\
\underline{\mu}_{4} & =\{0.376,0.029,0.289,0.999,0.385,0.615,0.107,0.049,0.034,0.176,0.105\}
\end{aligned}
$$

## Width Parameter: 1.345287

Weights from output node $k$ to each centre (including extra bias node):

$$
\begin{aligned}
k & =1:\{-0.368835,0.813622,2.567841,-1.570488,-0.327698\} \\
k & =2:\{0.097225,0.615604,1.304199,-0.663660,-0.218613\} \\
k & =3:\{0.071648,0.625912,-1.078751,0.925899,0.637878\} \\
k & =4:\{0.092682,0.585405,-0.161758,-0.713787,1.265244\} \\
k & =5:\{0.819168,-0.440451,-2.518150,2.101263,0.090710\} \\
k & =6:\{0.076971,0.070652,-2.066561,2.049984,0.217453\}
\end{aligned}
$$

Predictions: 2, 2, 2, 2, 2, 4, 3, 2, 4, 4, 2, 3, 2, 3, 2, 2, 2, 4, 2, 2

## Network 5:

Random Number Seed $=804$
Number of Centres $=4$
Centres were randomly chosen as:

$$
\begin{aligned}
& \mu_{1}=\{0.495,0.029,0.889,0.001,0.462,0.231,0.041,0.033,0.001,0.059,0.158\} \\
& \mu_{2}=\{0.574,0.088,0.711,0.999,0.538,0.385,0.187,0.148,0.103,0.353,0.211\} \\
& \mu_{3}=\{0.703,0.029,0.644,0.999,0.231,0.385,0.058,0.033,0.034,0.088,0.105\} \\
& \underline{\mu}_{4}=\{0.287,0.001,0.133,0.001,0.999,0.231,0.091,0.066,0.069,0.118,0.105\}
\end{aligned}
$$

Width Parameter: 2.054398
Weights from output node $k$ to each centre (including extra bias node):

$$
\begin{aligned}
k & =1:\{-0.496120,0.659004,2.716137,-1.931986,0.219772\} \\
k & =2:\{-0.044708,0.647316,1.163872,-0.694355,0.085007\} \\
k & =3:\{-0.138527,0.283928,-0.291748,0.760921,0.540497\} \\
k & =4:\{0.042103,-0.400177,-0.033504,0.405100,1.129868\} \\
k & =5:\{1.015712,0.229465,-3.061208,2.533210,-0.855728\} \\
k & =6:\{0.008986,0.480789,-1.694976,1.783416,-0.380005\}
\end{aligned}
$$

Predictions: 2, 2, 2, 2, 2, 4, 3, 2, 3, 2, 2, 3, 2, 3, 2, 2, 2, 2, 2, 2

Network 6:

Random Number Seed $=14$
Number of Centres $=5$
Centres were randomly chosen as:

$$
\begin{aligned}
\mu_{1} & =\{0.614,0.059,0.756,0.999,0.385,0.385,0.096,0.066,0.001,0.176,0.211\} \\
\mu_{2} & =\{0.574,0.088,0.756,0.999,0.538,0.385,0.187,0.148,0.103,0.353,0.211\} \\
\mu_{3} & =\{0.703,0.001,0.289,0.999,0.308,0.231,0.044,0.066,0.001,0.118,0.105\} \\
\mu_{4} & =\{0.644,0.235,0.733,0.001,0.538,0.462,0.489,0.459,0.138,0.971,0.263\} \\
\mu_{5} & =\{0.663,0.029,0.733,0.001,0.385,0.462,0.074,0.049,0.034,0.118,0.263\}
\end{aligned}
$$

Width Parameter: 2.486194
Weights from output node $k$ to each centre (including extra bias node):

$$
\begin{aligned}
k & =1:\{-0.621368,-3.372803,4.485840,-0.273254,-0.087952,0.982971\} \\
k & =2:\{-0.085686,-0.361816,1.148682,-0.332859,-0.010559,0.746399\} \\
k & =3:\{-0.157692,0.403076,-0.669434,0.684608,0.031006,0.761414\} \\
k & =4:\{-0.153404,-3.631836,1.626709,2.495237,0.318787,0.453918\} \\
k & =5:\{0.880739,12.572174,-12.667389,-0.470413,1.978851,-2.094276\} \\
k & =6:\{0.121167,5.248474,-4.486267,-0.803239,-0.061462,0.129044\}
\end{aligned}
$$

Predictions: 2, 2, 2, 2, 2, 4, 3, 2, 3, 4, 2, 3, 2, 3, 1, 2, 2, 2, 2, 2

## Network 7:

Random Number Seed $=42$
Number of Centres $=5$
Centres were randomly chosen as:

$$
\begin{aligned}
& \mu_{1}=\{0.604,0.088,0.667,0.001,0.154,0.769,0.220,0.197,0.103,0.471,0.263\} \\
& \mu_{2}=\{0.386,0.088,0.667,0.999,0.385,0.462,0.209,0.197,0.069,0.441,0.211\} \\
& \mu_{3}=\{0.663,0.088,0.756,0.999,0.385,0.385,0.203,0.213,0.001,0.382,0.263\} \\
& \mu_{4}=\{0.396,0.882,0.600,0.999,0.231,0.308,0.030,0.033,0.001,0.059,0.158\} \\
& \mu_{5}=\{0.297,0.001,0.001,0.999,0.385,0.308,0.066,0.066,0.034,0.147,0.001\}
\end{aligned}
$$

Width Parameter: 2.151013
Weights from output node $k$ to each centre (including extra bias node):

$$
\begin{aligned}
k & =1:\{-0.211445,0.638014,1.540771,0.717499,-1.959526,0.091232\} \\
k & =2:\{-0.049915,0.715805,1.545959,-1.160339,0.354324,-0.226288\} \\
k & =3:\{-0.116955,0.787764,-2.905426,2.052307,0.574234,0.852631\} \\
k & =4:\{-0.071869,0.722025,0.772614,-1.447540,0.212128,0.946533\} \\
k & =5:\{0.273493,0.027996,-3.161896,0.588211,3.102493,-0.295128\} \\
k & =6:\{-0.042212,0.139519,-3.720276,2.944519,0.489536,0.552151\}
\end{aligned}
$$

## Predictions: 2, 2, 2, 2, 2, 4, 2, 2, 4, 4, 2, 3, 2, 3, 1, 2, 2, 2, 2, 2

## Network 8:

Random Number Seed $=548$
Number of Centres $=9$
Centres were randomly chosen as:

$$
\begin{aligned}
& \mu_{1}=\{0.248,0.001,0.111,0.001,0.231,0.231,0.060,0.033,0.034,0.059,0.105\} \\
& \mu_{2}=\{0.624,0.118,0.667,0.999,0.154,0.385,0.269,0.213,0.103,0.500,0.211\} \\
& \mu_{3}=\{0.347,0.147,0.667,0.999,0.308,0.462,0.352,0.361,0.034,0.676,0.263\} \\
& \underline{\mu}_{4}=\{0.663,0.088,0.756,0.999,0.385,0.385,0.203,0.213,0.001,0.382,0.263\} \\
& \mu_{5}=\{0.356,0.059,0.556,0.001,0.538,0.385,0.157,0.148,0.034,0.324,0.158\} \\
& \mu_{6}=\{0.406,0.001,0.001,0.001,0.154,0.385,0.168,0.098,0.103,0.353,0.001\} \\
& \underline{\mu}_{7}=\{0.495,0.001,0.222,0.999,0.154,0.385,0.027,0.033,0.034,0.059,0.158\} \\
& \underline{\mu}_{8}=\{0.495,0.029,0.489,0.001,0.154,0.308,0.063,0.066,0.069,0.176,0.158\} \\
& \mu_{9}=\{0.376,0.059,0.378,0.999,0.308,0.462,0.209,0.197,0.034,0.471,0.158\}
\end{aligned}
$$

Width Parameter: 1.998337
Weights from output node $k$ to each centre (including extra bias node):

$$
\begin{aligned}
k=1: & \{-0.335327,-3.125977,-4.697266,-2.996094,2.554199,0.072266, \\
& 0.031250,-3.246094,3.546875,8.796875\} \\
k= & 2: \\
& \{-0.021729,2.694336,2.117188,-4.269531,-0.873535,-0.825159,- \\
k= & 3.271484,-5.097656,2.001953,8.281250\} \\
& \{-0.081177,-5.726562,-6.889648,3.976562,3.184082,2.348633, \\
& 3.644531,6.007812,0.359375,-5.593750\} \\
k= & 4: \begin{array}{l}
-0.113037,0.887695,5.859375,-2.972656,-4.781250,3.453125,- \\
\\
1.185547,0.140625,-2.488281,2.109375\}
\end{array}
\end{aligned}
$$

```
\(k=5:\{0.555267,0.138916,4.523926,7.046387,-0.801270,0.328491\),
    3.963135, 7.817383, -4.350586, -18.260742\}
\(k=6:\{-0.101654,-2.108398,-1.536621,6.797852,-0.288818,2.426025\),
    \(1.474854,8.219727,-1.593750,-12.535156\}\)
```

Predictions: 2, 2, 2, 2, 2, 4, 4, 2, 2, 4, 2, 2, 2, 3, 2, 2, 2, 4, 2, 2

## Appendix VI

## Results of Structural Importance Measure Variable Ordering Heuristic

Program used to generate ordering - strimpgsq.f
Program used to generate BDD and related nodes for each scheme - schemes. $f$

| Tree | Number Of Minimal Cut Sets | Number of Nodes for Previous Best | Nodes for Structural Measure | Comparison and tally |
| :---: | :---: | :---: | :---: | :---: |
| A ${ }^{\text {aaaaaa }}$ | 2 | 3 | 3 | = (1) |
| Artqual | 7 | 11 | 11 | = (2) |
| Artree | 2 | 4 | 4 | = (3) |
| Astolfo | 27 | 107 | 123 | $>1$ |
| Bddtest | 9 | 36 | 62 | $>2$ |
| Benjiam | 43 | 76 | 78 | $>3$ |
| Bpfeg03 | 8716 | 82007 | 364508 | $>4$ |
| Bpfen05 | 7471 | 51497 | 151568 | $>5$ |
| Bpfig05 | 7056 | 49067 | 142628 | $>6$ |
| Bpfin05 | 416 | 2915 | 5287 | $>7$ |
| Bpfpp02 | 3 | 4 | 4 | $=(4)$ |
| Bpfsw02 | 84424 | 110698 | 112258 | $>8$ |
| Ch8tree | 5 | 11 | 12 | $>9$ |
| Dre1019 | 63 | 69 | 69 | = (5) |
| Dre1032 | 75 | 87 | 81 | $<1$ |
| Drel057 | 2100 | 2478 | 2300 | $<2$ |
| Dre1058 | 11934 | 24764 | 23132 | $<3$ |
| Drel059 | 36990 | 61476 | 61036 | $<4$ |
| Dresden | 11934 | 221217 | 23132 | < 5 |
| Emerh2o | 13 | 16 | 16 | = (6) |
| Fatram2 | 6 | 10 | 11 | $>10$ |
| Hpisf02 | 255 | 164539 | 370980 | $>11$ |
| Hpisf03 |  | 202 | 182 | $<6$ |
| Hpisf21 | 7777 | 10593 | 30056 | $>12$ |
| Hpisf36 | 61 | 132 | 132 | $=(7)$ |
| Jdtreel | 4 | 10 | 12 | $>13$ |
| Jdtree2 | 4 | 10 | 12 | $>14$ |
| Jdtree3 | 36 | 71 | 79 | $>15$ |
| Jdtree4 | 30 | 59 | 67 | $>16$ |
| Jdtree5 | 10 | 70 | 76 | $>17$ |
| Khictre | 21 | 1134 | 982 | $<7$ |
| Modtree | 2 | 4 | 4 | = (8) |
| Nakashi | 20 | 460 | 326 | <8 |
| Newtre2 | 3 | 9 | 10 | $>18$ |
| Newtre3 | 2 | 6 | 6 | = (9) |
| Newtree | 3 | 9 | 10 | $>19$ |
| Relcour | 6 | 9 | 9 | = (10) |
| Rstree 1 | 3 | 4 | 4 | $=(11)$ |
| Rstree2 | 3 | 4 | 4 | $=(12)$ |
| Rstree3 | 6 | 11 | 10 | <9 |
| Rstree4 | 4 | 5 | 5 | $=(13)$ |
| Rstree5 | 2 | 2 | 2 | $=(14)$ |


| \% Tree | Minimal Cut Sets | Nodes for Previous Best | Structural Nodes | Comparison |
| :---: | :---: | :---: | :---: | :---: |
| Rstree6 | 4 | 4 | 4 | $=(15)$ |
| Rstree7 | 8 | 33 | 16 | <10 |
| Usatree | 2 | 4 | 4 | $=(16)$ |
| Worrell | 10 | 17 | 17 | = (17) |
| Trials1 | 45 | 913 | 416 | $<11$ |
| Trials2 | 6 | 29 | 12 | < 12 |
| Trials 3 | 2 | 2 | 2 | $=(18)$ |
| Trials4 | 49 | 302 | 288 | <13 |
| Random1 | 5 | 6 | 6 | = (19) |
| Random2 | 2 | 2 | 2 | $=(20)$ |
| Random3 | 235 | 2341 | 2647 | $>20$ |
| Random4 | 5 | 5 | 5 | $=(21)$ |
| Random6 | 93 | 40210 | 116218 | $>21$ |
| Random7 | 2 | 4 | 4 | $=(22)$ |
| Random8 | 4 | 31 | 30 | <14 |
| Random9 | 2 | 6 | 6 | = (23) |
| Randol0 | 4 | 4 | 4 | = (24) |
| Randol1 |  | $>500000$ | $>500000$ | $=(25)$ |
| Randol2 | 68 | 9545 | 6241 | <15 |
| Randol3 | 73 | 394 | 417 | $>22$ |
| Rando14 | 2 | 2 | 2 | $=(26)$ |
| Randol5 | 5 | 5 | 5 | = (27) |
| Randol6 | 76 | 453 | 420 | $<16$ |
| Rando17 | 2 | 5 | 5 | $=(28)$ |
| Rando18 | 24 | 2364 | 1015 | $<17$ |
| Randol9 | 764 | 13692 | 8749 | $<18$ |
| Rando20 | 122 | 12097 | 8216 | $<19$ |
| Rando21 | 5 | 21 | 24 | $>23$ |
| Rando22 | 423 | 89013 | 18865 | $<20$ |
| Rando23 | 9 | 150 | 141 | $<21$ |
| Rando24 | 4 | 4 | 4 | $=(29)$ |
| Rando25 | 6 | 16 | 19 | $>24$ |
| Rando26 | 3 | 4 | 4 | $=(30)$ |
| Rando27 | 100 | 1150 | 601 | <22 |
| Rando28 | 2 | 2 | 2 | = (31) |
| Rando29 | 22 | 630 | 489 | <23 |
| Rando30 | 195 | 4266 | 9866 | $>25$ |
| Rando31 | 5 | 11 | 11 | = (32) |
| Rando32 | 5 | 5 | 5 | $=(33)$ |
| Rando33 | 11 | 18 | 27 | $>26$ |
| Rando34 | 35 | 212 | 199 | <24 |
| Rando35 | 8 | 180 | 47 | <25 |
| Rando36 | 10 | 62 | 44 | $<26$ |
| Rando37 | 29 | 136 | 72 | $<27$ |
| Rando38 | 9 | 50 | 57 | $>27$ |
| Rando39 | 51 | 626 | 581 | $<28$ |
| Rando40 | 9 | 44 | 33 | $<29$ |
| Rando41 | 2 | 5 | 5 | = (34) |
| Rando42 | 2 | 5 | 5 | = (35) |
| Rando43 | 22 | 94 | 94 | = (36) |
| Rando44 | 436 | 81850 | 230239 | $>28$ |
| Rando45 | 16 | 138 | 85 | $<30$ |


| Tree | Minimal Cut Sets | Nodes for Previous Best | Structural Nodes | Comparison |
| :---: | :---: | :---: | :---: | :---: |
| Rando46 | 10 | 16 | 16 | $=(37)$ |
| Rando47 | 15 | 1113 | 1148 | $>29$ |
| Rando48 | 16 | 34 | 34 | $=(38)$ |
| Rando49 | 4 | 18 | 20 | $>30$ |
| Rando50 | 2 | 5 | 5 | = (39) |
| Rando51 | 3 | 3 | 3 | $=(40)$ |
| Rando52 | 41 | 492 | 115 | $<31$ |
| Rando53 | 2 | 5 | 5 | $=(41)$ |
| Rando54 | 269 | 756 | 1151 | $>31$ |
| Rando55 | 9 | 24 | 25 | $>32$ |
| Rando56 | 3 | 7 | 7 | $=(42)$ |
| Rando57 | 2 | 6 | 6 | $=(43)$ |
| Rando58 | 3 | 9 | 9 | = (44) |
| Rando59 | 99 | 19289 | 15319 | <32 |
| Rando60 | 22 | 639 | 391 | $<33$ |
| Rando61 | 15 | 79 | 40 | $<34$ |
| Rando62 | 7 | 21 | 11 | <35 |
| Rando63 | 9 | 24 | 25 | $>33$ |
| Rando64 | 31 | 2568 | 782 | <36 |
| Rando65 | 13 | 92 | 93 | $>34$ |
| Rando66 | 5 | 131 | 84 | $<37$ |
| Rando67 | 1 | 4 | 4 | $=(45)$ |
| Rando68 | 5 | 6 | 6 | = (46) |
| Rando69 | 6 | 8 | 8 | = (47) |
| Rando70 | 27 | 80 | 74 | $<38$ |
| Rando71 | 2 | 7 | 6 | $<39$ |
| Rando 72 | 2 | 2 | 2 | = (48) |
| Rando73 | 80 | 614 | 199 | <40 |
| Rando74 | 2 | 2 | 2 | $=(49)$ |
| Rando75 | 4 | 16 | 16 | $=(50)$ |
| Rando76 | 24 | 299 | 223 | <41 |
| Rando 77 | 27 | 230 | 84 | $<42$ |
| Rando78 | 2 | 5 | 5 | = (51) |
| Rando79 | 4 | 4 | 4 | $=(52)$ |
| Rando80 | 22 | 119 | 118 | $<43$ |
| Rando81 | 4 | 4 | 4 | = (53) |
| Rando82 | 5 | 6 | 6 | $=(54)$ |
| Rando83 | 39 | 219 | 204 | $<44$ |
| Rando84 | 52 | 703 | 366 | $<45$ |
| Rando85 | 7 | 18 | 15 | $<46$ |
| Rando86 | 2 | 2 | 2 | $=(55)$ |
| Rando87 | 15 | 21 | 19 | $<47$ |
| Rando88 | 29 | 504 | 662 | $>35$ |
| Rando89 | 21 | 238 | 180 | $<48$ |
| Rando90 | 2 | 3 | 3 | $=(56)$ |
| Rando91 | 106 | 32108 | 12263 | <49 |
| Rando92 | 58 | 9526 | 3278 | $<50$ |
| Rando93 | 16 | 61 | 46 | $<51$ |
| Rando94 | 1 | 2 | 2 | $=(57)$ |
| Rando95 | 31 | 57 | 79 | $>36$ |
| Rando96 | 5 | 7 | 7 | = (58) |
| Rando97 | 2 | 3 | 3 | = (59) |


| Tree | Minimal Cut Sets | Nodes for Previous Best | Structural Nodes | Comparison |
| :---: | :---: | :---: | :---: | :---: |
| Rando98 | 283 | 24088 | 16086 | < 52 |
| Rando99 | 28 | 737 | 732 | $<53$ |
| Rand100 | 8 | 21 | 27 | >37 |
| Randl01 | 2 | 7 | 7 | $=(60)$ |
| Rand102 | 2 | 2 | 2 | $=(61)$ |
| Rand103 | 13 | 94 | 118 | $>38$ |
| Rand104 | 9 | 110 | 118 | > 39 |
| Rand105 | 96 | 953 | 936 | $<54$ |
| Rand106 | 8 | 41 | 17 | $<55$ |
| Rand107 | 5 | 5 | 5 | = (62) |
| Rand108 | 35 | 162 | 132 | $<56$ |
| Rand109 | 203 | 9197 | 3260 | $<57$ |
| Rand110 | 8 | 39 | 33 | $<58$ |
| Rand111 | 22 | 217 | 76 | $<59$ |
| Rand112 | 2 | 5 | 5 | $=(63)$ |
| Rand113 | 2 | 6 | 6 | $=(64)$ |
| Randl14 | 2 | 7 | 7 | $=(65)$ |
| Rand115 | 46 | 851 | 438 | $<60$ |
| Rand116 | 15 | 937 | 265 | $<61$ |
| Rand117 | 11 | 32 | 32 | = (66) |
| Rand118 | 52 | 703 | 366 | <62 |
| Rand119 | 84 | 309 | 133 | <63 |
| Rand120 | 48 | 3392 | 3287 | <64 |
| Rand121 | 80 | 156 | 132 | <65 |
| Rand 122 | 4 | 4 | 4 | = (67) |
| Rand 123 | 12 | 20 | 17 | <66 |
| Rand 124 | 27 | 178 | 144 | <67 |
| Rand125 | 13 | 24 | 21 | $<68$ |
| Rand126 | 59 | 1744 | 2956 | $>40$ |
| Rand127 | 43 | 218 | 292 | >41 |
| Rand128 | 52 | 886 | 882 | $<69$ |
| Rand129 | 2 | 4 | 4 | = (68) |
| Rand 130 | 5 | 5 | 5 | = (69) |
| Rand131 | 2 | 8 | 7 | < 70 |
| Rand132 | 67 | 5128 | 3685 | $<71$ |
| Rand133 | 4 | 5 | 5 | $=(70)$ |
| Rand134 | 60 | 620 | 566 | $<72$ |
| Rand135 | 24 | 382 | 386 | $>42$ |
| Rand 136 | 1 | 4 | 4 | = (71) |
| Rand137 | 15 | 139 | 99 | $<73$ |
| Rand138 | 2 | 2 | 2 | $=(72)$ |
| Rand139 | 49 | 478 | 469 | $<74$ |
| Rand140 | 5 | 5 | 5 | = (73) |
| Rand141 | 8 | 39 | 33 | $<75$ |
| Rand142 | 410 | 55680 | 53665 | $<76$ |
| Rand143 | 8 | 253 | 146 | $<77$ |
| Randl44 | 41 | 5197 | 1707 | $<78$ |
| Rand145 | 47 | 303 | 444 | $>43$ |
| Rand146 | 15 | 139 | 99 | $<79$ |
| Rand147 | 30 | 2548 | 5884 | $>44$ |
| Rand148 | 8 | 35 | 42 | $>45$ |
| Rand149 | 18 | 87 | 135 | $>46$ |


| \% Tree, | Minimal Cut Sets | Nodes for Previous Best | Structural Nodes | Comparison |
| :---: | :---: | :---: | :---: | :---: |
| Rand150 | 114 | 48700 | 36036 | $<80$ |
| Rand151 | 36 | 148 | 218 | $>47$ |
| Rand152 | 1 | 1 | 1 | = (74) |
| Rand 153 | 3 | 8 | 8 | $=(75)$ |
| Rand154 | 1 | 1 | 1 | $=(76)$ |
| Rand155 | 52 | 905 | 790 | <81 |
| Rand156 | 20 | 43 | 40 | $<82$ |
| Rand158 | 9 | 30 | 24 | $<83$ |
| Lisaba9 | 85 | 3515 | 3211 | $<84$ |
| Lisab30 | 17 | 129 | 87 | <85 |
| Lisab60 | 19 | 66 | 47 | $<86$ |
| Lisab59 | 3096 | 43242 | 114954 | >48 |
| Lisab31 | 164 | 4534 | 4896 | $>49$ |
| Lisaba4 | 827 | 9959 | 5523 | $<87$ |
| Lisab57 | 170 | 1245 | 1151 | $<88$ |
| Lisab28 | 66 | 156 | 162 | $>50$ |
| Lisab10 | 940 | 9593 | 6438 | < 89 |
| Lisab35 | 136 | 17901 | 12170 | $<90$ |
| Lisab51 | 11 | 103 | 91 | $<91$ |
| Lisab44 | 12 | 158 | 53 | $<92$ |
| Lisab25 | 35 | 165 | 158 | $<93$ |
| Lisab54 | 14 | 40 | 60 | $>51$ |
| Lisab53 | 15 | 25 | 21 | <94 |
| Lisab52 | 139 | 4981 | 2674 | $<95$ |
| Lisab34 | 14 | 33 | 34 | , $>252$, |
| Lisab36 | 52 | 480 | 316 |  |
| Lisab42 | 10 | 17 | 17 | \% $2(77)$ ) |

## Appendix VII

## Difference in BDDs Affected by The Structural Importance Measure With Additional Subroutine of Ordering Matched Components by The Number of Repetitions

Program used to generate ordering - strrep.f
Program used to generate BDD and number of nodes - schemes. $f$

| Tree | Nodes of Best of 6 Alternatives | Nodes of Structural Importance Measure | Nodes with addition of repeated event separation | Effect on results |
| :---: | :---: | :---: | :---: | :---: |
| Artqual | 11 | 11 | 11 | No change |
| Benjiam | 76 | 78 | 84 | $>$, but No change to result |
| Ch8tree | 11 | 12 | 12 | No change |
| Drel019 | 69 | 69 | 69 | No change |
| Dre1032 | 87 | 81 | 81 | No change |
| Dre1057 | 2478 | 2300 | 2300 | No change |
| Emerh2o | 16 | 16 | 16 | No change |
| Khictre | 1134 | 982 | 982 | No change |
| Newtre3 | 6 | 6 | 7 | Worse ( $>$ ) |
| Worrell | 17 | 17 | 18 | Worse ( $>$ ) |
| Trials 1 | 913 | 416 | 395 | < but No change to result |
| Trials2 | 29 | 12 | 12 | No change |
| Trials3 | 2 | 2 | 2 | No change |
| Random3 | 2341 | 2647 | 2647 | No change |
| Random4 | 5 | 5 | 5 | No change |
| Random6 | 40210 | 116218 | 116218 | No change |
| Rando10 | 4 | 4 | 4 | No change |
| Rando 12 | 9545 | 6241 | 6576 | $\geq$, but No change to result |
| Rando13 | 394 | 417 | 417 | No change |
| Rando15 | 5 | 5 | 5 | No change |
| Randol6 | 453 | 420 | 420 | No change |
| Randol7 | 5 | 5 | 5 | No change |
| Rando18 | 2364 | 1015 | 1015 | No change |
| Randol9 | 13692 | 8749 | 8749 | No change |
| Rando20 | 12097 | 8216 | 8216 | No change |
| Rando22 | 89013 | 18865 | 18865 | No change |
| Rando23 | 150 | 141 | 135 | <, but No change to result |
| Rando24 | 4 | 4 | 4 | No change |
| Rando25 | 16 | 19 | 19 | No change |
| Rando26 | 4 | 4 | 4 | No change |
| Rando27 | 1150 | 601 | 601 | No change |
| Rando28 | 2 | 2 | 2 | No change |
| Rando29 | 630 | 489 | 489 | No change |
| Rando31 | 11 | 11 | 11 | No change |
| Rando32 | 5 | 5 | 5 | No change |
| Rando33 | 18 | 27 | 23 | <, but No change to result |
| Rando34 | 212 | 199 | 199 | No change |
| Rando35 | 180 | 47 | 47 | No change |


| Tree | Nodes of Best | Structural Measure | Repeated Separation: | Effect on results |
| :---: | :---: | :---: | :---: | :---: |
| Rando36 | 62 | 44 | 44 | No change |
| Rando37 | 136 | 72 | 72 | No change |
| Rando38 | 50 | 57 | 57 | No change |
| Rando39 | 626 | 581 | 581 | No change |
| Rando41 | 5 | 5 | 5 | No change |
| Rando42 | 5 | 5 | 5 | No change |
| Rando43 | 94 | 94 | 94 | No change |
| Rando44 | 81850 | 230239 | 230239 | No change |
| Rando45 | 138 | 85 | 85 | No change |
| Rando46 | 16 | 16 | 16 | No change |
| Rando47 | 1113 | 1148 | 1148 | No change |
| Rando48 | 34 | 34 | 34 | No change |
| Rando49 | 18 | 20 | 20 | No change |
| Rando50 | 5 | 5 | 5 | No change |
| Rando51 | 3 | 3 | 3 | No change |
| Rando52 | 492 | 115 | 115 | No change |
| Rando53 | 5 | 5 | 5 | No change |
| Rando55 | 24 | 25 | 25 | No change |
| Rando56 | 7 | 7 | 7 | No change |
| Rando59 | 19289 | 15319 | 15319 | No change |
| Rando60 | 639 | 391 | 391 | No change |
| Rando61 | 79 | 40 | 40 | No change |
| Rando62 | 21 | 11 | 11 | No change |
| Rando63 | 24 | 25 | 25 | No change |
| Rando64 | 2568 | 782 | 782 | No change |
| Rando65 | 92 | 93 | 93 | No change |
| Rando66 | 131 | 84 | 84 | No change |
| Rando67 | 4 | 4 | 4 | No change |
| Rando68 | 6 | 6 | 6 | No change |
| Rando69 | 8 | 8 | 8 | No change |
| Rando72 | 2 | 2 | 2 | No change |
| Rando73 | 614 | 299 | 299 | No change |
| Rando 75 | 16 | 16 | 16 | No change |
| Rando 76 | 299 | 223 | 223 | No change |
| Rando77 | 230 | 84 | 84 | No change |
| Rando78 | 5 | 5 | 5 | No change |
| Rando79 | 4 | 4 | 4 | No change |
| Rando80 | 119 | 118 | 118 | No change |
| Rando81 | 4 | 4 | 4 | No change |
| Rando82 | 6 | 6 | 6 | No change |
| Rando83 | 219 | 204 | 204 | No change |
| Rando84 | 703 | 366 | 366 | No change |
| Rando85 | 18 | 15 | 15 | No change |
| Rando87 | 21 | 19 | 19 | No change |
| Rando89 | 238 | 180 | 180 | No change |
| Rando91 | 32108 | 12263 | 12263 | No change |
| Rando92 | 9526 | 3278 | 3278 | No change |
| Rando93 | 61 | 46 | 46 | No change |
| Rando96 | 7 | 7 | 7 | No change |
| Rando98 | 24088 | 16086 | 16086 | No change |
| Rando99 | 737 | 732 | 693 | <, but No change to result |
| Rand100 | 21 | 27 | 22 | <, but No change to result |


| Tree | Nodes of Best | Structural Measure | Repeated Separation | Effect on results |
| :---: | :---: | :---: | :---: | :---: |
| Rand101 | 7 | 7 | 7 | No change |
| Rand103 | 94 | 118 | 118 | No change |
| Rand104 | 110 | 118 | 88 | <, Result < |
| Rand106 | 41 | 17 | 17 | No change |
| Rand107 | 5 | 5 | 5 | No change |
| Rand108 | 162 | 132 | 132 | No change |
| Rand109 | 9197 | 3260 | 3260 | No change |
| Rand110 | 39 | 33 | 33 | No change |
| Rand111 | 217 | 76 | 76 | No change |
| Rand112 | 5 | 5 | 5 | No change |
| Rand113 | 6 | 6 | 6 | No change |
| Rand 114 | 7 | 7 | 7 | No change |
| Rand 115 | 851 | 438 | 438 | No change |
| Rand116 | 937 | 265 | 265 | No change |
| Rand118 | 703 | 366 | 366 | No change |
| Rand119 | 309 | 133 | 269 | $>$, but No change to result |
| Rand121 | 156 | 132 | 132 | No change |
| Rand123 | 20 | 17 | 17 | No change |
| Rand124 | 178 | 144 | 144 | No change |
| Rand 125 | 24 | 21 | 21 | No change |
| Rand 126 | 1744 | 2956 | 2956 | No change |
| Rand127 | 218 | 292 | 292 | No change |
| Rand128 | 886 | 882 | 882 | No change |
| Rand129 | 4 | 4 | 4 | No change |
| Rand130 | 5 | 5 | 5 | No change |
| Rand 131 | 8 | 7 | 7 | No change |
| Rand132 | 5128 | 3685 | 3685 | No change |
| Rand133 | 5 | 5 | 5 | No change |
| Rand134 | 620 | 566 | 566 | No change |
| Rand135 | 382 | 386 | 386 | No change |
| Rand136 | 4 | 4 | 4 | No change |
| Rand138 | 2 | 2 | 2 | No change |
| Rand139 | 478 | 469 | 469 | No change |
| Rand140 | 5 | 5 | 5 | No change |
| Rand 141 | 39 | 33 | 33 | No change |
| Rand142 | 55680 | 53665 | 53665 | No change |
| Rand143 | 253 | 146 | 146 | No change |
| Rand144 | 5197 | 1707 | 1707 | No change |
| Rand147 | 2548 | 5884 | 5884 | No change |
| Rand148 | 35 | 45 | 32 | <, Result < |
| Rand 149 | 87 | 135 | 159 | $>$, but No change in result |
| Rand150 | 48700 | 36036 | 36036 | No change |
| Rand151 | 148 | 218 | 218 | No change |
| Rand153 | 8 | 8 | 8 | No change |
| Rand154 | 1 | 1 | 1 | No change |
| Rand155 | 905 | 790 | 790 | No change |
| Rand156 | 43 | 40 | 40 | No change |
| Rand158 | 30 | 24 | 24 | No change |
| Lisaba9 | 3515 | 3211 | 3211 | No change |
| Lisab30 | 129 | 87 | 87 | No change |
| Lisab60 | 66 | 47 | 47 | No change |
| Lisab31 | 4534 | 4896 | 4160 | <, Result < |


| Tree | Nodes of Best | Structural Measure | Repeated Separation | Effect on results |
| :---: | :---: | :---: | :---: | :---: |
| Lisaba4 | 9959 | 5523 | 5523 | No change |
| Lisab57 | 1245 | 1151 | 1151 | No change |
| Lisab10 | 9593 | 6438 | 6438 | No change |
| Lisab35 | 17901 | 12170 | 12170 | No change |
| Lisab44 | 158 | 53 | 53 | No change |
| Lisab25 | 165 | 158 | 158 | No change |
| Lisab54 | 40 | 60 | 60 | No change |
| Lisab52 | 4981 | 2674 | 2674 | No change |
| Lisab36 | 480 | 316 | 316 | No change |
| Lisab42 | 17 | 17 | 17 | No change |

## Appendix VIII

## Comparison of Results for BDD Size Using The Best of Six Ordering Heuristics and Simpler Weighting Methods For The Structural Importance <br> Measures.

Approach 1 program - sim_strl.f
Approach 2 program - sim_str2.f
Approach 3 program - sim_str3.f

| Tree | Best Result | Simpler Weighting Methods |  |  |
| :---: | :---: | :---: | :---: | :---: |
|  |  | Approach 1 ac | 4 Approach 2 | Approach 3 |
| Aaaaaa | 3 | 3 | 3 | 3 |
| Artqual | 11 | 11 | 11 | 11 |
| Arttree | 4 | 4 | 4 | 4 |
| Astolfo | 107 | 121 | 121 | 119 |
| Bddtest | 36 | 62 | 90 | 62 |
| Benjiam | 76 | 87 | 86 | 87 |
| Bpfeg03 | 82007 | 321123 | 316983 | 316983 |
| Bpfen05 | 51497 | 151563 | 150543 | 150543 |
| Bpfig05 | 49067 | 143643 | 142623 | 142623 |
| Bpfin05 | 2915 | 5282 | 5282 | 5282 |
| Bpfpp02 | 4 | 4 | 4 | 4 |
| Bpfsw02 | 110698 | 112553 | 112553 | 112553 |
| Ch8tree | 11 | 11 | 11 | 12 |
| Dre1019 | 69 | 69 | 75 | 69 |
| Dre1032 | 87 | 81 | 87 | 81 |
| Drel057 | 2478 | 2310 | 2478 | 2310 |
| Drel058 | 24764 | 22652 | 27620 | 23732 |
| Dre1059 | 61476 | 57004 | 63764 | 59876 |
| Dresden | 221217 | $>500000$ | $>500000$ | >500000 |
| Emerh2o | 16 | 16 | 16 | 16 |
| Fatram2 | 10 | 13 | 13 | 10 |
| Hpisf02 | 164539 | 415085 | 276295 | 270005 |
| Hpisf03 | 202 | 202 | 202 | 202 |
| Hpisf21 | 10593 | 18299 | 32359 | 32353 |
| Hpisf32 | 132 | 210 | 390 | 210 |
| Jdtreel | 10 | 12 | 12 | 12 |
| Jdtree2 | 10 | 12 | 12 | 12 |
| Jdtree3 | 71 | 79 | 79 | 79 |
| Jdtree4 | 59 | 67 | 67 | 67 |
| Jdtree5 | 70 | 76 | 76 | 76 |
| Khictre | 1134 | 1369 | 1369 | 1364 |
| Modtree | 4 | 4 | 4 | 4 |
| Nakashi | 460 | 532 | 560 | 574 |
| Newtre2 | 9 | 9 | 9 | 9 |
| Newtre3 | 6 | 6 | 6 | 6 |
| Newtree | 9 | 9 | 9 | 9 |
| Relcour | 9 | 9 | 9 | 9 |
| Rstreel | 4 | 4 | 4 | 4 |
| Rstree2 | 4 | 4 | 4 | 4 |


| , M, Tree | Best | Approach 1 | Approach 2 | Approach 3 |
| :---: | :---: | :---: | :---: | :---: |
| Rstree3 | 11 | 14 | 14 | 14 |
| Rstree4 | 5 | 5 | 5 | 5 |
| Rstree5 | 2 | 2 | 2 | 2 |
| Rstree6 | 4 | 4 | 4 | 4 |
| Rstree 7 | 33 | 20 | 16 | 16 |
| Usatree | 4 | 4 | 4 | 4 |
| Worrell | 17 | 17 | 18 | 18 |
| Trials 1 | 913 | 1084 | 1125 | 754 |
| Trials2 | 29 | 13 | 11 | 12 |
| Trials3 | 2 | 2 | 2 | 2 |
| Trials4 | 302 | 235 | 222 | 199 |
| Random1 | 6 | 6 | 6 | 6 |
| Random2 | 2 | 2 | 2 | 2 |
| Random3 | 2341 | 4048 | 4665 | 2584 |
| Random4 | 5 | 5 | 5 | 5 |
| Random6 | 40210 | 44132 | 101282 | 49164 |
| Random7 | 4 | 4 | 4 | 4 |
| Random8 | 31 | 36 | 37 | 36 |
| Random9 | 6 | 6 | 6 | 6 |
| Randol0 | 4 | 4 | 4 | 4 |
| Randol1 | $>500000$ | $>500000$ | $>500000$ | $>500000$ |
| Randol2 | 9545 | 6420 | 6474 | 6419 |
| Randol3 | 394 | 1028 | 1167 | 762 |
| Randol4 | 2 | 2 | 2 | 2 |
| Randol5 | 5 | 5 | 5 | 5 |
| Randol6 | 453 | 601 | 665 | 665 |
| Rando17 | 5 | 5 | 5 | 5 |
| Randol8 | 2364 | 4939 | 5355 | 2685 |
| Randol9 | 13692 | 17769 | 25258 | 19066 |
| Rando20 | 12097 | 20682 | 26999 | 18718 |
| Rando21 | 21 | 36 | 36 | 36 |
| Rando22 | 89013 | 24855 | 20620 | 16073 |
| Rando23 | 150 | 144 | 144 | 138 |
| Rando24 | 4 | 4 | 4 | 4 |
| Rando25 | 16 | 29 | 17 | 17 |
| Rando26 | 4 | 4 | 4 | 4 |
| Rando27 | 1150 | 1264 | 1318 | 586 |
| Rando28 | 2 | 2 | 2 | 2 |
| Rando29 | 630 | 1756 | 1928 | 1471 |
| Rando30 | 4266 | 16758 | 15930 | 11945 |
| Rando31 | 11 | 29 | 53 | 53 |
| Rando32 | 5 | 5 | 5 | 5 |
| Rando33 | 18 | 23 | 30 | 21 |
| Rando34 | 212 | 1235 | 1233 | 208 |
| Rando35 | 180 | 73 | 67 | 63 |
| Rando36 | 62 | 48 | 44 | 44 |
| Rando37 | 136 | 148 | 242 | 227 |
| Rando38 | 50 | 99 | 87 | 57 |
| Rando39 | 626 | 572 | 703 | 591 |
| Rando40 | 44 | 39 | 30 | 35 |
| Rando41 | 5 | 5 | 5 | 4 |
| Rando42 | 5 | 5 | 5 | 5 |





## References

[ABa98]: J. D. Andrews and L. M. Bartlett. "Efficient Basic Event Orderings For Binary Decision Diagrams". Proceedings of the Annual Reliability and Maintainability Symposium, Anaheim, Los Angeles, 1998, p61-68.
[ABa00]: J. D. Andrews and L. M. Bartlett. "Binary Decision Diagram Ordering Heuristics Based On Structural Importance". MMR (International Mathematical Models in Reliability) Conference, Bordeaux, July 2000.
[ADu99]: J. D. Andrews and S. J. Dunnett. "Event Tree Analysis Using Binary Decision Diagrams". Submitted to IEEE Transactions on Reliability, 1999.
[Ake78]: S. B. Akers. "Binary Decision Diagrams". IEEE Transactions on Computers, vol. C-27, 1978, p509-516.
[AMo93]: J. D. Andrews and T. R. Moss. Reliability and Risk Assessment. Longman Group, UK, 1993.
[BAn98]: L. M. Bartlett and J. D. Andrews . "Efficient Basic Event Ordering Schemes for Fault Tree Analysis". Advances in Reliability Techniques Symposium (ARTS), Manchester, 1998.
[BAn99]: L. M. Bartlett and J. D. Andrews. "Efficient Basic Event Ordering Schemes for Fault Tree Analysis". Quality and Reliability Engineering International, vol. 15, 1999, p95-101.
[BAn991]: L. M. Bartlett and J. D. Andrews. "Comparison of Variable Ordering Heuristics/Algorithms for Binary Decision Diagrams". Proceedings of the Safety and Reliability Society Symposium '99, Advances in Safety and Reliability, 12, June 1999, pl-15.
[BAn001]: L. M. Bartlett and J. D. Andrews. "Comparison of Two New Approaches to Variable Ordering for Binary Decision Diagrams". Submitted to Quality and Reliability Engineering International, 2000.
[BAn002]: L. M. Bartlett and J. D. Andrews. "Selecting An Ordering Heuristic For The Fault Tree to Binary Decision Diagram Conversion Process Using Neural Networks". Submitted to IEEE Transactions on Reliability, 2000.
[BAn003]: L. M. Bartlett and J. D. Andrews. "An Ordering Heuristic To Develop The Binary Decision Diagram Based on Structural Importance". Submitted to Reliability Engineering and System Safety, 2000.
[BBR97]: M. Bouissou, F. Bruyere and A. Rauzy. "BDD Based Fault-Tree Processing: A Comparison of Variable Ordering Heuristics". Proceedings of ESREL '97 Conference, August 1997.
[Bis95]: C. M. Bishop. Neural Networks for Pattern Recognition. Clarendon Press, Oxford, 1995
[Boo82]: L. B. Booker. "Intelligent Behaviour as an Adaptation of the Task Environment". Dissertation Abstracts International, 1982, 43(2), 469B.
[Bou96]: M. Bouissou. "An Ordering Heuristic for Building Binary Decision Diagrams from Fault Trees". Proceedings of Reliability and Maintainability Symposium, Las Vegas, 1996, p208-214.
[Bry86]: R. E. Bryant. "Graph-Based Algorithms for Boolean Function Manipulation". IEEE Transactions on Computers, vol. C-35, No. 8, 1986.
[Cul90]: Lord Cullen. The Public Enquiry into the Piper Alpha Disaster. Vol. 1 \& 2, Department of Energy, HMSO, 1990.
[DRa96]: Y. Dutuit and A. Rauzy. "A linear-time algorithm to find modules of fault trees". IEEE Transactions on Reliability, vol. 45, No. 3, September 1996, p 422-425.
[ESD96]: R. Eberhart, P. Simpson, and R. Dobbins. Computational Intelligence and PC Tools. AP, 1996.
[FFK88]: M. Fujita, H. Fujisawa, N. Kawato. "Evaluation and Improvements of Boolean Comparison Method Based on Binary Decision Diagrams". IEEE Transactions on Computer Aided Design (Conference), Nov 1988, p2-5.
[FSu90]: S. J. Friedman and K. J. Supowit. "Finding the optimal variable ordering for binary decision diagrams". IEEE Transactions On Computers, vol. 39, No.5, May 1990, p710-713.
[GAr95]: Group Aralia, "Computation of Prime Implicants of a Fault Tree Within Aralia". Proceedings of the European Safety and Reliability Association Conference, ESREL'95, 1995, p190-202.
[Gol97]: D. F. Goldberg. Genetic Algorithms in Optimisation and Machine Learning. Addison Wesely, 1997.
[Hea98]: J. D. Healy. "Basic Reliability Techniques". Annual Reliability and Maintainability Symposium Tutorial Notes, Jan. 1998.
[HKu81]: E. J. Henley and H. Kumamato. Reliability Engineering and Risk Assessment. Prentice-Hall, NY, 1981.
[Hol71]: J. H. Holland. "Genetic Algorithms and the Optimal Allocation of Trials". SIAM Journal of Computers, 1971, 2(2), p88-105.
[Lam75]: H. E. Lambert. "Measures of Importance of Events and Cut Sets in Fault Trees". Reliability and Fault Tree Analysis: Theoretical and Applied Aspects of System Reliability and Safety Assessment, SIAM, Philadelphia, 1975, p77-100.
[Lee80]: F. P. Lees. Loss Prevention in the Process Industries: Hazard Identification, Assessment and Control. Vol. 2, Butterworth and Co. (publishers) Ltd, 1980.
[LOK91]: Y. Lee, S. Oh, and M. Kim. "The effect of initial weights on premature saturation in back-propagation learning". International Joint Conference on Neural Networks, Vol. 1, 1991, p765-770.
[Mas93]: T. Masters. Practical Neural Network Recipes in C++. Academic Press, 1993.
[Men70]: E. Mendelson. Boolean algebra and switching circuits. Mc Graw-Hill, 1970.
[MIY91]: S. Minato, N. Ishiura and S. Yajima. "On variable ordering of binary decision diagrams for the application of the multi-level logic synthesis".

Proceedings of the European Conference on Design Automation, 1991, p50-54.
[Ne154]: R. J. Nelson. "Simplest Normal Truth Functions". J. Symbol. Logic, vol. 20, June 1954, p105-108.
[Nik99]: M. Nikolskaia. "Binary Decision Diagrams and Applications to Reliability Analysis". Doctoral Thesis, 1999, University of Bordeaux.
[Rau93]: A. Rauzy. "New algorithms for fault tree analysis", Reliability Engineering and System Safety, vol. 40, 1993, p203-211.
[Rau96]: A. Rauzy. "A Brief Introduction to Binary Decision Diagrams". European Journal of Automation, vol. 30, No. 8, 1996.
[SA196]: R. M. Sinnamon and J. D. Andrews. "Quantitative Fault Tree Analysis Using Binary Decision Diagrams". European Journal of Automation, vol. 30, No. 8, 1996.
[SA296]: R. M. Sinnamon and J. D. Andrews. "Improved Efficiency in Qualitative Fault Tree Analysis". Advances in Reliability Technology Symposium, Manchester, 1996.
[SA396]: R. M. Sinnamon and J. D. Andrews. "Fault Tree Analysis and Binary Decision Diagrams". Proceedings of the Reliability and Maintainability Symposium, Las Vegas, January 1996.
[Sch89]: W. G. Schneeweiss. Boolean Functions with Engineering Applications and Computer Programs. Springer-Verlag, Berlin, 1989.
[Sin96]: R. M. Sinnamon. "Binary Decision Diagrams for Fault Tree Analysis". Doctoral Thesis, 1996, Loughborough University.
[Ves70]: W. E. Vesely. "A Time Dependent Methodology for Fault Tree Evaluation". Nuclear Design and Engineering, vol. 13, 1970, p337-360.
[VMR88]: T. P. Vogl, J. K. Mangis, A. K. Rigler, W. T. Zink, and D. L. Alkon. "Accelerating the convergence of the Back-propagation method". Biological Cybernetics, 1988, 59, p257-263.
[Wat61]: H. A. Watson. Launch Control Safety Study, Vol. 1, Section VII, Bell Labs, Murray Hill, NJ, 1961.
[Yaz89]: M. Yazdani. "Building an expert system". In Expert Systems: Principles and Case Studies, Ed. 2, R. Forsyth, London: Chapman and Hall, 1989.

