

Loughborough University Institutional Repository

System failure modelling using binary decision diagrams

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

Additional Information:

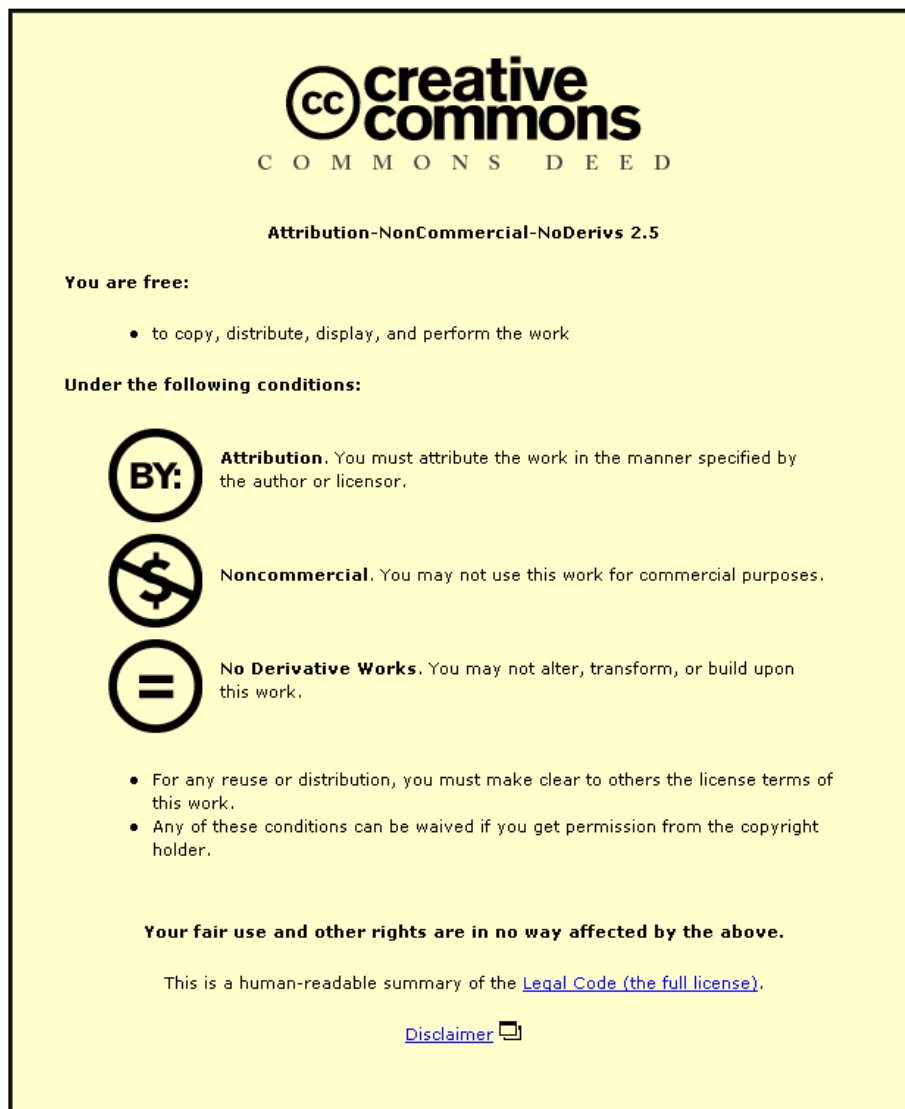
- A Doctoral Thesis. Submitted in partial fulfilment of the requirements for the award of Doctor of Philosophy of Loughborough University.

Metadata Record: <https://dspace.lboro.ac.uk/2134/12233>

Publisher: © Rasa Remenyte-Prescott

Please cite the published version.

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



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



University Library

Author/Filing Title REMENYTE - PRESCOTT

.....
Class Mark .. T

Please note that fines are charged on ALL
overdue items.

--	--	--

0403604389






SYSTEM FAILURE MODELLING USING BINARY DECISION DIAGRAMS

By
Rasa Remenyte-Prescott

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

June, 2007

©by Rasa Remenyte-Prescott, 2007

 Loughborough University Pulsator Library
Date <i>Nov 108</i>
Class <i>T</i>
Acc No <i>0403604389</i>

Abstract

The aim of this thesis is to develop the Binary Decision Diagram method for the analysis of coherent and non-coherent fault trees. At present the well-known technique for converting fault trees to BDDs is used. Difficulties appear when the ordering scheme for basic events needs to be chosen, because it can have a crucial effect on the size of a BDD. An alternative method for constructing BDDs from fault trees which addresses these difficulties has been proposed.

The Binary Decision Diagram method provides an accurate and efficient tool for analysing coherent and non-coherent fault trees. The method is used for the qualitative and quantitative analyses and it is a lot faster and more efficient than the conventional techniques of Fault Tree Analysis. The simplification techniques of fault trees prior to the BDD conversion have been applied and the method for the qualitative analysis of BDDs for coherent and non-coherent fault trees has been developed.

A new method for the qualitative analysis of non-coherent fault trees has been proposed. An analysis of the efficiency has been carried out, comparing the proposed method with the other existing methods for calculating prime implicant sets. The main advantages and disadvantages of the methods have been identified.

The combined method of fault tree simplification and the BDD approach has been applied to Phased Missions. This application contains coherent and non-coherent fault trees. Methods to perform their simplification, conversion to BDDs, minimal cut sets/prime implicant sets calculation, and the mission unreliability evaluation have been produced.

Acknowledgments

First of all, I would like to thank my supervisor Professor John Andrews for giving me the chance to conduct this research, for his guidance and for always being friendly and helpful.

I am grateful to my family and friends here and back at home for their support and encouragement throughout this research.

I also would like to thank the people in the Risk and Reliability research group for the possibility to share ideas and keeping up my motivation.

Finally, thanks to Darren, my husband and colleague, for the endless inspiration, continuous understanding and invaluable friendship.

Contents

1	Introduction	1
1.1	Risk and Reliability Assessment	1
1.2	Fault Tree Analysis	2
1.3	Binary Decision Diagrams	3
1.4	Research Objectives	4
2	Fault Tree Analysis	5
2.1	Introduction	5
2.2	Construction of Fault Tree	5
2.3	Qualitative Analysis	7
2.3.1	Boolean Laws of Algebra	9
2.3.2	Example - Obtaining the Minimal Cut Sets	10
2.4	Quantitative Analysis	11
2.4.1	Structure Functions	12
2.4.2	Shannon's Theorem	13
2.4.3	General Method for the Calculation of the Top Event Probability	14
2.4.3.1	Upper and lower bounds for system unavailability	15
2.4.3.2	Minimal cut set upper bound	15
2.4.4	Top Event Frequency	16
2.4.4.1	Approximation for the system unconditional failure intensity	17
2.4.4.2	Expected number of system failures	17
2.4.5	Importance Measures	17
2.4.5.1	Deterministic measures	18
2.4.5.2	Probabilistic measures of system unavailability	18
2.5	Simplification of the Fault Tree Structure	19
2.5.1	Fault Tree Reduction	20

2 5 1 1	Introduction	20
2 5.1 1.1	Reduction technique	20
2 5 1 2	Worked example of the reduction technique . . .	21
2.5 1 2 1	Contraction	22
2.5 1 2 2	Factorisation	22
2 5 1 2 3	Extraction	24
2 5.1 2.4	Absorption	24
2.5 1 3	Reduced fault tree	26
2.5.2	Fault Tree Modularisation	28
2 5 2 1	Rauzy's algorithm	28
2 6	Summary	31
3	Binary Decision Diagrams	32
3 1	Introduction	32
3.2	Properties of the BDD	32
3.2.1	Formation of the BDD Using If-Then-Else	34
3.3	Reduction	37
3 4	Modularisation	37
3.5	Qualitative Analysis	39
3 5.1	Incorporating Complex Events and Modules into the Analysis	42
3 6	Quantitative Analysis	44
3 6.1	System Unavailability	44
3.6 2	System Unconditional Failure Intensity	45
3 6 3	Worked Example	47
3.6 4	Incorporating Complex Events and Modules into the Analysis	48
3 6.4.1	Overview of the calculation procedure	49
3.6.4 2	Unavailability of complex and modular events . . .	50
3.6 4 3	Criticality of basic events within complex events	50
3 6 4 4	Criticality of basic events within modules . . .	53
3 6 5	The Algorithm for Incorporating Complex Events and Mod- ules into the Analysis	54
3.7	Summary	55
4	Component Connection Method for Building BDDs	56
4.1	Introduction	56
4 2	Connection Process	56
4 3	Rules of Simplification	62

4.4	Properties of the BDD Using the Component Connection Method	65
4.5	Measures of Efficiency	66
4.6	Selection Schemes in the Component Connection Method	68
4.6.1	Order of Basic Events Selection	68
4.6.1.1	Order as listed process	68
4.6.1.2	Defined ordering	69
4.6.1.2.1	Neighbourhood ordering schemes	70
4.6.1.2.2	Weighted ordering schemes	74
4.6.2	Order of Inputs Selection for the Top-Down Technique	81
4.6.2.1	Order as listed process	81
4.6.2.2	Ordering event inputs before gate inputs	81
4.6.2.3	Ordering gate inputs according to the number of their event inputs	81
4.6.3	Order of BDDs Selection for the Bottom-Up Technique	84
4.6.3.1	Order as listed process	84
4.6.3.2	According to a defined ordering of basic events	85
4.6.3.3	According to the number of available branches	87
4.6.4	Results	88
4.6.4.1	Top-down technique	90
4.6.4.1.1	Summary results	90
4.6.4.1.2	Variable ordering for the top-down approach	90
4.6.4.2	Bottom-up technique	92
4.6.4.2.1	Summary results	92
4.6.4.2.2	Variable ordering for the bottom-up approach	92
4.6.4.3	Comparison of top-down and bottom-up technique	93
4.6.4.4	Bottom-up technique, chosen trials	95
4.6.4.4.1	Introduction	95
4.6.4.4.2	First trial	96
4.6.4.4.3	Second trial	97
4.6.4.4.4	Third trial	98
4.6.4.4.5	Comparison of the bottom-up techniques	100
4.7	Comparison Between the Component Connection and the ite method	101
4.8	Sub-node Sharing	104
4.8.1	Presentation of the Sub-node Sharing in Component Connection Method	104

4 8 2	Comparison Between the <i>ite</i> and the Component Connection Method Using the Sub-node Sharing	107
4 9	Hybrid method	110
4.9.1	Presentation of the Method	111
4 9.2	Comparison Between the <i>ite</i> and the Hybrid Method . . .	116
4 9 2.1	Comparison between the Basic Hybrid method and the <i>ite</i> technique	117
4 9.2 2	Comparison between the Basic Hybrid Method and the Advanced Hybrid method	119
4 10	Summary	122
5	Non-coherent Systems	124
5.1	Introduction	124
5.2	Fault Tree Analysis of Non-coherent Fault Trees	124
5 2 1	Introduction	124
5.2.2	The Use of NOT Logic	125
5 2 3	Qualitative Analysis	128
5 2 3 1	Calculation of prime implicants	128
5 2 4	Quantitative Analysis	130
5 2 4.1	Calculating the system unavailability	130
5.2 4.2	Calculating the unconditional failure intensity . . .	131
5 2 4.3	Importance measures	132
5 3	Simplification Process of Non-coherent Fault Trees	134
5 3 1	Faunet Reduction in the Non-coherent Fault Tree Case . . .	134
5 3 2	Linear Modularisation in the Non-coherent FT Case . .	138
5.4	Summary	139
6	The BDD Method for the Analysis of Non-coherent Fault Trees	140
6 1	Introduction	140
6 2	Computing the SFBDD in Non-coherent Case	141
6 3	Qualitative Analysis	142
6 3 1	Coherent Approximation	143
6 3 2	Ternary Decision Diagram Method	143
6 3 2.1	Computing the TDD	145
6 3 2 2	Minimising the TDD	148
6 3 3	Established methods	151
6 3 3 1	Rauzy and Dutuit Meta-products BDD	151

6 3 3 1 1	MPPI algorithm	152
6 3.3.2	Zero-suppressed BDD Method	156
6.3 3 2 1	Presentation of a ZBDD	156
6 3 3.2 2	Decomposition rule	157
6 3 3 2 3	Worked example	158
6.3 3 3	Labelled Variable Method	160
6 3 3 3.1	Classification of variables	160
6 3 3.3.2	Construction of the L-BDD	161
6 3.3.3.3	Determination of prime implicants	163
6.3 3.3 4	Simplification of L-BDD	164
6 3.4	Comparison of the Four Methods	166
6 3.4.1	TDD method results	168
6 3 4 1 1	Variable ordering for the TDD method	169
6.3.4.2	Meta-products method results	170
6 3 4 2 1	Variable ordering for the MPPI method	171
6.3 4 3	ZBDD method results	172
6 3.4.3.1	Variable ordering for the ZBDD method	173
6.3.4 4	L-BDD method results	174
6 3 4.4 1	Variable ordering for the L-BDD method	175
6.3 5	Overall Variable Ordering for the Qualitative Analysis of Non-coherent Fault Trees	176
6.4	Quantitative Analysis	180
6.4.1	Introduction	180
6 4 2	System Unavailability	180
6 4 3	Importance Measures	181
6.4.3 1	Birnbaum's measure of failure and repair importance	181
6 4 3.1.1	The SFBDD method	182
6 4.3 1.2	The L-BDD method	183
6 4.3 1.3	The TDD method	185
6 5	Hybrid Method in the Non-coherent Fault Tree Case	187
6 6	Summary	189
7	Application of Proposed Methods in Phased Mission Analysis	193
7.1	Introduction	193
7.2	Fault Tree Method for Phased Mission Analysis	194
7.3	Phase FT simplification	198

7.4	Binary Decision Diagram Analysis for Phased Missions	204
7.5	Summary	207
8	Computer Implementation of BDD Method	208
8.1	Introduction	208
8.2	Overview	208
8.3	Established Modules	209
8.3.1	Data Input Module	209
8.3.2	Simplification Module	211
8.3.3	BDD Conversion Module	212
8.3.4	Quantitative Analysis Module	214
8.3.5	Qualitative Analysis Module	215
8.3.5.1	Minimisation module	215
8.3.5.2	Calculation of minimal cut sets	217
8.3.6	Component Connection Method	219
8.3.7	Hybrid Method	223
8.3.8	Non-coherent FT Input Format and Conversion to SFBDD	224
8.3.9	Non-coherent Fault Tree Conversion to BDDs for the Qualitative Analysis	225
9	Conclusions and Future Work	228
9.1	Summary of Work	228
9.1.1	Qualitative Analysis of Coherent Systems	228
9.1.2	Component Connection Method	229
9.1.3	Qualitative Analysis of Non-coherent Systems	232
9.1.4	Application of Proposed Methods in Phased Mission Analysis	235
9.2	Conclusions	235
9.3	Future Work	236
9.3.1	Component Connection Method	236
9.3.2	Application of Proposed Methods in Phased Mission Analysis	237
A	Results	242

Nomenclature

- $A(t)$ – Availability function
 C – Consequence of an event
 $F(t)$ – System unreliability function
 $G_i(\mathbf{q}(t))$ – Criticality function for event i (Birnbaum's measure of importance)
 $G_i^F(\mathbf{q}(t))$ – Component i failure criticality
 $G_i^R(\mathbf{q}(t))$ – Component i repair criticality
 I_i – Measure of importance for event i
 $I_{C_i}^F$ – Measure of failure importance for component or cut set i
 $I_{C_i}^R$ – Measure of repair importance for component or cut set i
 K_i – Minimal cut set i
 n – Number of components in a system
 N – Number of minimal cut sets (or prime implicant sets)
 N_p – Number of basic events in a prime implicant set
 $P2$ – Meta-products structure encoding prime implicant sets for which x is irrelevant
 $P1$ – Meta-products structure encoding prime implicant sets for which x is failure relevant
 $P0$ – Meta-products structure encoding prime implicant sets for which x is repair relevant
 P – Probability
 $P(K_i)$ – Probability of existence of minimal cut set i
 $P[F]$ – Probability value of node F in a BDD
 p_{x_i} – Probability of basic events encoded in the path from the root vertex to the current node encoding x_i
 $po_{x_i}^1(\mathbf{q}(t))$ – Probability of the path section from the 1 branch of a node encoding x_i to a terminal vertex 1 in the BDD
 $po_{x_i}^0(\mathbf{q}(t))$ – Probability of the path section from the 0 branch of a node encoding x_i to a terminal vertex 1 in the BDD

- $po_{x_i}^{1,c}(\mathbf{q}(t))$ – Probability of the path section from the 1 branch of a node encoding x_i to a terminal vertex 1 in the BDD via only 1 or 0 branches of non-terminal nodes (excluding the probability of x_i)
- $po_{x_i}^{0,c}(\mathbf{q}(t))$ – Probability of the path section from the 0 branch of a node encoding x_i to a terminal vertex 1 in the BDD via only 1 or 0 branches of non-terminal nodes (excluding the probability of x_i)
- $po_{x_i}^c(\mathbf{q}(t))$ – Probability of the path section from the 'C' branch of a node encoding x_i to a terminal vertex 1 in the BDD via only 1 or 0 branches of non-terminal nodes (excluding the probability of x_i)
- $pr_{x_i}(\mathbf{q}(t))$ – Probability of the path section from the root vertex to the node encoding x_i in the BDD
- $Q(t)$ – System unavailability (failure probability)
- $q_i(t)$ – Component unavailability (failure probability)
- q^{x_j} – Probability of a literal contained in any of the prime implicant sets
- R – Risk
- $R(t)$ – Reliability function
- $T_{i=1}$ – Failure relevance or irrelevance of component i
- $T_{i=0}$ – Repair relevance or irrelevance of component i
- $T_{i=-}$ – Irrelevance of component i
- $W(t_0, t_1)$ – Expected number of failures during the interval (t_0, t_1)
- $w_{sys}(t)$ – System unconditional failure intensity
- $w_i(t)$ – Component unconditional failure intensity
- $w_{K_i}(t)$ – Unconditional failure intensity of minimal cut set i
- $v_i(t)$ – Unconditional repair intensity of component i
- X_c – Complex event
- x_i – Binary indicator variable for component states
- $Z(\mathbf{q}(t))$ – Probability of paths from the root vertex to a terminal vertex 1 that do not pass through a node encoding x_i

- $\lambda(t)$ – Conditional failure rate
- $\phi(\mathbf{x})$ – Structure function
- ϕ_i^F – Failure criticality function
- ϕ_i^R – Repair criticality function
- $\rho_i(\mathbf{x})$ – Binary indicator function for each minimal cut set
- θ_i – Occurrence of minimal cut set i

Chapter 1

Introduction

1.1 Risk and Reliability Assessment

Reliability engineering is a rapidly developing field and is becoming increasingly important in various industries and technologies. It provides those theoretical and practical tools which can specify, design and predict the probability and capability of components and systems to perform their required functions for the desired period without failure.

Reliability assessment techniques enable the calculation of the probability or frequency of system failure to be performed. There are several measures that can be used to quantify system failure, including system reliability, availability and failure intensity.

The *reliability* of a system, $R(t)$, is defined as the probability that the system operates without failure for a specified period of time under stated conditions. The *unreliability* of a system, $F(t)$, is defined as the probability that the system has failed at least once in the interval $[0, t)$ given that it was working at $t = 0$. Since both functions are probabilistic,

$$R(t) + F(t) = 1. \quad (1.1)$$

The system *availability*, $A(t)$, is defined as the fraction of total time that the system is able to perform its required function. The *unavailability*, $Q(t)$, is defined as the fraction of total time that the system has failed and is unable to perform its task. Again, the relationship between the two functions is defined as:

$$A(t) + Q(t) = 1. \quad (1.2)$$

The *unconditional failure intensity*, $w(t)$, is the rate that a system fails per unit time at time t given that it was working at time $t = 0$. The rate that a system fails per unit time at time t given that it was working at time t and at time $t = 0$ is defined as the *conditional failure rate*, $\lambda(t)$. The difference between $w(t)$ and $\lambda(t)$ is that $w(t)$ is based on the whole population, whereas $\lambda(t)$ considers only those components that are working at time t .

For major hazard assessments risk is generally defined as the product of the consequences of a particular incident, C , and the probability over a time period or frequency of its occurrence, P

$$R = C \cdot P \quad (1.3)$$

The risk can be reduced by minimising the consequences of the incident (C) or by reducing the probability or frequency of its occurrence (P) Reliability assessment techniques evaluating the frequencies of incidents have been developed and the most widely used is Fault Tree Analysis, which is considered later in this chapter

After risks are identified and evaluated it must be judged if they are 'acceptable' or whether the risk is too high and some modifications to the design of the system should be made in order to improve the system reliability. Although risks can be decreased by spending money it is not possible to avoid them entirely. The difficulty faced by safety assessors is to convince regulators that the safety of a system is at an acceptable level

1.2 Fault Tree Analysis

Fault Tree Analysis was developed by H.A. Watson in the early 1960's. This is a deductive procedure for determining the causes of a particular system failure mode and the probability and frequency with which it could occur. The fault tree diagram represents the combinations of component failures and human errors that could combine to cause system failure. 'Top event' describes the system failure mode and branches below this event describe its causes. The events are redefined in terms of their causes, until each branch ends with a basic event

Kinetic Tree Theory, the technique for performing the quantitative analysis of fault trees, was presented in the early 1970's by Vesely [1]. Various system reliabil-

ity parameters, such as probability of top event existence, frequency of top event occurrence and component importance measures can be calculated. They are used to determine whether the risk of system failure is sufficiently small and therefore whether or not the system meets the required safety standards.

For large fault trees the analysis can become computationally intensive and can require the use of approximations. This is the disadvantage of the conventional method which leads to inaccuracies in the calculations. This issue has led to the development of a new method for analysing fault trees, known as the Binary Decision Diagram technique.

1.3 Binary Decision Diagrams

The Binary Decision Diagram (BDD) technique for Fault Tree Analysis was presented by Rauzy [2]. This method converts a fault tree to a BDD, which encodes the logic function of the fault tree. Conventional FTA techniques can be computationally intensive and sometimes inaccurate. The BDD method is an accurate and efficient method for system reliability assessment. Firstly, the BDD method provides an accurate quantification process because no approximations are required. Secondly the minimal cut sets are not required for the quantification process, therefore, it makes the BDD method efficient. The qualitative analysis can be performed and minimal cut sets obtained if required.

The size of the BDD depends upon the order in which the basic events are considered during the construction process. A problem can occur if the choice of the ordering scheme results in a time-consuming construction process and a large BDD. No one ordering scheme will produce the smallest possible BDD from every fault tree.

When NOT logic is included in the fault tree structure it becomes non-coherent and its analysis using the conventional techniques to produce prime implicant sets becomes even more problematic. BDDs offer advantages over the conventional methods for this class of fault trees. However, alternative techniques for converting fault trees to BDDs can improve the efficiency of the approach still further.

1.4 Research Objectives

The aim of this research is to improve the techniques which produce BDDs from fault trees and conduct the analysis. Four aspects are taken into consideration

In the previous research fault trees were reduced (applying reduction and modularisation) prior to BDD conversion. This can be used in fault tree quantification. The qualitative analysis using BDDs is extended to investigate the use of these reduced trees. The goal of this aspect is to perform the full BDD analysis and to obtain minimal cut sets in terms of basic events from the reduced fault trees

The second aspect explores a new fault tree to BDD conversion technique, as an alternative method to the well-known ite technique. The new method is based on connecting previously generated BDD sections. This technique is presented for the analysis of coherent fault trees. Different efficiency measures are used to investigate the optimum connection technique.

The third aspect proposes a new method for converting non-coherent fault trees to BDDs. The new approach utilises a structure where each node contains three branches. This part of research contains the comparison of the proposed technique and the established construction methods of BDDs and the mechanisms of calculating prime implicant sets. It also incorporates some of the methods of the coherent fault trees while seeking for a better efficiency.

The final part of the research covers some applications of the presented methods in system reliability assessment for Phased Missions.

Chapter 2

Fault Tree Analysis

2.1 Introduction

Fault Tree Analysis is the most widely used tool in safety and reliability assessment. It is a deductive technique for analysing the causal relationships between component failures and system failure. The fault tree itself provides a visual representation of the structure of the system by expressing a particular system failure mode in terms of component failures and human errors. It produces a complete description of the causes of system failure, which is important during the design stages of a system, as it allows weak areas to be identified and correction by re-design.

2.2 Construction of Fault Tree

The system failure mode to be considered is termed the top event and the fault tree is developed in branches below this event showing its causes. In this way events represented in the tree are continually redefined in terms of lower-resolution events [3]. This development process is terminated when component failure events, termed basic events, are encountered. These basic events can be component failures or human errors. Each fault tree considers only one of the many possible system failure modes and therefore more than one fault tree may be constructed during the assessment of any system. For example, a typical top event may be a hazardous event such as explosion or safety system unavailable, a basic event represents component failures such as pump failure to start or human errors such as operator failure to respond.

The fault tree diagram contains two basic elements, gates and events. Events are categorised as either intermediate or basic. Intermediate events, which can be further developed in terms of other events, are represented by rectangles in the tree, basic events cannot be developed any further and are represented by circles. These symbols are shown in Table 2.1. Gates allow or inhibit the passage of fault



Event symbol	Meaning of symbol
	Intermediate event further developed by a gate
	Basic event

Table 2.1 Event symbols

logic up the tree and show the relationships between the events needed for the occurrence of a higher event. The three fundamental types of gates used in fault trees are the 'AND' gate, 'OR' gate and 'NOT' gate. These gates combine events in the same way as the Boolean operations of 'intersection', 'union' and 'complementation'. Another frequently used gate is the k/n vote gate. This allows the flow of logic through the tree if at least k out of n inputs occur. The symbols for the gates and their causal relations are shown in Table 2.2. A system whose failure modes are expressed only in terms of component failures is known as a 'coherent' system. A coherent fault tree will contain only 'AND' and 'OR' logic. If the failure modes of a system are expressed in terms of both component failures and successes it is referred to as a 'non-coherent' system. In addition to the gates used in coherent fault trees non-coherent fault trees also contain 'NOT' logic. The work within this thesis will consider both types of fault trees.

Once a fault tree has been constructed for a system two types of analysis can be performed, qualitative and quantitative.

- Qualitative analysis involves obtaining the smallest sets of events that combine to cause system failure. In coherent fault trees these are called *minimal cut sets*; in non-coherent trees they are called *prime implicants*.
- Quantitative analysis contains calculating the system failure parameters (the top event probability and frequency) and event importance measures.



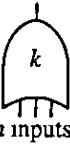


Gate symbol	Gate name	Causal relation
	AND gate	Output event occurs if all input events occur simultaneously
	OR gate	Output event occurs if at least one of the input events occurs
	k/n vote gate	Output event occurs if at least k -out-of- n input events occur
	Exclusive OR	Output event occurs if only one of the input events occur
	NOT gate	Output event occurs if the input event does not occur

Table 2 2: Common gate types and corresponding symbols

2.3 Qualitative Analysis

Each unique way that system failure can occur is a system failure mode and will involve the failure of individual components or combinations of components. To analyse a system and to eliminate the most likely causes of failure first requires that each failure mode is identified. One way to identify the system failure modes is to carry out a logical analysis on the system fault tree. The system failure modes are defined by the concepts of cut sets or minimal cut sets which are defined below

A cut set is a collection of basic events such that if they all occur the top event also occurs, i.e. if all components fail the system also fails.

For industrial engineering systems there is generally a very large number of cut sets each of which can contain many component failure events. However, only lists of component failure modes are interesting, which are both necessary and sufficient to produce system failure. For example, $\{a, b, c\}$ may be a cut set and the failure of these three components will guarantee system failure. But if the failure of a and b alone produce system failure this means that the state of component c is irrelevant and the system will fail whether c fails or not. This leads to the definition of a

minimal cut set:

A minimal cut set is the smallest combination of basic events, such that if any of the basic events is removed from the set the top event will not occur, i.e. if any of the components in the set works the system will not fail.

Two fault trees drawn using different approaches are logically equivalent if they produce identical minimal cut sets. The order of a minimal cut set is the number of components within the set. The first-order minimal cut sets represent single failures which cause the top event. Two-component minimal cut sets (second order) represent double failures which together will cause the top event to occur. In general the lower-order cut sets contribute most to system failure and it is with the elimination of these that effort should be concentrated in order to improve system performance.

If 'NOT' logic is used or implied the combinations of basic events that cause the top event are called implicants. Minimal sets of implicants are called prime implicants.

The minimal cut set expression for the top event (*Top*) can be written in the form

$$Top = K_1 + K_2 + \dots + K_N, \quad (2.1)$$

where $K_i, i = 1, \dots, N$ are the minimal cut sets (+ represents logical 'OR'). Each minimal cut set consists of a combination of component failures and hence the general k -component cut set can be expressed as:

$$K_i = x_1 \cdot x_2 \cdot \dots \cdot x_k, \quad (2.2)$$

where $x_i, i = 1, \dots, k$ are basic component failures on the tree (\cdot represents logical 'AND'). In other words, the top event must be transformed to a sum-of-products form.

To determine the minimal cut sets of a fault tree either a top-down or a bottom-up approach can be used, depending on which end of the tree is used to initiate the expansion process. The top-down procedure is described below and illustrated with the use of an example. The process starts with the top event, which is expanded

by substituting in the Boolean events appearing lower down in the tree and simplifying until the remaining expression has only basic component failures. Usually when analysing real fault trees which contain large numbers of repeated events the expression obtained may not be minimal. Redundancies must be removed from the expression using the laws of Boolean algebra to allow the extraction of the minimal cut sets. The laws are shown in the next section.

2.3.1 Boolean Laws of Algebra

1. Commutative Laws:

$$A + B = B + A \quad (2.3)$$

$$A \cdot B = B \cdot A \quad (2.4)$$

2. Associative Laws:

$$(A + B) + C = A + (B + C) \quad (2.5)$$

$$(A \cdot B) \cdot C = A \cdot (B \cdot C) \quad (2.6)$$

3. Distributive Laws:

$$A + (B \cdot C) = (A + B) \cdot (A + C) \quad (2.7)$$

$$A \cdot (B + C) = (A \cdot B) + (A \cdot C) \quad (2.8)$$

4. Identities:

$$\begin{aligned} A + 0 &= A & A \cdot 0 &= 0 \\ A \cdot 1 &= A & A + 1 &= 1 \end{aligned} \quad (2.9)$$

5. Idempotent Laws:

$$A + A = A \text{ (removes repeated cut sets)} \quad (2.10)$$

$$A \cdot A = A \text{ (removes repeated events within each cut set)} \quad (2.11)$$

6. Absorption Laws:

$$A + A \cdot B = A \text{ (removes non-minimal cut sets)} \quad (2.12)$$

$$A \cdot (A + B) = A \quad (2.13)$$

7. Complementation

$$\bar{A} = 1 - A \quad (2.14)$$

$$A \cdot \bar{A} = 0 \quad (2.15)$$

$$\overline{(\bar{A})} = A \quad (2.16)$$

8 De Morgan's Laws

$$\overline{(A + B)} = \bar{A} \cdot \bar{B} \quad (2.17)$$

$$\overline{(A \cdot B)} = \bar{A} + \bar{B} \quad (2.18)$$

2.3.2 Example - Obtaining the Minimal Cut Sets

The top-down approach for calculating the minimal cut sets is demonstrated using the example fault tree shown in Figure 2.1. Starting with the top event (*Top*) it is

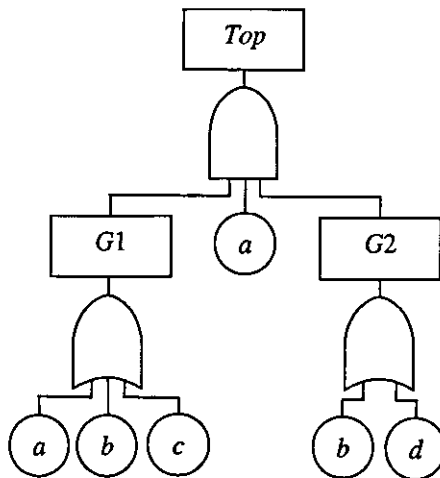


Figure 2.1 Example fault tree

an 'AND' gate with three inputs *G1*, *a* and *G2*. It can therefore be expressed as a product of these inputs:

$$Top = G1 \cdot a \cdot G2. \quad (2.19)$$

As *G1* is an 'OR' gate, made up of three events, *a*, *b* and *c*, it can be written as:

$$G1 = a + b + c \quad (2.20)$$

Substituting this into *Top* gives:

$$Top = (a + b + c) \cdot a \cdot G2 \quad (2.21)$$

Similarly, $G2$ can be written as the 'sum' of b and d , so Top becomes

$$Top = (a + b + c) \cdot a \cdot (b + d) \quad (2.22)$$

The expression now contains only basic events, so is expanded to give

$$\begin{aligned} Top &= a \cdot a \cdot b + a \cdot a \cdot d + a \cdot b \cdot b + a \cdot b \cdot d + a \cdot c \cdot b + a \cdot c \cdot d \quad (2.23) \\ &= a \cdot b + a \cdot d + a \cdot b + a \cdot b \cdot d + a \cdot c \cdot b + a \cdot c \cdot d \\ &\quad (\text{as } a \cdot a = a \text{ and } b \cdot b = b), \end{aligned}$$

which gives the cut sets of the fault tree. These are simply the cut sets expressed in sum-of-products form. Redundancies can then be removed using the idempotent and absorption laws:

$$Top = a \cdot b + a \cdot d \quad (2.24)$$

This is the minimal disjunctive form of the logic equation each term of which is a minimal cut set. For this fault tree there are two minimal cut sets, both of order two (i.e. they each contain two basic events). These are $\{a, b\}$ and $\{a, d\}$.

A complex system may produce thousands of minimal cut sets. Although the algorithm is not complex the process can be very time-consuming. For this reason approximations are often implemented which removes the cut sets above a certain order (for example, above order three) during the calculation process. This approximation reduces the number of computations and the time taken for the analysis. However, this obviously leads to a reduction in the accuracy of the minimal cut sets and therefore in the resulting quantitative analysis for which minimal cut sets are required.

2.4 Quantitative Analysis

Quantitative analysis of the fault tree allows the calculation of a number of parameters which are used to assess the system. The top event probability and frequency are used together with the expected number of occurrences of the top event and event importance measures to gain a full understanding of the system.

The methods for fault tree quantification are known as Kinetic Tree Theory [1] which is a time dependent methodology for system evaluation. This technique forms the basis of the approach used in the majority of commercial Fault Tree Analysis packages.

2.4.1 Structure Functions

The structure function is a binary function taking the following values

$$\phi(\mathbf{x}) = \begin{cases} 1 & \text{if the system is failed,} \\ 0 & \text{if the system is working,} \end{cases} \quad (2.25)$$

where $\mathbf{x} = (x_1, x_2, \dots, x_n)$ is an indicator vector showing the status (working or failed) of each component

For each system component, j , the binary indicator variable, x_j , is presented

$$x_j = \begin{cases} 1 & \text{if component } j \text{ is failed,} \\ 0 & \text{if component } j \text{ is working.} \end{cases} \quad (2.26)$$

The structure function for the top event of a fault tree shows the system state in relation to its components and is given by:

$$\phi(\mathbf{x}) = 1 - \prod_{i=1}^N (1 - \rho_i(\mathbf{x})), \quad (2.27)$$

where $\rho_i(\mathbf{x})$ is the binary indicator function for each minimal cut set $K_i, i = 1 \dots N$:

$$\rho_i(\mathbf{x}) = \prod_{j \in K_i} x_j \text{ such that } \rho_i = \begin{cases} 1 & \text{if cut set } K_i \text{ exists,} \\ 0 & \text{if cut set } K_i \text{ does not exist} \end{cases} \quad (2.28)$$

For the fault tree shown in Figure 2.1, which has minimal cut sets $K_1 = \{a, b\}$ and $K_2 = \{a, d\}$, the structure function is given by:

$$\phi(\mathbf{x}) = 1 - (1 - x_a x_b)(1 - x_a x_d). \quad (2.29)$$

The probability of the top event is given by the expected value of the structure function:

$$Q(t) = E[\phi(\mathbf{x})] \quad (2.30)$$

If each minimal cut set is independent (i.e. no event appears in more than one cut set) then it is also true that:

$$E[\phi(\mathbf{x})] = \phi[E(\mathbf{x})]. \quad (2.31)$$

Obtaining the expected value of the structure function for independent minimal cut sets would simply be a matter of substituting the probability of failure of each component into the structure function and calculating the result. However, the

minimal cut sets are not usually independent, and so in this case a full expansion of the structure function and the reduction of the indicator variables (i.e. $x_i = x_i^n$) must be undertaken.

Applying this to the structure function for the example fault tree in Equation 2.29, gives:

$$\begin{aligned}\phi(\mathbf{x}) &= 1 - (1 - x_a x_d - x_a x_b + x_a x_a x_b x_d) \\ &= x_a x_d + x_a x_b - x_a x_b x_d,\end{aligned}\tag{2.32}$$

using expansion and reduction. The probability of the top event is then given by the expected value of this structure function.

$$Q(t) = P(a) \cdot P(d) + P(a) \cdot P(b) - P(a) \cdot P(b) \cdot P(d)\tag{2.33}$$

A more efficient method of implementing this uses Shannon's Theorem.

2.4.2 Shannon's Theorem

Shannon's Theorem can be expressed as follows. A Boolean function $f(\mathbf{x})$ can be written as:

$$f(\mathbf{x}) = x_i \cdot f(1_i, \mathbf{x}) + \bar{x}_i \cdot f(0_i, \mathbf{x}),\tag{2.34}$$

where

$$\bar{x}_i = 1 - x_i,\tag{2.35}$$

$$f(1_i, \mathbf{x}) = f(x_1, \dots, x_{i-1}, 1, x_{i+1}, \dots, x_n),\tag{2.36}$$

$$f(0_i, \mathbf{x}) = f(x_1, \dots, x_{i-1}, 0, x_{i+1}, \dots, x_n).\tag{2.37}$$

$f(1_i, \mathbf{x})$ and $f(0_i, \mathbf{x})$ are known as the residues of $f(\mathbf{x})$ with respect to x_i .

The structure function is pivoted around the most repeated variable using Shannon's expansion. This is continued until no repeated variables exist in the residues.

Shannon's theorem can be applied to the structure function given in Equation 2.29. Pivoting around the repeated variable, a , gives

$$\begin{aligned}\phi(\mathbf{x}) &= x_a[1 - (1 - x_b)(1 - x_d)] + (1 - x_a)[0] \\ &= x_a(1 - (1 - x_b)(1 - x_d))\end{aligned}\tag{2.38}$$

The probability of the top event is therefore given by

$$\begin{aligned} Q(t) &= E(\phi(\mathbf{x})) \\ &= P(a)(1 - (1 - P(b))(1 - P(d))). \end{aligned} \quad (2.39)$$

Expanding this gives exactly the same result shown in Equation 2.33.

2.4.3 General Method for the Calculation of the Top Event Probability

This general method of calculating the top event probability (i.e. the system unavailability) uses the minimal cut sets obtained from the qualitative analysis [4]. This method can be used whether or not the fault tree contains repeated events

The top event occurs if at least one minimal cut set exists, therefore for a fault tree that has N minimal cut sets, K_i , $Q(t)$ is given by:

$$Q(t) = P\left(\bigcup_{i=1}^N K_i\right). \quad (2.40)$$

Expanding gives.

$$\begin{aligned} Q(t) &= \sum_{i=1}^N P(K_i) - \sum_{i=2}^N \sum_{j=1}^{i-1} P(K_i \cap K_j) + \\ &\quad + (-1)^{N-1} P(K_1 \cap K_2 \cap \dots \cap K_N), \end{aligned} \quad (2.41)$$

where $P(K_i)$ is the probability of the existence of minimal cut set i .

This expansion is known as the inclusion-exclusion expansion and generates the exact probability of the top event existence. Consider the example fault tree shown in Figure 2.1, which has minimal cut sets $K_1 = \{a, b\}$ and $K_2 = \{a, d\}$. Equation 2.41 gives the top event probability as:

$$\begin{aligned} Q(t) &= P(K_1) + P(K_2) - P(K_1 \cap K_2) \\ &= P(a) \cdot P(b) + P(a) \cdot P(d) - P(a) \cdot P(b) \cdot P(d), \end{aligned} \quad (2.42)$$

which is identical to the expression calculated in Equation 2.33.

It is usual to have fault trees for engineering systems which result in tens of thousands of minimal cut sets. Therefore it is impractical in these situations to calculate all terms in the complete expansion. For this reason the calculation is simplified by the use of approximations.

2.4.3.1 Upper and lower bounds for system unavailability

Truncation of the series in Equation 2.41 at an even-numbered term gives a lower bound for the top event probability, truncation at an odd-numbered term gives an upper bound for the top event probability

$$\sum_{i=1}^N P(K_i) - \sum_{i=2}^N \sum_{j=1}^{i-1} P(K_i \cap K_j) \leq Q(t) \leq \sum_{i=1}^N P(K_i). \quad (2.43)$$

The upper bound is known as the Rare Event Approximation, $P_{RE}(Top)$, as it is accurate if the component failure events are rare

$$P_{RE}(Top) = \sum_{i=1}^N P(K_i) \quad (2.44)$$

2.4.3.2 Minimal cut set upper bound

A more accurate upper bound is Minimal Cut Set Upper Bound, $P_{MCSUB}(Top)$. This is derived as follows

$$\begin{aligned} P(\text{system failure}) &= P(\text{at least one minimal cut set exists}) \quad (2.45) \\ &= 1 - P(\text{no minimal cut sets exist}). \end{aligned}$$

Also,

$$P(\text{no minimal cut sets exist}) \geq \prod_{i=1}^N P(\text{minimal cut set } i \text{ does not exist}). \quad (2.46)$$

Equality exists when no event occurs in more than one cut set.

Substituting Equation 2.46 into Equation 2.45 gives

$$P(\text{system failure}) \leq 1 - \prod_{i=1}^N P(\text{minimal cut set } i \text{ does not exist}), \quad (2.47)$$

which gives the Minimal Cut Set Upper Bound

$$P_{MCSUB}(Top) = 1 - \prod_{i=1}^N (1 - P(K_i)). \quad (2.48)$$

It can be shown that

$$Q(t) \leq 1 - \prod_{i=1}^N (1 - P(K_i)) \leq \sum_{i=1}^N P(K_i). \quad (2.49)$$

2.4.4 Top Event Frequency

The top event frequency is another system parameter that can be calculated - this is useful for systems where unreliability is an important issue. The system unconditional failure intensity, $w_{sys}(t)$, is defined as the expected number of times the top event occurs at time t , per unit time. Therefore $w_{sys}(t)dt$ is the expected number of times the top event occurs in t to $t + dt$. For the top event to occur in the interval $[t, t + dt)$ none of the cut set failures can exist at time t , and then at least one of them must fail in time t to $t + dt$. This can be written as:

$$w_{sys}(t)dt = P(A \bigcup_{i=1}^N \theta_i), \quad (2.50)$$

where:

A is the event that no minimal cut sets exist at time t ,

$\bigcup_{i=1}^N \theta_i$ is the event that one or more minimal cut sets occur in time $[t, t + dt)$

As $P(A) = 1 - P(\bar{A})$, the right hand of Equation 2.50 can be written:

$$P(A \bigcup_{i=1}^N \theta_i) = P(\bigcup_{i=1}^N \theta_i) - P(\bar{A} \bigcup_{i=1}^N \theta_i), \quad (2.51)$$

where \bar{A} is the event that at least one minimal cut set exists at t .

Therefore $w_{sys}(t)$ becomes:

$$w_{sys}(t)dt = P(\bigcup_{i=1}^N \theta_i) - P(\bar{A} \bigcup_{i=1}^N \theta_i). \quad (2.52)$$

The first term on the right-hand side gives the contribution from the occurrence of at least one minimal cut set. The second term gives the contribution of the minimal cut set occurrence while other minimal cut sets already exist (i.e. the system is already failed). These terms are denoted by $w_{sys}^{(1)}(t)dt$ and $w_{sys}^{(2)}(t)dt$ respectively to give:

$$w_{sys}(t) = w_{sys}^{(1)}(t) - w_{sys}^{(2)}(t). \quad (2.53)$$

The terms on the right of the above equation can be expanded using the inclusion-exclusion principle but as this is a computationally intensive operation, an approximation is required

2.4.4.1 Approximation for the system unconditional failure intensity

If component failures are rare then minimal cut set failures will also be rare events. The term $w_{sys}^{(2)}(t)$, which requires minimal cut sets to exist and occur at the same time, would become negligible if component failures are unlikely. Therefore, an upper bound for $w_{sys}(t)$ is simply:

$$w_{sys}(t)_{max} = w_{sys}^{(1)}(t). \quad (2.54)$$

As $w_{sys}(t)$ can be expanded using the inclusion-exclusion principle the series expansion is truncated after the first term (as for the top event probability) to give the Rare Event Approximation:

$$w_{sys}(t)_{max} dt \leq \sum_{i=1}^N P(\theta_i) \leq \sum_{k=1}^N w_{K_i}(t) dt, \quad (2.55)$$

where

$w_{K_i}(t)$ is the unconditional failure intensity of minimal cut set K_i

2.4.4.2 Expected number of system failures

The expected number of system failures in time t , $W(0, t)$, is given by the integral of the system unconditional failure intensity in the interval t :

$$W(0, t) = \int_0^t w_{sys}(u) du \quad (2.56)$$

For a reliable system the expected number of system failures is an upper bound for the system unreliability, $F(t)$ (i.e. $F(t) \leq W(0, t)$).

2.4.5 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. An importance analysis is a sensitivity analysis which identifies weak areas of the system and can be very valuable at the design stage. For each component its importance signifies the role that it plays in either causing or contributing to the occurrence of the top event. In general a numerical value is assigned to each basic event which allows it to be ranked according to the extent of its contribution to the occurrence of the top event. Importance measures can be categorized in two ways: deterministic and probabilistic

2.4.5.1 Deterministic measures

Deterministic importance measures evaluate the importance of a component without considering its probability to fail. One such measure is *Structural Measure of Importance*. It is given by:

$$I_i = \frac{\text{number of critical system states for component } i}{\text{total number of states for the } (n - 1) \text{ remaining components}}. \quad (2.57)$$

A critical system state for component i is a state for which the failure of component i will cause the system to go from a working to a failed state.

2.4.5.2 Probabilistic measures of system unavailability

Probabilistic measures are generally of more use than deterministic measures in reliability problems as they take into account the components' probability of failure.

Birnbaum's Measure of Importance is also known as the criticality function which defines the probability that the system is in a critical system state for component i . There are two expressions which can be used to obtain the criticality function

$$a) \quad G_i(\mathbf{q}(t)) = Q(1_i, \mathbf{q}(t)) - Q(0_i, \mathbf{q}(t)), \quad (2.58)$$

where

$$\begin{aligned} Q(t) & \text{ is the probability that the system fails,} \\ (1_i, \mathbf{q}(t)) & = (q_1, \dots, q_{i-1}, 1, q_{i+1}, \dots, q_n) \text{ component } i \text{ failed,} \\ (0_i, \mathbf{q}(t)) & = (q_1, \dots, q_{i-1}, 0, q_{i+1}, \dots, q_n) \text{ component } i \text{ is working.} \end{aligned}$$

The above expression gives the probability that the system fails with component i failed minus the probability of the system failing with component i working, which results in the probability that the system fails only if component i fails.

$$b) \quad G_i(\mathbf{q}(t)) = \frac{\partial Q(t)}{\partial q_i(t)}. \quad (2.59)$$

This is equivalent to Equation 2.58 as the probability function is linear in each $q_i(t)$

$$\frac{\partial Q(t)}{\partial q_i(t)} = \frac{Q(1_i, \mathbf{q}(t)) - Q(0_i, \mathbf{q}(t))}{1 - 0}. \quad (2.60)$$

This measure importance forms the basis for many importance measures.

In terms of Birnbaum's measure of component reliability importance, the expected number of system failures can be calculated as.

$$W(0, t) = \int_0^t \sum_{i=1}^n G_i(\mathbf{q}(t)) w_i(u) du, \quad (2.61)$$

where $w_i(t)$ denotes the component unconditional failure intensity and n denotes the total number of system components.

Criticality Measure of Importance calculates the probability that the system is in a critical state for component i and that i has failed. Unlike Birnbaum's measure of importance it also takes into account the failure probability of component i itself.

$$I_i = \frac{G_i(\mathbf{q}(t)) q_i(t)}{Q(t)} \quad (2.62)$$

Fussell-Vesely Measure of Importance calculates the probability that component i contributes to system failure and is defined as the probability of the union of the minimal cut sets containing i , given that the system has failed.

$$I_i = \frac{P(\bigcup_{j|i \in K_j} K_j)}{Q(t)}. \quad (2.63)$$

This measure gives very similar importance rankings to those obtained using the criticality measure.

Fussell-Vesely Measure of Minimal Cut Set Importance ranks the minimal cut sets in the order of their contribution to the top event, rather than considering the individual components. It is defined as the probability of existence of the cut set i , given that the system has failed.

$$I_i = \frac{P(K_i)}{Q(t)}. \quad (2.64)$$

2.5 Simplification of the Fault Tree Structure

Fault trees can be very large and their qualitative and quantitative analyses time-consuming. Therefore two pre-processing techniques can be applied to the fault tree in order to obtain the smallest possible subtrees [5]. The first stage of pre-processing is a reduction, technique used in the Faunet code [6], which restructures the fault tree to its most concise form. Once this has been applied it is possible to

simplify the analysis further by identifying independent subtrees (modules) within the fault tree that can be treated separately. The Rauzy's algorithm [7] is an extremely efficient method of modularisation and forms the second stage of fault tree pre-processing. This results in a set of independent fault trees each with the simplest possible structure, which together describe the original system.

2.5.1 Fault Tree Reduction

The reduction technique reduces the fault tree to its minimal form so eliminating any 'noise' from the system without altering the underlying logic.

2.5.1.1 Introduction

Fault trees are rarely written in their most concise format and this can have a significant effect on the efficiency of the resulting analysis. Their complexity can be reduced by applying fault tree reduction techniques, which optimise the structure of the tree whilst retaining the underlying logic. One such technique is the reduction approach, which is applied in four stages.

2.5.1.1.1 Reduction technique

This method of fault tree reduction consists of four stages.

1. **Contraction**

Subsequent gates of the same type are contracted to form a single gate. This gives an alternating sequence of 'AND' gates and 'OR' gates throughout the tree.

2. **Factorisation**

Pairs of events that always occur together in the same gate type are identified. They are combined to form a single complex event.

3. **Extraction**

The two structures shown in Figure 2.2 are identified and replaced as shown.

4. **Absorption**

The structures in the fault tree are identified that could be further simplified through the application of the absorption and idempotent laws to the fault tree logic. If primary and secondary gates with an event in common are of a different type, the structure is simplified by removing the whole secondary

gate and its descendants. If primary and secondary gates are of the same type, the structure is simplified by deleting the occurrence of the event beneath the secondary gate. These cases are illustrated in Figure 2.3

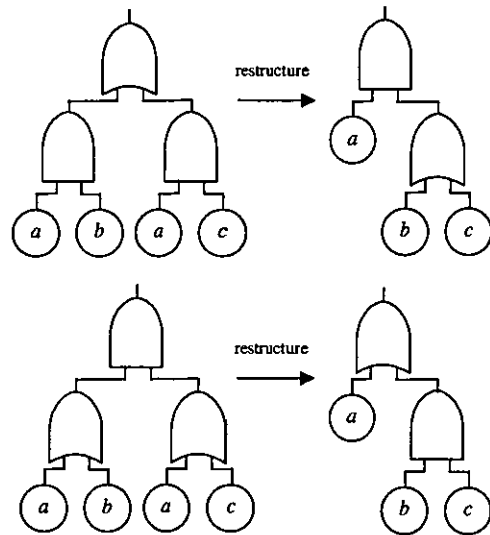


Figure 2.2: The extraction procedure

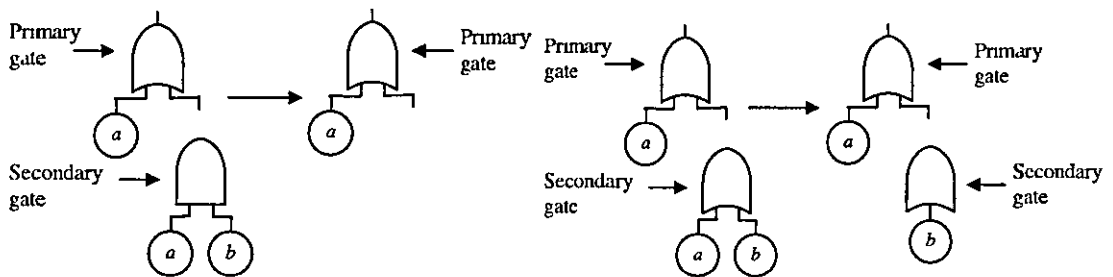


Figure 2.3 The absorption procedure

The above four steps are repeated until no further changes are possible in the fault tree, resulting in a more compact representation of the system.

2.5.1.2 Worked example of the reduction technique

This technique will be applied to the example fault tree. It is shown in Figure 2.4 together with its numerical form at the start of the reduction process

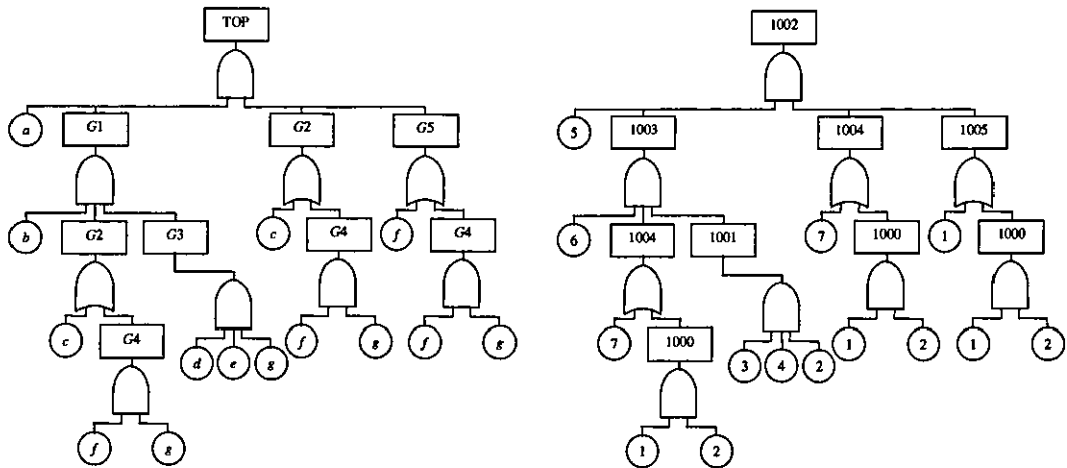


Figure 2.4: Example of fault tree

2.5.1.2.1 Contraction

The aim of the first stage is to identify subsequent gates in the tree structure that have the same gate type. Application of the contraction stage is implemented to the fault tree shown in Figure 2.4.

In this example gate 1003 appears as an input to gate 1002 and they both are 'AND' gates. Gate 1003 only appears once in the fault tree data, so its inputs are directed to gate 1002. Since gate 1004 is an input to gate 1002 it is not listed for the second time. Now gate 1001 appears as an input to gate 1002 and they are both 'AND' gates. As gate 1001 only appears once in the fault tree data, its inputs are directed to gate 1002. The resulting fault tree is shown in Figure 2.5.

2.5.1.2.2 Factorisation

The fault tree now has an alternating sequence of 'AND' and 'OR' gates and can be factorised. The input events to each gate are considered one by one, looking for pairs that always occur together. Once it has been established that they do always occur together and under the same gate type, a complex event is created, which is numbered from 2000 upwards. The new complex event is recorded together with the gate type and the two events from which it was formed. Application of factorisation to the fault tree shown in Figure 2.5 gives complex events listed in Table 2.3. The modified fault tree is shown in Figure 2.6.

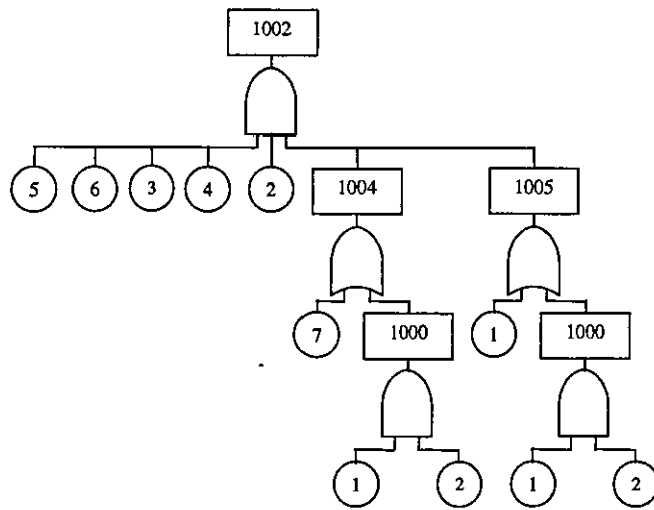


Figure 2.5: The fault tree after contraction

Complex event	Gate value	Event 1	Event 2
2000	AND	<i>a</i>	<i>b</i>
2001	AND	2000	<i>d</i>
2002	AND	2001	<i>e</i>

Table 2.3. Complex event data after factorisation

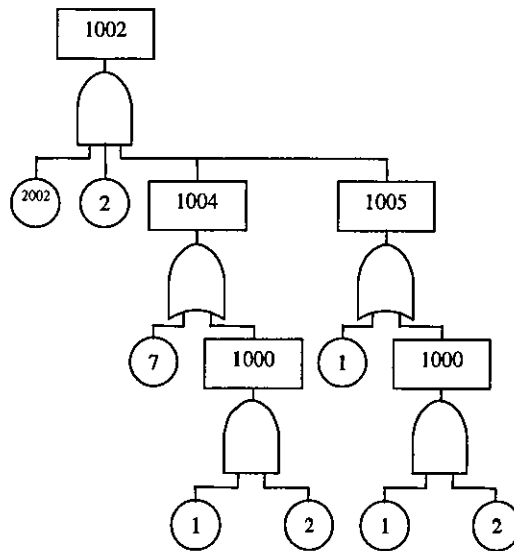


Figure 2.6: The fault tree after factorisation

2.5.1.2.3 Extraction

The extraction process searches for the structures shown in Figure 2.2. If the primary gate has two or more gates as inputs (referred to as the secondary gates) then the gates are selected in pairs. Both secondary gates are then checked to see if they are of the same type, but of a different type to the primary gate. If so, the inputs to the secondary gates are checked to see if they have a gate or event in common. If they do then extraction can take place.

Before extraction can occur, however, there may be some necessary adjustments to be made to the data. If the primary gate has more than two inputs then a new gate must be created which has the same gate type as the primary gate, but which has the primary gate and all its inputs, except the two secondary gates, as inputs. This restructures the fault tree into the form required for extraction by using an equivalent representation. Application of the extraction procedure is carried out on the fault tree shown in Figure 2.6.

The only gate that has two or more gates as inputs is top gate 1002, whose inputs are 1004 and 1005. These secondary gates are both of a different type to the primary gate, and have gate 1000 in common, which can be extracted. In order to get this tree into the required form for extraction gate 1006 is generated, as shown in Figure 2.7. Gate 1002 now only has its two secondary gates as inputs.

A new gate, 1007, is created, which is of the same type as the secondary gates and has the same common input, 1000, and the primary gate 1002, as its inputs. The resulting tree is shown in Figure 2.8.

It is clear from Figure 2.8 that another extraction can also be undertaken. Gates 1000 and 1002 also have event 1 in common, which can be extracted. Since gate 1006 is the same type as the secondary gates, the same common input 1 and the primary gate 1007 are added to the list of its inputs. The fault tree after the extraction is shown in Figure 2.9.

2.5.1.2.4 Absorption

During the absorption process the repeated events are considered in the fault tree. If the type of the primary gate is different from the type of the secondary gate,

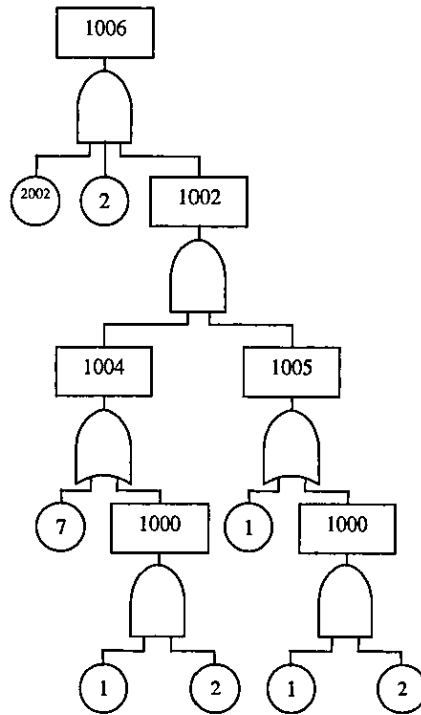


Figure 2.7: The fault tree during extraction, gate 1006 is created

the secondary gate is deleted. If the types of the primary and secondary gates are the same, the repeated event is deleted from the list of the inputs of the secondary gate. Application of the absorption procedure is carried out on the fault tree shown in Figure 2.9.

The only repeated event in the fault tree is event 2. The first time it occurs as an input to the primary gate 1006, which is an 'AND' gate, and the second time it occurs as an input to the secondary gate 1007, which is an 'OR' gate. Since the types of the primary and the secondary gates are different the secondary gate can be deleted. The fault tree after the absorption is shown in Figure 2.10.

Finally, the factorisation can be applied again and it finishes the reduction process. New complex events are shown in Table 2.4. The reduced fault tree is shown in Figure 2.11 in terms of original gate name and complex event.

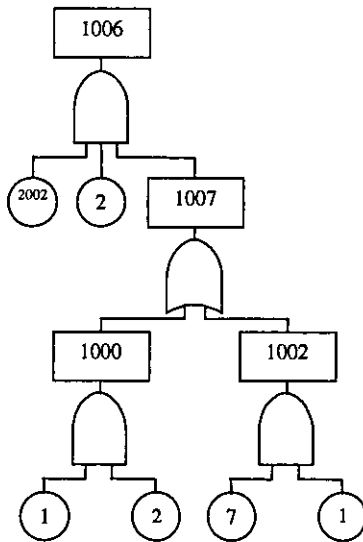


Figure 2.8: The fault tree during extraction, gate 1007 is created

Complex event	Gate value	Event 1	Event 2
2003	AND	2002	<i>f</i>
2004	AND	2003	<i>g</i>

Table 2.4. Complex event data after the second factorisation

2.5.1.3 Reduced fault tree

It can be verified that the reduced tree is equivalent to the original tree by examining their minimal cut sets. These will be identical for logically equivalent trees. The original tree has one minimal cut set of order 6

$$\{f, a, d, b, g, e\}. \quad (2.65)$$

The minimal cut set for the reduced tree is

$$\{2004\}. \quad (2.66)$$

This can be expanded out in terms of the basic events. The principle is that 'OR' gates increase the number of cut sets, whilst 'AND' gates increase the number of elements in the cut sets. Therefore the minimal cut set of the reduced tree can be expanded to give:

$$\begin{aligned} TOP &= 2004 = 2003 \cdot g = 2002 \cdot f \cdot g = 2001 \cdot e \cdot f \cdot g & (2.67) \\ &= 2000 \cdot d \cdot e \cdot f \cdot g = a \cdot b \cdot d \cdot e \cdot f \cdot g, \end{aligned}$$

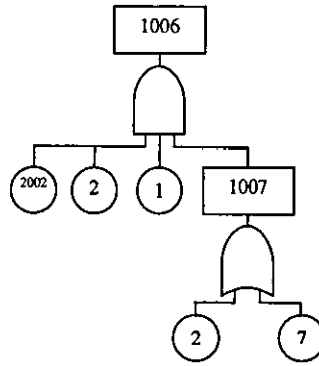


Figure 2.9 Fault tree after extraction

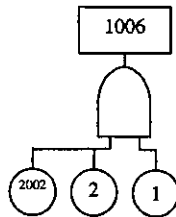


Figure 2.10: Fault tree after absorption

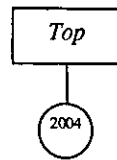


Figure 2.11: The reduced fault tree

which is equivalent to those obtained from the original tree.

Reduction has simplified the example fault tree considerably. In the original fault tree there were six gates and fourteen events, eight of them different; the reduced fault tree contains one event. However, this method rarely reduces a fault tree to a single event as it did in this simple example.

Having reduced the fault tree to a more concise form the second pre-processing technique of modularisation is considered.

2.5.2 Fault Tree Modularisation

Modularisation methods can be applied to fault trees in order to reduce their complexity and simplify the resulting analysis. The modularisation procedure identifies subtrees within the fault tree, known as modules. A module of a fault tree is a subtree that is completely independent from the rest of the tree. It contains no basic events that appear elsewhere in the fault tree. The advantage of identifying these modules is that each one can be analysed separately from the rest of the tree. The results from subtrees identified as modules are substituted into the higher-level fault trees where the modules occur.

Several modularisation techniques are available for detecting fault tree modules but one of particular interest is Rauzy's linear-time algorithm [7]. The advantage of this algorithm over other techniques is its efficiency, only two passes through the fault tree are required in order to obtain the modules.

2.5.2.1 Rauzy's algorithm

Using the linear-time algorithm the modules can be identified after just two depth-first traversals of the fault tree. The first of these performs a step-by-step traversal recording, for each gate and event, the step number at the first, second and final visits to that node. To demonstrate this process refer to the fault tree in Figure 2.12. Starting at the top event and progressing through the tree in a depth-first manner the gates and events are visited in the order shown in Table 2.5.

Step number	1	2	3	4	5	6	7	8	9	10	11	12	13
node	<i>Top</i>	<i>a</i>	<i>G1</i>	<i>b</i>	<i>G2</i>	<i>c</i>	<i>G4</i>	<i>f</i>	<i>g</i>	<i>G4</i>	<i>G2</i>	<i>G3</i>	<i>d</i>
Step number	14	15	16	17	18	19	20	21	22				
node	<i>e</i>	<i>G3</i>	<i>G1</i>	<i>G2</i>	<i>G5</i>	<i>h</i>	<i>G3</i>	<i>G5</i>	<i>Top</i>				

Table 2.5: Order in which the gates and events are visited in the depth-first traversal of the fault tree in Figure 2.12

Event inputs to any gate are considered before the gate inputs. Each gate is visited at least twice, once on the way down the tree and again on the way back up the tree. Once a gate has been visited it can be visited again, but the depth-first traversal beneath that gate is not repeated. This is shown at step 17 and step 20 where gate *G2* and *G3* are visited again but their descendants (any gates and

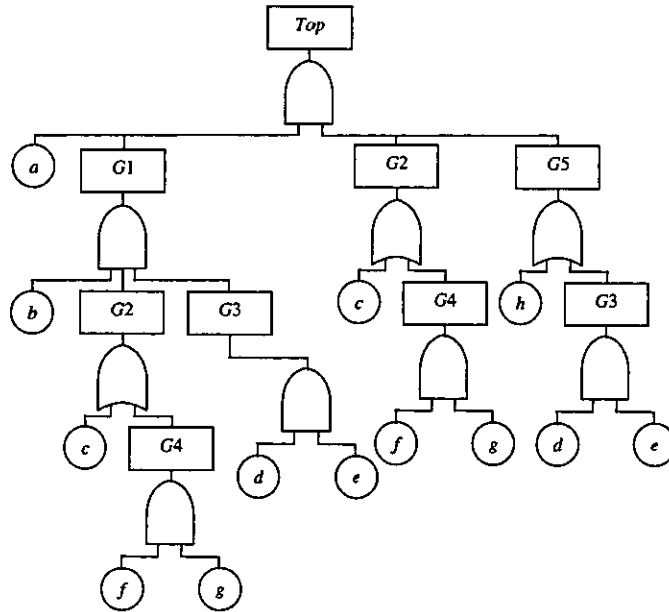


Figure 2.12. Example fault tree to demonstrate the linear-time algorithm

events appearing below that gate in the tree) are not re-visited. The step numbers of the visits (first, second and final) are recorded during this traversal and values for the gates are shown in Table 2.6.

Gate	<i>Top</i>	<i>G1</i>	<i>G2</i>	<i>G3</i>	<i>G4</i>	<i>G5</i>
1 st visit	1	3	5	12	7	18
2 nd visit	22	16	11	15	10	21
Final visit	22	16	17	20	10	21
Min	2	4	6	13	8	12
Max	21	17	10	14	9	20

Table 2.6. Data for the gates in the fault tree

As gates *G2* and *G3* are repeated gates the step numbers of the final visit are different to those of the second visit. The equivalent data for the events is shown in Table 2.7. It should be noted that the step number of the second visit to each basic event is equivalent to the step number of the first visit to that event

The second pass through the tree finds the maximum (max) of the last visits and the minimum (min) of the first visits to the descendants of each gate, these values are also shown in Table 2.6

Event	<i>a</i>	<i>b</i>	<i>c</i>	<i>d</i>	<i>e</i>	<i>f</i>	<i>g</i>	<i>h</i>
1 st visit	2	4	6	13	14	8	9	19
2 nd visit	2	4	6	13	14	8	9	19
Final visit	2	4	6	13	14	8	9	19

Table 2.7. Data for the events in the fault tree

The principle of the algorithm is that if any descendants of each gate has a first visit step number smaller than the first visit step number of the gate then it must occur beneath another gate. Conversely, if any descendant has a last visit step number greater than the second visit step number of the gate, then again it must occur elsewhere in the tree.

Therefore, a gate can be identified as heading a module if

- The first visit to each descendant is after the first visit to the gate,
- The last visit to each descendant is before the second visit to the gate.

That is, none of the descendants of a gate can appear anywhere else in the tree (unless beneath another occurrence of the same gate). Therefore, the final step of the algorithm simply compares the minimum (min) and maximum (max) values of the descendants visit numbers with the first and second visit step numbers for each gate.

From Table 2.6 it can be seen that gate *G1* cannot be a module as its descendants have a maximum step number greater than the second visit step number of this gate. Gate *G5* is also not a module as its descendants have a minimum step number smaller than the first visit step number of the gate.

The following gates can therefore be identified as heading modules

$$Top, G2, G3, G4. \quad (2.68)$$

The top event will always be a module of the fault tree. Each of the subtrees can be replaced by a single modular event in the fault tree structure and are assigned the following labels

$$G2 \rightarrow M1, G3 \rightarrow M2, G4 \rightarrow M3. \quad (2.69)$$

Four separate fault trees as shown in Figure 2 13 now replace the single tree shown in Figure 2 12

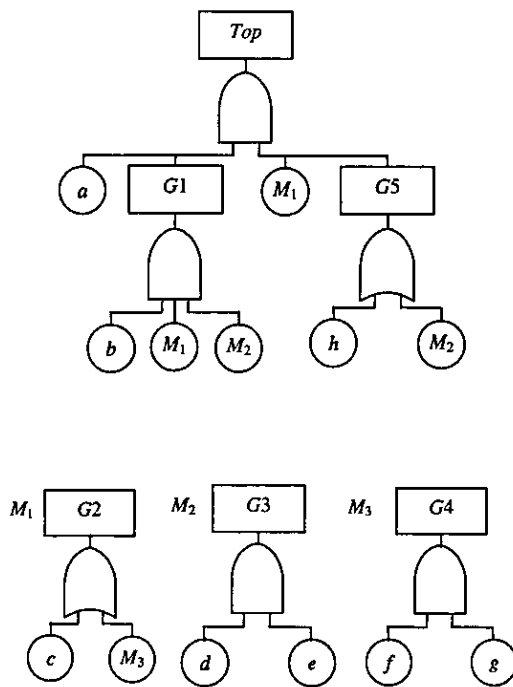


Figure 2 13 The four modules obtained from the fault tree shown in Figure 2.12 - Modularised fault tree, Module M_1 , Module M_2 and Module M_3

Having identified the modules each one can be analysed separately into the higher-level fault trees where the modules occur. This process can significantly reduce the number of calculations required in the subsequent analysis.

2.6 Summary

Fault Trees are an extremely good way of representing the failure logic of the system in a visual format. The qualitative analysis enables minimal cut sets of the tree to be found, which are the smallest combinations of basic events that cause system failure. A number of parameters, such as the top event probability and frequency, together with the expected number of occurrences of the top event and event importance measures, obtained from the quantitative analysis gain a full understanding of the system. But this analysis has a disadvantage - if the fault tree is large then performing analysis upon it can require extensive calculations. Approximations are needed for many parameters which leads to a loss of accuracy

Chapter 3

Binary Decision Diagrams

3.1 Introduction

Binary Decision Diagrams (BDDs) were first used by Lee [8] to represent switching circuits. Akers [9] introduced BDDs as a method for defining, analyzing, testing and implementing large digital functions. Bryant [10] [11] presented functions by directed, acyclic graphs in a manner similar to the previous representations but with further restrictions on the ordering of decision variables in the graph. Initially Schneeweiss [12] presented an algorithm for the production of short disjoint-products form for a fault tree output function using a sequential binary decision process. The use of BDDs in reliability analysis was developed predominantly by Rauzy [2], who suggested that they might provide an alternative, more efficient technique for performing fault tree analysis.

The BDD method does not analyse the fault tree directly, but converts the tree to a Binary Decision Diagram, which represents the Boolean equation for the top event. This representation of the logic equation is in a form that is much easier to manipulate than a fault tree. Both qualitative and quantitative analysis can be performed on the BDD, with the advantage that exact solutions can be calculated very efficiently without the need for the approximations necessary in the conventional approach of Kinetic Tree Theory.

3.2 Properties of the BDD

A BDD is a directed acyclic graph, which means that all paths through the BDD are in one direction and that no loops can exist. All paths through the BDD

terminate in one of two states: either a 1 state, which corresponds to system failure, or a 0 state, which corresponds to system success. The BDD is composed of terminal and non-terminal vertices (also called nodes) which are connected by branches. Terminal vertices correspond to the final state of the system (failure or success) and non-terminal vertices correspond to the basic events of the fault tree. Each non-terminal vertex has a 1 branch, which represents basic event occurrence, and a 0 branch, which represents basic event non-occurrence. By convention, the left-hand branch is a 1 branch; the right-hand branch is a 0 branch. The structure of the BDD is presented in Figure 3.1. The size of BDD is usually measured by its

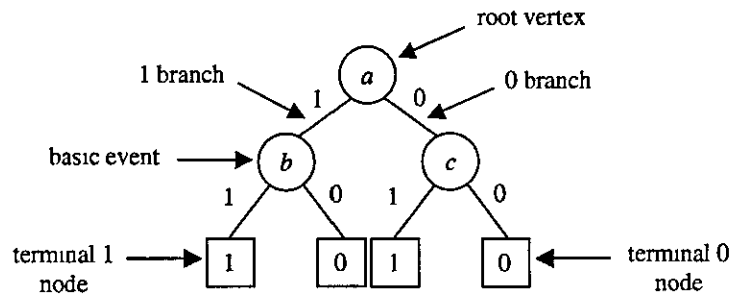


Figure 3.1: Example of Binary Decision Diagram

number of non-terminal vertices

All paths through the diagram start at the root vertex and proceed to a terminal vertex, which marks the end of the path. Each path that terminates in a 1 state gives a cut set of the fault tree, as that particular combination of component failures must result in system failure. Only vertices that lie on the 1 branches of these paths are included in the cut sets. For example, in the BDD shown in Figure 3.1 there are two possible paths that terminate in 1 state. These are:

1. a, b
2. \bar{a}, c

which give the two corresponding cut sets

1. $\{a, b\}$
2. $\{c\}$.

In this example the BDD is in its minimal form and so generates only minimal cut sets. However, this is not usually the case, as is discussed later in this chapter.

3.2.1 Formation of the BDD Using If-Then-Else

This method of constructing the BDD was developed by Rauzy [2] and proceeds by applying an **if-then-else** (**ite**) technique to each of the gates in the fault tree. The **ite** structure derives from Shannon's formula, which is discussed in detail in Chapter 2, Equations 2.34 - 2.37. If $f(\mathbf{x})$ is the Boolean function for the fault tree top event then by pivoting about any variable X Shannon's formula can be written as

$$f(\mathbf{x}) = X \cdot f_1 + \bar{X} \cdot f_2, \quad (3.1)$$

where f_1 and f_2 are Boolean functions with $X = 1$ and $X = 0$ respectively, and are of one order less than $f(\mathbf{x})$. The corresponding **ite** structure is $\text{ite} = (X, f_1, f_2)$, where X is the Boolean variable and f_1 and f_2 are logic functions. This means that **if** X fails **then** consider f_1 , **else** consider f_2 . Therefore, in the BDD structure f_1 lies below branch 1 of the node encoding X and f_2 lies below branch 0. This is shown in Figure 3.2.

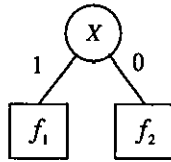


Figure 3.2 BDD showing $\text{ite} = (X, f_1, f_2)$

Once a variable ordering has been established, the following procedure can be implemented to construct the BDD.

- Each basic event X is assigned the **ite** structure $\text{ite}(X, 1, 0)$.

Let J and H be two nodes in the BDD where:

$$J = \text{ite}(X, F_1, F_2), \quad H = \text{ite}(Y, G_1, G_2).$$

- If $X < Y$ (i.e. X appears before Y in the variable ordering) then

$$J < op > H = \text{ite}(X, F_1 < op > H, F_2 < op > H). \quad (3.2)$$

- If $X = Y$ then

$$J < op > H = \text{ite}(X, F_1 < op > G_1, F_2 < op > G_2), \quad (3.3)$$

where $\langle op \rangle$ corresponds to a Boolean operation of the gates in the fault tree.

The following identities can also be used to simplify the results:

$$1 \langle op \rangle H = 1, \text{ if } \langle op \rangle \text{ is an 'OR' gate} \quad (3.4)$$

$$1 \langle op \rangle H = H, \text{ if } \langle op \rangle \text{ is an 'AND' gate} \quad (3.5)$$

$$0 \langle op \rangle H = H, \text{ if } \langle op \rangle \text{ is an 'OR' gate} \quad (3.6)$$

$$0 \langle op \rangle H = 0, \text{ if } \langle op \rangle \text{ is an 'AND' gate} \quad (3.7)$$

An advantage of the *ite* method for constructing the BDD is that the algorithm automatically makes use of sub-node sharing. This not only reduces the computer memory requirements, as each *ite* structure is only stored once, but it also increases the efficiency, since once an *ite* structure has been calculated the process does not need to be repeated.

This *ite* method can be demonstrated by constructing a BDD from the fault tree shown in Figure 3.3. The ordering $c < d < a < b$ represents simple top-down,

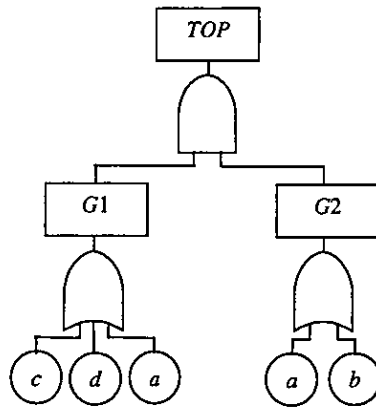


Figure 3.3: Example of fault tree

left-right traversal of the fault tree

G1 is expressed as:

$$\begin{aligned} G1 &= c + d + a & (3.8) \\ &= \text{ite}(c, 1, 0) + \text{ite}(d, 1, 0) + \text{ite}(a, 1, 0) \\ &= \text{ite}(c, 1, \text{ite}(d, 1, \text{ite}(a, 1, 0))). \end{aligned}$$

$G2$ is obtained in a similar way and it gives

$$\begin{aligned} G2 &= a + b \\ &= \text{ite}(a, 1, \text{ite}(b, 1, 0)). \end{aligned} \tag{3.9}$$

The **ite** structure for Top is given by

$$\begin{aligned} Top &= G1 \cdot G2 \\ &= \text{ite}(c, 1, \text{ite}(d, 1, \text{ite}(a, 1, 0))) \cdot \text{ite}(a, 1, \text{ite}(b, 1, 0)) \\ &= \text{ite}(c, \text{ite}(a, 1, \text{ite}(b, 1, 0)), \text{ite}(d, 1, \text{ite}(a, 1, 0))) \cdot \text{ite}(a, 1, \text{ite}(b, 1, 0)) \\ &= \text{ite}(c, \text{ite}(a, 1, \text{ite}(b, 1, 0)), \text{ite}(d, \text{ite}(a, 1, \text{ite}(b, 1, 0)), \text{ite}(a, 1, 0))) \end{aligned} \tag{3.10}$$

The BDD is constructed by considering branch 1 and branch 0 of each variable in turn. In this example 'c' is the first variable to be considered and it is encoded in the root vertex of the BDD. The structure $\text{ite}(a, 1, \text{ite}(b, 1, 0))$ lies below its branch 1 and $\text{ite}(d, \text{ite}(a, 1, \text{ite}(b, 1, 0)), \text{ite}(a, 1, 0))$ lies below branch 0. Event 'd' is the next variable to be considered and it is encoded in the node beneath the right-hand branch of the root vertex. Its outgoing branches are determined by breaking down the structure $\text{ite}(d, \text{ite}(a, 1, \text{ite}(b, 1, 0)), \text{ite}(a, 1, 0))$ into $\text{ite}(a, 1, \text{ite}(b, 1, 0))$ for branch 1 and $\text{ite}(a, 1, 0)$ for branch 0. This process is continued until all branches end with terminal vertices. The resulting BDD is shown in Figure 3.4. Now both qualitative and quantitative analysis can be carried out on the BDD.

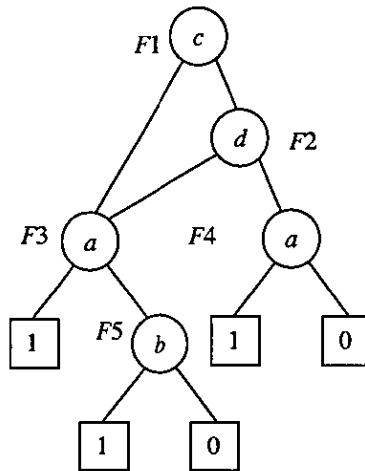


Figure 3.4 BDD obtained from the fault tree in Figure 3.3 using the **ite** technique

They are presented in the later sections

3.3 Reduction

Fault Tree Reduction, presented in Chapter 2, is a powerful method which obtains a smaller tree in terms of complex events. It is expected that a BDD constructed from a reduced fault tree will be substantially smaller than that constructed from a non-reduced fault tree. The original tree shown in Figure 2.4 was reduced and was presented in Figure 2.11. Since there is only one node in the reduced fault tree, the BDD from the reduced tree consists of one node only and the *ite* calculation is applied only once. Obviously, the BDD from the original fault tree would contain more nodes and require more *ite* calculations, since there are more than one basic event in the fault tree before the reduction.

3.4 Modularisation

The BDD construction process can be made more efficient by modularising the fault tree before the conversion procedure takes place. Modularisation identifies independent subtrees (modules) within the fault tree that can be analysed separately from the rest of the tree. A detailed discussion of the modularisation technique is given in Chapter 2.

Modularisation can significantly reduce the complexity of a fault tree by breaking it down into smaller, more manageable pieces that can be dealt with separately. In terms of the BDD process, the tree can then be analysed in several stages by obtaining smaller BDDs for each subtree. These can then be combined to form a BDD that represents the original fault tree structure. It is possible, therefore, that a BDD could be constructed for a tree that could not previously be analysed. The process can be demonstrated using the fault tree shown in Figure 3.5.

The modularised tree and modules $M1$, $M2$ and $M3$ are shown in Figure 3.6.

The following modules can be identified:

- TOP itself is a module.
- Module $M1$ is included in the module TOP .
- Module $M2$ is included in the module $M1$.
- Module $M3$ is included in the module $M2$.

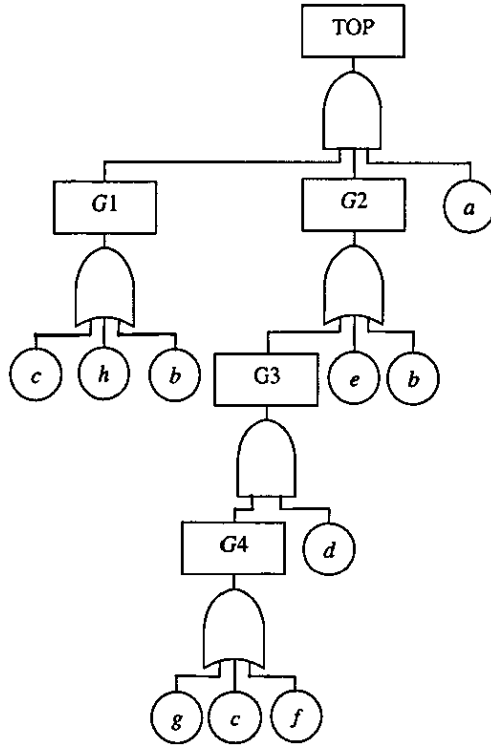


Figure 3 5: A fault tree that can be modularised

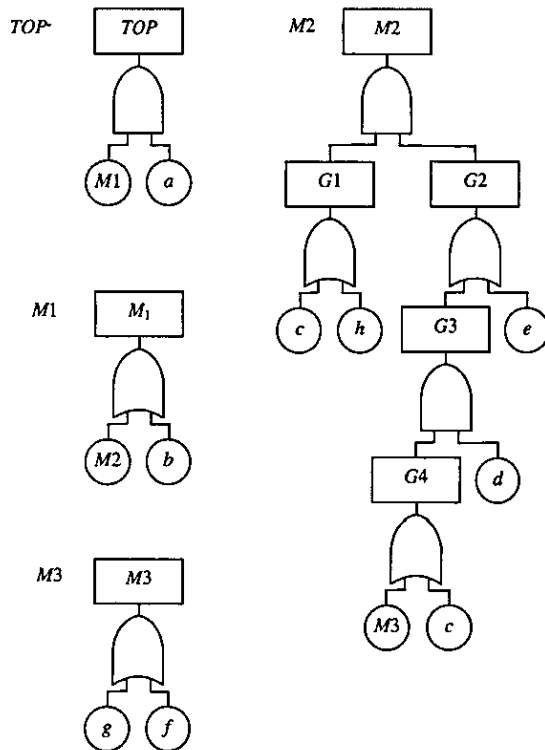


Figure 3 6: The modularised fault tree and three modules

To form the BDD from the modularised tree the modules are treated as events and so need to be ordered together with the basic events. Taking top-down order $M1 < a$, the *ite* structure for the top event can be formed:

$$\begin{aligned} TOP &= M1 \cdot a & (3.11) \\ &= \text{ite}(M1, \text{ite}(a, 1, 0), 0). \end{aligned}$$

Each module is then analysed independently to form its own BDD. The top-down orderings for the modules are as follows:

$$M1 : M2 < b \quad (3.12)$$

$$M2 : c < h < e < d < M3 \quad (3.13)$$

$$M3 : g < f \quad (3.14)$$

which result in the *ite* structures given by:

$$M1 = M2 + b \quad (3.15)$$

$$= \text{ite}(M2, 1, \text{ite}(b, 1, 0))$$

$$M2 = G1 \cdot G2 \quad (3.16)$$

$$= \text{ite}(c, \text{ite}(e, 1, \text{ite}(d, 1, 0)), \text{ite}(h, \text{ite}(e, 1, \text{ite}(d, \text{ite}(M3, 1, 0), 0)), 0), 0)$$

$$M3 = g + f \quad (3.17)$$

$$= \text{ite}(g, 1, \text{ite}(f, 1, 0)).$$

This corresponding set of BDDs is shown in Figure 3.7

3.5 Qualitative Analysis

Qualitative analysis of BDDs is an efficient way to obtain the cut sets of the fault tree, as shown in [13]. Every path through the BDD starts from the root vertex and proceeds down through the diagram to a terminal vertex. Paths which terminate at a 1 vertex yield the cut sets. If the BDD is in its minimal form it generates minimal cut sets. However, this is not always the case. The cut sets of the BDD presented in Figure 3.4 are:

1. $\{c, a\}$
2. $\{c, b\}$
3. $\{d, a\}$

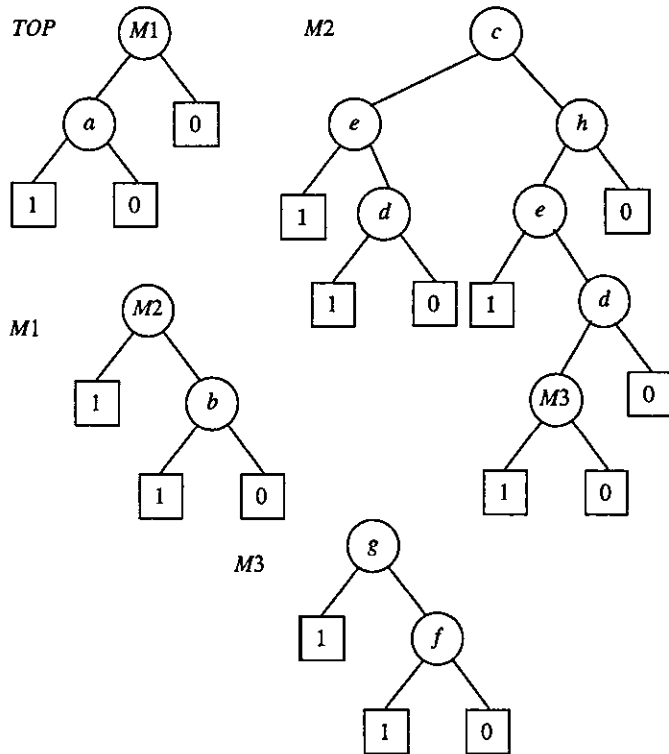


Figure 3 7: The BDDs obtained from the modularised fault tree and three modules

4. $\{d, b\}$

5 $\{a\}$

The cut sets are not minimal and the BDD is not in its minimal form. The first and third cut sets are redundant as they contain the fifth cut set as a subset. In order to obtain minimal cut sets the BDD has to undergo a minimisation procedure. This algorithm, introduced by Rauzy [2], is applied to the *ite* form of the BDD and creates a new BDD that exactly defines the minimal cut sets of the fault tree

Consider a general node in the BDD which is represented by the function F , where

$$F = ite(x, G, H). \quad (3 18)$$

If δ is a minimal solution of G , which is not a minimal solution of H , then the intersection of δ and x ($\{\delta\} \cap x$) will be a minimal solution of F . The set of all the minimal solutions of F ($sol_{min}(F)$) will also include the minimal solutions of H , so

$$sol_{min}(F) = [\{\delta\} \cap x] \cup [sol_{min}(H)]. \quad (3 19)$$

The 'without' operator was defined by Rauzy, which removes all the paths from G_{min} that are included in H_{min} . In this way the combined set $sol_{min}(F)$ represents the minimal solutions of F by removing any minimal solutions of G that are also minimal solutions of H .

This algorithm can be applied to the BDD in Figure 3.4. Each node is considered in turn:

$F1 = ite(c, F3, F2)$ - Event 'a' is included in a path on both the one branch ($F3$) and the zero branch ($F2$). Therefore 'a' is removed from the one branch by replacing the terminal vertex 1 with a terminal vertex 0.

$F2 = ite(d, F3, F4)$ - There are no events included in path on both the one branch ($F3$) and the zero branch ($F4$), because event 'a' has been removed from the one branch already.

$F3 = ite(a, 1, F5)$ - $F5$ does not contain any paths that are included in the one branch as it leads to a terminal vertex.

$F4 = ite(a, 1, 0)$ - Both the one and zero branches are terminal.

$F5 = ite(b, 1, 0)$ - Both the one and zero branches are terminal.

The minimised BDD is shown in Figure 3.8. This produces the following mini-

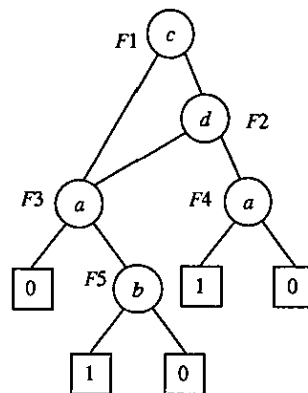


Figure 3.8: The minimised BDD

mal cut sets

1. $\{c, b\}$
2. $\{d, b\}$

3 {*a*}

Minimising the BDD has therefore removed the redundant cut sets {*c, a*} and {*d, a*}.

3.5.1 Incorporating Complex Events and Modules into the Analysis

The following section describes an extension of qualitative analysis, which considers BDDs encoding complex events and/or modular events produced as part of this research project, published in [14], [15]. The aim of this is to obtain the minimal cut sets of the system by extracting the minimal combinations of component failures which produce system failure from every complex and modular event. This is necessary because when reduction and modularisation are used to construct the BDDs, it is useful to be able to analyse the system in terms of its original components.

The calculation process starts at the root vertex of the primary BDD, computing all the paths that terminate at a 1 vertex, which represent the cut sets of the system. If any of the constituent events in those cut sets is modular the calculation is carried out in the BDDs of each module. If a complex event is included cut sets are expanded to the original basic events by following the rules of obtaining cut sets for fault trees, presented in Chapter 2. After this the components of the new expanded cut sets are combined with the primary minimal cut sets. The example is presented in Figure 3.9.

The primary BDD (*a*) produces the minimal cut set:

$$\{M1, a\}.$$

Since this minimal cut set includes the event *M1* further calculations are carried out on the BDD of module *M1*. The minimal cut sets of module *M1* are

1. {*M2*},
2. {*b*}.

As one of the components in these minimal cut sets is the modular event *M2* the BDD of *M2* (*c*) must now be investigated. Four minimal cut sets are obtained

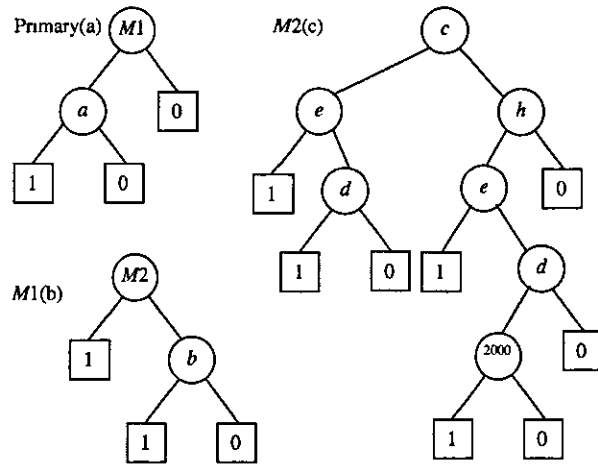


Figure 3 9: Example of BDDs for the modules: Primary, M_1 , M_2

1. $\{c, e\}$,
2. $\{c, d\}$,
3. $\{h, e\}$,
4. $\{h, d, 2000\}$.

This time the complex event 2000 must be taken into consideration. Since

$$2000 = g + f,$$

this complex event has two minimal cut sets - $\{g\}$ and $\{f\}$. These minimal cut sets contain only basic events, therefore the calculation is finished.

Now all the minimal cut sets must be constructed using the extractions from the modular (M_1 and M_2) and complex (2000) events. The substitution of the minimal cut sets of 2000 ($\{g\}$ and $\{f\}$) into $\{h, d, 2000\}$ gives two minimal cut sets

1. $\{h, d, g\}$,
2. $\{h, d, f\}$.

Similar substitutions are done using the minimal cut sets of M_2 to find the minimal cut sets of M_1 . Finally, the minimal cut sets are

1. $\{c, e, a\}$,
2. $\{c, d, a\}$,

3. $\{h, e, a\}$,
4. $\{h, d, g, a\}$,
5. $\{h, d, f, a\}$,
6. $\{b, a\}$.

These cut sets are minimal because the BDDs of every module underwent the Rauzy minimisation procedure explained earlier, which gives a minimal form for the BDDs.

3.6 Quantitative Analysis

The quantitative analysis of BDDs determines many probabilistic properties of the system. It is an efficient procedure with the advantage that exact solutions can be calculated without the need for the approximations necessary in the conventional approach of Kinetic Tree Theory. In this chapter the current procedures for performing the basic elements of quantitative analysis, such as calculating the system unavailability, the unconditional failure intensity and the criticality function for basic events, are explained, as shown in [16].

3.6.1 System Unavailability

The structure encoded in the BDD is derived from Shannon's formula, Equation 3.1. The probability of the top event (i.e. system unavailability) can be found by taking the expectation of each term of Equation 3.1, to give:

$$E(f(\mathbf{x})) = q_i(t) \cdot E(f_1) + (1 - q_i(t)) \cdot E(f_2), \quad (3.20)$$

where $q_i(t) = E(x_i)$, the probability that event x_i occurs.

Therefore the probability of occurrence of the top event, $Q(t)$, can be expressed as the sum of the probabilities of the disjoint paths through the BDD. The disjoint paths can be found by tracing all paths from the root vertex to terminal 1 vertices. Each disjoint path represents a combination of working and failed components that leads to system failure. Therefore events lying on both branches 1 and 0 are included in the probability calculation.

In order to calculate the top probability, Equation 3 20 can be applied to each node in the BDD. For any node, $F = \text{ite}(x_i, J, K)$, the probability value is given by:

$$P(F) = q_i(t) \cdot P(J) + (1 - q_i(t)) \cdot P(K), \quad (3\ 21)$$

where

$P(J)$ is the probability value of the node on branch 1 of F ,

$P(K)$ is the probability value of the node on branch 0 of F

Equation 3 21 is applied to the BDD in a bottom-up manner. Nodes that have terminal vertices on both their branches 1 and 0 are considered first. The values are then worked up through the BDD until the top event probability can be evaluated.

3.6.2 System Unconditional Failure Intensity

It is not possible to use a fault tree to calculate the unreliability for the top event, i.e. the probability that it will not work continuously over a given time period. However, an upper bound for this is the expected number of top event occurrences, $W(0, t)$:

$$W(0, t) = \int_0^t w_{sys}(t) dt, \quad (3\ 22)$$

where $w_{sys}(t)$ is the system unconditional failure intensity. This can be expressed as:

$$w_{sys}(t) = \sum_i G_i(\mathbf{q}(t)) w_i, \quad (3\ 23)$$

where $G_i(\mathbf{q}(t))$ is the criticality function for each component and w_i is the component unconditional failure intensity.

The criticality function $G_i(\mathbf{q}(t))$ 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 state to the failed state, i.e. the probability that the system fails only if component i fails. Therefore:

$$G_i(\mathbf{q}(t)) = Q(1, \mathbf{q}(t)) - Q(0, \mathbf{q}(t)), \quad (3\ 24)$$

where $Q(1_i, \mathbf{q}(t))$ is the probability of system failure with $q_i(t) = 1$ and $Q(0_i, \mathbf{q}(t))$ is the probability of system failure with $q_i(t) = 0$.

An efficient method of calculating the criticality function from the BDD considers the probabilities of the path sections in the BDD up to and after the relevant nodes. For example, consider the variable x_i , which occurs at two intermediate nodes in the BDD, as shown in Figure 3.10. $Q(1_i, \mathbf{q}(t))$ and $Q(0_i, \mathbf{q}(t))$ can be calculated

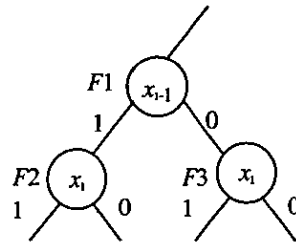


Figure 3.10. BDD section showing the locations of variable x_i

for this variable using

$$Q(1_i, \mathbf{q}(t)) = \sum_n (pr_{x_i}(\mathbf{q}(t)) \cdot po_{x_i}^1(\mathbf{q}(t))) + Z(\mathbf{q}(t)), \quad (3.25)$$

$$Q(0_i, \mathbf{q}(t)) = \sum_n (pr_{x_i}(\mathbf{q}(t)) \cdot po_{x_i}^0(\mathbf{q}(t))) + Z(\mathbf{q}(t)), \quad (3.26)$$

where

$pr_{x_i}(\mathbf{q}(t))$ - the probability of the path section from the root vertex to the node x_i (set to one for the root vertex),

$po_{x_i}^1(\mathbf{q}(t))$ - the probability of the path section from the one branch of a node encoding x_i to a terminal 1 node (or the probability value of the node beneath the one branch of x_i),

$po_{x_i}^0(\mathbf{q}(t))$ - the probability of the path section from the zero branch of a node encoding x_i to a terminal 1 node (or the probability value of the node beneath the zero branch of x_i),

$Z(\mathbf{q}(t))$ - the probability of paths from the root vertex to the terminal 1 node that do not go through a node encoding x_i ,

n - all nodes encoding variable x_i in the BDD

By substituting Equations 3.25 and 3.26 into Equation 3.24, the criticality function for each event can be expressed as

$$G_i(\mathbf{q}(t)) = \sum_n pr_{x_i}(\mathbf{q}(t)) [po_{x_1}^1(\mathbf{q}(t)) - po_{x_1}^0(\mathbf{q}(t))] \quad (3.27)$$

The values of $pr[F]$, $po^1[F]$ and $po^0[F]$ (known generally as the 'path probabilities') are calculated during one depth-first pass of the BDD, during which the structure beneath branch 1 of any node is always fully explored before returning to consider branch 0. Starting with the root vertex, values of $pr[F]$ are assigned to each node as the branches are descended. Once the terminal node is reached, the procedure continues by working back up through the BDD calculating values of $po^1[F]$ and $po^0[F]$ for each of the nodes.

The calculation of the system unavailability can be performed simultaneously, as $po^1[F]$ is equivalent to the probability value of the node beneath branch 1 of F , and $po^0[F]$ is equivalent to the probability value of the node beneath branch 0. Therefore at each stage of the calculation, both the path probabilities and the terms of Equation 3.21 are evaluated.

The calculation procedure is demonstrated in the following section, by means of a worked example.

3.6.3 Worked Example

Consider the BDD shown in Figure 3.11. The calculations will be carried out to yield the system unavailability and unconditional failure intensity. There are four paths through the BDD that lead to a terminal vertex 1. There are three paths on branch 1 of node $F1$, i.e. $F1 - F2 - F3$, $F1 - F2 - F3 - F5$ and $F1 - F2 - F4$, and one path on branch 0 of node $F1$, i.e. $F1 - F5$.

The calculation results are presented in Table 3.1. The final column of this table shows the criticality values. This gives the correct criticality functions for variables 'a', 'b' and 'd' as they each appear only once in the BDD. However, as variable 'c' is encoded in two nodes, i.e. $F3$ and $F4$, their criticality values must be added to give the total criticality function for 'c'.

$$\begin{aligned} G_c &= q_b q_a (1 - q_d) + q_a (1 - q_b) \\ &= q_a (1 - q_b q_d) \end{aligned} \quad (3.28)$$

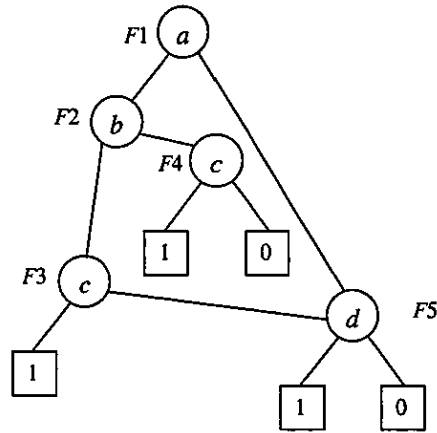


Figure 3 11: Example Binary Decision Diagram

Node	Variable	Probability value	pr	po^1	po^0	Criticality
F1	a	$q_a (q_b (q_c + (1 - q_c) q_d) + (1 - q_b) q_c) + (1 - q_a) q_d$	1	$q_b (q_c + (1 - q_c) q_d) + (1 - q_b) q_c$	q_d	$q_b (q_c + (1 - q_c) q_d) + (1 - q_b) q_c - q_d$
F2	b	$q_b (q_c + (1 - q_c) q_d) + (1 - q_b) q_c$	q_a	$q_c + (1 - q_c) q_d$	q_c	$q_a (1 - q_c) q_d$
F3	c	$q_c + (1 - q_c) q_d$	$q_a q_b$	1	q_d	$q_a q_b (1 - q_d)$
F4	c	q_c	$q_a (1 - q_b)$	1	0	$q_a (1 - q_b)$
F5	d	q_d	$1 - q_a + q_a q_b (1 - q_c)$	1	0	$1 - q_a + q_a q_b (1 - q_c)$

Table 3.1 Quantitative results for the BDD in Figure 3 11

The final stage of the analysis is to calculate the system unconditional failure intensity, which is given by:

$$\begin{aligned}
 w_{sys}(t) &= G_a w_a + G_b w_b + G_c w_c + G_d w_d & (3.29) \\
 &= w_a (q_c + q_b q_d (1 - q_c) - q_d) + w_b (q_a (1 - q_c) q_d) + w_c (q_a (1 - q_b q_d)) \\
 &\quad + w_d (1 - q_a + q_b q_a (1 - q_c)).
 \end{aligned}$$

The analysis so far has considered BDDs containing only basic events, but this could be extended to incorporate both complex events and modules.

3.6.4 Incorporating Complex Events and Modules into the Analysis

The following section describes the extension of the quantification methods to consider BDDs encoding complex events and/or modular events. The aim of the

analysis is to obtain not only the system unavailability and unconditional failure intensity, but to be able to extract the criticality function for the basic events that contribute to the complex events and modules. This is essential since, because reduction and modularisation may be used to simplify the original fault tree, it must be possible to analyse the system in terms of its original components. The approach was developed by Reay and Andrews [17].

3.6.4.1 Overview of the calculation procedure

The calculation process starts at the root vertex of the primary BDD and proceeds down through the branches, calculating the probabilities of the paths from the root vertex to each of the nodes. The unavailability of each encoded event is required as it enables the calculation of $pr[F]$ for the nodes beneath to be performed. Therefore the probabilities of both the complex and modular events are necessary for the analysis.

Values of $po^1[F]$ and $po^0[F]$ are calculated for the nodes on the way up through the primary BDD. If a node is encountered, that encodes either a complex or modular event, then the complex event or module must be analysed in order to assign appropriate values of $pr[F]$, $po^1[F]$ and $po^0[F]$ to its component nodes. This allows the calculation of the criticality functions of the basic events containing complex events and modules.

The criticality functions of basic events encoded within the primary BDD are calculated according to Equation 3.27 at the end of the analysis once the path probabilities of the nodes have been evaluated. The criticality functions of all basic events are then used together with their unconditional failure intensities to calculate the system unconditional failure intensity.

It is also possible to calculate $w_{sys}(t)$ by considering only the events encoded in the primary BDD. This would require both the criticality functions of any encoded modular and complex events and their unconditional failure intensities. Although these are relatively simple to calculate they are values that have no further use in the analysis. Instead the criticality functions of all basic events are calculated, which allows the analysis of the contributions to system failure through component or basic event importance measures.

The techniques for calculating the complex and modular event probabilities and the criticality functions of their constituent basic events are described in the following sections

3.6.4.2 Unavailability of complex and modular events

The probabilities of the complex events are calculated as they are formed, which ensures the process is as efficient as possible. Determining their probabilities is a straightforward procedure, as they are only a combination of two component events. The calculation depends on whether the events were combined under an 'AND' gate or an 'OR' gate, so for a complex event X_c that has constituent events X_1 and X_2 , the unavailability is given by

$$\text{'AND' gate } q_c = q_1 q_2, \quad (3.30)$$

$$\text{'OR' gate } q_c = q_1 + q_2 - q_1 q_2. \quad (3.31)$$

The probabilities of the modular events are not calculated before the quantitative analysis takes place, but are determined as and when required during the analysis (once a value has been calculated it is stored for later use). The calculation of the unavailability of a modular event is effectively that of finding the probability of the 'top event' of the module. A depth-first algorithm is used, which sums the probabilities of the disjoint paths through the module's BDD. If another modular event, x_i , is encoded within the module the algorithm identifies its root vertex, $M[x_i]$, and proceeds to call itself to calculate the required probability. Thus the unavailability of modules encoding only basic and complex events will necessarily be evaluated first

The calculation procedures for evaluating the probabilities of the complex and modular events are therefore relatively straightforward. At this stage they could be used alone to determine the system unavailability by performing the depth-first calculations on the primary BDD only. The calculation of the basic events' criticality functions does however require further analysis. This is discussed in the following sections

3.6.4.3 Criticality of basic events within complex events

Once the path probabilities have been calculated for a node encoding a complex event that complex event must be further analysed by assigning appropriate values

of $pr_{x_i}(\mathbf{q})$, $po_{x_i}^1(\mathbf{q})$ and $po_{x_i}^0(\mathbf{q})$ to its component events. Consider a node encoding the complex event, X_c , as shown in Figure 3.12.

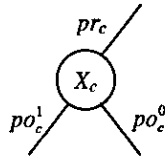


Figure 3.12: A complex event node within a BDD

The two events that combine to form this complex event are joined either by an 'AND' gate or an 'OR' gate, which gives the possible *ite* structures and corresponding BDDs as shown in Figure 3.13

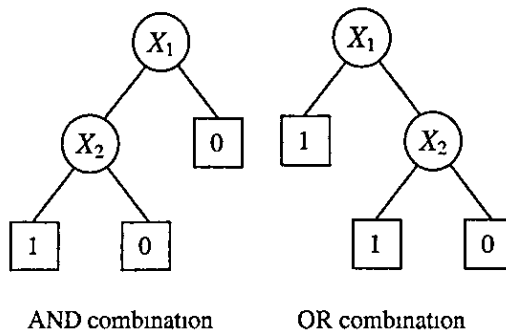


Figure 3.13: The possible BDD structures of a complex event

$$\text{'AND': } X_c = X_1 \cdot X_2 \quad (3.32)$$

$$X_c = \text{ite}(X_1, \text{ite}(X_2, 1, 0), 0)$$

$$\text{'OR': } X_c = X_1 + X_2 \quad (3.33)$$

$$X_c = \text{ite}(X_1, 1, \text{ite}(X_2, 1, 0))$$

The complex event node effectively replaces one of these structures in the original BDD (either the primary BDD or the BDD of a module). In order to evaluate the path probabilities of the nodes encoding these component events the terminal 1 vertices are simply replaced with the probability of the paths below branch 1 of the complex event node and the terminal 0 vertices are replaced with the probability of the paths below branch 0 of the complex event node. The probability of the paths preceding the root vertex does not have the usual value of one but takes

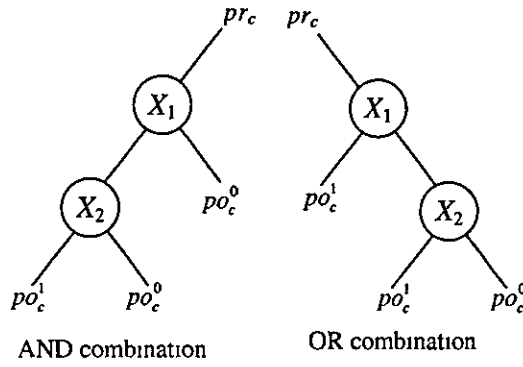


Figure 3 14: The complex event structure

the value of $pr[F]$ of the complex event node (pr_c). This is shown in Figure 3 14 Using Figure 3 14 the values of $pr_{x_i}(\mathbf{q})$, $po_{x_i}^1(\mathbf{q})$ and $po_{x_i}^0(\mathbf{q})$ can be calculated for the variables X_1 and X_2 . The resulting expressions are shown in Equations 3.34 - 3 45.

'AND' gate:

$$X_1 : pr_1 = pr_c \quad (3.34)$$

$$po_1^1 = q_2 \cdot po_c^1 + (1 - q_2) \cdot po_c^0 \quad (3.35)$$

$$po_1^0 = po_c^0 \quad (3.36)$$

$$X_2 : pr_2 = pr_c \cdot q_1 \quad (3.37)$$

$$po_2^1 = po_c^1 \quad (3.38)$$

$$po_2^0 = po_c^0 \quad (3.39)$$

'OR' gate:

$$X_1 : pr_1 = pr_c \quad (3.40)$$

$$po_1^1 = po_c^1 \quad (3.41)$$

$$po_1^0 = q_2 \cdot po_c^1 + (1 - q_2) \cdot po_c^0 \quad (3.42)$$

$$X_2 : pr_2 = pr_c \cdot (1 - q_1) \quad (3.43)$$

$$po_2^1 = po_c^1 \quad (3.44)$$

$$po_2^0 = po_c^0 \quad (3.45)$$

As the events X_1 and X_2 may be either basic events or other complex events this process is repeated until values have been calculated for all contributing basic events. The criticality functions of the basic events are then calculated according

to Equation 3.27.

Any complex event can appear more than once in the BDD, resulting in new values of $pr_{x_i}(\mathbf{q})$, $po_{x_i}^1(\mathbf{q})$ and $po_{x_i}^0(\mathbf{q})$ being calculated for its component events on each occasion. The criticality function for each of the contributing basic events must therefore be calculated in stages using the newly assigned values each time. Once this additional criticality value has been calculated for each of the contributing basic events it is added to the current value so that it is calculated as the analysis proceeds, rather than as a separate procedure at the end of the analysis as is the case for the basic events in the primary BDD.

3.6.4.4 Criticality of basic events within modules

Modular events are dealt with in a similar way to complex events. Once the path probabilities of the modular event are known the module is further analysed to determine the path probabilities of its component nodes. These probabilities must be assigned as they would have been had the module not been replaced by the single modular event. In order to do this the values of $po^1[F]$ and $po^0[F]$ of the modular event node replace any terminal 1 and 0 vertices within the module and the probability of the paths preceding the root vertex of the module is assigned the value of $pr[F]$ of the modular event node. This is shown in Figure 3.15. Unlike

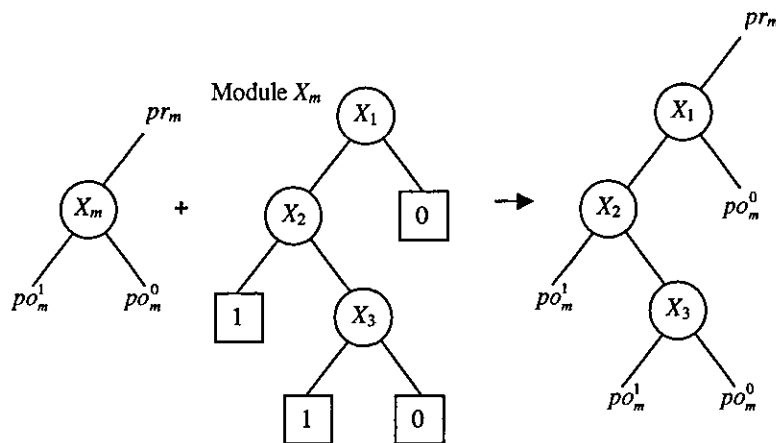


Figure 3.15: Replacing a modular event with the entire module structure

complex events the structure of modules is not fixed. They can obtain any number of events (basic, complex, or indeed other module events), connected by any number of gates. Therefore the path probabilities are assigned to the nodes by means

of a depth-first process, which is capable of dealing with any BDD structure. The method is very similar to that used for analysing a single BDD. The difference is that whenever a terminal node is encountered the probability of the paths below either branch 1 or branch 0 of the modular event node is used, rather than a terminal vertex probability values of one and zero. Obviously, $pr[F]$ of the root vertex will also be set to equal the probability of the paths preceding the modular event node.

As with complex events, the calculations required to obtain the path probabilities for the nodes within the module must be repeated for each occurrence of the modular event in the BDD. These values are used to calculate the additional contributions to the criticality functions of the basic events that arise due to the further occurrences of the modular event.

3.6.5 The Algorithm for Incorporating Complex Events and Modules into the Analysis

The analysis of the primary BDD is conducted in a similar manner to the analysis of single BDD structures, except for the processes instigated when a modular or complex event is encountered. As the probabilities of complex events are calculated as they are formed they are treated as basic events when descending the BDD. However, once the path probabilities have been evaluated for a complex event node Equations 3.34 - 3.45 are used to calculate the criticality functions of its constituent basic events.

If a modular event is encountered when descending the BDD the probability of the modular event is evaluated. When ascending the BDD a depth-first algorithm is used to calculate the criticality functions of the basic events that contribute to the module.

As the process for determining the path probabilities of the nodes within a module is so similar to the procedure used for dealing with the primary BDD a separate algorithm is not needed. The existing method is simply extended to include both options.

3.7 Summary

The BDD technique converts the fault tree to a Binary Decision Diagram, which represents the Boolean equation for the top event. The basic events of the fault tree need to be ordered before the conversion process takes place. The BDD construction process can be made more efficient if the fault trees are reduced and modularised beforehand. Both qualitative and quantitative analyses can be performed on the BDD. The minimal cut sets can be obtained if the BDD undergoes the minimisation process. Both analyses are extended to the BDDs encoding complex and modular events obtained after the reduction and modularisation processes. The extension to the qualitative analysis of BDDs encoding complex and modular events was conducted as a part of this research.

Chapter 4

Component Connection Method for Building BDDs

4.1 Introduction

A new construction method of BDDs from fault trees will be presented in this chapter. It will contain a description of connection and simplification rules with some alternative strategies for producing a BDD for any logic gate in a fault tree. The efficiency of the new algorithm will be measured by the computational time taken to convert a fault tree to a BDD, together with the size of the final BDD and the maximum required size of the dynamic memory data structure during the process.

4.2 Connection Process

This new proposed method, whose initial idea was presented in [18], for building BDDs encoding fault trees utilises the same *ite* structure, which was presented in Chapter 3, as the way to describe the structure function of the system. As a part of this research, a number of new ways for connecting nodes and merging BDDs during the conversion process will be presented. This method is an alternative to applying the *ite* technique for the conversion of fault trees to BDDs and its efficiency will be investigated.

Before the construction process starts, selection schemes for connecting gate inputs expressed as either basic events or BDDs needs to be established. Different ways for connecting basic events, gates and BDDs will be investigated later in this

chapter The basic event ordering as required in Rauzy's method does not necessarily need to be established here because the method can work without following any predetermined ordering scheme for the whole system

The following rules can be implemented to construct a BDD:

Rule 1. Each input X is assigned the ite structure $ite(X, 1, 0)$

Rule 2. If two inputs in a fault tree are inputs to an 'AND' gate, their representing nodes on a BDD are connected to each other through the 1 branch of the node. For an 'OR' gate the nodes are connected through the 0 branch.

Figure 4.1 shows the representation of BDDs for 'AND' and 'OR' gates with event inputs X and Y . These two rules explain the main idea of a new algorithm, pro-

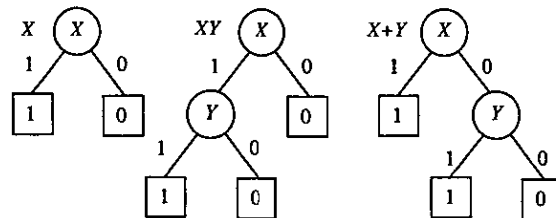


Figure 4.1. Example for the first and second connection rules

ducing BDDs for every gate

The last connection stage during building the final BDD contains one of two strategies depending on the way that the fault tree is traversed during the BDD construction process. There are two types of the traversal: the top-down and bottom-up approach.

In the **top-down** approach the BDD construction process is performed for a parent gate, before considering lower levels of a fault tree. Then a BDD, representing a gate input is constructed, a gate node is replaced by its BDD in the BDD structure of a parent gate. The process lasts until no gate nodes are left in the final BDD.

The **bottom-up** technique considers gate inputs to any parent gate in a left-right way so that a BDD for a subtree of the left-most gate is built before considering

the remaining gate inputs. Then all BDDs, representing gate inputs of a parent gate, are merged to obtain the BDD of the parent gate. The process is over when the BDDs, representing gate inputs of the top-event, are combined.

Since the nature of the connection process is different for those two approaches, the third rule is explained for 'AND' and 'OR' gates respectively for each approach.

Rule 3.

Top-down approach.

Rule 3a. During the development of a gate node, which represents an output to an 'AND' gate, a structure representing this gate in terms of its inputs is inserted into the BDD. The BDD structure on the 0 branch of the gate node in the original BDD is reconnected to every terminal 0 node of the inserted BDD. The single terminal 1 node in the new AND structure is connected to the connection on the 1 branch of the original node it replaced.

Rule 3b. During the development of a gate node, which represents an output to an 'OR' gate, a structure representing this gate in terms of its inputs is inserted into the BDD. The BDD structure on the 1 branch of the gate node in the original BDD is reconnected to every terminal 1 node of the inserted BDD. The single terminal 0 node in the new OR structure is connected to the connection on the 0 branch of the original node it replaced.

Bottom-up approach. If there are two BDDs, which represent two gate inputs of a parent gate, one of them is set to be the main BDD, according to the rule of selection.

Rule 3a. If two BDDs are inputs to an 'AND' gate, the secondary BDD is connected to every terminal 1 node of the main BDD.

Rule 3b. If two BDDs are inputs to an 'OR' gate, the secondary BDD is connected to every terminal 0 node of the main BDD.

The application of the third rule for the top-down and bottom-up approaches is

explained and applied to an example fault tree, presented in Figure 4 2. The top-

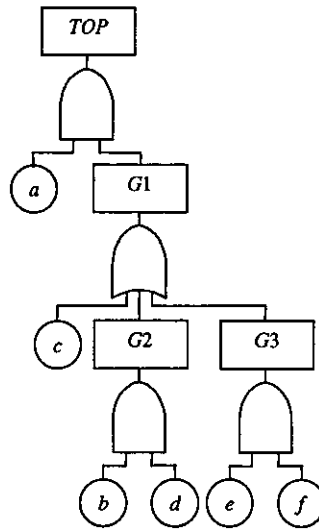


Figure 4 2: Example fault tree

down process for the example in Figure 4 2 is illustrated in Figure 4 3, considering the left-right variable ordering for every gate in a fault tree, i.e.

$$\begin{aligned}
 \text{Top} & . a < G1, \\
 G1 & : c < G2 < G3, \\
 G2 & . b < d, \\
 G3 & : e < f.
 \end{aligned}$$

First of all, the inputs of the top event, a and $G1$, are assigned **its** structures with terminal vertices and then they are connected to each other through the 1 branch of node a , as shown in Figure 4 3(i). This results in a BDD with gate node $G1$, which is replaced by the BDD of gate $G1$. The replacement for $G1$ is formed by connecting the $G1$ inputs c , $G2$ and $G3$ in an 'OR' chain and results in the BDD shown in Figure 4.3(ii). This BDD contains two gate nodes $G2$ and $G3$. The replacement of gate node $G2$ by its BDD gives the structure presented in Figure 4 3(iii). The structure beneath the 0 branch of gate node $G2$ is reconnected to every available 0 branch after the replacement. Finally, gate node $G3$ is replaced resulting in a BDD which consists of only basic events, as shown in Figure 4 3(iv). The BDD construction process is finished. Different strategies can be described for the selection of the order of inputs of each parent gate before the connection

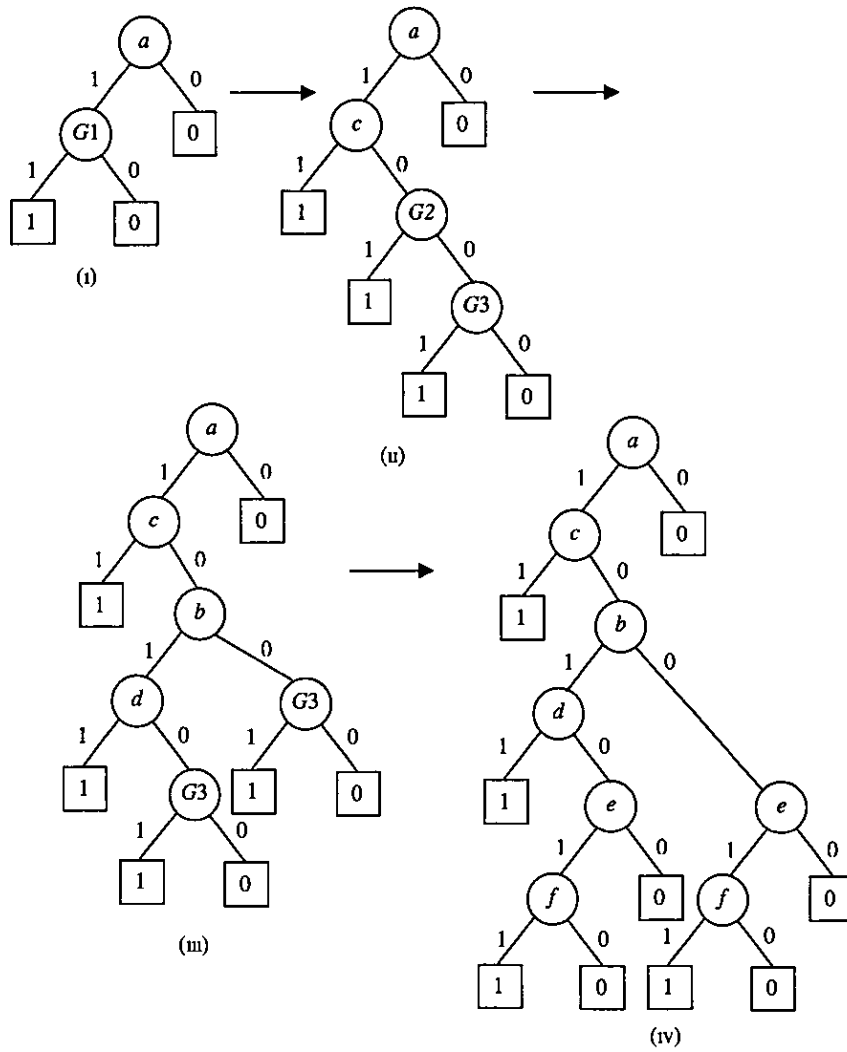


Figure 4.3: Process of top-down strategy

process. This will be discussed later in this chapter.

The application of the bottom-up technique for the fault tree example in Figure 4.2 is presented in Figure 4.4. In this example, it has been set that when combining BDDs representing inputs for gates they are considered in a left-right manner and the left-most BDD is set to be the main BDD to which the other is joined. The left-right variable ordering for every gate in a fault tree is considered, as it was presented for the top-down scheme. First of all, gates G_2 and G_3 are constructed, shown in Figure 4.4(i) and Figure 4.4(ii), building two BDDs, which are both 'AND' chains. Then gate G_1 is considered, creating a BDD for its basic

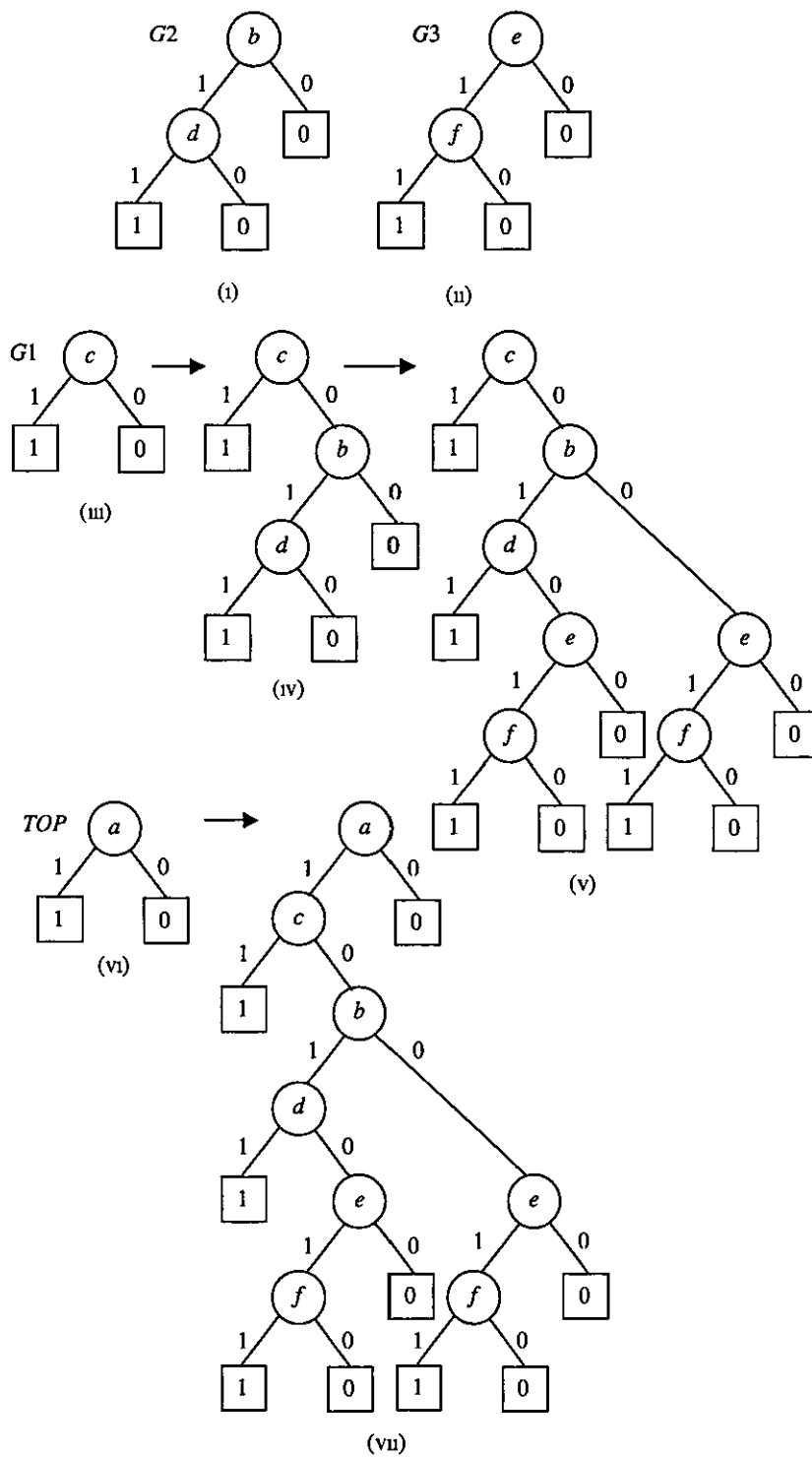


Figure 4.4 Process of bottom-up strategy

event c in Figure 4.4(*iii*) and establishing it to be the main BDD in the process. Two BDDs from Figure 4.4(*i*) and Figure 4.4(*ii*) are connected one by one on every available 0 branch of the main BDD. Connecting the BDD of gate $G2$ on the only 0 branch of the main BDD results in the BDD with two available 0 branches, shown in Figure 4.4(*iv*), where the BDD of gate $G3$ is connected, as shown in Figure 4.4(*v*). Finally, the top event is investigated, building a BDD for its basic event Figure 4.4(*vi*) and connecting the BDD of gate $G1$ on the 1 branch of the main BDD. The final BDD is presented in Figure 4.4(*vii*).

This completes the connection process. However, some simplification rules need to be applied prior to analysis if there are any repeated events in a fault tree.

4.3 Rules of Simplification

After every connecting operation the repetition of basic events on any path through the BDD are checked. If there is at least one repeated event, two simplification rules will be applied:

- Each path starting at the node that represents the first occurrence of a repeated event in a path, and proceeding to a terminal vertex, must be adjusted in order to avoid the contradictory states of the event in the BDD. The node, that represents the second occurrence of the event, needs to be replaced by the events below it on either its 'working' branch or 'failed' branch depending on the component state as specified by its first occurrence in the path. For example, if the traverse of the BDD starts with the 1 branch of a node, the second appearance of that node should be replaced by the BDD structure below the 1 branch of this second node for consistency.
- If the state of the system is the same regardless of the basic event occurrence or non-occurrence, the insignificant vertex must be removed. In other words, if the BDD structures below both branches of the node are the same, the node needs to be replaced by the structure below one of the branches.

In terms of Boolean Laws of Algebra during the simplification process the idempotent, complementation, identity and distribution laws will be applied. Some examples explain the rules of simplification in detail. Figure 4.5 presents the first simplification rule. Traversing through the BDD produces the path $F1 - F2 - F3 - F4$. The only repeated event in this path is a , which is represented by node $F1$ and

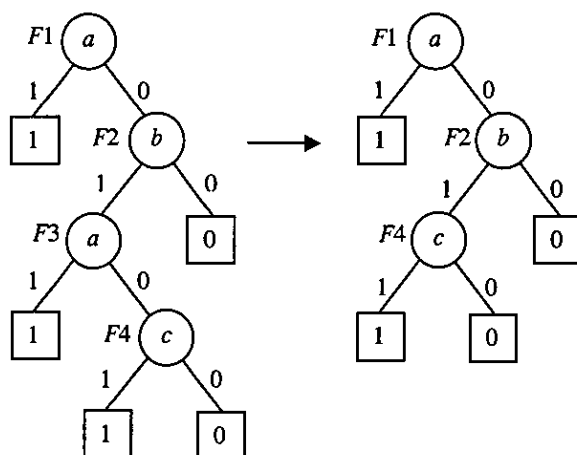


Figure 4.5 Example of the first simplification rule

F3. Node *F3* needs to be replaced. Since the path traverses the 0 branch of node *F1*, *F3* is replaced by node *F4* which can be reached by traversing the 0 branch of node *F3*. In this way redundancies are removed from the Boolean function BDD structure, $f(\mathbf{x})$. The Boolean function of the first BDD in Figure 4.5 is reduced to the second one:

$$\begin{aligned} f(x) &= a + \bar{a} \cdot b \cdot a + \bar{a} \cdot b \cdot \bar{a} \cdot c \\ &= a + \bar{a} \cdot b \cdot c, \end{aligned} \quad (4.1)$$

because

$$\bar{a} \cdot a = 0 \text{ (complementation),} \quad (4.2)$$

$$b \cdot 0 = 0 \text{ (identity),} \quad (4.3)$$

$$\bar{a} \cdot \bar{a} = \bar{a} \text{ (idempotent)} \quad (4.4)$$

The second rule is shown in Figure 4.6. The system failure does not depend on the failure or success of event a , i.e. the system will fail if b and c fail. Therefore, node *F1* is replaced by one of its branches, for example, *F2* - *F3*. Applying Boolean Laws to the logic function of the BDD gives

$$\begin{aligned} f(x) &= a \cdot F + \bar{a} \cdot F \\ &= (a + \bar{a}) \cdot F \\ &= F, \end{aligned} \quad (4.5)$$

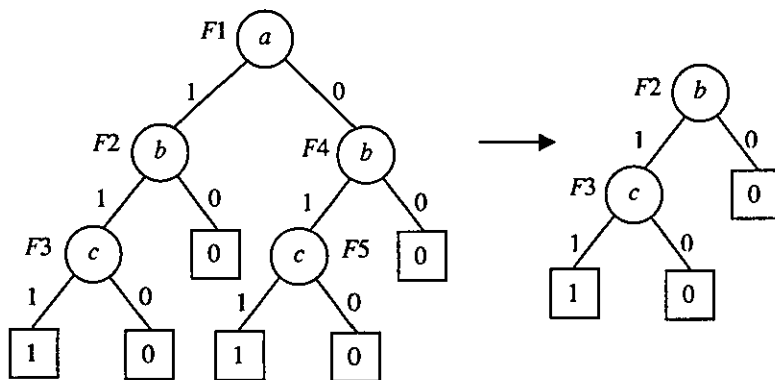


Figure 4.6: Example of the second simplification rule

where distributive, complementation and identity laws were used

The second rule is also applied to the case when the node has two identical terminal vertices. This situation might appear after applying the first rule if the required branch of the node with the second repeat leads to a terminal vertex. This case is shown in Figure 4.7

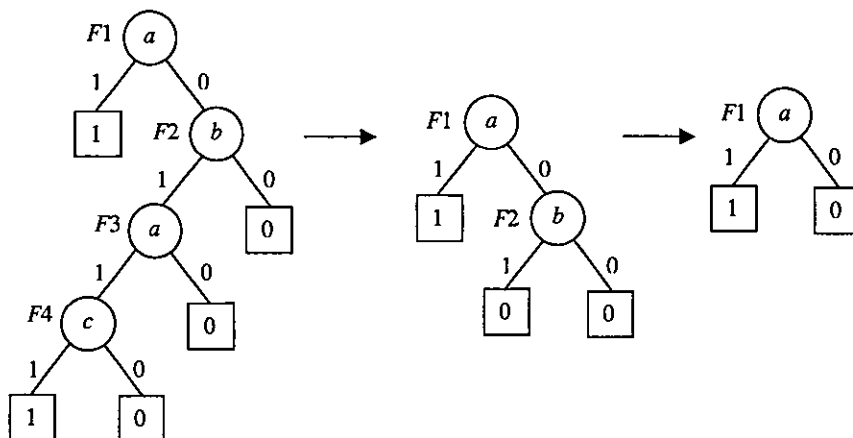


Figure 4.7: Example of both simplification rules

In this situation $F3$ needs to be replaced by the terminal 0 which gives a situation of a node with two identical terminal vertices. Therefore node $F2$ needs to be

removed This situation can be presented

$$\begin{aligned}
 f(x) &= a + \bar{a} \cdot b \cdot a \cdot c & (4.6) \\
 &= a.
 \end{aligned}$$

4.4 Properties of the BDD Using the Component Connection Method

BDDs constructed using the Component Connection Method preserve the main characteristics as BDDs obtained using the *ite* method, i.e. since the BDD encodes the structure function of the fault tree and paths through the BDD are disjoint, both qualitative and quantitative analyses can be carried out, as it was presented in Chapter 3

For example, consider the BDD in Figure 4.4(vii). For the qualitative analysis the BDD needs to be minimised using Equation 3.19 and then every path to a terminal vertex 1 describes a minimal cut set. The minimised BDD is shown in Figure 4.8. This minimal disjunctive form of the logic equation gives three minimal

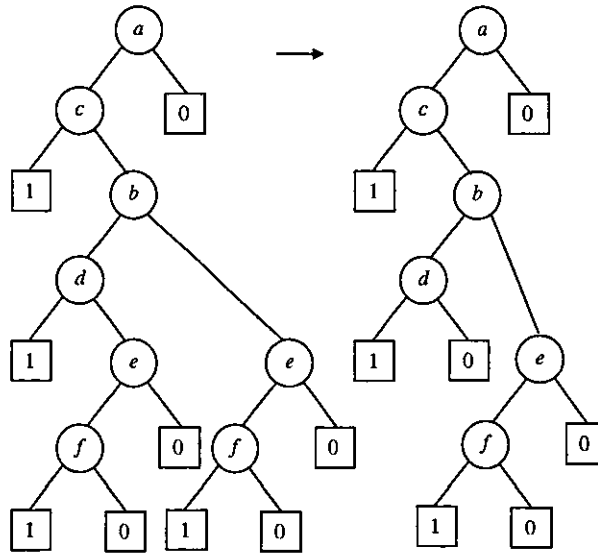


Figure 4.8: Minimised BDD from Figure 4.4(vii)

cut sets

$$\{a, c\}, \{a, b, d\}, \{a, e, f\} \quad (4.7)$$

The quantitative analysis for the BDD in Figure 4.4(vii) is also considered. According to the Equation 3.21, the probability of system failure can be calculated as a sum of probabilities of every disjoint path in the BDD. In this case

$$Q(t) = q_a q_c + q_a(1 - q_c)q_b q_d + q_a(1 - q_c)q_b(1 - q_d)q_e q_f + q_a(1 - q_c)(1 - q_b)q_e q_f. \quad (4.8)$$

So, the qualitative and quantitative analyses of BDDs obtained using the Component Connection Method can be applied in the same way as it was shown for the *ite* technique.

However, BDDs constructed using the Component Connection Method have some different properties. For example, BDDs are not ordered. A specified ordering of basic events is not required for the method and elements in the BDD appear in the order that they were considered in the construction process.

Also, the Component Connection Method does not use the sub-node sharing method, therefore, there are some parts in the BDD structure that are repeated but not shared. For example, in Figure 4.4(vii) the structure beneath the zero branch of node *b* is identical to the structure beneath the zero branch of node *d*. In the *ite* method the zero branches of nodes *b* and *d* would be sharing the same structure. The difference of both techniques in the sub-node sharing property is shown in Figure 4.9. This property of the Component Connection Method might lead to a very inefficient memory usage. The extension of the Component Connection Method introducing the sub-node sharing is presented in the Sub-node Sharing section of this chapter.

4.5 Measures of Efficiency

The order which gate inputs of a fault tree are considered to produce a gate BDD (top-down approach) or the order BDDs are considered for the combination (bottom-up approach) can have a crucial effect on the size of the final BDD. Therefore, efficiency measures are defined in order to be able to compare the different strategies proposed later in this chapter. Using the obtained information some indications of optimum selection schemes for the chosen method can be provided.

The size of the final BDD, in terms of the total number of nodes in the BDD,

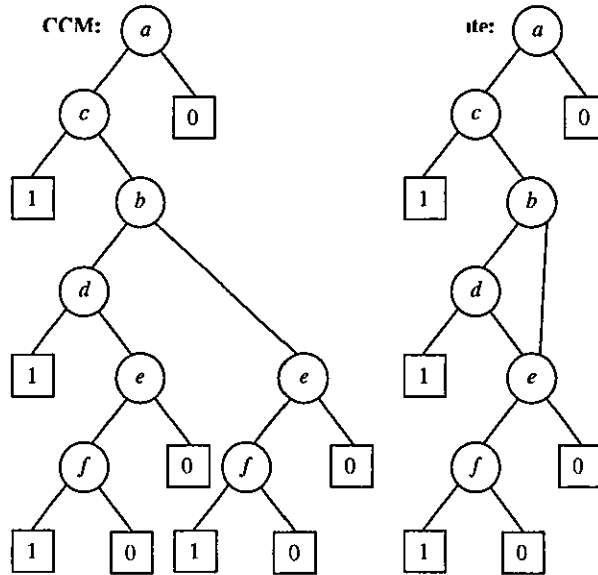


Figure 4.9: Difference of the two techniques, the sub-node sharing method

is the most important measure of efficiency. This measure helps to identify how different conversion approaches increase the size of the BDD. Despite the accuracy and the efficiency of the BDD method itself, sometimes the analysis of the system still cannot be performed if the conversion process of fault trees to BDDs is time-consuming. Hence *the computer run time*, measured in seconds, is another important measure of efficiency that is taken into account. Also, if it is required, the computational time of qualitative and quantitative analyses can be taken into account.

In some situations, where the size in terms of the number of nodes cannot be improved using a different strategy, *the maximum required size of array* is measured. This number is the size of the array required in the construction process. It is different from the final size of the BDD because along with the number of actual nodes of the BDD it also contains the number of nodes that were created during the conversion process but are not required anymore. This measurement is particularly useful for the efficiency analysis of the Hybrid method, presented later in this chapter.

The order in which the elements are combined to form the BDD must be selected so that all these measures are minimised as much as possible. Ideally, the optimum selection scheme should enable a BDD of the smallest size, with a minimum re-

quired array size in the shortest time. Further research contains the investigation of different approaches and orderings of elements in the BDD construction process.

4.6 Selection Schemes in the Component Connection Method

In order to convert a fault tree to a BDD by applying the Component Connection Algorithm presented, it is needed to know the

- Traverse approach (top-down or bottom-up)
- Order of basic events selection
- Order of inputs selection for the top-down technique
- Order of BDDs selection for the bottom-up technique

The traverse approach, either the top-down or the bottom-up method, can be chosen to build a BDD. Both strategies will give the same final BDD as long as the ordering parameters selected remain consistent. However, the big advantage of the bottom-up technique against the top-down is that a smaller memory resource is required. This is because with the bottom-up method the fault tree is investigated 'in portions', i.e. building a BDD for the left-most gate input is finished and simplifications applied before the investigation of other inputs. In the top-down technique the whole set of inputs for the top event is connected and then every gate node is replaced, still trailing the rest of the structure until the last gate node is replaced. This might have a crucial effect on big fault trees and even cause memory capacity problems.

4.6.1 Order of Basic Events Selection

4.6.1.1 Order as listed process

The simplest way to select basic events is to connect them according to the order that they appear in the list of gate inputs. This strategy is easy to use because basic events can be connected in the same way as they are presented without using any processing to determine the order of events for each gate. The BDDs for gates G_2 and G_3 in Figure 4.3 or in Figure 4.4 were built using this rule.

4.6.1.2 Defined ordering

Another way to select basic events is to use *an ordering scheme* for the whole fault tree, which is determined by traversing a fault tree in a chosen structured way. If the ordering of basic events in the system is $a < b < c < d \dots$, once the conversion process of a gate, which contains event inputs, to a BDD takes place, the nodes on the resulting BDD will be considered according to this ordering, example in Figure 4.10. The traverse of the BDD gives the ordering $a < b < c$. Eight

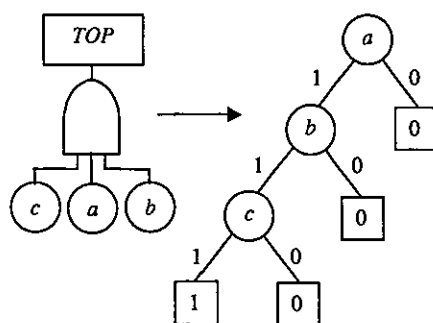


Figure 4.10: Example of ordered traversal of the tree structure

ordering schemes, some previously considered in [5] for BDD construction using the *ite* method, will be presented and their algorithms will be demonstrated on the example fault tree, shown in Figure 4.11. The eight ordering schemes are:

1. Modified top-down ordering
2. Modified depth-first ordering
3. Modified priority depth-first ordering
4. Depth-first, with number of leaves ordering
5. Non-dynamic top-down weighted ordering
6. Dynamic top-down weighted ordering
7. Bottom-up weighted ordering
8. Event criticality ordering

The names and the algorithms of the ordering schemes are retained from the original work. In this research the eight ordering schemes were chosen to test different conversion techniques and give a range of results for the efficiency analysis. The

modified methods (which give priority to repeated events in the ordering) were chosen because of their efficiency while converting fault trees to BDDs. These ordering approaches will be demonstrated by application to the fault tree shown in Figure 4.11

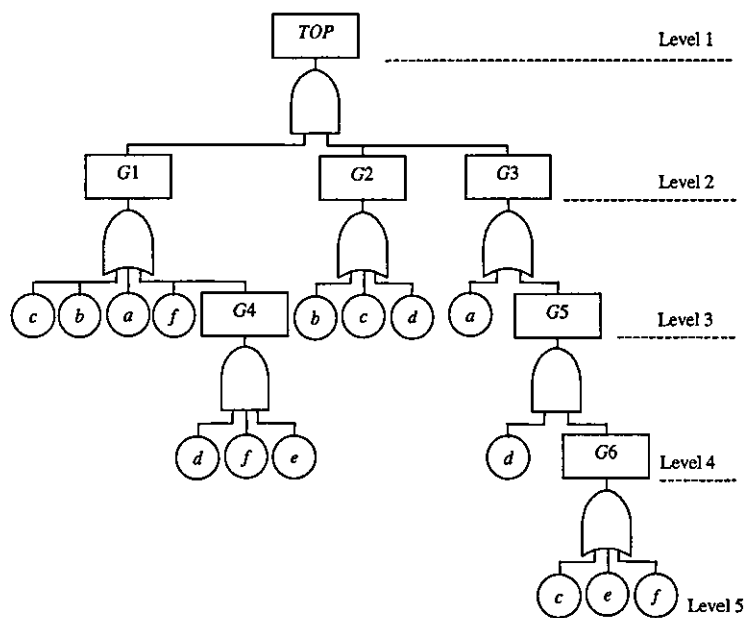


Figure 4.11: Example fault tree for different orderings

4.6.1.2.1 Neighbourhood ordering schemes

Neighbourhood ordering schemes are very commonly used. They order the variables during a systematic traverse of the fault tree. These schemes are likely to keep the neighbourhoods of the variables, therefore, events appearing close together in the fault tree structure are also close in the ordering and therefore in the BDD structure. The first neighbourhood ordering heuristic to be presented was the depth-first ordering scheme, which was applied by Rauzy [2] in his introductory article on using the BDD technique for Fault Tree Analysis. Four neighbourhood schemes are presented below.

1. Modified top-down ordering

The top-down ordering scheme is the most simple technique, ordering the variables as they appear on a top-down, left-right traversal of the fault tree structure.

Basic events from the higher levels of the fault tree will be allocated earlier in the ordering than those from the lower levels of the fault tree. According to the modified top-down scheme variables appearing on the same level of the fault tree are ordered according to their total number of occurrences in the fault tree, allocating those with higher occurrence first. If there is more than one event with an equal number of occurrences, then those events are ordered as they appear from left to right on that level. Each variable is placed in the ordering the first time it appears during the traverse; subsequent occurrences are ignored.

This method is demonstrated by application to the example fault tree, shown in Figure 4.11. There are no events to order on the first two levels, so the ordering process starts on the third level. Five basic events appear on this level, which consideration from left to right is: c , b , a , f and d . They need to be ordered according to their number of occurrences in the fault tree. Events c , f and d appear the same number of times (three occurrences), therefore, they remain in the left to the right order in which they are placed. In fact, events b and a both occur two times in the fault tree, so they are ordered after the other three events in the left to right order. The partial ordering for the third level is:

$$c < f < d < b < a \quad (4.9)$$

Level four is now considered and the only event appearing on this level that has not been ordered yet is event e . The ordering then becomes

$$c < f < d < b < a < e. \quad (4.10)$$

Level five is not considered, as all the events have already been placed in the ordering.

2. Modified depth-first ordering

Depth-first ordering considers the fault tree as being made up of many smaller subtrees, with each subtree ordered in a top-down left-right manner. The left-most gate is always completely explored before considering the remaining gate inputs, therefore, the lower levels of left-most subtrees are considered before higher levels of other subtrees. Any basic event inputs to a gate are ordered before the gate inputs. According to the modified depth-first technique the basic events with the greatest number of occurrences are ordered first, but if there are two or more

variables with the same number of occurrences they are ordered as they appear from left to right in the structure.

The ordering scheme can be applied to the example fault tree in Figure 4.11. Since the top event has no event inputs, its three gate inputs, $G1$, $G2$ and $G3$, are investigated in turn. The leftmost gate $G1$ is considered first. Gate $G1$ has four event inputs so they are considered before gate input $G4$. Events c and f are the most repeated events (each appearing three times), therefore they appear before b and a in the ordering, which both occur twice. Event c is ordered before f as it occurs leftmost in the list of inputs and event b appears before a for the same reason. This gives the partial ordering

$$c < f < b < a \quad (4.11)$$

Gate $G4$ is considered next, which consists of three events, d , f and e . Events d and e have not been ordered yet, whereas event f has. Event d appears a greater number of times (three occurrences) than event e (two occurrences) therefore, event d is ordered before e . This gives the final ordering:

$$c < f < b < a < d < e. \quad (4.12)$$

All the events have been considered, so it is unnecessary to consider gates $G2$ and $G3$.

3. Modified priority depth-first ordering

The priority depth-first scheme is an extension to the basic depth-first ordering, where rather than simply ordering the gate inputs from left to right, any gates which have only basic events as inputs are given preference. The modified version of this method orders basic events according to the number of occurrences, such that the most repeated event has a priority. If there is more than one variable with the same number of occurrences, then those variables with equal number of occurrences are ordered according to their appearance in the fault tree. Events continue to be considered before any gate inputs.

This ordering method can be applied to the example fault tree (Figure 4.11) in a similar way to the modified depth-first scheme. The top event has no event inputs, so its three gate inputs, $G1$, $G2$ and $G3$, are considered in turn. As gate

G_2 has no gate inputs, it is explored first, i.e. the gate inputs to the top event are considered in the order G_2 , G_1 and G_3 . Gate G_2 contains the events b , c and d . The most repeated events are c and d (three times), therefore they are ordered according to their appearance in the list of outputs. This gives the ordering

$$c < d < b \quad (4.13)$$

The subtree of the next gate G_1 is explored. Its event inputs are considered before gate input. It has two unordered event inputs a and f . They are placed in the ordering with f first, because this event appears the greater number of times (three occurrences) than event a (two occurrences). This results in the ordering

$$c < d < b < f < a \quad (4.14)$$

Then gate input G_4 is investigated and the last unordered event e is placed in the ordering, to give the final result:

$$c < d < b < f < a < e. \quad (4.15)$$

All the events have now been considered, therefore no remaining gates need to be investigated.

4. Depth-first ordering, with number of leaves

This scheme is an extension to the modified depth-first ordering, with a different way of choosing the order in which gate inputs are considered. They are chosen according to the number of 'leaves' beneath the gate itself. The number of leaves of a gate is the total number of basic events occurring beneath that gate. The gates with the smallest number of leaves that have not yet been considered are ordered first. If there are at least two gates with the same number of leaves, the gate with the fewest ordered leaves is chosen earlier. If an order still can not be decided, then the gates are simply ordered as they appear from left to right in the input list. Events are ordered in the same way as in the modified depth-first method, thus the most repeated events are chosen first. Again, basic events are considered before any gate inputs.

This technique is applied to the example fault tree shown in Figure 4.11. The top event has no basic event inputs to order, so its gate inputs G_1 , G_2 and G_3 are considered in turn. The number of leaves, shown in Table 4.1, determines the order

	$G1$	$G2$	$G3$
Number of unordered leaves	7	3	5
Number of ordered leaves	0	0	0

Table 4 1: Number of ordered and unordered leaves of gates $G1$, $G2$ and $G3$

in which they are explored. Since $G2$ has the fewest number of unordered leaves, it is considered first, followed by $G3$, then $G1$. $G2$ consists of three events b , c and d , which gives the partial ordering, following the rules of the modified depth-first method

$$c < d < b. \quad (4.16)$$

The subtree of the next gate, $G3$, is now considered. Its only unconsidered event input a is placed in the ordering

$$c < d < b < a. \quad (4.17)$$

Then its only gate input $G5$ is investigated. Since event d has already been ordered, the process is carried out on gate input $G6$. There are two unordered events e and f . Since event f occurs the greater number of times (three occurrences) compared to event e (two occurrences), event f is allocated before event e in the ordering:

$$c < d < b < a < f < e. \quad (4.18)$$

This concludes the ordering, so gate $G1$ is not examined.

4.6.1.2.2 Weighted ordering schemes

Weighted ordering schemes assign weights to the events, which are then used to determine their position in the ordering. These methods do not necessarily keep neighbourhoods in the same way as neighbourhood ordering schemes, so variables that appear together in the fault tree structure may not be close in the ordering. There are two categories of weighted ordering schemes: topological schemes, which determine weights according to the positions of the variables in the fault tree, and schemes based on importance measures, which determine weights in a way that is not dependent on the manner the fault tree is written. Four weighted methods are presented below. Three of them are topological methods using opposite ends

of the fault tree to initiate the weighting process with non-dynamic and dynamic approaches to calculate the weights and one of them is based on the Birnbaum's structural importance measure

5. Non-dynamic top-down weighted ordering

The calculation of the weights in this method starts by allocating a weight of 1.0 to the top event and then distributing the weight at each gate equally between its inputs. For the repeated events their corresponding weights are added together. After each basic event has been assigned a weight, the variables are placed in order of decreasing weight. If two or more events have equal weights, their order is considered according to their average level of appearance in the fault tree. The average level is worked out by summing the levels on which the event appears and dividing this sum by the number of occurrences. Then the event that appears, on average, on the highest level in the fault tree is allocated earlier in the ordering. If there is more than one event with the same characteristic the most repeated event is chosen. If the events still remain indistinguishable then they are ordered as they appeared in the modified top-down ordering.

Figure 4.12 shows the example fault tree from Figure 4.11 with the assigned weights for every gate and event.

Weights can be obtained for every variable:

$$a = 1/15 + 1/6 = 7/30 = 42/180 \quad (4.19)$$

$$b = 1/15 + 1/9 = 8/45 = 32/180 \quad (4.20)$$

$$c = 1/15 + 1/9 + 1/36 = 37/180 \quad (4.21)$$

$$d = 1/45 + 1/9 + 1/12 = 39/180 \quad (4.22)$$

$$e = 1/45 + 1/36 = 9/180 \quad (4.23)$$

$$f = 1/15 + 1/45 + 1/36 = 21/180. \quad (4.24)$$

The events can now be ordered by decreasing weights. There are no events with equal weights, therefore the summed weights simply give the ordering

$$a < d < c < b < f < e \quad (4.25)$$

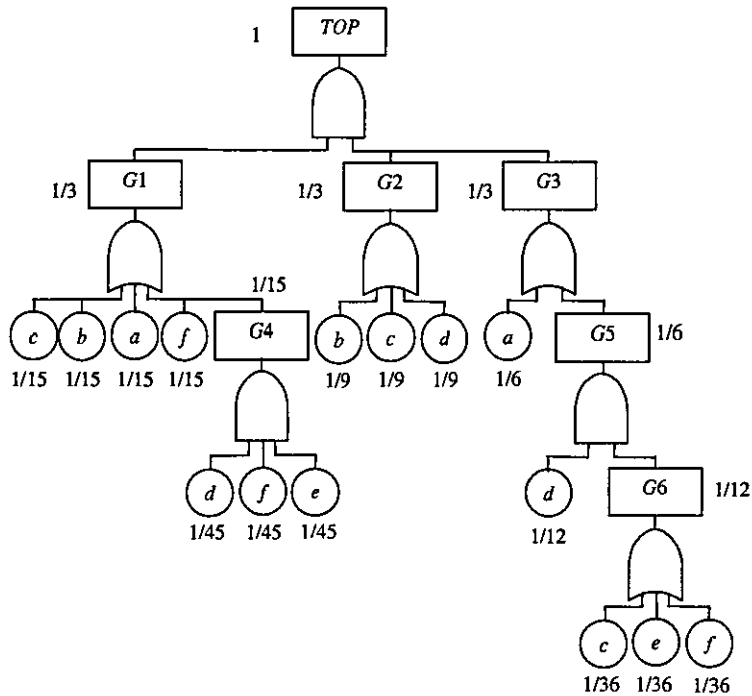


Figure 4.12: Assigned weights for gates and events

6. Dynamic top-down weighted ordering

This method follows the same rules as in the non-dynamic top-down weighted scheme, but once an event has been allocated in the ordering, it is removed from the fault tree by deleting all its occurrences. Using this modified fault tree, weights are reassigned from the beginning. This results in the selection of another event and the process continues until all events have been ordered. Applying the dynamic ordering method means that in many cases neighbouring events appear close in the final order.

Applying this method to the example fault tree gives the first set of weights as in the non-dynamic ordering. This means that event *a* is the first to be allocated in the ordering. Then *a* is removed from the fault tree to give the modified tree, which is shown in Figure 4.13. The new weights are.

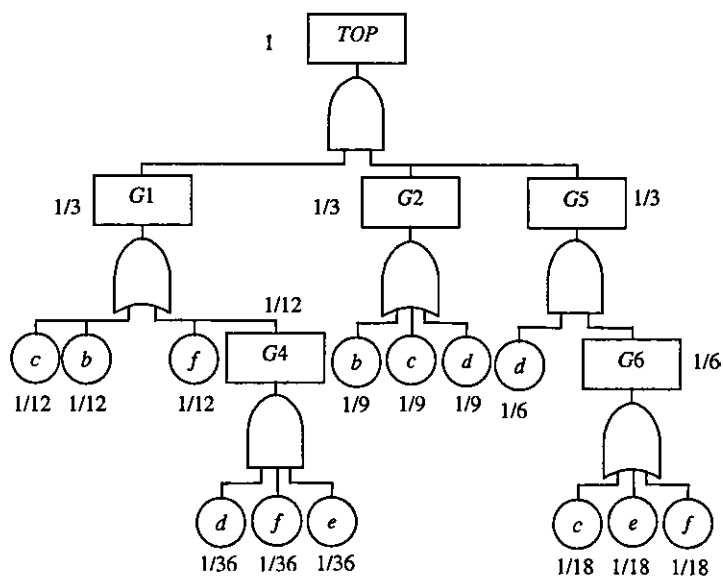


Figure 4.13: Modified example fault tree after event a has been removed

$$b = 1/12 + 1/9 = 7/36 \quad (4.26)$$

$$c = 1/12 + 1/9 + 1/18 = 9/36 \quad (4.27)$$

$$d = 1/36 + 1/9 + 1/6 = 11/36 \quad (4.28)$$

$$e = 1/36 + 1/18 = 3/36 \quad (4.29)$$

$$f = 1/12 + 1/36 + 1/18 = 6/36 \quad (4.30)$$

Event d has the largest weight value, so it is placed after event a in the ordering. Continuing the same algorithm, event d is removed from the fault tree and further weights calculated. This process is repeated until all events have been ordered. There were no situations with more than one variable with the largest weight value, therefore events were simply ordered according to their weight values. The final dynamic top-down weighted ordering is

$$a < d < c < b < f < e \quad (4.31)$$

7. Bottom-up weighted ordering

This technique starts from the bottom of the fault tree, rather than the top and in effect calculates weights for the gates, which are then used to determine the ordering in which they are considered within a depth-first exploration. First of all, a weight of $1/2$ is assigned to each basic event. Then the weights of the gates are

combined as 'probabilities' according to the type of the gate

$$\text{'AND' gate: } P(\text{gate}) = \prod_{i=1}^n q_i, \quad (4.32)$$

$$\text{'OR' gate: } P(\text{gate}) = 1 - \prod_{i=1}^n (1 - q_i), \quad (4.33)$$

where n is the number of inputs to the gate

Once each of the inputs to the top event has been assigned weights, the tree is explored in the modified depth-first manner, considering branches with the largest weight first. If gates have the same weight then they are considered according to the percentage of repeated events below that gate. This is calculated by summing the number of repeated events and dividing by the total number of events below that gate. The gate with the highest percentage of repeated events is considered first. But if the percentage is the same for at least two gates they are considered from left to right as they appear in the fault tree. The events of each gate are ordered before the gate inputs are explored and are ordered according to the highest number of occurrences in the fault tree. If events occur the same number of times then they are simply ordered from left to right as they appear in the input list.

This method can now be applied to the example fault tree. First of all, every event is given a weight of $1/2$ and so the weights of the gates are calculated following Equations 4.32 and 4.33 and presented in Table 4.2. The top event has

Gate name	Gate type	Inputs	Calculated gate weight
$G1$	OR	$G4, c, b, a, f$	$121/128$
$G2$	OR	b, c, d	$7/8$
$G3$	OR	$G5, a$	$23/32$
$G4$	AND	d, f, e	$1/8$
$G5$	AND	$G6, d$	$7/16$
$G6$	OR	c, e, f	$7/8$

Table 4.2: Weights for gates

three gate inputs, $G1$, $G2$ and $G3$. They are considered in order of highest weight

according to Table 4 2. In this case it gives that those gates are explored in the same order as they appear in the list, i.e. $G1$, $G2$ and $G3$. Gate $G1$ has four event inputs and following the rules of the modified depth-first method gives the partial ordering:

$$c < f < b < a. \quad (4\ 34)$$

Then gate input $G4$ is examined and two unordered events d and e are placed in the ordering. Event d goes first because its number of occurrences is higher (three times) than the number of occurrences for event e (two times). This gives the final ordering:

$$c < f < b < a < d < e. \quad (4\ 35)$$

The subtrees of gate $G2$ and $G3$ are not investigated because all the events have been ordered.

8. Event criticality ordering

This method applies the principle of Birnbaum's structural importance measure directly to the fault tree. The contribution of each basic event to the occurrence of the top event is calculated according to

$$I_i = Q(1_i, 1/2) - Q(0_i, 1/2), \quad (4\ 36)$$

where

- $Q(1_i, 1/2)$ is the top event probability with a failure probability of 1 for element i and failure probabilities of $1/2$ for the remaining components and
- $Q(0_i, 1/2)$ is the top event probability with a failure probability of 0 for element i and failure probabilities of $1/2$ for the remaining components

The top event probability calculation for this method needs to be fast and approximations are acceptable as it is only used to fix the variable ordering. As such all basic events and gates are assumed to be independent and the probabilities worked up through the fault tree.

The selected basic event assumes the failure probabilities of 1 and 0 on two consecutive computations of the top event probability, with the remaining components

being given failure probabilities of $1/2$. The result of the second traversal (with a failure probability 0) is subtracted from the first traversal (with failure probability 1) to give Birnbaum's measure of importance for that component. The basic events are ordered allocating those with a greater contribution before those with smaller contributions. If two events have the same contribution, then the event with the highest average level of occurrence is ordered first. If the events have the same highest average level the most repeated event is selected first and if they are still indistinguishable then they are simply ordered as they appear in the modified top-down ordering.

This scheme can be applied to the example fault tree. The contributions are shown in Table 4.3. The events are placed in the ordering such that those with

Event	Probability of system failure with event failure probability 1	Probability of system failure with event failure probability 0	Contribution to system failure
<i>a</i>	0.875	0.341	0.534
<i>b</i>	0.719	0.480	0.239
<i>c</i>	0.750	0.459	0.291
<i>d</i>	0.894	0.352	0.542
<i>e</i>	0.625	0.564	0.061
<i>f</i>	0.656	0.526	0.130

Table 4.3: Calculated contributions of every basic event to system failure

larger contributions appear earlier than events with smaller contributions. The event criticality ordering is:

$$d < a < c < b < f < e. \quad (4.37)$$

All eight ordering schemes can be used to select basic events for every gate in the top-down or bottom-up technique. Different schemes will have a different effect on the efficiency measures. The neighbourhood schemes might be faster, because no special calculations are needed to be carried out as for the weighted schemes. They may also require a smaller number of connections, because the neighbourhoods of events are kept. However, the weighted schemes might result in smaller BDDs, because the most important events will appear on the high level of

a BDD. Therefore, the simplification rules will be applied high in a BDD, which might result in a smaller BDD. All those assumptions will be tested later in this chapter.

4.6.2 Order of Inputs Selection for the Top-Down Technique

Before the connection process in the top-down technique, gate inputs need to be selected in the order that they will be connected to represent a BDD of a parent gate. For the bottom-up method this is not an issue, because every gate input is analysed separately.

4.6.2.1 Order as listed process

The simplest way is to connect gate inputs in the way that they appear in the fault tree. This strategy was applied in Figure 4.3, when gate inputs for the construction process were considered in the top-down left-right manner.

4.6.2.2 Ordering event inputs before gate inputs

Since the new construction method does not use sub-node sharing, the replacement process, where new gate BDD structures are incorporated into the existing BDD, results in the duplications of structures which are connected to each 1 branch (for OR gates) or each 0 branch (for AND gates). The number of duplications can be minimised by placing basic events before gate events when considering the inputs for any gate. Compare two cases in Figure 4.14.

In case (i) the variable ordering is chosen to be $a < G1$, and during the replacement of the gate node no duplications occur, because there are no structures beneath the 1 branch of node $G1$ to repeat. In case (ii) the variable ordering $G1 < a$ results in the duplication of node a , because node a is a structure beneath the 1 branch of node $G1$, which is connected to the 1 branches after the replacement. Case (ii) results in a bigger BDD. The number of nodes in a BDD using the top-down scheme can be minimised by placing basic events prior to gate events in the BDD.

4.6.2.3 Ordering gate inputs according to the number of their event inputs

The maximum required size for the array can be minimised if gate inputs are selected according to their number of basic events, selecting the gate with the smallest

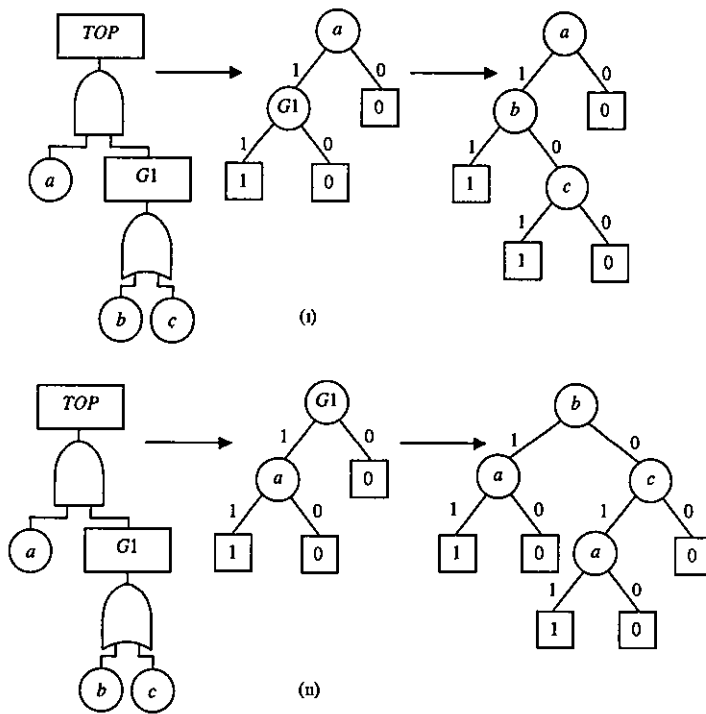


Figure 4.14: Process of top-down strategy, different ordering

number of basic events first. Compare two strategies for the example fault tree in Figure 4.15. The first strategy considers gates $G1$ and then $G2$ in a way that they

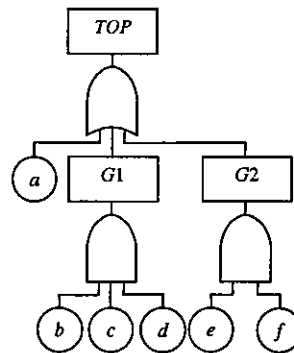


Figure 4.15: Example fault tree

appear in the list of inputs, i.e. $a < G1 < G2$, Figure 4.16. In this case the number of nodes in the BDD is 10. For the second strategy the number of inputs is taken into consideration. Since gate $G1$ has more inputs than gate $G2$, it is placed after $G2$, i.e. $a < G2 < G1$, Figure 4.17. The number of nodes in the BDD is 9.

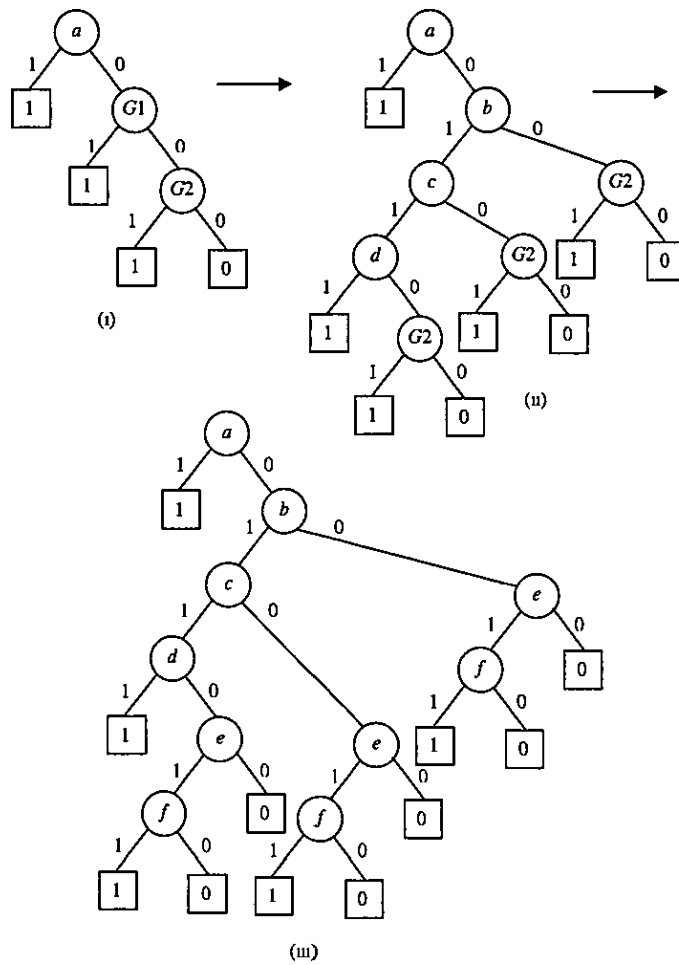


Figure 4.16: Ordering of inputs for the top event is $a < G1 < G2$

If there are no repeated events in the fault tree, this strategy always produces the smallest size of the final BDD, as it was presented in this example.

Even when the top gate has more than one level below it, following this rule minimises the required size for the array. The maximum required size of array for all 6 different possible gate ordering strategies for the fault tree in Figure 4.18 is presented in Table 4.4.

The ordering, which results in the smallest required size, is $G1 < G3 < G2$, where gate events are placed according to their increasing number of event inputs

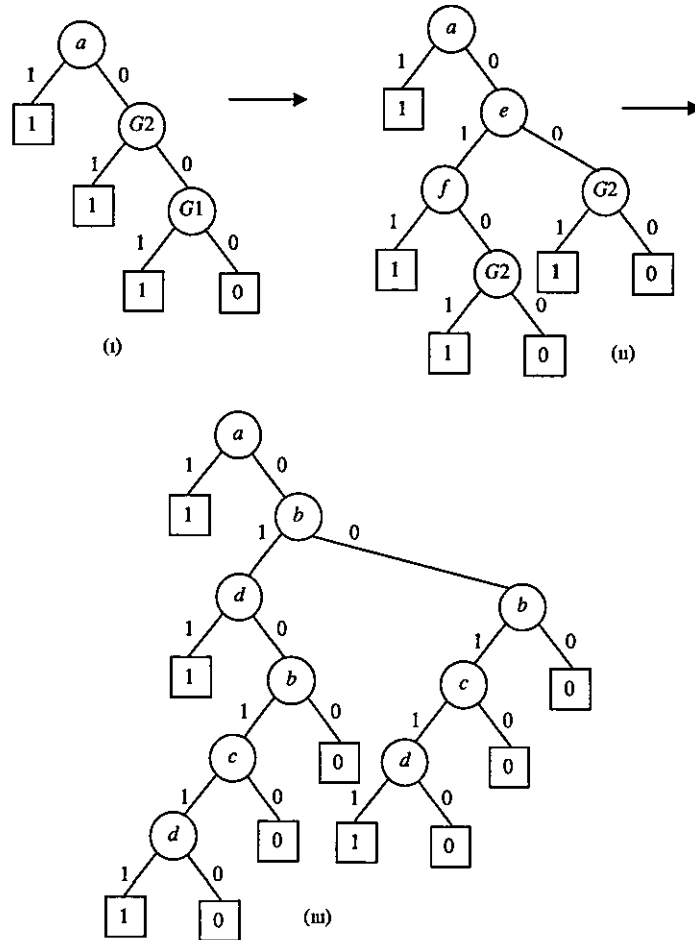


Figure 4.17: Ordering of inputs for the top event is $a < G2 < G1$

4.6.3 Order of BDDs Selection for the Bottom-Up Technique

In forming the BDD of a parent gate the BDDs of its input events are merged together, one at a time. A decision needs to be made about the order in which BDDs, representing the gate inputs, will be merged in order to obtain the final BDD for a parent gate.

4.6.3.1 Order as listed process

BDDs can be selected according to the order that gate inputs are listed, i.e. the BDD, presenting the left-most gate, is set to be the main BDD. If a gate has any event inputs, their BDD is set to be the main BDD to which the rest of the BDDs

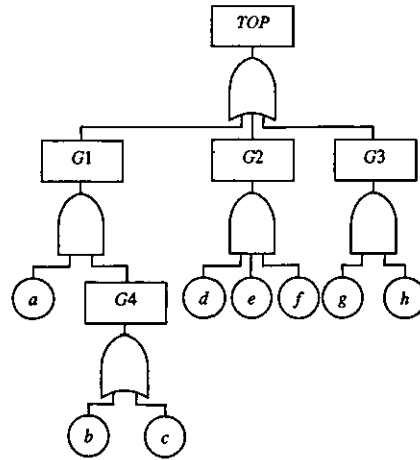


Figure 4 18: Example fault tree

Ordering	Value
$G1 < G2 < G3$	21
$G1 < G3 < G2$	19
$G2 < G1 < G3$	24
$G2 < G3 < G1$	27
$G3 < G1 < G2$	20
$G3 < G2 < G1$	26

Table 4.4. Maximum required size for 6 different strategies

are connected. An example illustrating this strategy is presented in Figure 4 4.

4.6.3.2 According to a defined ordering of basic events

The main BDD can be chosen according to the position of the top node in the ordering scheme of the whole system. Some situations are explained below.

1. The main BDD is selected according to the order of their root nodes. If two BDDs need to be connected and their top nodes are a and b respectively, the first BDD is set to be the main BDD, if the ordering is $a < b < c < d$, (Figure 4 19). As can be seen from this example, the resulting BDD does not necessarily preserve the same specified ordering scheme throughout the BDD, i.e. the final BDD ordering is $a < c < b < d$.

2. If there are two BDDs with the same root node the decision of which BDD

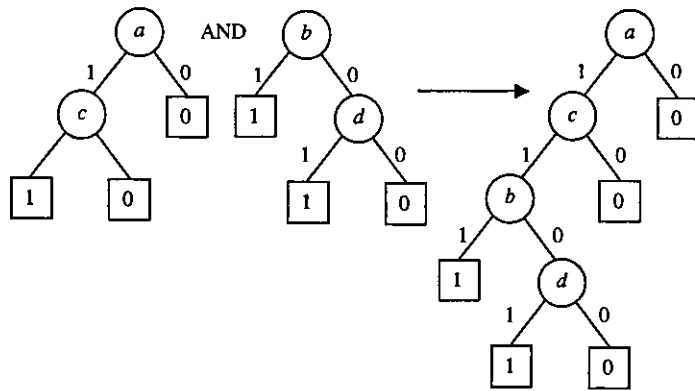


Figure 4 19: Example of two BDDs with different top nodes

is going to be the main BDD is made according to the order of the nodes below on the 1 and 0 branches. A BDD is set to be the main BDD if any of its descendant nodes appear as the earliest node in the ordering scheme. This rule is explained by considering the example in Figure 4.20. Since the ordering is $a < b < c < d < e$, and c , the 'earliest' node of the first BDD at the second level, is after b , the 'earliest' node of the second BDD at the second level, therefore, the second BDD is set to be the main BDD. This rule can be applied to the lowest possible level in the BDD until the choice about the ordering can be made.

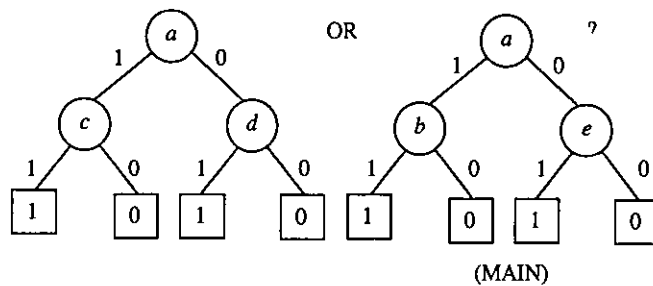


Figure 4 20. Example of two BDDs with the same top and different descendants

3. If there are two BDDs with the same 'earliest' node at the second level and one of them consists a node with a terminal vertex, the BDD with a terminal vertex is set to be the main BDD, as shown in Figure 4 21

This strategy keeps variables, which are close in the ordering scheme, close together in the BDD structure. However, when a parent gate contains both event

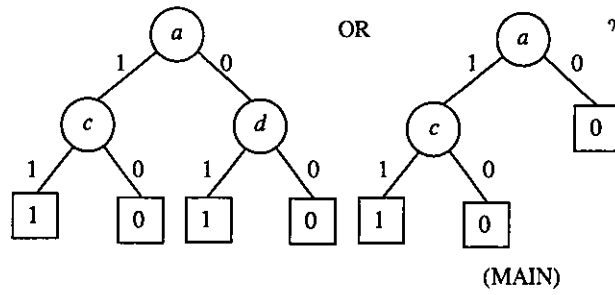


Figure 4 21: Example of two BDDs with some terminal vertices

and gate inputs, it does not use the BDD of basic events as the main BDD. This rule was highlighted to be efficient in the top-down approach, where event inputs were considered before gate inputs, constructing a BDD for the parent gate. This aspect can have a crucial effect on the size of the final BDD. Therefore, while analysing the efficiency of this strategy, an exception, that the BDD of basic events is set to be the main BDD regardless of its top node label, will be made.

4.6.3.3 According to the number of available branches

BDDs could be selected according to their number of available branches. The number of nodes and the maximum size for the final BDD can be minimised if a BDD with the smallest number of available branches (potential connection points) is set to be the main BDD, when the merging is performed. For example, in Figure 4 22 the BDDs represent gate inputs to an 'OR' gate, the first BDD has two available 'OR' branches for connection (two terminal 0 nodes) and the second has one. Therefore, the second BDD is chosen to be the main BDD, the final BDD has four nodes (the structure on the left) instead of six (the structure on the right) if the second BDD was set to be the main BDD. This rule also covers the previously mentioned case when a BDD of basic events is set to be the main BDD if the parent gate has some gate inputs. Analysing the efficiency this rule will result in not only the minimum size of array, but also the minimum final BDD, if no repeated events appear in the system.

All presented strategies will be investigated and efficiency measures obtained in the next section

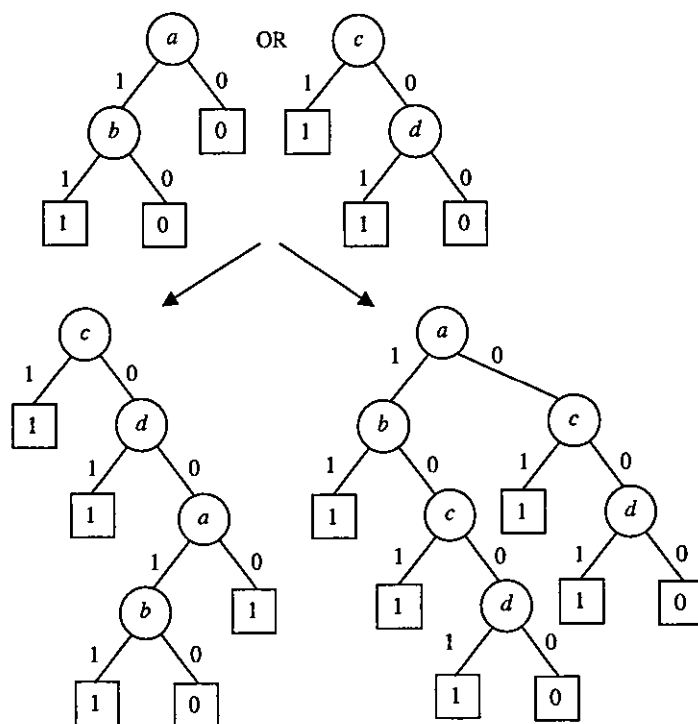


Figure 4 22: Example of connection process using the number of available branches

4.6.4 Results

Research has been carried out testing all the presented strategies of selecting gates, basic events and BDDs in order to work out their relative advantages. A specific approach will work well on some fault trees and not on others. It is the performance of an approach over the range of problems that it may encounter that should be established. All algorithms are applied to a library of 265 coherent fault trees. Characteristics of test fault trees are shown in Tables A.1 - A.6. The first column is a label to identify an example fault tree, then three following columns present the complexity of a fault tree in terms of the number of gates, the number of basic events and the number of repeated events. The fifth and the sixth columns show the results of the simplification process, presented in Chapter 2, i.e. the number of complex events and the number of modules after the reduction and the modularisation respectively. The simplification process is applied in order to maximise the number of fault trees converted to BDDs because for some 'large' fault trees the conversion process of the original fault tree would be impossible. The last column presents the number of minimal cut sets.

Example fault trees are categorised as small fault trees (221 FTs) and 'large' fault trees (44 FTs). 'Large' fault trees are classified as those with a large number of minimal cut sets. Also, high complexity fault trees, i.e. with a large number of gates and non-repeated basic events, are ranked as 'large' fault trees. These two properties of fault trees result in a time consuming analysis process or even makes the analysis impossible in reasonable times. If results do not appear in 24 hours (86400s) the conversion process is terminated. It is 'large' fault trees which present the highest degree of difficulty in their analysis. As such it is these fault trees which will really test the competence of any algorithm.

A major part of 'large' example fault trees consists of a benchmark of problems obtained from Rauzy [19], identified by a star in the column of the FT name in the complexity Table A.6.

During the analysis every fault tree undergoes the BDD construction process. The first two parts of the results reported in this section examine merits of the top-down and bottom-up techniques respectively. In those two trials basic events are selected as listed or according to one of the eight ordering schemes. Gates (top-down approach) and BDDs (bottom-up approach) are chosen as listed. The main difference between the two chosen trials is the starting point (top or bottom) when traversing a fault tree. Therefore, the comparison of the two techniques gives an indication of advantages and disadvantages of the two ways of traversing fault trees during the BDD conversion procedure. The two efficiency measures, the number of nodes and the processing time, are calculated and analysed and then a comparison between the top-down and bottom-up techniques is made.

The 8 basic event ordering schemes are ranked in order according to the effectiveness of the conversion process that they produce. In some methods '0' ordering scheme is considered, where elements are considered in the order that they appear in the fault tree. The performance of the schemes is assessed in three ways. Firstly, the sum of each characteristic measure is calculated over the whole set of test fault trees, for example, the time taken to build BDDs for the whole set of fault trees is obtained. Secondly, the number of times that each scheme produces the highest (best) ranking is assessed. Finally, the average ranking of each scheme across the set of fault trees is considered. These measures give an indication of the overall

performance of the ordering scheme. The ordering schemes are ranked for all the efficiency measures according to their performance on small fault trees, 'large' fault trees and the whole set of fault trees. This is done in order to be able to identify the 'optimum' ordering scheme for every method in each complexity category if one exists.

4.6.4.1 Top-down technique

Description Gates are selected as listed. Basic events are selected as listed or according to one of the eight ordering schemes.

4.6.4.1.1 Summary results

Summary details of test results for the fault trees are presented in Tables A 7 - A 11, which show the number of nodes for small fault trees. Table A 12 shows the number of nodes for the large fault trees. The time taken to perform calculations for small fault trees is shown in Tables A.13 - A.17, together with Table A.18 for the large fault trees. The first column in all tables identifies the test fault tree, then the next 9 columns show the outcome of a particular efficiency measure for 9 different schemes, i.e. the 8 presented ordering schemes are extended along with the 'order as listed' strategy, called the 0 scheme. The last column represents the best (minimum) result over all ordering schemes.

Summarizing these tables gives that the nine ordering schemes gave identical results in the number of nodes for 100 small fault trees (out of 221) and 8 'large' fault trees (out of 44). Out of them 74 small fault trees and 5 'large' fault trees were simplified so much that their representing BDD contains only 1 node, therefore, their results will not be taken into account while the conclusions are drawn. For nine 'large' fault trees the computations were not possible in reasonable times.

4.6.4.1.2 Variable ordering for the top-down approach

Analysis of the number of nodes in BDDs and processing time for 'small', 'large' and all example fault trees is presented in this section, applying the three ranking techniques of the ordering schemes. The result of the analysis is shown in Table 4 5 and Table 4 6 for the number of nodes and the processing time respectively.

First of all, these results show that in almost all the cases any of the eight or-

		Scheme	0	1	2	3	4	5	6	7	8
Small fault trees	Total quantity ranking	Number of nodes	99284	62117	66056	65807	62260	61165	61108	66673	63104
		Rank	9	3	7	6	4	2	1	8	5
	Highest scheme ranking	Number of FTs with the highest rank	2	78	67	69	77	88	90	77	93
		Rank	9	4	8	7	5-6	3	2	5-6	1
	Added scheme ranking	Added ranking	910	256	355	328	294	213	233	302	227
		Rank	9	4	8	7	5	1	3	6	2
Large fault trees	Total quantity ranking	Number of nodes	10628393	5652002	5824414	6001786	5714308	5362361	5553721	5528696	5637389
		Rank	9	5	7	8	6	1	3	2	4
	Highest scheme ranking	Number of FTs with the highest rank	0	8	6	6	6	14	11	9	12
		Rank	9	5	6-7-8	6-7-8	6-7-8	1	3	4	2
	Added scheme ranking	Added ranking	204	83	117	118	99	60	70	98	72
		Rank	9	4	7	8	6	1	2	5	3
All fault trees	Total quantity ranking	Number of nodes	10727677	5714119	5890470	6067593	5776568	5423526	5614829	5595369	5700493
		Rank	9	5	7	8	6	1	3	2	4
	Highest scheme ranking	Number of FTs with the highest rank	2	86	73	75	83	102	101	86	105
		Rank	9	4-5	8	7	6	2	3	4-5	1
	Added scheme ranking	Added ranking	1114	339	472	446	393	273	303	400	299
		Rank	9	4	8	7	5	1	3	6	2

Table 4 5. Top-down method, number of nodes

dering schemes can perform significantly better than the 'order as listed' method. In other words, choosing any of the defined ordering schemes in the top-down connection method is more efficient than connecting basic events in the order that they are listed. Therefore, the 'order as listed' method is ranked last.

Secondly, the ranking of the other eight ordering schemes (1-8) depends on the efficiency measure and the ranking technique. However, some general remarks can be made. According to the number of nodes in the final BDD the best performance was obtained using the two top-down weighted schemes, (5) and (6), and the worst performance was obtained by the two modified depth-first schemes, (2) and (3). It is quite clear, that the four weighted ordering schemes (5-8) gave smaller BDDs than the four neighbourhood ordering schemes (1-4). This can be a valuable result while choosing the ordering schemes.

Finally, for the processing time the event criticality scheme (8) and the non-dynamic top-down weighted scheme (5) were ranked high because they resulted in a fast conversion process. The two modified depth-first schemes, (2) and (3), performed poorly, however, the distinction in the efficiency between the weighted and neighbourhood schemes was still present but not as marked as for the number

		Scheme	0	1	2	3	4	5	6	7	8
Small fault trees	Total quantity ranking	Time	17 70	15 02	11 91	11 14	9 57	10 38	9 16	10 47	9 07
		Rank	9	8	7	6	3	4	2	5	1
	Highest scheme ranking	Number of FTs with the highest rank	5	11	3	4	6	8	7	8	11
		Rank	7	1-2	9	8	6	3-4	5	3-4	1-2
	Added scheme ranking	Added ranking	525	510	401	416	434	534	428	410	397
		Rank	8	7	2	4	6	9	5	3	1
Large fault trees	Total quantity ranking	Time	425 21	221 73	217 64	225 37	252 15	173 10	234 95	204 10	180 54
		Rank	9	5	4	6	8	1	7	3	2
	Highest scheme ranking	Number of FTs with the highest rank	5	11	3	4	6	8	7	8	11
		Rank	7	1-2	9	8	6	3-4	5	3-4	1-2
	Added scheme ranking	Added ranking	165	89	114	110	117	90	102	99	78
		Rank	9	2	7	6	8	3	5	4	1
All fault trees	Total quantity ranking	Time	442 91	236 75	229 55	236 51	261 72	183 48	244 11	214 57	189 61
		Rank	9	6	4	5	8	1	7	3	2
	Highest scheme ranking	Number of FTs with the highest rank	49	51	65	65	63	40	57	74	71
		Rank	8	7	3-4	3-4	5	9	6	1	2
	Added scheme ranking	Added ranking	690	599	515	526	551	624	530	509	475
		Rank	9	8	3	5	7	4	6	2	1

Table 4 6. Top-down method, processing time

of nodes

4.6.4.2 Bottom-up technique

Description. BDDs are selected as listed. Basic events are selected as listed or according to one of the eight ordering schemes.

4.6.4.2.1 Summary results

Summary details of test fault trees are presented in Tables A.19 - A 23, which show the number of nodes for small fault trees and Table A.24 shows the number of nodes for 'large' fault trees Tables A 25 - A 29 show time taken to perform calculations for small fault trees and Table A 30 show time for 'large' fault trees The representation of results is the same as for the top-down strategy

4.6.4.2.2 Variable ordering for the bottom-up approach

Analysis of the number of nodes in BDDs and processing time for 'small', 'large' and all example fault trees is presented in this section, applying the three ranking techniques of the ordering schemes The result of the analysis is shown in Table 4.7 and Table 4.8.

		Scheme	0	1	2	3	4	5	6	7	8
Small fault trees	Total quantity ranking	Number of nodes	99284	62117	66056	65807	62260	61165	61108	66673	63104
		Rank	9	3	7	6	4	2	1	8	5
	Highest scheme ranking	Number of FTs with the highest rank	2	78	67	69	77	88	90	77	93
		Rank	9	4	8	7	5-6	3	2	5-6	1
	Added scheme ranking	Added ranking	910	256	355	328	294	213	233	302	227
		Rank	9	4	8	7	5	1	3	6	2
Large fault trees	Total quantity ranking	Number of nodes	10628393	5652002	5824414	6001786	5714308	5362361	5553721	5528696	5637389
		Rank	9	5	7	8	6	1	3	2	4
	Highest scheme ranking	Number of FTs with the highest rank	0	8	6	6	6	14	11	9	12
		Rank	9	5	6-7-8	6-7-8	6-7-8	1	3	4	2
	Added scheme ranking	Added ranking	204	83	117	118	99	60	70	98	72
		Rank	9	4	7	8	6	1	2	5	3
All fault trees	Total quantity ranking	Number of nodes	10727677	5714119	5890470	6067593	5776568	5423526	5614829	5595369	5700493
		Rank	9	5	7	8	6	1	3	2	4
	Highest scheme ranking	Number of FTs with the highest rank	2	86	73	75	83	102	101	86	105
		Rank	9	4-5	8	7	6	2	3	4-5	1
	Added scheme ranking	Added ranking	1114	339	472	446	393	273	303	400	299
		Rank	9	4	8	7	5	1	3	6	2

Table 4 7 Bottom-up method, number of nodes

Similarly to the top-down approach, these results show that in almost all the cases any of the eight ordering schemes can perform significantly better than the 'order as listed' method. The ranking of the other eight ordering schemes (1-8) according to the number of nodes is the same as the ranking of the top-down technique, because the number of nodes obtained is the same in the two approaches. The smallest BDDs were obtained using the two top-down weighted schemes, (5) and (6), and the largest BDDs were resulted in using the two modified depth-first schemes, (2) and (3).

For the processing time the event criticality scheme (8) was ranked high and the two modified depth-first schemes, (2) and (3), performed poorly. However, the overall ranking of the eight ordering schemes was dependent on the set of fault trees and the ranking technique chosen, therefore, no further conclusions on the suitability of the ordering schemes can be drawn.

4.6.4.3 Comparison of top-down and bottom-up technique

The total summary of the performance parameters for both the top-down and the bottom-up methods during the conversion of 121 small fault trees and 27 'large'

		Scheme	0	1	2	3	4	5	6	7	8
Small fault trees	Total quantity ranking	Number of nodes	99284	62117	66056	65807	62260	61165	61108	66673	63104
		Rank	9	3	7	6	4	2	1	8	5
	Highest scheme ranking	Number of FTs with the highest rank	2	78	67	69	77	88	90	77	93
		Rank	9	4	8	7	5-6	3	2	5-6	1
	Added scheme ranking	Added ranking	910	256	355	328	294	213	233	302	227
		Rank	9	4	8	7	5	1	3	6	2
Large fault trees	Total quantity ranking	Number of nodes	10628393	5652002	5824414	6001786	5714308	5362361	5553721	5528696	5637389
		Rank	9	5	7	8	6	1	3	2	4
	Highest scheme ranking	Number of FTs with the highest rank	0	8	6	6	6	14	11	9	12
		Rank	9	5	6-7-8	6-7-8	6-7-8	1	3	4	2
	Added scheme ranking	Added ranking	204	83	117	118	99	60	70	98	72
		Rank	9	4	7	8	6	1	2	5	3
All fault trees	Total quantity ranking	Number of nodes	10727677	5714119	5890470	6067593	5776568	5423526	5614829	5595369	5700493
		Rank	9	5	7	8	6	1	3	2	4
	Highest scheme ranking	Number of FTs with the highest rank	2	86	73	75	83	102	101	86	105
		Rank	9	4-5	8	7	6	2	3	4-5	1
	Added scheme ranking	Added ranking	1114	339	472	446	393	273	303	400	299
		Rank	9	4	8	7	5	1	3	6	2

Table 4 8 Bottom-up method, processing time

fault trees to BDDs is presented in Table 4.9 and Table 4 10

Efficiency measure	0 scheme	1 scheme	2 scheme	3 scheme	4 scheme	5 scheme	6 scheme	7 scheme	8 scheme
Number of nodes for the top-down approach	99284	62117	66056	65807	62260	61165	61108	66673	63104
Number of nodes for the bottom-up approach	99284	62117	66056	65807	62260	61165	61108	66673	63104
Time for the top-down approach	17 70	15 02	11 91	11 14	9 57	10 38	9 16	10 47	9 07
Time for the bottom-up approach	3 13	3 30	2 78	2 86	2 60	3 01	3 09	2 61	2 77

Table 4 9: Total summary of the top-down and the bottom-up technique for 'small' fault trees

According to these results it is clear that the number of nodes in the final BDD for every example fault tree does not depend on the construction method. In both trials basic events are connected in a BDD chain in the same manner and gates or BDDs are selected in the same order Therefore, despite the fact that the top-down and the bottom-up approaches apply a different traversal of a fault tree, they both are expected to give the same final structure for a BDD.

Efficiency measure	0 scheme	1 scheme	2 scheme	3 scheme	4 scheme	5 scheme	6 scheme	7 scheme	8 scheme
Number of nodes for the top-down approach	10628393	5652002	5824414	6001786	5714308	5362361	5553721	5528696	5637389
Number of nodes for the bottom-up approach	10628393	5652002	5824414	6001786	5714308	5362361	5553721	5528696	5637389
Time for the top-down approach	425 21	221 73	217 64	225 37	252 15	173 1	234 95	204 1	180 54
Time for the bottom-up approach	269 79	118 84	125 93	137 6	212 15	122 72	135 31	117 58	125 74

Table 4.10: Total summary of the top-down and the bottom-up technique for 'large' fault trees

However, using the bottom-up technique converts fault trees to BDDs faster than applying the top-down approach. The slower process for the latter technique can be explained by a higher usage of memory resources. When the BDD for the parent gate is constructed, the nodes beneath the required branch are reconnected every time after the replacement of a gate node in the structure.

Therefore, the top-down approach is discarded and further developments of the bottom-up method are presented. Since the eight ordering schemes resulted in smaller number of nodes in BDDs and shorter time of calculations than in the case of 'order as listed' manner (0 scheme), it is expected that further investigations of the bottom-up technique using the eight ordering schemes can give even better efficiency.

4.6.4.4 Bottom-up technique, chosen trials

4.6.4.4.1 Introduction

After the top-down scheme was discarded because of its inefficient usage of memory, further investigations were carried out on other strategies using the bottom-up technique. The purpose was to test different ways that BDDs can be selected during the connection process. The values of efficiency measures, such as number of nodes and processing time, were calculated and the ordering schemes were ranked.

In the first trial, whose results were presented and compared with the top-down technique, basic events are ordered according to one of the 9 ordering schemes but BDDs are selected as listed, i.e. a BDD which represents the first input of a gate in a list of inputs is chosen to be the main BDD.

In the second trial the ordering schemes are used not only to order basic events but also to choose the order that BDDs are considered in the connection process. The detailed description of this strategy was presented earlier in this chapter.

In the third trial BDDs are merged according to the number of points at which the BDDs are combined. Intuitively, this trial should be efficient, since the minimal number of points should be a good criteria for choosing the main BDD during the connection process. The ordering schemes are still applied for the selection of basic events but not for the selection of BDDs. Results of this method are also presented in this section.

The comparison of the three techniques is performed using the results of 'large' fault trees only since their complexity allows a reasonable test of each method. The ordering schemes are ranked in the same way as it was performed in the previous section, i.e. the three different ranking techniques are applied.

4.6.4.4.2 First trial

Description. Selection of BDDs is 'order as listed', as presented in Section 4.6.3.1. Selection of basic events is according to one of the nine ordering schemes.

There were 44 'large' fault trees analysed. The analysis of 9 fault trees from this set was not possible in reasonable times, and for 5 examples the simplification process was so efficient that the final BDD contains only 1 node. So, those fault trees were not considered in the analysis. For the remaining 30 'large' fault trees results were presented in Table A.24 and Table A.30 for number of nodes and processing time respectively. The result of the analysis is shown in Table 4.11 and Table 4.12, which is a part of the results presented in Section 4.6.4.2.2.

For all 'large' fault trees considering the number of nodes and the processing time any of the eight ordering schemes performed significantly better than the 'order

		Scheme	0	1	2	3	4	5	6	7	8
Large fault trees	Total quantity ranking	Number of nodes	10628393	5652002	5824414	6001786	5714308	5362361	5553721	5528696	5637389
		Rank	9	5	7	8	6	1	3	2	4
	Highest scheme ranking	Number of FTs with the highest rank	0	8	6	6	6	14	11	9	12
		Rank	9	5	6-7-8	6-7-8	6-7-8	1	3	4	2
	Added scheme ranking	Added ranking	204	83	117	118	99	60	70	98	72
		Rank	9	4	7	8	6	1	2	5	3

Table 4 11: Bottom-up method, first trial, number of nodes

		Scheme	0	1	2	3	4	5	6	7	8
Large fault trees	Total quantity ranking	Time	269 79	118 84	125 93	137 60	212 15	122 72	135 31	117 58	125 74
		Rank	9	2	5	7	8	3	6	1	4
	Highest scheme ranking	Number of FTs with the highest rank	4	9	6	9	10	6	8	5	8
		Rank	9	2-3	6-7	2-3	1	6-7	4-5	8	4-5
	Added scheme ranking	Added ranking	145	71	110	98	92	83	89	100	82
		Rank	9	1	8	6	5	3	4	7	2

Table 4 12 Bottom-up method, first trial, processing time

as listed' method. For the number of nodes the non-dynamic top-down weighted scheme (5) performed best and the two modified depth-first schemes, (2) and (3), came last. Overall, the four weighted schemes (5-8) resulted in smaller BDDs than the four neighbourhood schemes (1-4). For the processing time the ranking of the ordering schemes was dependent on the ranking method. However, the modified top-down ordering scheme (1) gave good results and was ranked high using all three ranking methods. The distinction between the efficiency of the type of ordering scheme was minor and no further conclusions could be drawn.

4.6.4.4.3 Second trial

Description. Selection of BDDs is according to the ordering of their top nodes, as presented in Section 4.6.3.2. Selection of basic events is according to one of the eight ordering schemes

Summary details of test results for the fault trees are presented in Tables A 31 and A 32 for the number of nodes and the processing time respectively. As it was obtained in the first trial, the eight ordering schemes gave identical results in the number of nodes for 5 'large' fault trees (out of 44), i.e. BDDs of 1 node were pro-

duced For 9 'large' fault trees the computations were not possible in reasonable times Therefore, those fault trees are not taken into consideration while ranking the ordering schemes. The 'order as listed' technique is not applied because every step in this method requires a defined ordering.

The result of the analysis for the 30 'large' fault trees is shown in Table 4.13 and Table 4.14 for the number of nodes and processing time respectively

		Scheme	0	1	2	3	4	5	6	7	8
Large fault trees	Total quantity ranking	Number of nodes	N/A	8214468	20477527	18103597	5924252	3577896	5783722	6755913	3449351
		Rank	N/A	6	8	7	4	2	3	5	1
	Highest scheme ranking	Number of FTs with the highest rank	N/A	2	3	3	3	8	7	1	19
		Rank	N/A	7	4-5-6	4-5-6	4-5-6	2	3	8	1
	Added scheme ranking	Added ranking	N/A	115	133	126	116	77	94	142	56
		Rank	N/A	4	7	6	5	2	3	8	1

Table 4.13: Bottom-up method, second trial, number of nodes

		Scheme	0	1	2	3	4	5	6	7	8
Large fault trees	Total quantity ranking	Time	N/A	180 97	267 17	141 94	120 88	59 15	103 73	160 45	266 78
		Rank	N/A	6	8	4	3	1	2	5	7
	Highest scheme ranking	Number of FTs with the highest rank	N/A	5	5	3	8	4	4	5	11
		Rank	N/A	3-4-5	3-4-5	8	2	6-7	6-7	3-4-5	1
	Added scheme ranking	Added ranking	N/A	87	90	105	83	75	88	102	55
		Rank	N/A	4	6	8	3	2	5	7	1

Table 4.14: Bottom-up method, second trial, processing time

For the number of nodes the event criticality scheme (8) performed well together with the two top-down weighted schemes, (5) and (6) The two modified depth-first schemes, (2) and (3), resulted in a lot larger BDDs and came last. For the processing time the ranking of the ordering schemes was dependent on the ranking method However, the main pattern of the ranking is similar to the one for the number of nodes

4.6.4.4.4 Third trial

Description Selection of BDDs is according to the number of connection points available, presented in Section 4.6.3.3. Selection of basic events is according to one of the nine ordering schemes

Summary details of test results for the fault trees are presented in Tables A 33 and A 34 for the number of nodes and the processing time respectively. As it was obtained in the two previous trials, all the ordering schemes gave identical results in the number of nodes for 5 'large' fault trees (out of 44), i.e. BDDs of 1 node were produced. For 9 'large' fault trees the computations were not possible in reasonable times. Therefore, 30 'large' fault trees are taken into consideration while ranking the ordering schemes.

The result of the analysis is shown in Table 4.15 and Table 4.16 for the number of nodes and processing time respectively.

		Scheme	0	1	2	3	4	5	6	7	8
Large fault trees	Total quantity ranking	Number of nodes	12518644	4972154	5515658	5466903	5650688	5148732	4992002	4870866	5375739
		Rank	9	2	7	6	8	4	3	1	5
	Highest scheme ranking	Number of FTs with the highest rank	0	12	8	9	9	15	9	11	9
		Rank	9	2	8	4-5-6-7	4-5-6-7	1	4-5-6-7	3	4-5-6-7
	Added scheme ranking	Added ranking	200	71	96	88	97	61	79	83	97
		Rank	9	2	6	5	7-8	1	3	4	7-8

Table 4.15 Bottom-up method, third trial, number of nodes

		Scheme	0	1	2	3	4	5	6	7	8
Large fault trees	Total quantity ranking	Number of nodes	12518644	4972154	5515658	5466903	5650688	5148732	4992002	4870866	5375739
		Rank	9	2	7	6	8	4	3	1	5
	Highest scheme ranking	Number of FTs with the highest rank	0	12	8	9	9	15	9	11	9
		Rank	9	2	8	4-5-6-7	4-5-6-7	1	4-5-6-7	3	4-5-6-7
	Added scheme ranking	Added ranking	200	71	96	88	97	61	79	83	97
		Rank	9	2	6	5	7-8	1	3	4	7-8

Table 4.16 Bottom-up method, third trial, processing time

As it was observed previously, any of the eight ordering schemes performed better than the 'order as listed' approach. The modified top-down scheme (1) and the non-dynamic top-down weighted scheme (5) performed well according to the two measurements, i.e. the number of nodes and the processing time. The depth-first with number of leaves ordering scheme (4) gave the worst performance because it resulted in larger BDDs in a longer processing time than the majority of the eight ordering schemes (1-8).

4.6.4.4.5 Comparison of the bottom-up techniques

The three trials were compared and conclusions were drawn, taking into account the total number of nodes and the total processing time. Results are presented in Table 4.17 and Table 4.18.

Strategy\scheme	0	1	2	3	4	5	6	7	8
First trial	10628393	5652002	5824414	6001786	5714308	5362361	5553721	5528696	5637389
Second trial	N/A	8214468	20477527	18103597	5924252	3577896	5783722	6755913	3449351
Third trial	12518644	4972154	5515658	5466903	5650688	5148732	4992002	4870866	5375739

Table 4.17: Bottom-up, comparison of the three trials, total number of nodes, 'large' fault trees

Strategy\scheme	0	1	2	3	4	5	6	7	8
First trial	425 21	221 73	217 64	225 37	252 15	173 1	234 95	204 1	180 54
Second trial	N/A	180 97	267 17	141 94	120 88	59 15	103 73	160 45	266 78
Third trial	298 21	81 19	90 98	93 84	105 21	97 7	83 35	83 58	102 25

Table 4.18 Bottom-up, comparison of the three trials, total processing time, 'large' fault trees

The third trial of the bottom-up strategy approach gave the best result. The total number of nodes in BDDs using the third trial was smaller than the total number of BDDs from the first trial using the eight ordering schemes (1-8). Only using the 'order as listed' scheme (0) the total number of nodes was larger than in the first trial. However, the 'order as listed' scheme (0) was not ranked highly for any of the trials, so this result can be discounted. The total processing time was shorter for the third trial than for the first trial using all nine ordering schemes

Also, the third trial was more efficient than the second method for most of the ordering schemes. Only for the non-dynamic top-down weighted scheme (5) and the event criticality scheme (8) the total number of nodes using the third trial was larger than the total number of nodes using the second method. This can improve the efficiency of the bottom-up technique, especially when the two ordering schemes (5) and (8) were highly ranked for the bottom-up technique. The total processing time was shorter for the third trial than the second trial for every ordering scheme, except scheme 5. Those results make the third trial the best option for the bottom-up approach

Comparing the first and the second trials, it gave different results for the two efficiency measurements. The first trial resulted in the smaller total number of nodes than the second trial for almost every ordering scheme. Again, only for the non-dynamic top-down weighted scheme (5) and the event criticality scheme (8) the total number of nodes was smaller using the second trial rather than the first trial. For the total processing time the second trial performed better than the first trial, i.e. it resulted in a shorter total processing time of the conversion process.

Summarising the bottom-up approach method, the third trial that uses the selection of BDDs according to the number of connection points available and the selection of basic events according to one of the nine ordering schemes, is the most efficient method out of all proposed approaches in this work. The highly ranked ordering schemes were the non-dynamic top-down weighted scheme (5) and the modified top-down scheme (1). Using the ranking from the previous section the worst performance was obtained by the 'order as listed' scheme (0).

The efficiency of this new approach will be compared with other current BDD conversion techniques. Therefore, in the next section the Component Connection Method, using the third trial of the bottom-up approach, will be compared with the well-known BDD conversion technique, called the *ite* method, that was established by Rauzy [2] and was presented in Chapter 3.

4.7 Comparison Between the Component Connection and the *ite* method

Since the third attempt of the bottom-up technique gave the best performance it will be compared with the *ite* method using the number of nodes and the processing time, as metrics to judge the efficiency of the two methods.

Results of the number of nodes and the processing time for the *ite* method for 'large' fault trees are presented in Table A.35 and Table A.36. The variable ordering is required for the construction method, therefore, there are no results for the 0 scheme. The BDD conversion process was performed for all 44 'large' fault trees. Only for 1 example fault tree in one ordering scheme the computations were not possible in reasonable times and for 5 fault trees the eight ordering schemes gave

the same result.

The ranking results for 38 'large' fault trees are shown in Table 4.19 and Table 4.20 according to the number of nodes and the processing time respectively.

		Scheme	1	2	3	4	5	6	7	8
Large fault trees	Total quantity ranking	Number of nodes	2176461	722103	718377	572428	2464680	675169	715325	1619301
		Rank	7	5	4	1	8	2	3	6
	Highest scheme ranking	Number of FTs with the highest rank	1	2	6	5	4	11	6	9
		Rank	8	7	3-4	5	6	1	3-4	2
	Added scheme ranking	Added ranking	214	147	131	152	162	111	168	147
		Rank	8	3-4	2	5	6	1	7	3-4

Table 4.19 *ite* method, number of nodes

		Scheme	1	2	3	4	5	6	7	8
Large fault trees	Total quantity ranking	Time	15602 82	1776 48	1764 99	1832 96	21710 86	2012 57	1839 74	11092 22
		Rank	7	2	1	3	8	5	4	6
	Highest scheme ranking	Number of FTs with the highest rank	7	8	10	13	7	14	7	8
		Rank	6-7-8	5	3	2	6-7-8	1	6-7-8	4
	Added scheme ranking	Added ranking	186	134	128	120	166	113	157	156
		Rank	8	4	3	2	7	1	6	5

Table 4.20: *ite* method, processing time

The dynamic top-down weighted scheme (6) gave good results for both measurements and was ranked high. Additionally according to the processing time the modified priority depth-first scheme (3) performed well. The worst performance was achieved while using the modified top-down scheme (1) and the non-dynamic top-down weighted scheme (5). There is no clear indication of which ordering schemes are most efficient.

The total number of nodes of the 'large' fault trees, that were converted to BDDs using both the *ite* method and the Component Connection Method, was compared for the eight ordering schemes in Table 4.21

The total number of nodes is a lot smaller for the *ite* method than for the Component Connection Method. Also, using the *ite* method the analysis of the complete

Strategy\scheme	1	2	3	4	5	6	7	8
ite method	94996	92843	73205	38500	55259	110687	76790	33042
Connection method	4972154	5515658	5466903	5650688	5148732	4992002	4870866	5375739

Table 4 21: Comparison Between the ite and the Component Connection Method, total number of nodes

set of fault trees was performed, whereas calculations for 9 trees using the Component Connection Method were not possible in reasonable times. The biggest disadvantage of the Component Connection Method is that of the inefficient memory usage, where parts of a BDD are repeated but not shared.

The comparison in the total processing time for 'large' fault trees is shown in Table 4 22. The ite method resulted in a shorter fault tree conversion process.

Strategy\scheme	1	2	3	4	5	6	7	8
ite method	207 94	71 34	70 00	102 64	29 44	57 19	116 88	38 19
Connection method	81 19	90 98	93 84	105 21	97 70	83 35	83 58	102 25

Table 4.22 Comparison Between the ite and the Component Connection Method, total processing time

However, there were some cases where the Component Connection Method has produced better results than the ite method, for example, in the modified top-down scheme (1) or the bottom-up weighted scheme (7). If those ordering schemes were chosen to build BDDs for example fault trees using the Component Connection Method, a faster process would be obtained. This is only a minor advantage of this method, because if the number of nodes was critical for the analysis none of the eight ordering schemes performed better using the Component Connection Method than in the ite method

In summary, the Component Connection Method produced significantly bigger BDDs than the ite method. For some examples the calculations were not even finished in reasonable times. This approach has a high demand for memory space since the identical parts in the BDD structure are repeated but not shared. Therefore, further investigations are carried out and the application of the sub-node sharing technique is introduced in the Component Connection Method

4.8 Sub-node Sharing

Using the Component Connection Method there are some parts in the final BDD structure, which are identical. This causes an inefficient memory usage. The sub-node sharing will be introduced during the combining of two gates, i.e. all available branches would point to the same structure, instead of making a separate copy for each of them. However, since after the connection process the simplification is applied if repeated events appear in the structure, there are some defined rules that need to be followed in order to achieve the effective connection process.

4.8.1 Presentation of the Sub-node Sharing in Component Connection Method

The sub-node sharing can be implemented during the connection of two BDDs, as it is illustrated in Figure 4.23. In this example the left BDD is set to be the main

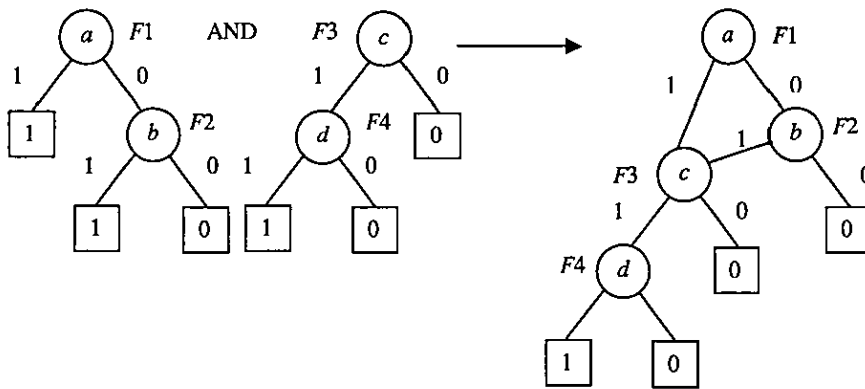


Figure 4.23 The process of sub-node sharing

BDD. There are two available points, i.e. two terminal vertices 1, that can share the same copy of the second BDD

This connection is always suitable if there are no repeated events in a fault tree. Otherwise, this implementation can cause ambiguous situations which require further processing. For example, consider the example in Figure 4.24. If the second appearance of an element can be reached traversing both branches of the first occurrence of an element, it is described as an ambiguous situation. Element *a* is repeated. The second appearance of element *a* (node *F4*) can be reached by

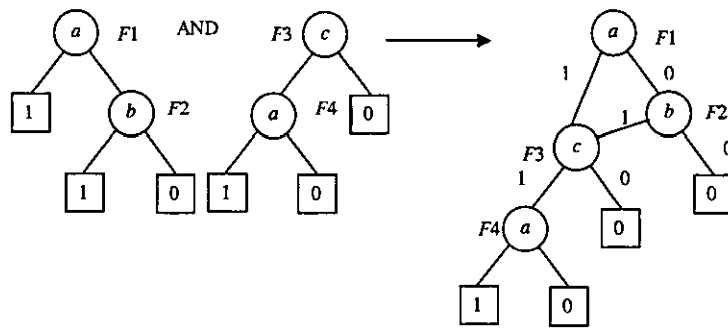


Figure 4.24: An ambiguous situation in a structure

traversing the 1 branch of the first appearance of element a (node $F1$) and by traversing the 0 branch of node $F1$. This situation is inconsistent during the simplification process.

In order to avoid ambiguous situations, they need to be identified before the connection process takes place. To do this, for every connection point that will be used in the connection process a vector of states of repeated events is required. Those alternative states, that can have a value 1, 0 or -1, define if a repeated event was visited on a path from the root node to the available point and on which branch of the first occurrence of the repeated event was traversed. So, the states 1 and 0 mean that a path includes the repeated event on its 1 and 0 branch respectively, and the state -1 describes the situation that the repeated event is not included in this path.

The sub-node sharing rule is:

If paths to two available connection points of the main BDD have the same record of repeated events, i.e. the same states of repeated events are obtained, the same copy of the second BDD can be connected to both of the two available branches

The connection algorithm which considers the sub-node sharing is shown in Figure 4.25. In the representation of this algorithm array $A(z)$ contains records of states for every repeated event throughout the whole system (not only those between two BDDs that are to be connected!), because ambiguous situations can appear during the later connections. Only if there are no more connections to proceed, i.e. the last connection is calculated, repeated events between two BDDs (instead of the whole system) can be considered. Application of the algorithm is

```

connection(node F, branch b, node G )
{
    if ((b=1) and (F=1)) /* inputs of an AND gate are considered */
    or ((b=0) and (F=0)) /* inputs of an OR gate are considered */
    {
        if (computation_table has entry {array{A(i), R}})
            return R,
        else
            /*create a connection*/
            F=copy(G);
            Simplify(F, A(i));
            insert_in_computation_table array{A(i), F}
    }
    else
    {
        if (F->node) is repeated
            adjust A(i),
            /*traverse further*/
            connection(F->left, b, G),
            connection(F->right, b, G);
    }
}

```

Figure 4.25: Connection algorithm using the sub-node sharing

presented for an example in Figure 4.26. Two BDDs, presented in Figure 4.26, are to be connected, where the first BDD is chosen to be the main BDD. Since those two BDDs represent two gate inputs of an 'OR' gate, the second BDD will be connected on every available 0 branch of the main BDD, i.e. to the 0 branches of nodes $F1$, $F2$ and $F3$. Since the only repeated event in the example fault tree is event 'a', array $A(i)$ will contain only one element $A(0)$.

The connection process starts from the root node $F1$, event 'a'. The first available vertex 0 is reached on the right of node $F1$, traversing the 0 branch of 'a', therefore, $A(0) = 0$. The second BDD is connected, as shown Figure 4.26(ii). Then the simplifications are applied and an entry (0,0) is included in the computational table, i.e. the 0 branch of the repeated event 'a' was traversed and the terminal vertex 0 was retained after the connection and simplification process. Then the second available vertex 0 is reached on the right branch of node $F2$, traversing the 1 branch of node $F1$, therefore, this vertex holds a value $A(0) = 1$. Since in the computational table there is no input with a value $A(0) = 1$, a new copy of the second BDD is created. Then it is connected to the available vertex in Figure 4.26(iii), and simplifications are applied. An entry (1, $F7$) is included in the

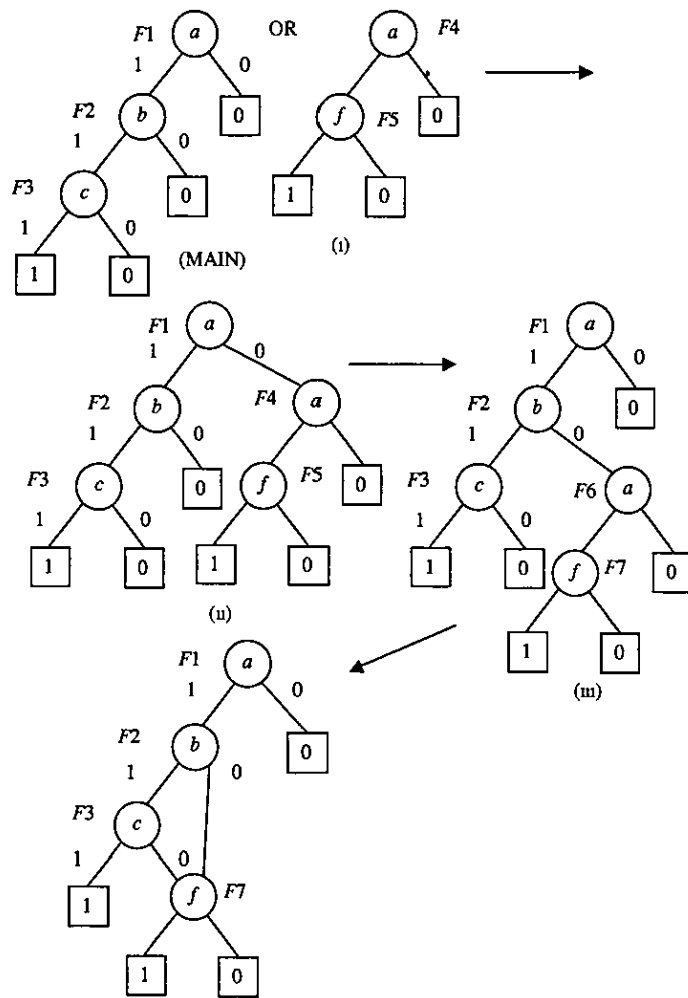


Figure 4.26: Application of the algorithm

computational table. Finally, the last available vertex 0 is visited and the record (1, $F7$) from the computational table is used, since this vertex contains the same value $A(0) = 1$, Figure 4.26(v). The connection process is finished.

The efficiency of the sub-node sharing technique in the Component Connection Method was shown by applying this technique to the library of 'large' fault trees.

4.8.2 Comparison Between the ite and the Component Connection Method Using the Sub-node Sharing

After the sub-node sharing procedure was introduced in the Component Connection method (the bottom-up approach, the third trial), the algorithm was applied

to the same library of 'large' fault trees. Then this method was compared with the **ite** method using the two main efficiency measures - the number of nodes and the processing time

Results of the number of nodes and the processing time for the Component Connection method with sub-node sharing property converting 'large' fault trees were presented in Table A 37 and in Table A 38. 44 'large' fault trees were investigated. As it was obtained in the Component Connection Method before applying the sub-node sharing, for 9 fault trees the process was not finished in reasonable times. 5 fault trees were simplified a lot, so that the final BDD contained only 1 node. Therefore, 30 fault trees were analysed. Some of the results were published in [21].

The ranking results are shown in Table 4 23 and Table 4.24 for the number of nodes and processing time respectively

		Scheme	0	1	2	3	4	5	6	7	8
Large fault trees	Total quantity ranking	Number of nodes	8958795	3787943	4526422	4458395	4262936	4023426	3824815	3972504	4046246
		Rank	9	1	8	7	6	4	2	3	5
	Highest scheme ranking	Number of FTs with the highest rank	0	12	8	9	10	14	8	11	9
		Rank	9	2	7-8	5-6	4	1	7-8	3	5-6
	Added scheme ranking	Added ranking	196	66	96	89	92	62	81	87	88
Rank		9	2	8	6	7	1	3	4	5	

Table 4.23: Component Connection Method with sub-node sharing, number of nodes

		Scheme	0	1	2	3	4	5	6	7	8
Large fault trees	Total quantity ranking	Time	3205 40	1105 64	1081 30	1092 37	1126 38	1163 28	1164 15	1118 70	1312 41
		Rank	9	3	1	2	5	6	7	4	8
	Highest scheme ranking	Number of FTs with the highest rank	6	11	8	7	5	7	9	8	6
		Rank	7-8	1	3-4	5-6	9	5-6	2	3-4	7-8
	Added scheme ranking	Added ranking	151	86	83	101	115	102	109	99	114
Rank		9	2	1	4	8	5	6	3	7	

Table 4 24: Component Connection Method with sub-node sharing, processing time

The ranking results are similar to the ones using the Component Connection Method before introducing the sub-node sharing. First of all, any of the eight

ordering schemes (1-8) performed better than the 'order as listed' scheme. The modified top-down scheme (1) as well as the non-dynamic top-down weighted scheme (5) performed well and was ranked high. For the number of nodes the two modified depth-first schemes, (2) and (3), gave worst results. However, for the processing time those two schemes gave a good performance. The depth-first with number of leaves ordering scheme (4) performed poorly and was ranked low. This repeats the pattern of the ranking in the Component Connection Method without the sub-node sharing.

The comparison of the *ite* technique and the Component Connection Method with sub-node sharing is shown in Table 4.25. The total number of nodes in final BDDs

Strategy/scheme	1	2	3	4	5	6	7	8
<i>ite</i> technique	94996	92843	73205	38500	55259	110687	76790	33042
Component Connection Method	3787943	4526422	4458395	4262936	4023426	3824815	3972504	4046246

Table 4.25: Comparison between the *ite* and the Component Connection Method with sub-node sharing, total number of nodes

using the Component Connection method was remarkably greater than using the *ite* method, despite the fact that the sub-node sharing option was introduced. However, introducing the sub-node sharing in the Component Connection Method has decreased the number of nodes, as shown in Table 4.26.

Strategy/scheme	0	1	2	3	4	5	6	7	8
Component Connection Method without the Sub-node sharing	12518644	4972154	5515658	5466903	5650688	5148732	4992002	4870866	5375739
Component Connection Method with the Sub-node sharing	8958795	3787943	4526422	4458395	4262936	4023426	3824815	3972504	4046246

Table 4.26: Comparison between the Component Connection Method without the Sub-node sharing and with the Sub-node sharing, total number of nodes

The comparison of the Component Connection Method and the *ite* technique is shown in Table 4.27. The total processing time for converting large fault trees to BDDs using the Component Connection Method was greater than using the *ite* method. Also, introducing the sub-node sharing in the Component Connection

Strategy/scheme	1	2	3	4	5	6	7	8
ite technique	207 94	71 34	70 00	102 64	29 44	57 19	116 88	38 19
Component Connection Method	1105 64	1081 3	1092 37	1126 38	1163 28	1164 15	1118 7	1312 41

Table 4.27: Comparison Between the ite and the Component Connection Method with sub-node sharing, total processing time

Method has increased the processing time, as shown in Table 4 28 It can be explained by the fact, that parts of a BDD suitable for the sub-node sharing need to be identified and this takes some extra processing time.

Strategy/scheme	0	1	2	3	4	5	6	7	8
Component Connection Method without the Sub-node sharing	298 21	81 19	90 98	93 84	105 21	97 7	83 35	83 58	102 25
Component Connection Method with the Sub-node sharing	3205 40	1105 64	1081 3	1092 37	1126 38	1163 28	1164 15	1118 7	1312 41

Table 4 28: Comparison between the Component Connection Method without the Sub-node sharing and with the Sub-node sharing, processing time

Summarising, the sub-node sharing in the Component Connection Method reduced the number of nodes in the final BDD. However, there was still the same number of unfinished fault trees like it was in the Component Connection Method without this property. Also, the processing time increased after the sub-node sharing was introduced in the method. It was shown that this extension of the method has not made it as efficient as the ite method and further development that would combine the two methods is required.

4.9 Hybrid method

The Hybrid method combines the best features of the two construction methods of BDDs, the ite method, presented in Chapter 3 and the Component Connection Method, presented earlier in this chapter. The new method incorporates the most efficient parts of both algorithms. The results in the previous section showed that

- in the Component Connection Method using the gate constructs for basic events and branches without repeated events BDDs can be immediately

formed without any of the pre-processing time required by the *ite* method

- in the *ite* technique the sub-node sharing feature provides an efficient representation of the logic function.

Therefore, a new algorithm has been created based on the effective features of each algorithm to obtain the most overall efficient approach.

In the sections presented previously, using the Component Connection Method does not require a variable ordering. However, since the new approach also uses the *ite* method a variable ordering needs to be introduced from the start of the process. This then produces ordered BDDs, which can be utilised in the *ite* technique.

4.9.1 Presentation of the Method

Rule 1

First of all, a variable ordering needs to be established which will be used when applying the rules of both methods. Then gates containing event inputs only can be considered. In this situation events are put in a chain according to the type of the gate, applying the rules of the Component Connection Method. For example, if a gate is an 'AND' gate the nodes representing its basic events are connected to each other through the 1 branch of the node. Respectively, if a gate is an 'OR' gate the nodes representing its basic events are connected to each other through the 0 branch of the node. This construction process can be applied regardless of the number of events into a gate without breaking them down into pairs. This is required in the *ite* technique because it deals only with two *ite* structures at once. The variable ordering is retained while putting basic events in a chain. This rule for the two types of gate is presented in Figure 4.27.

Rule 2

When considering gates of the fault tree which do not contain any repeated basic events - and so only basic events which occur once in the fault tree structure are considered - the straightforward connection can also be applied. However, the variable ordering needs to be taken into consideration, since it needs to be retained for

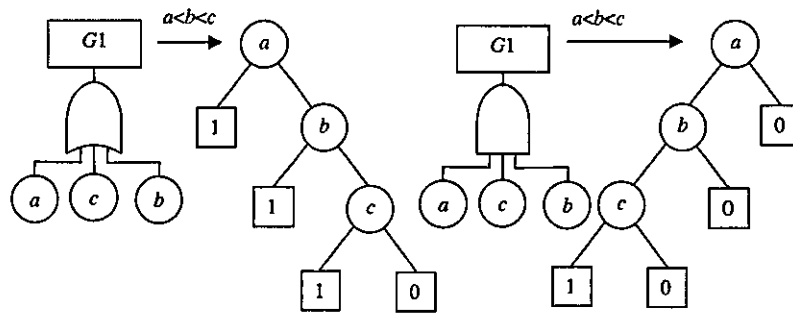


Figure 4 27: BDDs for gates containing event inputs only

the further stages of connection where **ite** rules might be applied Two possibilities are suggested.

- The merging of the two BDDs can be applied only if all the events of the main BDD are before the events of the secondary BDD in the variable ordering
- The BDDs can be merged without considering the variable ordering and then nodes must be swapped to retain the ordered BDD.

The first option is presented in Figure 4 28, where the left-most BDD is set to be the main BDD. The variable ordering is $a < b < c < d$. The events in the main BDD are before the events in the secondary BDD, therefore the BDDs can be connected in the straightforward way.

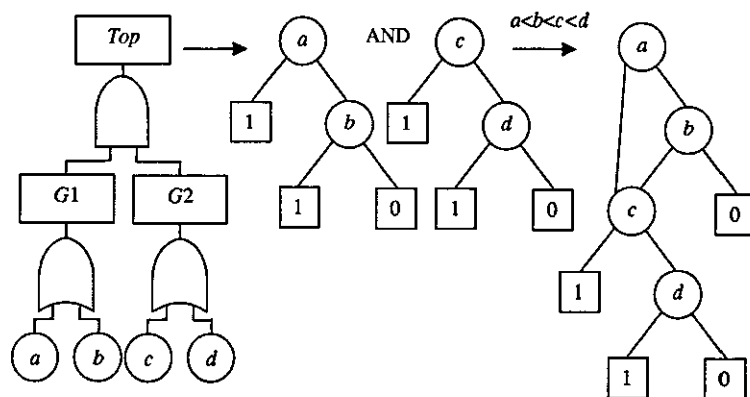


Figure 4 28: The straightforward connection of two BDDs, no reordering

In the second approach the variable ordering is not initially considered during the connection process, but interchanging some nodes is required afterwards in order to retain the variable ordering The reordering of ordered BDDs [20] is presented

in Figure 4.29. The original ordering is $y < z < w < x$, Figure 4.29(i). The new variable ordering is set to be $x < y < z < w$. According to it, nodes with variable x , that appear on the bottom of the BDD, need to be removed to the top of the BDD. First of all, variable x is swapped with variable z on the 1 branch of variable y and with variable w on the 0 branch of variable y . Variables are exchanged together with the swap of their terminal functions, Figure 4.29(ii). The second swap involves the exchange of variables y and x and the swap of their terminal function, Figure 4.29(iii). The rule is deduced by applying the distributive laws to the expression of function ϕ .

$$\begin{aligned}
 \phi &= y(z(x \cdot f_1 + \bar{x} \cdot f_2) + \bar{z}(x \cdot f_3 + \bar{x} \cdot f_4)) + & (4.38) \\
 &\quad \bar{y}(w(x \cdot f_5 + \bar{x} \cdot f_6) + \bar{w}(x \cdot f_7 + \bar{x} \cdot f_8)) \\
 &= y(x(z \cdot f_1 + \bar{z} \cdot f_3) + \bar{x}(z \cdot f_2 + \bar{z} \cdot f_4)) + \\
 &\quad \bar{y}(x(w \cdot f_5 + \bar{w} \cdot f_7) + \bar{x}(w \cdot f_6 + \bar{w} \cdot f_8)) \\
 &= x(y(z \cdot f_1 + \bar{z} \cdot f_3) + \bar{y}(w \cdot f_5 + \bar{w} \cdot f_7)) + \\
 &\quad \bar{x}(y(z \cdot f_2 + \bar{z} \cdot f_4) + \bar{y}(w \cdot f_6 + \bar{w} \cdot f_8))
 \end{aligned}$$

An example is shown in Figure 4.30.

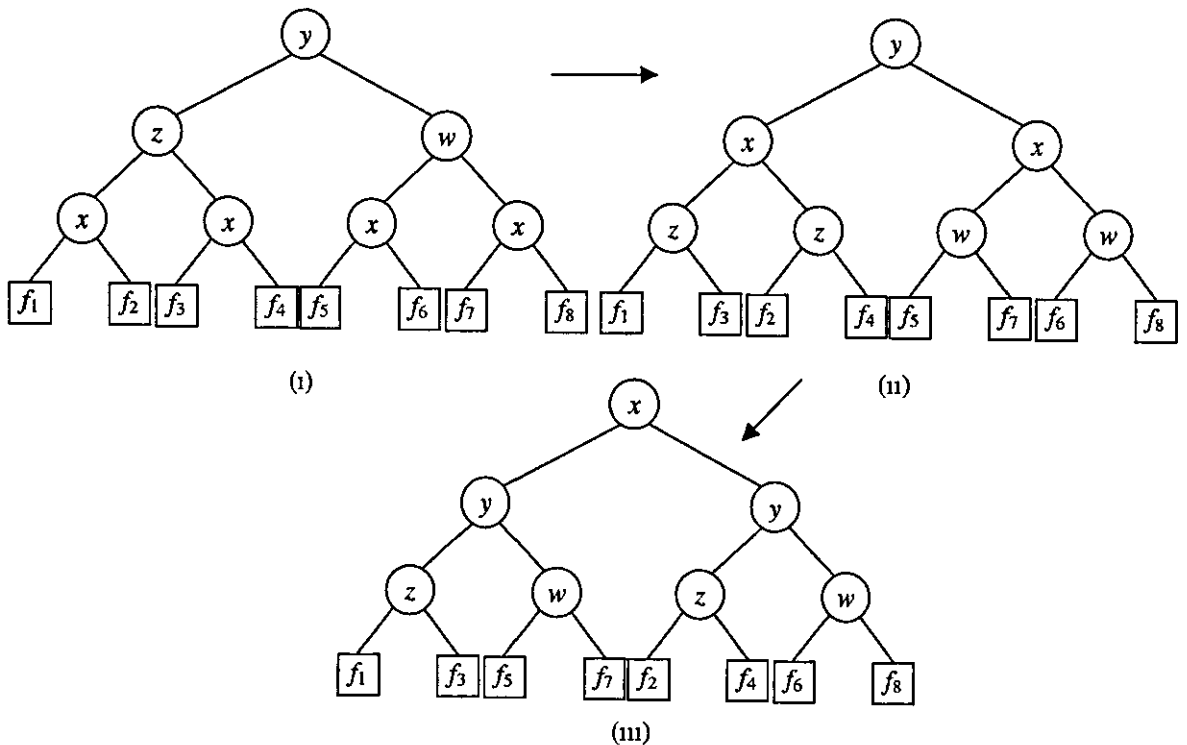


Figure 4.29: Reordering BDDs

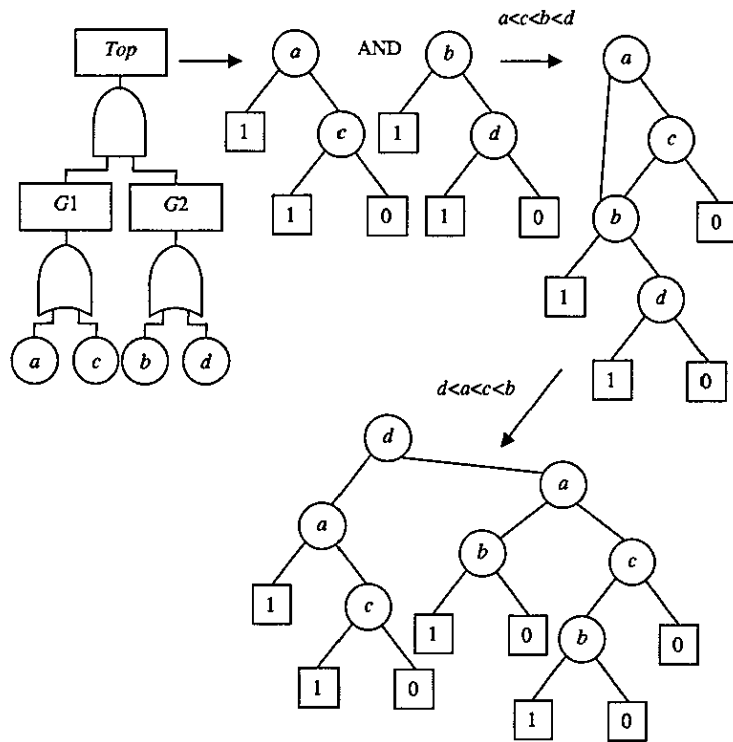


Figure 4.30: The straightforward connection of two BDDs, with reordering

Rule 3

While building the BDD for gates with repeated events, the *ite* construction rules are applied. The Hybrid method can be demonstrated by constructing a BDD for the fault tree shown in Figure 4.31.

First of all, the variable ordering needs to be introduced, $a < b < c < g < d < e < f < h$. Then the gates with event inputs only are investigated and 'AND' and 'OR' chains for gates G_1 , G_2 , G_4 and G_5 constructed, shown in Figure 4.32. After that gate G_3 is investigated. There is only one repeated event in the fault tree, a , but it does not appear as an input for gate G_3 . Therefore, the straightforward connection can be applied for this part of the fault tree. The left-most BDD is set to be the main BDD. Since not all its elements are before the elements of the secondary BDD in the ordering scheme, the second solution of rule 2 needs to be applied, i.e. the secondary BDD is attached on the 0 branches of the main BDD and then nodes are reordered. The connection is shown in Figure 4.33. Now considering the *Top* gate, again, the left-most BDD is set to be the main BDD, the

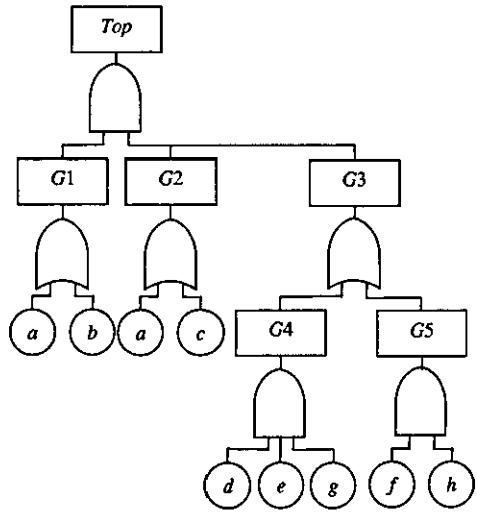


Figure 4 31: Example of fault tree

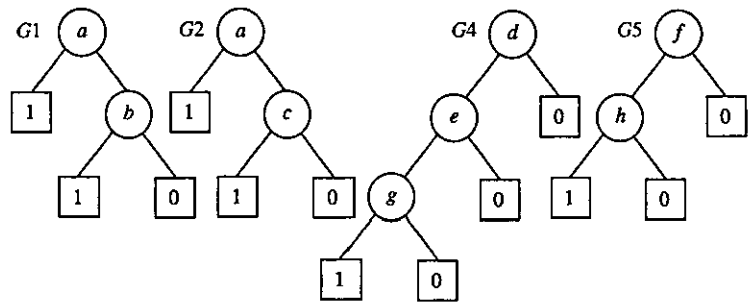


Figure 4 32. BDDs for gates with event inputs only

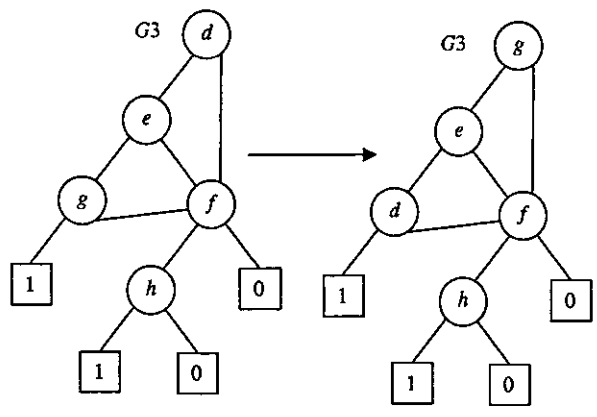


Figure 4 33· BDDs for a gate with no repeated events, node swap applied

secondary BDD is that for gate G2. These two BDDs contain the repeated event,

therefore, they are merged using the *ite* rules, i.e.

$$\begin{aligned}
 G1 \cdot G2 &= \text{ite}(a, 1, \text{ite}(b, 1, 0)) \cdot \text{ite}(a, 1, \text{ite}(c, 1, 0)) & (4.39) \\
 &= \text{ite}(a, 1, \text{ite}(b, \text{ite}(c, 1, 0), 0))
 \end{aligned}$$

The final step contains connection of the BDD of gate $G3$ onto the BDD obtained for $G1$ and $G2$. This can be performed applying the first rule. The BDD of gate $G3$ is connected to all the available '1' branches of the main BDD. The resulting BDD is pictured in Figure 4.34.

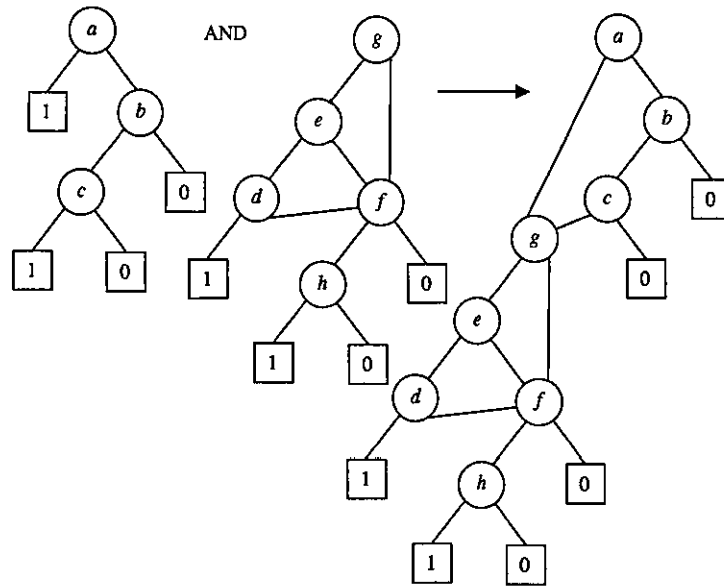


Figure 4.34: Final BDD for fault tree shown in Figure 4.31

4.9.2 Comparison Between the *ite* and the Hybrid Method

The Hybrid method was applied to the library of large fault trees, using the 8 ordering schemes for the variables, presented earlier. Two variations of the Hybrid method were used. The first variation, called the Basic Hybrid method, consisted of using rule 1 for gates with event inputs only and rule 3 for all other gates. The second variation, called the Advanced Hybrid method, used all three rules as described in the previous section. Then the Hybrid method (Basic and Advanced) was compared with the original *ite* method, using the three efficiency measures, the number of nodes, the maximum required size of the array and the processing time.

The comparison between the Basic Hybrid method and the *ite* method was done using the library of 44 'large' example fault trees used in the previous analysis. However, the efficiency of the Advanced Hybrid method was analysed using a set of small fault trees. This is because rule 2 applied in the Advanced Hybrid method only comes into use if the factorisation (the method in the reduction technique) on fault trees is not applied. If 'large' fault trees from the example library were not reduced (and not factorised) the calculations of some fault trees became impossible in reasonable times. Therefore, while comparing the Basic Hybrid method and the Advanced Hybrid method a set of smaller fault trees, that did not need to be reduced, was used in order to show the efficiency of the two techniques. Some of the results were published in [22], [23]

4.9.2.1 Comparison between the Basic Hybrid method and the *ite* technique

The number of nodes in the resulting BDDs was the same using the Hybrid (Basic or Advanced) method and the *ite* technique. This is because the Hybrid method expresses the structure function of a fault tree in terms of a BDD in the same way as it does the *ite* technique. Therefore, the ranking of the ordering schemes according to the number of nodes for the *ite* method is shown in Table 4.19 and it is the same for the Hybrid method.

Processing times for large fault trees using the *ite* technique and the Basic Hybrid method are presented in Table A 36 and Table A 39. Since the number of nodes was the same for the *ite* technique and the Hybrid method and no further analysis could be carried out, another efficiency measure, i.e. maximum required size of array, was applied. Results of the maximum required size are presented in Table A 40 and Table A 41 for the *ite* technique and the Basic Hybrid method respectively.

The ranking results are shown in Table 4.29 and Table 4.30 for the processing time and the maximum required size respectively.

The efficient process was obtained using the dynamic top-down weighted scheme (6) as well as the two depth-first ordering schemes, (3) and (4). The slowest process resulting in a large size of array was obtained using the modified top-down scheme (1). The ranking results match the results for the *ite* technique (Section 4.7.)

		Scheme	1	2	3	4	5	6	7	8
Large fault trees	Total quantity ranking	Time	15783 83	1773 32	1760 93	1808 42	21760 04	1966 18	1840 76	11798 72
		Rank	7	2	1	3	8	5	4	6
	Highest scheme ranking	Number of FTs with the highest rank	4	6	8	15	6	16	6	11
		Rank	8	5-6-7	4	2	5-6-7	1	5-6-7	3
	Added scheme ranking	Added ranking	189	143	134	114	165	101	157	154
		Rank	8	4	3	2	7	1	6	5

Table 4 29 Basic Hybrid method, processing time

		Scheme	1	2	3	4	5	6	7	8
Large fault trees	Total quantity ranking	Maximum required size	2186898	735938	731472	603274	2472127	696889	736031	1624807
		Rank	7	4	3	1	8	2	5	6
	Highest scheme ranking	Number of FTs with the highest rank	1	2	6	5	4	12	5	9
		Rank	8	7	3	4-5	6	1	4-5	2
	Added scheme ranking	Added ranking	210	150	135	154	160	106	172	146
		Rank	8	4	2	5	6	1	7	3

Table 4 30 Basic Hybrid method, maximum size of array

The comparison of the two techniques according to the processing time is shown in Table 4.31. This table shows that the Basic Hybrid method has slightly improved

Strategy\scheme	1	2	3	4	5	6	7	8
lite method	15602 82	1776 48	1764 99	1832 96	21710 86	2012 57	1839 74	11092 22
basic hybrid method	15783 83	1773 32	1760 93	1808 42	21760 04	1966 18	1840 76	11798 72

Table 4 31 Comparison of the two techniques, processing time, 'large' fault trees

the processing time for some ordering schemes, but for some ordering schemes the processing time was longer. In general, the Basic Hybrid method resulted in a comparable length of analysis process.

The comparison of the two techniques according to the maximum array size is shown in Table 4.32. This table shows that the maximum required size of the array has decreased using the Basic Hybrid method. The decrease in the maximum required size (and the processing time) can be explained by a more efficient conversion process in the Basic Hybrid method, where gates containing only event inputs do not need to be broken down to contain only two inputs. Those gates can be directly converted to a BDD according to the type of the gate. However, for

Strategy/scheme	1	2	3	4	5	6	7	8
ite method	2177226	737808	733349	605039	2473927	705358	735930	1579836
basic hybrid method	2186898	735938	731472	603274	2472127	696889	736031	1624807

Table 4.32 Comparison of the two techniques, maximum required size, 'large' fault trees

some ordering schemes the maximum required size has increased. This can be a consequence of this straightforward connection. If a pair of events, which appear in the BDD built directly from more than two events, also appear somewhere else in the fault tree, an additional **ite** structure is required. This can happen because in the Basic Hybrid method gates are not broken down to contain only two inputs and a required **ite** structure for a pair of events does not exist. In the **ite** method every gate has only two inputs and every **ite** structure for a pair of basic events is kept in memory and can be reused when required.

The comparison results of the two techniques using the maximum required array are similar to the results using the processing time. Therefore, it can be concluded that the Basic Hybrid method resulted in a shorter process for some ordering schemes because there were fewer calculations to perform and a smaller size of the array was required.

4.9.2.2 Comparison between the Basic Hybrid Method and the Advanced Hybrid method

As it was said before, the efficiency of the Advanced Hybrid method is more noticeable if fault trees are not reduced, therefore, a set of small fault trees is used, that do not need to be reduced. The complexity of the small fault trees used in this part of the analysis was shown in Tables A.1-A.5.

The results of the number of nodes for small fault trees are shown in Tables A.42-A.45. The number of nodes in the resulting BDDs was the same using the Basic Hybrid method and the Advanced Hybrid technique, because both techniques allow the structure function of a fault tree to be expressed in terms of a BDD in the same way. The results of processing time and maximum required size for small fault trees using the Basic Hybrid Method are shown in Tables A.46 - A.53. The ranking results in the Basic Hybrid method for the processing time and the maximum required size are shown in Table 4.33 and Table 4.34

		Scheme	1	2	3	4	5	6	7	8
Small fault trees	Total quantity ranking	Time	73 97	101 39	103 05	106 16	86 27	96 18	116 15	84 92
		Rank	1	5	6	7	3	4	8	2
	Highest scheme ranking	Number of FTs with the highest rank	141	3	1	93	6	14	9	12
		Rank	1	7	8	2	6	3	5	4
	Added scheme ranking	Added ranking	342	924	1016	457	876	978	1038	815
		Rank	1	5	7	2	4	6	8	3

Table 4.33 Basic Hybrid method, small fault trees, processing time

		Scheme	1	2	3	4	5	6	7	8
Small fault trees	Total quantity ranking	Maximum required size	121644	141061	137387	151249	106811	112716	141690	105958
		Rank	4	6	5	8	2	3	7	1
	Highest scheme ranking	Number of FTs with the highest rank	17	44	35	43	30	68	37	47
		Rank	8	3	6	4	7	1	5	2
	Added scheme ranking	Added ranking	1035	905	872	798	841	642	925	790
		Rank	8	6	5	3	4	1	7	2

Table 4.34 Basic Hybrid method, small fault trees, maximum size of array

The rankings are different for the two efficiency measurements. The modified top-down scheme (1) resulted faster process than any other ordering scheme but it gave poor results for the maximum required size. The dynamic top-down weighted scheme (6) and the event criticality scheme (8) gave good results according to the maximum required size but resulted in average results for the processing time.

The results of processing time and maximum required size for small fault trees using the Advanced Hybrid Method are shown in Tables A.54 - A.61. The ranking results in the Advanced Hybrid method for the processing time and the maximum required size are shown in Table 4.35 and Table 4.36.

The rankings are very similar to the ones for the Basic Hybrid method.

The comparison of the two techniques according to the processing time is shown in Table 4.37. The comparison shows that the conversion process using the Advanced Hybrid method took longer than using the Basic Hybrid method. This result was expected because of the node swap rule (rule 2) applied in the Ad-

		Scheme	1	2	3	4	5	6	7	8
Small fault trees	Total quantity ranking	Time	80 37	107 72	109 42	117 48	92 62	101 23	121 9	92 92
		Rank	1	5	6	7	2	4	8	3
	Highest scheme ranking	Number of FTs with the highest rank	137	4	4	80	13	3	10	10
		Rank	1	6-7	6-7	2	3	8	4-5	4-5
	Added scheme ranking	Added ranking	344	917	956	576	858	860	991	862
		Rank	1	6	7	2	3	4	8	5

Table 4 35: Advanced Hybrid method, small fault trees, processing time

		Scheme	1	2	3	4	5	6	7	8
Small fault trees	Total quantity ranking	Maximum required size	121248	140429	136972	150647	106399	112216	140708	105668
		Rank	4	6	5	8	2	3	7	1
	Highest scheme ranking	Number of FTs with the highest rank	16	43	36	43	31	66	39	46
		Rank	8	3	6	4	7	1	5	2
	Added scheme ranking	Added ranking	1050	909	882	799	849	646	893	793
		Rank	8	7	5	3	4	1	6	2

Table 4 36 Advanced Hybrid method, small fault trees, maximum size of array

vanced Hybrid method The ordering of nodes was not taken into account during the process, therefore, the ordering of nodes in the BDD was adjusted by swapping nodes around that required additional processing time.

The comparison of the two techniques according to the maximum required size is shown in Table 4 38. The comparison shows that using the Advanced Hybrid method decreased the maximum required size for all orderings Despite the fact that the node swap was performed, the maximum required size was smaller than in the Basic Hybrid method

Overall, the Hybrid method (Basic or Advanced) gave slightly better results than the *ite* method, where it resulted in a more efficient process of conversion This was due to the fact that the Hybrid method (Basic or Advanced) applied more 'straightforward' connections than in the *ite* technique where all connections were done using the *ite* rule for each pair of events While comparing the two types of the Hybrid method, the Advanced Hybrid method, which allowed the node swap, gave a slightly better performance than the Basic Advanced method The max-

Strategy\scheme	1	2	3	4	5	6	7	8
basic hybrid method	73 97	101 39	103 05	106 16	86 27	96 18	116 15	84 92
advanced hybrid method	80 37	107 72	109 42	117 48	92 62	101 23	121 9	92 92

Table 4.37: Comparison of the two techniques, processing time, 'small' fault trees

Strategy\scheme	1	2	3	4	5	6	7	8
basic hybrid method	121644	141061	137387	151249	106811	112716	141690	105958
advanced hybrid method	121248	140429	136972	150647	106399	112216	140708	105668

Table 4 38: Comparison of the two techniques, maximum required size, 'small' fault trees

imum required size of array was decreased using the Advanced Hybrid method instead of the Basic Hybrid method. However, this was true only for small fault trees that have not been reduced in order to test the Advanced Hybrid method.

In terms of the ordering ranking, the dynamic top-down scheme (6) and the event criticality scheme (8) performed very well, it gave the best result for the two measurements (the number of nodes and the maximum required size). For the processing time the modified top-down scheme (1) performed its best, but it gave poor results for the other two measurements

4.10 Summary

This chapter presents an alternative technique by which fault trees are converted to BDDs. The new Component Connection Method combines gate structures according to their types and applies simplification rules if repeated events appear in the structure. Example fault trees have been used and the results for a number of different connection strategies were compared. Number of nodes, processing time and the maximum required size were used as efficiency measures.

- Top-down and bottom-up approaches were introduced and analysed. It has been shown that the bottom-up technique converted fault trees to BDDs faster than the top-down approach. The slower process for the latter technique can be explained by a higher requirement of memory resources. Therefore, the top-down approach was discarded and further developments of the bottom-up method were presented

- Three trials of the bottom-up technique were presented, incorporating 8 ordering schemes for basic events and three different rules for the way of the connection of two BDDs during the building process. The connection techniques were 'as listed' method, according to the ordering of the root vertex technique and according to the number of connection points in the structure method. The third trial (according to the number of connection points) gave the best results and could be used as an efficient strategy in the Component Connection Method presented.
- It has been shown that the Component Connection Method has a high demand for memory space since the identical parts in the BDD structure are repeated but not shared. Even after the sub-node sharing introduction in the Component Connection Method, it was shown that as a general fault tree to BDD conversion technique the *ite* method was to perform a lot better than the proposed method.
- The Hybrid method, which combines the *ite* method and the Component Connection Method, was developed introducing some additional conversion rules. The Basic Hybrid Method and the Advanced Hybrid Method were analysed. It has been shown that the Hybrid method (Basic and Advanced) compares with the *ite* method well and can provide an efficient alternative tool for constructing BDDs for fault trees. The Advanced Hybrid method has a slight advantage over the Basic Hybrid method, especially in the size of the structure.
- Any of the eight ordering schemes used performed significantly better in any of the analysed construction methods than the 'order as listed' method that considers basic events in the way that they appear in the list. The four weighted ordering schemes, (5-8), resulted in a more efficient process in the majority of the methods than the four neighbourhood ordering schemes, (1-4). According to the results for the particular set of example fault trees, there were some indications that the two top-down weighted schemes, (5) and (6), were favourite according to their performance. The two modified depth-first schemes, (2) and (3), gave poor results for the majority of the methods.

Chapter 5

Non-coherent Systems

5.1 Introduction

The initial guidelines for fault tree construction for practical engineering systems recommends that failure logic should be restricted to the use of 'AND' and 'OR' gates ([1], [3]) This makes the fault tree coherent. Non-coherent structures can be obtained if the third logic operator 'NOT' is used. In this case components 'NOT' failing (working) contribute to the failure of the system. The objections for using 'NOT' logic are that it can be considered a bad design of the system if a repaired component makes the system fail. Also, it results in a remarkable increase in complexity for the analysis of the system. However, Andrews [24] demonstrated that in the case of multi-tasking systems 'NOT' logic is essential. This is also relevant for event tree analysis in which the consideration of success states is an important feature of the technique [25] The consideration of 'NOT' logic is important and even essential for some system assessments, since it gives a better understanding of the system and provides an accurate analysis.

5.2 Fault Tree Analysis of Non-coherent Fault Trees

5.2.1 Introduction

Fault trees can be described as either coherent or non-coherent systems according to their logic. If the failure logic consists of the 'AND' and 'OR' gates only, the resulting fault tree is said to be coherent. Otherwise, if the 'NOT' gate is used or directly implied, the resulting fault tree can be non-coherent.

5.2.2 The Use of NOT Logic

In this section it will be illustrated that if a system performs more than one task, the use of 'NOT' logic during fault tree construction is very important for meaningful and accurate analysis. In the multi-function system the outcomes of the system performance can produce combinations of some tasks being performed whilst others have failed. The causes of each system outcome cannot be identified correctly without accounting for the parts of the system which have worked. Consider the simplified gas detection system shown in Figure 5.1. Two gas sensors, *D1* and

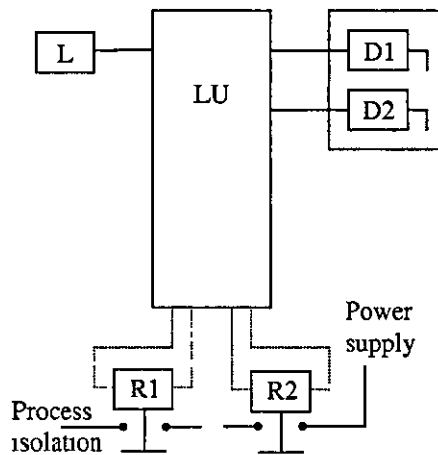


Figure 5.1. Simplified gas detection system

D2, are used to detect leakage of gas in a confined space. The signals from the detectors are sent along individual cables to the computer logic control unit, *LU*. When the signal of a gas leak from any sensor is obtained, three functions must be performed:

1. Process shut-down (isolation) by de-energising relay *R1*
2. Inform the operator of the leak by a lamp/siren labelled *L*
3. Remove the power supply (potential ignition sources) to affected areas by de-energising relay *R2*

The system can be considered failed if it does not perform any of the three tasks, following detection of the leak occurrence. There are seven possible failure states for this system, listed in Table 5.1. Consider one of these outcomes - outcome 3. Fault Tree Analysis can be performed avoiding the use of 'NOT' logic but it

Failure state	Operator informed	Process shut-down	Power isolation
1	W	W	F
2	W	F	W
3	W	F	F
4	F	W	W
5	F	W	F
6	F	F	W
7	F	F	F

Table 5.1 The seven possible failure states of the system in Figure 5.1

delivers less information. The constructed fault tree for outcome 3 is shown in Figure 5.2. Three minimal cut sets can be identified from this fault tree:

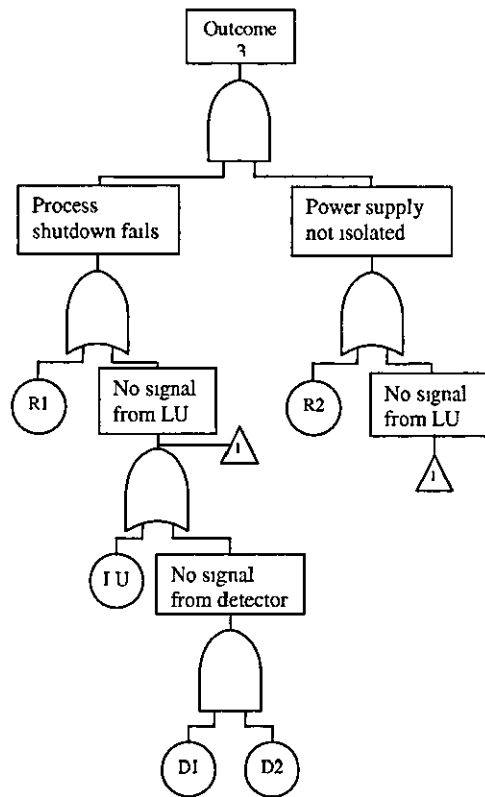


Figure 5.2. Fault tree obtained from a coherent assessment of the outcome 3

$$\{R1, R2\}, \{D1, D2\}, \{LU\} \quad (5.1)$$

Although this fault tree has been constructed in a logical manner it is inaccurate. If the operator is informed, then either $D1$ and LU or $D2$ and LU must be working.

Thus the second and the third minimal cut sets listed would not cause outcome 3 failure. Consequently quantification of this fault tree will result in an overestimate of the probability of the top event. For a correct assessment it is important to use 'NOT' logic so that the working part of the system is taken into account. The non-coherent fault tree for outcome 3 is shown in Figure 5.3. Working in a top-down

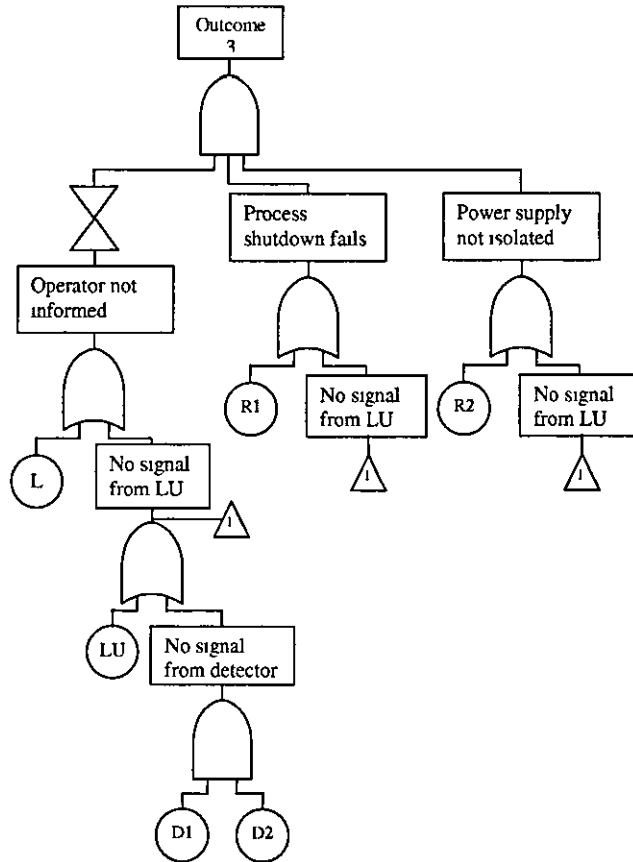


Figure 5.3: Fault tree obtained from a non-coherent assessment of the outcome 3

way the following logic expression is obtained:

$$\begin{aligned}
 Top &= (\overline{L + LU + D1 \cdot D2}) \cdot (R1 + LU + D1 \cdot D2) \cdot (R2 + LU + D1 \cdot D2) \\
 &= \overline{L} \cdot \overline{LU} \cdot (\overline{D1} + \overline{D2}) \cdot (R1 + LU + D1 \cdot D2) \cdot (R2 + LU + D1 \cdot D2) \\
 &= \overline{L} \cdot \overline{LU} \cdot R1 \cdot R2 \cdot (\overline{D1} + \overline{D2}).
 \end{aligned} \tag{5.2}$$

The *coherent approximation* of prime implicant sets can be performed, which involves identifying only the positive parts of the prime implicant sets (i.e. failing components), known as the *minimal p-cuts* of the fault tree. In this example the coherent approximation is $R1 \cdot R2$ thus the 'NOT' logic has successfully removed the inappropriate failure combinations and will enable accurate quantitative analysis to be performed. Therefore, whilst 'NOT' logic can increase the complexity

of analysis in the case of multitasking systems the use of 'NOT' logic is essential. The method used to obtain the prime implicants is presented in the next section.

5.2.3 Qualitative Analysis

The objective is to determine the combinations of component conditions (working or failed) which are necessary and sufficient to cause system failure. They are called *prime implicants*. The analysis to determine prime implicants requires more work than for the minimal cut sets in a coherent case. The order of the prime implicants, which represents the number of components, working or failed, tend to be larger than minimal cut sets due to the incorporation of the working states, which do not appear in the expression of the minimal cut sets

5.2.3.1 Calculation of prime implicants

In order to calculate the prime implicants of a non-coherent fault tree it is necessary to remove any 'NOT' gates from the fault tree structure. De-Morgan's laws presented in Equations 5.3 and 5.4 can be used to push down the 'NOT' logic down the fault tree to complement the basic events:

$$\overline{(A + B)} = \bar{A} \cdot \bar{B}, \quad (5.3)$$

$$\overline{(A \cdot B)} = \bar{A} + \bar{B} \quad (5.4)$$

To illustrate this process consider the output to the 'NOT' gate in the non-coherent fault tree in Figure 5.3

$$\begin{aligned} \overline{L + LU + D1 \cdot D2} &= \bar{L} \cdot \overline{(LU + D1 \cdot D2)} \\ &= \bar{L} \cdot \bar{LU} \cdot \overline{D1 \cdot D2} \\ &= \bar{L} \cdot \bar{LU} \cdot (\bar{D1} + \bar{D2}). \end{aligned} \quad (5.5)$$

The application of De-Morgan's laws to this fault tree results in an equivalent fault tree that contains only 'AND' and 'OR' gates, see Figure 5.4, i.e. from the original structure the effect is to interchange 'AND' and 'OR' gates and to negate the basic events

After all of the 'NOT' gates have been eliminated, the logic expression of the Top event is analysed. As a simple example, consider the fault tree illustrated in Figure 5.5. Deriving a logic expression for the Top event gives

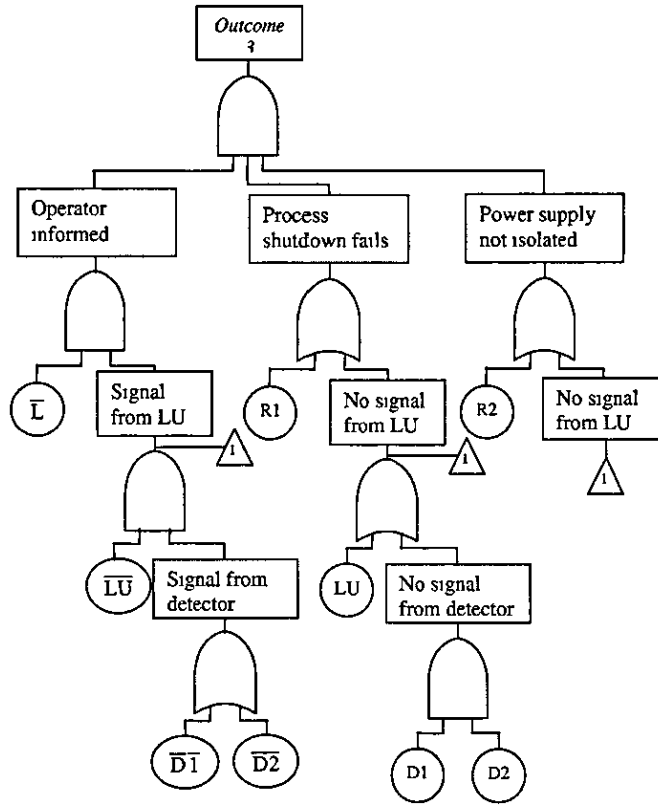


Figure 5 4 Restructured non-coherent fault tree

$$Top = a \cdot b + \bar{a} \cdot c. \quad (5.6)$$

Since this is a sum-of-products expression in its simplest form, two prime implicant sets $\{a, b\}$, $\{\bar{a}, c\}$ can be identified. However, this is not a complete list of prime implicant sets. In fact, if both component b and c are in a failed state then the system will be in a failed state regardless of the state of component a . The third prime implicant set $\{b, c\}$ can be identified by applying the consensus law, given in Equation 5.7:

$$AX + \bar{A}Y = AX + \bar{A}Y + XY \quad (5.7)$$

The following expression is obtained for the top gate

$$Top = a \cdot b + \bar{a} \cdot c + b \cdot c \quad (5.8)$$

In summary, for the identification of a full list of prime implicant sets an expression of the logic is obtained using the top-down approach. Then the consensus law is applied to pairs of prime implicant sets involving a normal and negated literal. For larger fault trees it may not be possible to identify a complete list of the prime

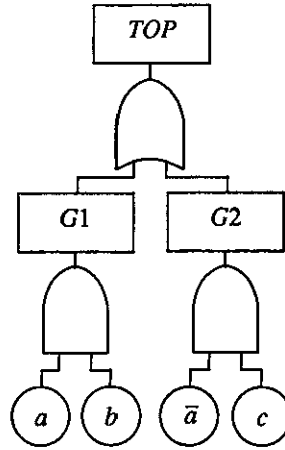


Figure 5.5: Non-coherent fault tree

implicant sets. One of the solutions to this is to obtain the minimal p-cuts that can be used to quantify the system approximately using the conventional FTA techniques.

5.2.4 Quantitative Analysis

Quantification of a non-coherent fault tree cannot be achieved using the quantification methods for coherent fault trees presented in Chapter 2 because those methods do not take into account the working states that contribute to system failure. System unavailability, unconditional failure intensity and importance measures will be presented in this section.

5.2.4.1 Calculating the system unavailability

Inagaki and Henley [26] have modified the inclusion-exclusion method used in the coherent case to enable the calculation of the system unavailability of non-coherent fault trees. $Q(t)$ is given by

$$Q(t) = \sum_{i=1}^N P(K_i) - \sum_{i=2}^N \sum_{j=1}^{i-1} P(K_i \cap K_j) + (-1)^{N-1} P(K_1 \cap K_2 \cap \dots \cap K_N), \quad (5.9)$$

where $P(K_i)$ is the probability of the existence of prime implicant set i . This probability is calculated as follows

$$P(K_i) = \prod_{j=1}^{N_p} q^{x_j}(t), \quad (5.10)$$

where

$$q^{x_j}(t) = \begin{cases} q_j(t) & \text{if } x_j = 1, \text{ i.e. } j \text{ appears,} \\ p_j(t) & \text{if } x_j = 0, \text{ i.e. } \bar{j} \text{ appears,} \end{cases} \quad (5.11)$$

$$p_j = 1 - q_j \quad (5.12)$$

q^{x_j} stands for the probabilities of literals contained in any of the prime implicant sets. Conflicting literals, i.e. $x_i \bar{x}_i = 0$, and all redundancies, i.e. $x_i \cdot x_i = x_i$, are eliminated from prime implicant sets. N_p denotes the number of basic events in a prime implicant set.

When calculations can be unmanageable even for moderate sized fault trees the approximations are applied as it was presented in the coherent case. An alternative means is to obtain a coherent approximation for qualitative analysis and use this to calculate the Rare Event Approximation and the Minimal Cut Set Upper Bound presented in Chapter 2.

5.2.4.2 Calculating the unconditional failure intensity

The calculation of the unconditional failure intensity that was introduced in Chapter 2 was extended by Inagaki and Henley for use with non-coherent fault trees.

The top event occurs in the interval $[t, t + dt)$ if and only if no prime implicant sets exist at time t and at least one prime implicant set occurs in the interval $[t, t + dt)$:

$$w_{sys}(t)dt = P\left(\bigcup_{i=1}^N K_i\right) - P\left(\bar{A} \bigcup_{i=1}^N K_i\right) = w_{sys}^{(1)}(t) - w_{sys}^{(2)}(t)dt, \quad (5.13)$$

here A is the event that at least one prime implicant set exists at time t and $\bigcup_{i=1}^N K_i$ is the event that one or more prime implicants K_i occur in time $[t, t + dt)$. The probability $P\left(\bigcup_{i=1}^N K_i\right)$ is calculated as follows

$$P\left(\bigcup_{i=1}^N K_i\right) = \sum_{j=1}^{N_p} z^{x_j}(t) \prod_{l=1, l \neq j}^{N_p} q^{x_l}(t), \quad (5.14)$$

where

$$q^{x_l}(t) = \begin{cases} q_l(t) & \text{if } x_l = 1 \\ p_l(t) & \text{if } x_l = 0 \end{cases}, \quad z^{x_j}(t) = \begin{cases} w_j(t) & \text{if } x_j = 1 \\ v_j(t) & \text{if } x_j = 0 \end{cases} \quad (5.15)$$

x_i is set to 1 if the literal exists in its positive form and 0 if the literal is negated $w_j(t)$ is the unconditional failure intensity of component j and $v_j(t)$ is the unconditional repair intensity of component j .

Both terms in Equation 5.13 can be expanded using the inclusion-exclusion formula. The calculation of unconditional failure intensity is a time consuming and exhaustive process. Rare Event Approximation and Upper Bound Approximation can be used as it was presented in the coherent case in Chapter 2.

The second method was developed by Becker and Camarinopoulos [27] where the unconditional failure intensity can be expressed as the probability that the system is in a critical state for one or more components at time t and one of those critical components fails in the interval $[t, t + dt)$. In the non-coherent case components can be both failure and repair critical, therefore two types of system criticality functions are defined:

- **Failure criticality function.** System is working when component i is working and failed when component i is failed.

$$\phi_i^f = Q(1_i, \mathbf{q}(t))(1 - Q(0_i, \mathbf{q}(t))). \quad (5.16)$$

- **Repair criticality function.** System is working when component i is failed and working when component i is working:

$$\phi_i^r = Q(0_i, \mathbf{q}(t))(1 - Q(1_i, \mathbf{q}(t))). \quad (5.17)$$

From this the unconditional failure intensity can be calculated:

$$w_{sys} dt = \sum_{i=1}^{N_C} \phi_i^f w_i dt + \sum_{i=1}^{N_C} \phi_i^r v_i dt. \quad (5.18)$$

This method can only be applied if $Q(1_i, \mathbf{q}(t))$ and $Q(0_i, \mathbf{q}(t))$ are independent.

5.2.4.3 Importance measures

Birnbaum's Measure of component reliability importance is a fundamental measure of importance. It calculates the probability that component i is critical to the system state. In the non-coherent case this probability can be expressed as the probability that component i is repair critical, $G_i^R(\mathbf{q}(t))$, or the probability that component i is failure critical, $G_i^F(\mathbf{q}(t))$, as shown in [28]:

$$G_i(\mathbf{q}(t)) = G_i^R(\mathbf{q}(t)) + G_i^F(\mathbf{q}(t)). \quad (5.19)$$

The top event can only exist at time t if at least one prime implicant set exists at time t . Hence the failure and repair criticality can be calculated separately by differentiating the system unavailability function, $Q(t)$, with respect to $q_i(t)$ and $p_i(t)$ respectively

$$G_i^F(\mathbf{q}(t)) = \frac{\partial Q(t)}{\partial q_i(t)}, \quad (5.20)$$

$$G_i^R(\mathbf{q}(t)) = \frac{\partial Q(t)}{\partial p_i(t)} \quad (5.21)$$

In terms of Birnbaum's measure of component reliability importance the expected number of system failures can be calculated as:

$$W(0, t) = \int_0^t \left(\sum_{i=1}^{N_C} G_i^F(\mathbf{q}(t)) w_i(u) + \sum_{i=1}^{N_C} G_i^R(\mathbf{q}(t)) v_i(u) \right) du, \quad (5.22)$$

where $w_i(t)$ denotes the component unconditional failure intensity, $v_i(t)$ denotes the component unconditional repair intensity and N_C is the total number of system components. The first term in the equation calculates the number of occurrences of system failure due to the failure of component i in a given time interval and the second term is the number of occurrences of system failure due to the repair of component i in a given time interval.

Component Failure Criticality Measure of Importance is defined as the probability that component i is failure critical to the system and i has failed weighted by the system unavailability:

$$I_i^F = \frac{G_i^F(\mathbf{q}(t)) q_i(t)}{Q(t)} \quad (5.23)$$

Similarly, *Component Repair Criticality Measure of Importance* is calculated as the probability that component i is repair critical and is in a working state weighted by the system unavailability:

$$I_i^R = \frac{G_i^R(\mathbf{q}(t)) p_i(t)}{Q(t)} \quad (5.24)$$

The total criticality measure of importance is obtained by summing the failure and repair part:

$$I_i = I_i^F + I_i^R. \quad (5.25)$$

Fussell-Vesely's Measure of Component Importance can also be extended for non-coherent analysis. The Fussell-Vesely failure importance is expressed as

$$I_i^F = \frac{P(\bigcup_{j|i \in K_j} K_j)}{Q(t)} \quad (5.26)$$

The Fussell-Vesely repair importance is calculated

$$I_i^R = \frac{P(\bigcup_{j \in K_j} K_j)}{Q(t)}. \quad (5.27)$$

5.3 Simplification Process of Non-coherent Fault Trees

Dealing with complex industrial systems can result in very large fault trees, whose analysis is time consuming. As it was presented in the coherent case in Chapter 2 two pre-processing techniques can be applied to the fault tree in order to obtain the smallest possible subtrees and reduce the size of the problem. The first part of the simplification process is a reduction technique which resizes the fault tree to its simplest form. The second part identifies independent modules (subtrees) within the fault tree that can be dealt with separately. The linear-time algorithm is applied to the second part and a set of independent fault trees in their simplest possible structure is obtained. It is equivalent to the original system failure causes and is easier to manipulate during the analysis process.

5.3.1 Faunet Reduction in the Non-coherent Fault Tree Case

The Faunet Reduction technique for non-coherent FTs can be described in four stages, as it was presented in the coherent case

1. Contraction
2. Factorisation
3. Extraction
4. Absorption.

First of all, the fault tree is manipulated so that the NOT logic is 'pushed' down the fault tree until it is applied to basic events using De Morgan's laws, presented in Equations 5.3 and 5.4.

For the contraction, subsequent gates of the same type are contracted to form a single gate so that the fault tree becomes an alternating sequence of 'AND' and 'OR' gates

During the factorisation, pairs of events that always occur together as inputs to the same gate type are identified and combined forming a single complex event. If events appear in their working and failed states in the fault tree, only those basic events that appear together in their negated state under the opposite gate type can be combined. (Note. in this sense the 'AND' type gate is opposite to the 'OR' type gate). By De Morgan's equations 5.3 and 5.4, if $a + b$ and/or $\bar{a} \cdot \bar{b}$ appear in the fault tree, then $a + b$ forms a complex event, or if $a \cdot b$ and/or $\bar{a} + \bar{b}$ appear in the fault tree, then $a \cdot b$ forms a complex event. The complex events identified are then substituted into the fault tree structure.

In the extraction stage, the two structures shown in Figure 2.2 are identified and replaced in order to reduce the repeated occurrence of events to a single occurrence and facilitate further reduction. If another component is repeated in the structure and it is repeated in its negated state, the structures shown in Figure 5.6 can be simplified even more, i.e. the whole structure is replaced by the component that appears only in one state, failed or working.

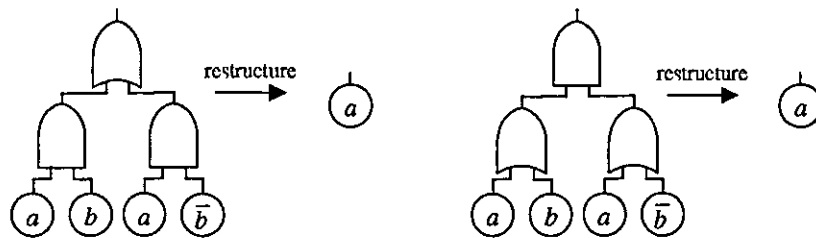


Figure 5.6. Extraction procedure in non-coherent case

During absorption, structures were identified that could be further simplified through the application of the absorption and idempotent laws to the fault tree logic, Figure 2.3. If a component is repeated in its negated state, the absorption rule can also be applied. In this case the absorption cannot be applied if the primary gate is an 'OR' gate. Therefore, if the primary gate is an 'AND' gate and the secondary gate is an 'OR' gate, then the structure is simplified by deleting the occurrence of the event beneath the secondary gate. If both the primary gate and the secondary gate are 'AND' gates, the whole secondary gate can be deleted. These situations are presented in Figure 5.7. The order of appearance of positive and negative events in primary and secondary gates is irrelevant. The above four steps are repeated until no further changes take place in the fault tree.

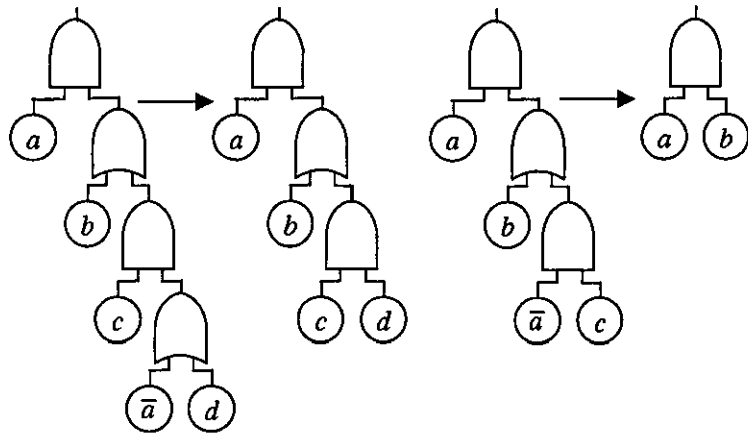


Figure 5.7. Absorption procedure in non-coherent case

Consider a simple example in Figure 5.8, on the left the fault tree in terms of its original components is presented and on the right the fault tree in its numerical form is shown. An array of negated data is also assigned that holds the information

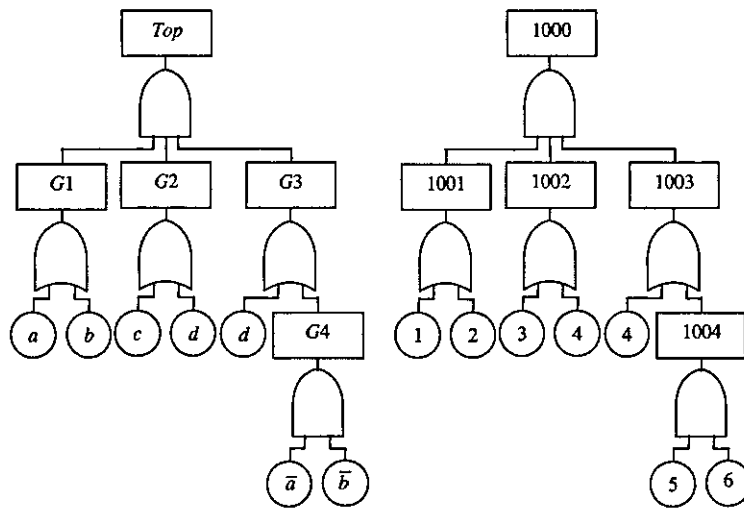


Figure 5.8 Non-coherent fault tree for reduction

that basic events 5 and 6 are negated forms of basic events 1 and 2.

The fault tree has an alternating sequence of 'AND' and 'OR' gates, therefore the contraction is not needed

The input events to each gate are considered one by one, looking for pairs that always occur together. The factorised fault tree is shown in Figure 5.9 The com-

plex events are shown in Table 5 2

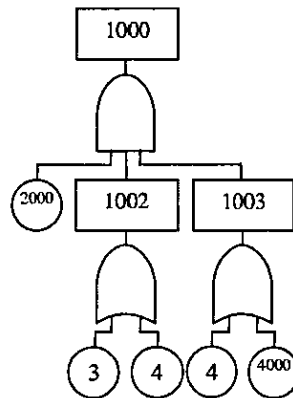


Figure 5 9 Non-coherent fault tree after factorisation

Complex event	Gate value	Event 1	Event 2
2000	OR	a	b
4000	AND	\bar{a}	\bar{b}

Table 5 2 Complex event data for non-coherent fault tree

The number of repeated events can be minimised during the extraction process. The only gate that has two or more gate inputs is top gate 1000, whose inputs are 1002 and 1003. These secondary gates are both of a different type to the primary gate, and have basic event 4 in common, that can be extracted. In order to get this into the required form for the extraction, gate 1005 is generated. Another new gate 1006 is created which is of the same type as the secondary gates and has the same common output, event 4, and the primary gate 1000, as inputs. The sequence of the extraction procedure is shown in Figure 5 10.

Finally, the only repeated event in the fault tree is event 2000. It is repeated in its negated form. The first time it occurs as an input to the primary gate 1005, which is a 'AND' gate, and the second time it occurs in its negated form as an input to the secondary gate, which is also an 'AND' gate. Since the types of the primary and the secondary gates are the same the secondary gate can be deleted. The fault tree after the absorption is shown in Figure 5 11.

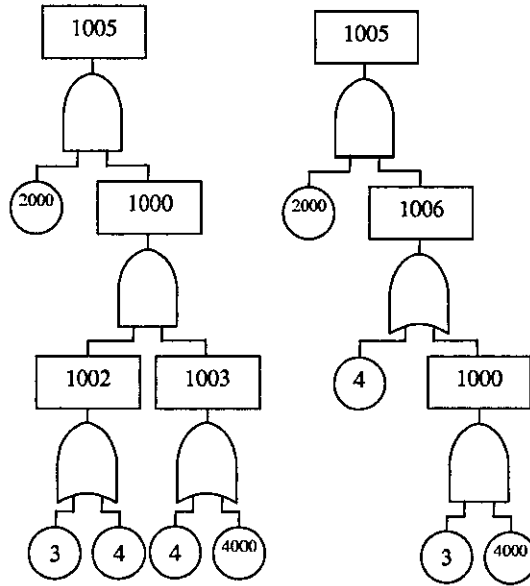


Figure 5.10: Non-coherent fault tree during extraction, gate 1005 and 1006 created

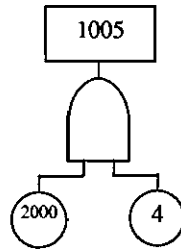


Figure 5.11: Non-coherent fault tree after absorption

The reduction process is completed.

5.3.2 Linear Modularisation in the Non-coherent FT Case

Modularisation identifies independent modules within the fault tree that can be analysed separately from the rest of the tree. The linear-time algorithm in the non-coherent case can be applied in the same way like it was presented in the coherent case in Chapter 2. The state of the basic event is not relevant to the traversal and the modularisation of the fault tree.

The application of the simplification techniques in non-coherent case has been published in [29]

5.4 Summary

The use of NOT logic during fault tree construction can add to the complexity of analysis. However, it has been demonstrated by Andrews [24] that NOT logic can be essential for meaningful and accurate analysis of certain systems. Therefore, it is essential to be able to analyse non-coherent fault trees accurately.

Conventional methods of fault tree analysis have been extended for the purposes of non-coherent FTA. However, NOT logic increases the complexity of the analysis, and even for moderate sized examples the analysis might not always be possible. Although a coherent approximation can be used to reduce the work required for the analysis, the techniques are still computationally intensive.

The fault tree diagram is a useful description of the system being analysed, but alternative techniques for both qualitative and quantitative analysis are required, so that the efficiency and the accuracy could be improved. The BDD method, as a means for a better analysis of non-coherent fault trees, is presented in the next chapter.

Chapter 6

The BDD Method for the Analysis of Non-coherent Fault Trees

6.1 Introduction

Conventional techniques of FTA can be used to perform the qualitative and quantitative analysis of non-coherent fault trees. However, even for moderate sized trees approximations are unavoidable. The BDD method is more efficient and accurate than conventional FTA methods. The fault tree is converted to the SFBDD (structure function binary decision diagram) from which exact analysis is performed.

However, it is not possible to identify the prime implicant sets directly from the SFBDD, this requires more calculations. A full set of prime implicants is determined by applying the consensus theorem [24] to pairs of prime implicant sets involving a normal and negated literal. A new alternative method for performing the qualitative analysis of non-coherent fault trees is proposed as a part of this research. In this approach a fault tree is converted to a Ternary Decision Diagram (TDD). The main concept of a TDD was presented by Sasao [30]. The method was developed further introducing the conversion rules from fault trees to TDDs, applying the minimisation technique and presenting an efficient way to obtain prime implicant sets.

There are several methods for the calculation of prime implicant sets proposed in the literature. The first of these methods was introduced by Courdet and Madre [31] and then developed by Rauzy and Dutuit [32]. This method involves calculation of the Meta-products BDD from which prime implicant sets can be

identified. The second alternative method presented by Rauzy [33] uses the obtained SFBDD and converts it to the Zero-suppressed BDD (ZBDD), presented by Minato [34]. The third alternative method produces a Labelled Binary Decision Diagram (L-BDD), presented by Contini [35]. All these methods produce prime implicant sets and have their advantages and disadvantages in the conversion and representation techniques. Therefore, the analysis will be performed and the efficiency of the proposed TDD method estimated.

6.2 Computing the SFBDD in Non-coherent Case

The SFBDD for a non-coherent fault tree is computed using the same *ite* procedure presented in Chapter 3. The only extra rule is the *ite* structure for negated events. In this case, the one and zero branches have been switched compared to the *ite* expression for the positive event:

$$\bar{x} = \text{ite}(x, 0, 1), \quad (6.1)$$

Consider the fault tree example given in Figure 5.5. Introducing a variable ordering $b < a < c$ and assigning each basic event an *ite* structure gives:

$$a = \text{ite}(a, 1, 0), \quad (6.2)$$

$$\bar{a} = \text{ite}(a, 0, 1), \quad (6.3)$$

$$b = \text{ite}(b, 1, 0), \quad (6.4)$$

$$c = \text{ite}(c, 1, 0) \quad (6.5)$$

$G1$ is expressed

$$G1 = b \cdot a \quad (6.6)$$

$$= \text{ite}(b, 1, 0) \cdot \text{ite}(a, 1, 0)$$

$$= \text{ite}(b, \text{ite}(a, 1, 0), 0)$$

Then dealing with $G2$ gives.

$$G2 = \bar{a} \cdot c \quad (6.7)$$

$$= \text{ite}(a, 0, 1) \cdot \text{ite}(c, 1, 0)$$

$$= \text{ite}(a, 0, \text{ite}(c, 1, 0)).$$

Finally the *Top* gate is expressed

$$\begin{aligned}
 Top &= G1 + G2 && (6.8) \\
 &= \text{ite}(b, \text{ite}(a, 1, 0), 0) + \text{ite}(a, 0, \text{ite}(c, 1, 0)) \\
 &= \text{ite}(b, \text{ite}(a, 1, \text{ite}(c, 1, 0)), \text{ite}(a, 0, \text{ite}(c, 1, 0))).
 \end{aligned}$$

So, this is the *ite* structure computed for the fault tree from Figure 5.5 and the SFBDD is shown in Figure 6.1.

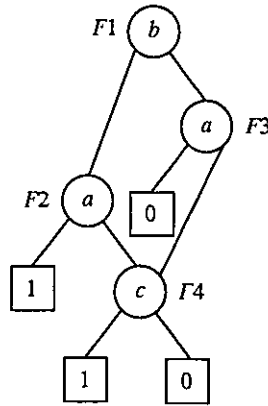


Figure 6 1: SFBDD for non-coherent fault tree

6.3 Qualitative Analysis

Knowledge of prime implicant sets can be valuable in gaining an understanding of the system under analysis. It can help to develop a repair schedule for failed components if a system cannot be taken off line for repair. For example, there is a system with three components A, B, C . One failure state of the system is represented by the prime implicant set $\{A, B, \bar{C}\}$ and this failure state results in a hazard. Therefore, it is important to know this prime implicant set. If all components have failed, it is obvious from this set that component C should not be repaired until either component A or B had been repaired. Hence unnecessary system failures could be avoided.

The SFBDD which encodes the structure function cannot be used directly to produce the complete list of prime implicant sets of a non-coherent fault tree. For example, consider a general component x in a non-coherent system. Component x can be in a failed or working state, or can be excluded from the failure mode. In the

first two situations x is said to be relevant, in the third case it is irrelevant to the system state. Component x can be either failure relevant (the prime implicant set contains x) or repair relevant (the prime implicant set contains \bar{x}). A general node in the SFBDD, which represents component x , has two branches. The 1 branch corresponds to the failure of x ; therefore, x is either failure relevant or irrelevant. Similarly, the 0 branch corresponds to the functioning of x and so x is either repair relevant or irrelevant. Hence it is impossible to distinguish between the two cases for each branch and the prime implicant sets cannot be identified from the SFBDD.

The SFBDD encodes the structure function of the fault tree and its minimal form can only be used to obtain a coherent approximation for qualitative analysis. This is presented later in this section. In order to obtain prime implicant sets additional calculations are required. A new method will be proposed for the qualitative analysis and its efficiency will be compared with the established methods.

6.3.1 Coherent Approximation

The coherent approximation for qualitative analysis involves identifying only the positive parts of the prime implicants, known as minimal p-cuts. The SFBDD of a non-coherent fault tree needs to be minimised, removing non-minimal cut sets and then a full list of minimal p-cuts is obtained tracing the paths to a terminal 1 through the minimised structure. The SFBDD in Figure 6.1 is non-minimal and thus must be minimised before the minimal p-cuts can be identified exactly. Traversing node $F4$ on the zero branch of node $F2$ results in the non-minimal combination $\{b, c\}$. Therefore the zero branch of node $F2$ is terminated with 0 resulting in the minimised SFBDD shown in Figure 6.2. This SFBDD produces two minimal p-cuts

$$\{a, b\}, \{c\} \quad (6.9)$$

They are the coherent approximation of the three prime implicant sets

$$\{a, b\}, \{\bar{a}, c\}, \{b, c\}. \quad (6.10)$$

6.3.2 Ternary Decision Diagram Method

A new approach to build a Ternary Decision Diagram (TDD) for the analysis of non-coherent fault trees is proposed in this section. It employs the consensus theorem and creates, in addition to the two branches of the BDD, a third branch for

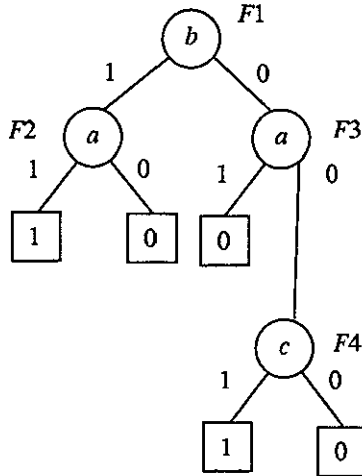


Figure 6.2. Minimised SFBDD

every node, called the consensus branch. This third branch encodes the 'hidden' prime implicant sets. The minimisation algorithm [2] is applied to remove non-minimal paths and obtain prime implicant sets only.

The representation now lends itself to a node structure with three exit branches. A new **ifre** structure is presented which distinguishes not only between relevant and irrelevant components but also it distinguishes between the type of relevancy, i.e. failure relevant and repair relevant. The **ifre** structure for a component x is given below:

$$\text{ifre}(x, f_1, f_0, f_2). \quad (6.11)$$

The 1 branch encodes prime implicant sets for which component x is failure relevant, the 0 branch encodes prime implicant sets for which component x is repair relevant, and the 'C' branch encodes prime implicant sets for which component x is irrelevant. The **ifre** structure shown in Figure 6.3 can be interpreted as follows:

$$\text{If } x \text{ is failure relevant} \quad \text{consider function } f_1, \quad (6.12)$$

$$\text{or } x \text{ is repair relevant} \quad \text{consider function } f_0, \quad (6.13)$$

$$\text{else} \quad \text{consider function } f_2 \quad (6.14)$$

Function f_2 encodes prime implicant sets for which x is irrelevant, but this branch is not important for all components. For components that are only failure or repair relevant, but not both, this branch can be kept 'empty'. In this method $f_2 = NIL$.

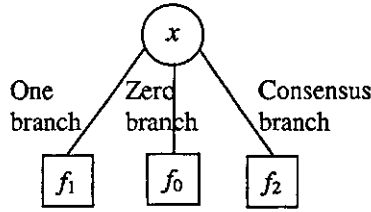


Figure 6.3. Three-way ite structure

is assigned, if the conjunction of the two branches $f_1 \cdot f_0$ is not required. While operating the new symbol in the Boolean algebra, it is defined that $NIL < op > A = NIL$. Symbol NIL is used to identify cases when the 'C' branch is not required and no Boolean operations that involve this branch are needed.

6.3.2.1 Computing the TDD

The conversion process for computing the TDD is similar to the previous method. Basic events of the fault tree must be ordered. Then the following process is used:

1. Assign each basic event x an **ifre** structure.
If a is only failure or repair relevant, then:

$$x = \text{ifre}(x, 1, 0, NIL), \quad (6.15)$$

$$\bar{x} = \text{ifre}(x, 0, 1, NIL) \quad (6.16)$$

If a is failure and repair relevant, then:

$$x = \text{ifre}(x, 1, 0, 0), \quad (6.17)$$

$$\bar{x} = \text{ifre}(x, 0, 1, 0) \quad (6.18)$$

2. By the application of De Morgan's laws push any 'NOT' gates down through the fault tree until it reaches basic event level.
3. If the two gate inputs are G and H such that:

$$G = \text{ite}(x, F_1, F_0, F_2), \quad (6.19)$$

$$H = \text{ite}(y, H_1, H_0, H_2), \quad (6.20)$$

then the following rules are applied:

$$\text{If } x < y, G < op > H = \text{ifre}(x, K_1, K_0, [K_1 \cdot K_0]), \quad (6.21)$$

where $K_1 = F_1 < op > H$ and $K_0 = F_0 < op > H$, and $K_1 \cdot K_0$ represents the consensus of K_1 and K_0

$$\text{If } x = y, G < op > H = \mathbf{ifre}(x, L_1, L_0, [L_1 \cdot L_0]), \quad (6\ 22)$$

where $L_1 = F_1 < op > H_1$ and $L_0 = F_0 < op > H_0$, and $L_1 \cdot L_0$ represents the consensus of L_1 and L_0

Remark. If component x is failure or repair relevant, $K_1 \cdot K_0 = NIL$ and $L_1 \cdot L_0 = NIL$ in Equations 6.21 and 6.22.

These rules are used in conjunction with the following identities:

$$1 < op > H = H, \quad 0 < op > H = 0, \quad (6\ 23)$$

if $< op >$ is an 'AND' gate

$$1 < op > H = 1, \quad 0 < op > H = H, \quad (6\ 24)$$

if $< op >$ is an 'OR' gate.

Within each **ite** calculation an additional consensus calculation is performed to ensure all the 'hidden' prime implicant sets are encoded in the BDD obtained. It calculates the product of the 1 and the 0 branch of every node and thus identifies the consensus of each node. If a node in the TDD encodes a component which is only failure or repair relevant the conjunction of the 1 and 0 branch for the node is not required, because there are no 'hidden' prime implicant sets associated with this component. This property makes the TDD method an efficient technique for performing the qualitative analysis of non-coherent fault trees.

Consider the non-coherent fault tree in Figure 5.5. A variable ordering is $b < a < c$. Component b is failure relevant, component c is repair relevant and component a is both, failure and repair, relevant. Each variable is assigned an **ifre** structure:

$$a = \mathbf{ifre}(a, 1, 0, 0), \quad b = \mathbf{ifre}(b, 1, 0, NIL), \quad (6\ 25)$$

$$\bar{a} = \mathbf{ifre}(a, 0, 1, 0), \quad c = \mathbf{ifre}(c, 1, 0, NIL).$$

Computing the **ifre** structure for gate $G1$:

$$G1 = b \cdot a \quad (6\ 26)$$

$$= \mathbf{ifre}(b, 1, 0, NIL) \cdot \mathbf{ifre}(a, 1, 0, 0)$$

$$= \mathbf{ifre}(b, f_1, f_0, f_1 \cdot f_0),$$

where

$$\begin{aligned} f_1 &= 1 \cdot \text{ifre}(a, 1, 0, 0) = \text{ifre}(a, 1, 0, 0), \\ f_0 &= 0 \cdot \text{ifre}(a, 1, 0, 0) = 0, \\ f_1 \cdot f_0 &= \text{NIL}. \end{aligned} \quad (6.27)$$

Therefore, the **ifre** structure for gate $G1$ is given below:

$$\text{ifre}(b, \text{ifre}(a, 1, 0, 0), 0, \text{NIL}) \quad (6.28)$$

Dealing with gate $G2$:

$$\begin{aligned} G2 &= \bar{a} \cdot c \\ &= \text{ifre}(a, 0, 1, 0) \cdot \text{ifre}(c, 1, 0, \text{NIL}) \\ &= \text{ifre}(a, f_1, f_0, f_1 \cdot f_0), \end{aligned} \quad (6.29)$$

where

$$\begin{aligned} f_1 &= 0 \cdot \text{ifre}(c, 1, 0, \text{NIL}) = 0, \\ f_0 &= 1 \cdot \text{ifre}(c, 1, 0, \text{NIL}) = \text{ifre}(c, 1, 0, \text{NIL}), \\ f_1 \cdot f_0 &= 0 \cdot \text{ifre}(c, 1, 0, \text{NIL}) = 0 \end{aligned} \quad (6.30)$$

Hence the **ifre** structure for gate $G2$ is

$$\text{ifre}(a, 0, \text{ifre}(c, 1, 0, \text{NIL}), 0) \quad (6.31)$$

Finally, calculating the top gate Top :

$$\begin{aligned} Top &= G1 + G2 \\ &= \text{ifre}(b, \text{ifre}(a, 1, 0, 0), 0, \text{NIL}) + \text{ifre}(a, 0, \text{ifre}(c, 1, 0, \text{NIL}), 0) \\ &= \text{ifre}(b, f_1, f_0, f_1 \cdot f_0), \end{aligned} \quad (6.32)$$

where

$$\begin{aligned} f_1 &= \text{ifre}(a, 1, 0, 0) + \text{ifre}(a, 0, \text{ifre}(c, 1, 0, \text{NIL}), 0) \\ &= \text{ifre}(a, 1, \text{ifre}(c, 1, 0, \text{NIL}), \text{ifre}(c, 1, 0, \text{NIL})), \\ f_0 &= 0 + \text{ifre}(a, 0, \text{ifre}(c, 1, 0, \text{NIL}), 0) = \text{ifre}(a, 0, \text{ifre}(c, 1, 0, \text{NIL}), 0), \\ f_1 \cdot f_0 &= \text{NIL} \end{aligned} \quad (6.33)$$

The final **ifre** structure obtained for the fault tree in Figure 5.5

$$\text{ifre}(b, \text{ifre}(a, 1, \text{ifre}(c, 1, 0, \text{NIL}), \text{ifre}(c, 1, 0, \text{NIL})), f, \text{NIL}), \quad (6.34)$$

where

$$f = \text{ifre}(a, 0, \text{ifre}(c, 1, 0, \text{NIL}), 0) \quad (6.35)$$

The obtained TDD is shown in Figure 6.4

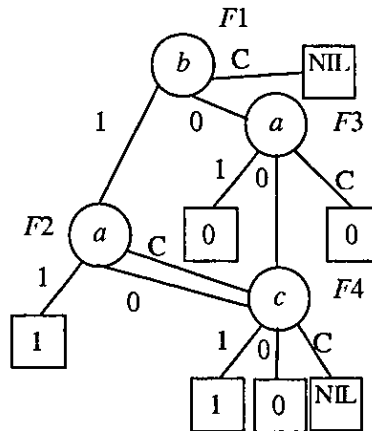


Figure 6.4: TDD for fault tree shown in Figure 5.5

6.3.2.2 Minimising the TDD

Once the TDD has been computed there is no guarantee that the resulting structure will be minimal, i.e. produce the prime implicant sets exactly. For quantitative analysis the initially generated TDD needs to be retained, but in order to perform the qualitative analysis a minimisation procedure needs to be implemented.

The algorithm developed by Rauzy for minimising the BDD [2] can be used to create a minimal TDD that encodes the prime implicant sets exactly.

Consider a general node in the TDD which is represented by the function F , where

$$F = \text{ifre}(x, G, H, K). \quad (6.36)$$

The process of minimisation is described in three cases:

- Component x is failure and repair relevant,
- Component x is failure relevant,
- Component x is repair relevant.

In case 1, the set of all minimal solutions of F will include minimal solutions of G (G_{min}) and H (H_{min}) that are not minimal solutions of K and also all minimal solutions of K (K_{min}). If δ is a minimal solution of G , which is not a minimal solution of K , then the intersection of δ and x ($\delta \cap x$) will be a minimal solution of F . Similarly, let γ be a minimal solution of H which is not a minimal solution of K , then the intersection of γ and \bar{x} ($\gamma \cap \bar{x}$) will be a minimal solution of F . The set of all the minimal solutions of F ($sol_{min}(F)$) will also include the minimal solutions of K , so:

$$sol_{min}(F) = (\delta \cap x) \cup (\gamma \cap \bar{x}) \cup K_{min}. \quad (6.37)$$

The 'without' operator removes all the paths from G_{min} and H_{min} that are included in K_{min} . In this way the combined set $sol_{min}(F)$ represents the minimal solutions of F by removing any minimal solutions of G and H that are also minimal solutions of K .

In case 2, where x is failure relevant, $K = NIL$ and the calculation of prime implicant sets is equivalent to the BDD case where the 'C' branch does not exist, i.e.

$$sol_{min}(F) = (\delta \cap x) \cup H_{min}. \quad (6.38)$$

The set $sol_{min}(F)$ represents the minimal solutions of F by removing any minimal solutions of G that are also minimal solutions of H .

In case 3, where x is repair relevant, $K = NIL$ and the calculation of prime implicant sets is defined as.

$$sol_{min}(F) = (\gamma \cap \bar{x}) \cup G_{min}. \quad (6.39)$$

Similarly to the previous cases, the set $sol_{min}(F)$ represents the minimal solutions of F by removing any minimal solutions of H that are also minimal solutions of G .

To illustrate this procedure consider the TDD in Figure 6.4. The nodes are considered in the top-down manner, starting with the root-node. The minimal solutions are computed for the 1 branch first, then for the 0 branch and finally for the consensus branch. Then all solutions that exist on either the 1 or 0 branch of a node that also exist on the consensus branch of the node are removed from the 1 or 0

branch by replacing the corresponding part with a terminal vertex 0. If a node is terminal it is minimal automatically, if it is non-terminal the structure below needs to be minimised before the minimisation of the current node can be completed.

Each node is considered in turn

$F1 = \text{ifre}(b, F2, F3, \text{NIL})$ - No minimisation is required at this stage

$F2 = \text{ifre}(a, 1, F4, F4) = \text{ifre}(a, 1, 0, F4)$ - Since the 1 branch is terminal, it does not contain any paths that are included in the consensus branch. Minimal solutions of the 0 branch and the consensus branch are the same, therefore, the minimal solutions of the 0 branch are removed by replacing the 0 branch with a terminal vertex 0.

$F3 = \text{ifre}(a, 0, F4, 0)$ - Since the consensus branch is terminal, it does not contain any paths that are included in the 1 and 0 branches.

$F4 = \text{ifre}(c, 1, 0, \text{NIL})$ - All the branches are terminal.

The minimal BDD is shown in Figure 6.5. Now it is possible to obtain a full list

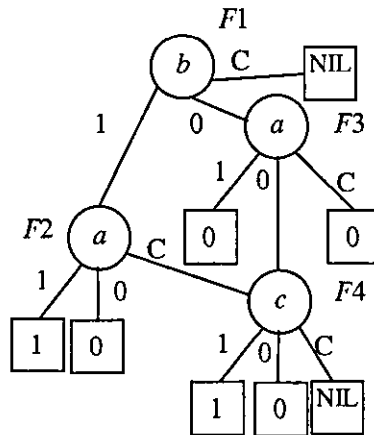


Figure 6.5: Minimised BDD obtained from the TDD in Figure 6.4

of prime implicant sets by tracing all the terminal paths through the minimised BDD:

$$\{a, b\}, \{\bar{a}, c\}, \{b, c\}. \tag{6.40}$$

This concludes the presentation of the TDD method, which has been published in [36]

6.3.3 Established methods

This section consists of the application of the three methods that were established in the literature. The first method is the meta-products BDD method [32]. After the SFBDD is constructed, every basic event in a meta-products BDD is represented by two variables, P_x and S_x . P_x represents the relevancy of the component (relevant or irrelevant) and S_x represents the type of the relevancy (failure relevant or repair relevant). The second approach is the zero-suppressed BDD method (ZBDD) [33]. In this method the system SFBDD is constructed and then used to build a ZBDD. All nodes in the ZBDD are labelled with failed and/or working states of basic events and prime implicant sets are decomposed according to the presence of a given state of a basic event. The ZBDD obtained is always in its minimal format. In the third alternative method, a labelled binary decision diagram (L-BDD) method [35], every basic event is labelled according to its type, i.e. failure or/and repair relevant. This additional information about the occurrence of every basic event is considered while converting a fault tree to an L-BDD. It does not provide all prime implicant sets, therefore the additional calculations are required followed by the minimisation technique. These three established methods will be considered for the efficiency test of the TDD method later in the section.

6.3.3.1 Rauzy and Dutuit Meta-products BDD

Rauzy and Dutuit developed an alternative notation that associates two variables with every component x . The first variable, P_x , denotes relevancy and the second variable, S_x , denotes the type of relevancy, i.e. failure or repair relevant. A meta-product, $MP(\pi)$, is the intersection of all the system components according to their relevancy to the system state and π represents the prime implicant set encoded in the meta-product $MP(\pi)$

$$MP(\pi) = \begin{cases} (P_x \wedge S_x) & \text{if } x \in \pi, \\ (P_x \wedge \bar{S}_x) & \text{if } \bar{x} \in \pi, \\ \bar{P}_x & \text{if neither } x \text{ nor } \bar{x} \text{ belongs to } \pi. \end{cases} \quad (6.41)$$

Rauzy proposed a method for calculating the Meta-products BDD of a fault tree from the SFBDD. The Meta-products BDD is always minimal, therefore it encodes the prime implicant sets exactly. A procedure called MPPI that converts the SFBDD into the Meta-products BDD is outlined below.

6.3.3.1.1 MPPI algorithm

A basic node in a SFBDD is described as

$$\text{ite}(x_i, f_1, f_0) \quad (6.42)$$

The meta-products structure for this node is denoted as:

$$PI[\text{ite}(x_i, f_1, f_0), L], \quad (6.43)$$

where L is the ordered list of all basic events except for those that appear on the current path from the root node to this node.

$PI[\text{ite}(x_i, f_1, f_0), L]$ is evaluated according to the following rules:

1. If x_i is the first basic event in L

$$PI[\text{ite}(x_i, f_1, f_0), L] = \text{ite}(P_{x_i}, \text{ite}(S_{x_i}, P1, P0), P2), \quad (6.44)$$

where

$$P2 = PI[f_1 \cdot f_0, L'], \quad (6.45)$$

$$P1 = PI[f_1, L'] \cdot \overline{P2}, \quad (6.46)$$

$$P0 = PI[f_0, L'] \cdot \overline{P2}, \quad (6.47)$$

and

$$L = x_i, x_{i+1}, \dots, x_n, \quad (6.48)$$

$$L' = x_{i+1}, x_{i+2}, \dots, x_n. \quad (6.49)$$

2. If x_i is not the first basic event in L , i.e. $L = x_j, x_{j+1}, \dots, x_n$ such that $i > j$.

$$PI[\text{ite}(x_i, f_1, f_0), L] = \text{ite}(P_{x_j}, 0, PI[\text{ite}(x_i, f_1, f_0), L']). \quad (6.50)$$

The following identities are applied:

$$PI[0, L] = 0, \quad (6.51)$$

$$PI[1, x \cdot L] = \text{ite}(P_x, 0, PI[1, L]) \quad (6.52)$$

For every vertex $\text{ite}(x, f_1, f_0)$ $P2$ encodes the prime implicant sets for which x is irrelevant, $P1$ encodes the prime implicant sets for which x is failure relevant and $P0$ encodes the prime implicant sets for which x is repair relevant.

In order to calculate $P2$ the basic **ite** structure of $f_2 = f_1 \cdot f_0$ must be calculated. Then the meta-products structure of f_2 , presented as $PI[f_2, L]$, must be computed. If f_2 is a terminal node

- 1 If $f_2 = 0$, $PI[f_2, L] = 0$,
- 2 If $f_2 = 1$, $PI[f_2, L] = \text{ite}(P_{x_i}, 0, \text{ite}(P_{x_{i+1}}, 0, \dots, \text{ite}(P_{x_n}, 0, 1)))$.

If f_2 is not terminal, **MPPI** calls itself to compute the meta-products structure of f_2 , before continuing the calculation at the previous level.

The same procedure is implemented for calculating the meta-products structure of f_1 and f_0 , denoted by $PI[f_1, L]$ and $PI[f_0, L]$ respectively. To ensure that the Meta-products BDD is minimal the conjunction of the meta-products structure for f_1 and f_0 and $\overline{P2}$ is performed. This eliminates repeated minimal solutions.

To illustrate how this algorithm is applied in practice consider the SFBDD in Figure 6.1, which has the following **ite** structure.

$$\text{ite}(b, \text{ite}(a, 1, \text{ite}(c, 1, 0)), \text{ite}(a, 0, \text{ite}(c, 1, 0))). \quad (6.53)$$

The meta-products structure must be computed for this **ite** structure

$$PI[\text{ite}(b, \text{ite}(a, 1, \text{ite}(c, 1, 0)), \text{ite}(a, 0, \text{ite}(c, 1, 0))), bac] = \quad (6.54)$$

$$\text{ite}(P_b, \text{ite}(S_b, P1, P0), P2),$$

where

$$P2 = PI[\text{ite}(a, 1, \text{ite}(c, 1, 0)) \cdot \text{ite}(a, 0, \text{ite}(c, 1, 0)), ac], \quad (6.55)$$

$$P1 = PI[\text{ite}(a, 1, \text{ite}(c, 1, 0)), ac] \cdot \overline{P2}, \quad (6.56)$$

$$P0 = PI[\text{ite}(a, 0, \text{ite}(c, 1, 0)), ac] \cdot \overline{P2}. \quad (6.57)$$

Calculating $P2$.

$$P2 = PI[\text{ite}(a, 0, \text{ite}(c, 1, 0)), ac] \quad (6.58)$$

$$= \text{ite}(P_a, \text{ite}(S_a, P1, P0), P2)$$

$$= \text{ite}(P_a, \text{ite}(S_a, 0, \text{ite}(P_c, \text{ite}(S_c, 1, 0), 0)), 0),$$

where

$$P2\ 1 = PI[0 \cdot \text{ite}(c, 1, 0), c] \quad (6\ 59)$$

$$= PI[0, c]$$

$$= 0,$$

$$P1\ 1 = PI[0, c] \cdot \overline{P2.1} \quad (6.60)$$

$$= 0 \cdot 1$$

$$= 0,$$

$$P0\ 1 = PI[\text{ite}(c, 1, 0), c] \cdot \overline{P2.1} \quad (6\ 61)$$

$$= \text{ite}(P_c, \text{ite}(S_c, 1, 0), 0) \cdot 1$$

$$= \text{ite}(P_c, \text{ite}(S_c, 1, 0), 0)$$

Calculating P1.

$$P1 = PI[\text{ite}(a, 1, \text{ite}(c, 1, 0)), ac] \cdot \overline{P2} \quad (6\ 62)$$

$$= \text{ite}(P_a, \text{ite}(S_a, P1.2, P0\ 2), P2.2) \cdot \overline{P2}$$

$$= \text{ite}(P_a, \text{ite}(S_a, \text{ite}(P_c, 0, 1), 0), \text{ite}(P_c, \text{ite}(S_c, 1, 0), 0)) \cdot$$

$$\text{ite}(P_a, \text{ite}(S_a, 1, \text{ite}(P_c, \text{ite}(S_c, 0, 1), 1)), 1)$$

$$= \text{ite}(P_a, \text{ite}(S_a, \text{ite}(P_c, 0, 1), 0), \text{ite}(P_c, \text{ite}(S_c, 1, 0), 0)),$$

where

$$P2\ 2 = PI[1 \cdot \text{ite}(c, 1, 0), c] \quad (6\ 63)$$

$$= PI[\text{ite}(c, 1, 0), c]$$

$$= \text{ite}(P_c, \text{ite}(S_c, 1, 0), 0),$$

$$P1\ 2 = PI[1, c] \cdot \overline{P2.2} \quad (6\ 64)$$

$$= \text{ite}(P_c, 0, 1) \cdot \text{ite}(P_c, \text{ite}(S_c, 0, 1), 1)$$

$$= \text{ite}(P_c, 0, 1),$$

$$P0\ 2 = PI[\text{ite}(c, 1, 0), c] \cdot \overline{P2.2} \quad (6\ 65)$$

$$= \text{ite}(P_c, \text{ite}(S_c, 1, 0), 0) \cdot \text{ite}(P_c, \text{ite}(S_c, 0, 1), 1)$$

$$= \text{ite}(P_c, \text{ite}(S_c, 0, 0), 0)$$

$$= 0$$

Calculating P0

$$\begin{aligned}
 P0 &= PI[\text{ite}(a, 0, \text{ite}(c, 1, 0)), ac] \cdot \overline{P2} & (6.66) \\
 &= \text{ite}(P_b, 0, PI[\text{ite}(c, 1, 0), c]) \cdot \overline{P2} \\
 &= \text{ite}(P_a, \text{ite}(S_a, 0, \text{ite}(P_c, \text{ite}(S_c, 1, 0), 0)), 0) \cdot \\
 &\quad \text{ite}(P_a, \text{ite}(S_a, 1, \text{ite}(P_c, \text{ite}(S_c, 0, 1), 1)), 1) \\
 &= \text{ite}(P_a, \text{ite}(S_a, 0, \text{ite}(P_c, \text{ite}(S_c, 0, 0), 0)), 0) \\
 &= 0
 \end{aligned}$$

The *ite* structure of Meta-products BDD for the SFBDD in Figure 6.1 is given below:

$$\begin{aligned}
 &\text{ite}(P_b, \text{ite}(S_b, \text{ite}(P_a, \text{ite}(S_a, \text{ite}(P_c, 0, 1), 0), f), 0), & (6.67) \\
 &\text{ite}(P_a, \text{ite}(S_a, 0, f), 0)),
 \end{aligned}$$

where

$$f = \text{ite}(P_c, \text{ite}(S_c, 1, 0), 0). \quad (6.68)$$

The Meta-products BDD is given in Figure 6.6. Now it is possible to obtain the

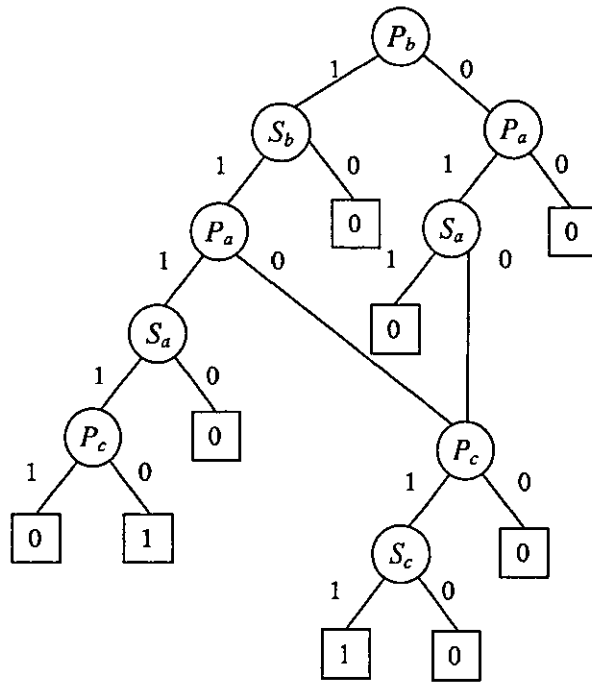


Figure 6.6: Meta-products BDD calculated from the BDD given in Figure 6.1

meta-products and identify the prime implicant sets:

$$P_b \wedge S_b \wedge P_a \wedge S_a \wedge \overline{P_c} = \{a, b\} \quad (6.69)$$

$$P_b \wedge S_b \wedge \overline{P_a} \wedge P_c \wedge S_c = \{b, c\} \quad (6.70)$$

$$\overline{P_b} \wedge P_a \wedge \overline{S_a} \wedge P_c \wedge S_c = \{\overline{a}, c\}. \quad (6.71)$$

For example, in the first meta-product means P_b signifies that component b is relevant and S_b signifies that component b is failure relevant. Component a is also failure relevant. Finally, $\overline{P_c}$ means that component c is irrelevant. Hence the additional prime implicant set $\{a, b\}$ is obtained.

Since in the Meta-products BDD every basic event is represented by two elements, P_x and S_x , the size of the BDD can expand quite a lot

6.3.3.2 Zero-suppressed BDD Method

As it was discussed earlier, working with non-coherent fault trees a SFBDD is not a sufficient means to identify prime implicant sets. Another alternative method presented by Rauzy [33] uses BDDs but with a different meaning. The idea of Zero-suppressed BDDs (ZBDD) introduced by Minato [34] is used to calculate prime implicant sets. This method requires nodes to be labelled with failed and/or working states of basic events and to decompose prime implicant sets according to the presence of a given state of a basic event.

6.3.3.2.1 Presentation of a ZBDD

Zero-suppressed BDDs are BDDs obtained after applying a reduction rule. This data structure brings a unique and compact representation of sets and it is more efficient and simpler than the usual BDDs when manipulating sets in combinatorial problems. The following reduction rules for BDDs are:

- Eliminate all the nodes that have the 1 branch pointing to terminal vertex 0. Then connect the branch that was pointing to the eliminated node to the BDD structure beneath the 0 branch of the eliminated node, shown in Figure 6.7
- Share all equivalent BDD structures as for original BDDs.

ZBDDs automatically suppress basic events that do not appear in prime implicant sets. It is very efficient when calculating sets with basic events that are far apart

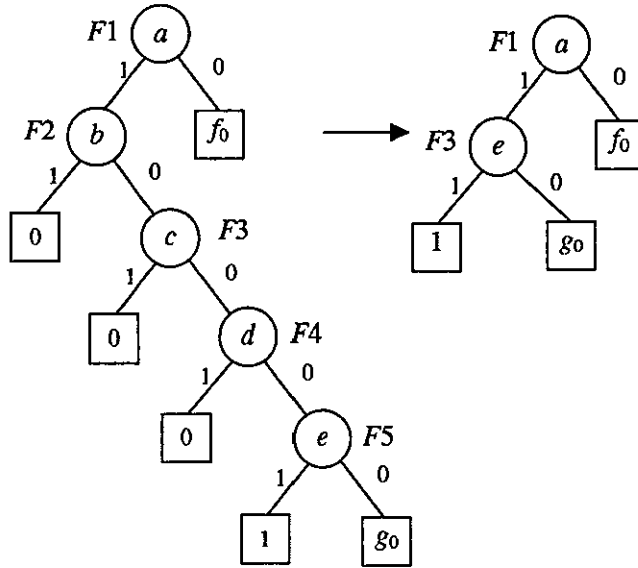


Figure 6.7: Elimination process

in the variable ordering scheme. For example, in Figure 6.7 the represented BDD contains a prime implicant set $\{a, e\}$. The established variable ordering is $a < b < c < d < e$. The obtained ZBDD brings basic events close since the intermediate nodes, $F2$, $F3$ and $F4$, can be eliminated.

6.3.3.2.2 Decomposition rule

The principle of the ZBDD algorithm is to traverse the SFBDD that encodes structure function $\phi = \text{ite}(x, f_1, f_0)$ in a depth-first way and to build a ZBDD that encodes the prime implicant sets in a bottom-up way. The rule is described in four cases

1. Basic event x appears in both, failed and working, states:

$$PI(\phi) = x \cdot S_1 + \bar{x} \cdot S_0 + S_2, \quad (6.72)$$

$$\text{where } S_2 = PI(f_1 \cdot f_0), \quad (6.73)$$

$$S_1 = PI(f_1) \setminus S_2, \quad (6.74)$$

$$S_0 = PI(f_0) \setminus S_2 \quad (6.75)$$

2. Basic event x appears in its failed state only.

$$PI(\phi) = x \cdot S_1 + S_0, \quad (6.76)$$

$$\text{where } S_0 = PI(f_0), \quad (6.77)$$

$$S_1 = PI(f_1) \setminus S_0 \quad (6.78)$$

3. Basic event x appears in its working state only This case is similar to case 2.
4. Basic event x does not appear in the system:

$$PI(\phi) = PI(f_1 + f_0). \quad (6.79)$$

Here \setminus is the 'without' operator used in the coherent case

Produced ZBDDs retain the variable ordering from the SFBDD case. In addition, working states of basic events that appear in both states are incorporated in the ordering scheme, i.e. they appear after the basic event that describes the failed state in the ordering scheme. For example, the variable ordering constructing a SFBDD for the example in Figure 5.5 is $b < a < \bar{c}$, in the ZBDD method the introduced ordering is $b < a < \bar{a} < c$.

6.3.3.2.3 Worked example

Consider the example fault tree in Figure 5.5. Using the variable ordering $b < a < c$ the SFBDD obtained is shown in Figure 6.1. Each node is considered in the bottom-up way.

$$F4 = \text{ite}(c, 1, 0), PI(F4) = \text{ite}(c, 1, 0), \quad (6.80)$$

because both vertices are terminal,

$$F2 = \text{ite}(a, 1, F4), PI(F2) = \text{ite}(a, S_1, \text{ite}(\bar{a}, S_0, S_2)), \quad (6.81)$$

because a appears in both states.

$$S_2 = PI(1 \cdot F4) = PI(F4), \quad (6.82)$$

$$S_1 = 1, \quad (6.83)$$

$$S_0 = PI(F4) \setminus PI(F4) = 0. \quad (6.84)$$

Therefore

$$PI(F2) = \text{ite}(a, 1, \text{ite}(c, 1, 0)). \quad (6.85)$$

Then

$$F3 = \text{ite}(a, 0, F4), PI(F3) = \text{ite}(a, S_1, \text{ite}(\bar{a}, S_0, S_2)), \quad (6.86)$$

$$S_2 = 0, \quad (6.87)$$

$$S_1 = 0, \quad (6.88)$$

$$S_0 = PI(F4). \quad (6.89)$$

Therefore

$$PI(F3) = \text{ite}(\bar{a}, \text{ite}(c, 1, 0), 0). \quad (6.90)$$

Finally

$$F1 = \text{ite}(b, F2, F3), PI(F1) = \text{ite}(b, S_1, S_0), \quad (6.91)$$

$$S_0 = PI(F3), \quad (6.92)$$

$$S_1 = PI(F2) \setminus PI(F3) = PI(F2). \quad (6.93)$$

Therefore:

$$PI(F1) = \text{ite}(b, \text{ite}(a, 1, \text{ite}(c, 1, 0)), \text{ite}(\bar{a}, \text{ite}(c, 1, 0), 0)). \quad (6.94)$$

The obtained ZBDD is shown in Figure 6.8. Every path in a ZBDD from the root

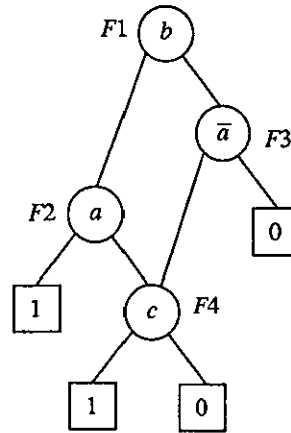


Figure 6.8: ZBDD for non-coherent fault tree

vertex to terminal vertex 1 presents a prime implicant set. Only vertices that lie on the 1 branch of a path are included in a prime implicant set. Therefore, this ZBDD contains three prime implicant sets shown in Table 6.1

Path	Prime implicant
$F1 - F2$	$\{b, a\}$
$F1 - F2 - F4$	$\{b, c\}$
$F1 - F3 - F4$	$\{\bar{a}, c\}$

Table 6.1: Prime implicant sets using the ZBDD

This method provides a compact representation of prime implicant sets and enables an efficient qualitative analysis to be performed

6.3.3.3 Labelled Variable Method

The labelled variable method is the third alternative method for constructing BDDs for non-coherent fault trees. BDDs obtained using this approach of conversion consist of variables that are labelled according to their type. The construction of a BDD with labelled variables (L-BDD) and the efficiency of the method is presented in this section.

6.3.3.3.1 Classification of variables

As it was recognised before, the structure function $\phi(x)$ of a non-coherent fault tree may contain three different types of basic events. In this method they are described as:

- Positive events that appear in the failed state only, referred to as single form positive variables (SFP),
- Negative events that appear in the working state only, referred to as single form negated variables (SFN),
- Events that appear both in positive and negative forms, i.e. that appear in the failed and working states, called double form variables (DF).

For example, the function $\phi(x) = a \cdot b + \bar{a} \cdot \bar{c} + b \cdot \bar{c}$ contains the DF variable a , the SFP variable b and the SFN variable c . Coherent fault trees contain only SFP variables.

In the presentation below the SFP variable x will be simply presented by x , the SFN variable x will be labelled '\$ x ' character and the DF variable x will be labelled '& x ' character.

Let $\phi(x)$ be a non-monotonic function and x the variable selected for expansion. The following three cases are possible.

- For SFP variables the expansion of $\phi(x)$ gives:

$$\phi(x) = x \cdot \phi(1, x) + \bar{x} \cdot \phi(0, x). \quad (6.95)$$

From the probabilistic viewpoint:

$$P(x) = q_x, P(\bar{x}) = 1 - q_x. \quad (6.96)$$

- For SFN variables the expansion of $\phi(x)$ is:

$$\phi(x) = \$x \cdot \phi(1, x) + \bar{\$}x \cdot \phi(0, x). \quad (6.97)$$

In terms of the probability:

$$P(\$x) = 1 - q_x, P(\bar{\$}x) = q_x. \quad (6.98)$$

The variable labelled with the \$ symbol is equivalent to variable \bar{x}

- If x is of DF type the expansion of $\phi(x)$ gives:

$$\phi(x) = \&x \cdot \phi(1, x) + \bar{\&}x \cdot \phi(0, x). \quad (6.99)$$

The probability can be calculated according to:

$$P(\&x) = q_x, P(\bar{\&}x) = 1 - q_x. \quad (6.100)$$

6.3.3.3.2 Construction of the L-BDD

To construct the L-BDD the first composition rule (Equation 3.2) from Chapter 3 needs to be extended, so that it would be possible to take into account the labelled variables. The second rule (Equation 3.3) can be applied as it was presented for the coherent case. Consider the ordering of the labelled variables $\&x < x < \$x$ the following additional rules are developed

Let J and H be two nodes in the BDD.

- Let $J = \text{ite}(x, F_1, F_2), H = \text{ite}(\$x, G_1, G_2)$.

Since $x < \$x$ then:

$$J < op > H = \text{ite}(\&x, F_1 < op > G_2, F_2 < op > G_1). \quad (6.101)$$

In this case:

$$\begin{aligned} J < op > H &= \text{ite}(\&x, (J < op > H) |_{\&x=1}, (J < op > H) |_{\&x=0}) = \\ &\text{ite}(\&x, (F |_{x=1} < op > G |_{\$x=0}), (F |_{x=0} < op > G |_{\$x=1})) \end{aligned}$$

- Let $J = \text{ite}(\&x, F_1, F_2), H = \text{ite}(x, G_1, G_2)$.

Since $\&x < x$ then:

$$J < op > H = \text{ite}(\&x, F_1 < op > G_1, F_2 < op > G_2). \quad (6.102)$$

Here

$$\begin{aligned} J < op > H &= \text{ite}(\&x, (J < op > H) |_{\&x=1}, (J < op > H) |_{\&x=0}) = \\ &\text{ite}(\&x, (F |_{\&x=1} < op > G |_{x=1}), (F |_{\&x=0} < op > G |_{x=0})). \end{aligned}$$

- Let $J = \text{ite}(\&x, F_1, F_2), H = \text{ite}(\$x, G_1, G_2)$.

Since $\&x < \$x$ then

$$J < op > H = \text{ite}(\&x, F_1 < op > G_2, F_2 < op > G_1). \quad (6.103)$$

In this case

$$\begin{aligned} J < op > H &= \text{ite}(\&x, (J < op > H) |_{\&x=1}, (J < op > H) |_{\&x=0}) = \\ &\text{ite}(\&x, (F |_{\&x=1} < op > G |_{\$x=0}), (F |_{\&x=0} < op > G |_{\$x=1})). \end{aligned}$$

here $< op >$ as before corresponds to the Boolean operations 'AND' or 'OR'.

The trivial relationships are considered as

$$0 \wedge \alpha = 0, \quad (6.104)$$

$$1 \wedge \alpha = \alpha, \quad (6.105)$$

$$0 \vee \alpha = \alpha, \quad (6.106)$$

$$1 \vee \alpha = 1, \quad (6.107)$$

where α is from the set $\{\&x, x, \$x\}$

This algorithm will be applied to the example fault tree in Figure 5.5. There are 3 variables in the fault tree. Two of them are the SFP variables, b, c , since they only appear in the failed state. One variable is the DP variable, a , since it appears in both working and failed states. The variable ordering $b < \&a < a < \$a < c$ is introduced. Then the composition rules are applied and **ite** structures for gates $G1$ and $G2$ are calculated

$$G1 = \text{ite}(b, 1, 0) \cdot \text{ite}(a, 1, 0) = \text{ite}(b, \text{ite}(a, 1, 0), 0) \quad (6.108)$$

$$G2 = \text{ite}(\$a, 1, 0) \cdot \text{ite}(c, 1, 0) = \text{ite}(\$a, \text{ite}(c, 1, 0), 0). \quad (6.109)$$

Finally, *Top* gate is considered

$$\text{Top} = \text{ite}(b, \text{ite}(a, 1, 0), 0) + \text{ite}(\$a, \text{ite}(c, 1, 0), 0) \quad (6.110)$$

$$= \text{ite}(b, \text{ite}(a, 1, 0) + \text{ite}(\$a, \text{ite}(c, 1, 0), 0), \text{ite}(\$a, \text{ite}(c, 1, 0), 0))$$

$$= \text{ite}(b, \text{ite}(\&a, 1, \text{ite}(c, 1, 0)), \text{ite}(\$a, \text{ite}(c, 1, 0), 0))$$

This connection considers the application of new rules in the case of $x < \$x$. The resulting BDD is shown in Figure 6.9. The sub-node sharing property is still applicable

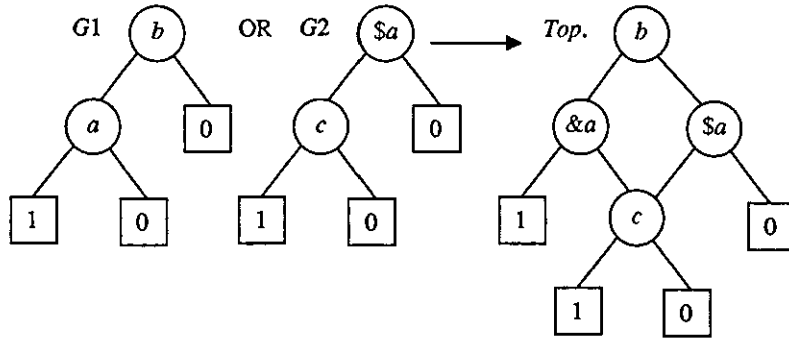


Figure 6.9. Resulting L-BDD for example tree in Figure 5.5

6.3.3.3.3 Determination of prime implicants

The labelled variables are considered for determining the set of prime implicants. For the SFP and the SFN variables the algorithm applied is presented in [2] as the method for the calculation of minimal cut sets in the coherent case. For the DF variables the algorithm proposed in [32] is used

Let the structure function be:

$$\phi(x) = x_i \cdot F + \bar{x}_i \cdot G, \quad (6.111)$$

where the two residues are

$$F = \phi(x_1, \dots, 1, \dots, x_n), \quad (6.112)$$

$$G = \phi(x_1, \dots, 0, \dots, x_n). \quad (6.113)$$

Let $PI(\phi)$ be the prime implicants of ϕ . Visiting the L-BDD in the bottom-up way the procedure to be applied to the node x_i to determine the prime implicants is as follows

- If x_i has label '&' then:

$$PI(\phi) = x_i \cdot F^* + \$x_i \cdot G^* + P, \quad (6.114)$$

$$\text{where } P = F \wedge G, F^* = F \setminus P, G^* = G \setminus P, \quad (6.115)$$

- else

$$PI(\phi) = \alpha \cdot F^* + G, \quad (6.116)$$

$$\text{where } \alpha = (x_i \text{ or } \$x_i), F^* = F \setminus G \quad (6.117)$$

'\ ' is the 'without' operator proposed by Rauzy [2]

In the case of the L-BDD for the application of the '\ ' operator it is necessary to take into account the type of label. The minimisation of the L-BDD is based on the algorithm presented in the coherent case. All prime implicant sets need to be obtained for '&' variables, by producing the conjunction of the '1' and '0' branches of every node. Therefore, there are some additional rules for minimisation considering situations with labelled variables, because the conjunction of the two branches can have some sets that are non-minimal.

The most time consuming operation is the intersection $F \wedge G$ that, however, is applied only when dealing with '&' type variables. If there are no '&' type variables, the L-BDD contains all prime implicant sets.

Once the minimisation is completed the rules for writing the prime implicant sets from an L-BDD are straightforward. On the path from the root to any terminal node:

- For variables x , $\$x$ do not consider the negated part,
- For variables $\&x$ consider the right branch as \bar{x} .

Since the heaviest operations in this algorithm for determining the prime implicants set are applied to DF variables, it is convenient to reduce their number. The simplification rules are presented in the next section.

6.3.3.3.4 Simplification of L-BDD

The simplification is possible if one of the residues is terminal, i.e. F or G is equal to 1 or 0. The simplification rules are presented below providing the expression for ϕ presented in Equation 6.111.

- If $F = 1$ then $P = F \wedge G = G, F^* = 1, G^* = 0$.
Therefore, $\phi = x + G$
- If $F = 0$ then $P = F \wedge G = 0, F^* = 0, G^* = G$.
Therefore, $\phi = \bar{x} + G$
- If $G = 1$ then $P = F \wedge G = F, F^* = 0, G^* = 1$.
Therefore, $\phi = \bar{x} + F$

- If $G = 0$ then $P = F \wedge G = 0, F^* = F, G^* = 0$.
Therefore, $\phi = x \ F$.

All four simplification rules are shown in Figure 6.10. In the first and the fourth

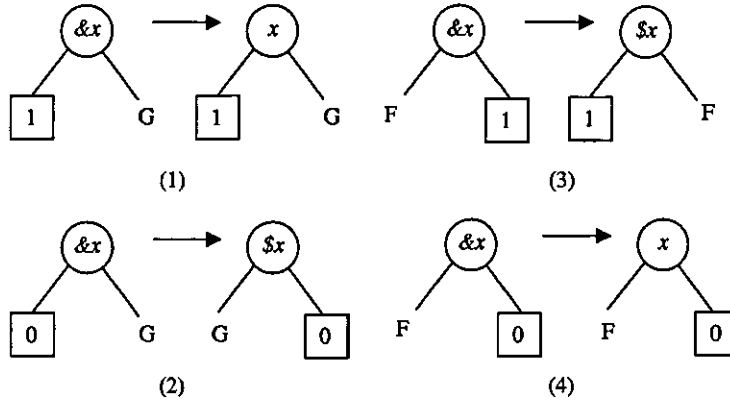


Figure 6.10 Four simplification rules

cases variable $\&x$ behaves as a positive variable x , in the second and the third cases variable $\&x$ behaves as a negative variable $\$x$.

Now the example shown in Figure 6.9 is considered. Before the calculation of prime implicant sets the simplification process can be implemented, trying to minimise the number of DP variables. The simplified example L-BDD is shown in Figure 6.11. The first simplification rule was applied in this example to the node

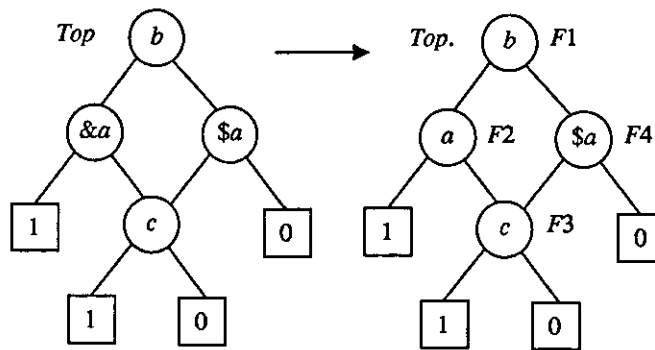


Figure 6.11 Simplified example L-BDD

variable $\&a$. Therefore, $\&a$ was simply replaced with a .

The prime implicant sets can be obtained using Equations 6.114 - 6.117 Traversal

of the L-BDD starts in the bottom-up way

$$PI(F3) = c \cdot F^* + G, \text{ here } F^* = 1, G = 0, \quad (6.118)$$

$$PI(F3) = c,$$

$$PI(F2) = a \cdot F^* + G, \text{ here } F^* = 1, G = F3 = PI(F3) = c, \quad (6.119)$$

$$PI(F2) = a + c,$$

$$PI(F4) = \$a \cdot F^* + G, \text{ here } F^* = F3 = PI(F3) = c, G = 0, \quad (6.120)$$

$$PI(F4) = \$a \cdot c,$$

$$PI(F1) = b \cdot F^* + G, \text{ here } F^* = F2 = PI(F2) = a + c, \quad (6.121)$$

$$G = F4 = PI(F4) = \$a \cdot c,$$

$$PI(F1) = b \cdot a + b \cdot c + \$a \cdot c$$

Therefore, for the above example the prime implicant sets are

$$\{b, a\}, \{b, c\}, \{\bar{a}, c\} \quad (6.122)$$

The calculation of prime implicant sets is shown in Figure 6.12 Variable $\$a$ is

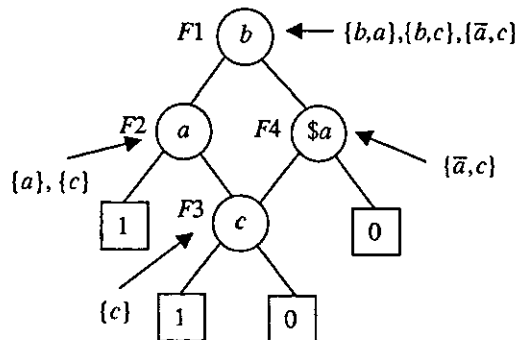


Figure 6.12: Calculation of prime implicant sets for example L-BDD

replaced by \bar{a} , since they are equivalent variables

This concludes the description of the fourth method for the non-coherent fault tree analysis. It provides a means for obtaining prime implicant sets. However, the introduction of labels increases the number of variables and the complexity of the process can affect the efficiency of the analysis.

6.3.4 Comparison of the Four Methods

Research has been carried out testing the proposed TDD method and comparing its efficiency with the other techniques in order to identify their strengths and

weaknesses. A specific approach will work well on some fault trees and not on others. It is the performance of an approach over the range of problems that it may encounter that should be established. All algorithms are applied to a library of 220 fault trees. The example fault trees from the coherent case were converted to non-coherent examples by randomly introducing some negated events and gates. Characteristics of test fault trees are shown in Tables A.62 - A.66. The structure of the tables is equivalent to the coherent case, except that the last column identifies the number of prime implicant sets instead of number of minimal cut sets.

Example fault trees are categorised as small fault trees (207 FTs) and 'large' fault trees (13 FTs). 'Large' fault trees are classified as those with a large number of prime implicant sets. Also, high complexity fault trees, i.e. with a large number of gates and non-repeated basic events, are ranked as 'large' fault trees. These two properties of fault trees result in a time consuming analysis process or even can make calculations impossible in reasonable times. It is these fault trees which present the largest degree of difficulty in their analysis. As such it is these fault trees which will really test the competence of any algorithm.

During the analysis every fault tree is converted to a SFBDD that represents the structure function of the fault tree. The SFBDD is not suitable for the full qualitative analysis, therefore other alternative methods are required. Four different techniques are investigated using the library of fault trees. In the first method, the Ternary Decision Diagram method, that has been developed as a part of this research, the TDD is calculated straight from the fault tree. Therefore, no additional BDD is required that would encode prime implicant sets. This could be an advantage over the other techniques. The 'missing' prime implicant sets that can not be found from the SFBDD are covered through the conjunction of the two branches of components that can be failure and repair relevant. However, it needs to be minimised before the calculation of prime implicant sets and this can increase the processing time.

The three established methods are also considered. During the first of them, i.e. the Meta-products BDD method, every SFBDD is converted to a Meta-products BDD that encodes all prime implicants and is minimal. Every basic event is presented by two variables that can increase the size of the structure. The second technique, the ZBDD method, converts a fault tree to a SFBDD and then an

additional Zero-suppressed BDD is obtained that encodes prime implicant sets. Despite the fact that an additional BDD needs to be created, it is minimal, zero-suppressed and can be an efficient technique for the qualitative analysis. The last method, the L-BDD method, converts a fault tree to a labelled BDD that allows the qualitative analysis. The conjunction of the two branches is similar to the TDD method. However, extra variables are introduced before the conversion process and the minimisation needs to be applied. Those two issues can increase the number of nodes in the structure and the processing time required.

The efficiency measures are calculated and analysed. The number of nodes and the processing time are the most important characteristics, therefore, according to their values a comparison between the four techniques is made.

The 8 basic event ordering schemes are ranked in order according to the effectiveness of the conversion process that they produce. The performance of the schemes is then assessed in three ways, as it was used in the coherent case - according to their totals, according to their highest ranking and according to their total ranking. The ordering schemes are ranked for all the efficiency measures according to their performance on small fault trees, 'large' fault trees and the whole set of fault trees. This is done in order to show if the performance of the ordering schemes depends on the complexity of the system and to give an indication of which ordering schemes should be used for small and 'large' fault trees.

6.3.4.1 TDD method results

Summary details of test fault trees are presented in Tables A.67 - A.70, which show the number of nodes in TDDs for small fault trees. Table A.71 shows the number of nodes for the large fault trees. The time taken to convert fault trees to TDDs and perform the full qualitative analysis is shown in Tables A.72 - A.75 and Table A.76 for small and large fault trees respectively.

Some fault trees were simplified so much that their TDDs contained only 1 node. There were 45 fault trees of that type. Therefore, the ranking was performed without taking into account those examples, together with the 26 other fault trees that had the same number of nodes in the TDD for all eight ordering schemes. Summarising, there were 149 example fault trees taken for the ranking according to the number of nodes, where 13 of them were 'large' examples.

Considering the processing time, there were 8 fault trees whose processing time was the same in each ordering scheme, therefore, those fault trees were not taken into account while ranking the ordering schemes according to the processing time. 199 small fault trees and 13 large fault trees were considered in the ranking analysis according to the processing time.

6.3.4.1.1 Variable ordering for the TDD method

Analysis of the number of nodes in TDDs and processing time for 'small', 'large' and all example fault trees is presented in this section, applying the three ranking techniques of the ordering schemes. The result of the analysis is shown in Table 6 2 and Table 6 3 for the number of nodes and the processing time respectively.

		Scheme	1	2	3	4	5	6	7	8
Small fault trees	Total quantity ranking	Number of nodes	24167	23050	21729	20434	19976	18138	23339	18515
		Rank	8	6	5	4	3	1	7	2
	Highest scheme ranking	Number of FTs with the highest rank	20	40	30	34	20	40	34	35
		Rank	7-8	1-2	6	4-5	7-8	1-2	4-5	3
	Added scheme ranking	Added ranking	599	472	462	449	492	387	509	480
		Rank	8	4	3	2	6	1	7	5
Large fault trees	Total quantity ranking	Number of nodes	21553	23436	22394	14861	20115	14159	17957	22782
		Rank	5	8	6	2	4	1	3	7
	Highest scheme ranking	Number of FTs with the highest rank	0	0	0	3	2	2	4	2
		Rank	6-8	6-8	6-8	2	3-5	3-5	1	3-5
	Added scheme ranking	Added ranking	74	68	69	44	58	40	49	56
		Rank	8	6	7	2	5	1	3	4
All fault trees	Total quantity ranking	Number of nodes	45720	46486	44123	35295	40091	32297	41296	41297
		Rank	7	8	6	2	3	1	4	5
	Highest scheme ranking	Number of FTs with the highest rank	20	40	30	37	22	42	38	37
		Rank	8	2	6	4-5	7	1	3	4-5
	Added scheme ranking	Added ranking	673	540	531	493	550	427	558	536
		Rank	8	5	3	2	6	1	7	4

Table 6 2 TDD method, number of nodes

There are some clear ranking results for the TDD method. The dynamic top-down weighted scheme (6) performed well for the number of nodes and the processing time. The modified top-down scheme (1) gave the worst results for the number of nodes for small fault trees and 'large' fault trees considering any of the three ranking methods. For the processing time scheme (1) gave average results, and the

		Scheme	1	2	3	4	5	6	7	8
Small fault trees	Total quantity ranking	Time	22 51	22 54	22 52	22 33	21 96	21 97	22 84	22 45
		Rank	5	7	6	3	1	2	8	4
	Highest scheme ranking	Number of FTs with the highest rank	71	79	71	79	78	83	64	71
		Rank	5-7	2-3	5-7	2-3	4	1	8	5-7
	Added scheme ranking	Added ranking	552	551	543	548	499	512	583	568
		Rank	6	5	3	4	1	2	8	7
Large fault trees	Total quantity ranking	Time	8 82	13 98	12 47	7 06	10 44	6 73	8 56	13 42
		Rank	4	8	6	2	5	1	3	7
	Highest scheme ranking	Number of FTs with the highest rank	1	1	0	6	3	1	3	4
		Rank	5-7	5-7	8	1	3-4	5-7	3-4	2
	Added scheme ranking	Added ranking	55	61	65	33	46	43	51	41
		Rank	6	7	8	1	4	3	5	2
All fault trees	Total quantity ranking	Time	31 33	36 52	34 99	29 39	32 4	28 7	31 4	35 87
		Rank	3	8	6	2	5	1	4	7
	Highest scheme ranking	Number of FTs with the highest rank	72	80	71	85	81	84	67	75
		Rank	6	4	7	1	3	2	8	5
	Added scheme ranking	Added ranking	607	612	608	581	545	555	634	609
		Rank	4	7	5	3	1	2	8	6

Table 6 3: TDD method, processing time

two modified depth-first ordering schemes (2) and (3) gave poor results Overall, the difference in performance of the eight ordering schemes is marginal, especially, for the processing time, therefore, no further conclusions can be drawn.

6.3.4.2 Meta-products method results

Summary details of the test fault trees are presented in Tables A 77 - A 80, which show the sum of nodes in the obtained SFBDDs and in Meta-products BDDs for small fault trees Table A 81 shows the sum of nodes for the large fault trees The time taken to convert fault trees to SFBDDs, obtain Meta-products BDDs and perform the full qualitative analysis is shown in Tables A 82 - A 85 and Table A 86 for small and large fault trees respectively.

Some fault trees were simplified so much that their SFBDDs contained only 1 node. There were 45 fault trees of that type Also, for 26 fault trees the eight ordering schemes gave the same number of nodes, therefore, the ranking analysis was performed without taking into account those examples. There were some 'large' example fault trees for which the conversion process and analysis could not be finished in a reasonable time, i.e. it took longer than 24 hours, and therefore the

calculation process was terminated. There were only 6 'large' fault trees (out of 13) where the analysis was performed for all the eight ordering schemes. In total, there were 142 fault trees taken for the ranking analysis according to the number of nodes, where 6 of them were 'large' examples.

For the processing time 186 small fault trees and 6 'large' fault trees were analysed, since for the rest of the fault trees either the eight ordering schemes gave the same processing time or the calculations were not finished.

6.3.4.2.1 Variable ordering for the MPPI method

Analysis of the number of nodes in meta-product BDDs and processing time for 'small', 'large' and all example fault trees is presented in this section, applying the three ranking techniques of the ordering schemes. The result of the analysis is shown in Table 6.4 and Table 6.5 for the number of nodes and the processing time respectively

		Scheme	1	2	3	4	5	6	7	8
Small fault trees	Total quantity ranking	Number of nodes	59491	56186	54162	51510	51778	48200	51931	49115
		Rank	8	7	6	3	4	1	5	2
	Highest scheme ranking	Number of FTs with the highest rank	15	33	27	33	16	37	51	25
		Rank	8	3-4	5	3-4	7	2	1	6
	Added scheme ranking	Added ranking	644	486	458	451	564	430	431	522
		Rank	8	5	4	3	7	1	2	6
Large fault trees	Total quantity ranking	Number of nodes	1779183	1689973	1774833	533464	1246614	508603	1493336	621596
		Rank	8	6	7	2	4	1	5	3
	Highest scheme ranking	Number of FTs with the highest rank	0	0	0	2	0	3	1	0
		Rank	4-8	4-8	4-8	2	4-8	1	3	4-8
	Added scheme ranking	Added ranking	22	16	20	6	16	8	12	8
		Rank	8	5-6	7	1	5-6	2-3	4	2-3
All fault trees	Total quantity ranking	Number of nodes	1838674	1746159	1828995	584974	1298392	556803	1545267	670711
		Rank	8	6	7	3	4	1	5	2
	Highest scheme ranking	Number of FTs with the highest rank	15	33	27	35	16	40	52	25
		Rank	8	4	5	3	7	2	1	6
	Added scheme ranking	Added ranking	666	502	478	457	580	438	443	530
		Rank	8	5	4	3	7	1	2	6

Table 6.4 MPPI method, number of nodes

The dynamic top-down weighted scheme (6) performed best and was ranked high for both the number of nodes and the processing time. It also resulted in the

		Scheme	1	2	3	4	5	6	7	8
Small fault trees	Total quantity ranking	Time	16072 92	9125 99	5941 00	8113 49	3961 66	2070 27	5511 35	1442 22
		Rank	8	7	5	6	3	2	4	1
	Highest scheme ranking	Number of FTs with the highest rank	55	64	45	50	48	70	63	67
		Rank	5	3	8	6	7	1	4	2
	Added scheme ranking	Added ranking	645	616	665	609	608	499	598	497
		Rank	7	6	8	5	4	2	3	1
Large fault trees	Total quantity ranking	Time	65906 72	83737 84	117648 78	13181 58	51992 24	12201 20	60442 25	11027 12
		Rank	6	7	8	3	4	1	5	2
	Highest scheme ranking	Number of FTs with the highest rank	0	0	0	2	0	2	1	1
		Rank	5-8	5-8	5-8	1-2	5-8	1-2	3-4	3-4
	Added scheme ranking	Added ranking	21	14	15	11	20	9	10	8
		Rank	8	5	6	4	7	2	3	1
All fault trees	Total quantity ranking	Time	81979 64	92863 83	123589 78	21295 07	55953 90	14271 47	65953 60	12469 34
		Rank	6	7	8	3	4	2	5	1
	Highest scheme ranking	Number of FTs with the highest rank	55	64	45	52	48	72	64	68
		Rank	5	3-4	8	6	7	1	3-4	2
	Added scheme ranking	Added ranking	666	630	680	620	628	508	608	505
		Rank	7	6	8	4	5	2	3	1

Table 6 5: MPPI method, processing time

smallest number of unfinished processes, i.e. only for 1 fault tree the calculation process could not be finished in a reasonable time, whereas for some ordering schemes, (1) and (3), there were 4 fault trees whose conversion was impossible. The depth-first with number of leaves ordering (4) performed well for the number of nodes and the event criticality scheme (8) gave good results for the processing time. The worst results were obtained by the modified top-down scheme (1) which was ranked last for both the efficiency measurements in almost all the ranking methods. The modified priority depth-first ordering scheme (3) also gave poor results for the processing time.

6.3.4.3 ZBDD method results

Summary details of the test fault trees are presented in Tables A 87 - A 90, which show the sum of nodes in the SFBDDs and obtained ZBDDs for small fault trees. Table A 91 shows the sum of the nodes for the large fault trees. The time taken to convert fault trees to SFBDDs, then obtain the ZBDDs and perform the full qualitative analysis is shown in Tables A.92 - A.95 and Table A 96 for small and large fault trees respectively.

The ranking analysis was performed without taking into account examples that were simplified to one complex event and the 37 other fault trees that had the same number of nodes in the ZBDD for all eight ordering schemes. Overall, for the ranking according to the number of nodes there were 138 example fault trees taken for the analysis, where 13 of them were 'large' examples.

For the processing time there were 205 small fault trees and 13 large fault trees taken into account, because there were 2 small fault trees where the processing time was the same for the eight ordering schemes

6.3.4.3.1 Variable ordering for the ZBDD method

Analysis of the number of nodes in ZBDDs and processing time for 'small', 'large' and all example fault trees is presented in this section, applying the three ranking techniques of the ordering schemes. The result of the analysis is shown in Table 6.6 and Table 6.7 for the number of nodes and the processing time respectively

		Scheme	1	2	3	4	5	6	7	8
Small fault trees	Total quantity ranking	Number of nodes	19332	15939	15920	15758	17169	15079	15808	16537
		Rank	8	5	4	2	7	1	3	6
	Highest scheme ranking	Number of FTs with the highest rank	12	52	40	40	12	34	39	21
		Rank	7-8	1	2-3	2-3	7-8	5	4	6
	Added scheme ranking	Added ranking	650	359	353	363	549	403	433	536
		Rank	8	2	1	3	7	4	5	6
Large fault trees	Total quantity ranking	Number of nodes	13432	10302	9370	8026	12967	8312	8730	11744
		Rank	8	5	4	1	7	2	3	6
	Highest scheme ranking	Number of FTs with the highest rank	0	3	1	2	1	2	4	0
		Rank	7-8	2	5-6	3-4	5-6	3-4	1	7-8
	Added scheme ranking	Added ranking	86	57	57	39	67	45	43	59
		Rank	8	4-5	4-5	1	7	3	2	6
All fault trees	Total quantity ranking	Number of nodes	32764	26241	25290	23784	30136	23391	24538	28281
		Rank	8	5	4	2	7	1	3	6
	Highest scheme ranking	Number of FTs with the highest rank	12	55	41	42	13	36	43	21
		Rank	8	1	4	3	7	5	2	6
	Added scheme ranking	Added ranking	736	416	410	402	616	448	476	595
		Rank	8	3	2	1	7	4	5	6

Table 6.6: ZBDD method, number of nodes

The depth-first, with number of leaves scheme (4) performed well for the number of nodes and the processing time. It was highly ranked for small and 'large' fault

		Scheme	1	2	3	4	5	6	7	8
Small fault trees	Total quantity ranking	Time	32 84	33 3	33 78	32 81	34 47	33 39	33 84	33 65
		Rank	2	3	6	1	8	4	7	5
	Highest scheme ranking	Number of FTs with the highest rank	104	91	63	100	50	70	64	55
		Rank	1	3	6	2	8	4	5	7
	Added scheme ranking	Added ranking	466	474	562	450	680	564	561	607
		Rank	2	3	5	1	8	6	4	7
Large fault trees	Total quantity ranking	Time	9 96	15 44	13 89	8 78	10 48	8 54	10 77	14 78
		Rank	3	8	6	2	4	1	5	7
	Highest scheme ranking	Number of FTs with the highest rank	0	3	3	4	5	5	2	4
		Rank	8	5-6	5-6	3-4	1-2	1-2	7	3-4
	Added scheme ranking	Added ranking	59	64	57	39	47	34	63	39
		Rank	6	8	5	2-3	4	1	7	2-3
All fault trees	Total quantity ranking	Time	42 8	48 74	47 67	41 59	44 95	41 93	44 61	48 43
		Rank	3	8	6	1	5	2	4	7
	Highest scheme ranking	Number of FTs with the highest rank	104	94	66	104	55	75	66	59
		Rank	1-2	3	5-6	1-2	8	4	5-6	7
	Added scheme ranking	Added ranking	525	538	619	489	727	598	624	646
		Rank	2	3	5	1	8	4	6	7

Table 6 7: ZBDD method, processing time

trees using three ranking methods. For the number of nodes the dynamic top-down weighted scheme (6) also performed well. The modified top-down scheme (1) performed poorly for the number of nodes but it gave good results for the processing time. The worst performance according to the processing time was obtained by the non-dynamic top-down weighted scheme (5). Overall, the difference in performance of the eight ordering schemes is marginal, especially, for the processing time, therefore, no further conclusions can be drawn.

6.3.4.4 L-BDD method results

Summary details of the test fault trees are presented in Tables A.97 - A 100, which show the sum of the number of nodes in the L-BDD before applying the intermediate calculations and the minimisation, which is equivalent to the SFBDD, and the number of nodes in the minimised L-BDD. Table A.101 shows the sum of nodes for the large fault trees. The time taken to convert fault trees to L-BDDs, then minimise the L-BDDs and perform the full qualitative analysis is shown in Tables A 102 - A 105 and Table A.106 for small and large fault trees respectively.

Some fault trees were simplified so much that their SFBDDs contain only 1 node.

There were 45 fault trees of that type. Also, for 27 fault trees the eight ordering schemes gave the same results. Those fault trees are not taken into account while ranking the schemes. Overall, there were 147 fault trees considered for the analysis, where 13 of them were 'large' fault trees. There were 10 fault trees that the processing time was the same for the eight ordering schemes. Those fault trees were not taken into account while performing the ranking according to the time.

6.3.4.4.1 Variable ordering for the L-BDD method

Analysis of number of nodes in LBDDs and processing time for 'small', 'large' and all example fault trees is presented in this section, applying the three ranking techniques of the ordering schemes. The result of the analysis is shown in Table 6.8 and Table 6.9 for the number of nodes and the processing time respectively.

		Scheme	1	2	3	4	5	6	7	8
Small fault trees	Total quantity ranking	Number of nodes	48400	50938	47662	45908	39778	38344	63687	36768
		Rank	6	7	5	4	3	2	8	1
	Highest scheme ranking	Number of FTs with the highest rank	13	33	22	35	23	45	32	35
		Rank	8	4	7	2-3	6	1	5	2-3
	Added scheme ranking	Added ranking	582	511	507	467	474	361	562	443
		Rank	8	6	5	3	4	1	7	2
Large fault trees	Total quantity ranking	Number of nodes	250867	351974	312349	300990	286085	219140	415505	364114
		Rank	2	6	5	4	3	1	8	7
	Highest scheme ranking	Number of FTs with the highest rank	2	0	0	2	2	2	2	3
		Rank	2-6	7-8	7-8	2-6	2-6	2-6	2-6	1
	Added scheme ranking	Added ranking	47	89	78	58	36	47	69	38
		Rank	3-4	8	7	5	1	3-4	6	2
All fault trees	Total quantity ranking	Number of nodes	299267	402912	360011	346898	325863	257484	479192	400882
		Rank	2	7	5	4	3	1	8	6
	Highest scheme ranking	Number of FTs with the highest rank	15	33	22	37	25	47	34	38
		Rank	8	5	7	3	6	1	4	2
	Added scheme ranking	Added ranking	629	600	585	525	510	408	631	481
		Rank	7	6	5	4	3	1	8	2

Table 6.8: LBDD method, number of nodes

For the number of nodes the dynamic top-down scheme (6) performed well and was ranked highly for the main part of fault trees using the three different ranking techniques. For the processing time this ordering scheme gave average results and the best performance was obtained by the depth first, with number of leaves scheme (4). For the number of nodes and the processing time the worst performance was

		Scheme	1	2	3	4	5	6	7	8
Small fault trees	Total quantity ranking	Time	253 01	244 02	170 02	79 01	95 47	114 4	261 63	77 36
		Rank	7	6	5	2	3	4	8	1
	Highest scheme ranking	Number of FTs with the highest rank	52	54	51	87	68	66	55	74
		Rank	7	6	8	1	3	4	5	2
	Added scheme ranking	Added ranking	632	576	600	449	536	539	623	521
		Rank	8	5	6	1	3	4	7	2
Large fault trees	Total quantity ranking	Time	15833 69	10838 13	7085 68	883 09	34101 91	3660 23	68085 38	54179 03
		Rank	5	4	3	1	6	2	8	7
	Highest scheme ranking	Number of FTs with the highest rank	0	2	0	4	1	4	1	1
		Rank	7-8	3	7-8	1-2	4-6	1-2	4-6	4-6
	Added scheme ranking	Added ranking	75	67	66	42	49	44	64	61
		Rank	8	7	6	1	3	2	5	4
All fault trees	Total quantity ranking	Time	16086 7	11082 15	7255 7	962 1	34197 38	3774 63	68347 01	54256 39
		Rank	5	4	3	1	6	2	8	7
	Highest scheme ranking	Number of FTs with the highest rank	52	56	51	91	69	70	56	75
		Rank	7	5-6	8	1	4	3	5-6	2
	Added scheme ranking	Added ranking	707	643	666	491	585	583	687	582
		Rank	8	5	6	1	4	3	7	2

Table 6.9: LBDD method, processing time

obtained by the modified top-down scheme (1) and the bottom-up weighted scheme (7).

6.3.5 Overall Variable Ordering for the Qualitative Analysis of Non-coherent Fault Trees

A theoretical comparison of the methods is explained in Table 6 10, where the main advantages and disadvantages are identified.

Analysing the results obtained of the four methods for the encoding prime implicant sets, it can be said that the four methods are quite different in their efficiency. Their performance is shown in the following tables. The comparison of the four techniques according to the number of nodes is shown in Table 6 11 Also, the comparison according to the processing time is shown in Table 6 12.

Analysing these results it was clear that there were two methods that performed well on the example fault trees They were the TDD method and the ZBDD method For the number of nodes the ZBDD method performed slightly better

Method	Minimisation	Construction technique	Advantages	Disadvantages
Ternary Decision Diagram (TDD)	Required	Basic event x is coded by a node with three branches. The third branch is the conjunction of the 1 and the 0 branches and the conjunction is performed for every node that represents failure and repair relevant component.	A clear representation of all implicant sets is provided.	1 Minimisation is required 2 TDD size is 3^n .
Meta-products BDD	Not required	Basic event x is coded by two variables P_x and S_x .	No minimisation is required.	1 Meta-products BDD size is 2^{2n} 2 It results in a time consuming process.
Zero-suppressed BDD (ZBDD)	Not required	Basic event x is coded by x and, in addition, by \bar{x} , if needed. The conjunction of the 1 and the 0 branches is performed only for dual state variables.	1 No minimisation is required 2 Compact representation of prime implicant sets is provided.	Dual state events are coded by two variables.
Labelled BDD (L-BDD)	Required	Basic event x is labelled according to its occurrence, ($\&x<x<\$ x$). The conjunction of the 1 and the 0 branches is performed only for dual state variables.	Labels identify where the conjunction of the 1 and the 0 branches is required.	1 Minimisation is required 2 Labelling introduces some additional variables.

Table 6.10 A theoretical comparison of the four techniques

than the TDD method and the TDD method resulted in a slightly faster process than the ZBDD method. The L-BDD method appeared to be the second worst method. The worst method was the Meta-products BDD method.

As expected the Meta-products BDD algorithm produced significantly larger final BDDs than any other method that was used for the calculation of prime implicant sets. Processing time was also greater. This is due to the fact that after the conversion of fault tree to a SFBDD another BDD, called the Meta-products BDD, is required, where all components are described by two variables. This increased the size of the BDD and the processing time unavoidably.

	Scheme	1	2	3	4	5	6	7	8
Small fault trees	TDD method	24167	23050	21729	20434	19976	18138	23339	18515
	MPPI method	59491	56186	54162	51510	51778	48200	51931	49115
	ZBDD method	19332	15939	15920	15758	17169	15079	15808	16537
	L-BDD method	48400	50938	47662	45908	39778	38344	63687	36768
Large fault trees	TDD method	6634	7558	7479	4739	5049	4688	5806	4409
	MPPI method	1779183	1689973	1774833	533464	1246614	508603	1493336	621596
	ZBDD method	5058	4860	4406	3337	3902	3189	3876	3405
	L-BDD method	45972	80406	73827	61089	39806	48299	60518	35790
All fault trees	TDD method	30801	30608	29208	25173	25025	22826	29145	22924
	MPPI method	1838674	1746159	1828995	584974	1298392	556803	1545267	670711
	ZBDD method	24390	20799	20326	19095	21071	18268	19684	19942
	L-BDD method	94372	131344	121489	106997	79584	86643	124205	72558

Table 6.11: Summary of the four methods, number of nodes

The TDD method resulted in an efficient fault tree conversion process and a fast qualitative analysis. In the TDD method the consensus terms were produced by finding the conjunction of the 1 and the 0 branches for every node that is failure and repair relevant. The ZBDD method also gave good results. The SFBDD was constructed and then converted to the ZBDD, which encoded the prime implicant sets. On one hand, the TDD method was slightly faster than the ZBDD method. That could be explained by the fact that only one structure (a TDD) was required in this method, whereas in the ZBDD method a SFBDD was constructed and then an additional ZBDD was built. On the other hand, the ZBDD was minimal therefore a smaller number of nodes was obtained than in the TDD method, whereas a TDD needed to be minimised before obtaining prime implicant sets.

Comparing the TDD method and the L-BDD method, the idea of both methods was very similar, i.e. the conjunction of the 1 branch and the 0 branch was only carried out for nodes that represent dual state basic events. However, there were some differences that made the L-BDD method perform less efficiently than the TDD method. In the L-BDD method the number of basic events was increased by introducing three different types of basic events ($\&x < x < \$x$), according to

	Scheme	1	2	3	4	5	6	7	8
Small fault trees	TDD method	22 51	22 54	22 52	22 33	21 96	21 97	22 84	22 45
	MPPI method	16072 92	9125 99	5941 00	8113 49	3961 66	2070 27	5511 35	1442 22
	ZBDD method	32 84	33 30	33 78	32 81	34 47	33 39	33 84	33 65
	L-BDD method	253 01	244 02	170 02	79 01	95 47	114 40	261 63	77 36
Large fault trees	TDD method	0 80	1 81	1 60	0 59	0 54	0 56	1 03	0 45
	MPPI method	65906 72	83737 84	117648 78	13181 58	51992 24	12201 20	60442 25	11027 12
	ZBDD method	0 99	1 87	1 61	0 66	0 64	0 65	1 88	0 59
	L-BDD method	506 19	148 08	117 17	38 09	247 84	55 26	918 48	110 32
All fault trees	TDD method	23 31	24 35	24 12	22 92	22 50	22 53	23 87	22 90
	MPPI method	81979 64	92863 83	123589 78	21295 07	55953 90	14271 47	65953 60	12469 34
	ZBDD method	33 83	35 17	35 39	33 47	35 11	34 04	35 72	34 24
	L-BDD method	759 20	392 10	287 19	117 10	343 31	169 66	1180 11	187 68

Table 6.12: Summary of the four methods, processing time

their occurrence. Also, some extra connection rules were used in the conversion process and in the calculation of prime implicant sets. The TDD technique was clearer and more efficient than the L-BDD technique, i.e. it simply encoded the third branch that enabled 'hidden' prime implicant sets to be calculated.

Overall, the efficiency of the four methods was very similar when small fault trees were considered. The difference in both measurements between the TDD method and the ZBDD method was marginal. Even the L-BDD method and the meta-products BDD gave close results, especially for the number of nodes, considering small fault trees. However, the analysis of 'large' fault trees allowed a better way to test the methods, because while dealing with 'large' fault trees, the advantages and disadvantages of the different techniques were more noticeable. The TDD method and the ZBDD method performed a lot better than the two other techniques and should be used for the BDD analysis of non-coherent fault trees. Also, considering 'large' fault trees the meta-products BDD method performed a lot worse than the L-BDD method. The difference between the two advantageous methods was still marginal. Therefore, any of the two methods used for a different library of example fault trees should provide a similar efficiency of the analysis.

6.4 Quantitative Analysis

6.4.1 Introduction

Unlike the conventional Fault Tree Analysis the BDD method does not require knowledge of prime implicant sets for quantification. It is therefore possible to use the SFBDD to perform full and exact quantitative analysis. Since the L-BDD is in a similar format to the SFBDD, it can also be quantified. The quantification of the TDD is presented as a part of this research.

6.4.2 System Unavailability

The SFBDD for a non-coherent fault tree encodes Shannon's decomposition. Therefore, for calculating the top event probability for a non-coherent fault tree the SFBDD can be employed. It is obtained as a sum of probabilities of the disjoint paths through the BDD, as it was shown in the coherent case in Chapter 3. Consider example SFBDD in Figure 6.1. There are three paths from the root node to the terminal vertex 1, therefore, the $Q(t)$ is expressed as

$$Q(t) = q_b q_a + q_b p_a q_c + p_b p_a q_c. \quad (6.123)$$

There are three prime implicant sets represented by this SFBDD and using the inclusion-exclusion expansion gives the same expression:

$$Q(t) = q_b q_a + q_b q_c + p_a q_c - q_b q_a q_c - q_b p_a q_c = q_b q_a + q_b q_c p_a + p_a q_c p_b \quad (6.124)$$

Since the L-BDD also encodes the SFBDD before starting the calculations for the qualitative analysis. Therefore, the L-BDD can be used in the same way as the SFBDD, assuming that $q_{\&x} = q_x$ and $q_{\$x} = p_x$. Traversing the example L-BDD in Figure 6.11 gives the $Q(t)$ expression

$$Q(t) = q_b q_a + q_b p_a q_c + p_b q_{\$a} q_c = q_b q_a + q_b p_a q_c + p_b p_a q_c \quad (6.125)$$

The system unavailability can be calculated directly from the TDD, because the SFBDD is encoded within the TDD. The SFBDD could be obtained if all consensus branches were removed from the TDD. The system unavailability can be calculated as a sum of probabilities of the disjoint paths through the BDD that only pass through the 1 and 0 branches of non-terminal nodes. Consider example TDD in Figure 6.4. Again, there are three paths from the root vertex to the terminal

vertex 1, because paths passing through consensus branches of non-terminal nodes are ignored. The three paths are ba , $b\bar{a}c$, $\bar{b}ac$ therefore, the $Q(t)$ is calculated as

$$Q(t) = q_b q_a + q_b p_a q_c + p_b p_a q_c \quad (6.126)$$

6.4.3 Importance Measures

The BDD method enables exact and efficient calculation of the measures of importance to be performed eliminating both the intermediate stage of identifying the prime implicant sets and the need to evaluate lengthy series expansions. Birnbaum's measure of component failure and repair importance is presented for the case of the SFBDD, that has been established in [37]. The calculation process is easy to adapt in the L-BDD case, since the SFBDD and the L-BDD structure are very similar. The Birnbaum's measure for the TDD technique is developed in this research and presented later in the section.

6.4.3.1 Birnbaum's measure of failure and repair importance

The failure importance of component i is defined as the probability that the system is in a working state such that failure of component i would cause system failure. Therefore, it is possible to define Birnbaum's measure of component failure importance as

$$G_i^F(\mathbf{q}) = E[T_{i=1}] - E[T_{i=-}], \quad (6.127)$$

where $E[T_{i=1}]$ is the probability that component i is either failure relevant or irrelevant to the state of the system and $E[T_{i=-}]$ is the probability that component i is irrelevant.

Similarly, Birnbaum's measure of component repair importance can be defined as the probability that component i is repair relevant to the system state:

$$G_i^R(\mathbf{q}) = E[T_{i=0}] - E[T_{i=-}], \quad (6.128)$$

where $E[T_{i=0}]$ is the probability that component i is repair relevant or irrelevant to the system state.

Once Birnbaum's measure has been calculated it enables the unconditional failure intensity and the expected number of system failures in a given interval to be calculated.

6.4.3.1.1 The SFBDD method

The procedure for calculating the probabilities $E[T_{i=1}]$ and $E[T_{i=0}]$ is outlined here:

$$E[T_{i=1}] = \sum_{x_i} pr_{x_i}(\mathbf{q}) \cdot po_{x_i}^1(\mathbf{q}), \quad (6.129)$$

$$E[T_{i=0}] = \sum_{x_i} pr_{x_i}(\mathbf{q}) \cdot po_{x_i}^0(\mathbf{q}), \quad (6.130)$$

where the path probabilities $pr_{x_i}(\mathbf{q})$, $po_{x_i}^1(\mathbf{q})$, $po_{x_i}^0(\mathbf{q})$ were explained in the coherent case in Chapter 3, section 3.6.2.

The expression for $E[T_{i=-}]$ can be obtained by computing an intermediate BDD for each node, that is calculated by the conjunction of the 1 and 0 branches of the node. Then the $E[T_{i=-}]$ is obtained by multiplying the probability preceding the node in the SFBDD by the sum of probabilities of all paths to terminal vertex 1. Considering the example SFBDD in Figure 6.1 and its path probabilities in Table 6.13 Probabilities $E[T_{i=1}]$ and $E[T_{i=0}]$ will be calculated according to equa-

Node	$pr_{x_i}(\mathbf{q})$	$po_{x_i}^1(\mathbf{q})$	$po_{x_i}^0(\mathbf{q})$
F1	1	$q_a + p_a q_c$	$p_a q_c$
F2	q_b	1	q_c
F3	p_b	0	q_c
F4	$q_b p_a + p_b p_a$	1	0

Table 6.13: Path probabilities for each node in the SFBDD in Figure 6.1

tions 6.129 and 6.130. For calculation of $E[T_{i=-}]$ intermediate BDDs are obtained and their path probabilities used.

For node $F1 = \text{ite}(b, F2, F3)$:

$$F2 \cdot F3 = \text{ite}(a, 1, F4) \cdot \text{ite}(a, 0, F4) = \text{ite}(a, 0, F4). \quad (6.131)$$

The only path through this SFBDD is $\bar{a}c$, its probability is $p_a q_c$.

For node $F2 = \text{ite}(a, 1, F4)$

$$1 \cdot F4 = F4. \text{ The only path is } c, \text{ its probability is } q_c. \quad (6.132)$$

For node $F3 = \text{ite}(a, 0, F4)$

$$0 \cdot F4 = 0 \text{ The probability is } 0 \quad (6.133)$$

Dealing with node $F4 = \text{ite}(c, 1, 0)$ gives 0

Summary of results obtained for $E[T_{i=1}]$, $E[T_{i=0}]$ and $E[T_{i=-}]$ is shown in Table 6.14. Using these results and equations 6.127 and 6.128 the failure and repair

Node	$E[T_{i=1}]$	$E[T_{i=0}]$	$E[T_{i=-}]$
F1	$q_a + p_a q_c$	$p_a q_c$	$p_a q_c$
F2	q_b	$q_b q_c$	$q_b q_c$
F3	0	$p_b q_c$	0
F3	$q_b p_a + p_b p_a$	0	0

Table 6.14 Summary of results for the SFBDD of $E[T_{i=1}]$, $E[T_{i=0}]$ and $E[T_{i=-}]$

importance of each component is calculated

$$G_b^F = q_a + p_a q_c - p_a q_c = q_a, \quad (6.134)$$

$$G_a^F = q_b - q_b q_c + 0 - 0 = q_b p_c, \quad (6.135)$$

$$G_c^F = q_b p_a + p_b p_a = p_a, \quad (6.136)$$

$$G_b^R = p_a q_c - p_a q_c = 0, \quad (6.137)$$

$$G_a^R = q_b q_c - q_b q_c + p_b q_c - 0 = p_b q_c, \quad (6.138)$$

$$G_c^R = 0 - 0 = 0 \quad (6.139)$$

The unconditional failure intensity can be calculated:

$$w_{sys}(t) = q_a w_b + q_b p_c w_a + p_a w_c + p_b q_c v_a. \quad (6.140)$$

6.4.3.1.2 The L-BDD method

The similar approach to the SFBDD method can be applied in the L-BDD technique. The calculation of the probabilities is outlined below.

$$E[T_{i=1}] = \sum_{x_i, (\&x, x)} pr_{x_i}(\mathbf{q}) \cdot po_{x_i}^1(\mathbf{q}) + \sum_{x_i, (\$x)} pr_{x_i}(\mathbf{q}) \cdot po_{x_i}^0(\mathbf{q}), \quad (6.141)$$

$$E[T_{i=0}] = \sum_{x_i, (\&x, x)} pr_{x_i}(\mathbf{q}) \cdot po_{x_i}^0(\mathbf{q}) + \sum_{x_i, (\$x)} pr_{x_i}(\mathbf{q}) \cdot po_{x_i}^1(\mathbf{q}). \quad (6.142)$$

For $\$x$ variables extra summing is required 'switching' the 1 and 0 branches over.

Consider the L-BDD in Figure 6.11 and its path probabilities in Table 6.15. Probabilities

Node	$pr_{x_i}(\mathbf{q})$	$po_{x_i}^1(\mathbf{q})$	$po_{x_i}^0(\mathbf{q})$
F1	1	$q_a + p_a q_c$	$q_s a q_c$
F2	q_b	1	q_c
F3	$q_b p_a + p_b q_s a$	1	0
F4	p_b	q_c	0

Table 6 15 Path probabilities for each node in the L-BDD in Figure 6 11

$E[T_{i=1}]$ and $E[T_{i=0}]$ will be calculated according to equations 6 141 and 6.142. For calculation of $E[T_{i=-}]$ intermediate BDDs are obtained and their paths probabilities used.

For node $F1 = \text{ite}(b, F2, F4)$:

$$\begin{aligned} F2 \cdot F4 &= \text{ite}(a, 1, F3) \cdot \text{ite}(\$a, F3, 0) = \text{ite}(\&a, 0, F3) \\ &= \text{ite}(\$a, F3, 0) \end{aligned} \quad (6\ 143)$$

The only path through this L-BDD is $\$ac$, its probability is $q_s a q_c$

For node $F2 = \text{ite}(a, 1, F3)$:

$$1 \cdot F3 = F3. \text{ The only path is } c, \text{ its probability is } q_c \quad (6\ 144)$$

Dealing with node $F3 = \text{ite}(c, 1, 0)$ gives 0

For node $F4 = \text{ite}(\$a, F3, 0)$

$$F3 \cdot 0 = 0. \text{ The probability is } 0 \quad (6.145)$$

Summary of results obtained for $E[T_{i=1}]$, $E[T_{i=0}]$ and $E[T_{i=-}]$ are shown in Table 6 16. Using these results and equations 6 127 and 6 128 the failure and repair importance of each component is calculated.

$$G_b^F = q_a + p_a q_c - q_s a q_c = q_a, \quad (6\ 146)$$

$$G_a^F = q_b - q_b q_c + 0 - 0 = q_b p_c, \quad (6\ 147)$$

$$G_c^F = q_b p_a + p_b q_s a = p_a, \quad (6\ 148)$$

$$G_b^R = q_s a q_c - q_s a q_c = 0, \quad (6.149)$$

$$G_a^R = q_b q_c - q_b q_c + p_b q_c - 0 = p_b q_c, \quad (6\ 150)$$

$$G_c^R = 0 - 0 = 0 \quad (6.151)$$

Node	$E[T_{i=1}]$	$E[T_{i=0}]$	$E[T_{i=-}]$
F1	$q_a + p_a q_c$	$q_{s_a} q_c$	$q_{s_a} q_c$
F2	q_b	$q_b q_c$	$q_b q_c$
F3	$q_b p_a + p_b q_{s_a}$	0	0
F4	0	$p_b q_c$	0

Table 6 16: Summary of results for the L-BDD of $E[T_{i=1}]$, $E[T_{i=0}]$ and $E[T_{i=-}]$

These results are the same as the results obtained using the SFBDD in Equations 6.134 - 6.139. The expression of the unconditional failure intensity is also identical to Equation 6.140

6.4.3.1.3 The TDD method

Terms $E[T_{i=1}]$, $E[T_{i=0}]$ and $E[T_{i=-}]$ can be calculated directly from the TDD. The procedure is outlined below:

$$E[T_{i=1}] = \sum_{x_i} pr_{x_i}(\mathbf{q}) \cdot po_{x_i}^{1,c}(\mathbf{q}), \quad (6.152)$$

$$E[T_{i=0}] = \sum_{x_i} pr_{x_i}(\mathbf{q}) \cdot po_{x_i}^{0,c}(\mathbf{q}), \quad (6.153)$$

$$E[T_{i=-}] = \sum_{x_i} pr_{x_i}(\mathbf{q}) \cdot po_{x_i}^c(\mathbf{q}), \quad (6.154)$$

where

- $po_{x_i}^{1,c}$ is the probability of the path section from the one branch of node x_i to a terminal vertex 1 via only 1 and 0 branches of non-terminal nodes,
- $po_{x_i}^{0,c}$ is the probability of the path section from the zero branch of node x_i to a terminal vertex 1 via only 1 and 0 branches of non-terminal nodes,
- $po_{x_i}^c$ is the probability of the path section from the consensus branch of node x_i to a terminal vertex 1 via only 1 and 0 branches of non-terminal nodes

Therefore, the failure and repair criticality of component i are expressed as:

$$G_i^F(\mathbf{q}) = \sum_{x_i} pr_{x_i}(\mathbf{q}) [po_{x_i}^{1,c}(\mathbf{q}) - po_{x_i}^c(\mathbf{q})], \quad (6.155)$$

$$G_i^R(\mathbf{q}) = \sum_{x_i} pr_{x_i}(\mathbf{q}) [po_{x_i}^{0,c}(\mathbf{q}) - po_{x_i}^c(\mathbf{q})] \quad (6.156)$$

These expressions are true for every component i that is failure and repair relevant

The other two cases, i.e. when component i is failure relevant or repair relevant, are described below where the 'C' branch contains the value *NIL*

If component i is only failure relevant, its criticality is calculated as

$$G_i^F(\mathbf{q}) = \sum_{x_i} pr_{x_i}(\mathbf{q}) [po_{x_i}^{1,c}(\mathbf{q}) - po_{x_i}^{0,c}(\mathbf{q})], \quad (6.157)$$

$$G_i^R(\mathbf{q}) = 0 \quad (6.158)$$

If component i is only repair relevant, its criticality is calculated as

$$G_i^F(\mathbf{q}) = 0, \quad (6.159)$$

$$G_i^R(\mathbf{q}) = \sum_{x_i} pr_{x_i}(\mathbf{q}) [po_{x_i}^{0,c}(\mathbf{q}) - po_{x_i}^{1,c}(\mathbf{q})]. \quad (6.160)$$

The application of the algorithm in practice to calculate both the failure and repair importance of components is explained using the example TDD in Figure 6.4. Now

Node	$pr_{x_i}(\mathbf{q})$	$po_{x_i}^{1,c}(\mathbf{q})$	$po_{x_i}^{0,c}(\mathbf{q})$	$po_{x_i}^c(\mathbf{q})$
F1	1	$q_a + p_a q_c$	$p_a q_c$	-
F2	q_b	1	q_c	q_c
F3	p_b	0	q_c	0
F4	$q_b p_a + p_b p_a$	1	0	-

Table 6.17 Path probabilities for each node in the TDD in Figure 6.4

the failure and repair importance for each component can be calculated by summing the contributions of nodes of the same component. Thus using the expressions in equations 6.155 and 6.156 and probabilities in Table 6.17 gives:

$$G_b^F = 1 \cdot [q_a + p_a q_c - p_a q_c] = q_a, \quad (6.161)$$

$$G_a^F = q_b [1 - q_c] + p_b \cdot [0 - 0] = q_b p_c, \quad (6.162)$$

$$G_c^F = (q_b p_a + p_b p_a) \cdot [1 - 0] = p_a, \quad (6.163)$$

$$G_b^R = 0, \quad (6.164)$$

$$G_a^R = q_b \cdot [q_c - q_c] + p_b \cdot [q_c - 0] = p_b q_c, \quad (6.165)$$

$$G_c^R = 0 \quad (6.166)$$

These results, as well as the unconditional failure intensity, are the same as using the SFBDD or the L-BDD method.

This concludes the presentation of the quantitative analysis of non-coherent fault trees converted to BDDs.

6.5 Hybrid Method in the Non-coherent Fault Tree Case

As a good alternative BDD construction technique for coherent fault trees, the Hybrid method can also be utilised in the conversion of non-coherent fault trees to SFBDDs. As it was presented in the coherent case the Hybrid method combines the best features of the two construction methods of BDDs, the *ite* method, presented in Chapter 3 and the Component Connection Method, presented in Chapter 4. The new method incorporates the most efficient parts of both algorithms. In this section the two strategies are compared using the two efficiency measures, the number of nodes and the processing time.

According to the first strategy a non-coherent fault tree is simplified, then it is converted to an SFBDD using the *ite* technique. For the qualitative analysis its corresponding SFBDD is converted to an ZBDD which produces prime implicant sets. The ZBDD method was chosen as one of the two most efficient tools for the qualitative analysis of non-coherent fault trees. It does not matter which method is chosen to get the prime implicant sets, because the test on the Hybrid method will be performed while a SFBDD is constructed, i.e. prior to the qualitative analysis. The qualitative analysis is performed more for the validation purposes of the implementation of the method, rather than for the efficiency test of the method.

The second strategy utilises the Hybrid technique for the conversion of simplified non-coherent fault trees to SFBDDs and then the ZBDD method is also applied for the qualitative analysis. The Advanced Hybrid method is used in the analysis since it was more efficient than the Basic Hybrid method for the coherent fault trees.

The library of 13 large non-coherent fault trees is used in the analysis. Their complexity is shown in Table A.62. These are the same fault trees that were used in the analysis of non-coherent fault trees using the four different methods for the

qualitative analysis

The number of nodes in the resulting SFBDDs is the same using the *ite* technique and the Hybrid method. The Hybrid method expresses the structure function of a fault tree in terms of a BDD in the same way as it does the *ite* technique. Therefore, the final result of both methods was the same and was shown in Table A 91

The processing time and the maximum required size of array were also analysed as was done in the efficiency analysis of the Hybrid method in the coherent case. The results for the *ite* method are shown in Tables A 96 and A 108, and for the Hybrid method - in Tables A 107 and A 109. The comparison of the two techniques according to the processing time and the maximum required size of array is shown in Table 6.18

		Scheme	1	2	3	4	5	6	7	8
Time	<i>ite</i> method		9 96	15 44	13 89	8 78	10 48	8 54	10 77	14 78
	Hybrid method		9 97	15 1	13 47	9 08	10 55	8 5	10 89	14 88
Maximum array size	<i>ite</i> method		12488	10505	10176	8280	11940	7868	8825	10667
	Hybrid method		12330	10221	9891	8027	11780	7604	8682	10488

Table 6 18· Comparison of the two techniques

This table shows that the processing time is very similar using the *ite* technique and the Hybrid method while building SFBDDs. However, the maximum required size of array decreases if the Hybrid method is used instead of the *ite* technique. This comparison for non-coherent fault trees matches the results in the coherent case quite well. The Hybrid method appears to be a good alternative technique to the *ite* method for converting non-coherent fault trees to SFBDDs

The ranking of the eight ordering schemes was also performed. The results according to the number of nodes, the processing time and the maximum required size are shown in Tables 6 19 - 6 21. Ranking results are similar between the two methods. The best performance was obtained by the dynamic top-down scheme (6) for almost all efficiency measures and ranking methods. The modified top-down scheme (1) gave poor results. The rankings are similar to the ones obtained in the

		Scheme	1	2	3	4	5	6	7	8
ite method	Total quantity ranking	Number of nodes	13432	10302	9370	8026	12967	8312	8730	11744
		Rank	8	5	4	1	7	2	3	6
	Highest scheme ranking	Number of FTs with the highest rank	0	3	1	2	1	2	4	0
		Rank	7-8	2	5-6	3-4	5-6	3-4	1	7-8
	Added scheme ranking	Added ranking	86	57	57	39	67	45	43	59
		Rank	8	4-5	4-5	1	7	3	2	6
Hybrid method	Total quantity ranking	Number of nodes	13432	10302	9370	8026	12967	8312	8730	11744
		Rank	8	5	4	1	7	2	3	6
	Highest scheme ranking	Number of FTs with the highest rank	0	3	1	2	1	2	4	0
		Rank	7-8	2	5-6	3-4	5-6	3-4	1	7-8
	Added scheme ranking	Added ranking	86	57	57	39	67	45	43	59
		Rank	8	4-5	4-5	1	7	3	2	6

Table 6 19 ite and Hybrid method, ranking according to the number of nodes

coherent case for 'large' fault trees

6.6 Summary

The BDD method for the analysis of non-coherent fault trees can be more efficient and accurate than the conventional techniques of non-coherent FTA. The qualitative analysis can be performed fully using the BDD method and the system unavailability can be calculated without knowing prime implicant sets. Therefore, there are no long series expansions to perform that can be very inefficient. The importance measures can also be calculated using the BDDs.

In some situations the prime implicant sets can be valuable, especially while planning repair schedules and designing the incorporation of safety systems. The SF-BDD is only suitable for the quantitative analysis. It is not suitable for the qualitative analysis because in the SFBDD structure there is no distinction of component failure relevance, repair relevance or irrelevance.

A new method, the Ternary Decision Diagram method, was presented in this chapter for the calculation of prime implicant sets. This method produced a TDD where all prime implicants were obtained after the minimisation. In the second method, initially a SFBDD was produced and then it was converted to a Meta-products BDD, containing all prime implicant sets. The third method resulted in a Zero-

		Scheme	1	2	3	4	5	6	7	8
ite method	Total quantity ranking	Time	9 96	15 44	13 89	8 78	10 48	8 54	10 77	14 78
		Rank	3	8	6	2	4	1	5	7
	Highest scheme ranking	Number of FTs with the highest rank	0	3	3	4	5	5	2	4
		Rank	8	5-6	5-6	3-4	1-2	1-2	7	3-4
	Added scheme ranking	Added ranking	59	64	57	39	47	34	63	39
		Rank	6	8	5	2-3	4	1	7	2-3
Hybrid method	Total quantity ranking	Time	9 97	15 1	13 47	9 08	10 55	8 5	10 89	14 88
		Rank	3	8	6	2	4	1	5	7
	Highest scheme ranking	Number of FTs with the highest rank	0	2	2	1	4	4	2	6
		Rank	8	4-6	4-6	7	2-3	2-3	4-6	1
	Added scheme ranking	Added ranking	54	59	54	44	40	31	59	30
		Rank	5-6	7-8	5-6	4	3	2	7-8	1

Table 6 20 ite and Hybrid method, ranking according to the processing time

		Scheme	1	2	3	4	5	6	7	8
ite method	Total quantity ranking	Maximum array size	12488	10505	10176	8280	11940	7868	8825	10667
		Rank	8	5	4	2	7	1	3	6
	Highest scheme ranking	Number of FTs with the highest rank	0	1	1	3	2	4	2	0
		Rank	7-8	5-6	5-6	2	3-4	1	3-4	7-8
	Added scheme ranking	Added ranking	85	65	64	43	62	32	55	54
		Rank	8	7	6	2	5	1	4	3
Hybrid method	Total quantity ranking	Maximum array size	12330	10221	9891	8027	11780	7604	8682	10488
		Rank	8	5	4	2	7	1	3	6
	Highest scheme ranking	Number of FTs with the highest rank	0	1	1	3	2	4	2	0
		Rank	7-8	5-6	5-6	2	3-4	1	3-4	7-8
	Added scheme ranking	Added ranking	85	64	63	43	63	31	55	54
		Rank	8	7	5-6	2	5-6	1	4	3

Table 6.21: ite and Hybrid method, ranking according to the maximum array size

suppressed BDD that enabled an efficient qualitative process to be achieved. The fourth method computed a Labelled Binary Decision Diagram, where all variables were labelled and prime implicant sets were obtained after the minimisation. The efficiency of the new method was estimated comparing it to the conventional approaches.

A comparison of the four methods revealed that the TDD method and the ZBDD method performed well for both measurements, i.e. the number of nodes and the conversion time. The TDD method was slightly faster than the ZBDD method.

That possibly was due to the fact that one structure (a TDD) was required in this method, which presented a SFBDD as well as it was suitable for the qualitative analysis. In the ZBDD method a SFBDD was constructed and then an additional ZBDD was built that resulted in a longer conversion process. However, the ZBDD structure was minimal, therefore the number of nodes was smaller than in the TDD method where each TDD was minimised before obtaining prime implicant sets. Both methods gave close results and provided the efficient means for the analysis of non-coherent fault trees.

The Meta-products BDD algorithm produced a lot larger final BDDs than in any other methods that were used for the calculation of prime implicant sets. Processing time was also greater. This was due to the fact that after the conversion of fault tree to a SFBDD another BDD, called the Meta-products BDD, was required, where all components were described by two variables. This increased the size of the BDD and the processing time unavoidably.

During the L-BDD method labelled variables were introduced, using three different types of basic events ($\&x < x < \$x$). In order to get all prime implicant sets the conjunction of the 1 branch and the 0 branch was performed for nodes that represent dual state basic events, as it was done in the TDD method. However, the increase in number of variables and the introduction of some extra connection and minimisation rules made this method not as efficient as the TDD or ZBDD method.

Overall, as expected, the efficiency of the four methods was marginal when small fault trees were considered. While analysing 'large' fault trees the advantages and disadvantages of the different techniques were more considerable. The TDD method and the ZBDD method performed a lot better than the two other techniques and should be used for the BDD analysis of non-coherent fault trees. The meta-products BDD method performed a lot worse than the L-BDD method. The difference between the two advantageous methods, the TDD method and the ZBDD method, was still marginal. Therefore, any of the two methods used for a different library of example fault trees should provide an efficient analysis.

The eight ordering schemes were ranked for the four conversion methods. There are some clear conclusions about the efficiency of the schemes. Using the example fault tree library provided the dynamic top-down weighted scheme (6) performed

well for all the methods and both measurements. The depth-first, with number of leaves scheme (4) was also ranked highly. The modified top-down scheme (1) gave average or poor results, especially according to the number of nodes. The two depth-first schemes, (2) and (3), also performed poorly. Summarising the ranking, for the majority of the methods the weighted ordering schemes, (5)-(8), performed better than the neighbourhood ordering schemes, (1)-(4).

In some cases the fault tree to BDD conversion was impossible in a reasonable time, therefore, example fault trees were simplified prior to the BDD conversion process. In that case some complex and modular events appeared in the BDDs. While performing the qualitative and quantitative analyses these events were expanded back to the level of basic events in order to be able to obtain the results in terms of original components. Overall, this strategy provided is an efficient tool for the analysis of non-coherent fault trees.

The application of the Hybrid method in the analysis of non-coherent fault trees was also performed. The Hybrid method was used for building SFBDDs. This approach resulted in some further improvement on the efficiency of the method in terms of the processing time and maximum size of array required.

Chapter 7

Application of Proposed Methods in Phased Mission Analysis

7.1 Introduction

Many types of system operate for missions that are made up of several phases. For example, an aircraft mission could be considered in the following phases: taxiing to the runway, take-off, climbing to a certain altitude, cruising, descending, landing and taxiing back to the terminal. The system must operate successfully during each phase so that the mission would be completed successfully. Components can fail at any point during the mission but it can be critical only for one particular phase. Therefore, it may be that the transition from one phase to another is the critical event causing mission failure and the component failures may have occurred previously.

FTA can be used as a means for analysing the reliability of non-repairable systems that undergo phased missions. The required results are the system failure modes and failure probability of each phase, followed by the total mission unreliability. The complexity of fault trees might make the analysis impossible, therefore, some alternative methods will be incorporated in the calculation process. First of all, non-coherent fault trees representing phase failures will be simplified. Then the BDD method will be employed to calculate the unreliability of each phase. This will allow a more efficient and accurate Phased Mission Analysis.

7.2 Fault Tree Method for Phased Mission Analysis

A very simple phased mission problem consisting of non-repairable components A, B, C, D and E representing component failures in each of the phases is used to demonstrate the approach. Example is shown in Figure 7.1. During phase 1,

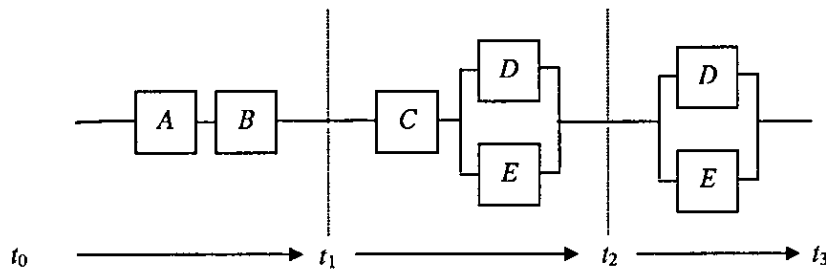


Figure 7.1 Example of a simple phased mission system

which lasts until time t_1 , the success of the mission depends on the success of the two components, A, B . After the successful completion of phase 1 the system then enters phase 2, which requires component C to function between times t_1 and t_2 along with at least one of the two remaining components D and E . In the final phase only one of the two components D and E is required to function between times t_2 and t_3 in order to accomplish the mission successfully.

Considering the phases as separate systems individual phase failures are presented by fault trees in Figure 7.2. Component failure in phase i is A_i, B_i, C_i, D_i and E_i

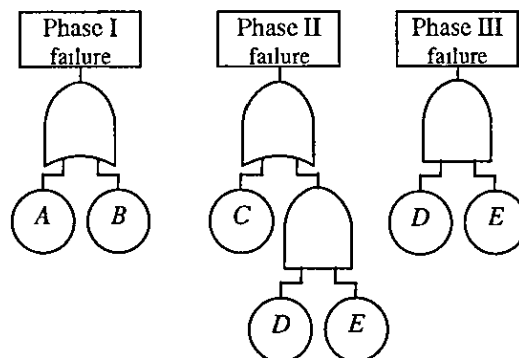


Figure 7.2 Fault trees of individual phase failures

for components A, B, C, D and E respectively.

The reliability of a phased mission cannot be obtained simply by the multiplication of the reliabilities of each of the individual phases because it would involve assumptions that the phases are independent and all components are working at the beginning of each phase. One of the methods proposed in [38] contains the transformation of a mission to that of an equivalent single-phase mission. This method involves three stages and is only concerned with the failure of the mission, i.e. it does not take into account failures of the individual phases. It is impossible to calculate individual phase failure probabilities.

An alternative method was proposed by La Band and Andrews in [39] that enhances the fault tree approach but also enables the probability of failure in each phase to be determined in addition to the whole mission reliability. For every phase the method combines the causes of success in previous phases with the causes of failure for the phase being considered. This allows both qualitative and quantitative analyses of both phase failure and mission failure.

System failure in phase i is presented by the AND of the success of phases 1 to $i - 1$ and the failure during the phase i , as shown in Figure 7.3. The mission unreliability, Q_{miss} , is then obtained as a sum of failure probabilities in phase i , Q_i :

$$Q_{miss} = \sum_{i=1}^N Q_i, \quad (7.1)$$

here N is the total number of phases

For the example shown in Figure 7.1 the fault tree to show the initial phase failure remains identical to the fault tree representation of the individual phase failure of phase I in Figure 7.2. Phase II failure can then be shown as the combination of phase I success and failure in phase II, as shown in Figure 7.4. Also, every basic event in the phase II fault tree is replaced by an OR combination of the failure events for that and all preceding phases. In the same way phase III failure can be represented as the combination of phase I and phase II successes and failure in phase III, presented in Figure 7.5.

To determine the minimal cut sets of a phase or a mission a top-down or a bottom-up approach is applied to the relevant fault tree. The notation used to represent

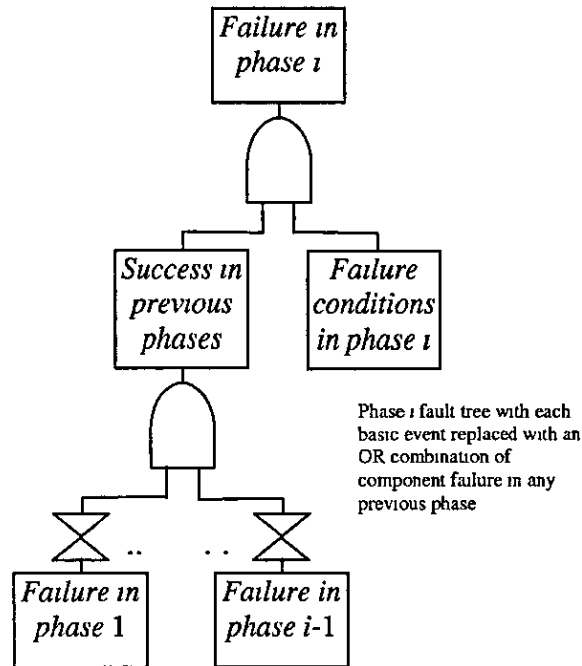


Figure 7.3. Phase failure fault tree in a general case

the failure of component A in phase i is A_i ; \bar{A}_i represents the functioning of component A throughout phase i . The notation used to indicate the failure of component in phase i to j is $A_{i,j}$, i.e. the component fails at some point from the start of phase i to the end of phase j . This notation defines an algebra over the phases to manipulate the logic equations

$$A_i \cdot A_i = A_i, \quad (7.2)$$

$$A_i \cdot A_j = 0, \quad (7.3)$$

$$A_i \cdot A_{i,j} = A_i, \quad (7.4)$$

$$\bar{A}_i \cdot A_i = 0, \quad (7.5)$$

$$\bar{A}_i \cdot A_{i,j} = A_{i+1,j}, \quad (7.6)$$

$$\bar{A}_i \bar{A}_{i+1} \bar{A}_j = \bar{A}_{i,j}, \quad (7.7)$$

$$A_i + A_{i+1} \cdot A_j = A_{i,j} \quad (7.8)$$

Therefore, if two implicant sets contain exactly the same components, where all but one occur over the same time intervals and the other is a failure in contiguous phases, the two implicant sets may be combined with the period of failure for the component with time discrepancy adjusted, e.g. two implicant sets $A_1 B_1$ and $A_1 B_2$ can be replaced by $A_1 B_{1,2}$. This simplification approach allows the prime

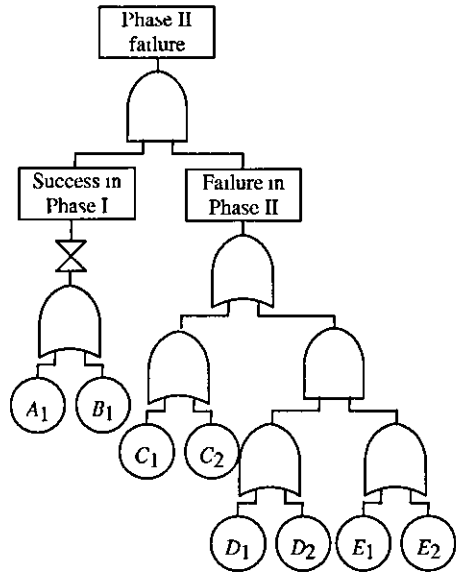


Figure 7.4. Phase II failure fault tree

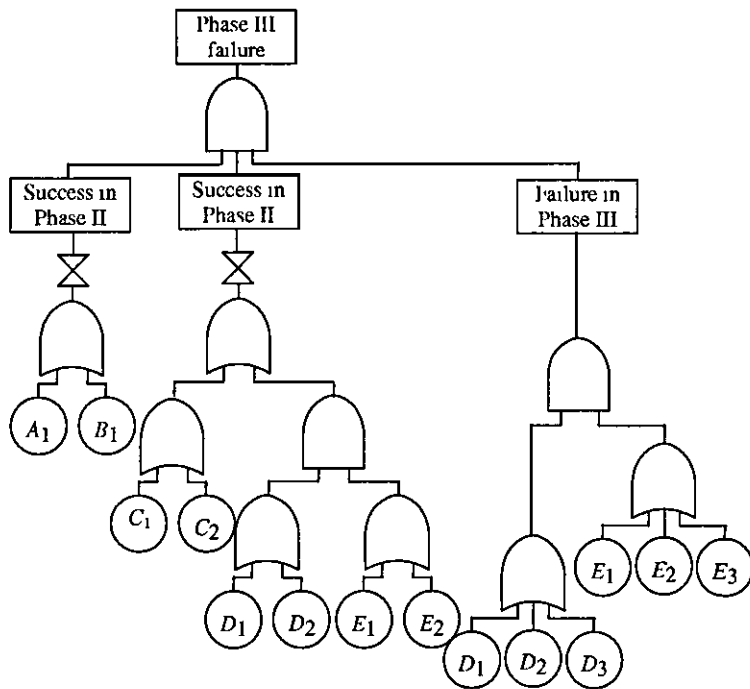


Figure 7.5: Phase III failure fault tree

implicants for the simple example given in Figure 7.1 to be expressed as follows:

$$\text{Phase I:} \tag{7.9}$$

$$T_1 = A_1 + B_1$$

This gives two minimal cut sets $\{A_1\}$ and $\{B_1\}$

Phase II: (7.10)

$$\begin{aligned} T_2 &= \bar{A}_1\bar{B}_1(C_1 + C_2 + (D_1 + D_2)(E_1 + E_2)) \\ &= \bar{A}_1\bar{B}_1C_{1,2} + \bar{A}_1\bar{B}_1D_{1,2}E_{1,2}. \end{aligned}$$

Prime implicants obtained are $\bar{A}_1\bar{B}_1C_{1,2}$ and $\bar{A}_1\bar{B}_1D_{1,2}E_{1,2}$.

Finally,

Phase III: (7.11)

$$\begin{aligned} T_3 &= \bar{A}_1\bar{B}_1\bar{C}_1\bar{C}_2(\bar{D}_1\bar{D}_2 + \bar{E}_1\bar{E}_2)(D_1 + D_2 + D_3)(E_1 + E_2 + E_3) \\ &= \bar{A}_1\bar{B}_1\bar{C}_1\bar{C}_2D_3E_{1,3} + \bar{A}_1\bar{B}_1\bar{C}_1\bar{C}_2E_3D_{1,3}. \end{aligned}$$

This gives two prime implicant sets $\bar{A}_1\bar{B}_1\bar{C}_{12}D_3E_{13}$ and $\bar{A}_1\bar{B}_1\bar{C}_{12}E_3D_{13}$

Having established the prime implicant sets for each phase, they may now be used to quantify the probability of phase and mission failure. Using the inclusion-exclusion expansion in Equation 2.41 the event of phase failure for this simple three-phase mission is expressed as:

$$Q_1 = q_{A_1} + q_{B_1} - q_{A_1}q_{B_1}, \quad (7.12)$$

$$\begin{aligned} Q_2 &= (1 - q_{A_1})(1 - q_{B_1})q_{C_{1,2}} + (1 - q_{A_1})(1 - q_{B_1})q_{D_{1,2}}q_{E_{1,2}} - \\ &\quad (1 - q_{A_1})(1 - q_{B_1})q_{C_{1,2}}q_{D_{1,2}}q_{E_{1,2}}, \end{aligned} \quad (7.13)$$

$$\begin{aligned} Q_3 &= (1 - q_{A_1})(1 - q_{B_1})(1 - q_{C_1} - q_{C_2})q_{D_3}q_{E_{1,3}} + \\ &\quad (1 - q_{A_1})(1 - q_{B_1})(1 - q_{C_1} - q_{C_2})q_{E_3}q_{D_{1,3}} - \\ &\quad (1 - q_{A_1})(1 - q_{B_1})(1 - q_{C_1} - q_{C_2})q_{D_3}q_{E_3} \end{aligned} \quad (7.14)$$

As the failure of each of the phases produces mutually exclusive causes, the probability of mission failure may be expressed as the sum of the unreliability of the individual phases:

$$Q_{MISS} = \sum_{i=1}^N Q_i \quad (7.15)$$

7.3 Phase FT simplification

Fault tree simplification techniques are helpful to reduce the size of a fault tree to enable prime implicant sets to be found more efficiently. They were presented in

Chapter 2 for coherent fault trees and in Chapter 5 for non-coherent fault trees. These techniques reduce both memory and time requirements. The factorisation of phase fault trees is a part of this research and will be explained more in detail.

Fault trees of every individual phase are considered

Rule. *If there is a pair of basic events that always occur together and under the same gate type, throughout the set of fault trees of every individual phase, that pair can be replaced by a single complex event.*

Introduce a phased mission with N phases as shown in Figure 7.6. Since $A \text{ AND } B$

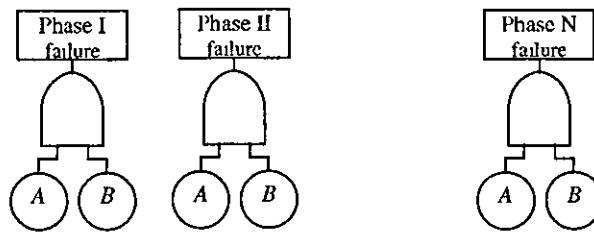


Figure 7.6: N-phase mission

always occur together they can be factorised, i.e. $A \text{ AND } B = 2000$. This example contains a set of AND gates, however, the process of simplification to OR gates can be applied in the same way. The factorised fault trees are shown in Figure 7.7.

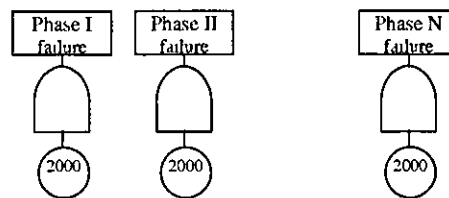


Figure 7.7 Factorised fault trees

Following the method, presented in the previous section, OR combinations of event failures in each phase replace every basic event in a FT. Also, the success of every preceding phase is included in the phase failure logic. Fault trees representing failures for phase I, phase II and phase III are shown in Figure 7.8. After the absorption has been applied on the fault trees, prime implicant sets can be calculated from fault trees in Figure 7.9

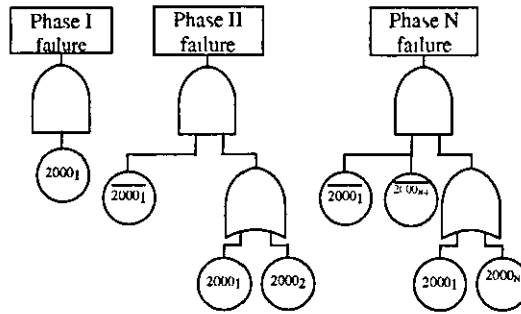


Figure 7.8: Phase I, phase II and phase III failure fault trees

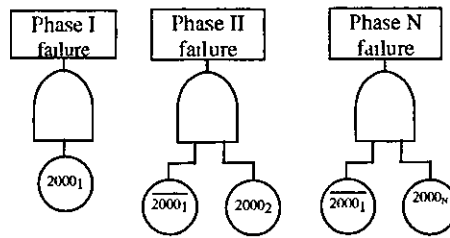


Figure 7.9 Fault trees after the absorption

$$\text{Phase I: } T_1 = 2000_1, \quad (7.16)$$

$$\text{Phase II: } T_2 = \overline{2000_1} 2000_2,$$

.

$$\text{Phase N: } T_N = \overline{2000_1} 2000_N$$

This gives prime implicant sets:

$$\text{Phase I : } 2000_1, \quad (7.17)$$

$$\text{Phase II : } 2000_2,$$

..

$$\text{Phase N : } 2000_N.$$

While extracting prime implicant sets in terms of basic events, two types of gate are considered. For an AND gate, i.e. $2000 = A \cdot B$

$$\text{Phase I : } A_1 B_1, \quad (7.18)$$

$$\text{Phase II : } A_2 B_{1,2} + B_2 A_{1,2},$$

.

$$\text{Phase N : } A_N B_{1,N} + B_N A_{1,N}.$$

For an OR gate, i e $2000 = A + B$:

$$\begin{aligned}
 \text{Phase I} & \quad A_1 + B_1, & (7.19) \\
 \text{Phase II} & \quad \bar{B}_1 A_2 + \bar{A}_1 B_2, \\
 & \quad \dots \\
 \text{Phase N} & \quad \bar{B}_{1,N-1} A_N + \bar{A}_{1,N-1} B_N.
 \end{aligned}$$

Also, while calculating the unreliability of each phase two types of gate will be considered. For an AND gate.

$$\begin{aligned}
 Q_1 & = q_{2000_1} = q_{A_1} q_{B_1}, & (7.20) \\
 Q_2 & = q_{2000_2} = q_{A_2} q_{B_{1,2}} + q_{B_2} q_{A_{1,2}} - q_{A_2} q_{B_2}, \\
 Q_N & = q_{2000_N} = q_{A_N} q_{B_{1,N}} + q_{B_N} q_{A_{1,N}} - q_{A_N} q_{B_N}
 \end{aligned}$$

For an OR gate:

$$\begin{aligned}
 Q_1 & = q_{2000_1} = q_{A_1} + q_{B_1} - q_{A_1} q_{B_1}, & (7.21) \\
 Q_2 & = q_{2000_2} = (1 - q_{B_1}) q_{A_2} + (1 - q_{A_1}) q_{B_2} - q_{A_2} q_{B_2}, \\
 & \quad \dots \\
 Q_N & = q_{2000_N} = (1 - q_{B_1} - \dots - q_{B_{N-1}}) q_{A_N} + \\
 & \quad (1 - q_{A_1} - \dots - q_{A_{N-1}}) q_{B_N} - q_{A_N} q_{B_N}.
 \end{aligned}$$

Remark.

For an AND gate:

$$\begin{aligned}
 2000_{1N} & = 2000_1 + 2000_2 + \dots + 2000_N & (7.22) \\
 & = A_1 B_1 + \dots + A_N B_{1,N} + B_N A_{1,N} = A_{1N} B_{1,N}
 \end{aligned}$$

For an OR gate:

$$\begin{aligned}
 2000_{1N} & = 2000_1 + 2000_2 + \dots + 2000_N & (7.23) \\
 & = A_1 + B_1 + \dots + \bar{B}_{1,N} A_N + \bar{A}_{1,N} B_N.
 \end{aligned}$$

This simplification technique will be applied using the example mission in Figure 7.1

First of all, the factorisation of fault trees shown in Figure 7.2 is performed resulting in the fault trees shown in Figure 7.10. Events D AND E are replaced by

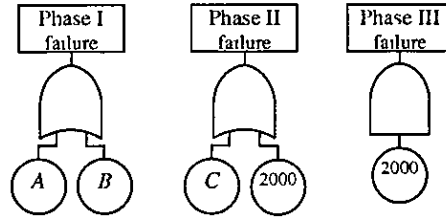


Figure 7.10: Factorised example fault trees

a complex event 2000.

Now fault trees for every phase are constructed using the method explained before. Phase I fault tree remains identical to the fault tree representation of the individual phase failure of phase I in Figure 7.10. Phase II and phase III fault trees are presented in Figure 7.11 and Figure 7.12 respectively.

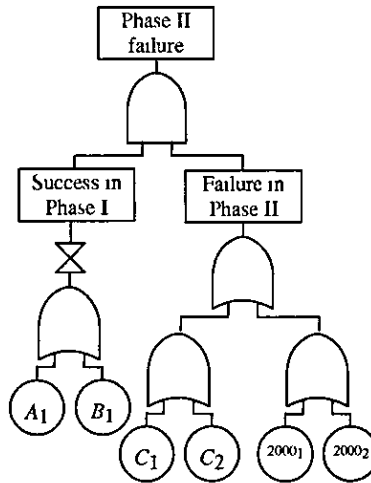


Figure 7.11 Phase II failure fault tree from the example

Now prime implicant sets can be calculated. Phase I failure fault tree is identical to the individual phase failure fault tree, therefore:

$$\text{Phase I:} \tag{7.24}$$

$$T_1 = A_1 + B_1$$

Now for phase II the rules for an AND gate in Equation 7.18 are followed.

$$\text{Phase II:} \tag{7.25}$$

$$\begin{aligned} T_2 &= \bar{A}_1 \bar{B}_1 (C_1 + C_2 + 2000_1 + 2000_2) \\ &= \bar{A}_1 \bar{B}_1 (C_{1,2} + 2000_{1,2}) \\ &= \bar{A}_1 \bar{B}_1 (C_{1,2} + D_{1,2} E_{1,2}) \end{aligned}$$

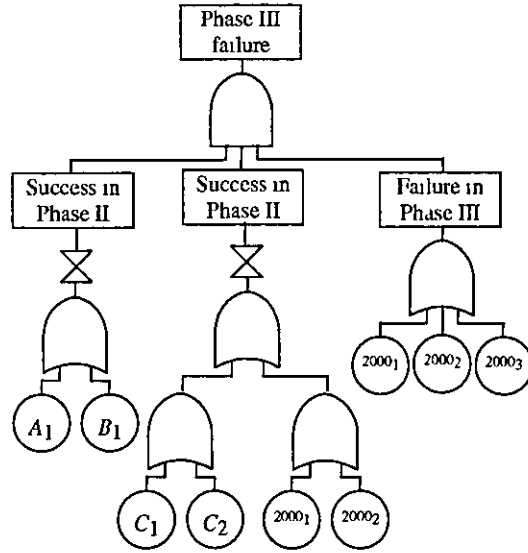


Figure 7.12: Phase III failure fault tree from the example

Therefore, the two prime implicant sets obtained are $\bar{A}_1\bar{B}_1C_{1,2}$ and $\bar{A}_1\bar{B}_1D_{1,2}E_{1,2}$. For phase III Equation 7.18 is used once again.

Phase III (7 26)

$$\begin{aligned}
 T_2 &= \bar{A}_1\bar{B}_1\bar{C}_1\bar{C}_2\overline{2000_1}\overline{2000_2}(2000_1 + 2000_2 + 2000_3) \\
 &= \bar{A}_1\bar{B}_1\bar{C}_1\bar{C}_22000_3 \\
 &= \bar{A}_1\bar{B}_1\bar{C}_1\bar{C}_2(D_3E_{1,3} + E_3D_{1,3})
 \end{aligned}$$

Two prime implicant sets obtained are $\bar{A}_1\bar{B}_1\bar{C}_1\bar{C}_2D_3E_{1,3}$ and $\bar{A}_1\bar{B}_1\bar{C}_1\bar{C}_2E_3D_{1,3}$. Prime implicant sets match the ones obtained from the original fault trees, therefore, the unreliability of each phase will also have the same expression.

For phase I:

$$Q_1 = q_{A_1} + q_{B_1} - q_{A_1}q_{B_1}. \quad (7 27)$$

For phase II:

$$\begin{aligned}
 Q_2 &= (1 - q_{A_1})(1 - q_{B_1})q_{C_{1,2}} \\
 &\quad + (1 - q_{A_1})(1 - q_{B_1})q_{2000_{1,2}} \\
 &\quad - (1 - q_{A_1})(1 - q_{B_1})q_{C_{1,2}}q_{2000_{1,2}} \\
 &= (1 - q_{A_1})(1 - q_{B_1})q_{C_{1,2}} \\
 &\quad + (1 - q_{A_1})(1 - q_{B_1})q_{D_{1,2}}q_{E_{1,2}} \\
 &\quad - (1 - q_{A_1})(1 - q_{B_1})q_{C_{1,2}}q_{D_{1,2}}q_{E_{1,2}}
 \end{aligned} \quad (7.28)$$

For phase III.

$$\begin{aligned}
 Q_3 &= (1 - q_{A_1})(1 - q_{B_1})(1 - q_{C_1} - q_{C_2})q_{2000_3} & (7.29) \\
 &= (1 - q_{A_1})(1 - q_{B_1})(1 - q_{C_1} - q_{C_2}) \\
 &\quad (q_{D_3}q_{E_{1,3}} + q_{E_3}q_{D_{1,3}} - q_{E_3}q_{D_3}).
 \end{aligned}$$

This concludes the calculation of mission unreliability using the simplified phase failure fault trees.

7.4 Binary Decision Diagram Analysis for Phased Missions

A fault tree represents system failure logic efficiently but is not ideal for mathematical analysis. Therefore, the BDD method can be applied that provides an accurate and efficient analysis. Especially this will be particularly useful for complex fault trees or fault trees that are non-coherent, such as the phase failure fault trees.

The representation of the BDD method is explained in Chapter 3. This method will be applied to the simple three-phase mission illustrated in Figure 7.2, where each phase can be represented by a SFBDD and then the unreliability of each phase can be calculated. The SFBDD represents the structure function of the non-coherent fault tree, as it was explained in Chapter 6, and it can be applied for the quantitative analysis but not for the qualitative analysis. So, the fault trees for phase I, phase II and phase III in Figure 7.10, 7.11 and 7.12 are first converted to SFBDDs. These SFBDDs are shown in Figures 7.13 and 7.14. Now their quantitative analysis can be performed.

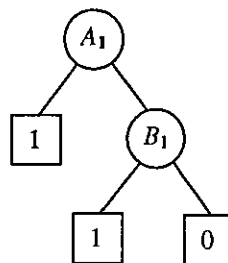


Figure 7.13: SFBDD representing phase I failure

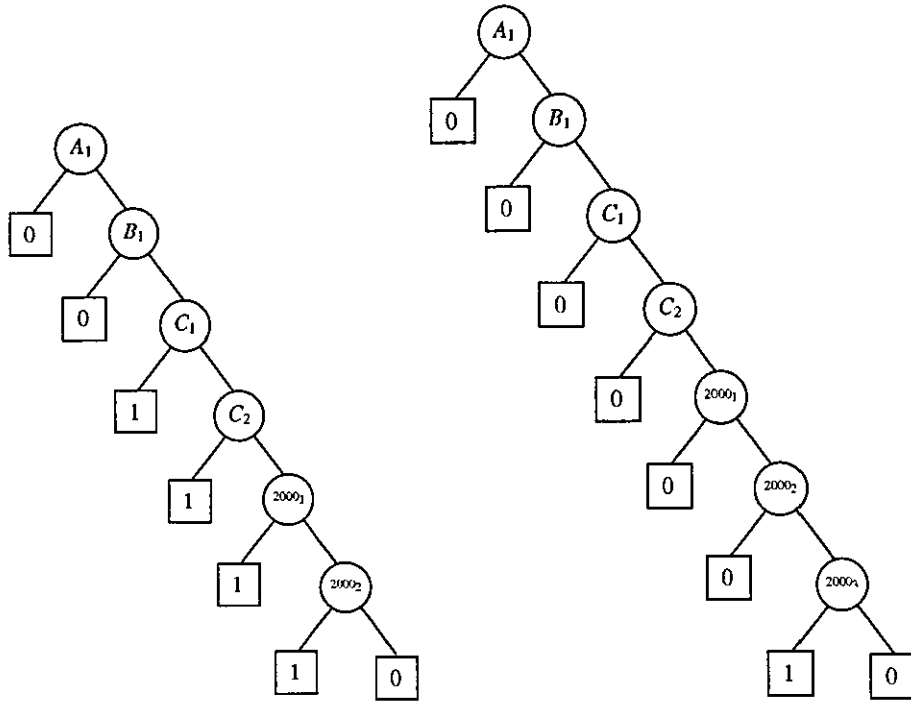


Figure 7.14: SFBDDs representing phase II and phase III failures

For phase I, the *ite* structure represented by the SFBDD in Figure 7.13 is.

$$\text{ite}(A_1, 1, \text{ite}(B_1, 1, 0)) \quad (7.30)$$

There are two disjoint paths to terminal 1, A_1 and $\overline{A_1}B_1$. Therefore

$$Q(t) = q_{A_1} + p_{A_1}q_{B_1}. \quad (7.31)$$

The quantification process is equivalent to the method, presented in Chapter 3. The SFBDD for failure during phase II is given by the following *ite* structure

$$\text{ite}(A_1, 0, \text{ite}(B_1, 0, \text{ite}(C_1, 1, \text{ite}(C_2, 1, \text{ite}(2000_1, 1, \text{ite}(2000_2, 1, 0)))))) \quad (7.32)$$

There are four paths to terminal 1:

$$\begin{aligned} &\overline{A_1}\overline{B_1}C_1, \\ &\overline{A_1}\overline{B_1}\overline{C_1}C_2, \\ &\overline{A_1}\overline{B_1}\overline{C_1}\overline{C_2}2000_1, \\ &\overline{A_1}\overline{B_1}\overline{C_1}\overline{C_2}\overline{2000_1}2000_2. \end{aligned} \quad (7.33)$$

During quantification the fact, that the events representing the same component failure in different phases, like C_1 and C_2 , are mutually exclusive, needs to be taken

into account. If component C fails in phase II it must have survived phase I. This rule is covered by Equation 7.6. Therefore, phase II failure probability is calculated as

$$Q_2 = (1 - q_{A_1})(1 - q_{B_1})(q_{C_1} + q_{C_2}) + (1 - q_{A_1})(1 - q_{B_1})(1 - q_{C_1} - q_{C_2})(q_{2000_1} + q_{2000_2}). \quad (7.34)$$

Since $2000_1 + 2000_2 = D_{1,2}E_{1,2}$ according to Equation 7.22 it gives

$$Q_2 = (1 - q_{A_1})(1 - q_{B_1})(q_{C_{1,2}} + (1 - q_{C_{1,2}})q_{D_{1,2}}q_{E_{1,2}}) = (1 - q_{A_1})(1 - q_{B_1})(q_{C_{1,2}} + q_{D_{1,2}}q_{E_{1,2}} - q_{C_{1,2}}q_{D_{1,2}}q_{E_{1,2}}). \quad (7.35)$$

Finally, the SFBDD representation for the fault tree of the failure during phase III is

$$\text{ite}(A_1, 0, \text{ite}(B_1, 0, \text{ite}(C_1, 0, \text{ite}(C_2, 0, t)))), \quad (7.36)$$

where

$$t = \text{ite}(2000_1, 0, \text{ite}(2000_2, 0, \text{ite}(2000_3, 1, 0))).$$

The only path is

$$\overline{A_1} \overline{B_1} \overline{C_1} \overline{C_2} \overline{2000_1} \overline{2000_2} 2000_3. \quad (7.37)$$

The failure probability of phase III is expressed as

$$Q_3 = (1 - q_{A_1})(1 - q_{B_1})(1 - q_{C_1} - q_{C_2})q_{2000_3}. \quad (7.38)$$

Following the expression in Equation 7.20 gives

$$Q_3 = (1 - q_{A_1})(1 - q_{B_1})(1 - q_{C_1} - q_{C_2})(q_{D_3}q_{E_{1,3}} + q_{E_3}q_{D_{1,3}} - q_{E_3}q_{D_3}) \quad (7.39)$$

Therefore, it can be seen that the unreliability of each of the phases found by the BDD method is identical to that obtained using the fault tree analysis of original or simplified fault trees.

The quantitative analysis of the phase failure probability using BDDs is efficient, because the paths to terminal vertices are disjoint and the phase failure probability can be calculated by summing the probabilities of each path. This approach also shows that the size of phase fault trees can be reduced by applying the simplification process. Therefore, the analysis becomes even more efficient because the size of BDDs is also reduced since a smaller number of variables appears in the fault tree after the simplification.

7.5 Summary

The accurate assessment of mission unreliability for systems with non-repairable components operating over a sequence of phases can be performed using non-coherent fault tree structures. This provides a full description of the performance of the system and allows the calculation of not only the failure probability for the whole mission but also for every phase. Applying algebraic rules the prime implicant sets are obtained and used in the inclusion-exclusion expression for the phase failure probability.

Since the direct quantification of the fault trees is frequently problematic even for moderately sized problems, fault trees can be reduced before calculating prime implicants. Fault trees for individual phases can be reduced and then fault trees of every phase failure are built. New rules for expressing prime implicant sets and calculating phase failure probability were considered in this chapter when fault trees were factorised beforehand.

As a further extension, the use of the BDD method to calculate the failure probability of each phase in the mission provides an efficient and accurate means of evaluating mission unreliability. The prime implicant sets are not required for the quantitative analysis and this property makes the BDD method efficient. This method incorporates building BDDs for both coherent and non-coherent fault trees, tracing all disjoint paths and estimating phase failure probability followed by mission failure probability.

Chapter 8

Computer Implementation of BDD Method

8.1 Introduction

A system failure modelling and analysis tool based on the Binary Decision Diagram analysis is developed. This program integrates the fault tree simplification process, presented in Chapter 2, and the BDD conversion techniques, explained in Chapters 3, 4, 5, prior to the qualitative and quantitative analyses. The system failure analysis software runs on personal computers running Microsoft's Windows 95, 98 and NT operating systems. The tool enables users to edit and display fault trees in textual form. The type of the analysis can be selected from menu options.

8.2 Overview

This tool is a procedure for performing system failure analysis based on the Binary Decision Diagram method. The facility includes:

1. System data input in a fault tree format
2. Component failure data input
3. Conversion of fault tree to a numerical form
4. Simplification pre-processing of fault trees
5. Fault tree conversion to a binary decision diagram
6. Identification of minimal cut sets/prime implicant sets using the BDD

7. Calculation of probability and frequency of top event occurrence from the BDD
8. Importance measures facility
9. Tabular presentation of results

The process begins with a particular failure mode of an industrial system which is represented by a fault tree. The fault tree is then converted to its numerical form and then simplified, if requested, identifying independent modules and simplifying the logic function. After that the fault tree is converted to a BDD and the required analysis performed, identifying minimal cut sets or prime implicant sets of component conditions that cause system failure. The probability of the top event occurrence is calculated together with the frequency of the top event. Importance measures which rank the contribution of every component are also calculated. If the system being analysed is subdivided into modules the program assesses each module in turn and then combines the module information to obtain results for the overall system. The flowchart of the program is shown in Figure 8.1.

8.3 Established Modules

The main modules of the program are presented in the following sections explaining the data format and highlighting the important parts of the algorithm

8.3.1 Data Input Module

A fault tree provides a structured description of causes of a particular system failure mode. Data describing the fault tree structure is specified such that each line in the data file provides details for each gate. The line starts with the name and the type of a gate, followed by the number of gate inputs, the number of event inputs and concludes with a list of its gate and event inputs. The top gate can appear on any line of the data file. During the input the data is converted to a numerical representation for ease of manipulation. Every gate is assigned a unique number from 10000 upwards (not more than 19999) and every event is assigned a unique number from 1 upwards (not more than 9999). An 'OR' gate is coded by 1 and an 'AND' gate is coded by 2. Complex events that are created during the reduction process are described starting with a number from 20000 upwards. Modular events that are created during the modularisation process are assigned a

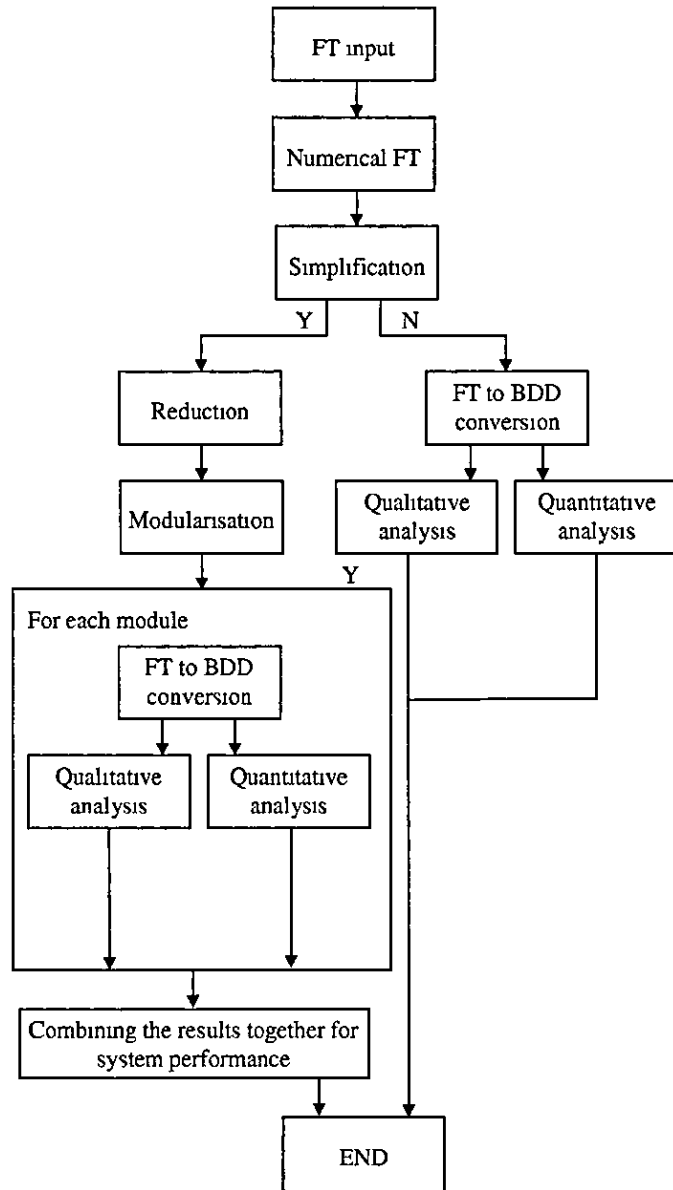


Figure 8.1: Flowchart of the developed program

number 20000 + *namenum3* upwards, here *namenum3* is the number of complex events in the fault tree. Arrays are used to represent a fault tree. Figure 8.2 shows the way the data input scheme works. On the left there is an example fault tree with its equivalent data file on the top right, followed by the list of events. The numerical representation of the fault tree is shown on the bottom right. This numerical data is manipulated further during the analysis.

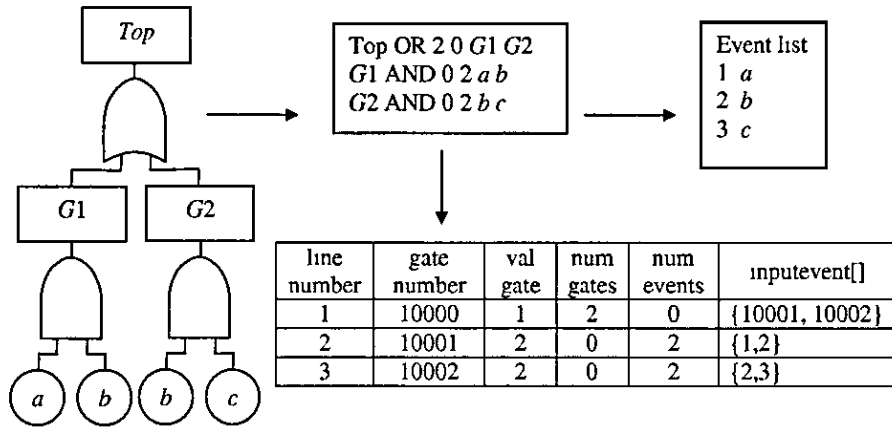


Figure 8.2 Data input scheme

8.3.2 Simplification Module

This part of the program provides the fault tree simplification tool applying both the reduction and the linear modularisation, presented in Chapter 2.

First of all, the numerical array that represents the input fault tree is scanned from the beginning and all the four reduction phases are applied by factorising pairs of events that always occur together and removing redundancies of repeated events by performing extraction and absorption procedures. The numerical fault tree is scanned until there are no more changes possible. A reduced numerical fault tree is obtained together with an array that describes the complex events constructed.

The linear modularisation algorithm is applied by traversing the fault tree twice - the first time every basic event and gate are assigned a visit number, the second time independent modules are identified according to the visit numbers that were assigned. The set of modules is stored in a numerical format together with the in-

formation identifying the top event of every module. This data is used performing the analyses later in the process.

8.3.3 BDD Conversion Module

A fault tree (or a set of fault trees if independent modules were obtained) is converted to a BDD. A two-dimensional array is used to store the BDD representation. Every node in a BDD is presented by a line of three entries in a BDD array. The first element represents the numerical code for the basic event associated with the node. The second entry contains a pointer to the number of the line where the BDD structure on the 1 branch of the node starts. Consequently, the third entry of every line in the BDD array identifies a pointer to the number of the line that marks the start of the BDD structure on the 0 branch.

Applying the conventional *ite* method the fault tree is traversed and basic elements are ordered according to a selected variable ordering scheme from the ones presented in Chapter 4. Then the fault tree needs to undergo pre-processing tasks where every gate is reconstructed allowing only two inputs. In this manner new gates are created. Then all basic events are replaced by their number of the position in the variable ordering. This allows the connection rules of the *ite* technique to be used in a more convenient way. An example fault tree is shown in Figure 8.3. The variable ordering is assigned $b < a < c$, in the numerical format it is described

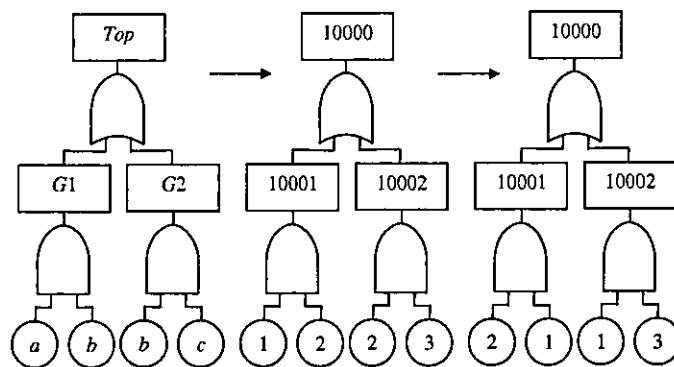


Figure 8.3: Example fault tree for the *ite* method

as $2 < 1 < 3$. First of all, every basic event is assigned a line in the BDD array with two terminal vertices, i.e. -1 for the second entry (1 branch) and 0 for the third entry (0 branch). -1 is used to represent the terminal 1 vertex so that the ambiguity between the reference to line 1 in the array and the terminal 1 vertex is

avoided. This representation is shown in Table 8.1. Then the numerical fault tree

Line number	Node	1 branch	0 branch
1	1	-1	0
2	2	-1	0
3	3	-1	0
4	1	2	0
5	1	3	0
6	1	7	0
7	2	-1	3

Table 8.1: *ite* array

is traversed converting every gate to its representing BDD and recording the fact that the conversion process of the gate has been completed

Gates with event inputs only are considered first. Considering gate $G1$ which has two basic events as its inputs, new line 4 in the *ite* array is created. It encodes an *ite* structure for gate $G1$, $\text{ite}(1, \text{ite}(2,-1,0),0)$. Therefore, line 4 contains 1 as the first entry, since $1 < 2$ in the variable ordering. The second entry is 2, i.e. pointing to the second line in the *ite* array that represents basic event 2, and the third entry is 0 because it points to terminal vertex 0. The sub-node sharing property is applied using the existing line 2 in the *ite* array for the $\text{ite}(2,-1,0)$.

At the same time a computational table is generated where every line represents the first and the second input of the gate together with the type of the gate and the resulting line of the root node in the *ite* array, presented in Table 8.2. Therefore,

Line number	Type of gate	First input	Second input	Resulting line
1	2	2	1	4
2	2	1	3	5
3	1	4	5	6

Table 8.2 Computational table in the *ite* method

for gate $G1$ the first entry in the computational table is constructed. The gate type is 2, the first input is from line 2, the second input is from line 1 and resulting *ite* structure is placed in line 4 of the *ite* array. This entry can be used further in the

conversion process, i.e. if the same two inputs, i.e. input 2 and input 1, appear elsewhere in the fault tree again as inputs to a gate of type 2 the resulting conversion of the gate can be referenced immediately reusing the *ite* structure obtained earlier

New line 5 in the BDD is created for gate $G2$, $\text{ite}(1, \text{ite}(3,-1,0),0)$. Considering the top gate two new lines 6 and 7 are created that represent the *ite* for the top event, $\text{ite}(1, \text{ite}(2,-1,\text{ite}(3,-1,0)),0)$. Two more entries are made in the computational table. The number of the line representing the root node of the BDD is returned and the conversion process is concluded by replacing numbers in nodes with the basic events if required. Therefore, the BDD for this example is expressed as $\text{ite}(b, \text{ite}(a, 1, \text{ite}(c, 1, 0)), 0)$. This resulting BDD can be used to perform the qualitative and quantitative analyses described in the next sections

The BDD module is used in the research investigating the conversion techniques of FTs to BDDs considering different methods and ordering schemes. In the BDD module eight alternative ordering schemes for basic events can be used. An efficiency measure facility is also incorporated in the module that allows the calculation of the number of repeated and non-repeated nodes in the *ite* array.

8.3.4 Quantitative Analysis Module

This part of the program performs the quantification of the system where the failure mode is represented by the BDD. This module performs the calculation of the probability and the frequency of top event occurrence and the computation of Birnbaum's measure of importance for every basic event in the system. The detailed explanation of the application is presented in Chapter 3. The probability of occurrence of the top event is expressed as the sum of the probabilities of the disjoint paths through the BDD. All the disjoint paths can be found by tracing all paths from the root vertex to terminal 1 vertices. This calculation process is performed in a bottom-up manner. Nodes with terminal vertices on both their branches are considered first. Once the probability of a particular part of the BDD is calculated the process does not need to be repeated again. The criticality of every basic event is calculated during the same traversal of the BDD by recording path probabilities

8.3.5 Qualitative Analysis Module

This module consists of the minimisation algorithm of the BDD and the procedures to calculate the minimal cut sets (or prime implicant sets).

8.3.5.1 Minimisation module

In the application of the minimisation algorithm every path is analysed in a bottom-up manner. Nodes with terminal vertices on both of their branches are considered first. First of all, the structure beneath the 1 branch of the node is minimised. Then the 'without' operator is applied removing the paths from the structure on the 1 branch that are repeated in the structure on the 0 branch. Finally, the structure beneath the 0 branch is minimised. Each of those three steps can produce new entries in the BDD array that replace existent lines. The minimisation result of every node, i.e. a new line that will replace the structure after the minimisation, is stored in the array of minimal solutions and can be reused if required. Also, the information generated when applying the 'without' operator is stored in the 'without' array and can be reused during the application of the 'without' operator.

For example, consider the *ite* structure $\text{ite}(1, \text{ite}(2, \text{ite}(3,1,0), \text{ite}(4,1,0)), \text{ite}(3,1,0))$ as shown in Figure 8.4. The BDD is traversed in a bottom-up way. The minimi-

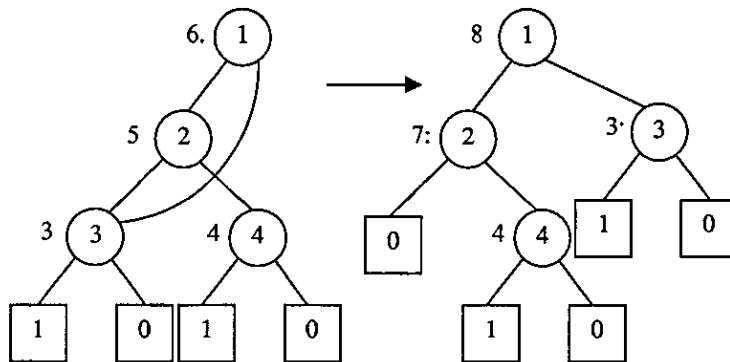


Figure 8.4: Example BDD before and after the minimisation

sation process is explained in Figure 8.5 where every step of the algorithm and the resulting line in the *ite* array are traced recursively.

I step - The minimal solution of $\text{ite}(3,1,0)$ is obtained and stored in the array of minimal solutions, line 3 in the Table 8.3.

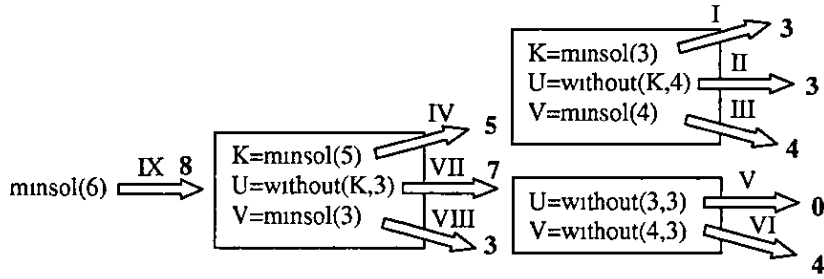


Figure 8 5: Application of the minimisation process

Line number	Min solution
1	-
2	-
3	3
4	4
5	5
6	8

Table 8 3: Minimal solutions table

II step - The 'without(3,4)' operator is applied. Since there are no paths in the structure described by line 3 ($\text{ite}(3,1,0)$) that are repeated in the structure described by line 4 ($\text{ite}(4,1,0)$), the 'without' operator returns the value 3 ($\text{ite}(3,1,0)$). The first line is assigned in the 'without' array, i.e. the first entry is 3 (the first parameter of 'without' operator), the second entry is 4 (the second parameter of 'without' operator) and the third entry is 3 (the returned value), shown in Table 8 4

Line number	F	G	"without" solution
1	3	4	3
2	4	3	4
3	5	3	7

Table 8.4: 'without' solutions table

III step - After that the minimal solution of $\text{ite}(4,1,0)$ is obtained and stored in the array of minimal solutions, line 4 in the Table 8 3.

IV step - Since the ite structure $\text{ite}(2, \text{ite}(3,1,0), \text{ite}(4,1,0))$ is minimal, the solution is kept in the array of minimal solutions, line 5 in the Table 8.3

V step - The 'without(3, 3)' operator removes the repeated path, i.e. line 3, from the structure replacing it by 0

VI step - The 'without(4, 3)' operator is applied. Since there are no repeated paths, the 'without' operator returns the value 4 ($\text{ite}(4,1,0)$). The second line is assigned in the 'without' array, shown in Table 8 4

VII step - While applying the 'without(5, 3)' operator, a new entry in the BDD array is created, line 7, shown in Table 8.5. Also a new entry, line 3, is created in the 'without' array as shown in Table 8 4.

Line number	Node	1 branch	0 branch
1	1	-1	0
2	2	-1	0
3	3	-1	0
4	4	-1	0
5	2	3	4
6	1	5	3
7	2	0	4
8	1	7	3

Table 8.5: ite table for minimisation

VIII step - During the minimisation of $\text{ite}(3,1,0)$ the value 3 (line 3) is reused from the array of minimal solutions

IX step - Finally, the last line in the BDD array is created that represents the root node of the minimised BDD Also, the last entry in the array of minimal solutions is produced as it is shown in Table 8 3

8.3.5.2 Calculation of minimal cut sets

During the calculation of minimal cut sets every path to a terminal 1 vertex in the minimised BDD is passed to collect basic events on 1 branches in a bottom-up way.

Consider the minimised BDD on the right in Figure 8 4. The process of obtaining minimal cut sets is shown in Figure 8.6

I step - The process starts from the root vertex traversing the first path to the terminal vertex 1. In this way the node $\text{ite}(4,1,0)$ that contains a terminal vertex

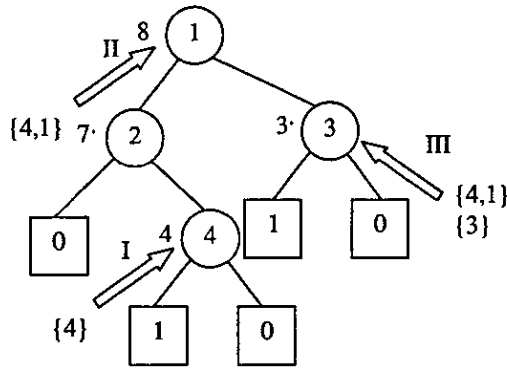


Figure 8 6 Application of minimal cut sets computation

is reached, therefore, the first minimal cut set containing one element is created and basic event 4 is included in it as shown in Table 8 6.

II step - Then following the path back up basic event 1 is also included in the first minimal cut set before analysing the 0 branch of the root vertex.

III step - The final minimal cut set is conducted, creating a new entry in the minimal cut sets array and allocating basic event 3 as shown in Table 8.6.

Row \ Column	1	2
1	4	1
2	3	

Table 8 6 Minimal cut sets array

In the event that a fault tree has been simplified before the qualitative analysis the resulting minimal cut sets will contain complex and/or modular events. It is essential to be able to analyse the system in terms of its original components, therefore, the next stage of the qualitative analysis is to extract the combinations of component failures from every complex and modular event. A key point of the expansion algorithm, which is the same as the MOCUS method [3] for calculating minimal cut sets from fault trees, is that an AND gate increases the number of basic events in each minimal cut set (or prime implicant set) and an OR gate increases the number of minimal cut sets (or prime implicant sets) in the system. The array is repeatedly scanned replacing

1. Each complex event which is an OR gate by a vertical expansion including

the input events to the gate (duplicating all other events in this row)

- 2 Each complex event which is an AND gate by a horizontal expansion including the input events to the gate
3. Each modular event by a vertical and/or horizontal expansion including the minimal cut sets obtained from the BDD, which represents the modular event.

The process is over when only basic events appear in the array

8.3.6 Component Connection Method

When the Component Connection Method is used for converting fault trees to BDDs, an alternative data structure of a binary tree is used to store the BDD. A binary tree is an efficient form of data representation because the memory for a new node in the binary tree is assigned when it is required not all in advance. A binary tree has a root node, a left binary tree and a right binary tree. The root node contains numerical data of the node variable, the left tree corresponds to the BDD structure beneath the 1 branch of the node and the right tree corresponds to the BDD structure beneath the 0 branch of the node. No pre-processing time is required to prepare fault trees for the conversion to BDDs process if an ordering scheme is not used.

In the Component Connection Method two alternative approaches of traverse are used - the top-down method and the bottom-up method. In the top-down method BDDs are created for every gate starting from the top event. Then every node that represents a gate is replaced by its BDD. In the bottom-up way the gates of the fault tree are traversed constructing BDDs for gates with event inputs only. Then in a bottom-up way the BDDs are constructed for every gate until the top event is reached. A detailed explanation of the bottom-up method is provided in this section.

Initially every gate that has as inputs only basic events are put in a binary tree according to the type of the gate. If the gate is an 'AND' gate the basic events are connected on the left branch (the 1 branch) of the tree, if the gate is an 'OR' gate the basic events are connected on the right branch (the 0 branch) of the tree. If a basic event ordering scheme is used the set of event inputs is ordered before the connection process. For any gate with gate inputs a selection rule is implemented

in order to ascertain which is selected to be the main BDD during the conversion process. Then the process of connection starts merging two chosen BDDs. The main binary tree is traversed searching for terminal vertices, i.e. terminal vertices 1 for an 'AND' gate and terminal vertices 0 for an 'OR' gate. A new copy of the secondary BDD is then created and connected to the main BDD at every appropriate terminal vertex. After the connection process the new, combined, BDD is scanned. If there are some repeated events on any path through the BDD the simplification process is applied. The resulting BDD is then set to be the main BDD for the next connection step for the gate until all BDDs representing gate inputs are merged.

In order to determine when sub-node sharing can be used during the connection process, paths in the main BDD to where the second BDD will be attached are checked for the occurrence of the repeated events. Those terminal vertices that are reached traversing the same branches of the nodes with repeated basic events can be replaced by the same copy of the secondary BDD. In other cases a new copy of the secondary BDD is created. BDDs for repeated gates can be reused.

For example consider the fault tree shown in Figure 8.3. Let the fault tree conversion to BDD technique be the bottom-up method, with no variable ordering required and the left-most BDD always chosen to be the main BDD. First of all, two binary trees are created for gates 10001 and 10002, shown in Figure 8.7. Note: NULL is the null-pointer value used with the pointer operation. An information

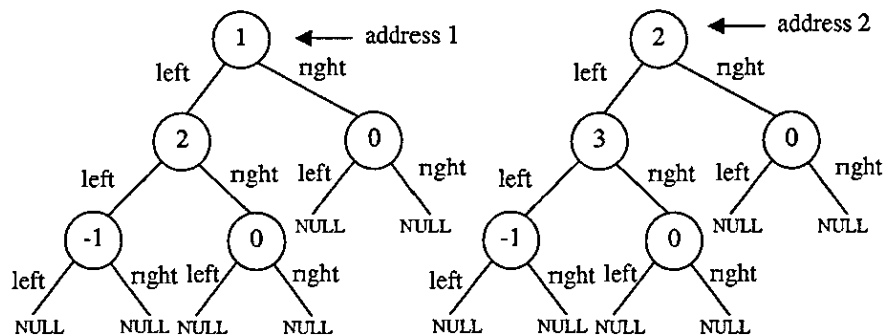


Figure 8.7: Binary trees for gates 10001 and 10002

array is also created, shown in Table 8.7. The first entry is the address of the root of each binary tree, the second entry is the label of the gate represented by the

Line number	top	gate	number	members[]	used
1	address 1	10001	2	{1,2}	0
2	address 2	10002	2	{2,3}	0

Table 8 7: References table

binary tree, the third entry is the number of different events in the tree, the fourth entry is the array of those events and the last entry identifies if the binary tree has been used while merging BDDs of a parent gate.

Then the left-most BDD (of gate 10001) is chosen to be the main BDD. Since gate 10000 is an 'OR' gate every node with value 0 is replaced by the secondary BDD. First of all, the vertex 0 is reached on the right branch of the node with basic event 2. It is replaced by the secondary BDD. Then the second vertex 0 is found on the right branch of root node 1 and it is replaced by a new copy of the secondary BDD. The result is shown in Figure 8 8 and Table 8 8

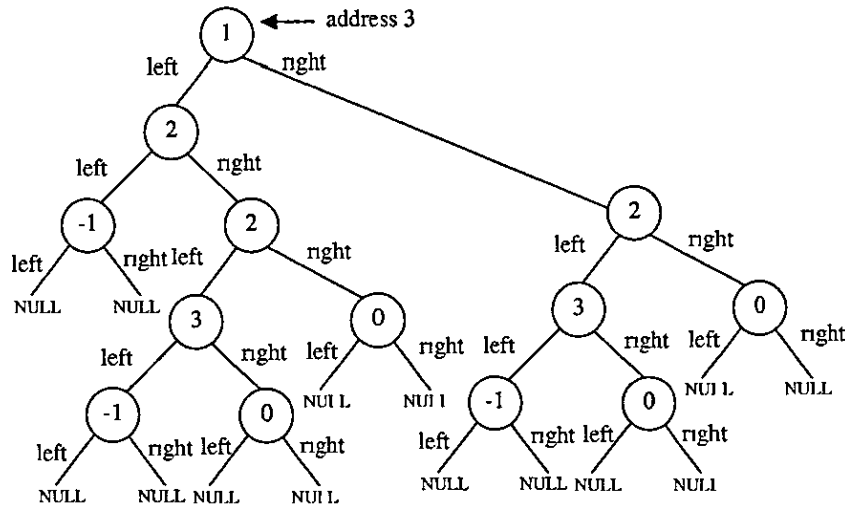


Figure 8 8 Binary tree for gate 10000, before the simplification

Line	top	gate	number	members[]	used
1	address 1	10001	2	{1,2}	1
2	address 2	10002	2	{2,3}	1
3	address 3	10000	3	{1,2,3}	0

Table 8 8 References table after connection

The two arrays of events of the binary trees with 'address 1' and 'address 2' are scanned and a repeated event 2 is found. Therefore, the simplification process is applied in the resulting binary tree removing repeated events from every path. The only repetition is on the right branch of the first node with basic event 2, the second appearance of basic event 2 is replaced by the binary tree on its right branch, i.e. node with terminal vertex 0. The list of events for the resulting BDD of gate 10000 is created merging basic events of the two BDDs. Sometimes, after the simplification process this array needs to be adjusted because some events might not appear in the tree anymore. This new entry is added to the reference table shown in Table 8.8.

In the sub-node sharing version of the Component Connection Method a record of every visited repeated event is stored before the connection process, therefore, it can be decided if the same copy of the secondary BDD can be reused or a new copy needs to be made. Considering the two binary trees for gates 10001 and 10002 in Figure 8.7, for the first terminal vertex on the right branch of the node (gate 10001) with basic event 2 the state of the repeated event 2, which is 0, is recorded. An original secondary BDD can be used. The simplifications are made before the connection process since all the required information about repeated events is known. A simplified BDD is connected replacing the terminal vertex 0. The address of the simplified BDD is recorded and is reused for the terminal vertices with the same record of repeated events. For the second occurrence of a terminal 0 vertex (right branch of node 1) and the record of the repeated event is recorded as -1, because repeated event 2 is not encountered on this path. A new copy of a secondary BDD is obtained and the connection performed. No simplifications are performed because no repeated events were passed. The final tree and the changes in the record array are shown in Figure 8.9 and Table 8.9.

Line number	element	record	root	Line number	element	record	root
1	2	0	address 4	1	2	-1	address 5

Table 8.9: Record of visits of repeated events

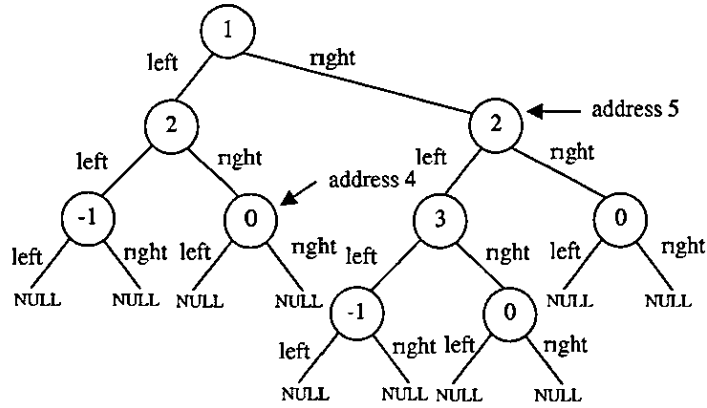


Figure 8 9 Binary tree for gate 10000, sub-node sharing

8.3.7 Hybrid Method

The core of the algorithm for the Hybrid method is based on the *ite* technique. Parts of the fault tree where the Component Connection Method can be introduced are identified before the conversion process. These parts are gates that consist of event inputs only and gates whose descendants do not have any repeated events. They are not pre-processed to create gates with two events only.

For example consider the left-hand fault tree in Figure 8 10. In the pre-processing

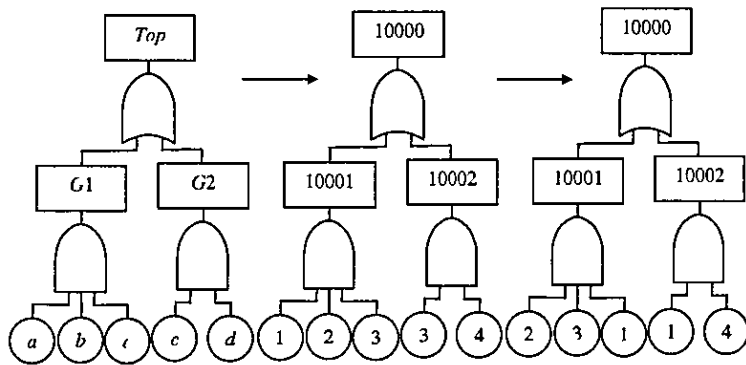


Figure 8 10: Example fault tree for the Hybrid method

stage, having converted it to a numerical format (centre fault tree), gate 10001 is not altered because it has event inputs only. When considering the *ite* method, the variable ordering is assigned, in this case, $c < a < b < d$ and every basic event is assigned a position in the BDD array with two terminal vertices. Then gates 10001 and 10002 are considered and their basic events are linked in a chain

according to the specified order, entry 5 for gate 10001 and entry 7 for gate 10002 in the BDD array. The final connection is performed using the ite technique, since there is event 1 (c) repeated, and the BDD for gate 10000 starts at entry 8. The complete BDD array is shown in Table 8 10

	Line number	Node	1 branch	0 branch
	1	1	-1	0
	2	2	-1	0
	3	3	-1	0
	4	4	-1	0
10001 →	5	1	6	0
	6	2	3	0
10002 →	7	1	4	0
10000 →	8	1	9	0
	9	2	10	4
	10	3	-1	4

Table 8.10: ite array of the Hybrid method

8.3.8 Non-coherent FT Input Format and Conversion to SF-BDD

Non-coherent fault trees contain some basic events that can appear in both, failed and working, states. In the data file every negated basic event or gate is presented starting with a '-' sign, i.e. \bar{a} is equivalent to $-a$ and $\overline{G1}$ is equivalent to $-G1$. During the data input two states of a basic event are assigned two separate numbers and the information is kept in the array of negated variables. After the data input all negations of gates are pushed down to the level of basic events adjusting their basic events and the content of the array of negative values according to it.

During the conversion of a fault tree to a SFBDD the ite technique is applied in the same way as it was in the coherent case. The only additional rule is that the negated basic events (that appear in their working states) are assigned an entry in the ite array in a different way. The first entry is a numerical value of its failed state if basic event appears in both states or a numerical value of its working state if basic event appears only in its working state. The second entry is a terminal vertex 0 and the third entry is a terminal vertex 1, coded by -1. Then the conversion process is carried out in the same way as was described in the ite module. Consider the example fault tree shown in Figure 8 11. The array of negative values

is presented in Table 8 11. After the conversion process the final BDD is shown in Table 8 12.

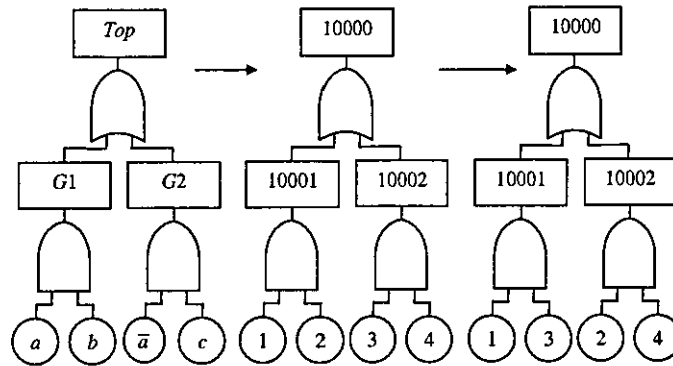


Figure 8.11: Example non-coherent fault tree

Line number	element _char	element _int	element _pos
1	-a	1	3

Table 8.11: Array of negative values

Line number	Node	1 branch	0 branch
1	1	-1	0
2	1	0	-1
3	3	-1	0
4	4	-1	0
10001 →	5	1	3
10002 →	6	1	0
10000 →	7	1	3

Table 8 12: ite array for non-coherent fault tree

8.3.9 Non-coherent Fault Tree Conversion to BDDs for the Qualitative Analysis

The following modules represent four different algorithms to perform qualitative analysis of non-coherent fault trees.

The **TDD module** converts a fault tree to a TDD. In this case the **ite** array has an additional fourth column that represents the 'C' branch. The basic **ite** technique is followed when creating a TDD with the addition that it has an entry in the fourth column in every row which is the consensus of the entry of the second column and the entry of the third column. The 'C' branch is only required for those nodes that represent failure and repair relevant components. For the other entries, i.e. where nodes represent failure or repair relevant components the value in the fourth column is assigned -1000, which is just a number chosen to identify the fact that the 'C' branch is not required. During the minimisation of the TDD and the calculation of prime implicant sets updated versions of the minimisation module and minimal cut sets calculation module from the coherent case are used. While minimising a TDD repeated paths that appear on the 'C' branch are removed not only from the 1 branch (as it was in the conventional minimisation module) but also from the 0 branch. During the identification of prime implicant sets every path from the root vertex to a terminal vertex 1 is traversed. Basic events on the 1 branch of the path enter the prime implicant set in their failed state (as it was in the conventional minimal cut sets calculation module) and basic events on the 0 branch of the path enter the prime implicant set in their working state. Basic events passed on the 'C' branch are not included in a set. If the 'C' branch does not exist the minimisation and the calculation of minimal cut sets is equivalent to the processes in the coherent case.

The **Meta-products BDD module** traverses the SFBDD in a top-down way and converts it to a Meta-products BDD. The result of the process is stored in the same **ite** array together with the SFBDD. Then the meta-products are obtained utilising the module which calculates minimal cut sets. Finally, the decoding of prime implicant sets from the meta-products is applied resulting in the prime implicant sets.

The **ZBDD module** like the Meta-products BDD module handles the SFBDD developed in the BDD module. The SFBDD is traversed in a bottom-up way. The rules of decomposition are applied and a new two-dimensional array is obtained that contains the resulting ZBDD. During the process the consensus of two branches is calculated for certain parts of the SFBDD (as was presented in the TDD method). The 'without' operator from the minimisation module is also used.

The **L-BDD module** calculates a BDD structure, called an L-BDD, that can be used for both, qualitative and quantitative analyses. An additional variable is introduced for every basic event that appears in both working and failed states. The composition of the L-BDD depends on the type of the basic events encountered. Thus the rules of the main BDD construction module are augmented with some additional laws for those cases. During the minimisation procedure some additional rules of operator 'without' are used. Since the consensus of two branches is required for some nodes the information is saved in a reference array. This data is used while identifying prime implicant sets and it incorporates the minimal cut sets calculation module.

In all four methods the computational table is kept so that the **ite** entries (or the **ifre** entries in the TDD method) can be reused. Minimal solutions and values of 'without' operator are also stored and can be used again later in the process.

Chapter 9

Conclusions and Future Work

9.1 Summary of Work

The four main areas of the BDD method development were

- The simplification techniques of fault trees prior to their conversion to BDDs and their qualitative analysis;
- The alternative Component Connection Method for conversion of fault trees to BDDs;
- The new Ternary Decision Diagram method and the efficiency analysis of the other established methods for the conversion and the qualitative analysis of non-coherent fault trees,
- The system reliability assessment of Phased Mission Systems using the BDD method and simplification methods for fault trees.

The summary of each part of the research is explained in the following sections

9.1.1 Qualitative Analysis of Coherent Systems

The qualitative analysis of coherent fault trees using BDDs was extended to the method, where fault trees were simplified and modularised prior to conversion of fault trees to BDDs. The simplification of fault trees can minimise the size of the problem remarkably which is very important for memory management and performance of analysis. After the fault tree simplification, BDDs were obtained. Their qualitative analysis resulted in minimal cut sets that contained modular and complex events that were not easy to understand and might not have had a clear

reliability meaning behind them. Therefore, the qualitative analysis was extended by proposing an algorithm to extract basic events from modular and complex events. This method allowed system failure combinations to be obtained in terms of its original basic events that could be analysed.

9.1.2 Component Connection Method

The *ite* technique for converting fault trees to BDDs involves applying the *if-then-else (ite)* technique to each of the gates in the fault tree. Before the process takes place every gate is broken down to two inputs. A variable ordering for basic events is introduced and applied in every step of the process. Once the BDD is constructed, the quantitative and qualitative analyses are performed. An advantage of the *ite* method for constructing the BDD is that the algorithm automatically makes use of sub-node sharing. This not only reduces the computer memory requirements, as each *ite* structure is only stored once, but it also increases the efficiency, since once an *ite* structure has been calculated the process does not need to be repeated. However, in the *ite* method the ordering scheme needs to be introduced and it can have a crucial effect on the size of the BDD.

An alternative technique was developed as part of the research reported in this thesis. The Component Connection Method is applied for any logic gate in a fault tree. BDDs are built for every gate and then recursively combined until the BDD for the top gate is obtained. If there are some repeated events in the fault tree, simplification rules are applied that remove repeated nodes from every path in the BDD. The advantages of the Component Connection Method are that no ordering scheme of variables is required because basic events can be connected in the order that they appear in the fault tree. Also, gates do not need to be broken down to two inputs, since the connection process can deal with all the inputs of the gate.

Different connection techniques were investigated. Selection schemes for the traverse approach (top-down or bottom-up), for the order of basic events selection, for the order of inputs selection (for the top-down technique) and for the order of BDDs selection (for the bottom-up technique) were assigned. The efficiency of the algorithm and proposed different techniques was measured by the computational time taken to convert a fault tree to a BDD, together with the size of the final BDD and the maximum required size of the dynamic memory data structure during the

process.

The traverse approach, the top-down or the bottom-up method, was chosen to build a BDD. Both strategies gave the same final BDD as long as the ordering parameters selected remain consistent. However, the big advantage of the bottom-up technique against the top-down was that a smaller memory resource was required. This was because in the bottom-up method the fault tree was investigated 'in portions', i.e. building a BDD for the left-most gate input was finished and simplifications applied before the investigation of other inputs. In the top-down technique the whole set of inputs for the top event was connected and then every gate node was replaced, still trailing the rest of the structure until the last gate node was replaced. Therefore, for the further analysis the top-down traverse approach was discarded. Also, eight ordering schemes for selecting basic events were assigned and analysed together with the 'order as listed' scheme, where basic events were put in a BDD according to the order that they appeared in a fault tree.

The bottom-up approach was investigated using three BDD selection schemes, i.e. three different ways that BDDs representing gate inputs of a parent gate can be connected. In the first trial, basic events were ordered according to the one out of the 8 ordering schemes or 'order as listed'. BDDs were selected as listed, i.e. a BDD which represented the first input of a gate in a list of inputs was chosen to be the main BDD. In the second trial the ordering schemes were used not only to order basic events but also to choose the order that BDDs were considered in the connection process. The priority was given to the BDD whose top event was the highest in the ordering scheme. Finally, in the third trial BDDs were merged according to the number of points at which the BDDs were combined. The ordering schemes were applied for the selection of basic events but not for the selection of BDDs. The comparison of the three techniques was performed using the example fault tree library with different complexity of fault trees.

The ordering schemes were ranked using the three ranking methods - the total quantity ranking, the highest scheme ranking and the added scheme ranking. Firstly, the sum of each characteristic measure was calculated over the whole set of test fault trees, for example, the time taken to build BDDs for the whole set of fault trees was obtained. Secondly, the number of times that each scheme produced the highest (best) ranking was assessed. Finally, the average ranking of each scheme

across the set of fault trees was considered. These measures gave an indication of the overall performance of the ordering scheme. The ordering schemes were ranked for all the efficiency measures according to their performance on small fault trees, 'large' fault trees and the whole set of fault trees. This was done in order to be able to identify the 'optimum' ordering scheme for every method in each complexity category if one exists.

The third trial of the bottom-up strategy approach gave the best result. The total number of nodes in BDDs using the third trial was smaller than the total number of BDDs from the first trial or the second trial. The total processing time was shorter for the third trial than for the first or the second trial using different ordering schemes. The third trial that uses the selection of BDDs according to the number of connection points available and the selection of basic events according to one of the nine ordering schemes, was the most efficient method out of the all proposed bottom-up approaches.

The Component Connection Method could not convert some fault trees to BDDs in reasonable times, whereas the *ite* technique was giving good results. The main disadvantage of the method was identified. The strongest property of the *ite* technique is the sub-node sharing method, which is the property that the Component Connection Method lacks. The Component Connection Method produced significantly bigger BDDs than the *ite* method. This approach had a high demand for memory space since the identical parts in the BDD structure were repeated but not shared. Therefore, the sub-node sharing was introduced in the proposed method. This property was introduced while combining two gates, i.e. all available branches would point to the same structure, instead of making a separate copy for each of them. This rule has increased the efficiency of the technique. However, since after the connection process the simplifications were applied if repeated events appeared in the structure, there were some limitations on the described rule. Therefore, overall the Component Connection Method could not give as good efficiency as the *ite* method.

Finally, the Hybrid method was proposed that combined the best features of the two construction methods of BDDs, the *ite* method and the Component Connection Method. The new method incorporated the most efficient parts of both algorithms. By using the gate constructs for basic events and branches without

repeated events BDDs were immediately formed without any of the processing required by the *ite* method. For the rest of the fault tree the sub-node sharing feature of the *ite* method was employed which provided a more efficient representation of the logic function. The results of the Hybrid method that combined the two techniques were comparable with the *ite* method, because an improvement, albeit slight, in all efficiency measures was observed. Two types of the Hybrid method were proposed - the Basic Hybrid method and the Advanced Hybrid method. While comparing the two types of the Hybrid method, the Advanced Hybrid method, which allowed the node swap, gave a slightly better performance than the Basic Hybrid method according to the number of nodes but not according to the processing time.

Analysing the ranking of the ordering schemes it was clear that the two top-down weighted schemes, (5) and (6), were favourite in many cases. The two modified depth-first schemes, (2) and (3), performed poorly for the majority of examples. As it was expected, some orderings performed better on certain fault trees. However, there was a tendency that the weighted ordering schemes (5-8), that order events according to their importance in the fault tree, performed better than the neighbourhood ordering schemes (1-4), that order events according to their position in the fault tree. Also, choosing any of the proposed ordering schemes was better than using 'ordered as listed' method. Therefore, an ordered approach for the Component Connection Method was advisable even when it was not required. This was due to the fact that the ordering schemes brought the repeated events 'closer' and there were fewer parts in a BDD that were repeated.

9.1.3 Qualitative Analysis of Non-coherent Systems

Qualitative analysis using BDDs was expanded on non-coherent systems, because the BDD method is more efficient and accurate than conventional FTA methods. Every fault tree is converted to a BDD from which exact quantitative analysis is performed. However, it is not possible to identify the prime implicant sets directly from the BDD. In non-coherent fault trees components can be failure or/and repair relevant, and it is impossible to distinguish between the two cases from the BDD while performing the qualitative analysis. Therefore, alternative methods are required.

A new Ternary Decision Diagram Method has been proposed in this work. A TDD has three branches leaving every node. It encodes all prime implicant sets, since the consensus branch for every node is calculated by the conjunction of the 1 and the 0 branches for every node. The TDD is non-minimal, therefore, the minimisation process needs to be performed in order to be able to obtain prime implicant sets. The TDD can be used for the quantitative analysis if required. The efficiency of the new approach was analysed comparing it with the three established methods for the analysis of non-coherent fault trees, identifying the advantages and disadvantages of the different techniques, while testing them on a range of fault trees.

In 1998 Rauzy and Dutuit developed a technique for computing prime implicant sets using the Meta-products BDD. The produced Meta-products BDD is minimised, therefore, prime implicant sets can be obtained without the minimisation operation. However, using this method it is necessary to compute two BDDs to perform the qualitative analysis. Also, all components are described by two variables. While converting example fault trees to BDDs the size of the structure and the processing time increased unavoidably.

The second established method is called the Zero-suppressed BDD method. The conception of a ZBDD was introduced by Minato [34] and later it was accommodated by Rauzy [33] for a compact representation to express prime implicant sets. Like in the Meta-products BDD method, the technique involves computing an additional BDD, where nodes are labelled with failed and/or working states of basic events. Then prime implicant sets are decomposed according to the presence of a given state of a basic event. The resulting ZBDD is in its minimal form and prime implicant sets can be obtained. ZBDDs automatically suppress basic events that do not appear in prime implicant sets. It is very efficient when calculating sets with basic events that are far apart in the variable ordering scheme. The ZBDD method resulted in an efficient process, where all prime implicant sets were described by a compact and easy handling structure. The efficiency of the ZBDD method was very close to the performance of the TDD method in both measurements. This made the two methods most suitable for the analysis of the non-coherent example fault trees.

The Labelled Binary Decision Diagram (L-BDD) was developed by Contin in [35], where every basic event is labelled according to its type. The additional infor-

mation about the occurrence of every basic event is considered at an early stage of the algorithm. An L-BDD has two branches, therefore, it does not provide all prime implicant sets. The L-BDD obtained from the fault tree is simplified and then the determination of prime implicant sets is performed, where the rules of the calculation depend on the type of a basic event. The idea of the L-BDD method is similar to the TDD method, i.e. the conjunction of the 1 branch and the 0 branch is only carried out for nodes that represent dual state basic events. However, there are some differences that make the L-BDD method perform less efficiently than the TDD method. In the L-BDD method the number of basic events is increased by introducing three different types of basic events according to their occurrence. Also, some extra connection rules are used in the conversion process and in the calculation of prime implicant sets. While converting example fault trees, the L-BDD performed better than the Meta-products BDD method, but not as well as the TDD method or the ZBDD method, which provided efficient methods for the calculation of prime implicant sets. For 'large' fault trees the difference in efficiency between the techniques was larger than for small fault trees.

In summary, a comparison of the four methods revealed that the TDD method and the ZBDD method performed well for both measurements, i.e. the number of nodes and the conversion time. The TDD method was slightly faster than the ZBDD method. This was because of the fact that one structure (a TDD) was required in this method, which presented a SFBDD as well as it was suitable for the qualitative analysis. In the ZBDD method a SFBDD was constructed and then an additional ZBDD was built that resulted in a longer conversion process. However, the ZBDD structure was minimal, therefore the number of nodes was smaller than in the TDD method, where each TDD was minimised before obtaining prime implicant sets. Both methods gave close results and provided the efficient means for the analysis of non-coherent fault trees.

The Advanced Hybrid method was also introduced for non-coherent systems. One of the best chosen methods, the ZBDD method, was combined with the Advanced Hybrid method. The SFBDD, that was used for the quantitative analysis and the construction of the background for the ZBDD, was built using the Hybrid method, i.e. the combination of the *ite* technique and the Component Connection Method. The overall result was slightly better than the result obtained using the *ite* technique. Therefore, the Advanced Hybrid method in a non-coherent case provided

an efficient technique for converting fault trees to BDDs.

The eight ordering schemes were ranked for the four conversion methods. Using the example fault tree library provided the dynamic top-down weighted scheme (6) performed well for all the methods and both measurements. The depth-first, with number of leaves scheme (4) was also ranked highly. The modified top-down scheme (1) gave average or poor results, especially according to the number of nodes. The two depth-first schemes, (2) and (3), also performed poorly. Summarising the ranking, the weighted ordering schemes, (5)-(8), performed better than the neighbourhood ordering schemes, (1)-(4). Those results are based on example fault trees. For a different set of examples the ordering schemes will perform differently, however, the main efficiency pattern should be the same.

9.1.4 Application of Proposed Methods in Phased Mission Analysis

Many types of system operate for missions that are made up of several phases. FTA can be used as a means for analysing the reliability of non-repairable systems that undergo phased missions. The required results are the system failure modes and failure probability of each phase, followed by the total mission unreliability. The complexity of fault trees might make the analysis impossible, therefore, some alternative methods were incorporated in the analysis process. The accurate assessment of mission unreliability for systems with non-repairable components operating over a sequence of phases can be performed using non-coherent fault tree structures. Therefore, at the start non-coherent fault trees representing phase failures were simplified. Then the BDD method was employed to calculate the unreliability of each phase. New rules for expressing disjoint path sets and calculation of phase failure probability were considered. This method allowed a more efficient and accurate Phased Mission Analysis.

9.2 Conclusions

- The BDD method can be used to perform the efficient qualitative analysis and accurate quantitative analysis of system reliability. When analysing large industrial systems, fault trees can be simplified and modularised prior to the conversion to BDDs. Then during the qualitative analysis modular and

complex events in minimal cut sets need to be extracted so that system failure combinations could be analysed in terms of basic events. An algorithm was presented in order to be able to obtain minimal cut sets in terms of the original components of the system

- An alternative method to the *ite* technique was developed for converting fault trees to BDDs, which involves computing the BDDs in the Component Connection manner. Whilst this method has some advantages over the conventional *ite* technique, only the Hybrid method, that combines the efficiency of the two conversion techniques, gives comparable results.
- A new approach, the Ternary Decision Diagram method, has been proposed and developed for the conversion and analysis of non-coherent fault trees. Its efficiency was analysed comparing it with the established methods. The TDD method and the ZBDD method were identified to be efficient for calculating prime implicant sets of non-coherent fault trees. Application of the Hybrid method provided a good extension and additional efficiency in the qualitative analysis of non-coherent systems
- The simplification of phase failure fault trees and the application of the BDD method for obtaining mission failure probability is an efficient application of the proposed methods in the Phased Mission Analysis

9.3 Future Work

9.3.1 Component Connection Method

The efficiency of the Component Connection Method is not comparable with the efficiency of the *ite* technique. Only the Hybrid method, that combines the Component Connection Method and the *ite* method, gives better results. However, the improvement in the efficiency is marginal. Therefore, more work could be done on researching new ways that BDDs are considered in the connection process, or new ways of connecting nodes in the BDD structure. Also, trying to increase the sub-node sharing capacity and improving the algorithm proposed in the Component Connection Method, the sub-node sharing could be applied to all the available nodes and then the 'unsharing' process could be performed if needed.

For the currently proposed Component Connection Method there were some fault

trees where calculations were not finished in reasonable times. Alternative programming methods could be used to obtain a better efficiency of techniques, incorporating parallel programming option.

9.3.2 Application of Proposed Methods in Phased Mission Analysis

Application of the fault tree simplification and the BDD technique for Phased Mission Analysis could be applied to other system reliability tools. For example, a Cause Consequence Diagram could be built using the simplified fault trees and then quantified using the BDD method.

With increasing demand of the real-time modelling, the Phased Mission Analysis incorporating the efficient tool of the Binary Decision Diagram method could be used to model real systems. One of the examples could be a model for a Unmanned Autonomous Vehicle, where the flight could be described using the Phased Mission Analysis. The efficient prediction of future capability would enable the analysis to be performed in real-time and make decisions about the future of the mission.

Bibliography

- [1] Vesely, W.E 'A Time Dependent Methodology for Fault Tree Evaluation', Nuclear Design and Engineering, 13, pp337-360, 1970
- [2] Rauzy, A. 'New Algorithms for Fault Tree Analysis', Reliability Engineering and System Safety, 40, pp203-21, 1993
- [3] Andrews, J D and Moss, T R 'Reliability and Risk Assessment', Longman Scientific & Technical, pp144-199, 2002
- [4] Henley, E.J. and Kumamoto, H. 'Reliability Engineering and Risk Assessment', Prentice-Hall, pp170-209, 1981
- [5] Reay, K A. 'Efficient Fault Tree Analysis Using Binary Decision Diagrams', Doctoral Thesis, Loughborough University, 2002
- [6] Platz, O. and Olsen, J.V. 'FAUNET: A program Package for Evaluation of Fault Trees and Networks', Research Establishment Risk Report, No 348, DK-4000 Roskilde, Denmark, Sept 1976
- [7] Dutuit, Y. and Rauzy, A 'A Linear-Time Algorithm to Find Modules of Fault Trees', IEEE Trans. Reliability, 45, No 3, pp422-425, 1996
- [8] Lee, C 'Representation of Switching Circuits by Binary Decision Diagrams', Bell Syst Tech Journal, No 38, pp985-999, 1959
- [9] Akers, S.B. 'Binary Decision Diagrams', IEEE Trans Computers, C-27, No 6, pp509-516, 1978
- [10] Bryant, R.E. 'Graph-Based Algorithms for Boolean Function Manipulation', IEEE Trans Computers, C-35, No.8, pp677-690, 1986
- [11] Brace, K S , Rudell, R L , Bryant, R E 'Efficient Implementation of a BDD Package', 27th ACM/IEEE Design Automation Conference, Paper 3.1, pp40-45, 1990

- [12] Schneeweiss, W G. 'Fault-Tree Analysis Using a Binary Decision Tree', IEEE Trans. Reliability, R-34, No 5, pp453-457, 1985
- [13] Sinnamon, R M. and Andrews, J.D. 'Improved Efficiency in Qualitative Fault Tree Analysis', Quality and Reliability Engineering International, Vol. 13, pp293-298, 1997
- [14] Remenyte, R and Andrews, J.D 'System Failure Modelling using Binary Decision Diagrams', Proceedings of the 16th ARTS (Advances in Reliability Technology Symposium) , Loughborough University, UK, April 2005, pp 379-394, ISBN 0 904947 61 0
- [15] Remenyte, R and Andrews, J D 'Qualitative Analysis of Complex Modularised Fault Trees Using Binary Decision Diagrams', Journal of Risk and Reliability, Proceedings of the IMechE Part O, vol 220, pp45-54, June 2006
- [16] Sinnamon, R M and Andrews, J D. 'Improved Accuracy in Quantitative Fault Tree Analysis', Quality and Reliability Engineering International, Vol. 13, pp285-292, 1997
- [17] Reay, K A. and Andrews, J D. 'A Fault Tree Analysis Strategy Using Binary Decision Diagrams', Reliability Engineering and System Safety, 78, pp45-56, 2002
- [18] Way, Y.S. and Hsia, D.Y. 'A simple Component-Connection Method for Building Binary Decision Diagrams Encoding a Fault Tree', Reliability Engineering and System Safety, 70, pp59-70, 2000
- [19] A Benchmark of Boolean Formulae, <http://iml.univ-mrs.fr/arauzy/aralia/benchmark.html>
- [20] Meinel, C. and Sack, H 'Algorithmic Considerations for OBDD Reordering', Proceedings of IEEE/ACM Int. Workshop on Logic Synthesis, Lake Tahoe, CA, pp71-74, 1999
- [21] Andrews, J D and Remenyte, R. 'Fault Tree Conversion to Binary Decision Diagrams', Proceedings of the 23rd ISSC , San Diego, USA, August 2005, ISBN 0-9721385-5-2 , [CD-ROM]
- [22] Andrews, J.D. and Remenyte, R 'A Simple Component Connection Approach for Fault Tree Conversion to Binary Decision Diagram', Proceedings of the 1st AReS, pp449-456, Vienna, Austria, April 2006, ISBN 0-7695-2567-9/06

- [23] Remenyte-Prescott, R and Andrews, J D. 'An Enhanced Component Connection Method for Conversion of Fault Trees to Binary Decision Diagrams', *Reliability Engineering & System Safety*, April 2007, revisions submitted
- [24] Andrews, J.D. 'To Not or Not to Not!!', *Proceedings of the 18th International System Safety Conference*, pp267-274, 2000
- [25] Andrews, J D and Dunnett, S J. 'Improved accuracy in event tree analysis', *Foresight and Precaution*. Cottam, Harvey, Pape and Tate (eds). *Proceedings of ESREL 2000, SARS and SRA-EUROPE annual conference*, pp1525-1532, 2000
- [26] Inagaki, T. and Henley, E J 'Probabilistic Evaluation of Prime Implicants and Top Events for Non-coherent Systems', *IEEE Transactions on Reliability*, vol R-29, pp361-367, Dec 1980
- [27] Becker, G. and Camarinopoulos, L. 'Failure Frequencies of Non-coherent Structures', *Reliability Engineering and System Safety*, vol.41, pp209-215, 1993
- [28] Andrews, J D. and Beeson, S C. 'Birnbaum's Measure of Component Importance for Non-coherent Systems', *IEEE Transactions on Reliability*, vol 52, pp213-219, June 2003
- [29] Remenyte-Prescott, R. and Andrews, J.D. 'Prime Implicants for Modularised Non-coherent Fault Trees', *International Journal of Reliability and Safety*, 2007, accepted for publishing
- [30] Sasao, T. 'Ternary Decision Diagrams and their Applications', *Representations of Discrete Functions*, Chapter 12, pp269-292, 1996
- [31] Courdet, O and Madre, J.-C. 'A New Method to Compute Prime and Essential Prime Implicants of Boolean Functions', *Advanced Research in VLSI and Parallel Systems*, pp113-128, 1992
- [32] Dutuit, Y. and Rauzy, A. 'Exact and Truncated Computations of Prime Implicants of Coherent and Non-Coherent Fault Trees with Aralia', *Reliability Engineering and System Safety*, 58, pp225-235, 1997
- [33] Rauzy, A. 'Mathematical Foundation of Minimal Cutsets', *IEEE Transactions on Reliability*, volume 50, 4, pp389-396, 2001

- [34] Minato, S. 'Zero-Suppressed BDDs for Set Manipulation in Combinatorial Problems', Proceedings of the 30th ACM/IEEE Design Automation Conference, DAC'93, pp272-277, 1993
- [35] Contini, S. 'Binary Decision Diagrams with Labelled Variables for Non-Coherent Fault Tree Analysis', European Commission, Joint Research Centre, 2005
- [36] Remenyte-Prescott, R. and Andrews, J D 'Ternary Decision Diagrams for the Real-time Analysis of Non-coherent Fault Trees', The 5th International Mathematical Methods in Reliability (MMR) Conference, Glasgow, Scotland, July 2007
- [37] Beeson, S C and Andrews, J.D 'Calculating the Failure Intensity of a Non-coherent Fault Tree Using the BDD Technique', Quality and Reliability Engineering International, 20, pp225-235, 2004
- [38] Esary, J.D. and Ziehms, H 'Reliability of Phased Missions', Reliability and Fault-Tree Analysis, (Society for Industrial Applied Mathematics, Philadelphia, Pennsylvania), pp213-236, 1975
- [39] La Band, R.A. and Andrews, J.D 'Phased Mission Modelling Using Fault Tree Analysis', Proceedings of the Institution of Mechanical Engineers, Part E· J Process Mechanical Engineering, vol. 218, pp83-91, 2004
- [40] Courdet, O and Madre, J.-C 'Fault Tree Analysis 10²⁰ Prime Implicants and Beyond', Proceedings of Annual Reliability and Maintainability Symposium, pp240-245 (1993)
- [41] Beeson, S C 'Non-coherent Fault Tree Analysis', Doctoral Thesis, Loughborough University, 2002

Appendix A

Results

FT name	Number of gates	Number of basic events	Number of repeated events	Number of complex events	Number of modules	Number of cut sets
astolfo gsq	19	16	2	8	1	27
benjam gsq	15	11	7	0	1	43
bpfeg03 dat	20	63	0	62	1	8716
bpfen05 dat	17	61	0	60	1	7471
bpfig05 dat	17	60	0	59	1	7056
bpfin05 dat	14	40	0	39	1	416
bpfsw02 gsq	21	40	4	32	2	84423
dre1019 dat	4	19	1	18	1	63
dre1032 dat	4	21	1	20	1	75
dre1057 dat	7	32	1	31	1	2100
dre1058 dat	13	41	21	31	3	11934
dre1059 dat	17	57	21	40	1	36990
dresden dat	17	57	21	40	3	11934
hpsif02 dat	19	72	8	57	1	255
hpsif03 dat	7	31	2	24	1	71
hpsif21 gsq	15	61	57	52	1	7777
hpsif36 gsq	8	30	4	23	1	61
jdtree3 gsq	11	21	0	20	1	36
jdtree4 gsq	11	20	1	18	1	30
jdtree5 gsq	11	20	1	19	1	10
khictre dat	19	22	16	14	3	21
nakashi gsq	21	16	11	0	1	20
trials1 gsq	27	16	8	2	1	39
trials4 gsq	39	21	14	0	1	49
random3 gsq	24	49	12	27	1	235
random6 gsq	45	49	41	13	1	93
random8 gsq	7	15	6	13	1	4
rando12 gsq	32	68	22	35	1	68
rando13 gsq	46	56	44	16	1	73
rando16 gsq	31	46	25	17	4	76
rando18 gsq	62	85	58	29	1	24
rando23 gsq	19	39	16	26	1	9
rando25 gsq	14	16	11	10	1	6
rando27 gsq	45	46	37	16	1	100
rando28 gsq	17	35	14	7	1	1
rando29 gsq	25	38	18	14	1	22
rando30 gsq	17	41	4	25	1	195
rando31 gsq	47	36	33	10	1	5
rando33 gsq	17	32	19	14	2	11
rando34 gsq	24	36	17	17	1	35
rando35 gsq	19	24	17	9	4	8
rando36 gsq	15	29	7	21	1	10
rando37 gsq	27	30	25	9	1	29
rando38 gsq	11	21	5	15	1	9
rando39 gsq	27	26	22	3	1	51
rando40 gsq	8	17	5	6	2	9
rando42 gsq	9	17	7	6	1	2
rando43 gsq	10	27	3	23	1	22
rando44 gsq	27	59	9	32	1	436
rando45 gsq	22	28	19	8	2	16
rando46 gsq	22	41	18	23	1	10
rando47 gsq	20	42	17	22	1	15
rando48 gsq	16	21	14	9	1	16
rando52 gsq	33	34	24	13	4	41

Table A 1: Complexity of test fault trees, small trees, 1

FT name	Number of gates	Number of basic events	Number of repeated events	Number of complex events	Number of modules	Number of cut sets
rando53 gsq	13	21	11	7	1	2
rando54 gsq	13	34	5	21	5	269
rando55 gsq	15	23	14	11	3	9
rando58 gsq	10	17	10	9	1	3
rando59 gsq	23	42	12	17	1	99
rando60 gsq	36	70	16	39	4	22
rando61 gsq	19	20	15	8	1	15
rando62 gsq	13	18	12	10	1	7
rando63 gsq	15	23	14	11	3	9
rando64 gsq	19	35	10	18	3	31
rando65 gsq	11	15	6	4	1	13
rando66 gsq	17	26	14	9	2	5
rando70 gsq	12	24	4	12	1	27
rando73 gsq	22	34	22	24	1	80
rando75 gsq	12	17	8	12	1	4
rando76 gsq	15	32	10	21	3	24
rando77 gsq	31	37	24	7	1	27
rando78 gsq	17	30	7	16	1	2
rando80 gsq	9	26	3	19	2	22
rando83 gsq	14	21	8	10	1	39
rando84 gsq	19	39	8	20	2	52
rando85 gsq	13	26	11	19	1	7
rando87 gsq	11	22	5	17	1	15
rando88 gsq	11	22	3	13	1	29
rando89 gsq	24	41	15	32	1	21
rando91 gsq	32	58	29	19	1	106
rando92 gsq	41	64	46	22	2	58
rando93 gsq	19	40	12	25	4	16
rando95 gsq	11	22	8	8	3	31
rando98 gsq	22	52	12	26	1	283
rando99 gsq	26	40	26	13	1	28
rand100 gsq	19	27	15	10	1	8
rand103 gsq	13	23	5	12	2	13
rand104 gsq	16	22	13	14	1	9
rand105 gsq	15	33	4	21	1	96
rand106 gsq	31	37	23	12	1	8
rand108 gsq	32	35	26	5	1	35
rand109 gsq	27	56	11	29	1	203
rand110 gsq	24	30	18	10	1	8
rand111 gsq	19	22	15	8	1	22
rand115 gsq	21	29	13	13	1	46
rand116 gsq	24	33	23	9	2	15
rand117 gsq	10	17	5	9	1	11
rand118 gsq	19	39	8	20	2	52
rand119 gsq	14	30	6	28	1	84
rand120 gsq	20	39	7	20	1	58
rand121 gsq	18	37	13	18	1	80
rand123 gsq	9	17	4	12	1	12
rand124 gsq	12	24	5	21	1	27
rand125 gsq	6	14	5	9	1	13
rand126 gsq	25	37	14	22	1	59
rand127 gsq	12	28	3	27	1	43
rand128 gsq	24	35	22	13	1	52
rand129 gsq	8	20	6	9	1	1

Table A.2. Complexity of test fault trees, small trees, 2

FT name	Number of gates	Number of basic events	Number of repeated events	Number of complex events	Number of modules	Number of cut sets
rand130 gsq	13	23	13	6	1	5
rand132 gsq	31	39	26	4	1	67
rand134 gsq	34	56	27	22	1	60
rand135 gsq	24	33	23	7	1	24
rand137 gsq	10	21	5	9	1	15
rand138 gsq	10	18	9	4	1	2
rand139 gsq	21	29	16	9	1	53
rand141 gsq	24	30	18	10	1	8
rand142 gsq	32	46	29	8	2	410
rand143 gsq	17	28	10	18	1	8
rand144 gsq	29	48	26	22	1	41
rand145 gsq	11	33	1	27	2	47
rand146 gsq	10	21	5	9	1	15
rand147 gsq	36	43	29	8	2	30
rand148 gsq	12	27	3	20	1	8
rand149 gsq	22	57	6	44	1	18
rand150 gsq	29	44	24	11	1	114
rand151 gsq	10	28	4	25	1	36
rand153 gsq	16	21	13	9	1	3
rand154 gsq	11	21	7	8	1	1
rand155 gsq	20	33	10	10	1	52
rand156 gsq	10	22	6	21	1	20
rand158 gsq	49	71	34	29	1	9
lisaba1 gsq	27	68	8	47	4	1054
lisaba3 gsq	40	84	18	46	1	5396
lisaba4 gsq	26	44	15	16	2	827
lisaba5 gsq	29	57	19	34	1	228
lisaba6 gsq	22	56	6	39	1	990
lisaba7 gsq	27	68	8	47	4	1054
lisaba9 gsq	17	41	5	31	2	85
lisab10 gsq	27	48	22	13	1	940
lisab11 gsq	13	21	11	7	1	2
lisab13 gsq	24	31	18	10	1	8
lisab17 gsq	27	68	8	47	4	1054
lisab19 gsq	37	112	0	111	1	15420
lisab20 gsq	41	128	0	127	1	7899
lisab24 gsq	15	41	0	40	1	887
lisab25 gsq	15	26	10	11	1	35
lisab27 gsq	26	62	14	37	5	292
lisab28 gsq	9	22	0	21	1	66
lisab30 gsq	19	32	12	16	4	17
lisab31 gsq	31	47	32	13	1	164
lisab34 gsq	8	14	7	3	3	14
lisab35 gsq	19	40	12	16	1	136
lisab36 gsq	46	39	36	5	2	52
lisab37 gsq	10	30	3	24	3	64
lisab42 gsq	7	21	2	19	1	10
lisab44 gsq	10	20	11	8	3	12
lisab46 gsq	50	152	0	151	1	14669
lisab47 gsq	10	12	8	5	1	3
lisab48 gsq	8	17	10	6	1	4
lisab49 gsq	22	60	0	59	1	890
lisab50 gsq	10	14	8	8	1	2
lisab51 gsq	8	19	2	11	1	11

Table A 3. Complexity of test fault trees, small trees, 3

FT name	Number of gates	Number of basic events	Number of repeated events	Number of complex events	Number of modules	Number of cut sets
Isab52 gsq	31	38	32	7	1	139
Isab53 gsq	5	9	1	8	1	15
Isab54 gsq	6	15	4	6	1	14
Isab56 gsq	11	17	9	6	1	3
Isab57 gsq	18	28	16	7	1	170
Isab59 gsq	16	49	0	48	1	3096
Isab60 gsq	7	16	5	6	1	19
Isab61 gsq	22	40	14	19	1	14
Isab62 gsq	17	39	4	30	1	74
Isab63 gsq	8	18	5	12	1	6
Isab64 gsq	21	40	21	16	1	7
Isab65 gsq	7	19	2	18	1	5
Isab66 gsq	31	40	27	9	1	33
Isab67 gsq	38	77	13	49	1	1118
Isab68 gsq	9	25	0	24	1	180
Isab69 gsq	14	30	3	20	1	25
Isab70 gsq	19	48	5	31	1	88
Isab71 gsq	8	14	8	9	1	3
Isab72 gsq	34	51	25	22	1	34
Isab73 gsq	7	23	0	22	1	40
Isab74 gsq	46	122	13	83	1	68122
Isab75 gsq	14	29	10	9	1	1
Isab77 gsq	29	59	13	36	1	130
Isab78 gsq	16	39	9	17	1	503
Isab81 gsq	6	15	0	14	1	11
Isab82 gsq	31	85	9	59	1	33540
Isab83 gsq	19	33	13	14	1	61
Isab84 gsq	19	33	13	14	1	61
Isab86 gsq	21	40	10	14	1	383
Isab88 gsq	25	49	16	29	1	28
Isab89 gsq	32	61	13	38	1	84
Isab91 gsq	32	62	14	35	1	7598
Isab95 gsq	7	18	6	8	1	1
Isab97 gsq	11	26	3	23	1	19
Isab98 gsq	7	16	2	13	1	7
Isa100 gsq	31	63	13	32	1	313
Isa103 gsq	5	9	1	8	1	15
Isa104 gsq	10	20	7	15	1	4
Isa107 gsq	6	11	3	7	1	6
Isa109 gsq	21	20	19	3	1	20
Isa110 gsq	36	52	24	15	1	32
Isa111 gsq	17	50	4	48	1	46
Isa112 gsq	32	81	9	56	1	4769
Isa113 gsq	25	60	14	33	2	121
Isa115 gsq	12	25	8	13	1	37
Isa116 gsq	10	14	7	6	1	6
Isa119 gsq	7	14	2	9	1	15
Isa120 gsq	10	24	0	23	1	126
Isa121 gsq	21	41	5	33	3	72
Isa122 gsq	12	23	10	10	1	10
Isa123 gsq	15	27	9	10	1	37
Isa124 gsq	28	65	15	31	1	1112
rand159 gsq	25	34	20	11	1	13
rand161 gsq	22	38	19	12	1	114

Table A.4. Complexity of test fault trees, small trees, 4

FT name	Number of gates	Number of basic events	Number of repeated events	Number of complex events	Number of modules	Number of cut sets
rand163 gsq	37	58	28	21	2	715
rand164 gsq	32	58	16	28	1	4374
rand165 gsq	40	98	10	66	2	2072
rand166 gsq	31	55	21	15	1	262

Table A 5 Complexity of test fault trees, small trees, 5

FT name	Number of gates	Number of basic events	Number of repeated events	Number of complex events	Number of modules	Number of cut sets
rando11 gsq	48	94	38	33	1	6391
rando19 gsq	51	53	44	2	1	764
rando20 gsq	52	47	41	8	1	122
rando22 gsq	46	64	39	12	1	423
rand160 gsq	95	150	80	56	1	2621
lisaba2 gsq	48	114	27	64	1	66083
lisaba8 gsq	45	100	21	52	1	3344
lisab14 gsq	46	84	40	25	1	1633
lisab15 gsq	49	98	21	44	1	8113
lisab22 gsq	48	72	43	14	1	493
lisab29 gsq	56	146	0	145	1	2202755
lisab76 gsq	38	58	26	15	1	898
lisab87 gsq	54	96	34	32	1	93726
lisab92 gsq	88	228	0	227	1	>4500000
lisa102 gsq	54	110	26	56	1	200063
lisa117 gsq	80	147	47	65	1	239529
lisa118 gsq	37	77	17	38	2	45505
*baobab1 txt	119	60	7	0	1	43379
*baobab2 txt	64	31	3	0	1	4805
*chinese txt	35	24	24	14	1	392
*das9201 txt	81	121	11	104	1	14217
*das9202 txt	35	48	1	47	1	27778
*das9203 txt	29	50	1	43	3	16200
*das9204 txt	29	52	13	41	1	16704
*das9205 txt	19	50	1	49	1	17280
*das9206 txt	111	120	34	80	1	19518
*das9208 txt	144	102	43	40	1	8060
*das9209 txt	72	108	1	107	1	>5000000
*edf9201 txt	130	182	87	137	1	579720
*edf9202 txt	433	457	137	385	1	130112
*edf9205 txt	141	164	44	123	1	21308
*edf9206 txt	359	239	6	229	1	>4000000
*edfpa15o txt	137	282	26	195	1	>1000000
*edfpa15q txt	157	282	23	195	1	26354
*edfpa15r txt	109	87	57	1	1	26549
*elf9601 txt	241	144	81	57	2	165748
*ftr10 txt	93	174	47	147	1	305
*isp9601 txt	107	142	8	117	1	276785
*isp9602 txt	121	115	14	86	2	>1000000
*isp9603 txt	94	90	23	49	1	3434
*isp9604 txt	131	214	11	195	1	428808
*isp9605 txt	64	31	3	0	1	5630
*isp9606 txt	40	88	14	71	1	1776
*isp9607 txt	64	73	9	48	1	150436

Table A 6: Complexity of test fault trees, 'large' trees

FT name	0 scheme	1 scheme	2 scheme	3 scheme	4 scheme	5 scheme	6 scheme	7 scheme	8 scheme	Minimum
astolfo gsq	23	19	19	19	19	19	19	19	19	19
benjam gsq	102	71	71	71	75	71	71	71	71	71
bpfig03 dat	1	1	1	1	1	1	1	1	1	1
bpfen05 dat	1	1	1	1	1	1	1	1	1	1
bpfig05 dat	1	1	1	1	1	1	1	1	1	1
bpfin05 dat	1	1	1	1	1	1	1	1	1	1
bpfs02 gsq	26	20	20	20	20	20	20	20	20	20
dre1019 dat	1	1	1	1	1	1	1	1	1	1
dre1032 dat	1	1	1	1	1	1	1	1	1	1
dre1057 dat	1	1	1	1	1	1	1	1	1	1
dre1058 dat	33	27	27	27	27	27	27	27	27	27
dre1059 dat	256	220	220	220	220	220	220	220	220	220
dresden dat	41	35	35	35	35	35	35	35	35	35
hpsif02 dat	115	115	115	115	115	115	115	115	115	115
hpsif03 dat	15	14	14	14	14	14	14	14	14	14
hpsif21 gsq	29	27	27	27	27	27	27	27	26	26
hpsif36 gsq	15	13	13	13	13	13	13	13	13	13
jdtree3 gsq	1	1	1	1	1	1	1	1	1	1
jdtree4 gsq	1	1	1	1	1	1	1	1	1	1
jdtree5 gsq	1	1	1	1	1	1	1	1	1	1
khictre dat	19	19	19	19	19	19	19	19	19	19
nakashi gsq	162	162	162	162	168	162	168	162	162	162
trials1 gsq	164	172	173	173	173	172	172	173	162	162
trials4 gsq	169	149	149	149	149	148	153	147	149	147
random3 gsq	63	59	59	59	59	59	59	59	59	59
random6 gsq	3158	1607	1627	1627	1582	1625	1625	1625	1625	1582
random8 gsq	1	1	1	1	1	1	1	1	1	1
rando12 gsq	468	446	433	433	446	433	485	446	433	433
rando13 gsq	100	95	95	95	95	94	94	94	94	94
rando16 gsq	76	59	59	59	59	56	56	56	53	53
rando18 gsq	103	69	69	69	69	53	53	53	53	53
rando23 gsq	1	1	1	1	1	1	1	1	1	1
rando25 gsq	1	1	1	1	1	1	1	1	1	1
rando27 gsq	22	17	17	17	17	13	13	17	13	13
rando28 gsq	1	1	1	1	1	1	1	1	1	1
rando29 gsq	87	87	87	88	87	87	87	88	87	87
rando30 gsq	91	84	84	84	84	84	84	84	84	84
rando31 gsq	1	1	1	1	1	1	1	1	1	1
rando33 gsq	8	8	8	8	8	8	8	8	8	8
rando34 gsq	47	22	22	22	22	22	22	22	22	22
rando35 gsq	22	18	18	18	18	18	18	18	18	18
rando36 gsq	1	1	1	1	1	1	1	1	1	1
rando37 gsq	49	39	39	39	39	38	38	39	38	38
rando38 gsq	1	1	1	1	1	1	1	1	1	1
rando39 gsq	193	162	172	172	162	167	162	185	162	162
rando40 gsq	21	21	21	21	21	21	21	21	21	21
rando42 gsq	1	1	1	1	1	1	1	1	1	1
rando43 gsq	1	1	1	1	1	1	1	1	1	1
rando44 gsq	345	338	338	338	338	338	338	338	338	338
rando45 gsq	41	30	30	30	30	30	30	30	30	30
rando46 gsq	1	1	1	1	1	1	1	1	1	1
rando47 gsq	40	38	38	38	38	38	38	38	38	38
rando48 gsq	14	13	14	14	13	13	13	13	13	13
rando52 gsq	44	42	42	42	42	42	42	44	42	42

Table A.7. Top-down, number of nodes, small trees, 1

FT name	0 scheme	1 scheme	2 scheme	3 scheme	4 scheme	5 scheme	6 scheme	7 scheme	8 scheme	Minimum
rando53 gsq	1	1	1	1	1	1	1	1	1	1
rando54 gsq	29	25	25	25	25	25	25	25	25	25
rando55 gsq	15	14	14	14	14	14	14	14	14	14
rando58 gsq	1	1	1	1	1	1	1	1	1	1
rando59 gsq	1233	607	722	722	672	588	672	658	579	579
rando60 gsq	17	17	17	17	17	17	17	17	17	17
rando61 gsq	25	19	19	19	19	19	19	19	19	19
rando62 gsq	1	1	1	1	1	1	1	1	1	1
rando63 gsq	15	14	14	14	14	14	14	14	14	14
rando64 gsq	102	71	71	71	71	71	71	71	71	71
rando65 gsq	23	23	23	23	23	23	23	23	23	23
rando66 gsq	22	22	22	22	22	22	22	22	22	22
rando70 gsq	40	33	33	33	33	33	33	33	33	33
rando73 gsq	1	1	1	1	1	1	1	1	1	1
rando75 gsq	1	1	1	1	1	1	1	1	1	1
rando76 gsq	14	14	14	14	14	14	14	14	14	14
rando77 gsq	41	41	41	41	41	41	41	41	41	41
rando78 gsq	1	1	1	1	1	1	1	1	1	1
rando80 gsq	9	9	9	9	9	9	9	9	9	9
rando83 gsq	23	23	23	23	23	23	23	23	23	23
rando84 gsq	141	82	82	82	82	82	82	82	82	82
rando85 gsq	1	1	1	1	1	1	1	1	1	1
rando87 gsq	1	1	1	1	1	1	1	1	1	1
rando88 gsq	23	23	23	23	23	20	23	20	20	20
rando89 gsq	1	1	1	1	1	1	1	1	1	1
rando91 gsq	1211	932	943	943	943	916	906	943	906	906
rando92 gsq	392	281	281	281	281	281	281	281	281	281
rando93 gsq	15	15	15	15	15	15	15	15	15	15
rando95 gsq	41	35	33	35	33	35	35	35	35	33
rando98 gsq	167	136	164	164	164	164	164	164	164	136
rando99 gsq	69	58	58	58	58	58	58	58	58	58
rand100 gsq	1	1	1	1	1	1	1	1	1	1
rand103 gsq	31	22	22	22	22	22	22	22	22	22
rand104 gsq	1	1	1	1	1	1	1	1	1	1
rand105 gsq	26	25	25	25	25	25	25	26	25	25
rand106 gsq	1	1	1	1	1	1	1	1	1	1
rand108 gsq	124	103	105	111	103	103	103	111	103	103
rand109 gsq	252	171	171	171	171	171	171	171	171	171
rand110 gsq	6	6	6	6	6	6	6	6	6	6
rand111 gsq	59	39	39	39	39	37	37	36	37	36
rand115 gsq	123	101	102	102	101	101	102	102	101	101
rand116 gsq	70	70	70	70	70	70	70	70	70	70
rand117 gsq	10	10	10	10	10	10	10	10	10	10
rand118 gsq	141	82	82	82	82	82	82	82	82	82
rand119 gsq	1	1	1	1	1	1	1	1	1	1
rand120 gsq	172	158	168	158	158	158	158	168	158	158
rand121 gsq	23	23	23	23	23	23	23	23	23	23
rand123 gsq	7	5	5	5	5	5	5	5	5	5
rand124 gsq	1	1	1	1	1	1	1	1	1	1
rand125 gsq	6	6	6	6	6	6	6	6	6	6
rand126 gsq	76	56	56	56	56	60	55	61	60	55
rand127 gsq	1	1	1	1	1	1	1	1	1	1
rand128 gsq	73	64	67	64	64	64	64	64	67	64
rand129 gsq	1	1	1	1	1	1	1	1	1	1

Table A 8 Top-down, number of nodes, small trees, 2

FT name	0 scheme	1 scheme	2 scheme	3 scheme	4 scheme	5 scheme	6 scheme	7 scheme	8 scheme	Minimum
rand130 gsq	1	1	1	1	1	1	1	1	1	1
rand132 gsq	904	704	768	768	719	746	671	704	713	671
rand134 gsq	53	50	50	50	50	50	50	50	50	50
rand135 gsq	60	54	54	54	54	54	54	54	54	54
rand137 gsq	60	35	35	35	35	36	36	35	36	35
rand138 gsq	1	1	1	1	1	1	1	1	1	1
rand139 gsq	144	117	117	117	117	117	117	117	117	117
rand141 gsq	6	6	6	6	6	6	6	6	6	6
rand142 gsq	5885	2875	2801	2801	2488	2523	2498	3548	2814	2488
rand143 gsq	1	1	1	1	1	1	1	1	1	1
rand144 gsq	64	63	63	63	63	62	62	63	62	62
rand145 gsq	11	9	9	9	9	9	9	9	9	9
rand146 gsq	60	35	35	35	35	36	36	35	36	35
rand147 gsq	584	407	460	442	442	357	427	370	355	355
rand148 gsq	1	1	1	1	1	1	1	1	1	1
rand149 gsq	1	1	1	1	1	1	1	1	1	1
rand150 gsq	4796	2557	2623	2557	2551	2521	2521	2718	2635	2521
rand151 gsq	1	1	1	1	1	1	1	1	1	1
rand153 gsq	1	1	1	1	1	1	1	1	1	1
rand154 gsq	1	1	1	1	1	1	1	1	1	1
rand155 gsq	367	226	242	226	226	226	226	226	226	226
rand156 gsq	1	1	1	1	1	1	1	1	1	1
rand158 gsq	1	1	1	1	1	1	1	1	1	1
lisaba1 gsq	49	45	45	45	45	45	45	45	45	45
lisaba3 gsq	1738	706	739	739	706	706	706	706	739	706
lisaba4 gsq	971	606	618	613	625	625	637	596	625	596
lisaba5 gsq	124	106	103	103	106	103	103	106	103	103
lisaba6 gsq	116	85	85	85	85	85	85	85	85	85
lisaba7 gsq	49	45	45	45	45	45	45	45	45	45
lisaba9 gsq	16	14	14	14	14	14	14	14	14	14
lisab10 gsq	1961	1375	2019	1858	1363	1395	1395	1375	1516	1363
lisab11 gsq	1	1	1	1	1	1	1	1	1	1
lisab13 gsq	6	6	6	6	6	6	6	6	6	6
lisab17 gsq	49	45	45	45	45	45	45	45	45	45
lisab19 gsq	1	1	1	1	1	1	1	1	1	1
lisab20 gsq	1	1	1	1	1	1	1	1	1	1
lisab24 gsq	1	1	1	1	1	1	1	1	1	1
lisab25 gsq	74	51	52	51	51	51	51	52	51	51
lisab27 gsq	277	225	229	229	229	225	225	225	225	225
lisab28 gsq	1	1	1	1	1	1	1	1	1	1
lisab30 gsq	29	25	25	25	25	25	25	25	25	25
lisab31 gsq	731	507	507	507	507	507	507	507	507	507
lisab34 gsq	25	23	25	23	25	23	23	23	23	23
lisab35 gsq	407	452	388	452	452	391	458	388	400	388
lisab36 gsq	312	186	186	186	186	178	186	206	160	160
lisab37 gsq	13	11	11	11	11	11	11	11	11	11
lisab42 gsq	1	1	1	1	1	1	1	1	1	1
lisab44 gsq	23	19	19	19	19	19	19	19	19	19
lisab46 gsq	1	1	1	1	1	1	1	1	1	1
lisab47 gsq	7	7	7	7	7	7	7	7	7	7
lisab48 gsq	1	1	1	1	1	1	1	1	1	1
lisab49 gsq	1	1	1	1	1	1	1	1	1	1
lisab50 gsq	1	1	1	1	1	1	1	1	1	1
lisab51 gsq	18	16	16	16	16	16	16	16	16	16

Table A.9 Top-down, number of nodes, small trees, 3

FT name	0 scheme	1 scheme	2 scheme	3 scheme	4 scheme	5 scheme	6 scheme	7 scheme	8 scheme	Minimum
lisab52 gsq	1109	712	717	717	712	689	685	712	689	685
lisab53 gsq	1	1	1	1	1	1	1	1	1	1
lisab54 gsq	20	19	19	19	19	19	19	18	19	18
lisab56 gsq	1	1	1	1	1	1	1	1	1	1
lisab57 gsq	396	332	364	340	357	356	356	357	356	332
lisab59 gsq	1	1	1	1	1	1	1	1	1	1
lisab60 gsq	25	21	21	21	21	21	21	21	21	21
lisab61 gsq	25	25	25	25	25	25	25	25	25	25
lisab62 gsq	14	14	14	14	14	14	14	14	14	14
lisab63 gsq	9	8	8	8	8	8	8	8	8	8
lisab64 gsq	1	1	1	1	1	1	1	1	1	1
lisab65 gsq	1	1	1	1	1	1	1	1	1	1
lisab66 gsq	37	37	37	37	37	37	37	37	37	37
lisab67 gsq	328	278	278	278	278	278	278	278	278	278
lisab68 gsq	1	1	1	1	1	1	1	1	1	1
lisab69 gsq	23	21	21	21	21	21	21	21	21	21
lisab70 gsq	157	143	143	143	143	143	143	143	143	143
lisab71 gsq	1	1	1	1	1	1	1	1	1	1
lisab72 gsq	140	131	131	131	131	130	131	131	130	130
lisab73 gsq	1	1	1	1	1	1	1	1	1	1
lisab74 gsq	35277	23209	23209	23209	23209	23209	23209	24777	23915	23209
lisab75 gsq	1	1	1	1	1	1	1	1	1	1
lisab77 gsq	232	210	205	205	205	210	210	210	210	205
lisab78 gsq	307	241	245	241	241	212	241	245	241	212
lisab81 gsq	1	1	1	1	1	1	1	1	1	1
lisab82 gsq	1107	880	880	880	880	880	880	880	880	880
lisab83 gsq	84	80	80	80	80	60	60	64	73	60
lisab84 gsq	84	80	80	80	80	60	60	64	73	60
lisab86 gsq	2343	681	982	974	982	752	677	974	752	677
lisab88 gsq	13	11	11	11	11	11	11	11	11	11
lisab89 gsq	100	92	92	92	92	92	92	92	92	92
lisab91 gsq	297	224	224	224	224	224	224	224	224	224
lisab95 gsq	1	1	1	1	1	1	1	1	1	1
lisab97 gsq	1	1	1	1	1	1	1	1	1	1
lisab98 gsq	1	1	1	1	1	1	1	1	1	1
lisa100 gsq	2507	1462	1524	1524	1399	1462	1505	1524	1462	1399
lisa103 gsq	1	1	1	1	1	1	1	1	1	1
lisa104 gsq	1	1	1	1	1	1	1	1	1	1
lisa107 gsq	1	1	1	1	1	1	1	1	1	1
lisa109 gsq	39	38	38	38	38	38	38	38	38	38
lisa110 gsq	792	482	530	530	530	482	482	562	468	468
lisa111 gsq	1	1	1	1	1	1	1	1	1	1
lisa112 gsq	485	313	392	392	392	313	313	313	313	313
lisa113 gsq	295	189	210	210	210	189	189	187	187	187
lisa115 gsq	15	12	12	12	12	12	12	12	12	12
lisa116 gsq	6	6	6	6	6	6	6	6	6	6
lisa119 gsq	6	6	6	6	6	6	6	6	6	6
lisa120 gsq	1	1	1	1	1	1	1	1	1	1
lisa121 gsq	12	12	12	12	12	12	12	12	12	12
lisa122 gsq	41	28	28	28	28	28	28	28	28	28
lisa123 gsq	148	145	145	145	145	133	133	133	133	133
lisa124 gsq	3587	1505	1943	1943	1532	1439	1385	1783	1385	1385
rand159 gsq	42	41	41	41	41	41	41	41	41	41
rand161 gsq	228	220	243	223	220	220	220	220	220	220

Table A 10 Top-down, number of nodes, small trees, 4

FT name	0 scheme	1 scheme	2 scheme	3 scheme	4 scheme	5 scheme	6 scheme	7 scheme	8 scheme	Minimum
rand163 gsq	982	881	901	901	901	875	894	880	862	862
rand164 gsq	841	693	767	767	723	723	723	767	723	693
rand165 gsq	8181	4130	4130	4130	4130	3578	3578	3578	3578	3578
rand166 gsq	8911	4839	6735	6751	4741	4945	4741	6735	5725	4741
rand167 gsq	264	221	221	221	221	219	219	221	219	219

Table A 11: Top-down, number of nodes, small trees, 5

FT name	0 scheme	1 scheme	2 scheme	3 scheme	4 scheme	5 scheme	6 scheme	7 scheme	8 scheme	Minimum
rando11 gsq	294803	178598	203126	203126	210517	167418	184239	173963	179261	167418
rando19 gsq	3916	3311	3261	3261	3216	3320	3214	3437	3430	3214
rando20 gsq	1283	941	1083	1083	1015	872	940	1083	872	872
rando22 gsq	2212	1931	1952	1952	2040	1774	2040	2059	1798	1774
rand160 gsq	1647432	427504	435379	433293	412969	362422	350797	427016	373863	350797
lisaba2 gsq	400351	250962	328032	328032	266526	267483	267483	287221	267483	250962
lisaba8 gsq	288549	117248	137956	137956	126202	119548	117120	130120	124730	117120
lisab14 gsq	16333	10078	13337	13337	9163	9609	9163	9163	8224	8224
lisab15 gsq	8761	6054	6559	6559	6559	5858	5858	6559	5858	5858
lisab22 gsq	8214	4416	5509	5509	5851	4173	5287	3855	4495	3855
lisab29 gsq	1	1	1	1	1	1	1	1	1	1
lisab76 gsq	5399	5022	5248	5248	5035	5033	5035	5522	5282	5022
lisab87 gsq	921517	405965	407181	407181	422473	354915	372275	417433	394065	354915
lisab92 gsq	1	1	1	1	1	1	1	1	1	1
lisa102 gsq	146804	102181	102181	102181	100604	98942	98602	100649	97694	97694
lisa117 gsq	6446409	3888036	3927624	4107065	3885836	3700777	3886170	3718517	3906491	3700777
lisa118 gsq	11397	8990	10374	10374	10374	8990	9400	9772	8990	8990
baobab1 txt	-	-	-	-	-	-	-	-	-	-
baobab2 txt	-	-	-	-	-	-	-	-	-	-
chinese txt	45	45	45	45	45	45	45	45	45	45
das9201 txt	261	204	210	210	204	204	204	210	204	204
das9202 txt	1	1	1	1	1	1	1	1	1	1
das9203 txt	12	11	11	11	11	11	11	11	11	11
das9204 txt	12	12	12	12	12	12	12	12	12	12
das9205 txt	1	1	1	1	1	1	1	1	1	1
das9206 txt	106986	53309	52569	52569	54701	54701	54701	53596	52766	52569
das9208 txt	-	-	-	-	-	-	-	-	-	-
das9209 txt	1	1	1	1	1	1	1	1	1	1
edf9201 txt	6456	4306	4509	4526	4418	4283	4282	4526	7999	4282
edf9202 txt	-	-	-	-	-	-	-	-	-	-
edf9205 txt	14417	10119	10119	10119	10119	10119	10105	10073	10119	10073
edf9206 txt	68	68	68	68	68	68	68	68	68	68
edfpa15o txt	-	-	-	-	-	-	-	-	-	-
edfpa15q txt	-	-	-	-	-	-	-	-	-	-
edfpa15r txt	-	-	-	-	-	-	-	-	-	-
elf9601 txt	-	-	-	-	-	-	-	-	-	-
ftr10 txt	612	584	584	584	584	584	584	584	584	584
isp9601 txt	2134	1237	1237	1237	1237	1237	1237	1237	1237	1237
isp9602 txt	1955	1803	1837	1837	1826	1858	1848	1794	1875	1794
isp9603 txt	290275	168160	163490	163490	171798	177212	162083	159272	179040	159272
isp9604 txt	283	251	276	276	260	248	273	254	248	248
isp9605 txt	-	-	-	-	-	-	-	-	-	-
isp9606 txt	188	163	163	163	163	163	163	163	163	163
isp9607 txt	1304	488	477	477	477	477	477	477	477	477

Table A.12: Top-down, number of nodes, 'large' trees

FT name	0 scheme	1 scheme	2 scheme	3 scheme	4 scheme	5 scheme	6 scheme	7 scheme	8 scheme	Minimum
astolfo gsq	0 016	0 015	0	0 016	0	0 015	0 015	0	0	0
benjam gsq	0 016	0 015	0 015	0 015	0 016	0 016	0 015	0	0 016	0
bpfig03 dat	0 016	0	0 015	0 016	0	0	0 015	0	0	0
bpfen05 dat	0	0	0	0	0	0	0 015	0	0 016	0
bpfig05 dat	0	0	0	0	0	0	0 015	0	0 015	0
bpfin05 dat	0 016	0	0	0	0	0	0	0	0	0
bpfs02 gsq	0 016	0 016	0 016	0 015	0 016	0 015	0	0 016	0	0
dre1019 dat	0	0	0 015	0	0 016	0 015	0	0	0	0
dre1032 dat	0 015	0	0	0	0	0	0 015	0	0	0
dre1057 dat	0	0	0	0	0	0	0 016	0	0	0
dre1058 dat	0 016	0 015	0 016	0	0 016	0 016	0 016	0 015	0 016	0
dre1059 dat	0 031	0 016	0 015	0 016	0 016	0 016	0 015	0 016	0 015	0 015
dresden dat	0 016	0 016	0 016	0 016	0 016	0 031	0 016	0 016	0 015	0 015
hpsif02 dat	0 015	0	0 016	0 016	0 015	0 016	0 015	0 016	0	0
hpsif03 dat	0	0 016	0	0	0	0 016	0	0 016	0	0
hpsif21 gsq	0 015	0	0 016	0 015	0 016	0 016	0	0	0 016	0
hpsif36 gsq	0	0 015	0	0	0	0	0 015	0 016	0	0
jdtree3 gsq	0	0	0	0	0	0	0	0	0 015	0
jdtree4 gsq	0	0 016	0	0	0	0 016	0	0 016	0	0
jdtree5 gsq	0	0	0	0 016	0 016	0	0 015	0	0 015	0
khictre dat	0 016	0 016	0 015	0 016	0 016	0 016	0 016	0 015	0 016	0 015
nakashi gsq	0 015	0 016	0 016	0 015	0 016	0 031	0 016	0 015	0 015	0 015
trials1 gsq	0 016	0 016	0 015	0 015	0 016	0 015	0 016	0 016	0 016	0 015
trials4 gsq	0 047	0 047	0 031	0 032	0 031	0 031	0 047	0 047	0 047	0 031
random3 gsq	0 015	0 015	0	0	0	0 015	0 016	0 016	0 016	0
random6 gsq	0 344	0 328	0 313	0 328	0 312	0 344	0 344	0 328	0 375	0 312
random8 gsq	0	0	0	0	0	0	0 015	0	0	0
rando12 gsq	0 281	0 266	0 266	0 266	0 25	0 266	0 25	0 234	0 265	0 234
rando13 gsq	0	0 016	0 016	0	0	0	0 016	0	0 016	0
rando16 gsq	0 015	0 015	0 015	0 032	0 031	0 016	0 031	0 016	0 015	0 015
rando18 gsq	0 016	0 015	0	0	0 016	0	0 016	0 016	0 016	0
rando23 gsq	0	0	0	0	0	0 016	0 015	0	0	0
rando25 gsq	0 016	0	0	0	0 016	0	0 015	0 015	0 016	0
rando27 gsq	0 015	0	0 016	0	0	0	0	0 015	0 015	0
rando28 gsq	0	0	0 016	0	0	0	0	0	0	0
rando29 gsq	0 015	0	0 016	0 016	0 016	0 031	0 016	0	0 015	0
rando30 gsq	0	0 016	0 016	0	0 016	0 016	0	0 015	0 016	0
rando31 gsq	0	0 015	0 016	0 015	0 015	0 016	0 015	0	0	0
rando33 gsq	0 015	0 016	0 016	0 016	0 016	0 015	0 015	0 015	0 016	0 015
rando34 gsq	0 015	0 016	0	0 016	0	0 016	0 016	0 016	0 016	0
rando35 gsq	0 016	0 016	0 015	0 015	0	0 016	0 016	0 016	0 015	0
rando36 gsq	0	0	0	0	0	0 016	0	0 016	0	0
rando37 gsq	0	0	0 015	0 015	0 015	0 016	0 015	0 016	0	0
rando38 gsq	0 015	0 015	0	0	0	0	0 015	0	0 016	0
rando39 gsq	0 016	0 015	0	0 016	0 015	0 015	0 016	0 015	0 015	0
rando40 gsq	0 016	0 016	0	0	0	0 016	0 016	0 016	0 016	0
rando42 gsq	0 015	0	0	0	0 016	0	0	0	0	0
rando43 gsq	0	0 016	0	0	0	0	0	0 015	0	0
rando44 gsq	0 031	0 031	0 031	0 016	0 031	0 031	0 047	0 015	0 031	0 015
rando45 gsq	0 016	0 016	0 015	0	0 016	0 015	0 016	0	0 015	0
rando46 gsq	0	0	0	0	0	0	0	0	0 015	0
rando47 gsq	0 016	0 016	0	0 015	0	0 015	0	0	0 015	0
rando48 gsq	0	0	0	0	0	0	0	0	0 015	0
rando52 gsq	0 032	0 016	0 032	0 016	0 032	0 032	0 031	0 015	0 015	0 015

Table A.13: Top-down, time, small trees, 1

FT name	0 scheme	1 scheme	2 scheme	3 scheme	4 scheme	5 scheme	6 scheme	7 scheme	8 scheme	Minimum
rando53 gsq	0	0	0	0	0	0	0	0	0 016	0
rando54 gsq	0 031	0 016	0 031	0 015	0 031	0 032	0 047	0 031	0 016	0 015
rando55 gsq	0 015	0 016	0 015	0 015	0 015	0 016	0 015	0 016	0	0
rando58 gsq	0 016	0 016	0	0 016	0	0 015	0 015	0	0	0
rando59 gsq	0 047	0 047	0 047	0 031	0 047	0 032	0 031	0 031	0 031	0 031
rando60 gsq	0 016	0 016	0 016	0 032	0 016	0 031	0 016	0 015	0 016	0 015
rando61 gsq	0 016	0 016	0	0	0	0 015	0	0 015	0 016	0
rando62 gsq	0 015	0	0	0 016	0	0 016	0	0	0	0
rando63 gsq	0 015	0	0 016	0 016	0 015	0 016	0 031	0	0 016	0
rando64 gsq	0 016	0 016	0 016	0 016	0 016	0 016	0 016	0 015	0 016	0 015
rando65 gsq	0	0	0	0 015	0 016	0 016	0	0	0	0
rando66 gsq	0 015	0 016	0 016	0 016	0 016	0 015	0	0	0	0
rando70 gsq	0 015	0 016	0 016	0	0	0 015	0	0	0	0
rando73 gsq	0	0	0	0 015	0	0	0	0	0 015	0
rando75 gsq	0	0 016	0 016	0	0	0	0 016	0	0	0
rando76 gsq	0 015	0 016	0 016	0 016	0 015	0 015	0 031	0 016	0	0
rando77 gsq	0 016	0 015	0 016	0 016	0 015	0 016	0 016	0 015	0 016	0 015
rando78 gsq	0 015	0	0	0	0	0	0	0 016	0 015	0
rando80 gsq	0 016	0	0	0 015	0 015	0 016	0 015	0 016	0 015	0
rando83 gsq	0	0 015	0	0	0 015	0 016	0	0	0	0
rando84 gsq	0 016	0 016	0 016	0 016	0 016	0 016	0 015	0 015	0 016	0 015
rando85 gsq	0	0	0	0	0 016	0 016	0	0	0	0
rando87 gsq	0	0	0 016	0	0	0	0 016	0	0 016	0
rando88 gsq	0	0	0	0	0	0	0	0	0	0
rando89 gsq	0 015	0	0	0 015	0	0	0	0 016	0	0
rando91 gsq	0 203	0 187	0 141	0 157	0 141	0 109	0 124	0 141	0 125	0 109
rando92 gsq	0 031	0 032	0 032	0 031	0 031	0 031	0 032	0 032	0 047	0 031
rando93 gsq	0 016	0 016	0 015	0 016	0 015	0 015	0 015	0 016	0 015	0 015
rando95 gsq	0 016	0 016	0 016	0	0 015	0 015	0 015	0 016	0 015	0
rando98 gsq	0 016	0 016	0 016	0 015	0 016	0 016	0 031	0 032	0 015	0 015
rando99 gsq	0 016	0 016	0 015	0	0 016	0	0	0 016	0	0
rand100 gsq	0	0	0	0 015	0	0	0	0	0	0
rand103 gsq	0	0 016	0	0 016	0	0 015	0 016	0 016	0 016	0
rand104 gsq	0	0	0	0	0	0 016	0	0	0	0
rand105 gsq	0	0 015	0	0 016	0 016	0	0	0 016	0	0
rand106 gsq	0 016	0	0 015	0	0	0 031	0	0	0 015	0
rand108 gsq	0 016	0 031	0 015	0 015	0 016	0 016	0 016	0 016	0 015	0 015
rand109 gsq	0 031	0 015	0 016	0 015	0 016	0 016	0 032	0 016	0 016	0 015
rand110 gsq	0	0	0	0 015	0 016	0 016	0	0	0	0
rand111 gsq	0 016	0	0	0	0	0 016	0 015	0 015	0 016	0
rand115 gsq	0 015	0 016	0 016	0 016	0 016	0 016	0 047	0	0 015	0
rand116 gsq	0 016	0 016	0 016	0 016	0 016	0 031	0 031	0 016	0 016	0 016
rand117 gsq	0 015	0	0	0	0	0	0	0	0	0
rand118 gsq	0 015	0 015	0	0 016	0 016	0 016	0 031	0 015	0 015	0
rand119 gsq	0	0	0	0 016	0	0 016	0	0 015	0	0
rand120 gsq	0 015	0 016	0 015	0 016	0 015	0 016	0 015	0	0 015	0
rand121 gsq	0	0 016	0	0 016	0	0 016	0	0 016	0	0
rand123 gsq	0	0	0	0	0	0	0	0	0	0
rand124 gsq	0	0	0	0 016	0 016	0 016	0 016	0	0	0
rand125 gsq	0	0	0	0	0	0	0	0	0	0
rand126 gsq	0	0 015	0 015	0	0	0 015	0	0 016	0 016	0
rand127 gsq	0	0 015	0	0 016	0	0	0 015	0	0	0
rand128 gsq	0	0	0 016	0 015	0 015	0 015	0 015	0	0 016	0
rand129 gsq	0	0 015	0 016	0	0 015	0 016	0 015	0	0 016	0

Table A 14: Top-down, time, small trees, 2

FT name	0 scheme	1 scheme	2 scheme	3 scheme	4 scheme	5 scheme	6 scheme	7 scheme	8 scheme	Minimum
rand130 gsq	0 015	0	0	0 015	0	0 016	0 016	0	0	0
rand132 gsq	0 265	0 25	0 203	0 219	0 219	0 219	0 219	0 203	0 203	0 203
rand134 gsq	0	0 015	0 016	0 016	0 016	0 016	0 016	0 016	0 016	0
rand135 gsq	0	0	0	0 016	0	0	0	0 016	0	0
rand137 gsq	0	0 016	0 015	0	0	0	0	0 015	0 015	0
rand138 gsq	0 015	0	0	0	0	0 016	0	0	0	0
rand139 gsq	0 015	0	0 016	0 016	0	0 016	0 015	0 016	0 016	0
rand141 gsq	0 015	0 016	0	0	0	0	0	0	0	0
rand142 gsq	0 687	0 578	0 359	0 359	0 359	0 375	0 39	0 469	0 375	0 359
rand143 gsq	0	0	0	0	0	0	0 015	0	0	0
rand144 gsq	0 016	0 015	0 016	0 015	0 062	0 016	0 016	0	0 016	0
rand145 gsq	0 016	0 016	0 016	0	0 015	0 015	0 016	0 016	0	0
rand146 gsq	0	0	0	0	0 047	0 016	0 015	0	0	0
rand147 gsq	0 078	0 078	0 063	0 062	0 11	0 047	0 063	0 047	0 063	0 047
rand148 gsq	0	0	0	0	0	0	0	0	0	0
rand149 gsq	0	0	0	0	0	0 016	0	0	0	0
rand150 gsq	0 344	0 328	0 235	0 235	0 281	0 297	0 281	0 313	0 297	0 235
rand151 gsq	0	0	0	0	0 016	0	0 016	0	0	0
rand153 gsq	0	0	0	0	0 016	0	0	0 016	0 016	0
rand154 gsq	0	0	0 016	0	0	0 016	0	0	0	0
rand155 gsq	0 015	0 016	0 016	0 016	0 016	0 016	0 016	0 031	0 016	0 015
rand156 gsq	0	0	0	0	0	0	0 016	0	0	0
rand158 gsq	0	0	0	0	0 015	0 016	0	0	0	0
lisaba1 gsq	0 015	0 031	0 015	0 016	0 047	0 032	0 031	0 015	0 016	0 015
lisaba3 gsq	0 328	0 313	0 313	0 312	0 297	0 297	0 297	0 313	0 312	0 297
lisaba4 gsq	0 063	0 047	0 031	0 047	0 032	0 046	0 047	0 031	0 047	0 031
lisaba5 gsq	0 016	0	0 016	0 016	0 063	0 015	0	0 016	0	0
lisaba6 gsq	0 016	0	0	0 015	0	0 015	0	0 015	0 016	0
lisaba7 gsq	0 031	0 016	0 015	0 015	0 016	0 032	0 032	0 015	0 016	0 015
lisaba9 gsq	0	0 015	0 016	0	0	0 016	0 015	0 015	0	0
lisab10 gsq	0 391	0 359	0 422	0 438	0 235	0 203	0 203	0 203	0 265	0 203
lisab11 gsq	0	0 016	0	0	0	0	0	0	0	0
lisab13 gsq	0	0 016	0	0	0 016	0	0	0	0	0
lisab17 gsq	0 016	0 016	0 015	0 015	0 032	0 032	0 031	0 016	0 031	0 015
lisab19 gsq	0 016	0	0	0 016	0	0	0 016	0 016	0 015	0
lisab20 gsq	0	0	0	0	0	0 015	0	0	0	0
lisab24 gsq	0	0	0 015	0 015	0 016	0	0	0	0	0
lisab25 gsq	0	0	0 016	0	0 016	0 016	0	0 016	0 016	0
lisab27 gsq	0 031	0 031	0 031	0 031	0 032	0 047	0 031	0 032	0 031	0 031
lisab28 gsq	0	0 015	0	0	0	0 016	0 015	0 016	0 016	0
lisab30 gsq	0 031	0 016	0 016	0 016	0 015	0 015	0 031	0 031	0 031	0 015
lisab31 gsq	0 078	0 062	0 062	0 063	0 078	0 063	0 078	0 078	0 062	0 062
lisab34 gsq	0 015	0 016	0 016	0 015	0 015	0 016	0 016	0 015	0 016	0 015
lisab35 gsq	0 047	0 031	0 031	0 047	0 032	0 047	0 047	0 031	0 047	0 031
lisab36 gsq	0 031	0 016	0 031	0 031	0 031	0 032	0 031	0 015	0 031	0 015
lisab37 gsq	0 016	0 016	0 015	0 015	0 016	0 016	0 015	0 015	0 015	0 015
lisab42 gsq	0	0	0 016	0	0	0 016	0	0	0 016	0
lisab44 gsq	0	0	0 016	0 015	0 015	0 016	0 016	0 016	0 016	0
lisab46 gsq	0	0	0 016	0	0 015	0 016	0 016	0	0	0
lisab47 gsq	0 016	0	0	0	0	0	0	0	0	0
lisab48 gsq	0	0	0 016	0	0 015	0	0 015	0 016	0 016	0
lisab49 gsq	0	0	0	0	0	0	0	0 016	0 016	0
lisab50 gsq	0	0	0 015	0 015	0 016	0 015	0 016	0	0	0
lisab51 gsq	0 016	0	0	0	0	0	0 015	0	0	0

Table A.15 Top-down, time, small trees, 3

FT name	0 scheme	1 scheme	2 scheme	3 scheme	4 scheme	5 scheme	6 scheme	7 scheme	8 scheme	Minimum
lisab52 gsq	0 125	0 094	0 078	0 078	0 078	0 062	0 079	0 078	0 078	0 062
lisab53 gsq	0	0	0	0	0 016	0 016	0 016	0	0 015	0
lisab54 gsq	0 016	0	0 016	0	0	0 016	0 016	0	0	0
lisab56 gsq	0	0	0	0 016	0 016	0	0 015	0 015	0	0
lisab57 gsq	0 016	0 016	0 016	0 016	0 031	0 031	0 031	0 016	0 015	0 015
lisab59 gsq	0	0	0	0 016	0	0	0	0	0 016	0
lisab60 gsq	0 015	0	0	0	0	0 015	0	0	0	0
lisab61 gsq	0	0 016	0 016	0	0	0	0 016	0	0	0
lisab62 gsq	0	0	0 015	0	0	0	0	0	0	0
lisab63 gsq	0 016	0 016	0	0 016	0	0	0	0	0 016	0
lisab64 gsq	0	0	0	0	0	0	0 016	0 016	0	0
lisab65 gsq	0 015	0 015	0 015	0	0 015	0	0 015	0	0 016	0
lisab66 gsq	0	0	0 015	0 016	0	0 015	0 016	0 015	0 015	0
lisab67 gsq	0 015	0 016	0 016	0 016	0 016	0 016	0 016	0 016	0 016	0 015
lisab68 gsq	0	0	0	0 016	0 016	0 015	0	0	0 015	0
lisab69 gsq	0	0	0	0 016	0 015	0 016	0	0	0	0
lisab70 gsq	0	0 015	0	0	0	0 015	0	0 016	0	0
lisab71 gsq	0	0	0	0	0 016	0	0	0 016	0	0
lisab72 gsq	0	0 015	0 016	0 015	0 016	0 016	0	0 016	0 016	0
lisab73 gsq	0	0 015	0 016	0	0	0 016	0 015	0	0 016	0
lisab74 gsq	2 921	2 266	1 078	1 125	1 094	1 109	1 093	1 125	1 141	1 078
lisab75 gsq	0 016	0	0	0	0	0	0	0	0 016	0
lisab77 gsq	0 031	0 031	0 016	0 032	0	0 016	0 015	0 016	0 016	0
lisab78 gsq	0 032	0 016	0 015	0 016	0 031	0 016	0 031	0 016	0 015	0 015
lisab81 gsq	0 016	0	0	0	0	0 016	0	0 016	0	0
lisab82 gsq	0 047	0 047	0 047	0 031	0 047	0 047	0 032	0 032	0 031	0 031
lisab83 gsq	0 016	0 015	0 016	0 016	0	0 015	0	0	0 016	0
lisab84 gsq	0 016	0 016	0 015	0	0	0 015	0	0	0 015	0
lisab86 gsq	0 078	0 078	0 031	0 047	0 032	0 047	0 047	0 031	0 047	0 031
lisab88 gsq	0	0	0	0 016	0	0 016	0	0	0	0
lisab89 gsq	0 016	0	0 016	0	0	0 015	0 016	0 016	0 016	0
lisab91 gsq	0 016	0 016	0 015	0 015	0 016	0 031	0 016	0 015	0 015	0 015
lisab95 gsq	0 016	0	0	0	0 015	0	0	0	0	0
lisab97 gsq	0	0	0 015	0	0	0 016	0 016	0	0	0
lisab98 gsq	0	0	0	0 015	0	0	0 016	0	0	0
lisa100 gsq	0 75	0 813	0 172	0 188	0 172	0 171	0 172	0 171	0 157	0 157
lisa103 gsq	0	0 015	0	0	0	0	0	0	0	0
lisa104 gsq	0 016	0	0 016	0	0	0 015	0 016	0 016	0	0
lisa107 gsq	0	0 016	0	0	0	0 016	0	0 015	0	0
lisa109 gsq	0	0 015	0 015	0 015	0 016	0	0	0 015	0	0
lisa110 gsq	0 11	0 11	0 063	0 063	0 047	0 063	0 063	0 062	0 062	0 047
lisa111 gsq	0	0	0	0	0 015	0 015	0 016	0	0	0
lisa112 gsq	0 031	0 031	0 032	0 016	0 031	0 031	0 032	0 016	0 031	0 016
lisa113 gsq	0 032	0 016	0 031	0 015	0 016	0 016	0 031	0 031	0 016	0 015
lisa115 gsq	0 016	0	0	0 015	0 016	0	0	0 015	0	0
lisa116 gsq	0	0 015	0	0	0 016	0 015	0	0	0 093	0
lisa119 gsq	0 015	0	0	0	0 015	0 016	0	0	0	0
lisa120 gsq	0	0	0	0 016	0 016	0 016	0 015	0 015	0	0
lisa121 gsq	0 015	0 016	0 016	0 016	0 016	0 032	0 015	0 016	0 015	0 015
lisa122 gsq	0 016	0 015	0	0 016	0	0 016	0 016	0	0	0
lisa123 gsq	0 032	0 015	0 031	0 015	0 015	0 016	0 016	0 015	0 016	0 015
lisa124 gsq	0 703	0 141	0 125	0 125	0 14	0 14	0 14	0 125	0 11	0 11
rand159 gsq	0 016	0 015	0 016	0 015	0 016	0 015	0 015	0 015	0	0
rand161 gsq	0 031	0 016	0 015	0 016	0 016	0 016	0 016	0 015	0 016	0 015

Table A 16 Top-down, time, small trees, 4

FT name	0 scheme	1 scheme	2 scheme	3 scheme	4 scheme	5 scheme	6 scheme	7 scheme	8 scheme	Minimum
rand163 gsq	0 078	0 078	0 063	0 062	0 047	0 078	0 062	0 063	0 063	0 047
rand164 gsq	0 031	0 047	0 031	0 218	0 063	0 032	0 032	0 031	0 031	0 031
rand165 gsq	2 781	2 625	2 297	2 265	2 25	1 656	1 297	1 875	1 328	1 297
rand166 gsq	5 234	4 312	3 969	2 969	1 594	2 719	2 078	2 985	2 031	1 594
rand167 gsq	0 031	0 016	0 031	0 015	0 015	0 031	0 016	0 032	0 031	0 015

Table A 17: Top-down, time, small trees, 5

FT name	0 scheme	1 scheme	2 scheme	3 scheme	4 scheme	5 scheme	6 scheme	7 scheme	8 scheme	Minimum
rando11 gsq	22 21	15 17	15 04	14 72	15 9	13 49	14 62	14 36	14 65	13 49
rando19 gsq	0 18	0 18	0 15	0 15	0 16	0 17	0 14	0 17	0 3	0 14
rando20 gsq	0 06	0 13	0 05	0 66	0 04	0 07	0 07	0 04	0 07	0 04
rando22 gsq	0 83	0 4	0 39	0 27	0 58	0 38	0 28	0 34	0 23	0 23
rand160 gsq	58 21	10 92	11 68	11 51	12 26	12 19	12 24	11 48	11 13	10 92
lisaba2 gsq	3 59	2	2 28	2 28	2 4	2 39	2 39	2 27	2 45	2
lisaba8 gsq	2 01	1 04	1 25	1 25	1 16	1 05	1 09	1 2	1 15	1 04
lisab14 gsq	0 55	0 26	0 37	0 38	0 27	0 26	0 26	0 27	0 24	0 24
lisab15 gsq	1 2	0 65	0 66	0 66	0 67	0 66	0 65	0 67	0 66	0 65
lisab22 gsq	1 06	0 37	0 4	0 41	0 55	0 34	0 47	0 27	0 39	0 27
lisab29 gsq	0	0	0	0	0	0	0	0	0	0
lisab76 gsq	0 34	0 21	0 24	0 22	0 21	0 22	0 22	0 23	0 22	0 21
lisab87 gsq	101 3	78 67	73 26	75 72	72 06	50 4	52 12	72 24	56 23	50 4
lisab92 gsq	0	0	0	0	0	0	0	0	0	0
lisa102 gsq	1 06	0 68	0 76	0 7	0 72	0 69	0 7	0 75	0 71	0 68
lisa117 gsq	200 3	94 8	95 23	100 47	129 14	74 48	133 11	83 23	75 08	74 48
lisa118 gsq	0 1	0 09	0 1	0 1	0 1	0 09	0 09	0 08	0 08	0 08
baobab1 txt	-	-	-	-	-	-	-	-	-	0
baobab2 txt	-	-	-	-	-	-	-	-	-	0
chinese txt	0	0	0	0	0	0	0	0	0	0
das9201 txt	0	0	0 01	0	0	0	0	0	0 01	0
das9202 txt	0	0	0	0	0	0	0	0 01	0	0
das9203 txt	0	0	0	0 01	0 01	0	0	0	0 01	0
das9204 txt	0	0	0	0	0	0	0	0 01	0	0
das9205 txt	0	0 01	0	0	0	0	0	0	0	0
das9206 txt	0 91	0 62	0 65	0 66	0 7	0 68	0 67	0 64	0 43	0 43
das9208 txt	-	-	-	-	-	-	-	-	-	0
das9209 txt	0	0	0	0	0	0 01	0	0 01	0	0
edf9201 txt	5 27	2 71	2 63	2 62	2 67	2 91	2 8	2 62	7 34	2 62
edf9202 txt	-	-	-	-	-	-	-	-	-	0
edf9205 txt	14 69	6 86	6 75	6 86	6 77	6 63	6 9	7 28	6 19	6 19
edf9206 txt	0	0	0 01	0	0	0 01	0	0 01	0	0
edfpa15o txt	-	-	-	-	-	-	-	-	-	0
edfpa15q txt	-	-	-	-	-	-	-	-	-	0
edfpa15r txt	-	-	-	-	-	-	-	-	-	0
elf9601 txt	-	-	-	-	-	-	-	-	-	0
ftr10 txt	0 07	0 08	0 07	0 06	0 06	0 05	0 07	0 09	0 06	0 05
isp9601 txt	0 09	0 08	0 07	0 06	0 07	0 07	0 08	0 06	0 04	0 04
isp9602 txt	0 11	0 09	0 08	0 09	0 09	0 1	0 1	0 1	0 01	0 01
isp9603 txt	11 01	5 67	5 47	5 45	5 52	5 72	5 83	5 63	2 82	2 82
isp9604 txt	0 03	0 02	0 02	0 03	0 02	0 02	0 03	0 01	0 02	0 01
isp9605 txt	-	-	-	-	-	-	-	-	-	0
isp9606 txt	0	0	0	0 01	0	0	0	0	0 01	0
isp9607 txt	0 03	0 02	0 02	0 02	0 02	0 02	0 02	0 03	0 01	0 01

Table A.18: Top-down, time, 'large' trees

FT name	0 scheme	1 scheme	2 scheme	3 scheme	4 scheme	5 scheme	6 scheme	7 scheme	8 scheme	Minimum
astolfo gsq	23	19	19	19	19	19	19	19	19	19
benjam gsq	102	71	71	71	75	71	71	71	71	71
bpfig03 dat	1	1	1	1	1	1	1	1	1	1
bpfig05 dat	1	1	1	1	1	1	1	1	1	1
bpfin05 dat	1	1	1	1	1	1	1	1	1	1
bpfin05 dat	1	1	1	1	1	1	1	1	1	1
bpfs02 gsq	26	20	20	20	20	20	20	20	20	20
dre1019 dat	1	1	1	1	1	1	1	1	1	1
dre1032 dat	1	1	1	1	1	1	1	1	1	1
dre1057 dat	1	1	1	1	1	1	1	1	1	1
dre1058 dat	33	27	27	27	27	27	27	27	27	27
dre1059 dat	256	220	220	220	220	220	220	220	220	220
dresden dat	41	35	35	35	35	35	35	35	35	35
hpsif02 dat	115	115	115	115	115	115	115	115	115	115
hpsif03 dat	15	14	14	14	14	14	14	14	14	14
hpsif21 gsq	29	27	27	27	27	27	27	27	26	26
hpsif36 gsq	15	13	13	13	13	13	13	13	13	13
jdtree3 gsq	1	1	1	1	1	1	1	1	1	1
jdtree4 gsq	1	1	1	1	1	1	1	1	1	1
jdtree5 gsq	1	1	1	1	1	1	1	1	1	1
khictre dat	19	19	19	19	19	19	19	19	19	19
nakashi gsq	162	162	162	162	168	162	168	162	162	162
trials1 gsq	164	172	173	173	173	172	172	173	162	162
trials4 gsq	169	149	149	149	149	148	153	147	149	147
random3 gsq	63	59	59	59	59	59	59	59	59	59
random6 gsq	3158	1607	1627	1627	1582	1625	1625	1625	1625	1582
random8 gsq	1	1	1	1	1	1	1	1	1	1
rando12 gsq	468	446	433	433	446	433	485	446	433	433
rando13 gsq	100	95	95	95	95	94	94	94	94	94
rando16 gsq	76	59	59	59	59	56	56	56	53	53
rando18 gsq	103	69	69	69	69	53	53	53	53	53
rando23 gsq	1	1	1	1	1	1	1	1	1	1
rando25 gsq	1	1	1	1	1	1	1	1	1	1
rando27 gsq	22	17	17	17	17	13	13	17	13	13
rando28 gsq	1	1	1	1	1	1	1	1	1	1
rando29 gsq	87	87	87	88	87	87	87	88	87	87
rando30 gsq	91	84	84	84	84	84	84	84	84	84
rando31 gsq	1	1	1	1	1	1	1	1	1	1
rando33 gsq	8	8	8	8	8	8	8	8	8	8
rando34 gsq	47	22	22	22	22	22	22	22	22	22
rando35 gsq	22	18	18	18	18	18	18	18	18	18
rando36 gsq	1	1	1	1	1	1	1	1	1	1
rando37 gsq	49	39	39	39	39	38	38	39	38	38
rando38 gsq	1	1	1	1	1	1	1	1	1	1
rando39 gsq	193	162	172	172	162	167	162	185	162	162
rando40 gsq	21	21	21	21	21	21	21	21	21	21
rando42 gsq	1	1	1	1	1	1	1	1	1	1
rando43 gsq	1	1	1	1	1	1	1	1	1	1
rando44 gsq	345	338	338	338	338	338	338	338	338	338
rando45 gsq	41	30	30	30	30	30	30	30	30	30
rando46 gsq	1	1	1	1	1	1	1	1	1	1
rando47 gsq	40	38	38	38	38	38	38	38	38	38
rando48 gsq	14	13	14	14	13	13	13	13	13	13
rando52 gsq	44	42	42	42	42	42	42	44	42	42

Table A 19 Bottom-up, number of nodes, small trees, 1

FT name	0 scheme	1 scheme	2 scheme	3 scheme	4 scheme	5 scheme	6 scheme	7 scheme	8 scheme	Minimum
rando53 gsq	1	1	1	1	1	1	1	1	1	1
rando54 gsq	29	25	25	25	25	25	25	25	25	25
rando55 gsq	15	14	14	14	14	14	14	14	14	14
rando58 gsq	1	1	1	1	1	1	1	1	1	1
rando59 gsq	1233	607	722	722	672	588	672	658	579	579
rando60 gsq	17	17	17	17	17	17	17	17	17	17
rando61 gsq	25	19	19	19	19	19	19	19	19	19
rando62 gsq	1	1	1	1	1	1	1	1	1	1
rando63 gsq	15	14	14	14	14	14	14	14	14	14
rando64 gsq	102	71	71	71	71	71	71	71	71	71
rando65 gsq	23	23	23	23	23	23	23	23	23	23
rando66 gsq	22	22	22	22	22	22	22	22	22	22
rando70 gsq	40	33	33	33	33	33	33	33	33	33
rando73 gsq	1	1	1	1	1	1	1	1	1	1
rando75 gsq	1	1	1	1	1	1	1	1	1	1
rando76 gsq	14	14	14	14	14	14	14	14	14	14
rando77 gsq	41	41	41	41	41	41	41	41	41	41
rando78 gsq	1	1	1	1	1	1	1	1	1	1
rando80 gsq	9	9	9	9	9	9	9	9	9	9
rando83 gsq	23	23	23	23	23	23	23	23	23	23
rando84 gsq	141	82	82	82	82	82	82	82	82	82
rando85 gsq	1	1	1	1	1	1	1	1	1	1
rando87 gsq	1	1	1	1	1	1	1	1	1	1
rando88 gsq	23	23	23	23	23	20	23	20	20	20
rando89 gsq	1	1	1	1	1	1	1	1	1	1
rando91 gsq	1211	932	943	943	943	916	906	943	906	906
rando92 gsq	392	281	281	281	281	281	281	281	281	281
rando93 gsq	15	15	15	15	15	15	15	15	15	15
rando95 gsq	41	35	33	35	33	35	35	35	35	33
rando98 gsq	167	136	164	164	164	164	164	164	164	136
rando99 gsq	69	58	58	58	58	58	58	58	58	58
rand100 gsq	1	1	1	1	1	1	1	1	1	1
rand103 gsq	31	22	22	22	22	22	22	22	22	22
rand104 gsq	1	1	1	1	1	1	1	1	1	1
rand105 gsq	26	25	25	25	25	25	25	26	25	25
rand106 gsq	1	1	1	1	1	1	1	1	1	1
rand108 gsq	124	103	105	111	103	103	103	111	103	103
rand109 gsq	252	171	171	171	171	171	171	171	171	171
rand110 gsq	6	6	6	6	6	6	6	6	6	6
rand111 gsq	59	39	39	39	39	37	37	36	37	36
rand115 gsq	123	101	102	102	101	101	102	102	101	101
rand116 gsq	70	70	70	70	70	70	70	70	70	70
rand117 gsq	10	10	10	10	10	10	10	10	10	10
rand118 gsq	141	82	82	82	82	82	82	82	82	82
rand119 gsq	1	1	1	1	1	1	1	1	1	1
rand120 gsq	172	158	168	158	158	158	158	168	158	158
rand121 gsq	23	23	23	23	23	23	23	23	23	23
rand123 gsq	7	5	5	5	5	5	5	5	5	5
rand124 gsq	1	1	1	1	1	1	1	1	1	1
rand125 gsq	6	6	6	6	6	6	6	6	6	6
rand126 gsq	76	56	56	56	56	60	55	61	60	55
rand127 gsq	1	1	1	1	1	1	1	1	1	1
rand128 gsq	73	64	67	64	64	64	64	64	67	64
rand129 gsq	1	1	1	1	1	1	1	1	1	1

Table A 20 Bottom-up, number of nodes, small trees, 2

FT name	0 scheme	1 scheme	2 scheme	3 scheme	4 scheme	5 scheme	6 scheme	7 scheme	8 scheme	Minimum
rand130 gsq	1	1	1	1	1	1	1	1	1	1
rand132 gsq	904	704	768	768	719	746	671	704	713	671
rand134 gsq	53	50	50	50	50	50	50	50	50	50
rand135 gsq	60	54	54	54	54	54	54	54	54	54
rand137 gsq	60	35	35	35	35	36	36	35	36	35
rand138 gsq	1	1	1	1	1	1	1	1	1	1
rand139 gsq	144	117	117	117	117	117	117	117	117	117
rand141 gsq	6	6	6	6	6	6	6	6	6	6
rand142 gsq	5885	2875	2801	2801	2488	2523	2498	3548	2814	2488
rand143 gsq	1	1	1	1	1	1	1	1	1	1
rand144 gsq	64	63	63	63	63	62	62	63	62	62
rand145 gsq	11	9	9	9	9	9	9	9	9	9
rand146 gsq	60	35	35	35	35	36	36	35	36	35
rand147 gsq	584	407	460	442	442	357	427	370	355	355
rand148 gsq	1	1	1	1	1	1	1	1	1	1
rand149 gsq	1	1	1	1	1	1	1	1	1	1
rand150 gsq	4796	2557	2623	2557	2551	2521	2521	2718	2635	2521
rand151 gsq	1	1	1	1	1	1	1	1	1	1
rand153 gsq	1	1	1	1	1	1	1	1	1	1
rand154 gsq	1	1	1	1	1	1	1	1	1	1
rand155 gsq	367	226	242	226	226	226	226	226	226	226
rand156 gsq	1	1	1	1	1	1	1	1	1	1
rand158 gsq	1	1	1	1	1	1	1	1	1	1
lisaba1 gsq	49	45	45	45	45	45	45	45	45	45
lisaba3 gsq	1738	706	739	739	706	706	706	706	739	706
lisaba4 gsq	971	606	618	613	625	625	637	596	625	596
lisaba5 gsq	124	106	103	103	106	103	103	106	103	103
lisaba6 gsq	116	85	85	85	85	85	85	85	85	85
lisaba7 gsq	49	45	45	45	45	45	45	45	45	45
lisaba9 gsq	16	14	14	14	14	14	14	14	14	14
lisab10 gsq	1961	1375	2019	1858	1363	1395	1395	1375	1516	1363
lisab11 gsq	1	1	1	1	1	1	1	1	1	1
lisab13 gsq	6	6	6	6	6	6	6	6	6	6
lisab17 gsq	49	45	45	45	45	45	45	45	45	45
lisab19 gsq	1	1	1	1	1	1	1	1	1	1
lisab20 gsq	1	1	1	1	1	1	1	1	1	1
lisab24 gsq	1	1	1	1	1	1	1	1	1	1
lisab25 gsq	74	51	52	51	51	51	51	52	51	51
lisab27 gsq	277	225	229	229	229	225	225	225	225	225
lisab28 gsq	1	1	1	1	1	1	1	1	1	1
lisab30 gsq	29	25	25	25	25	25	25	25	25	25
lisab31 gsq	731	507	507	507	507	507	507	507	507	507
lisab34 gsq	25	23	25	23	25	23	23	23	23	23
lisab35 gsq	407	452	388	452	452	391	458	388	400	388
lisab36 gsq	312	186	186	186	186	178	186	206	160	160
lisab37 gsq	13	11	11	11	11	11	11	11	11	11
lisab42 gsq	1	1	1	1	1	1	1	1	1	1
lisab44 gsq	23	19	19	19	19	19	19	19	19	19
lisab46 gsq	1	1	1	1	1	1	1	1	1	1
lisab47 gsq	7	7	7	7	7	7	7	7	7	7
lisab48 gsq	1	1	1	1	1	1	1	1	1	1
lisab49 gsq	1	1	1	1	1	1	1	1	1	1
lisab50 gsq	1	1	1	1	1	1	1	1	1	1
lisab51 gsq	18	16	16	16	16	16	16	16	16	16

Table A.21: Bottom-up, number of nodes, small trees, 3

FT name	0 scheme	1 scheme	2 scheme	3 scheme	4 scheme	5 scheme	6 scheme	7 scheme	8 scheme	Minimum
lisab52 gsq	1109	712	717	717	712	689	685	712	689	685
lisab53 gsq	1	1	1	1	1	1	1	1	1	1
lisab54 gsq	20	19	19	19	19	19	19	18	19	18
lisab56 gsq	1	1	1	1	1	1	1	1	1	1
lisab57 gsq	396	332	364	340	357	356	356	357	356	332
lisab59 gsq	1	1	1	1	1	1	1	1	1	1
lisab60 gsq	25	21	21	21	21	21	21	21	21	21
lisab61 gsq	25	25	25	25	25	25	25	25	25	25
lisab62 gsq	14	14	14	14	14	14	14	14	14	14
lisab63 gsq	9	8	8	8	8	8	8	8	8	8
lisab64 gsq	1	1	1	1	1	1	1	1	1	1
lisab65 gsq	1	1	1	1	1	1	1	1	1	1
lisab66 gsq	37	37	37	37	37	37	37	37	37	37
lisab67 gsq	328	278	278	278	278	278	278	278	278	278
lisab68 gsq	1	1	1	1	1	1	1	1	1	1
lisab69 gsq	23	21	21	21	21	21	21	21	21	21
lisab70 gsq	157	143	143	143	143	143	143	143	143	143
lisab71 gsq	1	1	1	1	1	1	1	1	1	1
lisab72 gsq	140	131	131	131	131	130	131	131	130	130
lisab73 gsq	1	1	1	1	1	1	1	1	1	1
lisab74 gsq	35277	23209	23209	23209	23209	23209	23209	24777	23915	23209
lisab75 gsq	1	1	1	1	1	1	1	1	1	1
lisab77 gsq	232	210	205	205	205	210	210	210	210	205
lisab78 gsq	307	241	245	241	241	212	241	245	241	212
lisab81 gsq	1	1	1	1	1	1	1	1	1	1
lisab82 gsq	1107	880	880	880	880	880	880	880	880	880
lisab83 gsq	84	80	80	80	80	60	60	64	73	60
lisab84 gsq	84	80	80	80	80	60	60	64	73	60
lisab86 gsq	2343	681	982	974	982	752	677	974	752	677
lisab88 gsq	13	11	11	11	11	11	11	11	11	11
lisab89 gsq	100	92	92	92	92	92	92	92	92	92
lisab91 gsq	297	224	224	224	224	224	224	224	224	224
lisab95 gsq	1	1	1	1	1	1	1	1	1	1
lisab97 gsq	1	1	1	1	1	1	1	1	1	1
lisab98 gsq	1	1	1	1	1	1	1	1	1	1
lisa100 gsq	2507	1462	1524	1524	1399	1462	1505	1524	1462	1399
lisa103 gsq	1	1	1	1	1	1	1	1	1	1
lisa104 gsq	1	1	1	1	1	1	1	1	1	1
lisa107 gsq	1	1	1	1	1	1	1	1	1	1
lisa109 gsq	39	38	38	38	38	38	38	38	38	38
lisa110 gsq	792	482	530	530	530	482	482	562	468	468
lisa111 gsq	1	1	1	1	1	1	1	1	1	1
lisa112 gsq	485	313	392	392	392	313	313	313	313	313
lisa113 gsq	295	189	210	210	210	189	189	187	187	187
lisa115 gsq	15	12	12	12	12	12	12	12	12	12
lisa116 gsq	6	6	6	6	6	6	6	6	6	6
lisa119 gsq	6	6	6	6	6	6	6	6	6	6
lisa120 gsq	1	1	1	1	1	1	1	1	1	1
lisa121 gsq	12	12	12	12	12	12	12	12	12	12
lisa122 gsq	41	28	28	28	28	28	28	28	28	28
lisa123 gsq	148	145	145	145	145	133	133	133	133	133
lisa124 gsq	3587	1505	1943	1943	1532	1439	1385	1783	1385	1385
rand159 gsq	42	41	41	41	41	41	41	41	41	41
rand161 gsq	228	220	243	223	220	220	220	220	220	220

Table A 22 Bottom-up, number of nodes, small trees, 4

FT name	0 scheme	1 scheme	2 scheme	3 scheme	4 scheme	5 scheme	6 scheme	7 scheme	8 scheme	Minimum
rand163 gsq	982	881	901	901	901	875	894	880	862	862
rand164 gsq	841	693	767	767	723	723	723	767	723	693
rand165 gsq	8181	4130	4130	4130	4130	3578	3578	3578	3578	3578
rand166 gsq	8911	4839	6735	6751	4741	4945	4741	6735	5725	4741
rand167 gsq	264	221	221	221	221	219	219	221	219	219

Table A 23 Bottom-up, number of nodes, small trees, 5

FT name	0 scheme	1 scheme	2 scheme	3 scheme	4 scheme	5 scheme	6 scheme	7 scheme	8 scheme	Minimum
rando11 gsq	294803	178598	203126	203126	210517	167418	184239	173963	179261	167418
rando19 gsq	3916	3311	3261	3261	3216	3320	3214	3437	3430	3214
rando20 gsq	1283	941	1083	1083	1015	872	940	1083	872	872
rando22 gsq	2212	1931	1952	1952	2040	1774	2040	2059	1798	1774
rand160 gsq	1647432	427504	435379	433293	412969	362422	350797	427016	373863	350797
lisaba2 gsq	400351	250962	328032	328032	266526	267483	267483	287221	267483	250962
lisaba8 gsq	288549	117248	137956	137956	126202	119548	117120	130120	124730	117120
lisab14 gsq	16333	10078	13337	13337	9163	9609	9163	9163	8224	8224
lisab15 gsq	8761	6054	6559	6559	6559	5858	5858	6559	5858	5858
lisab22 gsq	8214	4416	5509	5509	5851	4173	5287	3855	4495	3855
lisab29 gsq	1	1	1	1	1	1	1	1	1	1
lisab76 gsq	5399	5022	5248	5248	5035	5033	5035	5522	5282	5022
lisab87 gsq	921517	405965	407181	407181	422473	354915	372275	417433	394065	354915
lisab92 gsq	1	1	1	1	1	1	1	1	1	1
lisa102 gsq	146804	102181	102181	102181	100604	98942	98602	100649	97694	97694
lisa117 gsq	6446409	3888036	3927624	4107065	3885836	3700777	3886170	3718517	3906491	3700777
lisa118 gsq	11397	8990	10374	10374	10374	8990	9400	9772	8990	8990
baobab1 txt	-	-	-	-	-	-	-	-	-	-
baobab2 txt	-	-	-	-	-	-	-	-	-	-
chinese txt	45	45	45	45	45	45	45	45	45	45
das9201 txt	261	204	210	210	204	204	204	210	204	204
das9202 txt	1	1	1	1	1	1	1	1	1	1
das9203 txt	12	11	11	11	11	11	11	11	11	11
das9204 txt	12	12	12	12	12	12	12	12	12	12
das9205 txt	1	1	1	1	1	1	1	1	1	1
das9206 txt	106986	53309	52569	52569	54701	54701	54701	53596	52766	52569
das9208 txt	-	-	-	-	-	-	-	-	-	-
das9209 txt	1	1	1	1	1	1	1	1	1	1
edf9201 txt	6456	4306	4509	4526	4418	4283	4282	4526	7999	4282
edf9202 txt	-	-	-	-	-	-	-	-	-	-
edf9205 txt	14417	10119	10119	10119	10119	10119	10105	10073	10119	10073
edf9206 txt	68	68	68	68	68	68	68	68	68	68
edfpa15o txt	-	-	-	-	-	-	-	-	-	-
edfpa15q txt	-	-	-	-	-	-	-	-	-	-
edfpa15r txt	-	-	-	-	-	-	-	-	-	-
elf9601 txt	-	-	-	-	-	-	-	-	-	-
ftr10 txt	612	584	584	584	584	584	584	584	584	584
isp9601 txt	2134	1237	1237	1237	1237	1237	1237	1237	1237	1237
isp9602 txt	1955	1803	1837	1837	1826	1858	1848	1794	1875	1794
isp9603 txt	290275	168160	163490	163490	171798	177212	162083	159272	179040	159272
isp9604 txt	283	251	276	276	260	248	273	254	248	248
isp9605 txt	-	-	-	-	-	-	-	-	-	-
isp9606 txt	188	163	163	163	163	163	163	163	163	163
isp9607 txt	1304	488	477	477	477	477	477	477	477	477

Table A.24: Bottom-up, number of nodes, 'large' trees

FT name	0 scheme	1 scheme	2 scheme	3 scheme	4 scheme	5 scheme	6 scheme	7 scheme	8 scheme	Minimum
astolfo gsq	0 016	0 015	0 016	0	0 016	0	0 016	0 016	0 015	0
benjam gsq	0 016	0 015	0 016	0 016	0 015	0 015	0 016	0 016	0 015	0 015
bpfig03 dat	0	0	0 016	0	0	0	0 015	0	0 016	0
bpfen05 dat	0	0	0	0	0	0	0 016	0	0	0
bpfig05 dat	0	0 016	0	0	0 015	0	0	0 016	0	0
bpfin05 dat	0 016	0	0 015	0 016	0	0 016	0	0	0	0
bpfs02 gsq	0 016	0 015	0 015	0 015	0 015	0 016	0 016	0	0	0
dre1019 dat	0	0 016	0	0	0 016	0 015	0 016	0 016	0	0
dre1032 dat	0	0	0 015	0	0	0	0 015	0	0 016	0
dre1057 dat	0 016	0 016	0	0	0	0	0	0	0	0
dre1058 dat	0 016	0 016	0 016	0 016	0 016	0 031	0 016	0 015	0 016	0 015
dre1059 dat	0 016	0 016	0 016	0 016	0 016	0 015	0 016	0	0 016	0
dresden dat	0 015	0 016	0 031	0 016	0 016	0 031	0 016	0 015	0 016	0 015
hpsf02 dat	0 015	0 015	0	0 016	0	0	0	0 016	0	0
hpsf03 dat	0 015	0	0	0	0 015	0	0	0	0 015	0
hpsf21 gsq	0	0	0	0	0 016	0	0	0 016	0 016	0
hpsf36 gsq	0	0	0 015	0	0	0	0	0 015	0 016	0
jdtree3 gsq	0	0 016	0	0 015	0	0 015	0	0	0	0
jdtree4 gsq	0	0	0	0	0 016	0 016	0 015	0	0 016	0
jdtree5 gsq	0	0	0	0 015	0	0	0	0	0	0
khictre dat	0 015	0 016	0 016	0 016	0 016	0 015	0 016	0 016	0 015	0 015
nakashi gsq	0 015	0 016	0 016	0 016	0 015	0 015	0 016	0 016	0 016	0 015
trials1 gsq	0 015	0 015	0 015	0 016	0 016	0 015	0 016	0 015	0 016	0 015
trials4 gsq	0 016	0 031	0 031	0 016	0 016	0 031	0 016	0 016	0 016	0 016
random3 gsq	0 016	0 015	0 015	0 015	0 016	0 015	0	0 015	0	0
random6 gsq	0 094	0 094	0 078	0 094	0 078	0 094	0 094	0 078	0 078	0 078
random8 gsq	0	0	0	0	0	0 016	0	0	0	0
rando12 gsq	0 078	0 062	0 062	0 063	0 063	0 078	0 062	0 078	0 062	0 062
rando13 gsq	0	0	0	0	0	0 016	0	0 016	0	0
rando16 gsq	0 015	0 015	0 015	0 016	0 016	0 031	0 031	0 015	0 015	0 015
rando18 gsq	0 016	0	0 015	0	0 016	0 016	0 015	0 016	0 016	0
rando23 gsq	0	0 016	0 016	0	0 015	0	0 016	0	0 016	0
rando25 gsq	0 016	0	0	0	0	0	0	0 016	0	0
rando27 gsq	0	0	0	0	0 016	0 016	0	0	0 016	0
rando28 gsq	0 015	0	0 016	0	0	0 016	0	0	0 016	0
rando29 gsq	0 016	0 016	0 016	0 016	0	0 015	0	0 016	0 015	0
rando30 gsq	0 016	0 015	0 015	0	0	0 016	0	0 015	0 016	0
rando31 gsq	0 015	0	0 016	0 015	0	0 016	0 016	0 016	0	0
rando33 gsq	0	0	0 016	0 016	0 016	0 016	0 015	0 016	0	0
rando34 gsq	0 016	0	0	0 016	0 016	0 015	0 016	0	0	0
rando35 gsq	0 016	0 015	0 015	0 015	0 015	0 031	0 031	0 015	0 016	0 015
rando36 gsq	0	0	0	0 015	0 015	0 015	0	0	0	0
rando37 gsq	0	0	0	0	0 016	0 016	0 016	0 016	0	0
rando38 gsq	0	0 015	0	0	0	0	0 015	0	0 016	0
rando39 gsq	0 016	0 016	0 016	0 016	0	0	0 015	0	0 016	0
rando40 gsq	0 016	0 016	0 016	0 016	0	0 015	0 016	0 016	0 015	0
rando42 gsq	0 016	0	0	0 016	0	0	0 016	0	0	0
rando43 gsq	0	0	0	0	0	0	0	0 015	0	0
rando44 gsq	0 062	0 016	0 016	0 016	0 016	0 016	0 015	0 016	0 015	0 015
rando45 gsq	0 016	0	0	0 016	0 016	0 016	0 016	0	0	0
rando46 gsq	0	0	0	0	0 016	0	0	0	0	0
rando47 gsq	0 016	0 016	0 016	0	0 015	0	0 015	0 015	0 016	0
rando48 gsq	0	0 016	0 016	0	0	0	0 015	0 016	0 015	0
rando52 gsq	0 016	0 015	0 015	0 031	0 016	0 032	0 031	0 016	0 015	0 015

Table A 25. Bottom-up, time, small trees, 1

FT name	0 scheme	1 scheme	2 scheme	3 scheme	4 scheme	5 scheme	6 scheme	7 scheme	8 scheme	Minimum
rando53 gsq	0 015	0	0 015	0	0 016	0 016	0 016	0	0	0
rando54 gsq	0 015	0 031	0 031	0 031	0 015	0 031	0 031	0 031	0 032	0 015
rando55 gsq	0 015	0 016	0 016	0 016	0	0 016	0 031	0 015	0 016	0
rando58 gsq	0	0	0	0 016	0	0 016	0	0	0	0
rando59 gsq	0 031	0 016	0 016	0 016	0 032	0 016	0 016	0 016	0 015	0 015
rando60 gsq	0 016	0 031	0 015	0 016	0 016	0 031	0 032	0 016	0 031	0 015
rando61 gsq	0	0	0	0 015	0 015	0 016	0 016	0	0	0
rando62 gsq	0	0	0	0	0	0	0 015	0	0	0
rando63 gsq	0 015	0	0 016	0 016	0 016	0 016	0 031	0 016	0 015	0
rando64 gsq	0 016	0 016	0 015	0 016	0 016	0 031	0 031	0 016	0 015	0 015
rando65 gsq	0	0 015	0	0 015	0 015	0 015	0 015	0 016	0 015	0
rando66 gsq	0	0 015	0	0 015	0 015	0 015	0 015	0 015	0 016	0
rando70 gsq	0	0 016	0	0	0 016	0	0 016	0 031	0 016	0
rando73 gsq	0	0	0	0	0	0	0 016	0	0	0
rando75 gsq	0	0	0	0	0 015	0 015	0	0	0	0
rando76 gsq	0 015	0 016	0 016	0	0 015	0 015	0 015	0 015	0 016	0
rando77 gsq	0 015	0 016	0 016	0 015	0 016	0 015	0 016	0 015	0	0
rando78 gsq	0	0 015	0	0	0 016	0 031	0 016	0 016	0	0
rando80 gsq	0 016	0 016	0	0 015	0	0 016	0 015	0 016	0 015	0
rando83 gsq	0 015	0	0	0 015	0	0	0 015	0 015	0	0
rando84 gsq	0 015	0 016	0 015	0 016	0	0 016	0 016	0 015	0 016	0
rando85 gsq	0	0	0	0	0	0 016	0	0	0 015	0
rando87 gsq	0 016	0	0	0	0	0	0 015	0	0	0
rando88 gsq	0 015	0 015	0	0 016	0	0	0	0	0	0
rando89 gsq	0	0	0 016	0	0 015	0 015	0	0	0 015	0
rando91 gsq	0 047	0 031	0 047	0 031	0 031	0 031	0 062	0 047	0 031	0 031
rando92 gsq	0 031	0 015	0 016	0 015	0 016	0 016	0 031	0 016	0 031	0 015
rando93 gsq	0 015	0 015	0 015	0 016	0 015	0 016	0 031	0 031	0 015	0 015
rando95 gsq	0 015	0 016	0 031	0 015	0 015	0 016	0 015	0 032	0 031	0 015
rando98 gsq	0 015	0 016	0 016	0 031	0 015	0	0 015	0 015	0 047	0
rando99 gsq	0 016	0	0	0	0 032	0	0 016	0 016	0 015	0
rand100 gsq	0	0	0 015	0	0	0	0	0 016	0 015	0
rand103 gsq	0 015	0 016	0 016	0 015	0 016	0 015	0 016	0 015	0 016	0 015
rand104 gsq	0 015	0 016	0	0	0	0	0	0	0	0
rand105 gsq	0 016	0	0	0	0 016	0	0	0	0	0
rand106 gsq	0 016	0	0	0 015	0	0	0 015	0	0	0
rand108 gsq	0 015	0 016	0 016	0 016	0 015	0 016	0 015	0 016	0 016	0 015
rand109 gsq	0 016	0 016	0 015	0 016	0 016	0 016	0 016	0	0	0
rand110 gsq	0 016	0	0	0 016	0	0	0	0	0	0
rand111 gsq	0 015	0	0 016	0 015	0	0 016	0 015	0	0 016	0
rand115 gsq	0 015	0	0 016	0 016	0	0 015	0 015	0	0	0
rand116 gsq	0 015	0	0 015	0 016	0 016	0	0 016	0 015	0 016	0
rand117 gsq	0 016	0	0 015	0 016	0	0 016	0	0	0 015	0
rand118 gsq	0 016	0 015	0 016	0 015	0 016	0 016	0 016	0 016	0 015	0 015
rand119 gsq	0	0 016	0	0 016	0 016	0	0	0	0	0
rand120 gsq	0 015	0 031	0 016	0 016	0 016	0 016	0 015	0	0	0
rand121 gsq	0	0 016	0 016	0 016	0	0 016	0 016	0 016	0	0
rand123 gsq	0 016	0 016	0	0	0	0 016	0 016	0 016	0	0
rand124 gsq	0	0	0	0 016	0 015	0	0	0	0	0
rand125 gsq	0	0	0 016	0	0	0	0	0	0	0
rand126 gsq	0 016	0 016	0 016	0	0	0 015	0 016	0 016	0 016	0
rand127 gsq	0 015	0 015	0	0	0 015	0 015	0 015	0	0 015	0
rand128 gsq	0 016	0 016	0 016	0 015	0 016	0 016	0 015	0	0 015	0
rand129 gsq	0 016	0	0	0	0	0	0	0	0	0

Table A.26 Bottom-up, time, small trees, 2

FT name	0 scheme	1 scheme	2 scheme	3 scheme	4 scheme	5 scheme	6 scheme	7 scheme	8 scheme	Minimum
rand130 gsq	0 015	0	0	0 016	0 015	0	0	0	0 015	0
rand132 gsq	0 079	0 078	0 063	0 062	0 063	0 062	0 062	0 062	0 062	0 062
rand134 gsq	0	0 015	0	0 015	0 016	0 016	0 015	0 016	0 016	0
rand135 gsq	0	0 016	0	0 016	0 016	0 016	0 016	0	0 016	0
rand137 gsq	0 016	0 016	0 016	0 016	0 015	0	0 016	0	0 015	0
rand138 gsq	0	0 015	0	0	0	0	0	0 016	0	0
rand139 gsq	0	0 015	0	0 016	0	0 016	0	0 016	0	0
rand141 gsq	0	0	0 016	0 016	0	0 016	0	0	0 015	0
rand142 gsq	0 141	0 11	0 094	0 093	0 094	0 109	0 094	0 109	0 094	0 093
rand143 gsq	0	0	0	0	0	0	0	0	0	0
rand144 gsq	0 016	0 015	0 016	0 016	0 015	0	0 015	0 016	0	0
rand145 gsq	0 016	0 015	0 016	0 015	0 016	0 016	0 016	0 016	0 015	0 015
rand146 gsq	0	0 016	0	0 015	0 016	0	0	0	0 016	0
rand147 gsq	0 031	0 032	0 015	0 031	0 032	0 031	0 016	0 016	0 032	0 015
rand148 gsq	0 015	0 016	0	0	0	0 015	0	0	0	0
rand149 gsq	0	0	0	0	0	0	0	0 016	0	0
rand150 gsq	0 078	0 062	0 062	0 062	0 063	0 063	0 078	0 062	0 078	0 062
rand151 gsq	0	0	0	0	0 016	0 016	0	0	0 015	0
rand153 gsq	0 015	0	0	0 016	0	0	0	0	0	0
rand154 gsq	0	0 015	0 015	0	0	0	0 016	0	0 015	0
rand155 gsq	0 015	0 016	0 015	0	0 016	0 016	0 016	0 016	0 016	0
rand156 gsq	0 015	0 016	0	0	0 015	0 015	0 015	0 016	0	0
rand158 gsq	0	0	0	0 015	0	0 015	0	0	0	0
lisaba1 gsq	0 016	0 031	0 031	0 015	0 031	0 031	0 031	0 015	0 031	0 015
lisaba3 gsq	0 078	0 125	0 078	0 078	0 078	0 078	0 078	0 078	0 078	0 078
lisaba4 gsq	0 032	0 031	0 016	0 016	0 015	0 016	0 016	0 016	0 016	0 015
lisaba5 gsq	0 016	0 016	0 016	0 016	0	0	0 015	0	0	0
lisaba6 gsq	0 015	0	0 015	0 015	0	0	0 015	0 015	0	0
lisaba7 gsq	0 015	0 016	0 031	0 031	0 016	0 031	0 031	0 031	0 031	0 015
lisaba9 gsq	0	0 016	0	0 015	0 016	0 015	0 016	0 016	0 016	0
lisab10 gsq	0 093	0 047	0 109	0 125	0 047	0 062	0 062	0 063	0 078	0 047
lisab11 gsq	0	0 015	0	0 015	0	0 016	0	0 016	0 016	0
lisab13 gsq	0 015	0	0 016	0	0	0 016	0	0	0	0
lisab17 gsq	0 015	0 016	0 015	0 016	0 016	0 032	0 047	0 015	0 016	0 015
lisab19 gsq	0	0	0	0	0	0 016	0 016	0 016	0 015	0
lisab20 gsq	0	0 015	0	0	0 015	0 015	0 016	0	0	0
lisab24 gsq	0	0 015	0	0 016	0	0	0	0	0 015	0
lisab25 gsq	0 015	0 016	0 016	0 016	0	0	0	0 016	0 015	0
lisab27 gsq	0 031	0 047	0 031	0 031	0 015	0 047	0 031	0 032	0 031	0 015
lisab28 gsq	0 015	0	0	0	0 015	0	0	0	0	0
lisab30 gsq	0 015	0 078	0 016	0 015	0 016	0 016	0 031	0 031	0 016	0 015
lisab31 gsq	0 016	0 141	0 032	0 031	0 016	0 031	0 015	0 016	0 032	0 015
lisab34 gsq	0 015	0 063	0 016	0 016	0 016	0 015	0 016	0 015	0 016	0 015
lisab35 gsq	0 015	0 031	0 016	0 015	0	0 015	0 016	0 016	0 016	0
lisab36 gsq	0 016	0 031	0 016	0 016	0 016	0 016	0 016	0 015	0 016	0 015
lisab37 gsq	0 016	0 032	0 015	0 016	0 015	0 015	0 032	0 016	0 015	0 015
lisab42 gsq	0	0 016	0 016	0 015	0	0 016	0	0	0	0
lisab44 gsq	0 015	0 031	0 016	0 015	0 016	0 016	0 015	0	0 016	0
lisab46 gsq	0	0	0	0 016	0	0	0	0	0 016	0
lisab47 gsq	0	0 015	0	0	0 016	0 015	0 016	0	0	0
lisab48 gsq	0	0 015	0 016	0	0	0 016	0 016	0 015	0	0
lisab49 gsq	0	0	0	0	0	0	0 015	0	0 016	0
lisab50 gsq	0	0	0 016	0	0	0 015	0	0 015	0	0
lisab51 gsq	0 015	0 015	0	0 016	0	0 015	0 016	0	0 016	0

Table A.27: Bottom-up, time, small trees, 3

FT name	0 scheme	1 scheme	2 scheme	3 scheme	4 scheme	5 scheme	6 scheme	7 scheme	8 scheme	Minimum
lisab52 gsq	0 015	0 031	0 015	0 032	0 031	0 016	0 032	0 016	0 031	0 015
lisab53 gsq	0	0 016	0 016	0	0	0	0 016	0	0	0
lisab54 gsq	0 016	0	0	0	0	0	0	0	0	0
lisab56 gsq	0	0	0	0	0 015	0 015	0	0	0 016	0
lisab57 gsq	0 015	0 015	0 016	0 016	0 031	0 016	0 016	0	0 015	0
lisab59 gsq	0	0	0 016	0 015	0	0 016	0	0	0 015	0
lisab60 gsq	0	0	0	0	0 016	0 015	0 016	0 016	0	0
lisab61 gsq	0 016	0	0	0	0 016	0 015	0 015	0	0	0
lisab62 gsq	0 015	0	0	0	0	0 016	0 016	0	0	0
lisab63 gsq	0	0	0 015	0 031	0 016	0 016	0	0 016	0 016	0
lisab64 gsq	0	0	0	0	0	0	0	0	0	0
lisab65 gsq	0 015	0	0 015	0	0	0	0	0 016	0 016	0
lisab66 gsq	0 016	0 015	0 015	0 032	0 016	0	0 016	0 016	0 016	0
lisab67 gsq	0 016	0 016	0 016	0 031	0 015	0 016	0 016	0 016	0 015	0 015
lisab68 gsq	0	0	0	0 015	0 016	0 015	0	0	0	0
lisab69 gsq	0 015	0	0	0	0	0 015	0	0 016	0 016	0
lisab70 gsq	0	0	0 016	0	0	0 016	0	0 015	0 016	0
lisab71 gsq	0 016	0	0 016	0 015	0	0	0	0	0	0
lisab72 gsq	0 016	0 015	0 015	0 016	0 016	0 015	0	0 016	0	0
lisab73 gsq	0	0	0	0	0	0	0	0 015	0 016	0
lisab74 gsq	0 297	0 234	0 203	0 218	0 219	0 219	0 218	0 219	0 219	0 203
lisab75 gsq	0 016	0	0	0 016	0	0 016	0	0	0	0
lisab77 gsq	0 016	0 016	0 016	0 047	0 016	0 015	0	0	0 016	0
lisab78 gsq	0 016	0	0 016	0 016	0 015	0 016	0 016	0 015	0 016	0
lisab81 gsq	0	0	0	0	0	0 016	0	0	0 016	0
lisab82 gsq	0 016	0 031	0 016	0 015	0 016	0 016	0 015	0	0 015	0
lisab83 gsq	0 016	0 016	0 016	0 016	0 016	0	0 015	0	0 016	0
lisab84 gsq	0	0 016	0 016	0	0 015	0 016	0 016	0	0 016	0
lisab86 gsq	0 031	0 032	0 015	0 016	0 016	0 031	0 032	0 016	0 016	0 015
lisab88 gsq	0	0	0 016	0	0 016	0	0	0	0	0
lisab89 gsq	0 015	0 016	0 016	0	0 016	0 015	0	0	0	0
lisab91 gsq	0 016	0 015	0 016	0	0 015	0 016	0 015	0	0 016	0
lisab95 gsq	0	0 016	0	0	0	0	0	0	0 016	0
lisab97 gsq	0	0	0 016	0 016	0	0 016	0	0	0	0
lisab98 gsq	0 015	0	0	0	0 015	0	0 015	0	0	0
lisa100 gsq	0 062	0 047	0 031	0 047	0 031	0 031	0 047	0 047	0 046	0 031
lisa103 gsq	0	0 015	0 015	0	0 015	0	0 015	0	0 016	0
lisa104 gsq	0	0	0	0 015	0	0	0	0	0	0
lisa107 gsq	0 016	0 016	0 015	0	0	0 016	0	0 015	0 016	0
lisa109 gsq	0 016	0 015	0 015	0	0 015	0 015	0 015	0	0 016	0
lisa110 gsq	0 032	0 031	0 016	0 016	0 016	0 031	0 031	0 016	0 016	0 016
lisa111 gsq	0	0	0	0	0	0 015	0 016	0 015	0	0
lisa112 gsq	0 016	0 016	0 015	0	0 016	0 016	0 015	0 015	0 015	0
lisa113 gsq	0 016	0 016	0 015	0 016	0 016	0 031	0 016	0 016	0 015	0 015
lisa115 gsq	0 015	0	0	0 016	0	0	0	0 016	0 016	0
lisa116 gsq	0	0 015	0 016	0	0	0	0	0	0	0
lisa119 gsq	0	0 016	0	0	0 016	0	0 015	0	0 016	0
lisa120 gsq	0 016	0	0	0 015	0	0 016	0 016	0 016	0	0
lisa121 gsq	0	0 016	0 016	0 015	0 016	0 016	0 016	0 015	0 016	0
lisa122 gsq	0	0	0 015	0	0	0	0	0	0	0
lisa123 gsq	0	0	0 016	0	0 015	0 016	0 016	0	0	0
lisa124 gsq	0 031	0 032	0 031	0 015	0 031	0 031	0 047	0 016	0 016	0 015
rand159 gsq	0 016	0	0	0	0	0 015	0 016	0	0	0
rand161 gsq	0 015	0 016	0	0	0	0 016	0 016	0	0	0

Table A 28 Bottom-up, time, small trees, 4

FT name	0 scheme	1 scheme	2 scheme	3 scheme	4 scheme	5 scheme	6 scheme	7 scheme	8 scheme	Minimum
rand163 gsq	0 016	0 032	0 016	0 016	0 016	0 031	0 032	0 015	0 015	0 015
rand164 gsq	0 016	0 015	0 016	0 015	0	0 015	0 032	0 015	0 016	0
rand165 gsq	0 109	0 125	0 11	0 125	0 125	0 125	0 109	0 109	0 109	0 109
rand166 gsq	0 187	0 109	0 141	0 156	0 094	0 11	0 109	0 156	0 125	0 094
rand167 gsq	0	0 125	0 015	0 016	0 015	0 015	0 016	0	0 016	0

Table A 29 Bottom-up, time, small trees, 5

FT name	0 scheme	1 scheme	2 scheme	3 scheme	4 scheme	5 scheme	6 scheme	7 scheme	8 scheme	Minimum
rando11 gsq	5 95	3 32	3 76	3 76	4	3 18	3 62	3 38	3 46	3 18
rando19 gsq	0 11	0 06	0 07	0 07	0 04	0 07	0 06	0 07	0 07	0 04
rando20 gsq	0 06	0 03	0 03	0 04	0 01	0 04	0 03	0 03	0 03	0 01
rando22 gsq	0 25	0 12	0 13	0 13	0 11	0 11	0 13	0 16	0 11	0 11
rand160 gsq	28 46	5 02	5 95	18 85	5 31	14 29	16 48	5 27	14 64	5 02
lisaba2 gsq	2 65	1 22	1 39	1 39	1 28	1 37	1 39	1 32	1 38	1 22
lisaba8 gsq	1 59	0 64	0 81	0 86	0 66	0 67	0 71	0 77	0 72	0 64
lisab14 gsq	0 34	0 2	0 25	0 25	0 2	0 21	0 21	0 2	0 18	0 18
lisab15 gsq	0 88	0 48	0 48	0 59	0 45	0 45	0 43	0 49	0 45	0 43
lisab22 gsq	0 57	0 26	0 29	0 3	0 38	0 27	0 33	0 22	0 3	0 22
lisab29 gsq	0	0	0	0	0	0	0	0	0	0
lisab76 gsq	0 27	0 17	0 17	0 16	4 07	0 17	0 17	0 18	0 17	0 16
lisab87 gsq	42 93	18 16	18 99	19 04	23 31	15 16	14 73	17 9	16 08	14 73
lisab92 gsq	0	0	0	0	0	0	0	0	0	0
lisa102 gsq	0 51	0 54	0 56	0 56	0 33	0 54	0 55	0 62	0 54	0 33
lisa117 gsq	164 16	77 4	81 73	80 56	130 1	74 77	85 27	75 26	73 32	73 32
lisa118 gsq	0 04	0 12	0 15	0 03	30 61	0 1	0 13	0 2	0 11	0 03
baobab1 txt	-	-	-	-	-	-	-	-	-	-
baobab2 txt	-	-	-	-	-	-	-	-	-	-
chinese txt	0	0	0	0	0	0	0	0	0	0
das9201 txt	0	0	0	0	0	0	0	0	0	0
das9202 txt	0	0	0	0	0	0	0	0	0	0
das9203 txt	0	0	0	0	0	0	0	0	0 01	0
das9204 txt	0	0	0	0	0	0	0	0	0	0
das9205 txt	0	0	0	0	0	0	0 01	0	0	0
das9206 txt	0 38	0 26	0 26	0 28	0 26	0 29	0 26	0 29	0 27	0 26
das9208 txt	-	-	-	-	-	-	-	-	-	-
das9209 txt	0	0	0	0	0 01	0	0	0	0	0
edf9201 txt	3 45	1 99	1 94	1 95	2 01	2 09	2 06	1 95	5 06	1 94
edf9202 txt	-	-	-	-	-	-	-	-	-	-
edf9205 txt	12 24	6 03	6 13	6 06	6 22	6 09	5 99	6 46	5 99	5 99
edf9206 txt	0	0 01	0	0	0	0	0	0 01	0	0
edfpa15o txt	-	-	-	-	-	-	-	-	-	-
edfpa15q txt	-	-	-	-	-	-	-	-	-	-
edfpa15r txt	-	-	-	-	-	-	-	-	-	-
elf9601 txt	-	-	-	-	-	-	-	-	-	-
fr10 txt	0	0	0 01	0	0 01	0	0	0	0	0
isp9601 txt	0 01	0 01	0	0 01	0	0 01	0 01	0	0 01	0
isp9602 txt	0 04	0 04	0 05	0 02	0 03	0 04	0 03	0 03	0 04	0 02
isp9603 txt	4 87	2 76	2 76	2 67	2 73	2 78	2 68	2 74	2 79	2 67
isp9604 txt	0 02	0	0 01	0 02	0 02	0 01	0 02	0 02	0 01	0
isp9605 txt	-	-	-	-	-	-	-	-	-	-
isp9606 txt	0	0	0 01	0	0	0	0	0	0	0
isp9607 txt	0 01	0	0	0	0	0 01	0 01	0 01	0	0

Table A.30: Bottom-up, time, 'large' trees

FT name	0 scheme	1 scheme	2 scheme	3 scheme	4 scheme	5 scheme	6 scheme	7 scheme	8 scheme	Minimum
rando11 gsq	N/A	260410	194716	194716	187646	108709	119892	150332	116436	108709
rando19 gsq	N/A	3427	3126	3126	2684	3272	2802	2705	2166	2166
rando20 gsq	N/A	872	931	824	840	1020	1153	1445	800	800
rando22 gsq	N/A	2768	2013	2459	3600	1880	2182	3079	1804	1804
rand160 gsq	N/A	609288	748585	714034	566814	457203	733413	1250643	319294	319294
lisaba2 gsq	N/A	87581	140448	140448	116132	104573	104573	172835	73706	73706
lisaba8 gsq	N/A	217737	141500	125162	159120	196168	154012	209606	51728	51728
lisab14 gsq	N/A	14534	10794	10794	12022	7395	8168	13499	6753	6753
lisab15 gsq	N/A	6252	6283	6406	6406	5972	5972	6087	5896	5896
lisab22 gsq	N/A	3871	4070	4070	3430	3133	3363	5401	2986	2986
lisab29 gsq	N/A	1	1	1	1	1	1	1	1	1
lisab76 gsq	N/A	4952	5666	5666	10148	10382	11175	10824	4227	4227
lisab87 gsq	N/A	305526	399635	412341	286072	252098	261666	314619	237445	237445
lisab92 gsq	N/A	1	1	1	1	1	1	1	1	1
lisa102 gsq	N/A	112893	121228	113574	99281	61004	63147	89476	62639	61004
lisa117 gsq	N/A	6471346	18547556	16219213	4305738	2259346	4178651	4322810	2472955	2259346
lisa118 gsq	N/A	8773	10419	10419	10419	7915	8953	13320	7915	7915
baobab1 txt	N/A	-	-	-	-	-	-	-	-	-
baobab2 txt	N/A	-	-	-	-	-	-	-	-	-
chinese txt	N/A	45	45	45	45	45	45	52	45	45
das9201 txt	N/A	167	210	210	167	167	167	210	167	167
das9202 txt	N/A	1	1	1	1	1	1	1	1	1
das9203 txt	N/A	11	11	11	11	11	11	11	11	11
das9204 txt	N/A	14	12	12	14	12	12	14	12	12
das9205 txt	N/A	1	1	1	1	1	1	1	1	1
das9206 txt	N/A	27313	44809	44809	55975	38718	41094	51653	25222	25222
das9208 txt	N/A	-	-	-	-	-	-	-	-	-
das9209 txt	N/A	1	1	1	1	1	1	1	1	1
edf9201 txt	N/A	4444	4627	4913	5157	4311	4256	4913	12660	4256
edf9202 txt	N/A	-	-	-	-	-	-	-	-	0
edf9205 txt	N/A	10617	12615	12161	12021	8351	7610	19470	9111	7610
edf9206 txt	N/A	58	62	62	34	58	30	62	30	30
edfpa15o txt	N/A	-	-	-	-	-	-	-	-	-
edfpa15g txt	N/A	-	-	-	-	-	-	-	-	-
edfpa15r txt	N/A	-	-	-	-	-	-	-	-	-
elf9601 txt	N/A	-	-	-	-	-	-	-	-	-
ft10 txt	N/A	1060	584	584	1197	408	247	293	302	247
isp9601 txt	N/A	691	691	691	536	678	668	508	829	508
isp9602 txt	N/A	1538	1860	1860	1241	1578	1590	1497	2033	1241
isp9603 txt	N/A	57416	74152	74152	76518	42482	67679	109190	31380	31380
isp9604 txt	N/A	196	276	196	213	184	214	319	196	184
isp9605 txt	N/A	-	-	-	-	-	-	-	-	-
isp9606 txt	N/A	287	163	287	223	287	287	287	163	163
isp9607 txt	N/A	376	435	347	543	531	685	748	435	347

Table A.31 Bottom-up, second trial, number of nodes, 'large' fault trees

FT name	0 scheme	1 scheme	2 scheme	3 scheme	4 scheme	5 scheme	6 scheme	7 scheme	8 scheme	Minimum
rando11 gsq	N/A	9 82	4 01	3 95	5 28	1 76	1 96	5 32	1 67	1 67
rando19 gsq	N/A	0 04	0 04	0 03	0 04	0 05	0 02	0 04	0 02	0 02
rando20 gsq	N/A	0 01	0 01	0	0	0 02	0 04	0 02	0 01	0
rando22 gsq	N/A	0 2	0 12	0 13	0 27	0 07	0 18	0 11	0 05	0 05
rand160 gsq	N/A	16 46	8 92	8 73	16 29	7 47	18 36	28 07	4 16	4 16
lisaba2 gsq	N/A	0 41	0 45	0 46	0 49	0 49	0 49	0 59	0 3	0 3
lisaba8 gsq	N/A	0 75	0 41	0 42	0 52	0 68	0 63	0 67	0 13	0 13
lisab14 gsq	N/A	0 1	0 08	0 08	0 08	0 16	0 15	0 09	0 12	0 08
lisab15 gsq	N/A	0 2	0 23	0 24	0 21	0 22	0 22	0 15	0 2	0 15
lisab22 gsq	N/A	0 02	0 12	0 14	0 01	0 01	0 03	0 13	0 06	0 01
lisab29 gsq	N/A	0	0	0	0	0	0	0	0	0
lisab76 gsq	N/A	0 04	0 04	0 05	0 09	0 15	0 15	0 1	0 04	0 04
lisab87 gsq	N/A	10 73	12 37	14 62	11 4	5 1	5 21	13 83	3 82	3 82
lisab92 gsq	N/A	0	0	0	0	0	0	0	0	0
lisa102 gsq	N/A	0 23	0 48	0 46	0 33	0 22	0 23	0 19	0 21	0 19
lisa117 gsq	N/A	128 95	225 45	98 27	72 41	32 28	64 77	83 35	237 05	32 28
lisa118 gsq	N/A	0 04	0 13	0 14	0 03	0 04	0 06	0 06	0 05	0 03
baobab1 txt	N/A	-	-	-	-	-	-	-	-	-
baobab2 txt	N/A	-	-	-	-	-	-	-	-	-
chinese txt	N/A	0	0	0	0	0	0	0	0	0
das9201 txt	N/A	0	0	0	0	0	0	0	0	0
das9202 txt	N/A	0	0	0	0	0	0	0	0	0
das9203 txt	N/A	0	0	0	0	0	0	0	0	0
das9204 txt	N/A	0	0	0	0	0	0	0	0	0
das9205 txt	N/A	0	0	0	0	0	0	0	0	0
das9206 txt	N/A	0 15	0 23	0 24	0 36	0 27	0 28	0 37	0 16	0 15
das9208 txt	N/A	-	-	-	-	-	-	-	-	-
das9209 txt	N/A	0	0	0	0	0	0	0	0	0
edf9201 txt	N/A	1 55	2 5	2 94	1 77	2 75	2 73	2 93	12 13	1 55
edf9202 txt	N/A	-	-	-	-	-	-	-	-	-
edf9205 txt	N/A	10 35	10 33	9 73	10 05	7 12	7 87	23 78	6 33	6 33
edf9206 txt	N/A	0	0	0	0	0	0	0	0	0
edfpa15o txt	N/A	-	-	-	-	-	-	-	-	-
edfpa15q txt	N/A	-	-	-	-	-	-	-	-	-
edfpa15r txt	N/A	-	-	-	-	-	-	-	-	-
elf9601 txt	N/A	-	-	-	-	-	-	-	-	-
ftr10 txt	N/A	0	0	0	0	0 01	0	0	0 01	0
isp9601 bxt	N/A	0 01	0	0 01	0 01	0	0 01	0	0 01	0
isp9602 bxt	N/A	0 04	0 04	0 03	0 02	0 03	0 05	0 03	0 03	0 02
isp9603 bxt	N/A	0 86	1 2	1 24	1 22	0 24	0 29	0 61	0 21	0 21
isp9604 bxt	N/A	0 01	0 01	0 01	0	0	0	0 01	0 01	0
isp9605 bxt	N/A	-	-	-	-	-	-	-	-	-
isp9606 bxt	N/A	0	0	0	0	0	0	0	0	0
isp9607 bxt	N/A	0	0	0 02	0	0 01	0	0	0	0

Table A 32: Bottom-up, second trial, time, 'large' fault trees

FT name	0 scheme	1 scheme	2 scheme	3 scheme	4 scheme	5 scheme	6 scheme	7 scheme	8 scheme	Minimum
rando11 gsq	196915	104214	152052	152052	153784	102286	146688	109658	139380	102286
rando19 gsq	3153	2105	2146	2148	2107	2115	2116	2106	2149	2105
rando20 gsq	1114	975	1062	1040	1036	975	992	1065	981	975
rando22 gsq	3876	2614	2443	2443	2614	2427	2427	2641	2457	2427
rand160 gsq	5779369	590785	1099803	1082687	1276538	903819	572526	540373	903819	540373
lisaba2 gsq	207800	95650	105643	105643	121636	109396	121636	103503	109396	95650
lisaba8 gsq	544421	141588	138886	138886	138886	160870	144958	162526	161422	138886
lisab14 gsq	11362	6650	7014	7014	7295	6650	7295	7295	6654	6650
lisab15 gsq	8177	3850	4591	4591	4591	3619	3619	4591	3619	3619
lisab22 gsq	4085	2921	3207	3207	2802	2921	2921	3119	2921	2802
lisab29 gsq	1	1	1	1	1	1	1	1	1	1
lisab76 gsq	13339	11751	10868	10868	12260	12208	12200	10343	10370	10343
lisab87 gsq	633529	262813	319715	319715	267699	242977	255213	267699	240793	240793
lisab92 gsq	1	1	1	1	1	1	1	1	1	1
lisa102 gsq	89466	72254	72254	72254	70607	69771	69807	70540	68801	68801
lisa117 gsq	4889999	3570701	3491158	3459450	3483425	3425044	3545389	3481266	3610554	3425044
lisa118 gsq	11729	8629	10156	10156	10156	8629	9217	9442	8629	8629
baobab1 txt	-	-	-	-	-	-	-	-	-	-
baobab2 txt	-	-	-	-	-	-	-	-	-	-
chinese txt	45	45	45	45	45	45	45	45	45	45
das9201 txt	189	136	136	136	136	136	136	136	136	136
das9202 txt	1	1	1	1	1	1	1	1	1	1
das9203 txt	12	11	11	11	11	11	11	11	11	11
das9204 txt	12	12	12	12	12	12	12	12	12	12
das9205 txt	1	1	1	1	1	1	1	1	1	1
das9206 txt	56754	44279	44279	44279	44739	44739	44739	44279	44283	44279
das9208 txt	-	-	-	-	-	-	-	-	-	-
das9209 txt	1	1	1	1	1	1	1	1	1	1
edf9201 txt	12564	7241	7203	7331	7388	7130	7202	7331	14926	7130
edf9202 txt	-	-	-	-	-	-	-	-	-	0
edf9205 txt	12157	9793	9793	9793	9793	9793	9707	9665	9793	9665
edf9206 txt	32	32	32	32	32	32	32	32	32	32
edfpa15o txt	-	-	-	-	-	-	-	-	-	-
edfpa15q txt	-	-	-	-	-	-	-	-	-	-
edfpa15r txt	-	-	-	-	-	-	-	-	-	-
elf9601 txt	-	-	-	-	-	-	-	-	-	-
ftr10 txt	1018	1018	1018	1018	1018	1018	1018	1018	1018	1018
isp9601 txt	1587	769	769	769	769	769	769	769	769	769
isp9602 txt	1364	1274	1301	1301	1287	1318	1305	1267	1324	1267
isp9603 txt	32438	29156	29156	29156	29156	29156	29156	29268	30540	29156
isp9604 txt	214	188	188	188	188	188	188	188	188	188
isp9605 txt	-	-	-	-	-	-	-	-	-	-
isp9606 txt	431	223	262	223	223	223	223	223	262	223
isp9607 txt	1488	472	450	450	450	450	450	450	450	450

Table A.33. Bottom-up, third trial, number of nodes, 'large' fault trees

FT name	0 scheme	1 scheme	2 scheme	3 scheme	4 scheme	5 scheme	6 scheme	7 scheme	8 scheme	Minimum
rando11 gsq	2 25	1 05	1 74	1 7	1 74	1 06	1 49	1 03	1 41	1 03
rando19 gsq	0 02	0 02	0 03	0 02	0 02	0 02	0 02	0 03	0 02	0 02
rando20 gsq	0 01	0 02	0 01	0 02	0 01	0 03	0 02	0 01	0 02	0 01
rando22 gsq	0 16	0 11	0 1	0 12	0 1	0 09	0 11	0 14	0 12	0 09
rand160 gsq	204 1	15 28	33 71	35 09	38 01	35 26	19 38	15 67	35 53	15 28
lisaba2 gsq	0 58	0 32	0 34	0 35	0 44	0 39	0 4	0 41	0 38	0 32
lisaba8 gsq	1 47	0 51	0 39	0 45	0 53	0 52	0 5	0 55	0 53	0 39
lisab14 gsq	0 15	0 12	0 15	0 13	0 13	0 1	0 12	0 13	0 11	0 1
lisab15 gsq	0 18	0 1	0 1	0 11	0 12	0 09	0 09	0 11	0 08	0 08
lisab22 gsq	0 11	0 06	0 06	0 06	0 06	0 05	0 06	0 06	0 05	0 05
lisab29 gsq	0	0	0	0	0	0	0	0	0	0
lisab76 gsq	0 11	0 1	0 07	0 08	0 1	0 09	0 09	0 08	0 08	0 07
lisab87 gsq	25 7	8 99	9 27	9 3	9 43	8 82	8 94	9 46	8 9	8 82
lisab92 gsq	0	0	0	0	0	0	0	0	0	0
lisa102 gsq	0 31	0 24	0 22	0 22	0 21	0 21	0 21	0 23	0 22	0 21
lisa117 gsq	46 46	46 47	36 95	36 27	46 25	43 36	44 41	47 49	41 19	36 95
lisa118 gsq	0 05	0 03	0 03	0 04	0 03	0 02	0 03	0 05	0 03	0 02
baobab1 txt	-	-	-	-	-	-	-	-	-	-
baobab2 txt	-	-	-	-	-	-	-	-	-	-
chinese txt	0	0	0	0	0	0	0	0	0	0
das9201 txt	0	0	0	0	0	0	0	0	0	0
das9202 txt	0	0	0	0	0	0	0	0	0	0
das9203 txt	0	0	0	0	0	0	0	0	0	0
das9204 txt	0	0	0	0	0	0	0	0	0	0
das9205 txt	0	0	0	0	0	0	0	0	0	0
das9206 txt	0 36	0 25	0 27	0 25	0 27	0 28	0 27	0 27	0 26	0 25
das9208 txt	-	-	-	-	-	-	-	-	-	-
das9209 txt	0	0	0	0	0	0	0	0	0	0
edf9201 txt	5 38	2 77	2 83	2 97	2 92	2 75	2 78	2 94	8 9	2 75
edf9202 txt	-	-	-	-	-	-	-	-	-	-
edf9205 txt	10 49	4 51	4 48	4 35	4 59	4 31	4 17	4 64	4 18	4 17
edf9206 txt	0	0	0	0 01	0	0	0	0	0	0
edfpa15o txt	-	-	-	-	-	-	-	-	-	-
edfpa15q txt	-	-	-	-	-	-	-	-	-	-
edfpa15r txt	-	-	-	-	-	-	-	-	-	-
elf9601 txt	-	-	-	-	-	-	-	-	-	-
ftr10 txt	0 01	0	0	0	0	0	0 01	0 01	0	0
isp9601 txt	0 01	0 02	0	0 01	0 01	0 01	0	0	0	0
isp9602 txt	0 03	0 01	0 02	0 03	0 02	0 02	0 03	0 02	0 02	0 01
isp9603 txt	0 26	0 21	0 21	0 25	0 2	0 21	0 21	0 24	0 21	0 2
isp9604 txt	0	0	0	0	0	0	0	0 01	0	0
isp9605 txt	-	-	-	-	-	-	-	-	-	-
isp9606 txt	0	0	0	0	0 01	0	0	0	0 01	0
isp9607 txt	0 01	0	0	0 01	0 01	0 01	0 01	0	0	0

Table A 34 Bottom-up, third trial, processing time, 'large' fault trees

FT name	0 scheme	1 scheme	2 scheme	3 scheme	4 scheme	5 scheme	6 scheme	7 scheme	8 scheme	Minimum
rando11 gsq	N/A	13308	15006	15006	7141	7489	5694	5137	13675	5137
rando19 gsq	N/A	860	825	833	1197	767	1092	1082	624	624
rando20 gsq	N/A	377	473	408	397	367	296	556	347	296
rando22 gsq	N/A	1556	1225	1155	2277	899	1374	2359	754	754
rand160 gsq	N/A	15450	13315	10311	29304	13194	17888	12313	9371	9371
lisaba2 gsq	N/A	21380	5606	5606	5613	14549	9455	5343	15428	5343
lisaba8 gsq	N/A	19774	2303	2377	1325	5280	2140	2666	3987	1325
lisab14 gsq	N/A	1820	1930	1914	2337	1449	2213	3025	1174	1174
lisab15 gsq	N/A	2317	2132	2152	2530	795	946	3686	553	553
lisab22 gsq	N/A	1681	1494	1494	1296	1475	1082	3182	1570	1082
lisab29 gsq	N/A	1	1	1	1	1	1	1	1	1
lisab76 gsq	N/A	1764	2679	2679	853	1479	1096	889	925	853
lisab87 gsq	N/A	16973	40518	42567	10139	6282	3740	12282	4666	3740
lisab92 gsq	N/A	1	1	1	1	1	1	1	1	1
lisa102 gsq	N/A	8985	1094	1073	1505	3025	1448	2395	2076	1073
lisa117 gsq	N/A	77479	15911	14620	23537	13651	19699	58105	22017	13651
lisa118 gsq	N/A	1123	1695	1342	1302	880	803	1860	959	803
baobab1 txt	N/A	-	25013	25013	31638	52407	22965	21501	24911	21501
baobab2 txt	N/A	175858	11415	11415	10384	170874	17747	21705	9936	9936
chinese txt	N/A	89	125	125	125	89	106	100	89	89
das9201 txt	N/A	332	243	243	236	276	224	261	274	224
das9202 txt	N/A	1	1	1	1	1	1	1	1	1
das9203 txt	N/A	29	29	29	29	29	29	29	28	28
das9204 txt	N/A	43	37	37	48	38	36	48	36	36
das9205 txt	N/A	1	1	1	1	1	1	1	1	1
das9206 txt	N/A	6760	1568	1568	1161	5073	1211	2438	3483	1161
das9208 txt	N/A	24531	38370	38370	9597	18357	9207	17882	19602	9207
das9209 txt	N/A	1	1	1	1	1	1	1	1	1
edf9201 txt	N/A	4098	2036	3196	3560	3544	1923	3185	11086	1923
edf9202 txt	N/A	69258	987	987	1052	162514	1323	256340	54228	987
edf9205 txt	N/A	1791	2757	2723	2843	1608	1634	5598	2051	1608
edf9206 txt	N/A	126	74	73	93	86	81	73	110	73
edfpa15o txt	N/A	584086	132522	132522	270374	426733	115759	74481	424173	74481
edfpa15q txt	N/A	339878	195494	195494	47017	574081	200057	47774	226332	47017
edfpa15r txt	N/A	340532	149575	148002	48447	592435	199072	45943	176908	45943
elf9601 txt	N/A	35027	61286	61286	68855	27933	31787	95026	34272	27933
ifr10 txt	N/A	2107	181	181	280	729	318	440	613	181
isp9601 txt	N/A	419	315	315	311	540	406	262	466	262
isp9602 txt	N/A	1807	972	972	1133	1581	686	1016	2024	686
isp9603 txt	N/A	5852	4435	4131	3256	4618	1773	2790	2227	1773
isp9604 txt	N/A	247	413	238	416	239	232	453	290	232
isp9605 txt	N/A	397677	12445	12445	11859	400846	21993	23986	572149	11859
isp9606 txt	N/A	171	120	89	132	161	132	108	155	89
isp9607 txt	N/A	891	493	394	462	710	462	502	638	394

Table A 35: The ite method, number of nodes, 'large' fault trees

FT name	0 scheme	1 scheme	2 scheme	3 scheme	4 scheme	5 scheme	6 scheme	7 scheme	8 scheme	Minimum
rando11 gsq	N/A	3 73	6 04	6 22	1 33	1 48	1 08	1 39	3 61	1 08
rando19 gsq	N/A	0 22	0 24	0 24	0 25	0 23	0 25	0 24	0 23	0 22
rando20 gsq	N/A	0 22	0 23	0 23	0 21	0 22	0 22	0 23	0 22	0 21
rando22 gsq	N/A	0 27	0 22	0 26	0 43	0 24	0 28	0 43	0 23	0 22
rand160 gsq	N/A	10 83	8 50	4 44	53 05	6 97	26 11	14 08	3 28	3 28
lisaba2 gsq	N/A	10 14	1 97	1 92	0 88	4 85	2 07	1 30	5 57	0 88
lisaba8 gsq	N/A	9 41	0 40	0 38	0 31	0 87	0 43	0 48	0 58	0 31
lisab14 gsq	N/A	0 20	0 37	0 33	0 33	0 27	0 34	0 42	0 24	0 20
lisab15 gsq	N/A	0 32	0 37	0 38	0 43	0 25	0 26	0 65	0 23	0 23
lisab22 gsq	N/A	0 32	0 32	0 28	0 28	0 28	0 27	0 51	0 25	0 25
lisab29 gsq	N/A	0 00	0 00	0 00	0 00	0 00	0 00	0 00	0 00	0 00
lisab76 gsq	N/A	0 29	0 38	0 37	0 25	0 26	0 26	0 24	0 22	0 22
lisab87 gsq	N/A	6 86	33 61	38 85	3 15	1 16	0 57	4 02	0 67	0 57
lisab92 gsq	N/A	0 00	0 00	0 00	0 00	0 00	0 00	0 00	0 00	0 00
lisa102 gsq	N/A	1 92	0 30	0 28	0 29	0 43	0 29	0 45	0 30	0 28
lisa117 gsq	N/A	156 81	13 46	10 85	36 60	6 34	20 20	86 97	14 00	6 34
lisa118 gsq	N/A	0 47	0 51	0 53	0 50	0 45	0 45	0 56	0 51	0 45
baobab1 txt	N/A	-	261 41	233 21	355 14	466 04	75 46	140 10	360 42	75 46
baobab2 txt	N/A	628 97	2 71	2 72	2 32	611 52	6 59	9 82	2 19	2 19
chinese txt	N/A	0 21	0 21	0 20	0 21	0 23	0 22	0 23	0 21	0 20
das9201 txt	N/A	0 21	0 21	0 22	0 22	0 20	0 20	0 22	0 21	0 20
das9202 txt	N/A	0 00	0 00	0 00	0 00	0 00	0 00	0 00	0 00	0 00
das9203 txt	N/A	0 63	0 63	0 63	0 61	0 65	0 67	0 63	0 64	0 61
das9204 txt	N/A	0 20	0 22	0 21	0 21	0 22	0 21	0 21	0 20	0 20
das9205 txt	N/A	0 00	0 00	0 00	0 00	0 00	0 00	0 00	0 01	0 00
das9206 txt	N/A	1 12	0 25	0 24	0 23	0 72	0 24	0 36	0 43	0 23
das9208 txt	N/A	11 93	31 73	31 97	1 76	6 79	1 70	6 36	7 37	1 70
das9209 txt	N/A	0 00	0 00	0 00	0 00	0 01	0 00	0 00	0 00	0 00
edf9201 txt	N/A	0 54	0 28	0 38	0 45	0 46	0 29	0 40	2 70	0 28
edf9202 txt	N/A	91 82	0 28	0 27	0 27	519 06	0 31	1306 66	59 14	0 27
edf9205 txt	N/A	0 27	0 31	0 31	0 32	0 25	0 30	0 76	0 29	0 25
edf9206 txt	N/A	0 21	0 21	0 21	0 21	0 21	0 21	0 21	0 21	0 21
edfpa15o txt	N/A	6785 02	354 47	354 65	1533 56	3726 86	285 42	118 94	2990 56	118 94
edfpa15q txt	N/A	2342 00	794 80	795 53	44 30	6479 81	810 65	47 41	999 43	44 30
edfpa15r txt	N/A	2450 89	442 69	431 53	49 41	7176 05	822 83	44 43	634 77	44 43
elf9601 txt	N/A	19 40	75 36	75 21	96 12	12 89	18 70	178 17	18 74	12 89
ftr10 txt	N/A	0 31	0 21	0 21	0 23	0 23	0 21	0 21	0 22	0 21
isp9601 txt	N/A	0 22	0 22	0 21	0 21	0 21	0 20	0 21	0 22	0 20
isp9602 txt	N/A	0 47	0 44	0 43	0 44	0 46	0 43	0 45	0 49	0 43
isp9603 txt	N/A	0 90	0 60	0 55	0 38	0 63	0 29	0 38	0 62	0 29
isp9604 txt	N/A	0 20	0 21	0 22	0 21	0 21	0 22	0 21	0 58	0 20
isp9605 txt	N/A	3064 85	3 10	3 11	2 58	3148 44	9 18	11 07	6341 83	2 58
isp9606 txt	N/A	0 22	0 21	0 21	0 21	0 23	0 21	0 22	0 58	0 21
isp9607 txt	N/A	0 22	0 21	0 21	0 21	0 22	0 21	0 21	0 44	0 21

Table A 36: The ite method, processing time, 'large' fault trees

FT name	0 scheme	1 scheme	2 scheme	3 scheme	4 scheme	5 scheme	6 scheme	7 scheme	8 scheme	Minimum
rando11 gsq	173804	102869	149072	149072	150868	101025	144076	107857	137572	101025
rando19 gsq	3153	2105	2146	2148	2107	2115	2116	2106	2149	2105
rando20 gsq	1114	951	1062	1040	1036	951	992	1065	959	951
rando22 gsq	3825	2614	2443	2443	2614	2427	2427	2641	2457	2427
rand160 gsq	3702156	440250	859855	842739	959612	695087	440789	417665	695087	417665
lisaba2 gsq	158222	81862	83505	83505	103734	99571	103734	93664	99571	81862
lisaba8 gsq	450927	100069	97999	97999	97999	109731	102553	110973	110145	97999
lisab14 gsq	11306	6650	6906	6906	7295	6650	7295	7295	6654	6650
lisab15 gsq	5657	3316	4057	4057	4057	3166	3166	4057	3166	3166
lisab22 gsq	4085	2921	3207	3207	2802	2921	2921	3119	2921	2802
lisab29 gsq	1	1	1	1	1	1	1	1	1	1
lisab76 gsq	13324	11751	10856	10856	12260	12200	12200	10337	10370	10337
lisab87 gsq	567101	260089	316991	316991	264975	240253	252489	264975	238069	238069
lisab92 gsq	1	1	1	1	1	1	1	1	1	1
lisa102 gsq	43157	42191	42191	42191	41146	40649	40542	40713	39536	39536
lisa117 gsq	3737589	2665767	2880204	2829224	2544463	2640323	2640323	2838366	2624344	2624344
lisa118 gsq	9697	7986	9370	9370	9370	7986	8530	8700	7986	7986
baobab1 txt	-	-	-	-	-	-	-	-	-	-
baobab2 txt	-	-	-	-	-	-	-	-	-	-
chinese txt	45	45	45	45	45	45	45	45	45	45
das9201 txt	189	136	136	136	136	136	136	136	136	136
das9202 txt	1	1	1	1	1	1	1	1	1	1
das9203 txt	12	11	11	11	11	11	11	11	11	11
das9204 txt	12	12	12	12	12	12	12	12	12	12
das9205 txt	1	1	1	1	1	1	1	1	1	1
das9206 txt	15876	13968	13968	13968	15876	15876	15876	13968	13972	13968
das9208 txt	-	-	-	-	-	-	-	-	-	-
das9209 txt	1	1	1	1	1	1	1	1	1	1
edf9201 txt	12542	7241	7203	7331	7388	7130	7202	7331	14850	7130
edf9202 txt	-	-	-	-	-	-	-	-	-	-
edf9205 txt	11953	9793	9793	9793	9793	9793	9707	9665	9793	9665
edf9206 txt	32	32	32	32	32	32	32	32	32	32
edfpa15o txt	-	-	-	-	-	-	-	-	-	-
edfpa15q txt	-	-	-	-	-	-	-	-	-	-
edfpa15r txt	-	-	-	-	-	-	-	-	-	-
elf9601 txt	-	-	-	-	-	-	-	-	-	-
ftr10 txt	596	596	596	596	596	596	596	596	596	596
isp9601 txt	1157	752	752	752	752	752	752	752	752	752
isp9602 txt	1364	1274	1301	1301	1287	1318	1305	1267	1324	1267
isp9603 txt	27770	21804	21804	21804	21804	21804	24122	24290	22832	21804
isp9604 txt	214	188	188	188	188	188	188	188	188	188
isp9605 txt	-	-	-	-	-	-	-	-	-	-
isp9606 txt	423	223	262	223	223	223	223	223	262	223
isp9607 txt	1488	472	450	450	450	450	450	450	450	450

Table A.37: The Component Connection method with sub-node sharing, number of nodes, 'large' fault trees

FT name	0 scheme	1 scheme	2 scheme	3 scheme	4 scheme	5 scheme	6 scheme	7 scheme	8 scheme	Minimum
rando11 gsq	9 46	5 85	8 59	8 72	8 65	5 87	8 71	5 88	8 62	5 85
rando19 gsq	0 1	0 09	0 08	0 08	0 09	0 1	0 09	0 09	0 12	0 08
rando20 gsq	0 02	0 08	0 04	0 05	0 03	0 07	0 12	0 02	0 12	0 02
rando22 gsq	0 66	0 59	0 65	0 66	0 59	0 59	0 63	0 76	0 73	0 59
rand160 gsq	101 15	7 11	16 95	13 86	18 65	17 78	10 15	7 38	17 88	7 11
lisaba2 gsq	10 01	1 21	1 53	1 55	1 76	1 22	1 77	1 06	1 27	1 06
lisaba8 gsq	1 42	7 16	4 99	5	6 99	6 62	7 1	6 71	6 75	1 42
lisab14 gsq	4 88	3 22	3 89	3 87	3 87	3 24	3 88	3 87	3 24	3 22
lisab15 gsq	16 24	7 57	7 56	7 65	7 58	7 55	7 57	7 58	7 55	7 55
lisab22 gsq	10 4	2 62	2 47	2 48	2 55	2 42	2 43	3 04	2 46	2 42
lisab29 gsq	0	0	0	0	0	0	0	0	0	0
lisab76 gsq	0 34	0 31	0 27	0 26	0 32	0 32	0 32	0 3	0 27	0 26
lisab87 gsq	1977 47	735 9	696 99	698 97	735 21	772 67	781 25	736 99	772 63	696 99
lisab92 gsq	0	0	0	0	0	0	0	0	0	0
lisa102 gsq	0 95	0 83	0 82	0 85	0 83	0 9	0 79	0 91	0 85	0 79
lisa117 gsq	44 24	43 76	43 94	52 79	40 21	40 15	36 19	53 03	44 93	36 19
lisa118 gsq	0 37	0 28	0 34	0 32	0 32	0 28	0 32	0 29	0 3	0 28
baobab1 txt	-	-	-	-	-	-	-	-	-	-
baobab2 txt	-	-	-	-	-	-	-	-	-	-
chinese txt	0	0 01	0	0	0	0	0	0	0	0
das9201 txt	0	0	0	0	0 1	0	0	0	0	0
das9202 txt	0	0	0	0	0	0	0	0	0	0
das9203 txt	0	0	0	0	0	0	0	0	0	0
das9204 txt	0	0	0	0	0	0	0	0	0	0
das9205 txt	0	0	0	0	0	0	0	0	0	0
das9206 txt	7 84	3 31	3 27	3 33	6 15	6 04	6 05	3 33	3 32	3 27
das9208 txt	-	-	-	-	-	-	-	-	-	-
das9209 txt	0	0	0	0	0	0	0	0	0	0
edf9201 txt	96 21	40 2	37 41	40 01	40 14	42 6	44 09	40 59	187 68	37 41
edf9202 txt	-	-	-	-	-	-	-	-	-	-
edf9205 txt	921 87	244 21	250 12	250 61	251 03	253 41	251 27	245 42	252 29	244 21
edf9206 txt	0	0	0	0	0	0 01	0	0	0	0
edfpa15o txt	-	-	-	-	-	-	-	-	-	-
edfpa15q txt	-	-	-	-	-	-	-	-	-	-
edfpa15r txt	-	-	-	-	-	-	-	-	-	-
elf9601 txt	-	-	-	-	-	-	-	-	-	-
ftr10 txt	0 01	0	0 01	0 01	0 01	0 01	0	0	0 01	0
isp9601 txt	0 21	0 1	0 09	0 09	0 09	0 09	0 08	0 1	0 12	0 08
isp9602 txt	0 06	0 05	0 06	0 05	0 05	0 05	0 05	0 04	0 07	0 04
isp9603 txt	1 47	1 14	1 21	1 14	1 14	1 26	1 28	1 28	1 19	1 14
isp9604 txt	0 01	0 01	0 01	0	0 01	0 01	0 01	0 02	0	0
isp9605 txt	-	-	-	-	-	-	-	-	-	-
isp9606 txt	0	0	0	0 01	0 01	0	0	0	0	0
isp9607 txt	0 01	0 03	0 01	0 01	0	0 02	0	0 01	0 01	0

Table A.38. The Component Connection method with sub-node sharing, time, 'large' fault trees

FT name	0 scheme	1 scheme	2 scheme	3 scheme	4 scheme	5 scheme	6 scheme	7 scheme	8 scheme	Minimum
rando11 gsq	N/A	3 70	5 90	5 86	1 33	1 45	0 99	1 38	3 58	0 99
rando19 gsq	N/A	0 23	0 24	0 26	0 25	0 22	0 24	0 25	0 23	0 22
rando20 gsq	N/A	0 22	0 22	0 24	0 22	0 22	0 21	0 24	0 25	0 21
rando22 gsq	N/A	0 27	0 27	0 24	0 42	0 23	0 26	0 43	0 17	0 17
rand160 gsq	N/A	10 96	8 43	4 83	52 72	6 92	25 44	14 09	3 01	3 01
lisaba2 gsq	N/A	10 31	1 99	1 89	0 86	5 06	2 16	1 25	5 42	0 86
lisaba8 gsq	N/A	9 62	0 40	0 45	0 37	0 94	0 41	0 44	0 64	0 37
lisab14 gsq	N/A	0 29	0 37	0 36	0 32	0 30	0 32	0 43	0 26	0 26
lisab15 gsq	N/A	0 33	0 37	0 37	0 44	0 26	0 26	0 68	0 24	0 24
lisab22 gsq	N/A	0 30	0 28	0 28	0 31	0 28	0 25	0 51	0 29	0 25
lisab29 gsq	N/A	0 00	0 00	0 00	0 00	0 00	0 00	0 00	0 00	0 00
lisab76 gsq	N/A	0 30	0 36	0 36	0 26	0 29	0 25	0 25	0 23	0 23
lisab87 gsq	N/A	7 17	33 37	37 30	2 83	1 16	0 57	4 18	0 79	0 57
lisab92 gsq	N/A	0 00	0 00	0 00	0 00	0 00	0 00	0 00	0 00	0 00
lisa102 gsq	N/A	1 98	0 28	0 27	0 31	0 45	0 29	0 36	0 34	0 27
lisa117 gsq	N/A	157 07	13 64	10 88	32 82	6 93	19 90	86 43	13 75	6 93
lisa118 gsq	N/A	0 47	0 56	0 49	0 47	0 48	0 45	0 56	0 45	0 45
baobab1 txt	N/A	-	233 38	257 84	354 26	431 18	73 82	139 88	365 53	73 82
baobab2 txt	N/A	620 20	2 67	2 68	2 29	610 79	6 26	9 32	2 10	2 10
chinese txt	N/A	0 22	0 22	0 21	0 22	0 21	0 22	0 21	0 20	0 20
das9201 txt	N/A	0 21	0 22	0 22	0 21	0 21	0 21	0 22	0 20	0 20
das9202 txt	N/A	0 00	0 00	0 00	0 00	0 00	0 01	0 00	0 00	0 00
das9203 txt	N/A	0 64	0 64	0 63	0 63	0 64	0 63	0 64	0 64	0 63
das9204 txt	N/A	0 21	0 22	0 21	0 22	0 21	0 20	0 22	0 21	0 20
das9205 txt	N/A	0 00	0 00	0 00	0 00	0 01	0 00	0 00	0 01	0 00
das9206 txt	N/A	1 11	0 25	0 25	0 23	0 70	0 23	0 34	0 45	0 23
das9208 txt	N/A	11 85	31 41	31 45	1 88	6 74	1 76	6 70	7 99	1 76
das9209 txt	N/A	0 00	0 00	0 00	0 00	0 00	0 00	0 01	0 00	0 00
edf9201 txt	N/A	0 50	0 27	0 38	0 41	0 43	0 27	0 37	2 56	0 27
edf9202 txt	N/A	91 53	0 26	0 27	0 26	517 49	0 28	1311 76	58 65	0 26
edf9205 txt	N/A	0 26	0 33	0 33	0 34	0 25	0 25	0 81	0 29	0 25
edf9206 txt	N/A	0 22	0 22	0 21	0 20	0 21	0 22	0 21	0 22	0 20
edfpa15o txt	N/A	6948 50	347 04	347 86	1515 73	3701 29	260 33	116 63	3687 14	116 63
edfpa15q txt	N/A	2377 85	800 91	800 85	44 99	6536 92	800 82	48 15	1014 10	44 99
edfpa15r txt	N/A	2442 74	441 75	431 28	46 46	7184 17	813 10	44 92	629 82	44 92
elf9601 txt	N/A	19 11	75 04	74 88	95 72	12 85	18 45	175 83	18 37	12 85
ftr10 txt	N/A	0 30	0 21	0 22	0 21	0 23	0 21	0 22	0 22	0 21
isp9601 txt	N/A	0 22	0 22	0 20	0 21	0 23	0 21	0 21	0 22	0 20
isp9602 txt	N/A	0 49	0 44	0 45	0 43	0 47	0 43	0 43	0 51	0 43
isp9603 txt	N/A	0 89	0 58	0 55	0 39	0 64	0 27	0 37	0 61	0 27
isp9604 txt	N/A	0 21	0 22	0 21	0 21	0 23	0 21	0 21	0 57	0 21
isp9605 txt	N/A	3062 90	3 10	3 10	2 83	3159 50	9 19	11 07	6342 98	2 83
isp9606 txt	N/A	0 23	0 21	0 21	0 20	0 21	0 21	0 21	0 57	0 20
isp9607 txt	N/A	0 22	0 21	0 20	0 22	0 22	0 21	0 22	0 44	0 20

Table A.39 Time taken to convert large fault trees to BDDs using the Basic Hybrid method

FT name	0 scheme	1 scheme	2 scheme	3 scheme	4 scheme	5 scheme	6 scheme	7 scheme	8 scheme	Minimum
rando11 gsq	N/A	13544	15718	15718	7720	7759	6136	7166	13970	6136
rando19 gsq	N/A	1138	1220	1227	1509	981	1363	1441	841	841
rando20 gsq	N/A	593	841	716	704	534	496	951	496	496
rando22 gsq	N/A	1876	1718	1626	3382	1220	1646	3279	1056	1056
rand160 gsq	N/A	22147	16385	12510	50931	17716	35729	18891	11856	11856
lisaba2 gsq	N/A	21569	7833	7833	6032	14703	9740	6181	15563	6032
lisaba8 gsq	N/A	20154	2981	3087	2045	5516	2661	3280	4356	2045
lisab14 gsq	N/A	2050	2637	2621	2821	1632	2451	3527	1376	1376
lisab15 gsq	N/A	2530	3036	3058	3447	969	1137	4822	727	727
lisab22 gsq	N/A	2026	1932	1932	1848	1841	1533	3646	1979	1533
lisab29 gsq	N/A	1	1	1	1	1	1	1	1	1
lisab76 gsq	N/A	2024	3057	3057	1266	1696	1314	1115	1126	1115
lisab87 gsq	N/A	17306	42256	44410	11822	6584	4050	14007	5012	4050
lisab92 gsq	N/A	1	1	1	1	1	1	1	1	1
lisa102 gsq	N/A	78192	1413	1403	1876	3188	1632	3346	2241	1403
lisa117 gsq	N/A	9175	17283	16077	24886	14029	20181	61810	22317	9175
lisa118 gsq	N/A	1276	2046	1674	1631	1009	966	2438	1066	966
baobab1 txt	N/A	-	25015	25015	31047	52307	22919	21461	24912	21461
baobab2 txt	N/A	176627	11500	11500	10474	171023	17999	21839	10114	10114
chinese txt	N/A	89	125	125	125	89	106	100	89	89
das9201 txt	N/A	351	262	262	255	295	243	280	293	243
das9202 txt	N/A	1	1	1	1	1	1	1	1	1
das9203 txt	N/A	29	29	29	29	29	29	29	28	28
das9204 txt	N/A	48	41	41	53	43	41	53	41	41
das9205 txt	N/A	1	1	1	1	1	1	1	1	1
das9206 txt	N/A	6769	1573	1573	1175	5086	1226	2447	3485	1175
das9208 txt	N/A	24333	38535	38535	9287	18323	8742	17255	19900	8742
das9209 txt	N/A	1	1	1	1	1	1	1	1	1
edf9201 txt	N/A	4281	2126	3289	3773	3707	2070	3271	11297	2070
edf9202 txt	N/A	69349	1078	1078	1143	162605	1414	256369	54319	1078
edf9205 txt	N/A	1825	2459	2425	2529	1646	1656	5411	2049	1646
edf9206 txt	N/A	123	74	73	93	86	81	73	110	73
edfpa15o txt	N/A	576731	133893	133893	271134	427951	121088	75036	381055	75036
edfpa15q txt	N/A	336520	194806	194806	46707	569879	201399	47328	224208	46707
edfpa15r txt	N/A	339925	149925	148353	49951	596150	200137	45682	177209	45682
elf9601 txt	N/A	35169	61504	61504	69002	28091	31949	95161	34422	28091
ftr10 txt	N/A	2261	199	199	292	747	337	459	630	199
isp9601 txt	N/A	441	337	337	333	562	428	284	488	284
isp9602 txt	N/A	1837	1006	1006	1170	1592	722	1052	1984	722
isp9603 txt	N/A	5864	4446	4142	3266	4629	1785	2801	2239	1785
isp9604 txt	N/A	278	440	267	431	266	250	470	327	250
isp9605 txt	N/A	397685	12451	12451	11275	400852	21999	23994	570743	11275
isp9606 txt	N/A	184	134	102	145	174	145	119	171	102
isp9607 txt	N/A	902	504	405	472	720	472	512	648	405

Table A 40. Maximum required size for the conversion of large fault trees to BDDs using the ite method

FT name	0 scheme	1 scheme	2 scheme	3 scheme	4 scheme	5 scheme	6 scheme	7 scheme	8 scheme	Minimum
rando11 gsq	N/A	13521	15696	15696	7703	7735	6115	7142	13948	6115
rando19 gsq	N/A	1121	1201	1207	1492	961	1346	1424	821	821
rando20 gsq	N/A	571	819	695	681	513	473	930	474	473
rando22 gsq	N/A	1847	1687	1595	3357	1191	1615	3257	1027	1027
rand160 gsq	N/A	22115	16354	12478	50899	17680	35693	18858	11820	11820
lisaba2 gsq	N/A	21554	7818	7818	6016	14686	9723	6165	15546	6016
lisaba8 gsq	N/A	20131	2958	3064	2022	5492	2637	3258	4333	2022
lisab14 gsq	N/A	2027	2614	2598	2796	1608	2426	3501	1351	1351
lisab15 gsq	N/A	2514	3019	3041	3430	951	1119	4806	709	709
lisab22 gsq	N/A	2002	1907	1907	1823	1815	1507	3626	1952	1507
lisab29 gsq	N/A	1	1	1	1	1	1	1	1	1
lisab76 gsq	N/A	2000	3029	3029	1238	1665	1283	1087	1093	1087
lisab87 gsq	N/A	17286	42239	44393	11802	6562	4028	13990	4989	4028
lisab92 gsq	N/A	1	1	1	1	1	1	1	1	1
lisa102 gsq	N/A	9166	1404	1394	1866	3177	1621	3337	2230	1394
lisa117 gsq	N/A	78170	17264	16057	24857	14000	20150	61783	22291	14000
lisa118 gsq	N/A	1259	2032	1660	1617	992	950	2422	1048	950
baobab1 txt	N/A	-	25013	25013	31638	52407	22965	21501	24911	21501
baobab2 txt	N/A	175858	11415	11415	10384	170874	17747	21705	9936	9936
chinese txt	N/A	89	125	125	125	89	106	100	89	89
das9201 txt	N/A	332	243	243	236	276	224	261	274	224
das9202 txt	N/A	1	1	1	1	1	1	1	1	1
das9203 txt	N/A	29	29	29	29	29	29	29	28	28
das9204 txt	N/A	43	37	37	48	38	36	48	36	36
das9205 txt	N/A	1	1	1	1	1	1	1	1	1
das9206 txt	N/A	6760	1568	1568	1161	5073	1211	2438	3483	1161
das9208 txt	N/A	24531	38370	38370	9597	18357	9207	17882	19602	9207
das9209 txt	N/A	1	1	1	1	1	1	1	1	1
edf9201 txt	N/A	4098	2036	3196	3560	3544	1923	3185	11086	1923
edf9202 txt	N/A	69258	987	987	1052	162514	1323	256340	54228	987
edf9205 txt	N/A	1791	2757	2723	2843	1608	1634	5598	2051	1608
edf9206 txt	N/A	126	74	73	93	86	81	73	110	73
edfpa15o txt	N/A	584086	132522	132522	270374	426733	115759	74481	424173	74481
edfpa15q txt	N/A	339878	195494	195494	47017	574081	200057	47774	226332	47017
edfpa15r txt	N/A	340532	149575	148002	48447	592435	199072	45943	176908	45943
elf9601 txt	N/A	35027	61286	61286	68855	27933	31787	95026	34272	27933
ftr10 txt	N/A	2107	181	181	280	729	318	440	613	181
isp9601 txt	N/A	419	315	315	311	540	406	262	466	262
isp9602 txt	N/A	1807	972	972	1133	1581	686	1016	2024	686
isp9603 txt	N/A	5852	4435	4131	3256	4618	1773	2790	2227	1773
isp9604 txt	N/A	247	413	238	416	239	232	453	290	232
isp9605 txt	N/A	397677	12445	12445	11859	400846	21993	23986	572149	11859
isp9606 txt	N/A	171	120	89	132	161	132	108	155	89
isp9607 txt	N/A	891	493	394	462	710	462	502	638	394

Table A.41: Maximum required size for the conversion of large fault trees to BDDs using the Basic Hybrid method

FT name	1 scheme	2 scheme	3 scheme	4 scheme	5 scheme	6 scheme	7 scheme	8 scheme	Minimum
astoifo gsq	40	26	26	25	40	27	25	48	25
benjam gsq	47	34	34	32	47	39	32	47	32
bpfig03 dat	101	63	63	63	93	63	63	70	63
bpfen05 dat	90	61	61	61	82	61	61	85	61
bpfig05 dat	88	60	60	60	81	60	60	63	60
bpfin05 dat	45	40	40	40	40	40	40	42	40
bpfsw02 gsq	150	61	61	61	150	150	62	150	61
dre1019 dat	19	19	19	19	19	19	19	19	19
dre1032 dat	21	21	21	21	21	21	21	21	21
dre1057 dat	43	32	32	32	43	43	32	32	32
dre1058 dat	164	186	90	80	190	103	70	72	70
dre1059 dat	232	385	404	403	282	167	333	152	152
dresden dat	327	273	87	80	378	103	430	164	80
hpsif02 dat	414	96	96	98	361	357	134	398	96
hpsif03 dat	45	42	42	42	45	45	42	46	42
hpsif21 gsq	220	196	196	239	447	424	210	832	196
hpsif36 gsq	42	44	40	40	40	40	40	42	40
jdtree3 gsq	37	21	21	21	37	35	21	37	21
jdtree4 gsq	31	19	19	19	31	31	19	31	19
jdtree5 gsq	35	20	20	20	35	32	20	35	20
khictre dat	30	30	30	30	30	30	30	30	30
nakashi gsq	147	47	65	80	118	71	49	111	47
trials1 gsq	127	95	125	82	139	66	76	108	66
trials4 gsq	122	171	151	145	124	111	138	101	101
random3 gsq	133	124	130	136	135	132	80	146	80
random6 gsq	1086	4817	4817	5067	1594	1341	5684	1485	1086
random8 gsq	18	14	14	14	18	14	14	18	14
rando12 gsq	413	298	310	317	295	267	361	226	226
rando13 gsq	219	76	76	149	113	129	346	123	76
rando16 gsq	136	75	75	225	122	131	62	131	62
rando18 gsq	303	737	737	64	259	113	64	233	64
rando23 gsq	38	31	31	31	37	37	31	35	31
rando25 gsq	16	13	13	28	17	21	18	12	12
rando27 gsq	109	187	187	270	90	99	217	94	90
rando28 gsq	1	1	1	1	1	1	1	1	1
rando29 gsq	124	183	174	172	124	96	115	108	96
rando30 gsq	295	86	93	94	272	185	90	249	86
rando31 gsq	11	11	11	145	17	25	11	25	11
rando33 gsq	18	31	31	48	16	13	19	152	13
rando34 gsq	64	62	58	58	57	66	44	48	44
rando35 gsq	45	51	54	55	26	27	30	29	26
rando36 gsq	28	17	17	16	20	16	16	18	16
rando37 gsq	71	179	179	203	73	226	70	97	70
rando38 gsq	24	19	19	16	32	22	17	22	16
rando39 gsq	140	170	170	347	135	259	361	237	135
rando40 gsq	27	19	19	24	25	19	24	19	19
rando42 gsq	5	5	5	5	5	5	5	5	5
rando43 gsq	31	28	28	22	35	30	22	31	22
rando44 gsq	734	245	239	385	470	236	461	476	236
rando45 gsq	54	50	50	52	47	35	60	40	35
rando46 gsq	20	16	16	16	23	21	29	30	16
rando47 gsq	128	154	154	81	77	64	86	70	64
rando48 gsq	34	25	23	25	28	26	38	27	23
rando52 gsq	106	55	124	130	115	129	86	112	55

Table A 42: Number of nodes for small fault trees in the Basic/Advanced Hybrid method, 1

FT name	1 scheme	2 scheme	3 scheme	4 scheme	5 scheme	6 scheme	7 scheme	8 scheme	Minimum
rando53 gsq	5	5	5	5	5	5	5	5	5
rando54 gsq	68	45	68	68	70	68	69	61	45
rando55 gsq	24	25	25	21	25	25	24	26	21
rando58 gsq	9	9	9	9	9	9	9	9	9
rando59 gsq	557	239	239	182	452	293	126	235	126
rando60 gsq	98	34	37	34	73	59	42	57	34
rando61 gsq	41	37	33	57	35	44	53	30	30
rando62 gsq	13	12	11	11	11	11	12	11	11
rando63 gsq	24	25	25	21	25	25	24	26	21
rando64 gsq	172	91	101	93	198	104	56	103	56
rando65 gsq	35	33	21	29	33	38	32	38	21
rando66 gsq	59	32	32	47	54	46	56	53	32
rando70 gsq	42	35	35	35	43	37	33	48	33
rando73 gsq	136	114	127	127	108	105	240	64	64
rando75 gsq	13	13	13	13	11	13	13	11	11
rando76 gsq	71	30	39	39	39	45	39	40	30
rando77 gsq	72	82	82	60	48	48	132	41	41
rando78 gsq	5	5	8	8	5	5	5	5	5
rando80 gsq	31	26	26	29	31	29	36	29	26
rando83 gsq	62	54	58	34	46	34	58	47	34
rando84 gsq	179	91	93	84	121	83	70	96	70
rando85 gsq	15	15	15	17	15	15	15	22	15
rando87 gsq	19	20	20	16	15	15	15	15	15
rando88 gsq	64	31	49	37	41	38	52	45	31
rando89 gsq	34	28	30	34	33	33	39	32	28
rando91 gsq	490	1353	1341	696	432	458	1381	338	338
rando92 gsq	326	444	492	484	313	304	303	291	291
rando93 gsq	43	47	46	29	34	31	45	31	29
rando95 gsq	45	35	43	43	45	45	43	40	35
rando98 gsq	301	192	192	250	290	240	184	145	145
rando99 gsq	151	343	342	290	153	176	216	135	135
rand100 gsq	13	13	13	13	15	15	11	15	11
rand103 gsq	52	34	34	42	51	46	38	49	34
rand104 gsq	35	31	31	32	40	34	19	34	19
rand105 gsq	103	60	61	52	74	55	49	95	49
rand106 gsq	14	877	877	13	13	17	13	16	13
rand108 gsq	95	219	199	147	80	92	216	86	80
rand109 gsq	239	127	137	127	254	167	220	240	127
rand110 gsq	27	28	28	29	29	29	13	30	13
rand111 gsq	79	99	91	66	74	71	47	62	47
rand115 gsq	176	111	111	168	161	112	111	94	94
rand116 gsq	98	112	142	158	82	188	172	63	63
rand117 gsq	27	29	27	29	27	27	25	30	25
rand118 gsq	179	91	93	84	121	83	70	96	70
rand119 gsq	47	30	52	29	34	32	30	34	29
rand120 gsq	395	82	84	150	373	198	95	372	82
rand121 gsq	56	44	44	47	47	39	135	43	39
rand123 gsq	18	17	17	19	17	17	17	17	17
rand124 gsq	29	21	23	23	27	31	21	27	21
rand125 gsq	18	20	22	17	17	17	22	17	17
rand126 gsq	109	117	110	138	121	115	100	115	100
rand127 gsq	66	29	29	31	63	33	31	50	29
rand128 gsq	157	70	71	93	154	126	85	91	70
rand129 gsq	4	4	4	4	4	4	4	4	4

Table A.43: Number of nodes for small fault trees in the Basic/Advanced Hybrid method, 2

FT name	1 scheme	2 scheme	3 scheme	4 scheme	5 scheme	6 scheme	7 scheme	8 scheme	Minimum
rand130 gsq	5	5	5	5	5	5	5	5	5
rand132 gsq	467	587	587	1084	376	845	1062	561	376
rand134 gsq	122	330	332	614	118	315	453	109	109
rand135 gsq	64	108	103	134	70	57	120	88	57
rand137 gsq	40	32	32	32	35	32	32	49	32
rand138 gsq	2	2	2	2	2	2	2	2	2
rand139 gsq	122	76	77	191	121	190	113	146	76
rand141 gsq	27	28	28	29	29	29	13	30	13
rand143 gsq	33	38	20	17	26	26	19	30	17
rand144 gsq	188	402	334	284	177	214	377	211	177
rand145 gsq	46	37	37	34	43	34	37	43	34
rand146 gsq	40	32	32	32	35	32	32	49	32
rand147 gsq	234	1474	1750	351	247	531	485	239	234
rand148 gsq	24	20	14	15	21	15	15	15	14
rand149 gsq	33	21	21	21	36	24	21	36	21
rand150 gsq	785	1896	1418	1634	997	960	1348	992	785
rand151 gsq	34	25	25	25	34	34	25	34	25
rand153 gsq	8	8	8	8	8	8	8	8	8
rand154 gsq	1	1	1	1	1	1	1	1	1
rand155 gsq	226	146	115	97	158	106	172	174	97
rand156 gsq	29	22	22	22	22	22	28	22	22
rand158 gsq	27	18	18	21	20	21	21	18	18
lisaba1 gsq	1003	152	185	127	478	142	152	364	127
lisaba3 gsq	804	821	760	659	649	503	881	767	503
lisaba4 gsq	420	278	294	246	276	273	148	203	148
lisaba5 gsq	197	146	130	128	158	227	205	153	128
lisaba6 gsq	192	137	131	125	154	124	148	165	124
lisaba7 gsq	1003	152	185	127	478	142	152	364	127
lisaba9 gsq	167	59	59	57	127	56	55	107	55
lisab10 gsq	820	534	400	302	372	277	476	243	243
lisab11 gsq	5	5	5	5	5	5	5	5	5
lisab13 gsq	27	28	28	29	29	29	13	30	13
lisab17 gsq	1003	152	185	127	478	142	152	364	127
lisab24 gsq	107	41	41	41	81	71	41	77	41
lisab25 gsq	65	64	69	55	62	56	49	66	49
lisab26 gsq	7	7	7	7	7	7	7	7	7
lisab27 gsq	282	313	268	141	270	144	94	200	94
lisab28 gsq	38	22	22	22	30	27	22	30	22
lisab30 gsq	44	37	37	34	40	34	36	33	33
lisab31 gsq	332	506	506	245	362	176	422	330	176
lisab34 gsq	22	20	20	20	23	20	28	23	20
lisab35 gsq	645	449	449	369	546	592	596	342	342
lisab36 gsq	134	114	110	324	162	267	79	98	79
lisab37 gsq	74	36	36	38	59	38	31	46	31
lisab42 gsq	19	17	17	17	17	17	18	17	17
lisab44 gsq	32	32	32	41	33	41	32	41	32
lisab47 gsq	11	11	11	11	12	11	11	16	11
lisab48 gsq	4	4	4	4	4	4	4	4	4
lisab49 gsq	146	60	60	60	160	92	60	178	60
lisab50 gsq	11	8	8	8	10	10	11	10	8
lisab51 gsq	33	27	27	30	42	36	24	36	24
lisab52 gsq	598	778	778	659	423	524	623	350	350
lisab53 gsq	11	11	11	9	11	11	9	9	9
lisab54 gsq	25	22	20	20	20	20	20	22	20

Table A.44: Number of nodes for small fault trees in the Basic/Advanced Hybrid method, 3

FT name	1 scheme	2 scheme	3 scheme	4 scheme	5 scheme	6 scheme	7 scheme	8 scheme	Minimum
lisab55 gsq	5	5	5	5	5	5	5	5	5
lisab56 gsq	3	3	3	3	3	3	3	3	3
lisab57 gsq	137	118	110	110	131	125	151	175	110
lisab59 gsq	133	49	49	49	172	81	49	144	49
lisab60 gsq	34	29	29	25	35	25	25	30	25
lisab61 gsq	51	33	33	33	46	52	42	66	33
lisab62 gsq	88	69	53	84	84	111	69	97	53
lisab63 gsq	35	26	26	23	29	23	23	30	23
lisab64 gsq	12	12	12	12	12	12	13	12	12
lisab65 gsq	24	19	19	19	24	24	19	24	19
lisab66 gsq	64	181	181	54	42	43	207	38	38
lisab67 gsq	378	224	222	163	204	197	340	178	163
lisab68 gsq	30	25	25	25	30	30	25	30	25
lisab69 gsq	66	52	50	44	56	64	56	53	44
lisab70 gsq	191	162	162	78	156	97	107	134	78
lisab71 gsq	8	8	8	8	8	8	8	8	8
lisab72 gsq	210	225	204	165	167	132	448	150	132
lisab73 gsq	23	23	23	23	33	23	23	23	23
lisab75 gsq	4	4	4	4	4	4	4	4	4
lisab77 gsq	632	474	442	406	604	316	358	537	316
lisab78 gsq	313	111	149	111	196	140	195	167	111
lisab81 gsq	15	15	15	15	15	15	15	15	15
lisab82 gsq	1687	439	439	777	1015	934	476	904	439
lisab83 gsq	153	83	135	106	123	105	52	106	52
lisab84 gsq	153	83	135	106	123	105	52	106	52
lisab86 gsq	202	230	199	160	207	198	261	195	160
lisab88 gsq	107	45	45	51	76	72	165	63	45
lisab89 gsq	579	180	335	156	559	373	67	462	67
lisab91 gsq	595	134	133	131	377	223	128	315	128
lisab95 gsq	3	3	3	3	3	3	3	3	3
lisab97 gsq	29	22	22	22	29	31	22	22	22
lisab98 gsq	11	11	11	11	11	11	11	11	11
lisa100 gsq	1519	523	530	600	1158	845	664	705	523
lisa103 gsq	11	11	11	9	11	11	9	9	9
lisa104 gsq	13	13	13	13	13	13	13	13	13
lisa107 gsq	8	8	8	8	8	8	8	8	8
lisa109 gsq	55	79	48	49	44	37	115	54	37
lisa110 gsq	316	385	367	374	432	382	316	522	316
lisa111 gsq	134	47	47	47	107	74	47	106	47
lisa112 gsq	1623	234	239	505	1725	830	371	1202	234
lisa113 gsq	245	238	239	278	209	163	233	184	163
lisa115 gsq	53	46	36	36	39	33	30	37	30
lisa116 gsq	11	8	8	8	7	8	6	8	6
lisa119 gsq	25	15	15	15	17	17	16	17	15
lisa120 gsq	41	24	24	24	37	26	24	33	24
lisa121 gsq	168	41	41	57	136	71	44	107	41
lisa122 gsq	43	38	38	38	38	38	38	35	35
lisa123 gsq	82	40	40	33	75	44	84	124	33
lisa124 gsq	1228	674	660	654	1293	503	774	1134	503
rand159 gsq	195	244	225	225	165	73	81	121	73
rand161 gsq	191	354	331	340	208	189	362	217	189
rand162 gsq	1	1	1	1	1	1	1	1	1
rand163 gsq	556	1196	890	889	490	398	3532	437	398
rand164 gsq	1419	433	263	257	583	247	275	451	247

Table A 45 Number of nodes for small fault trees in the Basic/Advanced Hybrid method, 4

FT name	1 scheme	2 scheme	3 scheme	4 scheme	5 scheme	6 scheme	7 scheme	8 scheme	Minimum
astolfo gsq	0 37	0 42	0 41	0 36	0 45	0 41	0 45	0 41	0 36
benjam gsq	0 35	0 42	0 42	0 38	0 41	0 42	0 42	0 43	0 35
bpfeq03 dat	0 4	0 44	0 46	0 42	0 46	0 47	0 45	0 45	0 4
bpfen05 dat	0 37	0 43	0 44	0 39	0 44	0 45	0 43	0 44	0 37
bpfig05 dat	0 36	0 43	0 43	0 39	0 46	0 46	0 44	0 43	0 36
bpfin05 dat	0 35	0 42	0 43	0 37	0 44	0 43	0 42	0 42	0 35
bpfsw02 gsq	0 22	0 48	0 48	0 44	0 27	0 29	0 47	0 27	0 22
dre1019 dat	0 35	0 42	0 43	0 37	0 43	0 42	0 41	0 42	0 35
dre1032 dat	0 35	0 41	0 42	0 37	0 44	0 42	0 41	0 41	0 35
dre1057 dat	0 35	0 42	0 43	0 39	0 4	0 43	0 4	0 42	0 35
dre1058 dat	0 22	0 28	0 44	0 39	0 25	0 42	0 41	0 43	0 22
dre1059 dat	0 22	0 28	0 28	0 25	0 26	0 25	0 28	0 28	0 22
dresden dat	0 24	0 34	0 43	0 4	0 31	0 41	0 33	0 44	0 24
hpsif02 dat	0 24	0 43	0 44	0 39	0 29	0 29	0 38	0 3	0 24
hpsif03 dat	0 35	0 42	0 42	0 38	0 42	0 43	0 42	0 39	0 35
hpsif21 gsq	0 22	0 28	0 28	0 25	0 27	0 29	0 29	0 3	0 22
hpsif36 gsq	0 35	0 42	0 42	0 39	0 41	0 43	0 41	0 43	0 35
ldtree3 gsq	0 34	0 42	0 42	0 44	0 42	0 41	0 43	0 44	0 34
ldtree4 gsq	0 35	0 42	0 42	0 46	0 41	0 43	0 41	0 45	0 35
ldtree5 gsq	0 34	0 42	0 43	0 46	0 41	0 42	0 42	0 46	0 34
khictre dat	0 35	0 42	0 42	0 48	0 42	0 42	0 42	0 42	0 35
nakashi gsq	0 21	0 43	0 42	0 3	0 26	0 41	0 44	0 17	0 17
trials1 gsq	0 22	0 26	0 28	0 31	0 26	0 43	0 26	0 38	0 22
trials4 gsq	0 21	0 28	0 28	0 3	0 27	0 26	0 29	0 3	0 21
random3 gsq	0 22	0 27	0 27	0 28	0 42	0 41	0 29	0 21	0 21
random6 gsq	0 28	1 09	1 1	1 18	0 4	0 34	1 49	0 46	0 28
random8 gsq	0 35	0 42	0 43	0 35	0 42	0 44	0 43	0 42	0 35
rando12 gsq	0 23	0 28	0 27	0 23	0 16	0 3	0 29	0 29	0 16
rando13 gsq	0 25	0 28	0 29	0 26	0 17	0 31	0 36	0 27	0 17
rando16 gsq	0 31	0 32	0 46	0 29	0 28	0 31	0 5	0 33	0 28
rando18 gsq	3 02	13 89	13 27	29 91	2 01	10 78	27 7	2 18	2 01
rando23 gsq	0 35	0 42	0 42	0 35	0 41	0 46	0 37	0 43	0 35
rando25 gsq	0 34	0 42	0 4	0 41	0 41	0 51	0 39	0 42	0 34
rando27 gsq	0 42	0 37	0 37	0 43	0 51	0 59	0 5	0 52	0 37
rando28 gsq	0 35	0 42	0 43	0 36	0 41	0 46	0 42	0 43	0 35
rando29 gsq	0 2	0 28	0 27	0 22	0 27	0 29	0 28	0 23	0 2
rando30 gsq	0 22	0 43	0 42	0 35	0 26	0 28	0 44	0 23	0 22
rando31 gsq	0 39	0 45	0 45	0 43	0 45	0 5	0 47	0 39	0 39
rando33 gsq	0 35	0 42	0 42	0 35	0 4	0 51	0 43	0 35	0 35
rando34 gsq	0 35	0 43	0 43	0 35	0 42	0 47	0 44	0 37	0 35
rando35 gsq	0 36	0 41	0 42	0 34	0 42	0 37	0 45	0 41	0 34
rando36 gsq	0 35	0 42	0 43	0 34	0 42	0 34	0 47	0 41	0 34
rando37 gsq	0 34	0 28	0 27	0 22	0 41	0 26	0 46	0 35	0 22
rando38 gsq	0 35	0 42	0 44	0 35	0 42	0 44	0 43	0 36	0 35
rando39 gsq	0 21	0 43	0 41	0 23	0 27	0 28	0 33	0 25	0 21
rando40 gsq	0 35	0 42	0 42	0 36	0 4	0 4	0 44	0 39	0 35
rando42 gsq	0 36	0 42	0 43	0 34	0 41	0 26	0 44	0 35	0 26
rando43 gsq	0 36	0 42	0 42	0 36	0 41	0 3	0 43	0 35	0 3
rando44 gsq	0 25	0 28	0 28	0 22	0 3	0 18	0 33	0 25	0 18
rando45 gsq	0 35	0 41	0 42	0 34	0 42	0 34	0 46	0 41	0 34
rando46 gsq	0 36	0 42	0 43	0 35	0 42	0 4	0 44	0 43	0 35
rando47 gsq	0 22	0 28	0 28	0 35	0 42	0 4	0 49	0 42	0 22
rando48 gsq	0 37	0 42	0 42	0 34	0 41	0 43	0 47	0 42	0 34
rando52 gsq	0 36	0 44	0 46	0 37	0 43	0 45	0 48	0 41	0 36

Table A 46: Processing time for small fault trees in the Basic Hybrid method, 1

FT name	1 scheme	2 scheme	3 scheme	4 scheme	5 scheme	6 scheme	7 scheme	8 scheme	Minimum
rando53 gsq	0 35	0 42	0 41	0 35	0 42	0 45	0 47	0 4	0 35
rando54 gsq	0 35	0 43	0 44	0 35	0 41	0 42	0 49	0 42	0 35
rando55 gsq	0 35	0 42	0 41	0 35	0 41	0 46	0 45	0 43	0 35
rando58 gsq	0 34	0 42	0 43	0 35	0 41	0 45	0 44	0 44	0 34
rando59 gsq	0 24	0 28	0 26	0 22	0 27	0 3	0 29	0 27	0 22
rando60 gsq	0 35	0 42	0 43	0 34	0 44	0 47	0 44	0 44	0 34
rando61 gsq	0 34	0 42	0 43	0 34	0 44	0 45	0 48	0 44	0 34
rando62 gsq	0 35	0 42	0 43	0 35	0 43	0 43	0 45	0 45	0 35
rando63 gsq	0 35	0 43	0 42	0 34	0 43	0 44	0 43	0 45	0 34
rando64 gsq	0 21	0 42	0 28	0 33	0 28	0 47	0 47	0 28	0 21
rando65 gsq	0 34	0 42	0 43	0 34	0 43	0 44	0 44	0 43	0 34
rando66 gsq	0 34	0 42	0 43	0 36	0 42	0 44	0 31	0 43	0 31
rando70 gsq	0 35	0 42	0 44	0 34	0 43	0 43	0 4	0 41	0 34
rando73 gsq	0 21	0 43	0 44	0 35	0 39	0 46	0 39	0 26	0 21
rando75 gsq	0 35	0 42	0 43	0 34	0 43	0 45	0 43	0 29	0 29
rando76 gsq	0 34	0 42	0 4	0 35	0 41	0 42	0 43	0 41	0 34
rando77 gsq	0 34	0 43	0 43	0 36	0 43	0 48	0 44	0 4	0 34
rando78 gsq	0 34	0 42	0 43	0 35	0 42	0 46	0 43	0 41	0 34
rando80 gsq	0 35	0 43	0 43	0 34	0 41	0 44	0 43	0 42	0 34
rando83 gsq	0 33	0 42	0 44	0 35	0 41	0 45	0 43	0 41	0 33
rando84 gsq	0 23	0 27	0 29	0 21	0 28	0 31	0 44	0 42	0 21
rando85 gsq	0 34	0 42	0 45	0 35	0 41	0 46	0 43	0 42	0 34
rando87 gsq	0 35	0 43	0 43	0 35	0 38	0 49	0 43	0 42	0 35
rando88 gsq	0 34	0 42	0 41	0 34	0 43	0 44	0 43	0 42	0 34
rando89 gsq	0 36	0 43	0 45	0 35	0 39	0 45	0 44	0 43	0 35
rando91 gsq	0 23	0 31	0 32	0 24	0 28	0 28	0 31	0 28	0 23
rando92 gsq	1 16	2 12	2 9	3 7	0 82	0 76	0 51	1 16	0 51
rando93 gsq	0 34	0 43	0 44	0 36	0 42	0 44	0 44	0 42	0 34
rando95 gsq	0 36	0 41	0 43	0 34	0 42	0 43	0 43	0 43	0 34
rando98 gsq	0 23	0 28	0 28	0 24	0 28	0 29	0 25	0 28	0 23
rando99 gsq	0 35	0 3	0 28	0 23	0 27	0 29	0 26	0 27	0 23
rand100 gsq	0 35	0 42	0 43	0 34	0 42	0 43	0 42	0 42	0 34
rand103 gsq	0 35	0 42	0 43	0 34	0 42	0 31	0 43	0 42	0 31
rand104 gsq	0 31	0 43	0 42	0 34	0 42	0 29	0 42	0 39	0 29
rand105 gsq	0 33	0 28	0 42	0 35	0 42	0 31	0 43	0 41	0 28
rand106 gsq	0 35	0 49	0 49	0 36	0 42	0 42	0 44	0 38	0 35
rand108 gsq	0 36	0 28	0 28	0 22	0 43	0 43	0 28	0 42	0 22
rand109 gsq	0 21	0 27	0 27	0 22	0 28	0 29	0 27	0 27	0 21
rand110 gsq	0 35	0 43	0 43	0 35	0 42	0 44	0 43	0 39	0 35
rand111 gsq	0 35	0 42	0 43	0 34	0 45	0 31	0 43	0 4	0 31
rand115 gsq	0 21	0 42	0 42	0 21	0 28	0 27	0 27	0 36	0 21
rand116 gsq	0 36	0 27	0 27	0 24	0 41	0 15	0 29	0 37	0 15
rand117 gsq	0 34	0 43	0 41	0 35	0 39	0 34	0 42	0 41	0 34
rand118 gsq	0 22	0 27	0 29	0 21	0 27	0 27	0 43	0 41	0 21
rand119 gsq	0 34	0 42	0 42	0 34	0 43	0 37	0 43	0 41	0 34
rand120 gsq	0 22	0 42	0 43	0 22	0 29	0 27	0 42	0 28	0 22
rand121 gsq	0 35	0 43	0 42	0 35	0 42	0 37	0 43	0 42	0 35
rand123 gsq	0 35	0 42	0 42	0 34	0 43	0 4	0 42	0 43	0 34
rand124 gsq	0 35	0 42	0 43	0 35	0 41	0 45	0 43	0 38	0 35
rand125 gsq	0 35	0 43	0 43	0 34	0 41	0 4	0 41	0 43	0 34
rand126 gsq	0 21	0 28	0 27	0 22	0 27	0 31	0 43	0 28	0 21
rand127 gsq	0 35	0 43	0 43	0 35	0 42	0 45	0 42	0 43	0 35
rand128 gsq	0 22	0 43	0 42	0 34	0 29	0 29	0 43	0 43	0 22
rand129 gsq	0 34	0 42	0 43	0 34	0 41	0 47	0 42	0 42	0 34

Table A 47 Processing time for small fault trees in the Basic Hybrid method, 2

FT name	1 scheme	2 scheme	3 scheme	4 scheme	5 scheme	6 scheme	7 scheme	8 scheme	Minimum
rand130 gsq	0 34	0 43	0 42	0 35	0 43	0 47	0 42	0 42	0 34
rand132 gsq	0 22	0 28	0 29	0 24	0 26	0 33	0 31	0 29	0 22
rand134 gsq	0 22	0 28	0 28	0 24	0 27	0 34	0 3	0 28	0 22
rand135 gsq	0 35	0 42	0 43	0 34	0 42	0 5	0 43	0 42	0 34
rand137 gsq	0 35	0 42	0 43	0 35	0 42	0 5	0 42	0 42	0 35
rand138 gsq	0 34	0 42	0 42	0 34	0 43	0 47	0 38	0 42	0 34
rand139 gsq	0 22	0 43	0 43	0 21	0 44	0 33	0 27	0 43	0 21
rand141 gsq	0 35	0 43	0 44	0 36	0 44	0 45	0 41	0 41	0 35
rand143 gsq	0 35	0 43	0 42	0 36	0 43	0 42	0 41	0 42	0 35
rand144 gsq	0 22	0 29	0 29	0 23	0 27	0 27	0 3	0 27	0 22
rand145 gsq	0 34	0 42	0 43	0 35	0 42	0 43	0 43	0 43	0 34
rand146 gsq	0 35	0 42	0 43	0 35	0 43	0 42	0 42	0 41	0 35
rand147 gsq	0 22	0 4	0 44	0 22	0 27	0 3	0 28	0 27	0 22
rand148 gsq	0 35	0 43	0 47	0 34	0 41	0 45	0 41	0 32	0 32
rand149 gsq	0 34	0 43	0 44	0 37	0 43	0 43	0 27	0 36	0 27
rand150 gsq	0 24	0 36	0 34	0 27	0 3	0 3	0 21	0 28	0 21
rand151 gsq	0 34	0 42	0 45	0 35	0 43	0 44	0 32	0 4	0 32
rand153 gsq	0 35	0 42	0 45	0 35	0 44	0 43	0 43	0 36	0 35
rand154 gsq	0 35	0 42	0 44	0 33	0 42	0 44	0 42	0 44	0 33
rand155 gsq	0 21	0 27	0 27	0 22	0 27	0 43	0 28	0 28	0 21
rand156 gsq	0 35	0 43	0 44	0 35	0 42	0 42	0 43	0 37	0 35
rand158 gsq	3 01	1 56	1 56	1 48	1 9	1 33	1 44	1 58	1 33
lisaba1 gsq	0 25	0 31	0 33	0 38	0 3	0 38	0 46	0 32	0 25
lisaba3 gsq	0 29	0 41	0 4	0 28	0 33	0 25	0 33	0 38	0 25
lisaba4 gsq	0 23	0 28	0 28	0 22	0 27	0 3	0 28	0 27	0 22
lisaba5 gsq	0 23	0 29	0 44	0 36	0 24	0 32	0 27	0 27	0 23
lisaba6 gsq	0 22	0 27	0 29	0 34	0 15	0 29	0 43	0 28	0 15
lisaba7 gsq	0 26	0 31	0 34	0 37	0 29	0 37	0 46	0 3	0 26
lisaba9 gsq	0 22	0 42	0 42	0 35	0 23	0 25	0 42	0 42	0 22
lisab10 gsq	0 22	0 3	0 29	0 21	0 27	0 23	0 29	0 27	0 21
lisab11 gsq	0 35	0 42	0 42	0 35	0 42	0 41	0 43	0 42	0 35
lisab13 gsq	0 36	0 44	0 44	0 37	0 43	0 36	0 43	0 43	0 36
lisab17 gsq	0 25	0 3	0 35	0 38	0 3	0 44	0 45	0 31	0 25
lisab24 gsq	0 35	0 43	0 42	0 35	0 42	0 28	0 43	0 37	0 28
lisab25 gsq	0 35	0 27	0 43	0 35	0 43	0 43	0 42	0 39	0 27
lisab26 gsq	0 35	0 42	0 42	0 34	0 44	0 47	0 41	0 4	0 34
lisab27 gsq	0 22	0 27	0 28	0 23	0 29	0 29	0 43	0 29	0 22
lisab28 gsq	0 35	0 42	0 43	0 34	0 42	0 43	0 43	0 42	0 34
lisab30 gsq	0 34	0 43	0 42	0 35	0 43	0 43	0 41	0 43	0 34
lisab31 gsq	0 23	0 3	0 31	0 23	0 26	0 29	0 32	0 28	0 23
lisab34 gsq	0 35	0 43	0 43	0 34	0 37	0 43	0 42	0 4	0 34
lisab35 gsq	0 22	0 28	0 28	0 22	0 29	0 28	0 28	0 22	0 22
lisab36 gsq	0 24	0 3	0 29	0 24	0 29	0 29	0 44	0 26	0 24
lisab37 gsq	0 34	0 42	0 42	0 35	0 43	0 42	0 42	0 43	0 34
lisab42 gsq	0 34	0 42	0 42	0 35	0 41	0 42	0 42	0 42	0 34
lisab44 gsq	0 36	0 42	0 42	0 36	0 43	0 43	0 42	0 45	0 36
lisab47 gsq	0 34	0 42	0 42	0 35	0 43	0 43	0 41	0 38	0 34
lisab48 gsq	0 34	0 43	0 44	0 35	0 44	0 43	0 44	0 41	0 34
lisab49 gsq	0 22	0 44	0 44	0 36	0 28	0 44	0 47	0 29	0 22
lisab50 gsq	0 35	0 42	0 42	0 34	0 38	0 44	0 45	0 4	0 34
lisab51 gsq	0 34	0 43	0 43	0 34	0 4	0 43	0 45	0 37	0 34
lisab52 gsq	0 22	0 3	0 3	0 23	0 28	0 29	0 3	0 27	0 22
lisab53 gsq	0 34	0 42	0 43	0 35	0 43	0 44	0 43	0 38	0 34
lisab54 gsq	0 35	0 42	0 43	0 34	0 41	0 44	0 46	0 45	0 34

Table A 48. Processing time for small fault trees in the Basic Hybrid method, 3

FT name	1 scheme	2 scheme	3 scheme	4 scheme	5 scheme	6 scheme	7 scheme	8 scheme	Minimum
lisab55 gsq	0 34	0 41	0 43	0 35	0 43	0 43	0 43	0 41	0 34
lisab56 gsq	0 35	0 42	0 43	0 35	0 43	0 43	0 43	0 36	0 35
lisab57 gsq	0 22	0 27	0 27	0 21	0 27	0 28	0 28	0 25	0 21
lisab59 gsq	0 22	0 42	0 44	0 36	0 28	0 44	0 46	0 22	0 22
lisab60 gsq	0 35	0 42	0 42	0 36	0 43	0 43	0 43	0 41	0 35
lisab61 gsq	0 35	0 42	0 44	0 35	0 43	0 43	0 42	0 44	0 35
lisab62 gsq	0 35	0 43	0 42	0 35	0 43	0 42	0 42	0 35	0 35
lisab63 gsq	0 35	0 42	0 43	0 35	0 41	0 43	0 46	0 28	0 28
lisab64 gsq	0 35	0 43	0 43	0 36	0 43	0 42	0 44	0 44	0 35
lisab65 gsq	0 36	0 43	0 44	0 34	0 42	0 42	0 43	0 42	0 34
lisab66 gsq	0 35	0 43	0 44	0 35	0 44	0 42	0 46	0 39	0 35
lisab67 gsq	0 23	0 29	0 3	0 22	0 27	0 29	0 29	0 27	0 22
lisab68 gsq	0 35	0 42	0 43	0 35	0 45	0 39	0 33	0 42	0 33
lisab69 gsq	0 35	0 42	0 43	0 35	0 44	0 42	0 32	0 42	0 32
lisab70 gsq	0 21	0 28	0 29	0 35	0 28	0 43	0 22	0 28	0 21
lisab71 gsq	0 35	0 43	0 43	0 35	0 44	0 38	0 43	0 36	0 35
lisab72 gsq	0 22	0 3	0 29	0 22	0 29	0 29	0 3	0 24	0 22
lisab73 gsq	0 34	0 43	0 42	0 35	0 45	0 43	0 43	0 39	0 34
lisab75 gsq	0 34	0 42	0 43	0 35	0 42	0 41	0 43	0 4	0 34
lisab77 gsq	0 22	0 28	0 29	0 22	0 29	0 27	0 3	0 24	0 22
lisab78 gsq	0 22	0 27	0 27	0 35	0 29	0 28	0 29	0 22	0 22
lisab81 gsq	0 34	0 43	0 43	0 35	0 44	0 42	0 43	0 37	0 34
lisab82 gsq	1 27	1 19	1 2	0 83	1 73	0 71	0 51	1 33	0 51
lisab83 gsq	0 22	0 28	0 29	0 35	0 27	0 42	0 44	0 42	0 22
lisab84 gsq	0 21	0 28	0 27	0 35	0 29	0 43	0 43	0 36	0 21
lisab86 gsq	0 24	0 27	0 28	0 22	0 28	0 28	0 28	0 23	0 22
lisab88 gsq	0 38	0 43	0 44	0 34	0 42	0 42	0 27	0 44	0 27
lisab89 gsq	0 25	0 29	0 28	0 21	0 3	0 28	0 44	0 27	0 21
lisab91 gsq	0 25	0 29	0 29	0 23	0 27	0 3	0 28	0 28	0 23
lisab95 gsq	0 35	0 43	0 44	0 34	0 3	0 42	0 43	0 42	0 3
lisab97 gsq	0 35	0 41	0 42	0 35	0 36	0 43	0 44	0 42	0 35
lisab98 gsq	0 33	0 42	0 43	0 34	0 44	0 43	0 44	0 41	0 33
lisa100 gsq	0 3	0 31	0 3	0 26	0 34	0 33	0 32	0 26	0 26
lisa103 gsq	0 34	0 42	0 42	0 34	0 44	0 43	0 43	0 4	0 34
lisa104 gsq	0 34	0 42	0 44	0 34	0 45	0 42	0 43	0 41	0 34
lisa107 gsq	0 34	0 42	0 43	0 35	0 45	0 42	0 43	0 38	0 34
lisa109 gsq	0 34	0 42	0 43	0 35	0 45	0 42	0 44	0 42	0 34
lisa110 gsq	0 23	0 3	0 29	0 23	0 32	0 3	0 29	0 26	0 23
lisa111 gsq	0 21	0 43	0 42	0 36	0 44	0 43	0 43	0 26	0 21
lisa112 gsq	0 3	0 29	0 31	0 24	0 37	0 31	0 31	0 31	0 24
lisa113 gsq	0 24	0 3	0 31	0 22	0 29	0 28	0 29	0 28	0 22
lisa115 gsq	0 34	0 42	0 42	0 34	0 45	0 42	0 43	0 41	0 34
lisa116 gsq	0 35	0 42	0 42	0 35	0 42	0 43	0 44	0 41	0 35
lisa119 gsq	0 34	0 43	0 43	0 36	0 44	0 42	0 43	0 36	0 34
lisa120 gsq	0 34	0 43	0 43	0 35	0 42	0 4	0 45	0 44	0 34
lisa121 gsq	0 22	0 42	0 44	0 35	0 28	0 43	0 43	0 28	0 22
lisa122 gsq	0 35	0 41	0 43	0 34	0 42	0 42	0 44	0 44	0 34
lisa123 gsq	0 34	0 43	0 43	0 36	0 43	0 44	0 44	0 42	0 34
lisa124 gsq	0 25	0 3	0 31	0 24	0 31	0 3	0 29	0 32	0 24
rand159 gsq	0 22	0 28	0 29	0 22	0 28	0 44	0 45	0 39	0 22
rand161 gsq	0 22	0 29	0 26	0 22	0 27	0 27	0 28	0 28	0 22
rand162 gsq	0 35	0 42	0 43	0 35	0 41	0 44	0 42	0 41	0 35
rand163 gsq	0 23	0 33	0 31	0 27	0 3	0 32	0 6	0 28	0 23
rand164 gsq	0 35	0 39	0 3	0 24	0 34	0 29	0 26	0 29	0 24

Table A.49 Processing time for small fault trees in the Basic Hybrid method, 4

FT name	1 scheme	2 scheme	3 scheme	4 scheme	5 scheme	6 scheme	7 scheme	8 scheme	Minimum
astolfo gsq	72	75	72	58	74	63	76	83	58
benjam gsq	83	82	82	79	83	74	82	108	74
bpfig03 dat	287	331	270	259	280	259	341	270	259
bpfen05 dat	263	313	255	242	259	242	320	264	242
bpfig05 dat	258	308	251	237	254	237	314	242	237
bpfin05 dat	160	215	143	143	143	143	207	148	143
bpfs02 gsq	217	151	151	151	217	217	153	217	151
dre1019 dat	45	45	45	45	45	45	49	45	45
dre1032 dat	55	55	55	49	49	49	55	49	49
dre1057 dat	92	100	90	84	86	86	103	84	84
dre1058 dat	269	415	250	223	302	201	294	199	199
dre1059 dat	345	644	677	676	379	276	586	297	276
dresden dat	859	1818	806	739	1215	805	1816	775	739
hpsif02 dat	650	292	292	338	662	644	471	893	292
hpsif03 dat	113	108	108	108	113	113	111	117	108
hpsif21 gsq	829	683	683	900	976	871	803	2072	683
hpsif36 gsq	134	136	128	122	126	122	136	142	122
jdtree3 gsq	67	59	59	59	67	65	59	67	59
jdtree4 gsq	72	66	66	66	72	72	66	72	66
jdtree5 gsq	66	58	58	58	66	64	58	66	58
khictre dat	199	176	199	196	196	196	176	192	176
nakashi gsq	245	113	143	141	183	130	118	172	113
trials1 gsq	314	306	323	265	323	236	274	279	236
trials4 gsq	466	603	527	474	414	401	486	373	373
random3 gsq	640	735	643	671	612	569	492	633	492
random6 gsq	1979	6846	6845	7629	2788	2685	8040	3080	1979
random8 gsq	77	71	73	73	77	71	71	71	71
rando12 gsq	774	621	631	658	594	591	746	558	558
rando13 gsq	998	1103	1103	1660	889	1213	1735	936	889
rando16 gsq	1918	1814	1577	1839	1893	1722	1509	1899	1509
rando18 gsq	12987	26717	25915	38697	9996	22673	36210	9830	9830
rando23 gsq	332	376	376	369	280	293	375	249	249
rando25 gsq	92	80	80	132	94	113	98	82	80
rando27 gsq	2283	2599	2597	2261	2590	2132	1972	2298	1972
rando28 gsq	258	260	250	249	244	246	253	254	244
rando29 gsq	467	606	468	466	446	390	430	403	390
rando30 gsq	380	192	205	203	356	298	214	336	192
rando31 gsq	1552	1431	1499	2209	1490	1537	1437	1668	1431
rando33 gsq	185	179	179	210	187	172	201	312	172
rando34 gsq	416	396	461	465	394	406	407	336	336
rando35 gsq	382	476	428	425	301	301	283	293	283
rando36 gsq	162	150	157	167	168	161	163	159	150
rando37 gsq	402	566	566	602	397	582	410	447	397
rando38 gsq	117	114	114	113	117	110	106	110	106
rando39 gsq	393	358	358	794	352	599	688	441	352
rando40 gsq	57	53	53	54	60	54	54	52	52
rando42 gsq	58	59	63	62	63	62	59	67	58
rando43 gsq	120	125	125	123	122	120	132	118	118
rando44 gsq	1028	437	429	621	648	518	724	726	429
rando45 gsq	260	251	251	306	240	221	254	228	221
rando46 gsq	708	616	616	545	544	520	460	517	460
rando47 gsq	268	309	309	269	224	228	268	220	220
rando48 gsq	154	145	145	143	145	136	155	149	136
rando52 gsq	1148	1081	1162	1230	1119	1170	1247	965	965

Table A 50: Maximum required size for small fault trees in the Basic Hybrid method, 1

FT name	1 scheme	2 scheme	3 scheme	4 scheme	5 scheme	6 scheme	7 scheme	8 scheme	Minimum
rando53 gsq	113	99	122	122	119	119	100	120	99
rando54 gsq	158	152	156	156	158	157	156	164	152
rando55 gsq	197	196	196	223	197	197	194	191	191
rando58 gsq	114	117	117	116	102	102	113	102	102
rando59 gsq	842	516	516	467	717	571	430	539	430
rando60 gsq	402	427	452	425	376	337	499	343	337
rando61 gsq	165	172	168	236	163	196	232	161	161
rando62 gsq	115	114	117	120	116	118	114	116	114
rando63 gsq	197	196	196	223	197	197	194	191	191
rando64 gsq	542	405	386	364	584	351	379	359	351
rando65 gsq	75	77	69	74	76	88	107	86	69
rando66 gsq	257	253	253	288	251	236	270	245	236
rando70 gsq	115	107	107	107	108	109	108	112	107
rando73 gsq	334	341	337	338	290	277	499	269	269
rando75 gsq	68	68	68	68	67	68	68	67	67
rando76 gsq	247	180	229	209	202	209	259	221	180
rando77 gsq	524	482	477	539	544	568	702	541	477
rando78 gsq	131	133	144	139	131	131	136	137	131
rando80 gsq	81	76	76	81	82	81	87	81	76
rando83 gsq	113	128	121	89	98	89	127	96	89
rando84 gsq	301	195	211	203	239	210	198	197	195
rando85 gsq	229	221	221	213	206	206	227	213	206
rando87 gsq	70	74	74	70	67	67	69	67	67
rando88 gsq	115	93	114	94	90	95	109	92	90
rando89 gsq	1014	892	860	849	937	872	649	997	649
rando91 gsq	914	1789	1775	1024	847	871	1846	800	800
rando92 gsq	7088	9872	11810	13746	5408	5043	3670	6795	3670
rando93 gsq	411	352	312	410	358	362	341	326	312
rando95 gsq	143	104	133	133	142	128	133	122	104
rando98 gsq	787	644	644	747	682	672	685	482	482
rando99 gsq	517	1104	1047	934	552	676	877	597	517
rand100 gsq	196	199	199	197	181	182	190	181	181
rand103 gsq	151	132	132	145	147	139	138	153	132
rand104 gsq	290	334	338	326	272	284	177	280	177
rand105 gsq	174	174	160	146	168	144	159	203	144
rand106 gsq	698	2071	2071	813	642	572	980	548	548
rand108 gsq	387	598	576	517	367	438	584	337	337
rand109 gsq	569	384	390	502	576	462	590	705	384
rand110 gsq	404	562	505	501	466	474	408	473	404
rand111 gsq	336	382	338	293	324	282	285	296	282
rand115 gsq	337	310	310	348	323	250	250	256	250
rand116 gsq	525	388	437	805	479	748	764	360	360
rand117 gsq	74	73	71	71	69	69	68	70	68
rand118 gsq	301	195	211	203	239	210	198	197	195
rand119 gsq	143	148	153	126	132	130	151	129	126
rand120 gsq	525	244	278	304	505	345	311	492	244
rand121 gsq	236	223	223	217	219	214	305	224	214
rand123 gsq	65	71	71	67	65	61	71	66	61
rand124 gsq	108	119	124	110	115	116	130	118	108
rand125 gsq	57	60	58	54	54	54	58	54	54
rand126 gsq	316	287	287	293	323	266	301	327	266
rand127 gsq	120	90	90	86	112	87	86	109	86
rand128 gsq	673	458	443	654	658	705	634	446	443
rand129 gsq	64	64	64	63	63	63	63	61	61

Table A 51 Maximum required size for small fault trees in the Basic Hybrid method, 2

FT name	1 scheme	2 scheme	3 scheme	4 scheme	5 scheme	6 scheme	7 scheme	8 scheme	Minimum
rand130 gsq	210	181	181	174	179	175	211	179	174
rand132 gsq	767	820	820	1558	611	1203	1532	893	611
rand134 gsq	571	740	760	1512	577	1053	1151	554	554
rand135 gsq	331	336	331	529	342	284	404	359	284
rand137 gsq	98	92	92	92	94	91	92	101	91
rand138 gsq	107	107	107	107	108	108	107	108	107
rand139 gsq	336	243	290	523	336	527	284	377	243
rand141 gsq	404	562	505	501	466	474	408	473	404
rand143 gsq	184	159	170	162	171	166	161	176	159
rand144 gsq	570	1085	987	982	520	643	1104	588	520
rand145 gsq	110	118	118	102	111	106	122	117	102
rand146 gsq	98	92	92	92	94	91	92	101	91
rand147 gsq	635	2696	2963	770	631	1108	1035	667	631
rand148 gsq	103	87	86	93	98	93	97	94	86
rand149 gsq	263	292	279	269	277	246	292	290	246
rand150 gsq	1214	2357	1838	2076	1317	1347	1700	1315	1214
rand151 gsq	108	114	104	104	109	108	115	112	104
rand153 gsq	175	168	168	166	165	165	166	172	165
rand154 gsq	131	88	127	127	99	99	130	87	87
rand155 gsq	338	332	320	236	262	219	470	274	219
rand156 gsq	189	149	149	149	149	143	197	149	143
rand158 gsq	11421	7504	7504	7973	8818	6710	7158	7902	6710
lisaba1 gsq	1171	409	415	355	660	354	379	549	354
lisaba3 gsq	1608	2269	2101	1437	1498	1263	1951	2138	1263
lisaba4 gsq	711	772	793	692	455	491	427	459	427
lisaba5 gsq	622	728	597	535	506	603	826	523	506
lisaba6 gsq	309	321	313	280	267	241	314	310	241
lisaba7 gsq	1171	409	415	355	660	354	379	549	354
lisaba9 gsq	298	195	195	162	226	158	163	233	158
lisab10 gsq	1164	1059	902	702	680	701	1037	583	583
lisab11 gsq	113	99	122	122	119	119	100	120	99
lisab13 gsq	690	824	821	794	779	766	589	783	589
lisab17 gsq	1171	409	415	355	660	354	379	549	354
lisab24 gsq	189	145	145	138	162	152	145	158	138
lisab25 gsq	219	209	232	217	218	208	213	224	208
lisab26 gsq	34	34	34	34	34	34	34	38	34
lisab27 gsq	621	846	782	673	618	440	439	562	439
lisab28 gsq	64	69	62	59	61	66	62	70	59
lisab30 gsq	275	236	234	227	248	209	259	237	209
lisab31 gsq	851	1299	1299	996	809	732	1205	727	727
lisab34 gsq	63	64	64	64	66	65	73	65	63
lisab35 gsq	794	556	556	480	704	725	716	564	480
lisab36 gsq	1055	1274	1138	1309	1120	1254	1056	1015	1015
lisab37 gsq	164	142	142	129	140	129	135	131	129
lisab42 gsq	63	60	60	60	60	60	62	60	60
lisab44 gsq	114	116	116	119	110	122	113	119	110
lisab47 gsq	75	75	75	75	69	73	75	70	69
lisab48 gsq	51	60	60	60	56	56	54	55	51
lisab49 gsq	299	242	244	229	293	236	295	316	229
lisab50 gsq	78	82	69	69	67	66	83	65	65
lisab51 gsq	73	67	67	69	76	73	78	71	67
lisab52 gsq	925	1284	1284	1001	735	800	1161	654	654
lisab53 gsq	23	23	23	23	24	24	23	23	23
lisab54 gsq	71	63	66	66	66	66	67	65	63

Table A 52. Maximum required size for small fault trees in the Basic Hybrid method, 3

FT name	1 scheme	2 scheme	3 scheme	4 scheme	5 scheme	6 scheme	7 scheme	8 scheme	Minimum
lisab55 gsq	10	10	10	10	10	10	10	10	10
lisab56 gsq	72	73	73	74	71	69	75	73	69
lisab57 gsq	211	209	202	194	201	206	300	259	194
lisab59 gsq	231	178	175	185	272	213	174	265	174
lisab60 gsq	71	64	64	63	68	62	63	62	62
lisab61 gsq	281	320	320	304	273	277	370	282	273
lisab62 gsq	189	182	163	229	190	252	304	201	163
lisab63 gsq	70	66	66	64	65	61	64	69	61
lisab64 gsq	739	862	764	763	674	622	654	666	622
lisab65 gsq	54	59	52	52	54	57	59	56	52
lisab66 gsq	485	938	935	562	465	502	958	525	465
lisab67 gsq	658	636	598	492	473	474	826	456	456
lisab68 gsq	74	82	70	70	74	75	70	78	70
lisab69 gsq	136	132	123	114	128	132	140	122	114
lisab70 gsq	327	416	416	239	288	239	305	280	239
lisab71 gsq	57	61	61	61	51	57	61	57	51
lisab72 gsq	779	1111	757	648	666	585	937	577	577
lisab73 gsq	69	69	69	69	74	69	69	69	69
lisab75 gsq	122	125	125	126	121	119	127	121	119
lisab77 gsq	887	821	755	731	842	554	755	762	554
lisab78 gsq	436	302	336	264	303	262	355	282	262
lisab81 gsq	47	47	47	47	47	47	47	47	47
lisab82 gsq	1913	842	842	1120	1226	1279	989	1137	842
lisab83 gsq	567	396	584	448	434	428	263	450	263
lisab84 gsq	567	396	584	448	434	428	263	450	263
lisab86 gsq	339	413	330	312	344	327	447	298	298
lisab88 gsq	500	512	512	551	547	478	439	521	439
lisab89 gsq	1038	646	887	687	1016	783	581	837	581
lisab91 gsq	1470	931	752	750	1225	1042	713	1110	713
lisab95 gsq	77	60	82	82	77	70	82	72	60
lisab97 gsq	112	110	110	104	112	111	101	109	101
lisab98 gsq	57	54	54	56	60	54	56	60	54
lisa100 gsq	2024	964	882	1236	1642	1378	1248	1153	882
lisa103 gsq	23	23	23	23	24	24	23	23	23
lisa104 gsq	167	159	159	157	148	148	164	148	148
lisa107 gsq	28	28	28	28	28	28	28	28	28
lisa109 gsq	258	276	251	274	248	244	277	260	244
lisa110 gsq	1023	1248	1193	1073	1309	1085	1056	1550	1023
lisa111 gsq	287	224	224	224	262	230	257	258	224
lisa112 gsq	1840	530	538	761	1922	1033	723	1391	530
lisa113 gsq	1297	1357	1375	692	888	718	602	709	602
lisa115 gsq	110	113	105	105	92	91	94	89	89
lisa116 gsq	56	61	60	61	54	56	59	57	54
lisa119 gsq	53	42	42	42	43	42	53	43	42
lisa120 gsq	84	81	72	70	75	70	93	75	70
lisa121 gsq	285	217	181	180	244	181	228	235	180
lisa122 gsq	127	122	122	122	122	121	122	120	120
lisa123 gsq	191	179	179	164	204	150	195	263	150
lisa124 gsq	1394	932	926	919	1448	697	1152	1301	697
rand159 gsq	730	881	693	693	574	362	405	494	362
rand161 gsq	355	569	544	523	338	326	565	373	326
rand162 gsq	8	8	8	8	8	8	8	8	8
rand163 gsq	1305	1947	1728	1716	1205	1304	3971	1618	1205
rand164 gsq	2475	2372	1214	1202	1489	1010	1290	1335	1010

Table A 53 Maximum required size for small fault trees in the Basic Hybrid method, 4

FT name	1 scheme	2 scheme	3 scheme	4 scheme	5 scheme	6 scheme	7 scheme	8 scheme	Minimum
astolfo gsq	0 37	0 41	0 45	0 41	0 44	0 45	0 42	0 5	0 37
benjam gsq	0 34	0 42	0 43	0 42	0 43	0 43	0 42	0 43	0 34
bpfig03 dat	0 88	0 99	1 11	1 1	1 07	1 12	1 05	1 02	0 88
bpfen05 dat	0 76	0 88	0 89	0 94	0 67	0 99	0 94	0 91	0 67
bpfig05 dat	0 76	0 89	0 95	0 94	0 94	0 98	0 93	0 88	0 76
bpfin05 dat	0 66	0 8	0 85	0 86	0 87	0 75	0 84	0 81	0 66
bpfsw02 gsq	0 38	0 66	0 66	0 69	0 48	0 42	0 67	0 47	0 38
dre1019 dat	0 34	0 43	0 42	0 43	0 38	0 43	0 43	0 44	0 34
dre1032 dat	0 34	0 42	0 42	0 45	0 42	0 44	0 42	0 42	0 34
dre1057 dat	0 38	0 46	0 48	0 43	0 46	0 52	0 47	0 46	0 38
dre1058 dat	0 22	0 29	0 44	0 42	0 26	0 48	0 44	0 42	0 22
dre1059 dat	0 23	0 27	0 29	0 28	0 28	0 3	0 28	0 24	0 23
dresden dat	0 24	0 34	0 44	0 44	0 33	0 48	0 4	0 41	0 24
hpsif02 dat	0 27	0 48	0 48	0 46	0 3	0 35	0 5	0 39	0 27
hpsif03 dat	0 34	0 42	0 43	0 4	0 38	0 43	0 42	0 42	0 34
hpsif21 gsq	0 22	0 28	0 28	0 25	0 28	0 29	0 28	0 34	0 22
hpsif36 gsq	0 35	0 42	0 42	0 4	0 44	0 42	0 42	0 42	0 35
jdtree3 gsq	0 71	0 79	0 79	0 77	0 81	0 8	0 78	0 75	0 71
jdtree4 gsq	0 46	0 57	0 57	0 56	0 53	0 52	0 56	0 57	0 46
jdtree5 gsq	0 55	0 64	0 66	0 63	0 66	0 63	0 65	0 6	0 55
khictre dat	0 35	0 42	0 43	0 39	0 36	0 41	0 42	0 42	0 35
nakashi gsq	0 22	0 42	0 43	0 25	0 24	0 42	0 42	0 27	0 22
trials1 gsq	0 2	0 27	0 27	0 25	0 27	0 42	0 28	0 42	0 2
trials4 gsq	0 22	0 28	0 28	0 26	0 28	0 28	0 29	0 41	0 22
random3 gsq	0 22	0 28	0 29	0 27	0 45	0 43	0 27	0 27	0 22
random6 gsq	0 27	1 1	1 09	1 27	0 4	0 4	1 4	0 43	0 27
random8 gsq	0 35	0 43	0 43	0 42	0 39	0 43	0 42	0 47	0 35
random12 gsq	0 22	0 27	0 28	0 27	0 22	0 29	0 28	0 28	0 22
random13 gsq	0 22	0 29	0 3	0 3	0 21	0 29	0 32	0 29	0 21
random16 gsq	0 26	0 33	0 46	0 33	0 26	0 33	0 45	0 35	0 26
random18 gsq	3 02	13 89	13 29	30 18	2	10 23	27 07	2 02	2
random23 gsq	0 35	0 42	0 42	0 43	0 41	0 43	0 39	0 38	0 35
random25 gsq	0 35	0 42	0 43	0 43	0 41	0 43	0 43	0 41	0 35
random27 gsq	0 41	0 38	0 37	0 53	0 48	0 5	0 49	0 5	0 37
random28 gsq	0 36	0 43	0 42	0 46	0 41	0 43	0 27	0 42	0 27
random29 gsq	0 21	0 27	0 28	0 26	0 26	0 27	0 24	0 17	0 17
random30 gsq	0 26	0 46	0 48	0 46	0 33	0 32	0 44	0 2	0 2
random31 gsq	0 38	0 46	0 45	0 51	0 46	0 46	0 38	0 41	0 38
random33 gsq	0 35	0 43	0 44	0 42	0 42	0 43	0 41	0 32	0 32
random34 gsq	0 35	0 42	0 42	0 43	0 43	0 42	0 44	0 39	0 35
random35 gsq	0 34	0 43	0 43	0 44	0 41	0 44	0 43	0 39	0 34
random36 gsq	0 35	0 43	0 43	0 42	0 38	0 41	0 43	0 32	0 32
random37 gsq	0 35	0 27	0 28	0 28	0 39	0 28	0 43	0 35	0 27
random38 gsq	0 34	0 42	0 42	0 43	0 48	0 38	0 43	0 32	0 32
random39 gsq	0 22	0 42	0 42	0 25	0 26	0 27	0 29	0 3	0 22
random40 gsq	0 34	0 41	0 43	0 42	0 43	0 41	0 41	0 45	0 34
random42 gsq	0 34	0 42	0 43	0 44	0 42	0 41	0 47	0 43	0 34
random43 gsq	0 34	0 41	0 43	0 43	0 41	0 39	0 45	0 43	0 34
random44 gsq	0 24	0 27	0 28	0 29	0 29	0 28	0 31	0 29	0 24
random45 gsq	0 35	0 42	0 41	0 43	0 44	0 42	0 44	0 44	0 35
random46 gsq	0 35	0 43	0 43	0 44	0 43	0 43	0 46	0 46	0 35
random47 gsq	0 21	0 27	0 28	0 47	0 48	0 36	0 45	0 41	0 21
random48 gsq	0 34	0 36	0 43	0 45	0 43	0 35	0 39	0 42	0 34
random52 gsq	0 37	0 28	0 44	0 46	0 45	0 38	0 43	0 45	0 28

Table A.54: Processing time for small fault trees in the Advanced Hybrid method,

FT name	1 scheme	2 scheme	3 scheme	4 scheme	5 scheme	6 scheme	7 scheme	8 scheme	Minimum
rando53 gsq	0 35	0 43	0 43	0 42	0 42	0 4	0 44	0 43	0 35
rando54 gsq	0 34	0 43	0 44	0 44	0 42	0 38	0 44	0 41	0 34
rando55 gsq	0 35	0 36	0 43	0 43	0 42	0 42	0 42	0 44	0 35
rando58 gsq	0 34	0 36	0 42	0 43	0 45	0 44	0 44	0 41	0 34
rando59 gsq	0 23	0 27	0 28	0 28	0 29	0 27	0 28	0 28	0 23
rando60 gsq	0 35	0 43	0 42	0 43	0 44	0 42	0 44	0 44	0 35
rando61 gsq	0 35	0 38	0 44	0 43	0 43	0 44	0 43	0 44	0 35
rando62 gsq	0 33	0 38	0 42	0 43	0 43	0 42	0 44	0 45	0 33
rando63 gsq	0 34	0 38	0 44	0 42	0 42	0 43	0 43	0 44	0 34
rando64 gsq	0 21	0 4	0 28	0 42	0 27	0 42	0 45	0 28	0 21
rando65 gsq	0 35	0 38	0 42	0 43	0 42	0 43	0 43	0 36	0 35
rando66 gsq	0 34	0 38	0 42	0 44	0 42	0 43	0 43	0 42	0 34
rando70 gsq	0 35	0 37	0 42	0 43	0 4	0 42	0 44	0 46	0 35
rando73 gsq	0 22	0 37	0 42	0 43	0 41	0 4	0 42	0 47	0 22
rando75 gsq	0 35	0 39	0 42	0 43	0 42	0 41	0 42	0 43	0 35
rando76 gsq	0 35	0 38	0 43	0 42	0 42	0 4	0 43	0 44	0 35
rando77 gsq	0 35	0 44	0 44	0 42	0 42	0 44	0 44	0 44	0 35
rando78 gsq	0 35	0 43	0 43	0 43	0 43	0 36	0 42	0 43	0 35
rando80 gsq	0 34	0 42	0 43	0 43	0 44	0 37	0 43	0 46	0 34
rando83 gsq	0 35	0 42	0 43	0 43	0 44	0 41	0 43	0 46	0 35
rando84 gsq	0 22	0 27	0 28	0 28	0 22	0 27	0 44	0 38	0 22
rando85 gsq	0 34	0 42	0 43	0 38	0 37	0 41	0 42	0 35	0 34
rando87 gsq	0 35	0 41	0 43	0 34	0 43	0 41	0 39	0 25	0 25
rando88 gsq	0 39	0 45	0 48	0 39	0 29	0 45	0 45	0 3	0 29
rando89 gsq	0 36	0 43	0 42	0 36	0 25	0 43	0 42	0 32	0 25
rando91 gsq	0 23	0 33	0 31	0 23	0 17	0 28	0 31	0 17	0 17
rando92 gsq	1 16	2 13	2 94	3 72	0 82	0 69	0 47	1 13	0 47
rando93 gsq	0 35	0 44	0 42	0 35	0 42	0 36	0 35	0 43	0 35
rando95 gsq	0 34	0 44	0 43	0 35	0 43	0 36	0 31	0 43	0 31
rando98 gsq	0 23	0 3	0 28	0 22	0 28	0 22	0 26	0 27	0 22
rando99 gsq	0 35	0 29	0 3	0 22	0 28	0 22	0 28	0 29	0 22
rand100 gsq	0 34	0 38	0 43	0 35	0 42	0 38	0 44	0 46	0 34
rand103 gsq	0 35	0 38	0 43	0 35	0 43	0 41	0 44	0 45	0 35
rand104 gsq	0 34	0 43	0 43	0 36	0 42	0 42	0 43	0 42	0 34
rand105 gsq	0 39	0 32	0 32	0 39	0 48	0 48	0 52	0 5	0 32
rand106 gsq	0 36	0 5	0 5	0 38	0 42	0 43	0 47	0 43	0 36
rand108 gsq	0 34	0 3	0 28	0 22	0 42	0 42	0 3	0 41	0 22
rand109 gsq	0 22	0 29	0 29	0 22	0 28	0 28	0 27	0 25	0 22
rand110 gsq	0 34	0 45	0 41	0 34	0 43	0 42	0 44	0 42	0 34
rand111 gsq	0 34	0 43	0 44	0 35	0 43	0 42	0 44	0 37	0 34
rand115 gsq	0 21	0 43	0 42	0 21	0 28	0 27	0 28	0 37	0 21
rand116 gsq	0 34	0 27	0 27	0 24	0 44	0 26	0 3	0 43	0 24
rand117 gsq	0 35	0 43	0 42	0 34	0 43	0 39	0 4	0 42	0 34
rand118 gsq	0 22	0 28	0 28	0 21	0 28	0 26	0 38	0 37	0 21
rand119 gsq	0 33	0 43	0 43	0 35	0 42	0 4	0 44	0 39	0 33
rand120 gsq	0 28	0 52	0 52	0 33	0 41	0 4	0 58	0 33	0 28
rand121 gsq	0 35	0 43	0 43	0 35	0 43	0 44	0 43	0 36	0 35
rand123 gsq	0 36	0 42	0 43	0 35	0 42	0 43	0 45	0 42	0 35
rand124 gsq	0 38	0 47	0 48	0 39	0 47	0 47	0 48	0 45	0 38
rand125 gsq	0 35	0 43	0 41	0 35	0 42	0 39	0 36	0 43	0 35
rand126 gsq	0 25	0 32	0 32	0 25	0 32	0 31	0 44	0 33	0 25
rand127 gsq	0 43	0 51	0 51	0 43	0 54	0 5	0 56	0 52	0 43
rand128 gsq	0 22	0 43	0 44	0 36	0 28	0 26	0 45	0 42	0 22
rand129 gsq	0 35	0 43	0 43	0 35	0 42	0 4	0 45	0 42	0 35

Table A 55 Processing time for small fault trees in the Advanced Hybrid method,

FT name	1 scheme	2 scheme	3 scheme	4 scheme	5 scheme	6 scheme	7 scheme	8 scheme	Minimum
rand130 gsq	0 34	0 43	0 42	0 34	0 43	0 42	0 43	0 43	0 34
rand132 gsq	0 22	0 28	0 28	0 25	0 28	0 29	0 3	0 28	0 22
rand134 gsq	0 22	0 28	0 28	0 24	0 25	0 3	0 3	0 27	0 22
rand135 gsq	0 35	0 43	0 43	0 36	0 42	0 44	0 43	0 44	0 35
rand137 gsq	0 35	0 43	0 41	0 36	0 42	0 42	0 43	0 44	0 35
rand138 gsq	0 34	0 42	0 43	0 34	0 42	0 42	0 44	0 45	0 34
rand139 gsq	0 21	0 44	0 43	0 21	0 43	0 28	0 27	0 39	0 21
rand141 gsq	0 35	0 44	0 43	0 36	0 46	0 39	0 39	0 39	0 35
rand143 gsq	0 34	0 43	0 43	0 35	0 43	0 41	0 4	0 4	0 34
rand144 gsq	0 21	0 3	0 3	0 23	0 27	0 28	0 29	0 25	0 21
rand145 gsq	0 43	0 52	0 52	0 42	0 52	0 53	0 53	0 47	0 42
rand146 gsq	0 35	0 43	0 43	0 35	0 43	0 43	0 43	0 38	0 35
rand147 gsq	0 23	0 4	0 42	0 22	0 27	0 24	0 3	0 26	0 22
rand148 gsq	0 35	0 43	0 42	0 35	0 42	0 37	0 43	0 43	0 35
rand149 gsq	0 42	0 53	0 52	0 46	0 56	0 52	0 53	0 53	0 42
rand150 gsq	0 23	0 35	0 32	0 27	0 31	0 27	0 28	0 29	0 23
rand151 gsq	0 4	0 48	0 48	0 39	0 47	0 34	0 41	0 42	0 34
rand153 gsq	0 36	0 43	0 43	0 35	0 42	0 38	0 36	0 37	0 35
rand154 gsq	0 36	0 43	0 42	0 35	0 44	0 41	0 4	0 39	0 35
rand155 gsq	0 22	0 28	0 28	0 21	0 28	0 43	0 29	0 26	0 21
rand156 gsq	0 36	0 42	0 44	0 35	0 42	0 44	0 48	0 42	0 35
rand158 gsq	2 99	1 56	1 56	1 47	1 87	1 29	1 18	1 59	1 18
lisaba1 gsq	0 33	0 4	0 43	0 43	0 35	0 46	0 44	0 37	0 33
lisaba3 gsq	0 35	0 49	0 48	0 36	0 34	0 42	0 45	0 47	0 34
lisaba4 gsq	0 21	0 29	0 27	0 22	0 27	0 27	0 22	0 28	0 21
lisaba5 gsq	0 21	0 27	0 43	0 35	0 25	0 27	0 21	0 26	0 21
lisaba6 gsq	0 22	0 28	0 28	0 35	0 22	0 28	0 43	0 26	0 22
lisaba7 gsq	0 34	0 4	0 42	0 42	0 32	0 5	0 51	0 33	0 32
lisaba9 gsq	0 22	0 44	0 43	0 35	0 28	0 41	0 39	0 42	0 22
lisab10 gsq	0 23	0 29	0 28	0 22	0 29	0 28	0 3	0 21	0 21
lisab11 gsq	0 36	0 43	0 42	0 35	0 4	0 42	0 45	0 34	0 34
lisab13 gsq	0 36	0 43	0 44	0 36	0 44	0 43	0 44	0 36	0 36
lisab17 gsq	0 32	0 41	0 43	0 41	0 34	0 5	0 52	0 34	0 32
lisab24 gsq	0 58	0 71	0 71	0 75	0 71	0 79	0 76	0 74	0 58
lisab25 gsq	0 35	0 29	0 43	0 35	0 36	0 41	0 44	0 44	0 29
lisab26 gsq	0 35	0 43	0 42	0 34	0 36	0 42	0 44	0 42	0 34
lisab27 gsq	0 23	0 29	0 29	0 23	0 28	0 29	0 44	0 28	0 23
lisab28 gsq	0 42	0 53	0 56	0 46	0 52	0 52	0 59	0 53	0 42
lisab30 gsq	0 35	0 43	0 43	0 36	0 44	0 43	0 44	0 43	0 35
lisab31 gsq	0 23	0 32	0 31	0 23	0 28	0 28	0 31	0 27	0 23
lisab34 gsq	0 34	0 43	0 43	0 35	0 43	0 42	0 45	0 36	0 34
lisab35 gsq	0 22	0 28	0 28	0 21	0 28	0 28	0 29	0 23	0 21
lisab36 gsq	0 23	0 3	0 3	0 24	0 3	0 31	0 47	0 28	0 23
lisab37 gsq	0 35	0 42	0 42	0 35	0 42	0 41	0 44	0 4	0 35
lisab42 gsq	0 34	0 43	0 44	0 36	0 42	0 43	0 43	0 42	0 34
lisab44 gsq	0 35	0 42	0 42	0 35	0 42	0 42	0 44	0 41	0 35
lisab47 gsq	0 35	0 43	0 42	0 34	0 42	0 42	0 44	0 44	0 34
lisab48 gsq	0 34	0 43	0 42	0 35	0 43	0 42	0 41	0 41	0 34
lisab49 gsq	0 59	0 9	0 85	0 82	0 71	0 87	0 98	0 64	0 59
lisab50 gsq	0 34	0 43	0 43	0 35	0 42	0 42	0 44	0 43	0 34
lisab51 gsq	0 39	0 48	0 47	0 39	0 47	0 46	0 51	0 49	0 39
lisab52 gsq	0 23	0 3	0 3	0 22	0 28	0 29	0 31	0 29	0 22
lisab53 gsq	0 37	0 43	0 44	0 35	0 42	0 4	0 39	0 43	0 35
lisab54 gsq	0 36	0 44	0 42	0 34	0 43	0 41	0 38	0 4	0 34

Table A 56 Processing time for small fault trees in the Advanced Hybrid method,

FT name	1 scheme	2 scheme	3 scheme	4 scheme	5 scheme	6 scheme	7 scheme	8 scheme	Minimum
lisab55 gsq	0 41	0 47	0 48	0 37	0 47	0 47	0 43	0 46	0 37
lisab56 gsq	0 35	0 43	0 43	0 34	0 43	0 4	0 42	0 45	0 34
lisab57 gsq	0 22	0 27	0 26	0 22	0 26	0 27	0 28	0 28	0 22
lisab59 gsq	0 52	0 77	0 81	0 75	0 69	0 86	0 96	0 66	0 52
lisab60 gsq	0 36	0 43	0 43	0 34	0 42	0 42	0 45	0 42	0 34
lisab61 gsq	0 36	0 44	0 43	0 36	0 44	0 42	0 46	0 45	0 36
lisab62 gsq	0 41	0 49	0 47	0 39	0 47	0 45	0 48	0 47	0 39
lisab63 gsq	0 35	0 45	0 43	0 34	0 42	0 43	0 44	0 41	0 34
lisab64 gsq	0 37	0 45	0 44	0 35	0 43	0 41	0 44	0 43	0 35
lisab65 gsq	0 36	0 43	0 43	0 35	0 42	0 42	0 46	0 44	0 35
lisab66 gsq	0 36	0 4	0 44	0 36	0 43	0 43	0 46	0 42	0 36
lisab67 gsq	0 27	0 34	0 33	0 27	0 32	0 33	0 36	0 32	0 27
lisab68 gsq	0 48	0 57	0 61	0 52	0 57	0 54	0 66	0 62	0 48
lisab69 gsq	0 36	0 43	0 43	0 35	0 43	0 43	0 35	0 42	0 35
lisab70 gsq	0 28	0 33	0 32	0 4	0 33	0 47	0 34	0 34	0 28
lisab71 gsq	0 36	0 43	0 43	0 36	0 43	0 4	0 42	0 44	0 36
lisab72 gsq	0 24	0 3	0 28	0 21	0 27	0 29	0 28	0 29	0 21
lisab73 gsq	0 52	0 63	0 61	0 47	0 56	0 57	0 66	0 55	0 47
lisab75 gsq	0 36	0 41	0 43	0 36	0 44	0 43	0 42	0 44	0 36
lisab77 gsq	0 24	0 28	0 29	0 22	0 28	0 28	0 28	0 29	0 22
lisab78 gsq	0 23	0 28	0 28	0 34	0 27	0 28	0 28	0 26	0 23
lisab81 gsq	0 44	0 51	0 53	0 42	0 52	0 5	0 5	0 44	0 42
lisab82 gsq	1 3	1 26	1 27	0 88	1 74	0 75	0 5	1 42	0 5
lisab83 gsq	0 24	0 27	0 28	0 36	0 28	0 43	0 43	0 45	0 24
lisab84 gsq	0 22	0 27	0 28	0 35	0 28	0 42	0 42	0 44	0 22
lisab86 gsq	0 24	0 28	0 27	0 22	0 28	0 27	0 28	0 27	0 22
lisab88 gsq	0 37	0 44	0 39	0 35	0 42	0 44	0 27	0 43	0 27
lisab89 gsq	0 24	0 28	0 28	0 22	0 29	0 29	0 42	0 28	0 22
lisab91 gsq	0 3	0 34	0 33	0 25	0 36	0 3	0 34	0 36	0 25
lisab95 gsq	0 36	0 42	0 4	0 35	0 43	0 43	0 42	0 42	0 35
lisab97 gsq	0 42	0 47	0 44	0 42	0 52	0 52	0 47	0 5	0 42
lisab98 gsq	0 39	0 48	0 45	0 39	0 48	0 47	0 47	0 41	0 39
lisa100 gsq	0 36	0 35	0 34	0 28	0 37	0 37	0 35	0 37	0 28
lisa103 gsq	0 37	0 43	0 42	0 35	0 43	0 43	0 42	0 43	0 35
lisa104 gsq	0 37	0 42	0 42	0 36	0 43	0 45	0 42	0 43	0 36
lisa107 gsq	0 38	0 42	0 41	0 36	0 42	0 43	0 42	0 45	0 36
lisa109 gsq	0 38	0 43	0 42	0 37	0 43	0 42	0 42	0 44	0 37
lisa110 gsq	0 25	0 3	0 29	0 24	0 3	0 29	0 29	0 32	0 24
lisa111 gsq	0 33	0 51	0 51	0 45	0 53	0 51	0 58	0 38	0 33
lisa112 gsq	0 34	0 33	0 29	0 27	0 41	0 35	0 35	0 38	0 27
lisa113 gsq	0 26	0 31	0 27	0 23	0 3	0 28	0 29	0 3	0 23
lisa115 gsq	0 39	0 42	0 36	0 37	0 44	0 44	0 44	0 46	0 36
lisa116 gsq	0 38	0 43	0 44	0 37	0 44	0 44	0 42	0 45	0 37
lisa119 gsq	0 42	0 46	0 49	0 4	0 49	0 47	0 48	0 47	0 4
lisa120 gsq	0 58	0 65	0 79	0 61	0 69	0 68	0 67	0 73	0 58
lisa121 gsq	0 31	0 52	0 51	0 44	0 36	0 51	0 53	0 37	0 31
lisa122 gsq	0 37	0 42	0 45	0 36	0 43	0 42	0 43	0 41	0 36
lisa123 gsq	0 38	0 43	0 43	0 37	0 42	0 42	0 43	0 42	0 37
lisa124 gsq	0 27	0 3	0 3	0 25	0 31	0 29	0 31	0 29	0 25
rand159 gsq	0 24	0 28	0 29	0 24	0 28	0 45	0 43	0 37	0 24
rand161 gsq	0 24	0 28	0 26	0 23	0 31	0 27	0 28	0 27	0 23
rand162 gsq	0 38	0 42	0 35	0 36	0 43	0 42	0 44	0 41	0 35
rand163 gsq	0 26	0 33	0 31	0 27	0 31	0 31	0 58	0 31	0 26
rand164 gsq	0 42	0 43	0 34	0 27	0 36	0 34	0 36	0 35	0 27

Table A.57 Processing time for small fault trees in the Advanced Hybrid method,

FT name	1 scheme	2 scheme	3 scheme	4 scheme	5 scheme	6 scheme	7 scheme	8 scheme	Minimum
astolfo gsq	72	75	72	58	74	63	76	83	58
benjam gsq	83	82	82	79	83	74	82	108	74
bpfig03 dat	211	202	194	157	204	157	174	182	157
bpfen05 dat	201	184	193	156	197	156	156	204	156
bpfig05 dat	197	181	190	153	193	153	153	160	153
bpfin05 dat	118	125	97	97	97	97	97	103	97
bpfsw02 gsq	205	139	139	139	205	205	141	205	139
dre1019 dat	45	45	45	45	45	45	49	45	45
dre1032 dat	55	55	55	49	49	49	55	49	49
dre1057 dat	89	97	87	81	83	83	97	81	81
dre1058 dat	269	415	250	223	302	201	294	199	199
dre1059 dat	345	644	677	676	379	276	586	297	276
dresden dat	859	1818	806	739	1215	805	1816	775	739
hpsif02 dat	646	288	288	335	659	641	467	1076	288
hpsif03 dat	113	108	108	108	113	113	111	117	108
hpsif21 gsq	829	683	683	900	976	871	803	2072	683
hpsif36 gsq	134	136	128	122	126	122	136	142	122
jdtree3 gsq	59	51	51	51	59	57	51	59	51
jdtree4 gsq	69	63	63	63	69	69	63	69	63
jdtree5 gsq	61	53	53	53	61	59	53	61	53
khictre dat	199	176	199	196	196	196	176	192	176
nakashi gsq	245	113	143	141	183	130	118	172	113
trials1 gsq	314	306	323	265	323	236	274	279	236
trials4 gsq	466	603	527	474	414	401	486	373	373
random3 gsq	640	735	643	671	612	569	492	633	492
random6 gsq	1979	6846	6845	7629	2788	2685	8040	3080	1979
random8 gsq	77	71	73	73	77	71	71	71	71
rando12 gsq	774	621	631	658	594	591	746	558	558
rando13 gsq	998	1103	1103	1660	889	1213	1735	936	889
rando16 gsq	1918	1814	1577	1839	1893	1722	1509	1899	1509
rando18 gsq	12987	26717	25915	38697	9996	22673	36210	9830	9830
rando23 gsq	332	376	376	369	280	293	375	249	249
rando25 gsq	92	80	80	132	94	113	98	82	80
rando27 gsq	2283	2599	2597	2261	2590	2132	1972	2298	1972
rando28 gsq	258	260	250	249	244	246	253	254	244
rando29 gsq	467	606	468	466	446	390	430	403	390
rando30 gsq	376	188	201	199	352	294	210	332	188
rando31 gsq	1552	1431	1499	2209	1490	1537	1437	1668	1431
rando33 gsq	185	179	179	210	187	172	201	312	172
rando34 gsq	416	396	461	465	394	406	407	336	336
rando35 gsq	382	476	428	425	301	301	283	293	283
rando36 gsq	162	150	157	167	168	161	163	159	150
rando37 gsq	402	566	566	602	397	582	410	447	397
rando38 gsq	117	114	114	113	117	110	106	110	106
rando39 gsq	393	358	358	794	352	599	688	441	352
rando40 gsq	57	53	53	54	60	54	54	52	52
rando42 gsq	58	59	63	62	63	62	59	67	58
rando43 gsq	120	125	125	123	122	120	132	118	118
rando44 gsq	1028	437	429	621	648	518	724	726	429
rando45 gsq	260	251	251	306	240	221	254	228	221
rando46 gsq	708	616	616	545	544	520	460	517	460
rando47 gsq	268	309	309	269	224	228	268	220	220
rando48 gsq	154	145	145	143	145	136	155	149	136
rando52 gsq	1148	1081	1162	1230	1119	1170	1247	965	965

Table A 58 Maximum required size for small fault trees in the Advanced Hybrid method, 1

FT name	1 scheme	2 scheme	3 scheme	4 scheme	5 scheme	6 scheme	7 scheme	8 scheme	Minimum
rando53 gsq	113	99	122	122	119	119	100	120	99
rando54 gsq	158	152	156	156	158	157	156	164	152
rando55 gsq	197	196	196	223	197	197	194	191	191
rando58 gsq	114	117	117	116	102	102	113	102	102
rando59 gsq	842	516	516	467	717	571	430	539	430
rando60 gsq	402	427	452	425	376	337	499	343	337
rando61 gsq	165	172	168	236	163	196	232	161	161
rando62 gsq	115	114	117	120	116	118	114	116	114
rando63 gsq	197	196	196	223	197	197	194	191	191
rando64 gsq	542	405	386	364	584	351	379	359	351
rando65 gsq	75	77	69	74	76	88	107	86	69
rando66 gsq	257	253	253	288	251	236	270	245	236
rando70 gsq	115	107	107	107	108	109	108	112	107
rando73 gsq	334	341	337	338	290	277	499	269	269
rando75 gsq	68	68	68	68	67	68	68	67	67
rando76 gsq	247	180	229	209	202	209	259	221	180
rando77 gsq	524	482	477	539	544	568	702	541	477
rando78 gsq	131	133	144	139	131	131	136	137	131
rando80 gsq	81	76	76	81	82	81	87	81	76
rando83 gsq	113	128	121	89	98	89	127	96	89
rando84 gsq	301	195	211	203	239	210	198	197	195
rando85 gsq	229	221	221	213	206	206	227	213	206
rando87 gsq	70	74	74	70	67	67	69	67	67
rando88 gsq	113	91	112	92	88	93	107	90	88
rando89 gsq	1014	892	860	849	937	872	649	997	649
rando91 gsq	914	1789	1775	1024	847	871	1846	800	800
rando92 gsq	7088	9872	11810	13746	5408	5043	3670	6795	3670
rando93 gsq	411	352	312	410	358	362	341	326	312
rando95 gsq	143	104	133	133	142	128	133	122	104
rando98 gsq	787	644	644	747	682	672	685	482	482
rando99 gsq	517	1104	1047	934	552	676	877	597	517
rand100 gsq	196	199	199	197	181	182	190	181	181
rand103 gsq	151	132	132	145	147	139	138	153	132
rand104 gsq	290	334	338	326	272	284	177	280	177
rand105 gsq	200	196	192	168	204	164	153	229	153
rand106 gsq	698	2071	2071	813	642	572	980	548	548
rand108 gsq	387	598	576	517	367	438	584	337	337
rand109 gsq	569	384	390	502	576	462	590	705	384
rand110 gsq	404	562	505	501	466	474	408	473	404
rand111 gsq	336	382	338	293	324	282	285	296	282
rand115 gsq	337	310	310	348	323	250	250	256	250
rand116 gsq	525	388	437	805	479	748	764	360	360
rand117 gsq	74	73	71	71	69	69	68	70	68
rand118 gsq	301	195	211	203	239	210	198	197	195
rand119 gsq	143	148	153	126	132	130	151	129	126
rand120 gsq	523	242	276	299	500	340	306	487	242
rand121 gsq	236	223	223	217	219	214	305	224	214
rand123 gsq	65	71	71	67	65	61	71	66	61
rand124 gsq	106	117	122	108	113	114	128	116	106
rand125 gsq	57	60	58	54	54	54	58	54	54
rand126 gsq	315	286	286	292	322	265	300	326	265
rand127 gsq	117	87	87	83	109	84	83	106	83
rand128 gsq	673	458	443	654	658	705	634	446	443
rand129 gsq	64	64	64	63	63	63	63	61	61

Table A 59: Maximum required size for small fault trees in the Advanced Hybrid method, 2

FT name	1 scheme	2 scheme	3 scheme	4 scheme	5 scheme	6 scheme	7 scheme	8 scheme	Minimum
rand130 gsq	210	181	181	174	179	175	211	179	174
rand132 gsq	767	820	820	1558	611	1203	1532	893	611
rand134 gsq	571	740	760	1512	577	1053	1151	554	554
rand135 gsq	331	336	331	529	342	284	404	359	284
rand137 gsq	98	92	92	92	94	91	92	101	91
rand138 gsq	107	107	107	107	108	108	107	108	107
rand139 gsq	336	243	290	523	336	527	284	377	243
rand141 gsq	404	562	505	501	466	474	408	473	404
rand143 gsq	184	159	170	162	171	166	161	176	159
rand144 gsq	570	1085	987	982	520	643	1104	588	520
rand145 gsq	107	115	115	99	108	103	119	114	99
rand146 gsq	98	92	92	92	94	91	92	101	91
rand147 gsq	635	2696	2963	770	631	1108	1035	667	631
rand148 gsq	103	87	86	93	98	93	97	94	86
rand149 gsq	261	290	277	264	272	241	290	285	241
rand150 gsq	1214	2357	1838	2076	1317	1347	1700	1315	1214
rand151 gsq	107	113	103	103	108	107	114	111	103
rand153 gsq	175	168	168	166	165	165	166	172	165
rand154 gsq	131	88	127	127	99	99	130	87	87
rand155 gsq	338	332	320	236	262	219	470	274	219
rand156 gsq	189	149	149	149	149	143	197	149	143
rand158 gsq	11421	7504	7504	7973	8818	6710	7158	7902	6710
lisaba1 gsq	1167	405	411	354	659	353	378	548	353
lisaba3 gsq	1605	2266	2098	1434	1495	1260	1948	2135	1260
lisaba4 gsq	711	772	793	692	455	491	427	459	427
lisaba5 gsq	622	728	597	535	506	603	826	523	506
lisaba6 gsq	309	321	313	280	267	241	314	310	241
lisaba7 gsq	1167	405	411	354	659	353	378	548	353
lisaba9 gsq	298	195	195	162	226	158	163	233	158
lisab10 gsq	1164	1059	902	702	680	701	1037	583	583
lisab11 gsq	113	99	122	122	119	119	100	120	99
lisab13 gsq	690	824	821	794	779	766	589	783	589
lisab17 gsq	1167	405	411	354	659	353	378	548	353
lisab24 gsq	175	131	131	88	140	130	122	136	88
lisab25 gsq	219	209	232	217	218	208	213	224	208
lisab26 gsq	34	34	34	34	34	34	34	38	34
lisab27 gsq	621	846	782	673	618	440	439	562	439
lisab28 gsq	66	79	56	46	56	61	56	65	46
lisab30 gsq	275	236	234	227	248	209	259	237	209
lisab31 gsq	851	1299	1299	996	809	732	1205	727	727
lisab34 gsq	63	64	64	64	66	65	73	65	63
lisab35 gsq	794	556	556	480	704	725	716	564	480
lisab36 gsq	1055	1274	1138	1309	1120	1254	1056	1015	1015
lisab37 gsq	164	142	142	129	140	129	135	131	129
lisab42 gsq	63	60	60	60	60	60	62	60	60
lisab44 gsq	114	116	116	119	110	122	113	119	110
lisab47 gsq	75	75	75	75	69	73	75	70	69
lisab48 gsq	51	60	60	60	56	56	54	55	51
lisab49 gsq	268	185	213	164	267	202	172	282	164
lisab50 gsq	78	82	69	69	67	66	83	65	65
lisab51 gsq	70	64	64	66	73	70	75	68	64
lisab52 gsq	925	1284	1284	1001	735	800	1161	654	654
lisab53 gsq	23	23	23	23	24	24	23	23	23
lisab54 gsq	71	63	66	66	66	66	67	65	63

Table A 60: Maximum required size for small fault trees in the Advanced Hybrid method, 3

FT name	1 scheme	2 scheme	3 scheme	4 scheme	5 scheme	6 scheme	7 scheme	8 scheme	Minimum
lisab55 gsq	9	9	9	9	9	9	9	9	9
lisab56 gsq	72	73	73	74	71	69	75	73	69
lisab57 gsq	211	209	202	194	201	206	300	259	194
lisab59 gsq	215	162	156	127	244	185	129	237	127
lisab60 gsq	71	64	64	63	68	62	63	62	62
lisab61 gsq	281	320	320	304	273	277	370	282	273
lisab62 gsq	188	181	162	228	189	251	303	200	162
lisab63 gsq	70	66	66	64	65	61	64	69	61
lisab64 gsq	739	862	764	763	674	622	654	666	622
lisab65 gsq	54	59	52	52	54	57	59	56	52
lisab66 gsq	485	938	935	562	465	502	958	525	465
lisab67 gsq	657	635	597	491	472	473	825	455	455
lisab68 gsq	68	76	57	57	68	69	57	68	57
lisab69 gsq	136	132	123	114	128	132	140	122	114
lisab70 gsq	324	413	413	236	285	236	302	277	236
lisab71 gsq	57	61	61	61	51	57	61	57	51
lisab72 gsq	779	1111	757	648	666	585	937	577	577
lisab73 gsq	57	57	57	62	67	62	45	50	45
lisab75 gsq	122	125	125	126	121	119	127	121	119
lisab77 gsq	887	821	755	731	842	554	755	762	554
lisab78 gsq	436	302	336	264	303	262	355	282	262
lisab81 gsq	41	41	41	41	41	41	41	41	41
lisab82 gsq	1911	840	840	1118	1224	1277	987	1135	840
lisab83 gsq	567	396	584	448	434	428	263	450	263
lisab84 gsq	567	396	584	448	434	428	263	450	263
lisab86 gsq	339	413	330	312	344	327	447	298	298
lisab88 gsq	500	512	512	551	547	478	439	521	439
lisab89 gsq	1038	646	887	687	1016	783	581	837	581
lisab91 gsq	1469	930	751	749	1224	1041	712	1109	712
lisab95 gsq	77	60	82	82	77	70	82	72	60
lisab97 gsq	110	108	108	98	106	105	99	103	98
lisab98 gsq	55	52	52	54	58	52	54	58	52
lisa100 gsq	2022	962	880	1234	1640	1376	1246	1151	880
lisa103 gsq	23	23	23	23	24	24	23	23	23
lisa104 gsq	167	159	159	157	148	148	164	148	148
lisa107 gsq	28	28	28	28	28	28	28	28	28
lisa109 gsq	258	276	251	274	248	244	277	260	244
lisa110 gsq	1023	1248	1193	1073	1309	1085	1056	1550	1023
lisa111 gsq	285	222	222	222	260	228	249	256	222
lisa112 gsq	1839	529	537	760	1921	1032	722	1390	529
lisa113 gsq	1297	1357	1375	692	888	718	602	709	602
lisa115 gsq	110	113	105	105	92	91	94	89	89
lisa116 gsq	56	61	60	61	54	56	59	57	54
lisa119 gsq	49	38	38	38	39	38	49	39	38
lisa120 gsq	72	69	57	57	62	57	66	62	57
lisa121 gsq	296	248	192	196	251	197	221	242	192
lisa122 gsq	127	122	122	122	122	121	122	120	120
lisa123 gsq	191	179	179	164	204	150	195	263	150
lisa124 gsq	1394	932	926	919	1448	697	1152	1301	697
rand159 gsq	730	881	693	693	574	362	405	494	362
rand161 gsq	355	569	544	523	338	326	565	373	326
rand162 gsq	8	8	8	8	8	8	8	8	8
rand163 gsq	1305	1947	1728	1716	1205	1304	3971	1618	1205
rand164 gsq	2474	2371	1213	1201	1488	1009	1289	1334	1009

Table A.61: Maximum required size for small fault trees in the Advanced Hybrid method, 4

FT name	Number of gates	Number of basic events	Number of repeated events	Number of complex events	Number of modules	Number of prime implicants
random61 gsq	45	59	35	9	1	107
rando121 gsq	32	70	21	30	1	2128
rando591 gsq	23	44	11	13	1	119
rand1321 gsq	31	46	25	3	1	339
rand1501 gsq	29	45	23	10	1	114
rand1581 gsq	49	78	33	22	3	853
hsab521 gsq	31	45	31	4	1	227
hsab741 gsq	46	126	9	75	1	78158
hsab891 gsq	32	63	11	23	1	1044
hsa1001 gsq	31	66	11	28	1	1254
rand1641 gsq	32	63	12	18	1	13092
rand1651 gsq	40	99	10	61	2	2072
rand1661 gsq	31	60	16	10	1	1276

Table A.62: Complexity of test non-coherent fault trees, 'large' trees, overall analysis

FT name	Number of gates	Number of basic events	Number of repeated events	Number of complex events	Number of modules	Number of prime implicants
astoff01 gsq	19	17	2	8	1	34
benjam1 gsq	15	14	4	0	1	47
bpfig031 gsq	20	63	0	62	1	8716
bpfen051 gsq	17	61	0	60	1	7471
bpfig051 gsq	17	60	0	59	1	7056
bpfin051 gsq	14	40	0	39	1	416
bpfs021 gsq	21	42	2	30	2	104382
dre10191 gsq	4	19	1	18	1	63
dre10321 gsq	4	21	1	20	1	75
dre10571 gsq	7	32	1	31	1	2100
dre10581 gsq	13	46	16	22	5	41310
dre10591 gsq	17	58	20	39	1	42318
dresden1 gsq	17	63	17	34	1	53217
hpsf021 gsq	19	74	6	54	1	339
hpsf031 gsq	7	32	1	24	1	82
hpsf211 gsq	15	69	49	38	1	1357
hpsf361 gsq	8	30	4	23	1	61
jdtree31 gsq	11	21	0	20	1	36
jdtree41 gsq	11	20	1	18	1	30
jdtree51 gsq	11	20	1	19	1	10
khctre1 gsq	19	26	14	11	3	21
nakashi1 gsq	21	16	11	1	1	20
trials11 gsq	27	20	9	2	1	38
trials41 gsq	39	27	15	1	1	105
random31 gsq	24	52	9	21	1	435
random81 gsq	7	19	2	4	3	5
rand0131 gsq	46	63	43	12	1	73
rand0161 gsq	31	52	21	12	3	8
rand0231 gsq	19	43	13	19	1	13
rand0251 gsq	14	17	11	11	1	6
rand0271 gsq	45	57	32	12	2	120
rand0281 gsq	17	38	12	4	1	1
rand0291 gsq	25	43	17	9	1	43
rand0301 gsq	17	42	3	20	1	193
rand0311 gsq	47	45	35	3	1	2
rand0331 gsq	17	37	16	7	2	51
rand0341 gsq	24	40	15	10	1	5
rand0351 gsq	19	26	16	6	5	8
rand0361 gsq	15	29	7	19	1	10
rand0371 gsq	27	34	23	6	1	30
rand0381 gsq	11	21	5	15	1	9
rand0391 gsq	27	31	21	1	1	44
rand0401 gsq	8	18	4	4	2	11
rand0421 gsq	9	18	6	3	2	2
rand0431 gsq	10	29	2	16	1	28
rand0441 gsq	27	61	7	28	1	373
rand0451 gsq	22	30	19	5	2	16
rand0461 gsq	22	44	16	17	3	10
rand0471 gsq	20	48	14	10	5	20
rand0481 gsq	16	25	12	9	2	5
rand0521 gsq	33	38	21	10	3	46
rand0531 gsq	13	24	9	5	2	2
rand0541 gsq	13	35	4	18	3	317
rand0551 gsq	15	26	11	8	3	9

Table A 63: Complexity of test non-coherent fault trees, small trees, 1

FT name	Number of gates	Number of basic events	Number of repeated events	Number of complex events	Number of modules	Number of prime implicants
rando581 gsq	10	19	8	7	2	3
rando601 gsq	36	72	14	32	4	22
rando611 gsq	19	27	13	2	1	26
rando621 gsq	13	21	10	4	4	7
rando631 gsq	15	24	14	9	3	9
rando641 gsq	19	35	10	17	3	31
rando651 gsq	11	18	4	4	1	10
rando661 gsq	17	33	14	6	2	3
rando701 gsq	12	25	3	10	1	41
rando731 gsq	22	42	18	13	3	144
rando751 gsq	12	19	6	12	1	4
rando761 gsq	15	34	8	18	3	24
rando771 gsq	31	43	20	8	1	25
rando781 gsq	17	30	7	15	1	2
rando801 gsq	9	26	3	19	2	22
rando831 gsq	14	23	7	8	1	39
rando841 gsq	19	41	6	19	2	198
rando851 gsq	13	28	9	17	1	7
rando871 gsq	11	22	5	17	1	15
rando881 gsq	11	23	2	12	1	35
rando891 gsq	24	45	11	18	6	27
rando911 gsq	32	66	23	11	1	67
rando921 gsq	41	74	43	12	2	35
rando931 gsq	19	42	11	19	4	13
rando951 gsq	11	23	7	7	3	31
rando981 gsq	22	55	10	25	1	339
rando991 gsq	26	48	22	8	1	47
rand1001 gsq	19	31	13	7	3	8
rand1031 gsq	13	25	4	9	2	32
rand1041 gsq	16	24	13	7	4	9
rand1051 gsq	15	33	4	21	1	96
rand1061 gsq	31	43	19	6	6	8
rand1081 gsq	32	38	23	2	1	50
rand1091 gsq	27	57	10	25	1	203
rand1101 gsq	24	34	17	8	1	8
rand1111 gsq	19	24	14	6	1	22
rand1151 gsq	21	29	13	13	1	46
rand1161 gsq	24	41	22	10	2	9
rand1171 gsq	10	17	5	9	1	11
rand1181 gsq	19	39	8	19	2	52
rand1191 gsq	14	31	5	24	2	24
rand1201 gsq	20	41	5	17	1	74
rand1211 gsq	18	38	12	15	1	80
rand1231 gsq	9	17	4	11	1	12
rand1241 gsq	12	24	5	20	1	27
rand1251 gsq	6	17	2	7	1	7
rand1261 gsq	25	41	10	14	1	74
rand1271 gsq	12	28	3	27	1	43
rand1281 gsq	24	38	21	12	1	52
rand1291 gsq	8	21	5	8	1	1
rand1301 gsq	13	27	10	4	1	5
rand1341 gsq	34	64	23	14	1	152
rand1351 gsq	24	39	19	2	2	22
rand1371 gsq	10	21	5	8	1	15

Table A 64: Complexity of test non-coherent fault trees, small trees, 2

FT name	Number of gates	Number of basic events	Number of repeated events	Number of complex events	Number of modules	Number of prime implicants
rand1381 gsq	10	21	9	7	4	11
rand1391 gsq	21	32	15	4	1	69
rand1411 gsq	24	34	17	10	1	8
rand1421 gsq	32	47	29	6	2	155
rand1431 gsq	17	30	8	12	5	8
rand1441 gsq	29	50	26	16	1	83
rand1451 gsq	11	33	1	26	2	47
rand1461 gsq	10	22	4	8	1	20
rand1471 gsq	36	49	28	5	2	24
rand1481 gsq	12	28	2	18	2	8
rand1491 gsq	22	59	5	40	1	18
rand1511 gsq	10	28	4	25	1	36
rand1531 gsq	16	26	12	6	2	3
rand1541 gsq	11	22	7	4	1	1
rand1551 gsq	20	35	8	4	1	64
rand1561 gsq	10	22	6	21	1	20
lisaba11 gsq	27	70	6	43	4	1141
lisaba31 gsq	40	86	16	38	1	5447
lisaba41 gsq	26	50	9	11	1	1001
lisaba51 gsq	29	61	17	26	3	231
lisaba61 gsq	22	58	4	33	1	1230
lisaba71 gsq	27	69	7	41	4	1054
lisaba91 gsq	17	44	2	27	2	187
lisab101 gsq	27	55	19	17	1	220
lisab111 gsq	13	22	10	7	1	2
lisab131 gsq	24	32	17	10	1	8
lisab171 gsq	27	72	4	42	4	1201
lisab191 gsq	37	112	0	111	1	15420
lisab201 gsq	41	128	0	127	1	7899
lisab241 gsq	15	41	0	40	1	887
lisab251 gsq	15	30	6	7	1	17
lisab271 gsq	26	63	13	34	5	301
lisab281 gsq	9	22	0	21	1	66
lisab301 gsq	19	35	9	9	4	16
lisab311 gsq	31	53	32	5	1	611
lisab341 gsq	8	16	6	3	2	19
lisab351 gsq	19	42	11	14	1	152
lisab361 gsq	46	44	35	4	1	66
lisab371 gsq	10	30	3	24	3	64
lisab421 gsq	7	21	2	19	1	10
lisab441 gsq	10	20	11	8	3	12
lisab461 gsq	50	152	0	151	1	14669
lisab471 gsq	10	12	8	5	1	3
lisab481 gsq	8	19	9	4	1	5
lisab491 gsq	22	60	0	59	1	890
lisab501 gsq	10	18	4	3	1	1
lisab511 gsq	8	21	0	10	1	16
lisab531 gsq	5	9	1	8	1	15
lisab541 gsq	6	16	3	3	1	7
lisab561 gsq	11	18	9	5	1	3
lisab571 gsq	18	33	11	6	1	162
lisab591 gsq	16	49	0	48	1	3096
lisab601 gsq	7	16	5	5	1	19
lisab611 gsq	22	44	11	15	1	15

Table A 65. Complexity of test non-coherent fault trees, small trees, 3

FT name	Number of gates	Number of basic events	Number of repeated events	Number of complex events	Number of modules	Number of prime implicants
lisab621 gsq	17	40	3	27	3	74
lisab631 gsq	8	18	5	12	1	6
lisab641 gsq	21	46	18	14	5	26
lisab651 gsq	7	19	2	18	1	5
lisab661 gsq	31	43	26	6	1	77
lisab671 gsq	38	81	9	51	1	1718
lisab681 gsq	9	25	0	24	1	180
lisab691 gsq	14	31	2	19	1	59
lisab701 gsq	19	49	4	28	1	40
lisab711 gsq	8	15	7	6	2	3
lisab721 gsq	34	57	21	13	1	71
lisab731 gsq	7	23	0	22	1	40
lisab751 gsq	14	29	10	6	1	1
lisab771 gsq	29	63	9	31	1	7433
lisab781 gsq	16	39	9	15	1	503
lisab811 gsq	6	15	0	14	1	11
lisab821 gsq	31	88	6	56	1	41388
lisab831 gsq	19	36	10	14	1	75
lisab841 gsq	19	39	8	10	1	71
lisab861 gsq	21	42	8	12	1	289
lisab881 gsq	25	53	13	22	1	44
lisab911 gsq	32	63	13	30	1	5798
lisab951 gsq	7	20	4	5	1	1
lisab971 gsq	11	28	1	20	1	4
lisab981 gsq	7	16	2	13	1	7
lisa1031 gsq	5	9	1	8	1	15
lisa1041 gsq	10	20	7	14	1	4
lisa1071 gsq	6	12	3	3	3	7
lisa1091 gsq	21	24	16	0	1	21
lisa1101 gsq	36	55	22	14	1	15
lisa1111 gsq	17	53	2	34	2	63
lisa1121 gsq	32	83	7	47	1	4979
lisa1131 gsq	25	63	11	29	2	85
lisa1151 gsq	12	26	8	8	1	59
lisa1161 gsq	10	16	5	5	1	8
lisa1191 gsq	7	15	1	9	1	20
lisa1201 gsq	10	24	0	23	1	126
lisa1211 gsq	21	41	5	31	3	72
lisa1221 gsq	12	27	8	9	1	8
lisa1231 gsq	15	29	8	6	1	41
lisa1241 gsq	28	69	12	21	1	1186
rand1591 gsq	25	39	18	9	1	15
rand1611 gsq	22	43	17	7	1	153
rand1631 gsq	37	61	28	15	2	704
rand1671 gsq	19	39	5	14	2	301

Table A 66 Complexity of test non-coherent fault trees, small trees, 4

FT name	1 scheme	2 scheme	3 scheme	4 scheme	5 scheme	6 scheme	7 scheme	8 scheme	Minimum
astolfo1 gsq	38	40	40	39	37	39	39	35	35
benjam1 gsq	71	87	87	65	71	71	65	97	65
bpfig031 gsq	1	1	1	1	1	1	1	1	1
bpfen051 gsq	1	1	1	1	1	1	1	1	1
bpfig051 gsq	1	1	1	1	1	1	1	1	1
bpfn051 gsq	1	1	1	1	1	1	1	1	1
bpfs021 gsq	55	33	46	41	55	33	53	43	33
dre10191 gsq	1	1	1	1	1	1	1	1	1
dre10321 gsq	1	1	1	1	1	1	1	1	1
dre10571 gsq	1	1	1	1	1	1	1	1	1
dre10581 gsq	140	128	128	127	126	129	123	109	109
dre10591 gsq	184	233	235	254	138	121	225	111	111
dresden1 gsq	363	154	154	147	333	187	285	129	129
hpsif021 gsq	281	102	102	82	95	69	119	128	69
hpsif031 gsq	16	16	16	17	16	17	17	16	16
hpsif211 gsq	388	1076	645	415	476	556	1089	388	388
hpsif361 gsq	16	16	16	16	16	16	16	16	16
dtree31 gsq	1	1	1	1	1	1	1	1	1
dtree41 gsq	1	1	1	1	1	1	1	1	1
dtree51 gsq	1	1	1	1	1	1	1	1	1
khictre1 gsq	30	40	35	35	30	35	36	40	30
nakash1 gsq	183	68	105	135	168	95	68	160	68
trials11 gsq	254	209	209	209	272	142	255	134	134
trials41 gsq	706	1412	915	1274	740	1372	1619	617	617
random31 gsq	119	122	110	105	118	126	107	138	105
random81 gsq	19	19	19	19	19	19	19	19	19
rando131 gsq	100	82	80	80	91	96	118	96	80
rando161 gsq	22	22	22	22	22	22	25	22	22
rando231 gsq	63	58	58	58	61	55	58	65	55
rando251 gsq	1	1	1	1	1	1	1	1	1
rando271 gsq	41	41	41	40	38	35	40	36	35
rando281 gsq	1	1	1	1	1	1	1	1	1
rando291 gsq	737	599	584	609	526	405	493	352	352
rando301 gsq	137	102	120	120	150	77	69	131	69
rando311 gsq	6	6	6	6	6	6	6	6	6
rando331 gsq	114	127	127	114	108	110	56	89	56
rando341 gsq	18	15	15	15	12	12	15	13	12
rando351 gsq	46	46	46	46	41	46	34	38	34
rando361 gsq	1	1	1	1	1	1	1	1	1
rando371 gsq	82	76	76	76	80	79	45	79	45
rando381 gsq	1	1	1	1	1	1	1	1	1
rando391 gsq	139	65	65	105	107	105	161	143	65
rando401 gsq	42	32	38	38	42	42	45	40	32
rando421 gsq	7	7	7	7	7	7	7	7	7
rando431 gsq	22	21	21	26	21	21	26	21	21
rando441 gsq	288	259	249	291	329	285	260	268	249
rando451 gsq	67	70	70	60	53	70	70	53	53
rando461 gsq	12	12	12	12	12	12	12	12	12
rando471 gsq	115	96	131	170	147	98	113	137	96
rando481 gsq	10	10	10	10	10	10	10	10	10
rando521 gsq	157	77	147	147	143	141	81	143	77
rando531 gsq	6	6	6	6	6	6	6	6	6
rando541 gsq	73	40	71	71	73	69	71	76	40
rando551 gsq	28	28	28	28	28	28	28	27	27

Table A 67 TDD method, number of nodes, small trees, 1

FT name	1 scheme	2 scheme	3 scheme	4 scheme	5 scheme	6 scheme	7 scheme	8 scheme	Minimum
rando581 gsq	6	6	6	6	6	6	6	6	6
rando601 gsq	34	31	34	31	30	30	34	30	30
rando611 gsq	102	303	147	160	102	87	303	100	87
rando621 gsq	18	18	18	18	18	18	18	18	18
rando631 gsq	23	23	23	23	23	23	23	24	23
rando641 gsq	90	81	81	78	86	78	54	89	54
rando651 gsq	28	42	27	30	26	30	27	25	25
rando661 gsq	18	18	18	18	18	18	18	18	18
rando701 gsq	37	35	31	35	44	35	43	61	31
rando731 gsq	64	64	64	64	64	64	54	64	54
rando751 gsq	1	1	1	1	1	1	1	1	1
rando761 gsq	27	27	27	27	27	27	27	27	27
rando771 gsq	56	15	15	15	40	15	20	18	15
rando781 gsq	1	1	1	1	1	1	1	1	1
rando801 gsq	13	13	13	13	13	13	16	16	13
rando831 gsq	43	31	37	37	43	32	35	39	31
rando841 gsq	350	137	159	172	263	164	126	141	126
rando851 gsq	1	1	1	1	1	1	1	1	1
rando871 gsq	1	1	1	1	1	1	1	1	1
rando881 gsq	41	26	34	34	36	34	27	36	26
rando891 gsq	69	62	68	68	68	68	59	69	59
rando911 gsq	1067	941	941	700	738	617	1068	772	617
rando921 gsq	396	475	460	459	345	334	317	303	303
rando931 gsq	30	33	33	33	33	33	33	33	30
rando951 gsq	68	54	66	66	56	56	66	54	54
rando981 gsq	313	253	253	270	240	252	185	201	185
rando991 gsq	340	350	290	272	276	206	272	213	206
rand1001 gsq	10	10	10	10	10	10	10	10	10
rand1031 gsq	78	50	74	74	74	74	50	70	50
rand1041 gsq	17	17	17	17	17	17	17	17	17
rand1051 gsq	29	36	32	32	29	32	29	30	29
rand1061 gsq	22	22	22	22	22	22	22	22	22
rand1081 gsq	166	195	165	197	153	159	152	171	152
rand1091 gsq	217	132	142	191	213	224	159	202	132
rand1101 gsq	15	15	15	11	11	11	11	11	11
rand1111 gsq	65	56	56	53	61	56	50	50	50
rand1151 gsq	129	77	77	117	97	87	87	73	73
rand1161 gsq	36	30	29	29	32	30	29	32	29
rand1171 gsq	10	15	15	15	13	13	15	13	10
rand1181 gsq	129	98	87	92	110	92	68	111	68
rand1191 gsq	10	10	10	10	10	10	10	10	10
rand1201 gsq	196	103	90	156	200	173	152	200	90
rand1211 gsq	35	36	36	34	34	32	34	37	32
rand1231 gsq	6	6	6	6	6	6	6	8	6
rand1241 gsq	1	1	1	1	1	1	1	1	1
rand1251 gsq	9	11	9	9	9	9	11	9	9
rand1261 gsq	170	181	181	170	164	161	250	165	161
rand1271 gsq	1	1	1	1	1	1	1	1	1
rand1281 gsq	70	66	58	76	69	73	61	63	58
rand1291 gsq	1	1	1	1	1	1	1	1	1
rand1301 gsq	3	3	3	3	3	3	3	3	3
rand1341 gsq	122	83	83	146	122	118	141	125	83
rand1351 gsq	121	142	115	111	118	111	191	111	111
rand1371 gsq	33	33	33	33	34	34	31	33	31

Table A 68: TDD method, number of nodes, small trees, 2

FT name	1 scheme	2 scheme	3 scheme	4 scheme	5 scheme	6 scheme	7 scheme	8 scheme	Minimum
rand1381 gsq	24	24	24	24	24	24	24	24	24
rand1391 gsq	222	183	220	272	223	204	209	232	183
rand1411 gsq	6	6	6	6	6	6	9	9	6
rand1421 gsq	324	350	365	518	335	317	240	376	240
rand1431 gsq	19	19	19	19	19	19	19	19	19
rand1441 gsq	106	82	83	81	142	115	132	113	81
rand1451 gsq	15	15	15	15	15	15	15	16	15
rand1461 gsq	33	33	33	33	36	35	35	35	33
rand1471 gsq	413	464	465	507	350	348	416	459	348
rand1481 gsq	6	6	6	6	6	6	6	6	6
rand1491 gsq	1	1	1	1	1	1	1	1	1
rand1511 gsq	1	1	1	1	1	1	1	1	1
rand1531 gsq	7	7	7	7	7	7	7	7	7
rand1541 gsq	1	1	1	1	1	1	1	1	1
rand1551 gsq	275	212	146	226	229	181	213	211	146
rand1561 gsq	1	1	1	1	1	1	1	1	1
lsaba11 gsq	84	84	91	75	75	68	91	67	67
lsaba31 gsq	560	662	628	567	567	411	999	609	411
lsaba41 gsq	581	494	461	378	377	659	612	347	347
lsaba51 gsq	149	190	170	189	168	174	186	162	149
lsaba61 gsq	127	274	274	122	100	116	90	95	90
lsaba71 gsq	80	91	86	81	79	86	86	90	79
lsaba91 gsq	35	38	38	28	30	28	41	39	28
lsab101 gsq	267	155	129	129	153	148	114	140	114
lsab111 gsq	1	1	1	1	1	1	1	1	1
lsab131 gsq	6	6	6	6	6	6	9	9	6
lsab171 gsq	105	117	124	117	105	106	124	93	93
lsab191 gsq	1	1	1	1	1	1	1	1	1
lsab201 gsq	1	1	1	1	1	1	1	1	1
lsab241 gsq	1	1	1	1	1	1	1	1	1
lsab251 gsq	52	53	60	44	52	57	53	49	44
lsab271 gsq	296	277	277	127	245	144	142	247	127
lsab281 gsq	1	1	1	1	1	1	1	1	1
lsab301 gsq	58	50	50	50	51	50	48	45	45
lsab311 gsq	1972	1240	1319	851	781	699	1137	693	693
lsab341 gsq	42	35	45	37	42	37	45	47	35
lsab351 gsq	452	659	642	399	364	370	416	302	302
lsab361 gsq	157	167	149	96	137	99	146	182	96
lsab371 gsq	14	14	14	14	14	14	14	17	14
lsab421 gsq	1	1	1	1	1	1	1	1	1
lsab441 gsq	32	30	30	30	32	30	30	32	30
lsab461 gsq	1	1	1	1	1	1	1	1	1
lsab471 gsq	7	6	6	6	7	7	6	7	6
lsab481 gsq	6	6	6	6	6	6	6	6	6
lsab491 gsq	1	1	1	1	1	1	1	1	1
lsab501 gsq	1	1	1	1	1	1	1	1	1
lsab511 gsq	35	35	35	35	35	35	37	33	33
lsab531 gsq	1	1	1	1	1	1	1	1	1
lsab541 gsq	23	17	22	22	21	20	22	18	17
lsab561 gsq	2	2	2	2	2	2	2	2	2
lsab571 gsq	197	302	536	217	190	224	179	131	131
lsab591 gsq	1	1	1	1	1	1	1	1	1
lsab601 gsq	25	24	24	24	25	24	26	25	24
lsab611 gsq	40	38	38	38	44	39	38	39	38

Table A 69 TDD method, number of nodes, small trees, 3

FT name	1 scheme	2 scheme	3 scheme	4 scheme	5 scheme	6 scheme	7 scheme	8 scheme	Minimum
lisab621 gsq	30	25	25	29	27	27	29	28	25
lisab631 gsq	13	13	13	13	13	13	12	14	12
lisab641 gsq	51	49	49	49	51	51	49	51	49
lisab651 gsq	1	1	1	1	1	1	1	1	1
lisab661 gsq	225	160	160	222	214	227	105	139	105
lisab671 gsq	149	125	125	122	170	122	167	133	122
lisab681 gsq	1	1	1	1	1	1	1	1	1
lisab691 gsq	42	41	41	32	42	32	39	44	32
lisab701 gsq	87	69	69	63	71	56	51	80	51
lisab711 gsq	8	8	8	8	8	8	8	8	8
lisab721 gsq	258	171	171	162	241	193	359	189	162
lisab731 gsq	1	1	1	1	1	1	1	1	1
lisab751 gsq	1	1	1	1	1	1	1	1	1
lisab771 gsq	984	1280	1050	487	734	428	518	775	428
lisab781 gsq	284	122	173	222	230	222	178	128	122
lisab811 gsq	1	1	1	1	1	1	1	1	1
lisab821 gsq	784	681	681	974	655	464	714	693	464
lisab831 gsq	117	97	97	91	117	96	198	90	90
lisab841 gsq	104	108	84	81	92	81	335	80	80
lisab861 gsq	298	368	256	364	314	360	383	301	256
lisab881 gsq	59	50	50	43	62	49	50	62	43
lisab911 gsq	211	105	105	103	154	120	64	152	64
lisab951 gsq	3	3	3	3	3	3	3	3	3
lisab971 gsq	5	5	5	5	5	5	5	5	5
lisab981 gsq	1	1	1	1	1	1	1	1	1
lisa1031 gsq	1	1	1	1	1	1	1	1	1
lisa1041 gsq	1	1	1	1	1	1	1	1	1
lisa1071 gsq	15	15	15	15	15	15	15	15	15
lisa1091 gsq	59	65	65	65	62	59	65	62	59
lisa1101 gsq	31	35	35	35	31	29	66	32	29
lisa1111 gsq	32	32	32	32	32	32	35	32	32
lisa1121 gsq	412	177	181	339	438	313	339	536	177
lisa1131 gsq	245	248	248	348	186	158	358	175	158
lisa1151 gsq	44	44	40	40	36	44	68	46	36
lisa1161 gsq	14	10	14	10	14	13	13	12	10
lisa1191 gsq	10	9	12	12	10	10	12	10	9
lisa1201 gsq	1	1	1	1	1	1	1	1	1
lisa1211 gsq	18	20	24	24	20	20	20	20	18
lisa1221 gsq	25	25	25	25	25	25	24	23	23
lisa1231 gsq	98	66	66	50	92	69	125	87	50
lisa1241 gsq	1276	469	500	538	812	499	1465	552	469
rand1591 gsq	126	134	117	117	115	86	87	117	86
rand1611 gsq	557	656	616	574	420	347	605	510	347
rand1631 gsq	474	1023	878	432	341	320	486	433	320
rand1671 gsq	341	240	245	223	297	247	163	159	159

Table A 70 TDD method, number of nodes, small trees, 4

FT name	1 scheme	2 scheme	3 scheme	4 scheme	5 scheme	6 scheme	7 scheme	8 scheme	Minimum
random61 gsq	737	738	843	403	932	714	790	846	403
rando121 gsq	1414	1190	1162	638	903	741	1089	640	638
rando591 gsq	772	383	463	419	560	416	298	486	298
rand1321 gsq	4351	5791	5791	3333	2322	2657	3042	3518	2322
rand1501 gsq	1000	1422	1167	580	1167	562	503	942	503
rand1581 gsq	679	527	558	528	888	676	486	861	486
lisab521 gsq	1108	2733	2724	1531	970	1389	1042	869	869
lisab741 gsq	4525	1567	1566	1816	7787	1667	1466	9812	1466
lisab891 gsq	904	629	1081	542	600	546	878	609	542
lisa1001 gsq	1735	769	752	1354	1120	1168	1893	750	750
rand1641 gsq	1699	1885	1206	1206	1084	882	1709	959	882
rand1651 gsq	935	603	603	357	479	296	412	636	296
rand1661 gsq	1694	5199	4478	2154	1303	2445	4349	1854	1303

Table A.71: TDD method, number of nodes, 'large' trees

FT name	1 scheme	2 scheme	3 scheme	4 scheme	5 scheme	6 scheme	7 scheme	8 scheme	Minimum
astolfo1 gsq	0	0	0	0	0	0	0	0	0
benjam1 gsq	0.01	0.01	0.01	0.01	0.01	0.01	0	0.01	0
bpfeg031 gsq	0.43	0.39	0.4	0.41	0.39	0.39	0.41	0.39	0.39
bpfen051 gsq	0.28	0.27	0.27	0.28	0.28	0.28	0.27	0.27	0.27
bpfig051 gsq	0.23	0.24	0.24	0.23	0.28	0.25	0.24	0.24	0.23
bpfin051 gsq	0.02	0.01	0.01	0.01	0.01	0.01	0.01	0.01	0.01
bpfs021 gsq	1.25	1.23	1.23	1.28	1.24	1.14	1.17	1.21	1.14
dre10191 gsq	0	0	0.01	0.01	0.01	0.01	0.01	0.01	0
dre10321 gsq	0.01	0.01	0.01	0.01	0.01	0.01	0	0	0
dre10571 gsq	0.02	0.02	0.02	0.03	0.02	0.03	0.02	0.03	0.02
dre10581 gsq	0.23	0.23	0.21	0.21	0.21	0.22	0.21	0.26	0.21
dre10591 gsq	0.49	0.5	0.5	0.51	0.5	0.5	0.47	0.49	0.47
dresden1 gsq	0.46	0.53	0.51	0.46	0.46	0.45	0.48	0.45	0.45
hpsf021 gsq	0.01	0.01	0	0.02	0.01	0.02	0.02	0.02	0
hpsf031 gsq	0.01	0	0.01	0.01	0.01	0.01	0.01	0	0
hpsf211 gsq	0.04	0.1	0.06	0.06	0.04	0.04	0.12	0.04	0.04
hpsf361 gsq	0.01	0	0.01	0.01	0.01	0.01	0	0.01	0
dtree31 gsq	0.01	0.01	0.01	0.01	0.01	0	0.01	0.01	0
dtree41 gsq	0.01	0.02	0.01	0	0	0	0.01	0	0
dtree51 gsq	0	0	0	0.01	0.01	0.01	0	0.01	0
khictre1 gsq	0.13	0.14	0.12	0.15	0.12	0.12	0.14	0.15	0.12
nakashi1 gsq	0.02	0.02	0.02	0.01	0.02	0.01	0.02	0.02	0.01
trials11 gsq	0.02	0.02	0.02	0.02	0.02	0.02	0.02	0.02	0.02
trials41 gsq	0.06	0.19	0.14	0.13	0.05	0.17	0.16	0.03	0.03
random31 gsq	0.02	0.01	0.03	0.02	0.02	0.02	0.02	0.01	0.01
random81 gsq	0.15	0.16	0.2	0.17	0.17	0.15	0.16	0.16	0.15
rand0131 gsq	0.01	0.01	0.01	0.01	0.01	0.02	0.01	0	0
rand0161 gsq	0.16	0.17	0.18	0.16	0.16	0.15	0.16	0.16	0.15
rand0231 gsq	0.02	0.01	0.01	0.02	0.02	0.02	0.02	0.02	0.01
rand0251 gsq	0.01	0	0.01	0	0.01	0.01	0.01	0.01	0
rand0271 gsq	0.17	0.16	0.16	0.17	0.16	0.17	0.16	0.17	0.16
rand0281 gsq	0	0.01	0	0.01	0.01	0	0.01	0	0
rand0291 gsq	0.04	0.03	0.03	0.04	0.03	0.03	0.03	0.02	0.02
rand0301 gsq	0.02	0.02	0.02	0.02	0.02	0.02	0.02	0.02	0.02
rand0311 gsq	0.01	0.01	0.02	0.01	0.02	0.01	0.01	0.01	0.01
rand0331 gsq	0.16	0.16	0.16	0.15	0.15	0.17	0.16	0.17	0.15
rand0341 gsq	0.02	0.02	0.02	0.02	0.02	0.01	0.02	0.02	0.01
rand0351 gsq	0.22	0.17	0.18	0.19	0.18	0.18	0.22	0.17	0.17
rand0361 gsq	0.02	0.02	0.01	0.01	0.02	0.01	0.01	0.01	0.01
rand0371 gsq	0.01	0.01	0.01	0.01	0.01	0.02	0.02	0.01	0.01
rand0381 gsq	0	0	0.01	0.01	0	0.01	0.01	0.01	0
rand0391 gsq	0.02	0.01	0.01	0.02	0.02	0	0.01	0.02	0
rand0401 gsq	0.18	0.15	0.15	0.15	0.15	0.16	0.16	0.16	0.15
rand0421 gsq	0.16	0.16	0.16	0.15	0.15	0.14	0.16	0.15	0.14
rand0431 gsq	0.02	0.02	0.01	0.01	0.01	0.02	0.01	0.02	0.01
rand0441 gsq	0.02	0.02	0.02	0.03	0.03	0.02	0.02	0.02	0.02
rand0451 gsq	0.18	0.16	0.15	0.15	0.16	0.17	0.21	0.15	0.15
rand0461 gsq	0.16	0.16	0.15	0.14	0.15	0.16	0.15	0.16	0.14
rand0471 gsq	0.18	0.17	0.17	0.17	0.16	0.16	0.16	0.17	0.16
rand0481 gsq	0.16	0.15	0.15	0.15	0.15	0.14	0.16	0.16	0.14
rand0521 gsq	0.18	0.16	0.17	0.16	0.17	0.17	0.17	0.17	0.16
rand0531 gsq	0.15	0.18	0.16	0.16	0.16	0.15	0.15	0.15	0.15
rand0541 gsq	0.18	0.18	0.17	0.17	0.17	0.17	0.17	0.16	0.16
rand0551 gsq	0	0	0	0	0	0	0	0	0

Table A 72. TDD method, time, small trees, 1

FT name	1 scheme	2 scheme	3 scheme	4 scheme	5 scheme	6 scheme	7 scheme	8 scheme	Minimum
rando581 gsq	0 16	0 16	0 16	0 15	0 16	0 15	0 16	0 16	0 15
rando601 gsq	0 16	0 17	0 18	0 14	0 14	0 16	0 15	0 14	0 14
rando611 gsq	0 17	0 17	0 19	0 18	0 18	0 17	0 17	0 17	0 17
rando621 gsq	0 01	0 02	0 02	0 01	0 01	0 02	0 02	0 01	0 01
rando631 gsq	0 17	0 16	0 17	0 17	0 17	0 17	0 18	0 16	0 16
rando641 gsq	0 16	0 16	0 17	0 15	0 15	0 14	0 16	0 17	0 14
rando651 gsq	0 17	0 16	0 16	0 16	0 17	0 16	0 16	0 16	0 16
rando661 gsq	0 01	0 01	0 01	0 02	0 02	0 02	0 02	0 01	0 01
rando701 gsq	0 16	0 17	0 17	0 16	0 16	0 15	0 16	0 16	0 15
rando731 gsq	0 02	0 01	0 01	0 02	0 01	0 01	0 02	0 02	0 01
rando751 gsq	0 13	0 13	0 16	0 15	0 13	0 13	0 15	0 13	0 13
rando761 gsq	0 01	0 01	0 01	0 01	0 01	0 01	0	0 01	0
rando771 gsq	0 16	0 16	0 17	0 17	0 16	0 16	0 17	0 17	0 16
rando781 gsq	0 01	0 01	0 01	0 02	0 01	0 01	0 01	0 02	0 01
rando801 gsq	0 01	0 01	0 01	0 01	0 01	0 01	0	0	0
rando831 gsq	0 16	0 16	0 17	0 17	0 16	0 16	0 16	0 17	0 16
rando841 gsq	0 01	0	0 01	0 02	0 01	0 01	0 01	0	0
rando851 gsq	0 17	0 16	0 14	0 13	0 15	0 15	0 13	0 17	0 13
rando871 gsq	0 01	0 01	0 01	0 01	0 01	0 01	0 01	0 01	0 01
rando881 gsq	0 01	0 02	0 01	0 01	0 01	0 01	0 01	0 01	0 01
rando891 gsq	0 02	0 01	0 01	0 01	0 02	0 01	0 02	0 02	0 01
rando911 gsq	0 19	0 19	0 19	0 18	0 18	0 17	0 19	0 18	0 17
rando921 gsq	0 08	0 05	0 04	0 11	0 08	0 12	0 06	0 18	0 04
rando931 gsq	0 18	0 17	0 17	0 17	0 17	0 17	0 16	0 17	0 16
rando951 gsq	0 16	0 16	0 15	0 17	0 15	0 15	0 16	0 18	0 15
rando981 gsq	0 16	0 16	0 16	0 16	0 16	0 17	0 16	0 17	0 16
rando991 gsq	0 03	0 01	0 01	0 02	0 02	0 02	0 02	0 02	0 01
rand1001 gsq	0 02	0 03	0 02	0 01	0 01	0 01	0 02	0 02	0 01
rand1031 gsq	0 16	0 16	0 16	0 15	0 15	0 15	0 15	0 17	0 15
rand1041 gsq	0 15	0 16	0 15	0 15	0 15	0 16	0 16	0 15	0 15
rand1051 gsq	0 16	0 17	0 17	0 17	0 17	0 16	0 17	0 18	0 16
rand1061 gsq	0 01	0 02	0 01	0 02	0 02	0 02	0 02	0 02	0 01
rand1081 gsq	0 18	0 19	0 18	0 18	0 18	0 18	0 2	0 18	0 18
rand1091 gsq	0 01	0 01	0 01	0 01	0 01	0 02	0 01	0 02	0 01
rand1101 gsq	0 01	0 01	0 01	0 02	0 01	0 02	0 01	0 02	0 01
rand1111 gsq	0 01	0 02	0 01	0 01	0 01	0 01	0	0 01	0
rand1151 gsq	0 01	0 01	0 02	0 01	0 01	0 01	0 01	0 01	0 01
rand1161 gsq	0	0 01	0 01	0	0 01	0 01	0 01	0 01	0
rand1171 gsq	0 14	0 13	0 13	0 14	0 12	0 12	0 13	0 12	0 12
rand1181 gsq	0 01	0 02	0 01	0 02	0 01	0 02	0 01	0 01	0 01
rand1191 gsq	0 16	0 15	0 16	0 15	0 16	0 16	0 16	0 16	0 15
rand1201 gsq	0 15	0 16	0 16	0 16	0 16	0 15	0 16	0 15	0 15
rand1211 gsq	0 02	0	0 01	0 01	0 01	0 01	0 01	0	0
rand1231 gsq	0 01	0 02	0 01	0 01	0 02	0 01	0 02	0 01	0 01
rand1241 gsq	0 01	0	0	0 01	0	0	0	0 01	0
rand1251 gsq	0	0	0	0	0	0 01	0 01	0 02	0
rand1261 gsq	0 01	0 01	0	0 01	0 01	0 01	0 01	0	0
rand1271 gsq	0 01	0 02	0 01	0 01	0 01	0 01	0 01	0 01	0 01
rand1281 gsq	0 01	0 01	0	0	0 01	0	0 01	0 01	0
rand1291 gsq	0 01	0 01	0 02	0 01	0 01	0 01	0 01	0 01	0 01
rand1301 gsq	0 01	0 01	0 01	0	0 01	0	0	0	0
rand1341 gsq	0	0	0 01	0 01	0	0 01	0 01	0 01	0
rand1351 gsq	0 01	0 01	0 01	0	0 01	0	0 01	0 01	0
rand1371 gsq	0 12	0 13	0 12	0 14	0 13	0 13	0 13	0 13	0 12

Table A 73: TDD method, time, small trees, 2

FT name	1 scheme	2 scheme	3 scheme	4 scheme	5 scheme	6 scheme	7 scheme	8 scheme	Minimum
rand1381 gsq	0 16	0 17	0 2	0 17	0 17	0 18	0 16	0 16	0 17
rand1391 gsq	0 02	0 02	0 02	0 02	0 02	0 02	0 02	0 02	0 02
rand1411 gsq	0 01	0 01	0 01	0 01	0 01	0 01	0	0 01	0 01
rand1421 gsq	0 14	0 18	0 14	0 14	0 14	0 16	0 14	0 14	0 18
rand1431 gsq	0 17	0 17	0 17	0 17	0 17	0 18	0 17	0 17	0 17
rand1441 gsq	0 01	0 01	0 02	0 02	0 01	0 02	0 01	0 01	0 01
rand1451 gsq	0 15	0 16	0 14	0 16	0 16	0 16	0 14	0 15	0 16
rand1461 gsq	0 01	0 01	0 02	0 01	0 01	0 01	0 01	0 01	0 01
rand1471 gsq	0 14	0 14	0 13	0 15	0 13	0 14	0 13	0 14	0 14
rand1481 gsq	0 15	0 16	0 15	0 16	0 16	0 15	0 15	0 15	0 16
rand1491 gsq	0 01	0 01	0 01	0 01	0 01	0 01	0	0 01	0 01
rand1511 gsq	0 01	0	0 01	0 01	0 01	0 02	0	0 01	0
rand1531 gsq	0 15	0 15	0 16	0 14	0 16	0 16	0 14	0 15	0 15
rand1541 gsq	0 01	0 01	0	0 01	0 01	0	0	0 01	0 01
rand1551 gsq	0 01	0 01	0 01	0 01	0 02	0 02	0 01	0 01	0 01
rand1561 gsq	0 01	0 02	0 01	0 01	0	0 01	0	0 01	0 02
lisaba11 gsq	0 14	0 13	0 13	0 13	0 13	0 13	0 13	0 14	0 13
lisaba31 gsq	0 18	0 18	0 17	0 17	0 2	0 16	0 16	0 18	0 18
lisaba41 gsq	0 07	0 06	0 03	0 09	0 09	0 04	0 03	0 07	0 06
lisaba51 gsq	0 16	0 17	0 16	0 16	0 17	0 18	0 15	0 16	0 17
lisaba61 gsq	0 04	0 03	0 04	0 04	0 03	0 04	0 03	0 04	0 03
lisaba71 gsq	0 18	0 17	0 17	0 18	0 18	0 19	0 17	0 18	0 17
lisaba91 gsq	0 16	0 15	0 14	0 15	0 16	0 15	0 14	0 16	0 15
lisab101 gsq	0 01	0 01	0 02	0 02	0 01	0 02	0 01	0 01	0 01
lisab111 gsq	0 01	0	0 01	0	0 01	0 01	0	0 01	0
lisab131 gsq	0	0 02	0	0 01	0 01	0 01	0	0	0 02
lisab171 gsq	0 14	0 13	0 14	0 14	0 13	0 13	0 13	0 14	0 13
lisab191 gsq	2 25	2 26	2 23	2 26	2 31	2 31	2 22	2 25	2 26
lisab201 gsq	0 77	0 77	0 78	0 78	0 8	0 78	0 77	0 77	0 77
lisab241 gsq	0 03	0 03	0 03	0 03	0 03	0 03	0 03	0 03	0 03
lisab251 gsq	0 01	0 01	0 02	0 01	0 01	0 02	0 01	0 01	0 01
lisab271 gsq	0 19	0 17	0 18	0 18	0 2	0 2	0 17	0 19	0 17
lisab281 gsq	0 01	0	0	0 01	0 01	0 01	0	0 01	0
lisab301 gsq	0 16	0 17	0 15	0 16	0 16	0 17	0 15	0 16	0 17
lisab311 gsq	0 11	0 05	0 05	0 03	0 12	0 03	0 03	0 11	0 05
lisab341 gsq	0 15	0 15	0 15	0 13	0 18	0 15	0 13	0 15	0 15
lisab351 gsq	0 04	0 02	0 02	0 03	0 03	0 02	0 02	0 04	0 02
lisab361 gsq	0 01	0 02	0 01	0 01	0 01	0 01	0 01	0 01	0 02
lisab371 gsq	0 16	0 15	0 16	0 16	0 17	0 17	0 15	0 16	0 15
lisab421 gsq	0 01	0 01	0 01	0 01	0 01	0 01	0 01	0 01	0 01
lisab441 gsq	0 14	0 15	0 15	0 15	0 16	0 15	0 14	0 14	0 15
lisab461 gsq	1 97	2	1 97	1 96	1 98	2 07	1 96	1 97	2
lisab471 gsq	0 01	0 01	0 03	0 02	0 02	0 01	0 01	0 01	0 01
lisab481 gsq	0 02	0 01	0 01	0 02	0 02	0 01	0 01	0 02	0 01
lisab491 gsq	0 03	0 03	0 03	0 02	0 04	0 03	0 02	0 03	0 03
lisab501 gsq	0 01	0 01	0	0 01	0 01	0 01	0	0 01	0 01
lisab511 gsq	0 01	0	0 01	0 01	0 01	0 01	0	0 01	0
lisab531 gsq	0 01	0	0	0 01	0 01	0 01	0	0 01	0
lisab541 gsq	0 02	0	0 01	0 01	0 01	0 01	0	0 02	0
lisab561 gsq	0	0 01	0 01	0 01	0 01	0 01	0	0	0 01
lisab571 gsq	0 03	0	0 02	0 01	0 01	0 01	0	0 03	0
lisab591 gsq	0 07	0 07	0 07	0 07	0 11	0 08	0 07	0 07	0 07
lisab601 gsq	0	0 01	0 01	0 01	0	0	0	0	0 01
lisab611 gsq	0 02	0 01	0 01	0 02	0 01	0 02	0 01	0 02	0 01

Table A 74. TDD method, time, small trees, 3

FT name	1 scheme	2 scheme	3 scheme	4 scheme	5 scheme	6 scheme	7 scheme	8 scheme	Minimum
lisab621 gsq	0.14	0.12	0.13	0.11	0.13	0.13	0.15	0.14	0.11
lisab631 gsq	0.02	0.01	0.02	0.01	0.02	0.02	0.02	0.02	0.01
lisab641 gsq	0.17	0.16	0.19	0.17	0.17	0.17	0.2	0.2	0.16
lisab651 gsq	0.02	0.01	0	0.01	0.01	0.01	0	0.01	0
lisab661 gsq	0.01	0.02	0.02	0.01	0.01	0.01	0.02	0.01	0.01
lisab671 gsq	0.04	0.04	0.05	0.05	0.05	0.05	0.05	0.05	0.04
lisab681 gsq	0.01	0.01	0.01	0.01	0.01	0.01	0.01	0.01	0.01
lisab691 gsq	0	0	0	0.01	0	0.01	0.01	0.01	0
lisab701 gsq	0.01	0.01	0.01	0.02	0.02	0.01	0.01	0	0
lisab711 gsq	0.13	0.12	0.12	0.12	0.12	0.13	0.14	0.12	0.12
lisab721 gsq	0.02	0.01	0.03	0.02	0.02	0.03	0.03	0.02	0.01
lisab731 gsq	0.01	0.01	0.01	0.02	0.01	0.01	0.01	0.01	0.01
lisab751 gsq	0.01	0	0	0.01	0.01	0	0.01	0.01	0
lisab771 gsq	0.2	0.22	0.2	0.16	0.17	0.15	0.15	0.16	0.15
lisab781 gsq	0.02	0.02	0.02	0.02	0.01	0.02	0.01	0.02	0.01
lisab811 gsq	0.01	0	0	0	0.01	0	0.01	0.01	0
lisab821 gsq	1.43	1.49	1.48	1.45	1.42	1.4	1.42	1.37	1.37
lisab831 gsq	0	0	0.01	0	0.01	0.01	0.01	0.02	0
lisab841 gsq	0.01	0.01	0.01	0.01	0.01	0.01	0.01	0	0
lisab861 gsq	0.01	0.02	0.02	0.01	0.01	0.02	0.02	0.02	0.01
lisab881 gsq	0	0.01	0.01	0.01	0	0.01	0.02	0.01	0
lisab911 gsq	0.08	0.09	0.09	0.08	0.09	0.1	0.09	0.09	0.08
lisab951 gsq	0	0	0	0.01	0.01	0	0	0	0
lisab971 gsq	0.01	0.01	0.01	0	0.01	0.01	0.01	0.01	0
lisab981 gsq	0.01	0	0	0.01	0.01	0.01	0	0.01	0
lisa1031 gsq	0.01	0.01	0.01	0.01	0	0.01	0	0	0
lisa1041 gsq	0.01	0.01	0.01	0	0.01	0.01	0.01	0	0
lisa1071 gsq	0.13	0.13	0.12	0.14	0.12	0.12	0.12	0.15	0.12
lisa1091 gsq	0.02	0.01	0.01	0.02	0.01	0.01	0.01	0.01	0.01
lisa1101 gsq	0.01	0.02	0.02	0.02	0.01	0.01	0.02	0.01	0.01
lisa1111 gsq	0.14	0.14	0.16	0.16	0.15	0.14	0.16	0.15	0.14
lisa1121 gsq	0.17	0.16	0.16	0.17	0.16	0.16	0.16	0.16	0.16
lisa1131 gsq	0.13	0.13	0.16	0.13	0.12	0.12	0.16	0.13	0.12
lisa1151 gsq	0.02	0.01	0.01	0.02	0.01	0.01	0.02	0.02	0.01
lisa1161 gsq	0.01	0.01	0.01	0.01	0.02	0.01	0.01	0.01	0.01
lisa1191 gsq	0	0.01	0.01	0	0.01	0.01	0.01	0.01	0
lisa1201 gsq	0	0	0.01	0.01	0.01	0.01	0.01	0.01	0
lisa1211 gsq	0.14	0.13	0.13	0.12	0.12	0.12	0.13	0.14	0.12
lisa1221 gsq	0.02	0.02	0.01	0.01	0.01	0.01	0.02	0.02	0.01
lisa1231 gsq	0.01	0.02	0.01	0.02	0.02	0.02	0.02	0.01	0.01
lisa1241 gsq	0.09	0.05	0.06	0.06	0.07	0.05	0.14	0.04	0.04
rand1591 gsq	0.02	0.01	0.01	0	0	0	0.01	0.01	0
rand1611 gsq	0.04	0.05	0.03	0.04	0.03	0.02	0.04	0.03	0.02
rand1631 gsq	0.15	0.18	0.16	0.16	0.14	0.14	0.14	0.15	0.14
rand1671 gsq	0.16	0.15	0.16	0.15	0.15	0.16	0.15	0.16	0.15

Table A 75: TDD method, time, small trees, 4

FT name	1 scheme	2 scheme	3 scheme	4 scheme	5 scheme	6 scheme	7 scheme	8 scheme	Minimum
random61 gsq	0 03	0 06	0 07	0 02	0 07	0 04	0 08	0 07	0 02
rando121 gsq	0 19	0 16	0 15	0 08	0 1	0 09	0 1	0 08	0 08
rando591 gsq	0 05	0 03	0 03	0 02	0 04	0 03	0 01	0 02	0 01
rand1321 gsq	1 01	2 94	3 06	0 68	0 26	0 32	0 54	0 54	0 26
rand1501 gsq	0 07	0 2	0 16	0 04	0 08	0 03	0 03	0 06	0 03
rand1581 gsq	0 15	0 15	0 18	0 15	0 17	0 17	0 15	0 16	0 15
lisab521 gsq	0 07	0 9	0 92	0 14	0 06	0 12	0 11	0 05	0 05
lisab741 gsq	6 18	4 94	4 85	4 81	8 94	4 87	4 95	11 73	4 81
lisab891 gsq	0 07	0 07	0 14	0 04	0 04	0 05	0 11	0 05	0 04
lisa1001 gsq	0 18	0 06	0 06	0 16	0 08	0 09	0 29	0 05	0 05
rand1641 gsq	0 39	0 59	0 29	0 29	0 23	0 23	0 62	0 18	0 18
rand1651 gsq	0 22	0 2	0 2	0 18	0 21	0 2	0 19	0 21	0 18
rand1661 gsq	0 21	3 68	2 36	0 45	0 16	0 49	1 38	0 22	0 16

Table A 76 TDD method, time, 'large' trees

FT name	1 scheme	2 scheme	3 scheme	4 scheme	5 scheme	6 scheme	7 scheme	8 scheme	Minimum
astolfo1 gsq	89	90	90	91	80	91	91	82	80
benjam1 gsq	209	194	194	135	209	209	137	247	135
bpfeq031 gsq	1	1	1	1	1	1	1	1	1
bpfen051 gsq	1	1	1	1	1	1	1	1	1
bpfig051 gsq	1	1	1	1	1	1	1	1	1
bpfin051 gsq	1	1	1	1	1	1	1	1	1
bpfs021 gsq	109	61	73	77	109	61	94	86	61
dre10191 gsq	1	1	1	1	1	1	1	1	1
dre10321 gsq	1	1	1	1	1	1	1	1	1
dre10571 gsq	1	1	1	1	1	1	1	1	1
dre10581 gsq	276	294	294	284	249	248	277	194	194
dre10591 gsq	783	867	859	840	458	423	849	300	300
dresden1 gsq	1479	550	550	518	1247	727	721	349	349
hpsf021 gsq	402	251	251	273	291	222	279	291	222
hpsf031 gsq	45	45	45	48	45	48	48	45	45
hpsf211 gsq	1440	2646	2635	1832	1110	1789	2298	1078	1078
hpsf361 gsq	54	52	52	52	54	52	52	54	52
dtree31 gsq	1	1	1	1	1	1	1	1	1
dtree41 gsq	1	1	1	1	1	1	1	1	1
dtree51 gsq	1	1	1	1	1	1	1	1	1
khictre1 gsq	79	81	74	74	79	74	78	81	74
nakashi1 gsq	326	242	243	266	308	268	242	305	242
trials11 gsq	322	267	267	267	343	281	274	281	267
trials41 gsq	971	1237	1115	1210	994	1129	1046	895	895
random31 gsq	375	391	357	352	432	397	519	488	352
random81 gsq	39	39	39	39	39	39	39	39	39
rand0131 gsq	354	337	337	337	311	326	354	324	311
rand0161 gsq	107	107	107	107	107	107	103	107	103
rand0231 gsq	181	173	173	173	179	169	173	184	169
rand0251 gsq	1	1	1	1	1	1	1	1	1
rand0271 gsq	97	97	97	98	119	113	98	114	97
rand0281 gsq	1	1	1	1	1	1	1	1	1
rand0291 gsq	877	723	717	810	697	682	672	671	671
rand0301 gsq	395	263	288	288	410	308	217	374	217
rand0311 gsq	25	25	25	25	25	25	25	25	25
rand0331 gsq	257	239	239	224	253	221	140	200	140
rand0341 gsq	121	117	117	118	116	116	118	117	116
rand0351 gsq	82	82	82	82	74	82	71	72	71
rand0361 gsq	1	1	1	1	1	1	1	1	1
rand0371 gsq	212	212	209	209	211	209	115	212	115
rand0381 gsq	1	1	1	1	1	1	1	1	1
rand0391 gsq	473	300	300	393	471	393	320	404	300
rand0401 gsq	89	67	85	85	89	89	65	83	65
rand0421 gsq	13	13	13	13	13	13	13	13	13
rand0431 gsq	70	68	68	71	67	67	71	67	67
rand0441 gsq	834	796	786	767	946	805	751	845	751
rand0451 gsq	163	187	187	169	147	187	187	146	146
rand0461 gsq	25	25	25	25	25	25	25	25	25
rand0471 gsq	298	317	283	282	301	289	342	287	282
rand0481 gsq	18	18	18	18	18	18	18	18	18
rand0521 gsq	286	222	267	267	276	285	195	296	195
rand0531 gsq	13	13	13	13	13	13	13	13	13
rand0541 gsq	154	87	139	139	172	158	139	179	87
rand0551 gsq	62	62	62	62	62	62	62	71	62

Table A 77: MPPI method, number of nodes, small trees, 1

FT name	1 scheme	2 scheme	3 scheme	4 scheme	5 scheme	6 scheme	7 scheme	8 scheme	Minimum
rando581 gsq	13	13	13	13	13	13	13	13	13
rando601 gsq	72	72	60	72	76	72	60	72	60
rando611 gsq	222	299	201	225	214	184	299	209	184
rando621 gsq	37	37	37	37	37	37	37	37	37
rando631 gsq	52	52	52	52	52	52	52	52	52
rando641 gsq	195	181	181	178	185	178	153	190	153
rando651 gsq	73	86	69	71	66	71	69	63	63
rando661 gsq	32	32	32	32	32	32	32	32	32
rando701 gsq	128	105	83	105	162	105	96	188	83
rando731 gsq	160	143	143	138	160	160	117	160	117
rando751 gsq	1	1	1	1	1	1	1	1	1
rando761 gsq	44	44	44	44	44	44	44	44	44
rando771 gsq	208	110	110	110	151	110	100	128	100
rando781 gsq	1	1	1	1	1	1	1	1	1
rando801 gsq	33	33	33	33	33	33	32	32	32
rando831 gsq	142	76	109	109	127	80	91	114	76
rando841 gsq	544	315	297	388	486	377	287	369	287
rando851 gsq	1	1	1	1	1	1	1	1	1
rando871 gsq	1	1	1	1	1	1	1	1	1
rando881 gsq	99	88	95	95	94	91	89	93	88
rando891 gsq	141	141	141	141	141	141	136	141	136
rando911 gsq	744	679	679	742	747	708	896	768	679
rando921 gsq	741	809	774	773	673	713	767	692	673
rando931 gsq	82	82	82	82	82	82	82	82	82
rando951 gsq	124	110	111	111	146	146	111	132	110
rando981 gsq	871	619	619	620	702	595	580	675	580
rando991 gsq	633	512	488	471	521	488	471	515	471
rand1001 gsq	20	20	20	20	20	20	20	20	20
rand1031 gsq	140	131	141	141	138	138	131	138	131
rand1041 gsq	36	36	36	36	36	36	36	36	36
rand1051 gsq	99	99	93	93	99	93	95	96	93
rand1061 gsq	39	39	39	39	39	39	39	39	39
rand1081 gsq	535	442	478	521	495	525	555	507	442
rand1091 gsq	740	641	663	730	728	835	519	683	519
rand1101 gsq	41	41	41	38	38	38	38	38	38
rand1111 gsq	183	166	166	167	173	163	146	160	146
rand1151 gsq	310	222	222	297	279	261	240	231	222
rand1161 gsq	86	81	79	79	79	84	79	78	78
rand1171 gsq	43	49	47	47	46	44	47	46	43
rand1181 gsq	329	276	257	253	303	286	202	275	202
rand1191 gsq	21	21	21	21	21	21	21	21	21
rand1201 gsq	532	271	266	489	501	459	458	514	266
rand1211 gsq	109	91	91	87	109	100	87	117	87
rand1231 gsq	22	22	22	22	26	22	22	25	22
rand1241 gsq	1	1	1	1	1	1	1	1	1
rand1251 gsq	39	39	40	40	40	40	39	39	39
rand1261 gsq	532	510	510	455	560	534	436	558	436
rand1271 gsq	1	1	1	1	1	1	1	1	1
rand1281 gsq	201	205	216	198	191	175	217	197	175
rand1291 gsq	1	1	1	1	1	1	1	1	1
rand1301 gsq	12	12	12	12	12	12	12	12	12
rand1341 gsq	338	317	317	371	336	342	343	351	317
rand1351 gsq	229	204	190	186	215	209	238	233	186
rand1371 gsq	111	111	111	111	99	99	102	106	99

Table A 78 MPPI method, number of nodes, small trees, 2

FT name	1 scheme	2 scheme	3 scheme	4 scheme	5 scheme	6 scheme	7 scheme	8 scheme	Minimum
rand1381 gsq	43	43	43	43	43	43	43	43	43
rand1391 gsq	538	643	575	548	555	485	670	517	485
rand1411 gsq	27	27	27	27	27	27	26	26	26
rand1421 gsq	1038	1602	1754	1280	1168	1071	764	1107	764
rand1431 gsq	37	37	37	37	37	37	37	37	37
rand1441 gsq	308	374	362	369	345	341	410	284	284
rand1451 gsq	31	31	31	31	31	31	31	34	31
rand1461 gsq	121	121	121	121	124	82	82	103	82
rand1471 gsq	613	697	725	640	544	536	579	590	536
rand1481 gsq	13	13	13	13	13	13	13	13	13
rand1491 gsq	1	1	1	1	1	1	1	1	1
rand1511 gsq	1	1	1	1	1	1	1	1	1
rand1531 gsq	16	16	16	16	16	16	16	16	16
rand1541 gsq	1	1	1	1	1	1	1	1	1
rand1551 gsq	725	644	674	590	601	638	642	549	549
rand1561 gsq	1	1	1	1	1	1	1	1	1
lisaba11 gsq	194	155	161	149	185	176	161	172	149
lisaba31 gsq	2189	2347	2322	2130	2108	1802	2772	2120	1802
lisaba41 gsq	2692	3445	1854	1355	1471	2012	1668	1515	1355
lisaba51 gsq	465	450	406	392	456	415	381	473	381
lisaba61 gsq	431	393	393	310	306	353	307	311	306
lisaba71 gsq	199	172	169	164	192	205	169	189	164
lisaba91 gsq	102	78	78	75	90	75	81	98	75
lisab101 gsq	593	421	399	399	503	474	272	463	272
lisab111 gsq	1	1	1	1	1	1	1	1	1
lisab131 gsq	27	27	27	27	27	27	26	26	26
lisab171 gsq	227	184	190	184	216	222	190	192	184
lisab191 gsq	1	1	1	1	1	1	1	1	1
lisab201 gsq	1	1	1	1	1	1	1	1	1
lisab241 gsq	1	1	1	1	1	1	1	1	1
lisab251 gsq	175	175	171	173	177	179	175	185	171
lisab271 gsq	626	593	593	414	592	451	282	606	282
lisab281 gsq	1	1	1	1	1	1	1	1	1
lisab301 gsq	127	133	133	131	129	133	125	126	125
lisab311 gsq	3605	4280	4208	2508	2213	1862	2413	1863	1862
lisab341 gsq	107	87	103	80	107	80	103	119	80
lisab351 gsq	1309	1159	1202	1319	1058	1216	1022	849	848
lisab361 gsq	501	579	532	351	538	401	418	552	351
lisab371 gsq	30	30	30	30	30	30	30	32	30
lisab421 gsq	1	1	1	1	1	1	1	1	1
lisab441 gsq	62	70	70	70	62	70	61	62	61
lisab461 gsq	1	1	1	1	1	1	1	1	1
lisab471 gsq	24	21	21	21	24	24	21	24	21
lisab481 gsq	17	17	17	17	17	17	17	17	17
lisab491 gsq	1	1	1	1	1	1	1	1	1
lisab501 gsq	1	1	1	1	1	1	1	1	1
lisab511 gsq	73	73	73	73	73	73	78	75	73
lisab531 gsq	1	1	1	1	1	1	1	1	1
lisab541 gsq	87	85	85	85	87	85	86	85	85
lisab561 gsq	7	7	7	7	7	7	7	7	7
lisab571 gsq	438	593	511	412	457	453	561	389	389
lisab591 gsq	1	1	1	1	1	1	1	1	1
lisab601 gsq	87	61	61	61	86	61	70	86	61
lisab611 gsq	145	158	158	158	163	157	158	157	145

Table A.79 MPPI method, number of nodes, small trees, 3

FT name	1 scheme	2 scheme	3 scheme	4 scheme	5 scheme	6 scheme	7 scheme	8 scheme	Minimum
lisab621 gsq	82	77	77	83	80	80	83	80	77
lisab631 gsq	34	34	34	34	34	34	33	34	33
lisab641 gsq	100	88	88	88	100	100	88	100	88
lisab651 gsq	1	1	1	1	1	1	1	1	1
lisab661 gsq	545	590	590	786	560	602	366	485	366
lisab671 gsq	365	367	367	276	413	412	382	340	276
lisab681 gsq	1	1	1	1	1	1	1	1	1
lisab691 gsq	121	122	122	100	126	100	122	127	100
lisab701 gsq	261	241	237	224	244	211	218	257	211
lisab711 gsq	16	16	16	16	16	16	16	16	16
lisab721 gsq	816	581	581	565	752	691	740	675	565
lisab731 gsq	1	1	1	1	1	1	1	1	1
lisab751 gsq	1	1	1	1	1	1	1	1	1
lisab771 gsq	1366	1064	1002	1082	1243	962	890	1270	890
lisab781 gsq	926	478	641	739	902	747	517	669	478
lisab811 gsq	1	1	1	1	1	1	1	1	1
lisab821 gsq	2207	2427	2427	3231	1713	1338	2956	1830	1338
lisab831 gsq	303	284	284	274	303	281	295	272	272
lisab841 gsq	283	341	311	315	303	315	296	283	283
lisab861 gsq	880	821	797	739	866	803	710	818	710
lisab881 gsq	169	113	113	102	172	133	113	172	102
lisab911 gsq	814	421	421	386	646	476	245	586	245
lisab951 gsq	9	9	9	9	9	9	9	9	9
lisab971 gsq	18	18	18	18	18	18	18	18	18
lisab981 gsq	1	1	1	1	1	1	1	1	1
lisa1031 gsq	1	1	1	1	1	1	1	1	1
lisa1041 gsq	1	1	1	1	1	1	1	1	1
lisa1071 gsq	31	31	31	31	31	31	31	31	31
lisa1091 gsq	179	218	218	218	185	174	218	185	174
lisa1101 gsq	271	310	299	299	288	284	318	297	271
lisa1111 gsq	84	84	84	85	85	85	87	85	84
lisa1121 gsq	1854	612	626	961	1471	1202	1132	1624	612
lisa1131 gsq	564	520	520	649	498	437	505	474	437
lisa1151 gsq	143	143	143	143	114	136	124	129	114
lisa1161 gsq	37	31	37	31	37	30	30	35	30
lisa1191 gsq	24	20	23	23	24	24	23	24	20
lisa1201 gsq	1	1	1	1	1	1	1	1	1
lisa1211 gsq	52	54	54	54	52	52	54	52	52
lisa1221 gsq	66	66	66	66	66	66	55	55	55
lisa1231 gsq	346	289	289	237	307	299	474	286	237
lisa1241 gsq	3356	1593	1587	1915	2328	1816	3337	1993	1587
rand1591 gsq	263	278	280	280	250	249	250	252	249
rand1611 gsq	1014	945	1008	1025	897	877	1061	1245	877
rand1631 gsq	1482	2415	2132	1486	1168	1025	1314	1283	1025
rand1671 gsq	696	579	648	578	597	474	463	600	463

Table A 80: MPPI method, number of nodes, small trees, 4

FT name	1 scheme	2 scheme	3 scheme	4 scheme	5 scheme	6 scheme	7 scheme	8 scheme	Minimum
random61 gsq	286729	337198	368010	66026	261841	83111	162947	143224	66026
rando121 gsq	276681	212482	212671	71708	146849	50583	228036	129437	50583
rando591 gsq	563907	170421	225213	133177	520935	182838	54376	94041	54376
rand1321 gsq	750663	-	-	361713	199487	211555	305715	489675	199487
rand1501 gsq	792968	-	-	464739	-	495893	281552	468562	281552
rand1581 gsq	89379	656368	-	33635	82657	45901	26907	53116	26907
lisab521 gsq	322744	487690	481472	144853	143292	82177	533476	95799	82177
lisab741 gsq	-	626541	422154	469649	-	257008	-	-	257008
lisab891 gsq	121606	367312	402537	32770	72829	42453	317895	66335	32770
lisa1001 gsq	-	445238	470687	-	620601	358023	-	609048	358023
rand1641 gsq	207516	114870	84930	84930	100868	67441	196606	92760	67441
rand1651 gsq	-	506604	506604	330199	421818	553647	283340	-	283340
rand1661 gsq	-	-	-	-	-	-	-	-	-

Table A 81 MPPI method, number of nodes, 'large' trees

FT name	1 scheme	2 scheme	3 scheme	4 scheme	5 scheme	6 scheme	7 scheme	8 scheme	Minimum
astolfo1 gsq	0 01	0 02	0 02	0 02	0 01	0 01	0 02	0 01	0 01
benjam1 gsq	0 04	0 04	0 03	0 03	0 03	0 04	0 02	0 03	0 02
bpfig031 gsq	0 48	0 54	0 51	0 5	0 54	0 47	0 53	0 48	0 47
bpfen051 gsq	0 37	0 37	0 38	0 37	0 36	0 36	0 36	0 37	0 36
bpfig051 gsq	0 33	0 36	0 33	0 33	0 38	0 33	0 33	0 33	0 33
bpfin051 gsq	0 01	0 02	0 01	0 01	0 02	0 01	0 01	0 01	0 01
bpfsw021 gsq	1 31	1 3	1 3	1 28	1 28	1 26	1 28	1 29	1 26
dre10191 gsq	0	0	0	0	0	0	0	0	0
dre10321 gsq	0	0	0	0	0	0	0	0	0
dre10571 gsq	0 02	0 02	0 03	0 02	0 02	0 02	0 02	0 02	0 02
dre10581 gsq	0 12	0 14	0 13	0 11	0 12	0 12	0 12	0 1	0 1
dre10591 gsq	0 93	1 04	0 95	0 95	0 7	0 68	0 99	0 66	0 66
dresden1 gsq	3 01	1 06	1 06	0 93	2 08	1 08	1 13	0 86	0 86
hpsif021 gsq	13 6	0 83	0 82	0 61	1 66	0 33	1 09	2 63	0 33
hpsif031 gsq	0 03	0 02	0 02	0 01	0 02	0 01	0 01	0 01	0 01
hpsif211 gsq	34 1	184 29	130 58	52 25	12 81	53 77	149 97	9 21	9 21
hpsif361 gsq	0 01	0 02	0 02	0 01	0 02	0 01	0 01	0 01	0 01
dtree31 gsq	0 01	0	0	0	0	0	0 01	0	0
dtree41 gsq	0	0	0	0 01	0	0	0	0	0
dtree51 gsq	0	0	0	0	0	0	0	0	0
khictre1 gsq	0 04	0 06	0 05	0 05	0 05	0 05	0 04	0 06	0 04
nakashi1 gsq	0 66	0 39	0 2	0 43	0 57	0 63	0 41	0 28	0 2
trials11 gsq	0 13	0 34	0 34	0 34	0 19	0 03	0 6	0 03	0 03
trials41 gsq	9 16	153 19	169 56	94 86	7 22	49 81	70 56	11 51	7 22
random31 gsq	0 56	0 47	0 37	0 36	0 96	0 4	23 47	0 66	0 36
random81 gsq	0 05	0 05	0 06	0 05	0 05	0 05	0 06	0 05	0 05
rand0131 gsq	0 41	2 64	2 13	2 12	0 16	0 87	2 72	0 33	0 16
rand0161 gsq	0 06	0 06	0 07	0 06	0 06	0 05	0 05	0 05	0 05
rand0231 gsq	0 06	0 06	0 06	0 05	0 05	0 05	0 04	0 06	0 04
rand0251 gsq	0	0	0	0	0	0	0 01	0	0
rand0271 gsq	0 03	0 04	0 04	0 04	0 04	0 03	0 03	0 03	0 03
rand0281 gsq	0	0	0	0	0 01	0	0	0	0
rand0291 gsq	784 89	223 68	261 61	464 38	321 07	482 82	104 82	31 58	31 58
rand0301 gsq	0 12	0 1	0 11	0 11	0 14	0 04	0 05	0 12	0 04
rand0311 gsq	0 02	0 02	0 02	0 02	0 01	0 03	0 02	0 02	0 01
rand0331 gsq	0 06	0 1	0 11	0 06	0 05	0 06	0 05	0 05	0 05
rand0341 gsq	0 01	0 03	0 03	0 03	0 03	0 02	0 02	0 02	0 01
rand0351 gsq	0 09	0 09	0 1	0 08	0 08	0 07	0 08	0 08	0 07
rand0361 gsq	0	0	0	0	0 01	0	0 01	0	0
rand0371 gsq	0 03	0 04	0 04	0 04	0 02	0 03	0 02	0 02	0 02
rand0381 gsq	0	0	0	0	0	0	0	0	0
rand0391 gsq	0 28	0 05	0 05	0 16	0 22	0 16	0 35	0 18	0 05
rand0401 gsq	0 03	0 03	0 04	0 04	0 04	0 04	0 03	0 03	0 03
rand0421 gsq	0 03	0 05	0 04	0 04	0 03	0 03	0 03	0 04	0 03
rand0431 gsq	0 03	0 01	0 02	0 02	0 01	0 02	0 02	0 01	0 01
rand0441 gsq	14 74	115 06	105 29	6 13	71 19	4 96	4 35	11 72	4 35
rand0451 gsq	0 06	0 07	0 06	0 09	0 05	0 07	0 07	0 04	0 04
rand0461 gsq	0 05	0 04	0 03	0 05	0 05	0 04	0 05	0 05	0 03
rand0471 gsq	0 26	0 66	0 32	0 57	0 31	0 15	0 75	0 27	0 15
rand0481 gsq	0 03	0 02	0 03	0 03	0 05	0 04	0 03	0 03	0 02
rand0521 gsq	0 08	0 08	0 08	0 09	0 07	0 08	0 06	0 08	0 06
rand0531 gsq	0 04	0 03	0 02	0 03	0 04	0 03	0 03	0 02	0 02
rand0541 gsq	0 05	0 05	0 03	0 06	0 05	0 05	0 05	0 06	0 03
rand0551 gsq	0 05	0 04	0 06	0 05	0 05	0 06	0 06	0 05	0 04

Table A 82: MPPI method, time, small trees, 1

FT name	1 scheme	2 scheme	3 scheme	4 scheme	5 scheme	6 scheme	7 scheme	8 scheme	Minimum
rando581 gsq	0 04	0 03	0 03	0 04	0 03	0 03	0 03	0 05	0 03
rando601 gsq	0 06	0 06	0 07	0 08	0 08	0 06	0 07	0 07	0 06
rando611 gsq	0 02	0 26	0 06	0 05	0 02	0 02	0 26	0 02	0 02
rando621 gsq	0 06	0 06	0 06	0 08	0 07	0 07	0 06	0 07	0 06
rando631 gsq	0 05	0 04	0 06	0 04	0 06	0 05	0 05	0 06	0 04
rando641 gsq	0 1	0 07	0 08	0 1	0 1	0 08	0 19	0 12	0 07
rando651 gsq	0 02	0 01	0 02	0 02	0 03	0 02	0 02	0 02	0 01
rando661 gsq	0 03	0 04	0 03	0 03	0 03	0 04	0 03	0 04	0 03
rando701 gsq	0 03	0 01	0 02	0 03	0 03	0 01	0 03	0 02	0 01
rando731 gsq	0 05	0 05	0 05	0 05	0 06	0 05	0 05	0 06	0 05
rando751 gsq	0	0	0	0	0	0	0	0	0
rando761 gsq	0 04	0 04	0 05	0 05	0 05	0 05	0 05	0 05	0 04
rando771 gsq	0 04	0 02	0 01	0 02	0 02	0 02	0 02	0 03	0 01
rando781 gsq	0	0	0	0	0	0	0	0	0
rando801 gsq	0 03	0 03	0 04	0 03	0 04	0 04	0 02	0 03	0 02
rando831 gsq	0 01	0 02	0 02	0 03	0 01	0 02	0 03	0 03	0 01
rando841 gsq	2 94	0 42	0 51	0 43	2 58	0 42	0 37	0 21	0 21
rando851 gsq	0	0	0	0 01	0	0	0	0	0
rando871 gsq	0	0	0	0	0	0	0	0	0
rando881 gsq	0 02	0 01	0 02	0 02	0 02	0 02	0 02	0 02	0 01
rando891 gsq	0 11	0 1	0 1	0 11	0 11	0 09	0 11	0 1	0 09
rando911 gsq	226 84	26 46	26 36	34 58	302 66	104 53	71 78	41 95	26 36
rando921 gsq	15 05	27 77	24 07	18 7	13 93	13 11	14 5	4 79	4 79
rando931 gsq	0 05	0 07	0 07	0 07	0 08	0 07	0 07	0 07	0 05
rando951 gsq	0 05	0 05	0 05	0 06	0 06	0 06	0 05	0 05	0 05
rando981 gsq	1 09	0 52	0 53	0 53	0 58	0 48	0 49	0 56	0 48
rando991 gsq	29 33	2 75	1 05	1 32	3 68	1 3	1 3	2 09	1 05
rand1001 gsq	0 05	0 05	0 05	0 05	0 06	0 06	0 05	0 05	0 05
rand1031 gsq	0 03	0 03	0 03	0 03	0 04	0 04	0 03	0 04	0 03
rand1041 gsq	0 07	0 07	0 08	0 08	0 08	0 07	0 08	0 06	0 06
rand1051 gsq	0 02	0 01	0 01	0 02	0 01	0 02	0 02	0 02	0 01
rand1061 gsq	0 09	0 1	0 1	0 1	0 11	0 1	0 1	0 1	0 09
rand1081 gsq	0 93	0 17	0 24	1 2	0 43	0 78	1 11	0 3	0 17
rand1091 gsq	10 59	3 29	3 85	4 88	13 22	8 55	16	15 74	3 29
rand1101 gsq	0 02	0 02	0 02	0 01	0 01	0 03	0 02	0 02	0 01
rand1111 gsq	0 03	0 02	0 02	0 03	0 04	0 02	0 03	0 02	0 02
rand1151 gsq	1 05	0 12	0 13	1 59	0 59	0 2	0 22	0 08	0 08
rand1161 gsq	0 05	0 04	0 03	0 04	0 04	0 04	0 04	0 04	0 03
rand1171 gsq	0 01	0 01	0 01	0 01	0 02	0 01	0 02	0 03	0 01
rand1181 gsq	0 65	0 17	0 12	0 09	0 3	0 09	0 21	0 08	0 08
rand1191 gsq	0 04	0 03	0 04	0 03	0 03	0 04	0 04	0 04	0 03
rand1201 gsq	0 24	0 08	0 06	0 12	0 18	0 13	0 33	0 11	0 06
rand1211 gsq	0 02	0 02	0 03	0 02	0 02	0 02	0 02	0 03	0 02
rand1231 gsq	0 02	0 01	0 01	0 02	0 01	0 02	0 02	0 01	0 01
rand1241 gsq	0	0	0	0	0	0	0	0	0
rand1251 gsq	0 02	0 02	0 02	0 01	0 03	0 01	0 01	0 02	0 01
rand1261 gsq	0 73	4 42	4 45	1 69	0 77	0 58	0 74	0 67	0 58
rand1271 gsq	0	0	0	0	0	0	0	0	0
rand1281 gsq	0 06	0 12	0 05	0 07	0 05	0 06	0 04	0 06	0 04
rand1291 gsq	0	0	0	0	0	0	0	0	0
rand1301 gsq	0 02	0 02	0 02	0 02	0 03	0 01	0 02	0 01	0 01
rand1341 gsq	0 09	0 14	0 14	0 1	0 09	0 1	0 12	0 07	0 07
rand1351 gsq	0 08	0 08	0 09	0 1	0 09	0 09	0 24	0 09	0 08
rand1371 gsq	0 03	0 03	0 02	0 02	0 02	0 03	0 01	0 02	0 01

Table A 83· MPPI method, time, small trees, 2

FT name	1 scheme	2 scheme	3 scheme	4 scheme	5 scheme	6 scheme	7 scheme	8 scheme	Minimum
rand1381 gsq	0 07	0 06	0 06	0 08	0 07	0 06	0 07	0 08	0 06
rand1391 gsq	0 54	2 73	1 66	0 57	0 48	0 39	15 61	0 4	0 39
rand1411 gsq	0 01	0 01	0 02	0 02	0 02	0 02	0 01	0 02	0 01
rand1421 gsq	67 98	318 25	783 01	4554 33	259 3	358 16	7 13	461 89	7 13
rand1431 gsq	0 09	0 09	0 1	0 07	0 09	0 11	0 08	0 09	0 07
rand1441 gsq	1 65	4 18	3 8	3 43	0 76	0 35	0 54	0 26	0 26
rand1451 gsq	0 05	0 03	0 04	0 04	0 04	0 04	0 03	0 03	0 03
rand1461 gsq	0 02	0 02	0 03	0 01	0 02	0 01	0 02	0 03	0 01
rand1471 gsq	474 99	396 58	167 29	488 02	22 74	36 54	39 83	31 98	22 74
rand1481 gsq	0 03	0 03	0 04	0 04	0 04	0 04	0 05	0 05	0 03
rand1491 gsq	0	0 01	0	0	0	0	0	0	0
rand1511 gsq	0 01	0	0	0	0	0	0	0	0
rand1531 gsq	0 03	0 03	0 04	0 04	0 03	0 04	0 03	0 03	0 03
rand1541 gsq	0	0	0	0	0	0	0	0	0
rand1551 gsq	14 61	28 32	22 93	29 57	11 9	13 32	12 1	7 81	7 81
rand1561 gsq	0	0	0	0	0	0	0	0	0
lisaba11 gsq	0 08	0 08	0 08	0 08	0 08	0 06	0 07	0 07	0 06
lisaba31 gsq	526 2	1635 82	1861 9	792 73	331 53	161 55	698 05	126 89	126 89
lisaba41 gsq	140 54	1522 1	70 88	15 04	14 64	20 51	376 96	16 47	14 64
lisaba51 gsq	0 11	2 23	0 59	0 34	0 17	0 17	0 25	0 16	0 11
lisaba61 gsq	0 28	2 94	2 96	0 14	0 09	0 13	0 1	0 08	0 08
lisaba71 gsq	0 09	0 08	0 09	0 07	0 09	0 08	0 07	0 06	0 06
lisaba91 gsq	0 03	0 04	0 05	0 04	0 03	0 04	0 03	0 05	0 03
lisab101 gsq	3 03	0 76	0 74	0 73	0 57	0 45	0 07	0 41	0 07
lisab111 gsq	0	0	0	0	0	0 01	0	0	0
lisab131 gsq	0 02	0 02	0 02	0 02	0 01	0 01	0 01	0 02	0 01
lisab171 gsq	0 09	0 08	0 09	0 1	0 08	0 07	0 08	0 07	0 07
lisab191 gsq	3 03	3 02	3 05	3 08	3 03	3 05	3 09	3 07	3 02
lisab201 gsq	1 06	1 04	1 07	1 09	1 06	1 06	1 04	1 05	1 04
lisab241 gsq	0 02	0 03	0 04	0 04	0 03	0 02	0 03	0 03	0 02
lisab251 gsq	0 06	0 04	0 04	0 04	0 04	0 05	0 05	0 05	0 04
lisab271 gsq	0 46	0 53	0 56	0 19	0 34	0 17	0 34	0 34	0 17
lisab281 gsq	0	0	0	0	0	0 01	0	0	0
lisab301 gsq	0 07	0 07	0 07	0 08	0 07	0 08	0 07	0 08	0 07
lisab311 gsq	728 01	1805 92	382 98	535 08	60 43	122 33	227 28	31 6	31 6
lisab341 gsq	0 05	0 04	0 04	0 03	0 04	0 03	0 04	0 03	0 03
lisab351 gsq	158 84	44 34	141 74	172 27	42 4	106 05	14 22	11 97	11 97
lisab361 gsq	0 83	1 22	0 96	0 3	0 98	0 44	0 43	0 38	0 3
lisab371 gsq	0 05	0 05	0 05	0 04	0 04	0 05	0 06	0 06	0 04
lisab421 gsq	0	0	0	0	0	0	0	0	0
lisab441 gsq	0 05	0 06	0 06	0 05	0 05	0 06	0 04	0 05	0 04
lisab461 gsq	2 69	2 69	2 7	2 69	2 71	2 71	2 72	2 74	2 69
lisab471 gsq	0 02	0 01	0 01	0 02	0 02	0 02	0 01	0 02	0 01
lisab481 gsq	0 01	0 02	0 03	0 01	0 01	0 02	0 02	0 02	0 01
lisab491 gsq	0 04	0 03	0 03	0 03	0 03	0 03	0 03	0 03	0 03
lisab501 gsq	0	0	0	0	0	0	0	0	0
lisab511 gsq	0 01	0 03	0 02	0 01	0 01	0 01	0 03	0 01	0 01
lisab531 gsq	0	0	0	0	0	0	0	0	0
lisab541 gsq	0 02	0 02	0 02	0 02	0 02	0 03	0 01	0 02	0 01
lisab561 gsq	0 02	0 01	0 01	0 01	0 01	0 01	0 01	0 03	0 01
lisab571 gsq	0 21	2 03	2 89	0 38	0 24	0 73	1 33	0 12	0 12
lisab591 gsq	0 09	0 09	0 11	0 09	0 1	0 09	0 09	0 1	0 09
lisab601 gsq	0 02	0 01	0 01	0 02	0 01	0 01	0 02	0 01	0 01
lisab611 gsq	0 01	0 02	0 03	0 02	0 03	0 03	0 02	0 03	0 01

Table A 84: MPPI method, time, small trees, 3

FT name	1 scheme	2 scheme	3 scheme	4 scheme	5 scheme	6 scheme	7 scheme	8 scheme	Minimum
lisab621 gsq	0 06	0 06	0 05	0 04	0 05	0 06	0 06	0 06	0 04
lisab631 gsq	0 01	0 02	0 02	0 02	0 02	0 02	0 01	0 02	0 01
lisab641 gsq	0 08	0 08	0 08	0 07	0 09	0 09	0 08	0 09	0 07
lisab651 gsq	0	0	0	0	0	0	0	0	0
lisab661 gsq	0 43	0 43	0 43	1 15	0 45	0 58	0 15	0 18	0 15
lisab671 gsq	0 17	0 25	0 25	0 14	0 29	0 2	2 82	0 24	0 14
lisab681 gsq	0	0	0	0	0	0	0	0	0
lisab691 gsq	0 02	0 02	0 03	0 02	0 01	0 03	0 02	0 02	0 01
lisab701 gsq	1	1 02	0 91	0 42	0 64	0 51	0 19	0 55	0 19
lisab711 gsq	0 03	0 03	0 03	0 03	0 03	0 03	0 05	0 04	0 03
lisab721 gsq	60 07	18 8	18 92	9 53	28 45	26 35	209 24	8 85	8 85
lisab731 gsq	0	0	0	0	0	0	0	0	0
lisab751 gsq	0	0	0	0	0	0	0	0	0
lisab771 gsq	10530 99	1100 14	705 04	468 57	2016 72	373 3	1726 54	417 03	373 3
lisab781 gsq	4 37	0 76	1 02	2 64	1 46	2 5	0 93	0 47	0 47
lisab811 gsq	0	0	0	0 01	0	0	0	0	0
lisab821 gsq	484 24	259 32	263 4	94 67	148 23	56 93	118 44	101 64	56 93
lisab831 gsq	0 23	0 21	0 21	0 15	0 22	0 15	1 22	0 16	0 15
lisab841 gsq	0 32	1 05	0 32	0 31	0 23	0 25	2 93	0 16	0 16
lisab861 gsq	36 65	41 1	20 94	2 77	18 08	2 74	25 86	9 19	2 74
lisab881 gsq	0 02	0 02	0 03	0 02	0 02	0 02	0 02	0 02	0 02
lisab911 gsq	0 47	0 2	0 19	0 17	0 35	0 22	0 15	0 26	0 15
lisab951 gsq	0 02	0 01	0 02	0 01	0 01	0 02	0 01	0 01	0 01
lisab971 gsq	0 02	0 01	0 02	0 02	0 02	0 02	0 02	0 02	0 01
lisab981 gsq	0	0	0	0	0	0	0	0	0
lisa1031 gsq	0	0	0	0	0	0	0	0	0
lisa1041 gsq	0	0 01	0	0	0	0	0	0	0
lisa1071 gsq	0 04	0 04	0 05	0 04	0 05	0 04	0 05	0 06	0 04
lisa1091 gsq	0 02	0 03	0 04	0 03	0 03	0 03	0 03	0 04	0 02
lisa1101 gsq	0 04	0 13	0 14	0 11	0 03	0 02	0 13	0 03	0 02
lisa1111 gsq	0 04	0 03	0 04	0 03	0 04	0 04	0 03	0 04	0 03
lisa1121 gsq	50 78	1 27	1 23	3 13	8 54	4 1	4 57	7 42	1 23
lisa1131 gsq	2 84	84 08	67 26	36 75	1 77	0 98	2 52	1 18	0 98
lisa1151 gsq	0 02	0 04	0 03	0 01	0 02	0 02	0 05	0 02	0 01
lisa1161 gsq	0 02	0 01	0 03	0 02	0 02	0 02	0 01	0 02	0 01
lisa1191 gsq	0 02	0 03	0	0 01	0 01	0 02	0 01	0 02	0
lisa1201 gsq	0	0	0 01	0 01	0 01	0	0	0	0
lisa1211 gsq	0 06	0 04	0 06	0 04	0 05	0 05	0 06	0 06	0 04
lisa1221 gsq	0 01	0 01	0 02	0 02	0 01	0 01	0 01	0 02	0 01
lisa1231 gsq	3 58	0 61	0 64	0 23	0 9	1 86	2 24	0 86	0 23
lisa1241 gsq	1590 85	80 21	81 12	128 93	203 05	27 02	1423 51	40 3	27 02
rand1591 gsq	0 1	0 34	0 15	0 13	0 07	0 03	0 04	0 08	0 03
rand1611 gsq	8 73	652 03	525 13	25 03	8 8	9 32	25 01	8 21	8 21
rand1631 gsq	5 55	351 39	59 14	40 04	2 76	2 03	86 81	3 85	2 03
rand1671 gsq	7 31	0 77	0 92	2 38	2 07	0 4	1 4	0 43	0 4

Table A 85. MPPI method, time, small trees, 4

FT name	1 scheme	2 scheme	3 scheme	4 scheme	5 scheme	6 scheme	7 scheme	8 scheme	Minimum
random61 gsq	9603 349	19856 47	24544 7	1188 22	9406 072	1680 092	3573 924	7182 441	1188 22
rando121 gsq	6307 267	1159 585	1149 173	1497 93	4775 987	608 422	2318 144	1074 785	608 422
rando591 gsq	32816 38	1052 616	1640 589	3552 425	33400 18	7464 744	124 278	877 056	124 278
rand1321 gsq	79188 72	-	-	17751 12	2525 467	2945 356	18349 6	11495 92	2525 467
rand1501 gsq	78227 37	-	-	13761 7	-	19625 28	5079 594	19110 59	5079 594
rand1581 gsq	626 312	54602 88	-	82 077	560 351	170 189	81 42	202 027	81 42
lisab521 gsq	11414 73	40816 82	45867 19	6143 983	3145 263	1894 982	43089 68	926 523	926 523
lisab741 gsq	-	53316 42	37397 88	47906 38	-	18312 28	-	-	18312 28
lisab891 gsq	2092 689	19699 25	43847 79	203 23	390 496	205 487	8688 733	365 837	203 23
lisa1001 gsq	-	23340 76	29815 53	-	54022 41	29581 02	-	44256 81	23340 76
rand1641 gsq	3672 304	1153 103	599 34	595 794	874 238	347 473	2647 493	600 481	347 473
rand1651 gsq	-	40980 09	49485 72	15641 21	23734 34	49824 23	18252 61	-	15641 21
rand1661 gsq	-	-	-	-	-	-	-	-	-

Table A 86: MPPI method, time, 'large' trees

FT name	1 scheme	2 scheme	3 scheme	4 scheme	5 scheme	6 scheme	7 scheme	8 scheme	Minimum
astolfo1 gsq	39	33	33	33	35	33	33	38	39
benjam1 gsq	76	76	76	62	76	76	62	98	76
bpfeq031 gsq	1	1	1	1	1	1	1	1	1
bpfen051 gsq	1	1	1	1	1	1	1	1	1
bpfig051 gsq	1	1	1	1	1	1	1	1	1
bpfin051 gsq	1	1	1	1	1	1	1	1	1
bpfsw021 gsq	54	36	42	40	54	36	50	44	54
dre10191 gsq	1	1	1	1	1	1	1	1	1
dre10321 gsq	1	1	1	1	1	1	1	1	1
dre10571 gsq	1	1	1	1	1	1	1	1	1
dre10581 gsq	129	120	120	119	114	120	108	99	129
dre10591 gsq	204	256	252	259	148	121	260	104	204
dresden1 gsq	404	233	233	241	375	258	270	195	404
hpsf021 gsq	146	60	60	76	87	66	70	118	146
hpsf031 gsq	19	19	19	20	19	20	20	19	19
hpsf211 gsq	198	213	240	181	290	238	245	223	198
hpsf361 gsq	22	22	22	22	22	22	22	22	22
jdtree31 gsq	1	1	1	1	1	1	1	1	1
jdtree41 gsq	1	1	1	1	1	1	1	1	1
jdtree51 gsq	1	1	1	1	1	1	1	1	1
khictre1 gsq	40	41	36	36	40	36	39	41	40
nakashi1 gsq	177	72	100	121	163	90	72	152	177
trials11 gsq	143	150	150	150	132	132	143	125	143
trials41 gsq	337	461	354	373	361	371	392	363	337
random31 gsq	125	116	105	105	123	127	108	147	125
random81 gsq	20	20	20	20	20	20	20	20	20
rando131 gsq	122	92	87	87	108	113	134	109	122
rando161 gsq	22	22	22	22	22	22	22	22	22
rando231 gsq	71	62	62	62	69	63	62	70	71
rando251 gsq	1	1	1	1	1	1	1	1	1
rando271 gsq	45	45	45	44	44	41	44	42	45
rando281 gsq	1	1	1	1	1	1	1	1	1
rando291 gsq	293	211	205	277	228	192	194	214	293
rando301 gsq	117	98	94	94	129	101	85	119	117
rando311 gsq	10	10	10	10	10	10	10	10	10
rando331 gsq	85	89	89	85	84	83	56	71	85
rando341 gsq	18	16	16	16	16	16	18	16	18
rando351 gsq	43	43	43	43	39	43	36	37	43
rando361 gsq	1	1	1	1	1	1	1	1	1
rando371 gsq	82	76	77	77	81	80	57	82	82
rando381 gsq	1	1	1	1	1	1	1	1	1
rando391 gsq	114	71	71	93	94	93	142	120	114
rando401 gsq	40	32	36	36	40	40	39	41	40
rando421 gsq	8	8	8	8	8	8	8	8	8
rando431 gsq	27	23	23	24	26	26	24	26	27
rando441 gsq	206	190	181	299	275	300	290	302	206
rando451 gsq	62	67	67	54	52	67	67	52	62
rando461 gsq	14	14	14	14	14	14	14	14	14
rando471 gsq	110	80	101	100	111	107	111	101	110
rando481 gsq	10	10	10	10	10	10	10	10	10
rando521 gsq	112	70	105	105	105	106	76	102	112
rando531 gsq	8	8	8	8	8	8	8	8	8
rando541 gsq	74	46	72	72	76	74	72	80	74
rando551 gsq	36	36	36	36	36	36	36	38	36

Table A 87. ZBDD method, number of nodes, small trees, 1

FT name	1 scheme	2 scheme	3 scheme	4 scheme	5 scheme	6 scheme	7 scheme	8 scheme	Minimum
rando581 gsq	8	8	8	8	8	8	8	8	8
rando601 gsq	35	34	33	34	34	34	33	34	35
rando611 gsq	97	124	93	91	95	86	124	92	97
rando621 gsq	20	20	20	20	20	20	20	20	20
rando631 gsq	28	28	28	28	28	28	28	30	28
rando641 gsq	88	70	70	67	75	67	48	77	88
rando651 gsq	30	33	26	29	26	29	26	24	30
rando661 gsq	19	19	19	19	19	19	19	19	19
rando701 gsq	60	48	50	48	68	48	61	81	60
rando731 gsq	77	72	72	72	77	77	66	77	77
rando751 gsq	1	1	1	1	1	1	1	1	1
rando761 gsq	25	25	25	25	25	25	25	25	25
rando771 gsq	57	26	26	26	42	26	28	32	57
rando781 gsq	1	1	1	1	1	1	1	1	1
rando801 gsq	16	16	16	16	16	16	16	16	16
rando831 gsq	61	42	46	46	53	39	46	54	61
rando841 gsq	202	103	100	123	173	115	86	134	202
rando851 gsq	1	1	1	1	1	1	1	1	1
rando871 gsq	1	1	1	1	1	1	1	1	1
rando881 gsq	41	29	34	34	39	38	30	39	41
rando891 gsq	60	56	59	59	59	59	52	60	60
rando911 gsq	289	279	279	270	289	272	349	272	289
rando921 gsq	214	254	244	243	198	182	238	208	214
rando931 gsq	30	30	30	30	30	30	30	30	30
rando951 gsq	61	52	59	59	64	64	59	60	61
rando981 gsq	254	178	178	196	247	182	167	227	254
rando991 gsq	237	240	209	202	207	175	202	198	237
rand1001 gsq	12	12	12	12	12	12	12	12	12
rand1031 gsq	60	48	58	58	59	59	48	58	60
rand1041 gsq	20	20	20	20	20	20	20	20	20
rand1051 gsq	41	39	36	36	41	36	30	39	41
rand1061 gsq	24	24	24	24	24	24	24	24	24
rand1081 gsq	176	180	163	192	163	172	167	167	176
rand1091 gsq	220	133	153	179	215	206	153	218	220
rand1101 gsq	16	16	16	16	16	16	16	16	16
rand1111 gsq	71	58	58	59	67	59	58	59	71
rand1151 gsq	120	82	82	112	102	93	89	81	120
rand1161 gsq	38	31	31	31	33	33	31	34	38
rand1171 gsq	20	24	24	24	22	22	24	22	20
rand1181 gsq	123	86	80	86	103	89	68	98	123
rand1191 gsq	10	10	10	10	10	10	10	10	10
rand1201 gsq	223	117	120	179	218	192	184	216	223
rand1211 gsq	50	40	40	48	50	44	48	52	50
rand1231 gsq	12	12	12	12	12	12	12	12	12
rand1241 gsq	1	1	1	1	1	1	1	1	1
rand1251 gsq	14	14	14	14	14	14	14	14	14
rand1261 gsq	149	158	158	161	148	140	166	154	149
rand1271 gsq	1	1	1	1	1	1	1	1	1
rand1281 gsq	72	77	61	70	72	70	65	85	72
rand1291 gsq	1	1	1	1	1	1	1	1	1
rand1301 gsq	6	6	6	6	6	6	6	6	6
rand1341 gsq	113	70	70	110	113	111	128	113	113
rand1351 gsq	107	86	90	92	99	98	103	106	107
rand1371 gsq	38	38	38	38	44	44	43	44	38

Table A 88 ZBDD method, number of nodes, small trees, 2

FT name	1 scheme	2 scheme	3 scheme	4 scheme	5 scheme	6 scheme	7 scheme	8 scheme	Minimum
rand1381 gsq	24	24	24	24	24	24	24	24	24
rand1391 gsq	208	196	162	186	180	162	178	194	208
rand1411 gsq	12	12	12	12	12	12	12	12	12
rand1421 gsq	309	299	308	479	315	286	220	337	309
rand1431 gsq	22	22	22	22	22	22	22	22	22
rand1441 gsq	94	72	71	70	114	103	112	96	94
rand1451 gsq	18	18	18	18	18	18	18	18	18
rand1461 gsq	41	41	41	41	48	46	46	44	41
rand1471 gsq	283	352	372	289	232	226	247	262	283
rand1481 gsq	8	8	8	8	8	8	8	8	8
rand1491 gsq	1	1	1	1	1	1	1	1	1
rand1511 gsq	1	1	1	1	1	1	1	1	1
rand1531 gsq	10	10	10	10	10	10	10	10	10
rand1541 gsq	1	1	1	1	1	1	1	1	1
rand1551 gsq	277	222	198	171	239	191	220	212	277
rand1561 gsq	1	1	1	1	1	1	1	1	1
lisaba11 gsq	83	76	80	73	81	75	80	73	83
lisaba31 gsq	564	624	589	562	574	427	582	648	564
lisaba41 gsq	596	434	495	366	416	595	401	395	596
lisaba51 gsq	178	146	150	167	171	160	169	174	178
lisaba61 gsq	158	109	109	115	115	117	126	111	158
lisaba71 gsq	85	80	80	77	83	86	80	83	85
lisaba91 gsq	44	35	35	35	39	35	39	42	44
lisab101 gsq	195	100	109	109	171	139	121	143	195
lisab111 gsq	1	1	1	1	1	1	1	1	1
lisab131 gsq	12	12	12	12	12	12	12	12	12
lisab171 gsq	100	92	96	92	95	96	96	88	100
lisab191 gsq	1	1	1	1	1	1	1	1	1
lisab201 gsq	1	1	1	1	1	1	1	1	1
lisab241 gsq	1	1	1	1	1	1	1	1	1
lisab251 gsq	59	55	59	52	59	62	55	60	59
lisab271 gsq	262	208	208	131	243	157	113	234	262
lisab281 gsq	1	1	1	1	1	1	1	1	1
lisab301 gsq	47	44	44	45	45	45	44	41	47
lisab311 gsq	1378	687	805	472	677	469	696	588	1378
lisab341 gsq	44	37	47	36	44	36	47	45	44
lisab351 gsq	493	504	537	427	433	424	445	346	493
lisab361 gsq	178	190	177	107	164	111	161	217	178
lisab371 gsq	18	18	18	18	18	18	18	18	18
lisab421 gsq	1	1	1	1	1	1	1	1	1
lisab441 gsq	33	34	34	34	33	34	30	33	33
lisab461 gsq	1	1	1	1	1	1	1	1	1
lisab471 gsq	14	12	12	12	14	14	12	14	14
lisab481 gsq	8	8	8	8	8	8	8	8	8
lisab491 gsq	1	1	1	1	1	1	1	1	1
lisab501 gsq	1	1	1	1	1	1	1	1	1
lisab511 gsq	28	28	28	28	28	28	32	31	28
lisab531 gsq	1	1	1	1	1	1	1	1	1
lisab541 gsq	27	22	25	25	27	25	25	24	27
lisab561 gsq	4	4	4	4	4	4	4	4	4
lisab571 gsq	171	169	176	161	171	177	166	141	171
lisab591 gsq	1	1	1	1	1	1	1	1	1
lisab601 gsq	38	30	30	30	38	30	34	38	38
lisab611 gsq	48	47	47	47	55	49	47	49	48

Table A.89 ZBDD method, number of nodes, small trees, 3

FT name	1 scheme	2 scheme	3 scheme	4 scheme	5 scheme	6 scheme	7 scheme	8 scheme	Minimum
lisab621 gsq	36	30	30	33	35	35	33	36	36
lisab631 gsq	15	15	15	15	15	15	14	16	15
lisab641 gsq	56	52	52	52	56	56	52	56	56
lisab651 gsq	1	1	1	1	1	1	1	1	1
lisab661 gsq	183	126	126	180	173	167	124	116	183
lisab671 gsq	110	88	88	89	126	98	120	113	110
lisab681 gsq	1	1	1	1	1	1	1	1	1
lisab691 gsq	48	38	38	39	44	39	45	45	48
lisab701 gsq	93	67	67	60	80	59	53	88	93
lisab711 gsq	10	10	10	10	10	10	10	10	10
lisab721 gsq	284	215	215	218	277	259	149	227	284
lisab731 gsq	1	1	1	1	1	1	1	1	1
lisab751 gsq	1	1	1	1	1	1	1	1	1
lisab771 gsq	427	396	362	270	416	272	331	446	427
lisab781 gsq	300	122	181	215	258	215	166	144	300
lisab811 gsq	1	1	1	1	1	1	1	1	1
lisab821 gsq	580	395	395	653	496	379	534	477	580
lisab831 gsq	107	94	94	91	107	97	95	94	107
lisab841 gsq	94	102	81	81	83	80	88	84	94
lisab861 gsq	223	295	243	210	214	223	215	220	223
lisab881 gsq	71	56	56	56	74	65	56	74	71
lisab911 gsq	276	119	119	107	192	135	80	185	276
lisab951 gsq	6	6	6	6	6	6	6	6	6
lisab971 gsq	6	6	6	6	6	6	6	6	6
lisab981 gsq	1	1	1	1	1	1	1	1	1
lisa1031 gsq	1	1	1	1	1	1	1	1	1
lisa1041 gsq	1	1	1	1	1	1	1	1	1
lisa1071 gsq	16	16	16	16	16	16	16	16	16
lisa1091 gsq	73	79	79	79	76	72	79	76	73
lisa1101 gsq	40	36	36	36	43	39	67	46	40
lisa1111 gsq	30	30	30	30	30	30	32	30	30
lisa1121 gsq	496	220	224	268	473	331	291	541	496
lisa1131 gsq	206	173	173	213	153	142	182	150	206
lisa1151 gsq	47	47	43	43	40	47	45	50	47
lisa1161 gsq	18	17	18	17	18	16	16	17	18
lisa1191 gsq	14	12	14	14	14	14	14	14	14
lisa1201 gsq	1	1	1	1	1	1	1	1	1
lisa1211 gsq	24	24	24	24	24	24	24	24	24
lisa1221 gsq	26	26	26	26	26	26	22	22	26
lisa1231 gsq	89	61	61	56	84	70	103	83	89
lisa1241 gsq	869	431	457	484	705	490	738	401	869
rand1591 gsq	99	97	104	104	87	74	74	89	99
rand1611 gsq	397	288	289	431	325	329	441	535	397
rand1631 gsq	500	936	817	475	369	344	345	488	500
rand1671 gsq	224	157	159	146	217	156	111	171	224

Table A.90 ZBDD method, number of nodes, small trees, 4

FT name	1 scheme	2 scheme	3 scheme	4 scheme	5 scheme	6 scheme	7 scheme	8 scheme	Minimum
random61 gsq	604	449	579	361	601	488	491	567	361
rando121 gsq	835	731	721	558	694	510	476	533	476
rando591 gsq	537	318	329	272	518	324	197	349	197
rand1321 gsq	1449	1249	1249	967	979	1044	1009	1301	967
rand1501 gsq	932	1139	801	577	1046	555	496	959	496
rand1581 gsq	541	258	284	456	556	456	381	504	258
lisab521 gsq	893	1422	1424	945	799	830	712	733	712
lisab741 gsq	2341	714	740	753	4384	973	724	3371	714
lisab891 gsq	753	342	514	362	427	377	501	407	342
lisa1001 gsq	1064	368	363	544	781	640	718	589	363
rand1641 gsq	1436	1598	839	839	863	660	1499	816	660
rand1651 gsq	926	557	557	333	477	300	382	617	300
rand1661 gsq	1121	1157	970	1059	842	1155	1144	998	842

Table A 91: ZBDD method, number of nodes, 'large' trees

FT name	1 scheme	2 scheme	3 scheme	4 scheme	5 scheme	6 scheme	7 scheme	8 scheme	Minimum
astolfo1 gsq	0 04	0 04	0 04	0 04	0 04	0 05	0 04	0 04	0 04
benjam1 gsq	0 04	0 05	0 05	0 05	0 05	0 05	0 05	0 05	0 04
bpfeg031 gsq	0 53	0 48	0 54	0 48	0 53	0 53	0 51	0 53	0 48
bpfen051 gsq	0 37	0 37	0 37	0 37	0 38	0 37	0 37	0 38	0 37
bpfig051 gsq	0 33	0 32	0 39	0 33	0 33	0 33	0 37	0 39	0 32
bpfin051 gsq	0 02	0 01	0 01	0 01	0 02	0 01	0 01	0 02	0 01
bpfs021 gsq	1 51	1 48	1 52	1 5	1 61	1 49	1 58	1 47	1 47
dre10191 gsq	0	0	0	0	0 01	0 01	0 01	0	0
dre10321 gsq	0	0 01	0 01	0 01	0	0 01	0 01	0	0
dre10571 gsq	0 03	0 02	0 03	0 03	0 03	0 02	0 03	0 03	0 02
dre10581 gsq	0 33	0 39	0 36	0 33	0 37	0 35	0 39	0 39	0 33
dre10591 gsq	0 66	0 67	0 73	0 66	0 68	0 67	0 68	0 67	0 66
dresden1 gsq	0 62	0 66	0 65	0 62	0 64	0 67	0 66	0 64	0 62
hpfis021 gsq	0 04	0 05	0 03	0 04	0 05	0 05	0 04	0 04	0 03
hpfis031 gsq	0 04	0 04	0 04	0 04	0 05	0 04	0 04	0 05	0 04
hpfis211 gsq	0 04	0 08	0 05	0 05	0 07	0 07	0 07	0 06	0 04
hpfis361 gsq	0 04	0 03	0 04	0 03	0 04	0 05	0 05	0 04	0 03
dtree31 gsq	0 01	0	0 01	0	0	0 01	0	0 01	0
dtree41 gsq	0	0	0	0	0	0	0 01	0 01	0
dtree51 gsq	0	0	0 01	0	0	0	0	0	0
khitre1 gsq	0 23	0 26	0 24	0 23	0 25	0 25	0 24	0 25	0 23
nakashi1 gsq	0 04	0 04	0 05	0 04	0 05	0 05	0 04	0 06	0 04
trials11 gsq	0 04	0 04	0 04	0 04	0 05	0 05	0 03	0 05	0 03
trials41 gsq	0 06	0 22	0 15	0 16	0 05	0 15	0 17	0 06	0 05
random31 gsq	0 05	0 04	0 04	0 05	0 05	0 05	0 05	0 05	0 04
random81 gsq	0 24	0 23	0 24	0 22	0 25	0 25	0 24	0 24	0 22
rando131 gsq	0 04	0 04	0 05	0 04	0 04	0 05	0 05	0 03	0 03
rando161 gsq	0 23	0 23	0 24	0 22	0 24	0 26	0 24	0 24	0 22
rando231 gsq	0 04	0 03	0 04	0 04	0 05	0 05	0 04	0 04	0 03
rando251 gsq	0	0	0	0 01	0 01	0	0	0	0
rando271 gsq	0 19	0 2	0 2	0 19	0 2	0 21	0 2	0 22	0 19
rando281 gsq	0	0	0	0	0	0	0 01	0	0
rando291 gsq	0 05	0 05	0 05	0 04	0 05	0 05	0 05	0 03	0 03
rando301 gsq	0 04	0 04	0 04	0 04	0 04	0 05	0 05	0 04	0 04
rando311 gsq	0 04	0 05	0 04	0 05	0 05	0 05	0 04	0 04	0 04
rando331 gsq	0 19	0 2	0 2	0 19	0 21	0 2	0 2	0 21	0 19
rando341 gsq	0 04	0 04	0 03	0 03	0 04	0 04	0 04	0 03	0 03
rando351 gsq	0 31	0 31	0 35	0 31	0 34	0 34	0 32	0 31	0 31
rando361 gsq	0 01	0	0	0	0	0 01	0	0	0
rando371 gsq	0 04	0 03	0 04	0 04	0 04	0 04	0 04	0 04	0 03
rando381 gsq	0	0	0 01	0	0 01	0	0 01	0	0
rando391 gsq	0 04	0 04	0 04	0 04	0 04	0 05	0 03	0 03	0 03
rando401 gsq	0 19	0 2	0 21	0 19	0 2	0 2	0 2	0 2	0 19
rando421 gsq	0 19	0 19	0 21	0 18	0 2	0 21	0 2	0 22	0 18
rando431 gsq	0 04	0 04	0 04	0 04	0 04	0 03	0 04	0 05	0 03
rando441 gsq	0 05	0 04	0 05	0 06	0 06	0 05	0 05	0 05	0 04
rando451 gsq	0 19	0 2	0 2	0 25	0 22	0 22	0 2	0 2	0 19
rando461 gsq	0 23	0 24	0 25	0 25	0 25	0 26	0 24	0 25	0 23
rando471 gsq	0 3	0 33	0 34	0 35	0 33	0 34	0 34	0 33	0 3
rando481 gsq	0 19	0 2	0 19	0 21	0 21	0 2	0 2	0 21	0 19
rando521 gsq	0 23	0 23	0 23	0 26	0 24	0 24	0 23	0 24	0 23
rando531 gsq	0 19	0 21	0 2	0 21	0 21	0 19	0 2	0 2	0 19
rando541 gsq	0 23	0 23	0 24	0 27	0 25	0 24	0 24	0 24	0 23
rando551 gsq	0 23	0 23	0 23	0 24	0 25	0 22	0 24	0 24	0 22

Table A 92: ZBDD method, time, small trees, 1

FT name	1 scheme	2 scheme	3 scheme	4 scheme	5 scheme	6 scheme	7 scheme	8 scheme	Minimum
rando581 gsq	0 2	0 2	0 2	0 2	0 21	0 19	0 2	0 21	0 19
rando601 gsq	0 27	0 27	0 28	0 27	0 3	0 28	0 28	0 28	0 27
rando611 gsq	0 04	0 06	0 04	0 04	0 04	0 05	0 04	0 04	0 04
rando621 gsq	0 26	0 28	0 27	0 27	0 3	0 28	0 27	0 27	0 26
rando631 gsq	0 22	0 23	0 24	0 23	0 25	0 24	0 24	0 24	0 22
rando641 gsq	0 22	0 24	0 25	0 23	0 25	0 24	0 24	0 24	0 22
rando651 gsq	0 04	0 04	0 04	0 04	0 05	0 04	0 03	0 03	0 03
rando661 gsq	0 19	0 19	0 2	0 19	0 21	0 2	0 21	0 2	0 19
rando701 gsq	0 05	0 04	0 04	0 04	0 04	0 04	0 04	0 04	0 04
rando731 gsq	0 24	0 24	0 25	0 23	0 24	0 24	0 25	0 24	0 23
rando751 gsq	0	0 01	0	0	0	0	0	0	0
rando761 gsq	0 22	0 24	0 24	0 24	0 25	0 25	0 24	0 24	0 22
rando771 gsq	0 04	0 03	0 04	0 03	0 04	0 04	0 03	0 04	0 03
rando781 gsq	0	0	0	0 01	0	0	0 01	0	0
rando801 gsq	0 2	0 2	0 21	0 19	0 22	0 21	0 2	0 2	0 19
rando831 gsq	0 03	0 05	0 04	0 04	0 04	0 04	0 04	0 04	0 03
rando841 gsq	0 19	0 19	0 21	0 2	0 21	0 2	0 2	0 2	0 19
rando851 gsq	0	0	0	0	0	0	0	0 01	0
rando871 gsq	0	0 01	0	0	0	0	0	0 01	0
rando881 gsq	0 04	0 04	0 03	0 04	0 05	0 04	0 05	0 05	0 03
rando891 gsq	0 34	0 34	0 36	0 35	0 38	0 36	0 35	0 37	0 34
rando911 gsq	0 07	0 06	0 07	0 05	0 06	0 06	0 06	0 06	0 05
rando921 gsq	0 21	0 2	0 21	0 19	0 21	0 21	0 21	0 2	0 19
rando931 gsq	0 26	0 26	0 27	0 25	0 3	0 29	0 28	0 29	0 25
rando951 gsq	0 23	0 23	0 24	0 23	0 25	0 24	0 24	0 24	0 23
rando981 gsq	0 05	0 04	0 05	0 04	0 06	0 05	0 03	0 05	0 03
rando991 gsq	0 04	0 05	0 05	0 04	0 05	0 04	0 05	0 04	0 04
rand1001 gsq	0 23	0 23	0 24	0 23	0 24	0 24	0 24	0 24	0 23
rand1031 gsq	0 18	0 2	0 2	0 21	0 22	0 21	0 2	0 2	0 18
rand1041 gsq	0 26	0 27	0 27	0 26	0 28	0 28	0 28	0 28	0 26
rand1051 gsq	0 04	0 03	0 04	0 04	0 05	0 05	0 03	0 05	0 03
rand1061 gsq	0 34	0 35	0 37	0 34	0 38	0 35	0 35	0 37	0 34
rand1081 gsq	0 04	0 04	0 05	0 04	0 06	0 04	0 04	0 04	0 04
rand1091 gsq	0 04	0 03	0 04	0 05	0 06	0 04	0 05	0 05	0 03
rand1101 gsq	0 04	0 05	0 03	0 04	0 05	0 05	0 05	0 04	0 03
rand1111 gsq	0 05	0 03	0 04	0 05	0 05	0 05	0 04	0 04	0 03
rand1151 gsq	0 03	0 03	0 03	0 03	0 04	0 05	0 04	0 04	0 03
rand1161 gsq	0 19	0 2	0 2	0 18	0 2	0 24	0 2	0 2	0 18
rand1171 gsq	0 03	0 03	0 04	0 04	0 04	0 03	0 04	0 05	0 03
rand1181 gsq	0 2	0 2	0 21	0 19	0 21	0 2	0 2	0 23	0 19
rand1191 gsq	0 2	0 2	0 19	0 2	0 21	0 2	0 19	0 21	0 19
rand1201 gsq	0 05	0 03	0 04	0 04	0 05	0 04	0 04	0 05	0 03
rand1211 gsq	0 03	0 04	0 04	0 04	0 04	0 04	0 04	0 04	0 03
rand1231 gsq	0 03	0 04	0 04	0 04	0 04	0 04	0 04	0 04	0 03
rand1241 gsq	0	0 01	0 01	0 01	0	0	0 01	0	0
rand1251 gsq	0 02	0 04	0 04	0 04	0 05	0 05	0 04	0 05	0 02
rand1261 gsq	0 05	0 05	0 05	0 04	0 06	0 04	0 04	0 05	0 04
rand1271 gsq	0 01	0	0 01	0	0	0	0	0	0
rand1281 gsq	0 04	0 04	0 04	0 04	0 04	0 05	0 04	0 05	0 04
rand1291 gsq	0	0	0 01	0 01	0	0 01	0	0	0
rand1301 gsq	0 04	0 03	0 03	0 04	0 04	0 03	0 03	0 03	0 03
rand1341 gsq	0 03	0 05	0 04	0 04	0 05	0 04	0 04	0 05	0 03
rand1351 gsq	0 2	0 2	0 21	0 2	0 21	0 2	0 21	0 25	0 2
rand1371 gsq	0 05	0 04	0 04	0 04	0 04	0 04	0 04	0 04	0 04

Table A 93: ZBDD method, time, small trees, 2

FT name	1 scheme	2 scheme	3 scheme	4 scheme	5 scheme	6 scheme	7 scheme	8 scheme	Minimum
rand1381 gsq	0 27	0 27	0 27	0 26	0 29	0 27	0 27	0 28	0 26
rand1391 gsq	0 04	0 03	0 04	0 04	0 04	0 05	0 04	0 05	0 03
rand1411 gsq	0 04	0 03	0 04	0 04	0 05	0 03	0 05	0 05	0 03
rand1421 gsq	0 2	0 21	0 2	0 21	0 22	0 2	0 23	0 22	0 2
rand1431 gsq	0 29	0 31	0 3	0 3	0 33	0 32	0 31	0 32	0 29
rand1441 gsq	0 04	0 04	0 04	0 04	0 05	0 04	0 04	0 04	0 04
rand1451 gsq	0 19	0 2	0 21	0 19	0 22	0 2	0 21	0 2	0 19
rand1461 gsq	0 04	0 04	0 04	0 03	0 05	0 04	0 04	0 04	0 03
rand1471 gsq	0 2	0 21	0 21	0 21	0 21	0 21	0 22	0 21	0 2
rand1481 gsq	0 19	0 19	0 2	0 19	0 21	0 21	0 2	0 2	0 19
rand1491 gsq	0 01	0 01	0	0	0	0	0	0	0
rand1511 gsq	0 01	0	0	0 01	0 01	0	0	0 01	0
rand1531 gsq	0 19	0 19	0 2	0 21	0 21	0 19	0 19	0 2	0 19
rand1541 gsq	0	0	0 01	0	0	0	0	0 01	0
rand1551 gsq	0 03	0 04	0 04	0 05	0 04	0 04	0 05	0 05	0 03
rand1561 gsq	0 01	0 01	0	0	0	0	0 01	0 01	0
lisaba11 gsq	0 27	0 28	0 28	0 27	0 3	0 28	0 28	0 28	0 27
lisaba31 gsq	0 22	0 24	0 23	0 24	0 24	0 22	0 24	0 24	0 22
lisaba41 gsq	0 06	0 06	0 05	0 05	0 05	0 06	0 07	0 05	0 05
lisaba51 gsq	0 23	0 24	0 24	0 24	0 26	0 24	0 24	0 25	0 23
lisaba61 gsq	0 07	0 05	0 06	0 06	0 07	0 06	0 05	0 07	0 05
lisaba71 gsq	0 27	0 27	0 28	0 28	0 29	0 27	0 28	0 29	0 27
lisaba91 gsq	0 21	0 19	0 23	0 2	0 21	0 21	0 2	0 21	0 19
lisab101 gsq	0 05	0 05	0 05	0 03	0 06	0 03	0 04	0 05	0 03
lisab111 gsq	0	0 01	0	0 01	0	0	0	0	0
lisab131 gsq	0 03	0 03	0 04	0 04	0 05	0 03	0 04	0 04	0 03
lisab171 gsq	0 26	0 27	0 27	0 27	0 3	0 28	0 27	0 28	0 26
lisab191 gsq	3	3 01	3 03	3 01	3 11	3 03	3 15	3 04	3
lisab201 gsq	1 06	1 05	1 05	1 05	1 09	1 06	1 06	1 06	1 05
lisab241 gsq	0 03	0 03	0 03	0 04	0 03	0 04	0 03	0 03	0 03
lisab251 gsq	0 04	0 03	0 04	0 04	0 04	0 05	0 04	0 04	0 03
lisab271 gsq	0 32	0 32	0 31	0 3	0 35	0 32	0 31	0 33	0 3
lisab281 gsq	0 01	0 01	0	0 01	0	0 01	0	0 01	0
lisab301 gsq	0 27	0 27	0 27	0 27	0 29	0 27	0 27	0 28	0 27
lisab311 gsq	0 38	0 29	0 22	0 06	0 06	0 05	0 25	0 06	0 05
lisab341 gsq	0 19	0 21	0 2	0 2	0 22	0 19	0 21	0 21	0 19
lisab351 gsq	0 05	0 05	0 05	0 04	0 05	0 04	0 06	0 04	0 04
lisab361 gsq	0 04	0 04	0 04	0 04	0 05	0 03	0 04	0 04	0 03
lisab371 gsq	0 24	0 23	0 23	0 23	0 23	0 23	0 24	0 24	0 23
lisab421 gsq	0	0	0	0	0	0	0	0 01	0
lisab441 gsq	0 23	0 23	0 23	0 23	0 25	0 24	0 24	0 23	0 23
lisab461 gsq	2 68	2 67	2 68	2 68	2 73	2 7	2 7	2 7	2 67
lisab471 gsq	0 04	0 05	0 04	0 04	0 05	0 04	0 05	0 04	0 04
lisab481 gsq	0 04	0 03	0 04	0 03	0 05	0 04	0 04	0 05	0 03
lisab491 gsq	0 04	0 04	0 05	0 04	0 04	0 03	0 03	0 04	0 03
lisab501 gsq	0	0	0	0	0	0 01	0	0	0
lisab511 gsq	0 04	0 04	0 03	0 04	0 05	0 04	0 04	0 04	0 03
lisab531 gsq	0 01	0	0 01	0	0 01	0 01	0 01	0	0
lisab541 gsq	0 04	0 04	0 04	0 04	0 04	0 04	0 04	0 04	0 04
lisab561 gsq	0 03	0 04	0 04	0 03	0 04	0 05	0 04	0 04	0 03
lisab571 gsq	0 03	0 04	0 04	0 03	0 05	0 04	0 05	0 05	0 03
lisab591 gsq	0 09	0 09	0 1	0 1	0 1	0 09	0 09	0 09	0 08
lisab601 gsq	0 03	0 04	0 04	0 05	0 05	0 04	0 04	0 04	0 03
lisab611 gsq	0 05	0 04	0 03	0 05	0 05	0 04	0 02	0 04	0 02

Table A 94: ZBDD method, time, small trees, 3

FT name	1 scheme	2 scheme	3 scheme	4 scheme	5 scheme	6 scheme	7 scheme	8 scheme	Minimum
lsab621 gsq	0 24	0 24	0 24	0 23	0 25	0 24	0 23	0 24	0 23
lsab631 gsq	0 05	0 04	0 05	0 04	0 05	0 03	0 03	0 04	0 03
lsab641 gsq	0 31	0 31	0 3	0 29	0 33	0 31	0 3	0 31	0 29
lsab651 gsq	0 01	0	0	0	0 01	0 01	0	0	0
lsab661 gsq	0 04	0 05	0 03	0 03	0 05	0 04	0 04	0 04	0 03
lsab671 gsq	0 09	0 09	0 09	0 09	0 1	0 09	0 09	0 1	0 09
lsab681 gsq	0	0 01	0	0 01	0	0	0	0	0
lsab691 gsq	0 03	0 03	0 05	0 03	0 05	0 04	0 04	0 05	0 03
lsab701 gsq	0 04	0 04	0 05	0 04	0 05	0 04	0 04	0 04	0 04
lsab711 gsq	0 19	0 19	0 2	0 21	0 21	0 19	0 19	0 21	0 19
lsab721 gsq	0 04	0 04	0 05	0 04	0 05	0 05	0 04	0 03	0 03
lsab731 gsq	0	0	0	0	0	0 01	0	0	0
lsab751 gsq	0 01	0	0	0	0	0	0	0	0
lsab771 gsq	0 24	0 23	0 24	0 21	0 23	0 22	0 22	0 22	0 21
lsab781 gsq	0 03	0 03	0 04	0 04	0 03	0 04	0 04	0 04	0 03
lsab811 gsq	0	0	0	0	0	0	0 01	0	0
lsab821 gsq	1 91	1 99	2 01	1 96	1 92	1 84	1 94	1 9	1 84
lsab831 gsq	0 04	0 03	0 04	0 03	0 03	0 04	0 04	0 04	0 03
lsab841 gsq	0 05	0 04	0 04	0 05	0 04	0 04	0 04	0 04	0 04
lsab861 gsq	0 04	0 05	0 04	0 06	0 05	0 04	0 03	0 04	0 03
lsab881 gsq	0 03	0 04	0 05	0 04	0 04	0 03	0 03	0 04	0 03
lsab911 gsq	0 14	0 16	0 14	0 14	0 14	0 14	0 14	0 14	0 14
lsab951 gsq	0 04	0 04	0 02	0 03	0 04	0 04	0 04	0 05	0 02
lsab971 gsq	0 04	0 03	0 05	0 03	0 03	0 04	0 03	0 04	0 03
lsab981 gsq	0	0	0	0	0	0	0	0 01	0
lsa1031 gsq	0 01	0	0	0	0	0	0	0	0
lsa1041 gsq	0 01	0	0	0	0	0 01	0	0 01	0
lsa1071 gsq	0 23	0 23	0 23	0 23	0 22	0 22	0 23	0 23	0 22
lsa1091 gsq	0 04	0 03	0 04	0 04	0 03	0 04	0 05	0 04	0 03
lsa1101 gsq	0 05	0 04	0 03	0 04	0 04	0 05	0 05	0 03	0 03
lsa1111 gsq	0 19	0 2	0 19	0 2	0 2	0 2	0 2	0 2	0 19
lsa1121 gsq	0 25	0 25	0 25	0 24	0 24	0 24	0 25	0 24	0 24
lsa1131 gsq	0 19	0 2	0 19	0 2	0 2	0 2	0 21	0 19	0 19
lsa1151 gsq	0 04	0 04	0 04	0 04	0 04	0 04	0 04	0 04	0 04
lsa1161 gsq	0 05	0 05	0 03	0 04	0 04	0 03	0 04	0 04	0 03
lsa1191 gsq	0 04	0 03	0 04	0 04	0 03	0 04	0 04	0 03	0 03
lsa1201 gsq	0	0	0 01	0	0 01	0 01	0	0	0
lsa1211 gsq	0 24	0 22	0 24	0 22	0 22	0 23	0 23	0 23	0 22
lsa1221 gsq	0 03	0 03	0 04	0 03	0 04	0 04	0 03	0 03	0 03
lsa1231 gsq	0 04	0 03	0 04	0 05	0 04	0 03	0 03	0 04	0 03
lsa1241 gsq	0 1	0 08	0 08	0 07	0 08	0 07	0 13	0 08	0 07
rand1591 gsq	0 04	0 04	0 04	0 04	0 03	0 04	0 04	0 04	0 03
rand1611 gsq	0 04	0 13	0 09	0 05	0 05	0 04	0 05	0 05	0 04
rand1631 gsq	0 22	0 21	0 23	0 22	0 2	0 2	0 22	0 2	0 2
rand1671 gsq	0 2	0 2	0 21	0 22	0 19	0 2	0 2	0 21	0 19

Table A 95: ZBDD method, time, small trees, 4

FT name	1 scheme	2 scheme	3 scheme	4 scheme	5 scheme	6 scheme	7 scheme	8 scheme	Minimum
random61 gsq	0 09	0 07	0 07	0 08	0 11	0 11	0 12	0 09	0 07
rando121 gsq	0 16	0 18	0 14	0 14	0 11	0 11	0 16	0 11	0 11
rando591 gsq	0 05	0 04	0 05	0 04	0 04	0 05	0 05	0 06	0 04
rand1321 gsq	1 08	3 23	3 15	0 57	0 22	0 27	0 48	0 38	0 22
rand1501 gsq	0 07	0 13	0 09	0 06	0 08	0 05	0 05	0 06	0 05
rand1581 gsq	0 27	0 26	0 25	0 25	0 28	0 25	0 25	0 26	0 25
lisab521 gsq	0 1	1 06	0 94	0 13	0 06	0 12	0 14	0 07	0 06
lisab741 gsq	6 95	6 31	6 6	6 39	8 73	6 49	6 75	12 96	6 31
lisab891 gsq	0 07	0 13	0 19	0 05	0 08	0 06	0 2	0 06	0 05
lisa1001 gsq	0 15	0 08	0 07	0 16	0 1	0 1	0 24	0 07	0 07
rand1641 gsq	0 52	0 39	0 22	0 22	0 24	0 2	1 21	0 2	0 2
rand1651 gsq	0 28	0 29	0 29	0 26	0 29	0 26	0 27	0 26	0 26
rand1661 gsq	0 17	3 27	1 83	0 43	0 14	0 47	0 85	0 2	0 14

Table A 96 ZBDD method, time, 'large' trees

FT name	1 scheme	2 scheme	3 scheme	4 scheme	5 scheme	6 scheme	7 scheme	8 scheme	Minimum
astolfo1 gsq	53	56	56	57	51	57	57	51	51
benjam1 gsq	129	140	140	117	129	129	117	158	117
bpfig031 gsq	1	1	1	1	1	1	1	1	1
bpfen051 gsq	1	1	1	1	1	1	1	1	1
bpfig051 gsq	1	1	1	1	1	1	1	1	1
bpfin051 gsq	1	1	1	1	1	1	1	1	1
bpfs021 gsq	101	67	98	79	101	67	111	75	67
dre10191 gsq	1	1	1	1	1	1	1	1	1
dre10321 gsq	1	1	1	1	1	1	1	1	1
dre10571 gsq	1	1	1	1	1	1	1	1	1
dre10581 gsq	249	256	256	258	235	242	265	214	214
dre10591 gsq	486	662	563	547	402	352	659	360	352
dresden1 gsq	3154	2374	2374	2368	2722	2424	7602	2070	2070
hpfis021 gsq	480	107	107	120	146	100	127	257	100
hpfis031 gsq	15	15	15	19	15	19	19	15	15
hpfis211 gsq	1139	2069	1143	945	1160	1386	2076	868	868
hpfis361 gsq	20	20	20	20	20	20	20	20	20
jdree31 gsq	1	1	1	1	1	1	1	1	1
jdree41 gsq	1	1	1	1	1	1	1	1	1
jdree51 gsq	1	1	1	1	1	1	1	1	1
khicre1 gsq	35	38	34	34	35	34	36	38	34
nakashi1 gsq	299	140	156	172	216	213	140	208	140
trials11 gsq	215	295	295	295	217	216	332	183	183
trials41 gsq	836	1490	1421	1040	706	1132	1659	808	706
random31 gsq	193	136	129	122	158	153	418	198	122
random81 gsq	16	16	16	16	16	16	16	16	16
rando131 gsq	145	183	163	163	116	128	202	119	116
rando161 gsq	20	20	20	20	20	20	20	20	20
rando231 gsq	63	58	58	58	61	55	58	65	55
rando251 gsq	1	1	1	1	1	1	1	1	1
rando271 gsq	38	38	38	37	38	36	37	37	36
rando281 gsq	1	1	1	1	1	1	1	1	1
rando291 gsq	913	727	713	800	906	555	773	414	414
rando301 gsq	240	234	216	216	257	95	126	244	95
rando311 gsq	5	5	5	5	5	5	5	5	5
rando331 gsq	159	173	173	186	163	178	88	130	88
rando341 gsq	20	15	15	18	13	13	15	13	13
rando351 gsq	42	42	42	42	35	42	31	33	31
rando361 gsq	1	1	1	1	1	1	1	1	1
rando371 gsq	83	76	83	83	77	76	76	78	76
rando381 gsq	1	1	1	1	1	1	1	1	1
rando391 gsq	178	120	120	157	153	157	352	212	120
rando401 gsq	40	46	36	36	40	40	38	40	36
rando421 gsq	6	6	6	6	6	6	6	6	6
rando431 gsq	23	23	23	31	30	30	31	30	23
rando441 gsq	612	522	518	372	907	467	455	486	372
rando451 gsq	61	67	67	69	52	67	67	52	52
rando461 gsq	12	12	12	12	12	12	12	12	12
rando471 gsq	141	159	157	156	118	106	176	110	106
rando481 gsq	8	8	8	8	8	8	8	8	8
rando521 gsq	183	165	175	175	173	170	89	176	89
rando531 gsq	6	6	6	6	6	6	6	6	6
rando541 gsq	65	52	64	64	72	70	64	73	52
rando551 gsq	28	28	28	28	28	28	28	28	28

Table A 97. L-BDD method, number of nodes, small trees, 1

FT name	1 scheme	2 scheme	3 scheme	4 scheme	5 scheme	6 scheme	7 scheme	8 scheme	Minimum
rando581 gsq	6	6	6	6	6	6	6	6	6
rando601 gsq	33	29	32	29	28	28	32	28	28
rando611 gsq	120	224	149	136	115	65	224	114	65
rando621 gsq	18	18	18	18	18	18	18	18	18
rando631 gsq	23	23	23	23	23	23	23	23	23
rando641 gsq	88	72	72	70	76	70	100	80	70
rando651 gsq	34	42	33	42	34	42	33	31	31
rando661 gsq	23	23	23	23	23	23	23	23	23
rando701 gsq	55	43	35	43	65	43	49	70	35
rando731 gsq	97	77	77	78	97	97	89	97	77
rando751 gsq	1	1	1	1	1	1	1	1	1
rando761 gsq	22	22	22	22	22	22	22	22	22
rando771 gsq	70	15	15	15	42	15	28	26	15
rando781 gsq	1	1	1	1	1	1	1	1	1
rando801 gsq	13	13	13	13	13	13	14	14	13
rando831 gsq	67	36	49	49	51	36	47	49	36
rando841 gsq	461	223	222	238	378	336	350	238	222
rando851 gsq	1	1	1	1	1	1	1	1	1
rando871 gsq	1	1	1	1	1	1	1	1	1
rando881 gsq	41	35	44	44	34	33	30	34	30
rando891 gsq	62	54	61	61	61	61	57	62	54
rando911 gsq	2367	1376	1376	477	824	466	1504	518	466
rando921 gsq	264	360	344	343	226	244	627	222	222
rando931 gsq	28	28	28	28	28	28	28	28	28
rando951 gsq	62	51	60	60	52	52	60	52	51
rando981 gsq	561	343	343	330	490	316	310	438	310
rando991 gsq	488	712	649	599	435	430	599	377	377
rand1001 gsq	10	10	10	10	10	10	10	10	10
rand1031 gsq	113	69	112	112	111	111	69	104	69
rand1041 gsq	17	17	17	17	17	17	17	17	17
rand1051 gsq	39	39	33	33	38	33	36	30	30
rand1061 gsq	22	22	22	22	22	22	22	22	22
rand1081 gsq	271	300	265	319	252	262	272	253	252
rand1091 gsq	342	202	226	288	299	309	281	283	202
rand1101 gsq	14	14	14	11	11	11	11	11	11
rand1111 gsq	102	81	81	73	96	77	53	62	53
rand1151 gsq	186	112	112	176	128	135	127	83	83
rand1161 gsq	36	27	30	30	29	31	30	29	27
rand1171 gsq	15	19	18	18	17	16	18	17	15
rand1181 gsq	163	101	97	105	132	105	86	107	86
rand1191 gsq	8	8	8	8	8	8	8	8	8
rand1201 gsq	240	289	278	196	227	196	451	221	196
rand1211 gsq	39	51	51	37	39	34	37	42	34
rand1231 gsq	6	6	6	6	8	6	6	9	6
rand1241 gsq	1	1	1	1	1	1	1	1	1
rand1251 gsq	11	11	12	12	12	12	11	11	11
rand1261 gsq	267	279	279	251	271	261	280	276	251
rand1271 gsq	1	1	1	1	1	1	1	1	1
rand1281 gsq	77	116	76	86	72	75	84	80	72
rand1291 gsq	1	1	1	1	1	1	1	1	1
rand1301 gsq	3	3	3	3	3	3	3	3	3
rand1341 gsq	217	210	210	266	217	227	272	220	210
rand1351 gsq	124	135	108	106	116	110	240	127	106
rand1371 gsq	38	38	38	38	43	43	31	41	31

Table A.98 L-BDD method, number of nodes, small trees, 2

FT name	1 scheme	2 scheme	3 scheme	4 scheme	5 scheme	6 scheme	7 scheme	8 scheme	Minimum
rand1381 gsq	20	20	20	20	20	20	20	20	20
rand1391 gsq	299	369	317	376	277	290	423	276	276
rand1411 gsq	10	10	10	10	10	10	9	9	9
rand1421 gsq	1000	1064	1138	3185	1090	1327	619	1250	619
rand1431 gsq	19	19	19	19	19	19	19	19	19
rand1441 gsq	208	222	212	229	194	148	143	151	143
rand1451 gsq	15	15	15	15	15	15	15	14	14
rand1461 gsq	44	44	44	44	47	39	39	42	39
rand1471 gsq	685	767	762	697	477	489	548	529	477
rand1481 gsq	6	6	6	6	6	6	6	6	6
rand1491 gsq	1	1	1	1	1	1	1	1	1
rand1511 gsq	1	1	1	1	1	1	1	1	1
rand1531 gsq	7	7	7	7	7	7	7	7	7
rand1541 gsq	1	1	1	1	1	1	1	1	1
rand1551 gsq	344	578	626	395	287	220	666	257	220
rand1561 gsq	1	1	1	1	1	1	1	1	1
lsaba11 gsq	79	77	93	72	73	68	93	66	66
lsaba31 gsq	1704	2638	2628	2755	1752	1585	3383	1735	1585
lsaba41 gsq	2223	2581	2288	2230	1860	2167	4228	1908	1860
lsaba51 gsq	422	314	249	223	326	185	511	303	185
lsaba61 gsq	241	390	390	146	170	142	231	163	142
lsaba71 gsq	86	85	85	80	85	83	85	89	80
lsaba91 gsq	43	60	60	33	36	33	65	39	33
lsab101 gsq	444	291	279	279	333	294	287	305	279
lsab111 gsq	1	1	1	1	1	1	1	1	1
lsab131 gsq	10	10	10	10	10	10	9	9	9
lsab171 gsq	140	165	181	165	132	140	181	113	113
lsab191 gsq	1	1	1	1	1	1	1	1	1
lsab201 gsq	1	1	1	1	1	1	1	1	1
lsab241 gsq	1	1	1	1	1	1	1	1	1
lsab251 gsq	63	52	58	60	63	66	52	67	52
lsab271 gsq	563	529	529	383	323	388	361	291	291
lsab281 gsq	1	1	1	1	1	1	1	1	1
lsab301 gsq	47	43	43	44	43	44	42	37	37
lsab311 gsq	4511	7640	6746	4080	2598	3435	5544	2600	2598
lsab341 gsq	45	39	49	37	45	37	49	43	37
lsab351 gsq	732	862	965	649	614	563	767	541	541
lsab361 gsq	248	286	291	160	235	166	236	260	160
lsab371 gsq	14	14	14	14	14	14	14	15	14
lsab421 gsq	1	1	1	1	1	1	1	1	1
lsab441 gsq	32	31	31	31	32	31	28	28	28
lsab461 gsq	1	1	1	1	1	1	1	1	1
lsab471 gsq	10	9	9	9	10	10	9	10	9
lsab481 gsq	4	4	4	4	4	4	4	4	4
lsab491 gsq	1	1	1	1	1	1	1	1	1
lsab501 gsq	1	1	1	1	1	1	1	1	1
lsab511 gsq	36	36	36	36	36	36	41	36	36
lsab531 gsq	1	1	1	1	1	1	1	1	1
lsab541 gsq	27	17	26	26	26	21	22	18	17
lsab561 gsq	2	2	2	2	2	2	2	2	2
lsab571 gsq	273	551	595	307	267	283	683	223	223
lsab591 gsq	1	1	1	1	1	1	1	1	1
lsab601 gsq	33	31	31	31	33	31	34	33	31
lsab611 gsq	43	46	46	46	52	47	46	46	43

Table A 99: L-BDD method, number of nodes, small trees, 3

FT name	1 scheme	2 scheme	3 scheme	4 scheme	5 scheme	6 scheme	7 scheme	8 scheme	Minimum
lisab621 gsq	31	27	27	30	30	30	30	32	27
lisab631 gsq	13	13	13	13	13	13	12	14	12
lisab641 gsq	47	43	43	43	47	47	43	47	43
lisab651 gsq	1	1	1	1	1	1	1	1	1
lisab661 gsq	355	359	359	461	339	358	237	248	237
lisab671 gsq	209	199	199	195	205	174	312	188	174
lisab681 gsq	1	1	1	1	1	1	1	1	1
lisab691 gsq	55	57	57	37	56	37	53	60	37
lisab701 gsq	94	73	73	65	83	70	80	94	65
lisab711 gsq	8	8	8	8	8	8	8	8	8
lisab721 gsq	708	323	323	313	677	688	1408	520	313
lisab731 gsq	1	1	1	1	1	1	1	1	1
lisab751 gsq	1	1	1	1	1	1	1	1	1
lisab771 gsq	1231	1577	1465	1032	1459	1046	1175	1502	1032
lisab781 gsq	543	259	269	438	372	438	364	248	248
lisab811 gsq	1	1	1	1	1	1	1	1	1
lisab821 gsq	2942	3150	3150	3893	2415	2209	4348	2245	2209
lisab831 gsq	163	143	143	138	163	149	224	143	138
lisab841 gsq	152	147	124	123	128	123	205	132	123
lisab861 gsq	610	946	944	759	531	714	1215	540	531
lisab881 gsq	60	56	56	46	69	62	56	69	46
lisab911 gsq	419	182	182	165	332	194	356	328	165
lisab951 gsq	3	3	3	3	3	3	3	3	3
lisab971 gsq	3	3	3	3	3	3	3	3	3
lisab981 gsq	1	1	1	1	1	1	1	1	1
lisa1031 gsq	1	1	1	1	1	1	1	1	1
lisa1041 gsq	1	1	1	1	1	1	1	1	1
lisa1071 gsq	13	13	13	13	13	13	13	13	13
lisa1091 gsq	70	98	98	98	72	69	98	72	69
lisa1101 gsq	33	41	47	47	39	29	136	42	29
lisa1111 gsq	21	21	21	21	21	21	30	21	21
lisa1121 gsq	1874	279	284	1530	1637	1357	1498	1704	279
lisa1131 gsq	329	385	385	482	287	246	271	262	246
lisa1151 gsq	50	50	50	50	36	50	91	72	36
lisa1161 gsq	14	18	14	18	14	8	8	11	8
lisa1191 gsq	10	6	16	16	10	10	16	10	6
lisa1201 gsq	1	1	1	1	1	1	1	1	1
lisa1211 gsq	18	18	21	21	18	18	18	18	18
lisa1221 gsq	25	25	25	25	25	25	17	16	16
lisa1231 gsq	160	201	201	132	159	138	212	140	132
lisa1241 gsq	2991	1681	1700	1779	1902	1858	4246	1576	1576
rand1591 gsq	136	159	137	137	115	63	63	116	63
rand1611 gsq	848	1183	779	857	616	718	897	872	616
rand1631 gsq	1297	2284	1807	1709	1179	1037	3125	1314	1037
rand1671 gsq	594	669	676	699	470	667	486	283	283

Table A.100: L-BDD method, number of nodes, small trees, 4

FT name	1 scheme	2 scheme	3 scheme	4 scheme	5 scheme	6 scheme	7 scheme	8 scheme	Minimum
random61 gsq	7682	13010	11275	5601	6162	4309	10827	6908	4309
rando121 gsq	4286	5145	5114	6711	4081	5532	3176	4565	3176
rando591 gsq	6342	7084	7021	3966	5076	5303	1768	3357	1768
rand1321 gsq	9936	14811	14811	10789	4514	4840	11306	6529	4514
rand1501 gsq	15469	23831	25902	8157	13443	8256	9002	8928	8157
rand1581 gsq	2520	9649	6160	3504	2951	3884	5562	3180	2520
lisab521 gsq	7870	17486	17370	13627	4170	8268	13420	4091	4091
lisab741 gsq	128222	150879	131026	147645	196823	100050	236716	279618	100050
lisab891 gsq	4408	10041	8856	6993	5098	4883	10442	5571	4408
lisa1001 gsq	6013	7619	7405	12053	5925	6408	14871	4743	4743
rand1641 gsq	15384	27640	24191	24191	15219	20004	20885	11298	11298
rand1651 gsq	14797	10764	10764	8643	8984	12520	10051	8706	8643
rand1661 gsq	27938	54015	42454	49110	13659	34883	67479	16620	13659

Table A.101: L-BDD method, number of nodes, 'large' trees

FT name	1 scheme	2 scheme	3 scheme	4 scheme	5 scheme	6 scheme	7 scheme	8 scheme	Minimum
astolfo1 gsq	0 04	0 04	0 04	0 03	0 04	0 04	0 04	0 04	0 03
benjam1 gsq	0 1	0 1	0 11	0 11	0 1	0 11	0 1	0 1	0 1
bpfeg031 gsq	0 48	0 57	0 54	0 51	0 53	0 5	0 46	0 46	0 46
bpfen051 gsq	0 38	0 37	0 38	0 37	0 37	0 37	0 37	0 38	0 37
bpfig051 gsq	0 32	0 33	0 34	0 33	0 41	0 32	0 33	0 33	0 32
bpfin051 gsq	0 01	0 01	0 02	0 01	0 02	0 02	0 01	0 02	0 01
bpfsw021 gsq	1 52	1 5	1 55	1 52	1 58	1 49	1 56	1 5	1 49
dre10191 gsq	0	0	0 01	0	0 01	0	0	0 01	0
dre10321 gsq	0 01	0 01	0 01	0 01	0 01	0 01	0 01	0 01	0 01
dre10571 gsq	0 03	0 03	0 03	0 02	0 03	0 03	0 03	0 04	0 02
dre10581 gsq	0 29	0 3	0 3	0 31	0 3	0 31	0 3	0 3	0 29
dre10591 gsq	0 76	0 73	0 77	0 76	0 76	0 74	0 74	0 73	0 73
dresden1 gsq	1 69	0 99	1 02	1 02	1 33	1 09	4 23	1 15	0 99
hpsif021 gsq	0 12	0 12	0 12	0 13	0 13	0 13	0 12	0 13	0 12
hpsif031 gsq	0 03	0 03	0 03	0 03	0 03	0 03	0 02	0 03	0 02
hpsif211 gsq	0 36	0 9	0 54	0 35	0 39	0 36	0 36	0 41	0 35
hpsif361 gsq	0 03	0 02	0 02	0 02	0 02	0 03	0 02	0 03	0 02
dtree31 gsq	0 01	0 01	0 01	0 01	0	0	0	0 01	0
dtree41 gsq	0 01	0 01	0 01	0 01	0 01	0 01	0 02	0 01	0 01
dtree51 gsq	0 01	0 01	0 01	0	0	0 01	0	0 01	0
khictre1 gsq	0 21	0 2	0 2	0 2	0 19	0 2	0 19	0 19	0 19
nakash1 gsq	0 1	0 09	0 1	0 1	0 09	0 1	0 1	0 1	0 09
trials11 gsq	0 16	0 21	0 21	0 26	0 16	0 15	0 22	0 17	0 15
trials41 gsq	0 44	1 38	2 26	0 97	0 44	0 89	5 62	0 44	0 44
random31 gsq	0 23	0 21	0 22	0 21	0 22	0 21	0 23	0 22	0 21
random81 gsq	0 17	0 17	0 17	0 16	0 16	0 16	0 18	0 17	0 16
rando131 gsq	0 14	0 14	0 14	0 13	0 14	0 13	0 14	0 13	0 13
rando161 gsq	0 24	0 25	0 25	0 25	0 25	0 24	0 25	0 24	0 24
rando231 gsq	0 09	0 08	0 11	0 08	0 1	0 08	0 09	0 09	0 08
rando251 gsq	0 01	0 01	0 01	0 01	0 01	0 01	0 01	0 01	0 01
rando271 gsq	0 22	0 22	0 26	0 21	0 22	0 21	0 23	0 22	0 21
rando281 gsq	0	0 01	0 02	0 01	0 01	0 01	0 01	0 01	0
rando291 gsq	0 74	0 41	0 41	0 37	0 51	0 43	0 61	0 33	0 33
rando301 gsq	0 13	0 11	0 12	0 12	0 14	0 11	0 11	0 12	0 11
rando311 gsq	0 04	0 04	0 03	0 04	0 03	0 03	0 03	0 03	0 03
rando331 gsq	0 22	0 22	0 23	0 22	0 21	0 22	0 24	0 22	0 21
rando341 gsq	0 19	0 19	0 19	0 18	0 19	0 18	0 19	0 18	0 18
rando351 gsq	0 18	0 18	0 19	0 16	0 16	0 18	0 17	0 18	0 16
rando361 gsq	0 01	0 01	0 02	0 01	0 02	0 01	0 01	0 01	0 01
rando371 gsq	0 12	0 11	0 11	0 1	0 12	0 11	0 13	0 12	0 1
rando381 gsq	0 01	0 02	0 01	0	0 01	0 01	0	0 02	0
rando391 gsq	0 17	0 15	0 15	0 17	0 17	0 17	0 19	0 17	0 15
rando401 gsq	0 19	0 19	0 18	0 17	0 18	0 19	0 19	0 17	0 17
rando421 gsq	0 16	0 15	0 16	0 16	0 15	0 15	0 15	0 16	0 15
rando431 gsq	0 04	0 04	0 04	0 04	0 06	0 07	0 05	0 06	0 04
rando441 gsq	0 27	0 2	0 2	0 16	0 75	0 26	0 2	1 51	0 16
rando451 gsq	0 21	0 18	0 19	0 2	0 2	0 21	0 21	0 2	0 18
rando461 gsq	0 17	0 16	0 15	0 16	0 16	0 16	0 17	0 15	0 15
rando471 gsq	0 32	0 31	0 34	0 31	0 29	0 3	0 3	0 29	0 29
rando481 gsq	0 15	0 16	0 16	0 15	0 15	0 14	0 15	0 15	0 14
rando521 gsq	0 29	0 29	0 31	0 27	0 27	0 27	0 29	0 29	0 27
rando531 gsq	0 15	0 14	0 16	0 14	0 14	0 16	0 15	0 14	0 14
rando541 gsq	0 23	0 23	0 24	0 22	0 23	0 22	0 23	0 22	0 22
rando551 gsq	0 18	0 19	0 19	0 19	0 18	0 19	0 19	0 17	0 17

Table A.102: L-BDD method, time, small trees, 1

FT name	1 scheme	2 scheme	3 scheme	4 scheme	5 scheme	6 scheme	7 scheme	8 scheme	Minimum
rando581 gsq	0 15	0 15	0 15	0 16	0 15	0 15	0 15	0 16	0 15
rando601 gsq	0 19	0 2	0 21	0 18	0 19	0 18	0 2	0 19	0 18
rando611 gsq	0 11	0 12	0 12	0 11	0 11	0 11	0 12	0 12	0 11
rando621 gsq	0 19	0 17	0 17	0 18	0 17	0 17	0 17	0 2	0 17
rando631 gsq	0 2	0 18	0 18	0 17	0 18	0 18	0 18	0 19	0 17
rando641 gsq	0 21	0 19	0 2	0 19	0 2	0 21	0 2	0 21	0 19
rando651 gsq	0 05	0 04	0 05	0 04	0 05	0 05	0 05	0 05	0 04
rando661 gsq	0 18	0 17	0 18	0 17	0 17	0 17	0 19	0 17	0 17
rando701 gsq	0 08	0 07	0 08	0 07	0 07	0 08	0 07	0 07	0 07
rando731 gsq	0 22	0 21	0 21	0 2	0 23	0 22	0 23	0 22	0 2
rando751 gsq	0 01	0 01	0	0 01	0 01	0 01	0 01	0 02	0
rando761 gsq	0 18	0 17	0 17	0 18	0 18	0 18	0 18	0 16	0 16
rando771 gsq	0 12	0 11	0 11	0 1	0 1	0 11	0 11	0 12	0 1
rando781 gsq	0 01	0 01	0 01	0 01	0 01	0 01	0 01	0 01	0 01
rando801 gsq	0 17	0 16	0 17	0 15	0 17	0 16	0 16	0 17	0 15
rando831 gsq	0 07	0 07	0 07	0 07	0 06	0 07	0 07	0 06	0 06
rando841 gsq	0 26	0 24	0 25	0 24	0 26	0 24	0 25	0 24	0 24
rando851 gsq	0 01	0 01	0 01	0 01	0 01	0 02	0 01	0	0
rando871 gsq	0 01	0 02	0 01	0 01	0 01	0 01	0 01	0	0
rando881 gsq	0 04	0 04	0 05	0 04	0 05	0 04	0 04	0 05	0 04
rando891 gsq	0 21	0 22	0 21	0 21	0 21	0 22	0 22	0 21	0 21
rando911 gsq	9 92	0 38	0 38	1 5	5 54	38 7	0 58	8 79	0 38
rando921 gsq	0 5	0 48	0 46	0 45	0 48	0 45	0 76	0 48	0 45
rando931 gsq	0 19	0 18	0 19	0 17	0 2	0 2	0 19	0 19	0 17
rando951 gsq	0 23	0 23	0 22	0 23	0 23	0 22	0 22	0 23	0 22
rando981 gsq	0 16	0 12	0 12	0 12	0 15	0 12	0 11	0 16	0 11
rando991 gsq	0 26	0 25	0 27	0 22	0 22	0 24	0 23	0 22	0 22
rand1001 gsq	0 16	0 16	0 15	0 16	0 16	0 17	0 15	0 16	0 15
rand1031 gsq	0 2	0 21	0 2	0 19	0 21	0 2	0 21	0 2	0 19
rand1041 gsq	0 15	0 15	0 15	0 15	0 15	0 16	0 15	0 15	0 15
rand1051 gsq	0 07	0 06	0 07	0 06	0 06	0 06	0 07	0 06	0 06
rand1061 gsq	0 16	0 15	0 15	0 16	0 16	0 16	0 16	0 14	0 14
rand1081 gsq	0 2	0 19	0 19	0 19	0 21	0 19	0 2	0 19	0 19
rand1091 gsq	0 16	0 15	0 15	0 15	0 14	0 15	0 15	0 16	0 14
rand1101 gsq	0 03	0 03	0 03	0 03	0 03	0 03	0 03	0 03	0 03
rand1111 gsq	0 09	0 09	0 08	0 09	0 08	0 09	0 09	0 09	0 08
rand1151 gsq	0 09	0 08	0 08	0 08	0 07	0 08	0 08	0 08	0 07
rand1161 gsq	0 18	0 18	0 18	0 16	0 18	0 19	0 19	0 17	0 16
rand1171 gsq	0 04	0 03	0 03	0 03	0 03	0 03	0 03	0 03	0 03
rand1181 gsq	0 23	0 22	0 22	0 22	0 21	0 21	0 23	0 22	0 21
rand1191 gsq	0 17	0 16	0 16	0 15	0 16	0 17	0 17	0 16	0 15
rand1201 gsq	0 16	0 17	0 17	0 15	0 16	0 15	0 17	0 16	0 15
rand1211 gsq	0 06	0 07	0 06	0 06	0 05	0 06	0 06	0 06	0 05
rand1231 gsq	0 02	0 02	0 02	0 02	0 03	0 03	0 03	0 02	0 02
rand1241 gsq	0 01	0 01	0 01	0 01	0	0 01	0 01	0 01	0
rand1251 gsq	0 02	0 03	0 03	0 03	0 02	0 03	0 02	0 02	0 02
rand1261 gsq	0 15	0 16	0 15	0 15	0 15	0 15	0 15	0 14	0 14
rand1271 gsq	0 01	0 01	0 01	0 01	0 01	0 01	0 01	0 01	0 01
rand1281 gsq	0 12	0 11	0 11	0 12	0 12	0 12	0 11	0 11	0 11
rand1291 gsq	0 01	0 01	0 01	0 01	0 01	0 01	0 01	0 01	0 01
rand1301 gsq	0 02	0	0 01	0 01	0 01	0 01	0 01	0 02	0
rand1341 gsq	0 16	0 17	0 15	0 17	0 16	0 17	0 16	0 16	0 15
rand1351 gsq	0 25	0 24	0 24	0 22	0 24	0 23	0 25	0 24	0 22
rand1371 gsq	0 07	0 06	0 06	0 06	0 06	0 06	0 06	0 06	0 06

Table A.103: L-BDD method, time, small trees, 2

FT name	1 scheme	2 scheme	3 scheme	4 scheme	5 scheme	6 scheme	7 scheme	8 scheme	Minimum
rand1381 gsq	0.17	0.16	0.16	0.16	0.16	0.18	0.18	0.16	0.16
rand1391 gsq	0.16	0.17	0.17	0.17	0.17	0.17	0.18	0.16	0.16
rand1411 gsq	0.02	0.02	0.02	0.02	0.03	0.02	0.02	0.02	0.02
rand1421 gsq	0.68	10.41	10.57	0.78	0.99	1.28	1.92	0.69	0.68
rand1431 gsq	0.15	0.17	0.17	0.15	0.15	0.16	0.16	0.16	0.15
rand1441 gsq	0.12	0.12	0.12	0.11	0.11	0.12	0.11	0.12	0.11
rand1451 gsq	0.16	0.16	0.16	0.17	0.16	0.16	0.18	0.16	0.16
rand1461 gsq	0.07	0.08	0.07	0.07	0.07	0.07	0.07	0.07	0.07
rand1471 gsq	0.53	0.5	0.49	0.5	0.47	0.47	0.66	0.51	0.47
rand1481 gsq	0.15	0.16	0.16	0.15	0.16	0.14	0.15	0.14	0.14
rand1491 gsq	0.01	0.01	0.01	0.01	0.01	0.01	0.01	0.01	0.01
rand1511 gsq	0.01	0.01	0.01	0.01	0.02	0.02	0.01	0.01	0.01
rand1531 gsq	0.15	0.16	0.15	0.16	0.16	0.16	0.16	0.15	0.15
rand1541 gsq	0.01	0	0	0.01	0	0.01	0.01	0.01	0
rand1551 gsq	0.16	0.19	0.21	0.16	0.16	0.16	0.2	0.15	0.15
rand1561 gsq	0.01	0.01	0.01	0.01	0.01	0.01	0.02	0.01	0.01
lsaba11 gsq	0.24	0.23	0.24	0.23	0.23	0.23	0.24	0.22	0.22
lsaba31 gsq	21.85	8.14	6.48	7.7	18.18	8.06	6.82	9.27	6.48
lsaba41 gsq	9.02	3.86	1.13	1.16	4.25	5	14.94	5.37	1.13
lsaba51 gsq	0.37	0.33	0.33	0.33	0.35	0.33	0.36	0.34	0.33
lsaba61 gsq	0.15	0.17	0.17	0.13	0.15	0.13	0.16	0.15	0.13
lsaba71 gsq	0.23	0.23	0.23	0.21	0.22	0.24	0.23	0.22	0.21
lsaba91 gsq	0.21	0.2	0.21	0.2	0.21	0.21	0.21	0.18	0.18
lsab101 gsq	0.2	0.12	0.13	0.12	0.16	0.16	0.13	0.15	0.12
lsab111 gsq	0.02	0.01	0.02	0.01	0.01	0.01	0.01	0.01	0.01
lsab131 gsq	0.02	0.02	0.02	0.02	0.02	0.02	0.03	0.03	0.02
lsab171 gsq	0.26	0.23	0.25	0.24	0.24	0.25	0.25	0.24	0.23
lsab191 gsq	3.15	3.02	3.01	3.07	3.04	3.02	3.01	3.02	3.01
lsab201 gsq	1.07	1.07	1.05	1.07	1.06	1.06	1.07	1.07	1.05
lsab241 gsq	0.04	0.04	0.04	0.04	0.04	0.04	0.04	0.04	0.04
lsab251 gsq	0.11	0.1	0.1	0.09	0.1	0.09	0.1	0.1	0.09
lsab271 gsq	0.29	0.28	0.3	0.28	0.27	0.29	0.36	0.27	0.27
lsab281 gsq	0.02	0.01	0.01	0	0.01	0.01	0.01	0.02	0
lsab301 gsq	0.21	0.2	0.21	0.19	0.23	0.2	0.21	0.22	0.19
lsab311 gsq	105.54	167.26	99.81	19.39	6.84	7.45	96.61	1.78	1.78
lsab341 gsq	0.2	0.2	0.2	0.2	0.22	0.2	0.22	0.21	0.2
lsab351 gsq	0.48	1.11	0.62	0.3	0.3	0.38	0.77	0.28	0.28
lsab361 gsq	0.1	0.11	0.11	0.11	0.11	0.1	0.11	0.1	0.1
lsab371 gsq	0.17	0.17	0.18	0.15	0.16	0.16	0.17	0.16	0.15
lsab421 gsq	0.01	0.01	0.01	0.01	0.01	0.01	0.01	0	0
lsab441 gsq	0.18	0.17	0.18	0.17	0.18	0.18	0.18	0.16	0.16
lsab461 gsq	2.71	2.73	2.68	2.71	2.71	2.69	2.69	2.69	2.68
lsab471 gsq	0.02	0.03	0.03	0.02	0.02	0.03	0.03	0.02	0.02
lsab481 gsq	0.03	0.02	0.02	0.03	0.02	0.03	0.02	0.02	0.02
lsab491 gsq	0.04	0.05	0.04	0.04	0.04	0.04	0.04	0.04	0.04
lsab501 gsq	0	0.01	0.01	0.01	0.01	0.01	0	0.01	0
lsab511 gsq	0.05	0.06	0.05	0.05	0.05	0.05	0.05	0.06	0.05
lsab531 gsq	0.01	0.01	0.01	0	0	0.01	0.01	0.01	0
lsab541 gsq	0.05	0.06	0.05	0.05	0.05	0.06	0.05	0.06	0.05
lsab561 gsq	0.01	0.01	0.01	0.01	0.01	0.02	0.01	0.02	0.01
lsab571 gsq	0.22	0.26	0.24	0.22	0.24	0.21	0.24	0.19	0.19
lsab591 gsq	0.1	0.11	0.09	0.1	0.1	0.09	0.1	0.11	0.09
lsab601 gsq	0.04	0.05	0.04	0.05	0.05	0.04	0.03	0.05	0.03
lsab611 gsq	0.06	0.07	0.07	0.07	0.07	0.08	0.07	0.07	0.06

Table A.104 L-BDD method, time, small trees, 3

FT name	1 scheme	2 scheme	3 scheme	4 scheme	5 scheme	6 scheme	7 scheme	8 scheme	Minimum
lisab621 gsq	0.2	0.2	0.18	0.18	0.2	0.19	0.18	0.18	0.18
lisab631 gsq	0.03	0.03	0.03	0.03	0.02	0.03	0.02	0.03	0.02
lisab641 gsq	0.18	0.19	0.2	0.19	0.2	0.2	0.18	0.18	0.18
lisab651 gsq	0	0.01	0.01	0.01	0.01	0.01	0.01	0.01	0
lisab661 gsq	0.18	0.18	0.18	0.18	0.18	0.18	0.17	0.17	0.17
lisab671 gsq	0.18	0.18	0.18	0.16	0.18	0.17	0.19	0.18	0.16
lisab681 gsq	0.01	0.01	0.01	0.01	0	0.02	0	0.02	0
lisab691 gsq	0.05	0.06	0.06	0.05	0.06	0.05	0.05	0.06	0.05
lisab701 gsq	0.08	0.09	0.09	0.08	0.09	0.09	0.08	0.08	0.08
lisab711 gsq	0.16	0.17	0.16	0.16	0.16	0.15	0.16	0.17	0.15
lisab721 gsq	1	0.37	0.37	0.21	0.66	0.53	29.09	0.3	0.21
lisab731 gsq	0.01	0.01	0.01	0.01	0.01	0.01	0.01	0	0
lisab751 gsq	0	0.01	0.01	0.01	0.01	0.02	0.01	0.02	0
lisab771 gsq	34.94	1.97	1.91	5.53	15.71	9.11	1.21	7.4	1.21
lisab781 gsq	0.16	0.13	0.12	0.13	0.12	0.14	0.13	0.12	0.12
lisab811 gsq	0	0.01	0	0.01	0.01	0.01	0.01	0	0
lisab821 gsq	4.68	3.23	3.17	4.06	3.81	4.75	5.05	4.07	3.17
lisab831 gsq	0.09	0.1	0.11	0.09	0.1	0.1	0.1	0.09	0.09
lisab841 gsq	0.12	0.13	0.11	0.11	0.11	0.12	0.14	0.11	0.11
lisab861 gsq	0.22	0.41	0.34	0.26	0.24	0.25	0.39	0.26	0.22
lisab881 gsq	0.06	0.05	0.04	0.05	0.05	0.06	0.04	0.05	0.04
lisab911 gsq	0.26	0.25	0.25	0.24	0.26	0.24	0.27	0.26	0.24
lisab951 gsq	0	0.02	0.01	0	0	0	0	0	0
lisab971 gsq	0.01	0.02	0.01	0.02	0.01	0.02	0.02	0.02	0.01
lisab981 gsq	0	0	0.01	0	0.01	0.01	0	0.01	0
lisa1031 gsq	0	0	0	0.01	0	0.01	0.01	0	0
lisa1041 gsq	0	0	0	0	0	0	0	0	0
lisa1071 gsq	0.16	0.16	0.16	0.16	0.16	0.15	0.17	0.16	0.15
lisa1091 gsq	0.1	0.11	0.11	0.1	0.11	0.09	0.1	0.1	0.09
lisa1101 gsq	0.25	0.25	0.25	0.25	0.25	0.24	0.25	0.25	0.24
lisa1111 gsq	0.2	0.21	0.21	0.2	0.18	0.19	0.2	0.2	0.18
lisa1121 gsq	1.03	0.4	0.41	0.44	1.42	0.69	0.52	1.21	0.4
lisa1131 gsq	0.51	0.3	0.31	0.31	0.31	0.29	0.27	0.31	0.27
lisa1151 gsq	0.08	0.08	0.08	0.07	0.07	0.08	0.1	0.1	0.07
lisa1161 gsq	0.03	0.05	0.04	0.04	0.03	0.03	0.03	0.03	0.03
lisa1191 gsq	0.02	0.02	0.03	0.04	0.02	0.02	0.03	0.02	0.02
lisa1201 gsq	0.01	0.02	0.01	0.01	0.01	0.01	0	0.02	0
lisa1211 gsq	0.18	0.19	0.18	0.17	0.16	0.17	0.18	0.17	0.16
lisa1221 gsq	0.04	0.04	0.04	0.04	0.04	0.04	0.04	0.04	0.04
lisa1231 gsq	0.17	0.16	0.16	0.15	0.16	0.14	0.16	0.15	0.14
lisa1241 gsq	26.1	6.02	6.06	2.95	1.79	3.33	56.77	1.86	1.79
rand1591 gsq	0.12	0.12	0.12	0.11	0.12	0.1	0.1	0.12	0.1
rand1611 gsq	0.99	3.18	1.83	0.34	0.27	0.26	0.37	0.34	0.26
rand1631 gsq	0.92	1.82	0.74	0.59	0.59	0.45	3.42	0.77	0.45
rand1671 gsq	0.31	0.29	0.29	0.29	0.29	0.28	0.3	0.28	0.28

Table A.105 L-BDD method, time, small trees, 4

FT name	1 scheme	2 scheme	3 scheme	4 scheme	5 scheme	6 scheme	7 scheme	8 scheme	Minimum
random61 gsq	5 41	6 05	6 17	3 63	0 58	0 55	4 17	4 92	0 55
rando121 gsq	14 64	2 89	3 45	12 33	3 75	3 72	3 27	18 47	2 89
rando591 gsq	2 49	1 42	1 14	10 14	1 25	1 17	0 2	2 7	0 2
rand1321 gsq	1195 39	4808 91	4814 03	11 51	39 25	55 59	15 81	40 19	11 51
rand1501 gsq	97 76	107 32	128 38	2 08	24 85	2 04	44 66	4 93	2 04
rand1581 gsq	1 03	2 75	27 23	1 02	0 74	0 7	1 16	1 42	0 7
lisab521 gsq	5 5	22 28	23 44	1 08	9 09	8 36	7 42	2 73	1 08
lisab741 gsq	13294 04	1022 95	1238 29	649 29	33664 83	3167 05	66053 05	53795 65	649 29
lisab891 gsq	3 08	92 4	73 23	1 19	0 47	0 45	92 03	0 75	0 45
lisa1001 gsq	5 98	3 17	2 67	5 24	2 88	3 89	6 64	2 56	2 56
rand1641 gsq	475 07	23 04	9 74	9 72	232 7	41 01	811 39	80 75	9 72
rand1651 gsq	10 28	1 99	2 05	3 74	4 08	4 11	2 64	6 45	1 99
rand1661 gsq	723 02	4742 96	755 86	172 12	117 44	371 59	1042 94	217 51	117 44

Table A.106: L-BDD method, time, 'large' trees

FT name	1 scheme	2 scheme	3 scheme	4 scheme	5 scheme	6 scheme	7 scheme	8 scheme	Minimum
random61 gsq	0 09	0 08	0 1	0 09	0 14	0 11	0 11	0 07	0 07
rando121 gsq	0 17	0 14	0 14	0 13	0 14	0 12	0 15	0 11	0 11
rando591 gsq	0 05	0 04	0 05	0 05	0 05	0 03	0 04	0 05	0 03
rand1321 gsq	1 07	3 2	3 1	0 58	0 23	0 28	0 49	0 38	0 23
rand1501 gsq	0 07	0 12	0 1	0 06	0 07	0 06	0 04	0 06	0 04
rand1581 gsq	0 26	0 28	0 26	0 26	0 27	0 25	0 24	0 26	0 24
lisab521 gsq	0 08	0 74	0 72	0 1	0 08	0 08	0 16	0 07	0 07
lisab741 gsq	6 91	6 32	6 32	6 61	8 74	6 42	6 74	13 02	6 32
lisab891 gsq	0 09	0 15	0 21	0 07	0 07	0 07	0 21	0 07	0 07
lisa1001 gsq	0 16	0 08	0 08	0 17	0 1	0 1	0 25	0 08	0 08
rand1641 gsq	0 52	0 41	0 23	0 23	0 23	0 21	1 25	0 22	0 21
rand1651 gsq	0 31	0 34	0 31	0 3	0 29	0 29	0 32	0 29	0 29
rand1661 gsq	0 19	3 2	1 85	0 43	0 14	0 48	0 89	0 2	0 14

Table A.107 Hybrid method for non-coherent fault trees, processing time

FT name	1 scheme	2 scheme	3 scheme	4 scheme	5 scheme	6 scheme	7 scheme	8 scheme	Minimum
random61 gsq	623	486	601	402	630	524	573	609	402
rando121 gsq	619	511	505	554	499	323	346	449	323
rando591 gsq	592	354	360	277	581	348	363	424	277
rand1321 gsq	1097	1100	1100	724	581	615	747	810	581
rand1501 gsq	1064	1298	1046	757	1124	723	574	985	574
rand1581 gsq	1198	1015	1091	949	1114	957	1000	1081	949
lisab521 gsq	852	1505	1504	975	789	819	620	669	620
lisab741 gsq	1946	606	626	648	3816	808	660	2700	606
lisab891 gsq	761	448	729	413	436	405	575	427	405
lisa1001 gsq	955	415	409	627	692	600	710	544	409
rand1641 gsq	960	1037	634	634	544	485	1093	528	485
rand1651 gsq	915	596	596	357	468	312	516	645	312
rand1661 gsq	906	1134	975	963	666	949	1048	796	666

Table A 108 ZBDD method, maximum required size

FT name	1 scheme	2 scheme	3 scheme	4 scheme	5 scheme	6 scheme	7 scheme	8 scheme	Minimum
random61 gsq	610	475	590	390	617	512	561	596	390
rando121 gsq	608	502	496	546	489	314	335	440	314
rando591 gsq	582	345	351	268	572	339	354	413	268
rand1321 gsq	1080	1080	1080	708	566	599	731	790	566
rand1501 gsq	1053	1288	1035	746	1110	708	560	971	560
rand1581 gsq	1186	1005	1081	937	1100	943	988	1067	937
lisab521 gsq	835	1346	1345	856	769	695	611	634	611
lisab741 gsq	1938	598	618	641	3808	800	651	2692	598
lisab891 gsq	752	439	720	403	426	395	565	416	395
lisa1001 gsq	940	404	398	610	679	587	697	532	398
rand1641 gsq	948	1029	626	626	533	475	1085	517	475
rand1651 gsq	909	590	590	351	462	306	511	639	306
rand1661 gsq	889	1120	961	945	649	931	1033	781	649

Table A 109. Hybrid method for non-coherent fault trees, maximum required size

