

**CRANFIELD UNIVERSITY**

**SYED ADEEL HUSSAIN**

**RISK BASED RELIABILITY ALLOCATION AT  
COMPONENT LEVEL IN NON-REPAIRABLE  
SYSTEMS BY USING EVOLUTIONARY  
ALGORITHM**

**SCHOOL OF APPLIED SCIENCES**

**PhD THESIS**

**CRANFIELD UNIVERSITY**

SCHOOL OF APPLIED SCIENCES

**PhD THESIS**

Academic Period – May 2004 to May 2007

**Syed Adeel Hussain**

**Risk based reliability allocation at component level in  
non-repairable systems by using evolutionary algorithm**

Supervisor: M. Todinov & R. Allwood

April 2007

This report is submitted in fulfilment of the requirements for the Degree  
of Doctor of Philosophy

© Cranfield University, 2007.

All rights reserved. No part of this publication may be reproduced without the written permission of  
the copyright holder.

# ABSTRACT

The approach for setting system reliability in the risk-based reliability allocation (RBRA) method is driven solely by the amount of ‘total losses’ (sum of *reliability investment* and *risk of failure*) associated with a non-repairable system failure. For a system consisting of many components, reliability allocation by RBRA method becomes a very complex combinatorial optimisation problem particularly if large numbers of alternatives, with different levels of reliability and associated cost, are considered for each component. Furthermore, the complexity of this problem is magnified when the relationship between cost and reliability assumed to be non-linear and non-monotone. An optimisation algorithm (OA) is therefore developed in this research to demonstrate the solution for such difficult problems.

The core design of the OA originates from the fundamental concepts of basic Evolutionary Algorithms which are well known for emulating Natural process of evolution in solving complex optimisation problems through computer simulations of the key genetic operations such as ‘reproduction’, ‘crossover’ and ‘mutation’. However, the OA has been designed with significantly different model of evolution (for identifying valuable parent solutions and subsequently turning them into even better child solutions) compared to the classical genetic model for ensuring rapid and efficient convergence of the search process towards an optimum solution. The vital features of this OA model are ‘generation of all populations (samples) with unique chromosomes (solutions)’, ‘working exclusively with the elite chromosomes in each iteration’ and ‘application of prudently designed genetic operators on the elite chromosomes with extra emphasis on mutation operation’. For each possible combination of alternatives, both system reliability and cost of failure is computed by means of Monte-Carlo simulation technique.

For validation purposes, the optimisation algorithm is first applied to solve an already published reliability optimisation problem with constraint on some target level of system reliability, which is required to be achieved at a minimum system cost. After successful validation, the viability of the OA is demonstrated by showing its application in optimising four different non-repairable sample systems in

view of the risk based reliability allocation method. Each system is assumed to have discrete choice of component data set, showing monotonically increasing cost and reliability relationship among the alternatives, and a fixed amount associated with cost of failure. While this optimisation process is the main objective of the research study, two variations are also introduced in this process for the purpose of undertaking parametric studies. To study the effects of changes in the reliability investment on system reliability and total loss, the first variation involves using a different choice of discrete data set exhibiting a non-monotonically increasing relationship between cost and reliability among the alternatives. To study the effects of risk of failure, the second variation in the optimisation process is introduced by means of a different cost of failure amount, associated with a given non-repairable system failure.

The optimisation processes show very interesting results between system reliability and total loss. For instance, it is observed that while maximum reliability can generally be associated with high total loss and low risk of failure, the minimum observed value of the total loss is not always associated with minimum system reliability. Therefore, the results exhibit various levels of system reliability and total loss with both values showing strong sensitivity towards the selected combination of component alternatives. The first parametric study shows that second data set (non-monotone) creates more opportunities for the optimisation process for producing better values of the loss function since cheaper components with higher reliabilities can be selected with higher probabilities. In the second parametric study, it can be seen that the reduction in the cost of failure amount reduces the size of risk of failure which also increases the chances of using cheaper components with lower levels of reliability hence producing lower values of the loss functions.

The research study concludes that the risk-based reliability allocation method together with the optimisation algorithm can be used as a powerful tool for highlighting various levels of system reliabilities with associated total losses for any given system in consideration. This notion can be further extended in selecting optimal system configuration from various competing topologies. With such information to hand, reliability engineers can streamline complicated system designs in view of the required level of system reliability with minimum associated total cost



of premature failure. In all cases studied, the run time of the optimisation algorithm increases linearly with the complexity of the algorithm and due to its unique model of evolution, it appears to conduct very detailed multi-directional search across the solution space in fewer generations - a very important attribute for solving the kind of problem studied in this research. Consequently, it converges rapidly towards optimum solution unlike the classical genetic algorithm which gradually reaches the optimum, when successful. The research also identifies key areas for future development with the scope to expand in various other dimensions due to its interdisciplinary applications.

# LIST OF CONTENTS

ABSTRACT.....	i-1
ACKNOWLEDGEMENT.....	ii
NOTATIONS .....	iii
LIST OF FIGURES .....	iv
.....1. CHAPTER ONE.....	
1.1 INTRODUCTION .....	1-2
1.2 GENERAL APPROACH FOR SETTING QUANTITATIVE RELIABILITY REQUIREMENTS.....	1-3
1.3 RISK BASED APPROACH FOR SETTING QUANTITATIVE RELIABILITY REQUIREMENTS.....	1-6
1.3.1 Statement of the Optimisation Problem .....	1-8
1.3.2 Optimisation Diagram.....	1-8
1.4 RESEARCH AIMS.....	1-10
1.4.1 Mathematical Formulation .....	1-10
1.4.2 Objective Function.....	1-12
1.5 RESEARCH OBJECTIVES.....	1-13
1.6 RESEARCH METHODOLOGY .....	1-15
.....2. CHAPTER TWO.....	
2.1 SYSTEM RELIABILITY .....	2-2
2.1.1 Basic Configurations of Reliability Systems.....	2-3
2.1.2 System Reliability Computation of Complex System.....	2-6
2.1.3 Common Probability Distributions for Modelling Time to Failure.....	2-8
2.1.3.1 Exponential Distribution.....	2-8
2.1.3.2 Weibull Distribution.....	2-10
2.1.3.3 Other Distributions.....	2-11
2.2 RELIABILITY OPTIMISATION .....	2-13
2.2.1 Modes of Optimisation .....	2-14
2.2.2 Optimisation Models.....	2-15
2.2.3 Classification of System Reliability Optimisation.....	2-15
2.2.3.1 Category One - by System configuration.....	2-16
2.2.3.2 Category Two - by Problem Type .....	2-16
2.2.3.3 Category Three - by Optimisation Techniques.....	2-17
2.2.4 Computational Methods of Optimisation.....	2-17
2.3 REVIEW OF OPTIMISATION PROCESSES USING THE RISK BASED RELIABILITY ALLOCATION METHOD.....	2-20
2.4 GENERAL REVIEW OF RELIABILITY OPTIMISATION .....	2-26
2.4.1 Heuristics for Redundancy Allocation.....	2-26
2.4.2 Meta-heuristic Algorithms for Redundancy Allocation (Genetic Algorithms, Simulated Annealing and Tabu Search).....	2-27

2.4.3	Exact Methods for Redundancy Allocation.....	2-28
2.4.4	Heuristics for Reliability-Redundancy Allocation.....	2-28
2.4.5	Multiple Objective Optimisation in Reliability Systems .....	2-29
2.4.6	Optimal Assignment of Interchangeable Components in Reliability Systems .....	2-30
2.4.7	Effort Function Optimisation .....	2-31
<b>.....3. CHAPTER THREE.....</b>		
3.1	INTRODUCTION .....	3-2
3.2	GENERAL STRUCTURE .....	3-3
3.3	FEATURES OF EVOLUTIONARY ALGORITHM.....	3-4
3.3.1	Biological Overview of Evolutionary Algorithm.....	3-4
3.3.2	Terminologies and Concepts in Evolutionary Algorithms.....	3-6
3.3.2.1	Chromosome Representation.....	3-6
3.3.2.2	Global and Local Optimum Results.....	3-7
3.3.2.3	Population Structure .....	3-9
3.3.2.4	Fitness Function .....	3-10
3.3.3	Processes of Variations in Evolutionary Algorithm .....	3-12
3.3.4	Selection & Replacement Process – The Model of Evolution .....	3-24
3.4	DESIGNING EVOLUTIONARY ALGORITHMS .....	3-27
3.4.1	Genotype & Phenotype Representation.....	3-27
3.4.2	Population Structure .....	3-28
3.4.3	Fitness Function .....	3-29
3.4.4	Variation Operators .....	3-29
3.4.5	Model of Evolution.....	3-30
3.4.6	Termination Criteria .....	3-30
3.5	TYPES OF EVOLUTIONARY ALGORITHMS .....	3-31
<b>.....4. CHAPTER FOUR.....</b>		
4.1	INTRODUCTION .....	4-2
4.2	GENERAL FRAMEWORK OF GENETIC ALGORITHMS .....	4-3
4.3	FEATURES OF GENETIC ALGORITHM .....	4-6
4.3.1	Terminologies and Concepts in Genetic Algorithm.....	4-6
4.3.1.1	Types of Chromosome Coding.....	4-6
4.3.1.2	Similarity Templates or Schemata .....	4-12
4.3.1.3	Niche Specialisation .....	4-17
4.3.2	Strength and Weaknesses of Genetic Algorithms.....	4-18
4.3.2.1	Convergence of Genetic Algorithms .....	4-19
4.3.2.2	Feasibility of Solutions.....	4-20
4.3.3	Comparative Analysis of Genetic Algorithms with Other EC Methods .....	4-21
4.4	TYPES OF GENETIC ALGORITHMS.....	4-23
4.5	APPLICATION OF GENETIC ALGORITHMS IN RELIABILITY OPTIMISATION .....	4-24

.....5. CHAPTER FIVE.....	
<b>5.1</b>	<b>INTRODUCTION ..... 5-2</b>
5.1.1	Epistasis Phenomenon ..... 5-2
5.1.2	Extremely Large Search Space..... 5-3
<b>5.2</b>	<b>THE OPTIMISATION ALGORITHM..... 5-10</b>
<b>5.3</b>	<b>FEATURES OF THE OPTIMISATION ALGORITHM..... 5-11</b>
5.3.1	Structure of the Chromosome ..... 5-12
5.3.2	Population Structure ..... 5-14
5.3.3	Crossover Operation..... 5-15
5.3.3.1	First Stage Crossover Operation (FSCO)..... 5-16
5.3.3.2	Second Stage Crossover Operation (SSCO)..... 5-18
5.3.3.3	Third Stage Crossover Operation (TSCO)..... 5-20
5.3.3.4	Numerical Example ..... 5-22
5.3.4	Mutation Operation..... 5-26
5.3.4.1	First Stage Mutation Operation (FSMO) ..... 5-26
5.3.4.2	Second Stage Mutation Operation (SSMO) ..... 5-29
5.3.4.3	Third Stage Mutation Operation (TSMO) ..... 5-29
5.3.4.4	Numerical Example ..... 5-32
5.3.5	Improvement Procedures..... 5-34
5.3.5.1	Types of Improvement Procedures ..... 5-35
5.3.6	Termination Criteria ..... 5-41
5.3.7	Process Diagram ..... 5-41
.....1. CHAPTER SIX.....	
<b>6.1</b>	<b>METHODOLOGY DEVELOPMENT ..... 6-2</b>
6.1.1	Component Characteristic ..... 6-2
6.1.2	System Reliability ..... 6-3
6.1.3	Exploring the Search Space ..... 6-3
6.1.4	Exploiting the Search Space..... 6-4
6.1.5	Complexity of the Method..... 6-4
<b>6.2</b>	<b>APPLICATION OF THE METHODOLOGY ..... 6-5</b>
6.2.1	Reliability Allocation Model ..... 6-6
6.2.2	The Optimisation Algorithm (OA)..... 6-7
6.2.3	Monte Carlo Method For Determining System Reliability And Total Loss..... 6-8
<b>6.3</b>	<b>RELIABILITY OPTIMISATION PROBLEM ..... 6-11</b>
6.3.1	Application of the Optimisation Algorithm..... 6-13
<b>6.4</b>	<b>RISK-BASED RELIABILITY ALLOCATION ..... 6-18</b>
6.4.1	Optimisation Results For System A ..... 6-21
6.4.2	Optimisation Results For System B ..... 6-28
6.4.3	Optimisation Results For System C ..... 6-31
6.4.4	Optimisation Results For System D ..... 6-33
<b>6.5</b>	<b>PARAMETRIC STUDY USING THE RISK BASED RELIABILITY   ALLOCATION METHOD AND OPTIMISATION ALGORITHM 6-35</b>
6.5.1	Optimisation Process Using Data Set Showing A Non- Monotonically Increasing Relationship Between Cost And Reliability ..... 6-37

6.5.1.1	Optimisation Results For System A .....	6-38
6.5.1.2	Optimisation Results For System B .....	6-39
6.5.1.3	Optimisation Results For System C .....	6-40
6.5.1.4	Optimisation Results For System D .....	6-41
6.5.2	Optimisation Process Using Lower Cost Of Failure.....	6-43
6.5.3	Parametric Study .....	6-44
6.5.3.1	Comparisons Of Results From The Two Cost-Reliability Data Tables .....	6-44
6.5.3.2	Comparisons Of Results From Two Different Costs Of Failure Amount .....	6-46
6.5.3.3	Comparisons Of Results For Establishing The Best System Topology .....	6-47
.....7. CHAPTER SEVEN.....		
7.1	OBSERVATIONS FROM THE OPTIMISATION PROCESS .....	7-2
7.1.1	Fundamental Process of Optimisation.....	7-3
7.1.2	Second Process of Optimisation.....	7-5
7.1.3	Third Process of Optimisation.....	7-7
7.2	COMPARATIVE ANALYSIS OF THE OPTIMISATION ALGORITHM.....	7-10
7.2.1	Risk Based Reliability Allocation .....	7-10
7.2.2	Optimisation using Evolutionary Algorithm.....	7-11
7.2.2.1	Population Structure .....	7-12
7.2.2.2	Chromosome Structure .....	7-12
7.2.2.3	Embedded Improvement Procedures .....	7-13
7.2.2.4	Software Implementation.....	7-15
.....8. CHAPTER EIGHT.....		
8.1	CONCLUSIONS .....	8-1
8.2	FUTURE RECOMMENDATIONS .....	8-3
.....APPENDIX I.....		
I.1	RELIABILITY & RISK ALGORITHMS.....	I-1
I.1.1	Method One .....	I-1
	Main Features .....	I-2
I.1.2	Method Two.....	I-2
I.1.3	The Algorithm .....	I-3
I.1.3.1.	Reliability of a complex lattice.....	I-5
I.2	APPLICATION OF THE CUT-SET AND TIE-SET SOFTWARE.....	I-7
I.2.1	Basic Concepts.....	I-7
I.2.1.1	Cut Set and Minimal Cut Set.....	I-8
I.2.1.2	Minimal Tie Set.....	I-9
I.2.1.3	Connection Matrix (Adjacency Matrix) .....	I-9

I.2.1.4	Reliability Evaluation of Bridge Network Using Cut Sets and Tie Sets .....	I-10
I.2.1.5	A Real Life Production System.....	I-14
I.3	RESEARCH PUBLICATIONS.....	I-17
.....APPENDIX II.....		
	OPTIMISATION RESULTS FOR SYSTEM B (FROM SECTION 7.3.2) .....	II-1
.....APPENDIX III.....		
	OPTIMISATION RESULTS FOR SYSTEM C (FROM SECTION 7.3.2) ...	III-1
.....APPENDIX IV.....		
	OPTIMISATION RESULTS FOR SYSTEM D (FROM SECTION 7.3.2) ....	IV-1
.....APPENDIX V.....		
	OPTIMISATION RESULTS FOR SYSTEM A (FROM SECTION 7.4.1.1) ..	V-1
.....APPENDIX VI.....		
	OPTIMISATION RESULTS FOR SYSTEM B (FROM SECTION 7.4.1.2) .	VI-1
.....APPENDIX VII.....		
	OPTIMISATION RESULTS FOR SYSTEM C (FROM SECTION 7.4.1.3)	VII-1
.....APPENDIX VIII.....		
	OPTIMISATION RESULTS FOR SYSTEM D (FROM SECTION 7.4.1.4) .....	VIII-1
.....APPENDIX IX.....		
	OPTIMISATION RESULTS FOR SYSTEM B (FROM SECTION 7.4.2) ....	IX-1
	System A .....	IX-2
	System B .....	IX-4
	System C .....	IX-6
	System D .....	IX-8
.....APPENDIX X.....		
	COMPUTER PROGRAM.....	CP-1
	References & Bibliography .....	R&B-I

# ACKNOWLEDGEMENT

Sincere thanks to **Professor Michael Todinov** for offering me the opportunity to undertake this challenging research degree at Cranfield University, and for his continuous support, assistance and widely accepted expertise in the field of reliability engineering and risk management. Michael has been an inspiration to me through out this doctorate degree both academically and socially; his supervision developed me into a more competitive individual with good thinking and problem solving skills. Also, his encouragements and can-do attitude really helped me to finish my degree on a high note despite experiencing very difficult times of my life which were encountered during my research. Thanks Michael !

I would like to offer my gratitude to **Robert Allwood** for accepting to supervise my work after Michael's departure and taken the time out from his demanding schedule to read and understand my thesis. Special thanks to **Jo Mosca** for proof reading my thesis and informing me with his valuable comments for improvements.

To **my family**, particularly **my parents** and **my wife, Oumna**, I would like to thank them for their support and encouragement they have given me, especially after the unfortunate circumstances in the final stages of the research. Finally, the love for my four little boys, **Sharjeel, Eshbeel, Rosheel** and **Yasheel** has also been a strong motivating factor throughout my research period.

---

# NOTATIONS

$M$	=	Number of component in a system
$N$	=	Number of alternatives of each component
$\Omega$	=	Total search space
$Q$	=	Reliability Investment (total cost of components)
$K$	=	Risk of Failure
$T_L$	=	Total loss from system failure
$p_f$	=	Probability of failure
$R_s$	=	System Reliability
$\bar{C}$	=	Expected cost given failure of a system
$p_{fi} \bar{c}_i$	=	Risk of failure of the $i^{th}$ component
$\gamma_{ij}$	=	Reliability of the $i^{th}$ component with $j^{th}$ alternative
$c_{ij}$	=	Cost of the $i^{th}$ component with $j^{th}$ alternative
$i$	=	1,2,3,..... $M$
$j$	=	1,2,3,..... $N$
$p_k$	=	Population Number, $k = (1,2,.....p_{RUN})$
$p_{RUN}$	=	Total number of populations
$p_{size}$	=	Size of the population
$\varepsilon_1$	=	Optimal chromosome
$\varepsilon_2$	=	Near optimal chromosome
$C_{RUN}$	=	Total number of crossover runs
$M_{RUN}$	=	Total number of mutation runs
$IMPROVE\_1()$	=	Improvement function for crossover operation
$IMPROVE\_2()$	=	Improvement function for mutation operation
$EA$	=	Evolutionary algorithm (optimisation algorithm)



---

# LIST OF FIGURES

Figure 1.1 Cost of Failure When Failure occurs before time interval ‘a’ .....	1-2
Figure 1.2 Risk Based Reliability Allocation Method.....	1-7
Figure 1.3 Before Optimisation: System with M Components with N alternatives ....	
.....	1-9
Figure 1.4 After Optimisation: System with M Components with N alternatives .....	
.....	1-9
Figure 1.5 Risk of Failure and System Reliability.....	1-13
Figure 1.6 Reliability Investment and System Reliability .....	1-14
Figure 2.1 Simple Block Diagram with one component, connected by points a and b	
.....	2-3
Figure 2.2 Reliability Block Diagram of a System in Series Configuration .....	2-4
Figure 2.3 Reliability Block Diagram of a System in Parallel Configuration.....	2-4
Figure 2.4 Example of a Complex System .....	2-6
Figure 2.5 Exponential Distribution.....	2-9
Figure 2.6 Risk Based Reliability Allocation Method.....	2-21
Figure 3.1 A simple evolutionary algorithm.....	3-3
Figure 3.2 Genotype decoding into phenotype.....	3-5
Figure 3.3 Chromosome Representation .....	3-6
Figure 3.4 Genotype and Phenotype Transition .....	3-7
Figure 3.5 Global Optimum And Local Optimum of a Function, $F(X)$ .....	3-9
Figure 3.6 Fitness of organism with respect to its environment.....	3-10
Figure 3.7 Fitness evaluation of chromosome in evolutionary algorithm .....	3-12
Figure 3.8 Roulette Wheel Selection Mechanism .....	3-25
Figure 4.1 A simple Genetic Algorithm .....	4-5
Figure 5.1 A system consisting of M components with N alternatives each.....	5-4
Figure 5.2 Component Surface Showing Coordinates of Cost and Reliability .....	5-5

---

Figure 5.3 Surface of the $i$ th Component with $N$ Alternatives Showing Monotonically Increasing Cost-Reliability Relationship .....	5-6
Figure 5.4 Surface of the $i$ th Component with $N$ Alternatives Showing Non-Monotonically Increasing Cost-Reliability Relationship .....	5-7
Figure 5.5 System Configuration Domain with Choice of $M$ Components, Represented as Surfaces. Also Shown are the Points in the Search Space Representing the Values of the Loss Function with Coordinates Located in Each of the $M$ Component Surfaces .....	5-9
Figure 5.6 Optimisation Algorithm .....	5-10
Figure 5.7 Chromosome Structure and Domain Mapping.....	5-13
Figure 5.8 Example Chromosome with Real Value Mapping of Cost and Reliability .....	5-14
Figure 5.9 First Stage Crossover Operation.....	5-17
Figure 5.10 Second Stage Crossover Operation .....	5-19
Figure 5.11 Third Stage Crossover Operation .....	5-21
Figure 5.12 First Stage Mutation Operation .....	5-28
Figure 5.13 Second Stage Mutation Operation.....	5-30
Figure 5.14 Third Stage Mutation Operation.....	5-31
Figure 5.15 Structure of the First Improvement Procedure .....	5-35
Figure 5.16 Structure of the Second Improvement Procedure.....	5-40
Figure 5.17 Pictorial Demonstration of the Optimisation Process .....	5-42
Figure 6.1 Structure of the Optimisation Algorithm.....	6-10
Figure 6.2 Reliability System with Nine Components .....	6-11
Figure 6.3 Total Search Area Explored By the Optimisation Algorithm .....	6-15
Figure 6.4 Effect of the Improvement Procedure In Crossover Process .....	6-16
Figure 6.5 Area Search by the Improvement Procedure Based On Random Mutations .....	6-16
Figure 6.6 Convergence of the Optimisation Toward Optimum Solution Using Improvement Procedures .....	6-17
Figure 6.7 Total Search Space And the Optimisation Process .....	6-17
Figure 6.8 Monotonically increasing relationship of cost-reliability from Table 7.1....	

---

---

.....	6-19
Figure 6.9 Four Reliability Systems with Nine Components Each .....	6-20
Figure 6.10 Optimisation Process of System Reliability and Total Loss in System A .....	6-22
Figure 6.11 Total Effect of Optimisation Process on System A.....	6-22
Figure 6.12 Crossover Process of System A.....	6-23
Figure 6.13 Effect of Mutation Process on Total Loss in System A .....	6-23
Figure 6.14 Region of the Search Space Examined By the Mutation Process in System A.....	6-24
Figure 6.15 Total Search Space examined by the optimisation algorithm for System A.....	6-24
Figure 6.16 Monotonically Increasing Relationship of Cost-Reliability for Data in Table 7.5 .....	6-33
Figure 6.17 Topology Comparison Using the First Cost-Reliability Data Set.....	6-48
Figure 6.18 Topology Comparison Using the Second Cost-Reliability Data Set ..	6-49
Figure 6.19 Topology Comparison Using the Lower Cost of Failure Amount.....	6-50
Figure 6.20 Run Time of the Optimisation Algorithm .....	6-48
Figure 7.1 Structure of the Optimisation Process .....	7-2
Figure 7.2 Effects of the Two Cost-Data Tables on Total Loss and System Reliability Values in All Four Systems .....	7-6
Figure 7.3 Effects of the Two Cost of Failure Amounts on Total Loss and System Reliability Values in all Four Systems .....	7-9
Figure I.1 Bridge network.....	I-7
Figure I.2 System Failure – Second Order Cut Set.....	I-12
Figure I.3 System Failure – Third Order Cut Set .....	I-12
Figure I.4 Minimal Cut Sets of Bridge Network .....	I-13
Figure I.5 Minimal Tie Sets of Bridge Network.....	I-13
Figure I.6 Reliability Network of a Real Life Production System .....	I-14
Figure I.7 Connection Matrix of the Real Life Reliability System .....	I-15
Figure I.8 Minimal Tie Sets of the Real Life Reliability System.....	I-15
Figure I.9 Minimal Cut Sets of the Real Life Reliability System.....	I-16

---

---

Figure II.1 Structure of System B .....	II-1
Figure II.2 Effect of Optimisation Process on System Reliability and Total Loss in System B .....	II-3
Figure II.3 Optimisation Process of System B .....	II-3
Figure II.4 Crossover Process of System B .....	II-4
Figure II.5 Effect of Mutation Process on Total Loss in System B .....	II-4
Figure II.6 Mutation Process in System B .....	II-5
Figure II.7 Total Search Space Examined By the Optimisation Algorithm for System B .....	II-5
Figure III.1 Structure of System C .....	III-1
Figure III.2 Effect of Optimisation Process on System Reliability and Total Loss in System C .....	III-3
Figure III.3 Optimisation Process of System C .....	III-3
Figure III.4 Crossover Process of System C .....	III-4
Figure III.5 Effect of Mutation Process on Total Loss in System C .....	III-4
Figure III.6 Mutation Process in System C .....	III-5
Figure III.7 Total Search Space Examined By the Optimisation Algorithm for System C .....	III-5
Figure IV.1 Structure of System D .....	IV-1
Figure IV.2 Effect of Optimisation Process on System Reliability and Total Loss in System D .....	IV-3
Figure IV.3 Optimisation Process of System D .....	IV-3
Figure IV.4 Crossover Process of System D .....	IV-4
Figure IV.5 Effect of Mutation Process on Total Loss in System D .....	IV-4
Figure IV.6 Mutation Process in System D .....	IV-5
Figure IV.7 Total Search Space Examined By the Optimisation Algorithm for System D .....	IV-5
Figure V.1 Structure of System A .....	V-1
Figure V.2 Effect of Optimisation Process on System Reliability and Total Loss in System A .....	V-3
Figure V.3 Optimisation Process of System A .....	V-3

---

---

Figure V.4 Crossover Process of System A.....	V-4
Figure V.5 Effect of Mutation Process on Total Loss in System A .....	V-4
Figure V.6 Mutation Process in System A .....	V-5
Figure V.7 Total Search Space Examined By the Optimisation Algorithm for System A.....	V-5
Figure VI.1 Structure of System B .....	VI-1
Figure VI.2 Effect of Optimisation Process on System Reliability and Total Loss in System B .....	VI-3
Figure VI.3 Optimisation Process of System B.....	VI-3
Figure VI.4 Crossover Process of System B.....	VI-4
Figure VI.5 Effect of Mutation Process on Total Loss in System B .....	VI-4
Figure VI.6 Mutation Process in System B .....	VI-5
Figure VI.7 Total Search Space Examined By the Optimisation Algorithm for System B .....	VI-5
Figure VII.1 Structure of System C.....	VII-1
Figure VII.2 Effect of Optimisation Process on System Reliability and Total Loss in System C .....	VII-3
Figure VII.3 Optimisation Process of System C.....	VII-3
Figure VII.4 Crossover Process of System C .....	VII-4
Figure VII.5 Effect of Mutation Process on Total Loss in System C.....	VII-4
Figure VII.6 Mutation Process in System C .....	VII-5
Figure VII.7 Total Search Space Examined By the Optimisation Algorithm for System C .....	VII-5
Figure VIII.1 Structure of System D .....	VIII-1
Figure VIII.2 Effect of Optimisation Process on System Reliability and Total Loss in System D.....	VIII-3
Figure VIII.3 Optimisation Process of System D .....	VIII-3
Figure VII.4 Crossover Process of System D.....	VIII-4
Figure VIII.5 Effect of Mutation Process on Total Loss in System D .....	VIII-4
Figure VIII.6 Mutation Process in System D .....	VIII-5

---

---

Figure VIII.7 Total Search Space Examined By the Optimisation Algorithm for System D.....	VIII-5
Figure IX.1 Structure of System A .....	IX-2
Figure IX.2 Effect of Optimisation Process on System Reliability and Total Loss in System A.....	IX-2
Figure IX.3 Structure of System B .....	IX-4
Figure IX.4. Effect of Optimisation Process on System Reliability and Total Loss in System B .....	IX-4
Figure IX.5 Structure of System C .....	IX-6
Figure IX.6 Effect of Optimisation Process on System Reliability and Total Loss in System C .....	IX-6
Figure IX.7 Structure of System D .....	IX-8
Figure IX.6 Effect of Optimisation Process on System Reliability and Total Loss in System D.....	IX-8

# 1

## RESEARCH INTRODUCTION

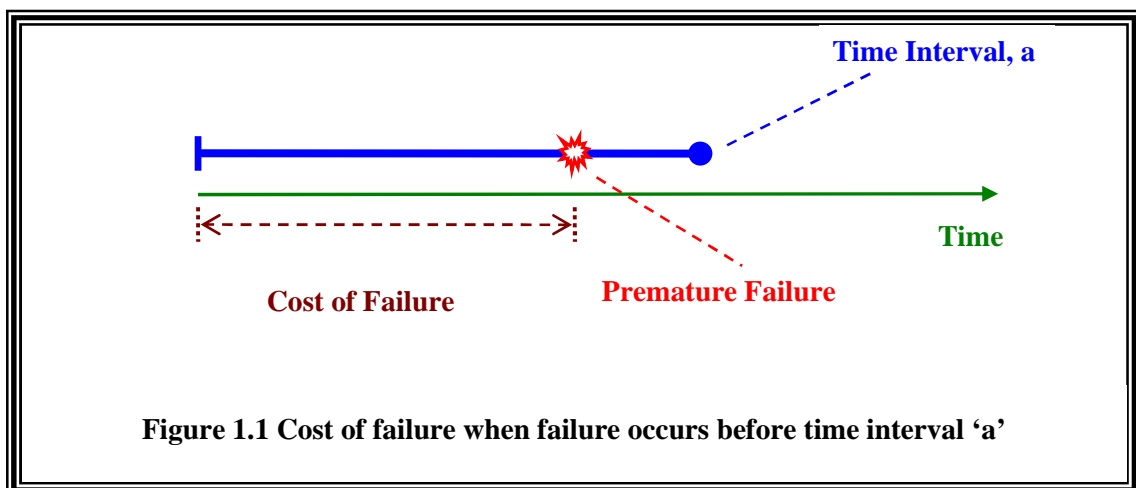
---

This chapter provides an introduction to the research study by detailing the various approaches for system reliability optimisation. The first half describes the two general approaches which are commonly found in the reliability literature along with their mathematical formulation. The second part of this chapter introduces the risk based reliability allocation method and emphasises its viability for using as an appropriate optimisation method for systems associated with high cost of failure by presenting the statement of the optimisation problem studied in this research. The final part of chapter highlights the research aim, objective and methodology.

## 1.1 INTRODUCTION

*“For industries characterised by a high cost of failure, the process of setting quantitative reliability requirements should be driven by the cost of failure”  
(Todinov, 2004)*

A product (system) associated with unacceptably large investment cost and high risk of premature failure before some specified time interval ‘ $a$ ’ (Fig. 1.1), can have detrimental impacts on overall business performance due to the extremely high losses from failures. For this reason, product reliability is of great importance to both manufacturers and buyers. The losses can be more expensive yet damaging for the reputation (and existence) of a manufacturer, if they are generated by the lost production time, amount of lost production, mass of released harmful chemicals in the environment, number of fatalities, lost customers, warranty payments, costs of mobilisation of emergency resources, insurance costs etc. For example, in sub sea oil and gas production systems, major components of the losses from failures are the amount of lost production time that is directly related to the volume of lost production, the cost of intervention and the cost of repair/replacement. Additionally, in an event of released harmful chemicals and oil in the sea, the cost of failure can be enormous. Similarly, in the aerospace industry for example, the cost of failure is also very high and in most instances, impossible to recover; therefore, reliability allocation must be driven in view of these critical failures and associated costs.





## 1.2 GENERAL APPROACH FOR SETTING QUANTITATIVE RELIABILITY REQUIREMENTS

The traditional approach for setting the quantitative reliability requirements in engineering systems (oil and gas production units, for example) is based mainly on achieving high *availability* targets. As a result, it does not necessarily guarantee a small *risk of failure* associated with premature system failures, as shown by Todinov (2004). Even at a very high availability, associated with a particular time interval ‘*a*’, the probability of a premature failure can be very large, consequently, increasing the cost of the failure or total losses (warranty costs, for example). The reliability requirements must therefore guarantee not only a high availability target, but also a low probability of premature failure and, consequently, a low risk of failure for minimising the amount of total losses.

Traditionally, the ‘*cost*’ factor is often considered as a parameter in terms of ‘price for achieving the required level of reliability’ for a given system. In other words, it is an allocated capital cost (budget) which includes the cost of implementing and operating a reliability program in addition to the overall development and production cost associated with the product. It mainly consists of direct material and labour costs as well as indirect costs such as taxes, insurance, energy, production facilities & equipment, and overhead costs such as administrative, marketing and product development costs. Allocated capital cost is generally (but not always) an increasing function of reliability, not only because more organisational resources must be committed to achieve a higher reliability, but also because the material and production costs of the product must increase as well. This may be a result of more costly parts selection, added redundancy, stricter tolerances, excess strength, and increased quality control and inspection sampling during the manufacturing process. As a result, allocated capital cost can be considered as an investment towards achieving the desired reliability.

In mathematical terms, system reliability ‘ $R_s$ ’ is related to component reliabilities ‘ $R_i$ ’, through a function given by,

$$R_s = \xi ( R_1, R_2, \dots, R_M ) \quad (1.1)$$

(Where  $1 \leq i \leq M$  for ‘ $M$ ’ number of components in a system)

Let ‘ $C_i(R_i)$ ’ denote the cost of component ‘ $i$ ’, which has reliability ‘ $R_i$ ’. This is generally (not always) an increasing function of ‘ $R_i$ ’, implying that the cost increases with increasing component reliability. The total system cost can be shown as,

$$C = \sum_{i=1}^M C_i(R_i) \quad (1.2)$$

where,

$$\{ R_s \}_1^M = \{ R_1, R_2, \dots, R_M \} \quad (1.3)$$

In view of the above relationships in equations (1.1-1.3), the common objectives of the existing reliability optimisation approaches appear to be:

- ***For a pre-defined level of system reliability, minimize the total cost of resources, required to achieve this reliability level***

The objective here is to determine the optimal reliability allocation so that the system reliability is at least ‘ $R_{Min}$ ’ and the total system cost ‘ $C$ ’ is minimised. Thus we have the following optimisation problem,

$$\min_{\{R\}_1^M} J = \sum_1^M C_i(R_i) \quad (1.4)$$

subject to the constraints

$$\begin{aligned} \xi(R_1, R_2, \dots, R_M) &\geq R_{Min} \\ 0 &\leq R_i \leq 1 \end{aligned} \quad (1.5)$$

- ***For a given budget allocated for a particular project, achieve maximum level of system reliability***

In the second case, the objective is to determine the optimal reliability allocation that maximises the system reliability ‘ $R_s$ ’ subject to the total cost ‘ $C$ ’ not exceeding some pre-specified budget value ‘ $C_{Max}$ ’. This results in the following optimisation problem,

$$\max_{\{R\}_1^M} J = \xi(R_1, R_2, \dots, R_M) \quad (1.6)$$

subject to the constraints

$$\begin{aligned} \sum_1^M C_i(R_i) &\leq C_{Max} \\ 0 &\leq R_i \leq 1 \end{aligned} \quad (1.7)$$

The fundamental notion of system reliability is described in the next chapter along with other approaches for system reliability optimisation besides the two very common types mentioned above. A very highly regarded literature review on this topic is provided by Kuo *et al.* (2000, 2001).

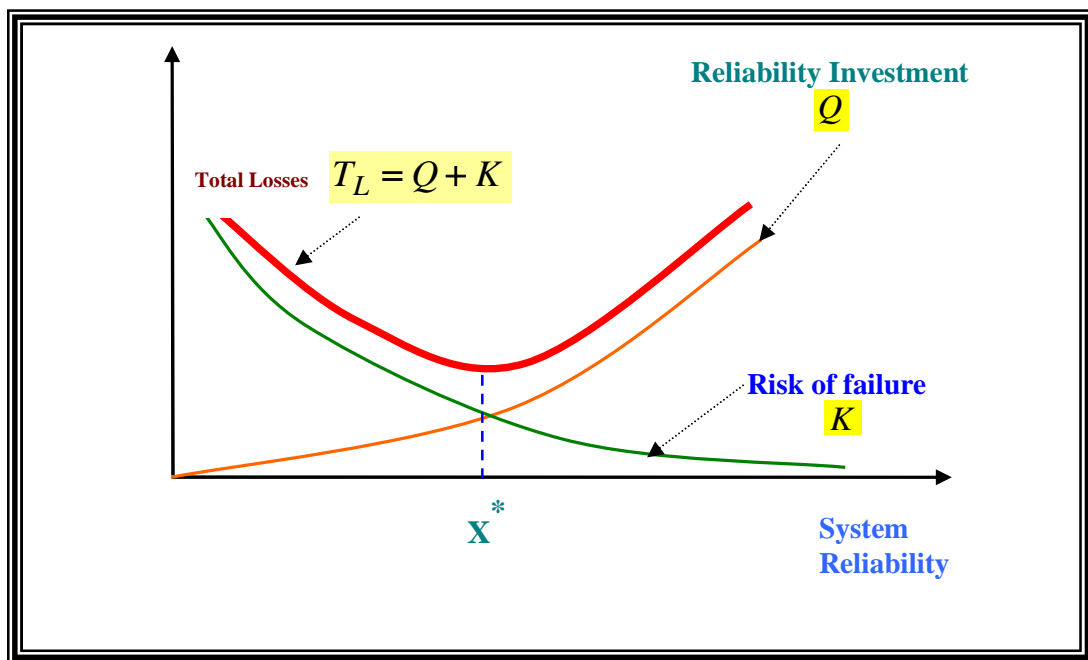
### 1.3 RISK BASED APPROACH FOR SETTING QUANTITATIVE RELIABILITY REQUIREMENTS

It is often seen that removing a failure mode at a design stage is significantly less expensive compared to removing it during service. Thus, it is vital that reliability is integrated early into the design of complex systems associated with high risk of failure. In order to achieve this, reliability engineers must be able to estimate the total losses from system failure or the cost of unreliability. This creates the opportunity to identify the inappropriate design solutions associated with large risk of failure and select an optimal solution characterised by minimum total losses from premature system failures.

This requires a system reliability analysis based on the total losses from failures. A powerful approach is proposed by Todinov (2004, 2006) for system reliability analysis based on minimising the *total loss* ' $T_L$ ' which is the sum of the *reliability investment* ' $Q$ ' and the *risk of failure* ' $K$ ' and also known as the *risk based reliability allocation* method.

$$T_L = ( Q + K ) \quad (1.8)$$

This approach is very useful for comparing the expected risk of failures associated with the competing solutions along with the capital and operational costs associated with them, as shown in Fig. 1.2. The point ' $X^*$ ' represents the optimal value of the system reliability as it is connected with the minimum value which exists on the total loss curve.



**Figure 1.2 Risk Based Reliability Allocation Method**

The risk based reliability allocation method is not about setting the highest level of system reliability (or availability), as observed in the conventional methods to date. Instead, the crux of this approach is to allocate reliabilities to the components of a system such that the total losses are minimised. The optimisation process is driven solely by the amount of total losses because maximising the reliability of the system does not necessarily mean minimum expected losses from failures. Increasing the reliability inappropriately can also increase the losses from failures despite reducing the probability of failure. (Todinov, 2004)

In view of the recently published work of Todinov (2004, 2006), it can be seen that the novel risk based reliability analysis offers a new generation of modelling technique for allocating system reliability particularly for industries associated with a high cost of failure. The aim of this research is to extend this work by demonstrating its application on complex, large scale non-repairable engineering systems, with a large choice of components alternatives. The statement of the optimisation problem is stated next.

### 1.3.1 Statement of the Optimisation Problem

For a system with specified reliabilities and corresponding costs for all alternatives, as in matrices ‘ $\gamma$ ’ and ‘ $C$ ’ respectively, the problem is to select an optimal set of alternatives such that the total loss from system failure is minimum. The statement of this problem was first presented in Todinov (2005) and is yet to be solved by developing a risk-based reliability allocation method for a non-repairable system.

$$\gamma = \begin{bmatrix} \gamma_{11}, \gamma_{12}, \gamma_{13} \dots\dots\dots, \gamma_{1N} \\ \gamma_{21}, \gamma_{22}, \gamma_{23} \dots\dots\dots, \gamma_{2N} \\ \gamma_{31}, \gamma_{32}, \gamma_{33} \dots\dots\dots, \gamma_{3N} \\ \dots\dots\dots \\ \dots\dots\dots \\ \gamma_{M1}, \gamma_{M2}, \gamma_{M3} \dots\dots\dots, \gamma_{MN} \end{bmatrix}$$

$$C = \begin{bmatrix} C_{11}, C_{12}, C_{13} \dots\dots\dots, C_{1N} \\ C_{21}, C_{22}, C_{23} \dots\dots\dots, C_{2N} \\ C_{31}, C_{32}, C_{33} \dots\dots\dots, C_{3N} \\ \dots\dots\dots \\ \dots\dots\dots \\ C_{M1}, C_{M2}, C_{M3} \dots\dots\dots, C_{MN} \end{bmatrix}$$

For a given system, the problem reduces to selecting ‘ $M$ ’ optimal alternatives from each row of the reliability matrix,  $(\gamma_{1,1}^*, \gamma_{2,2}^*, \dots, \gamma_{M,N}^*)$  such that the total loss function, ‘ $T_L$ ’ is minimised.

### 1.3.2 Optimisation Diagram

The optimisation process has also been shown diagrammatically in Fig. 1.3 and 1.4.

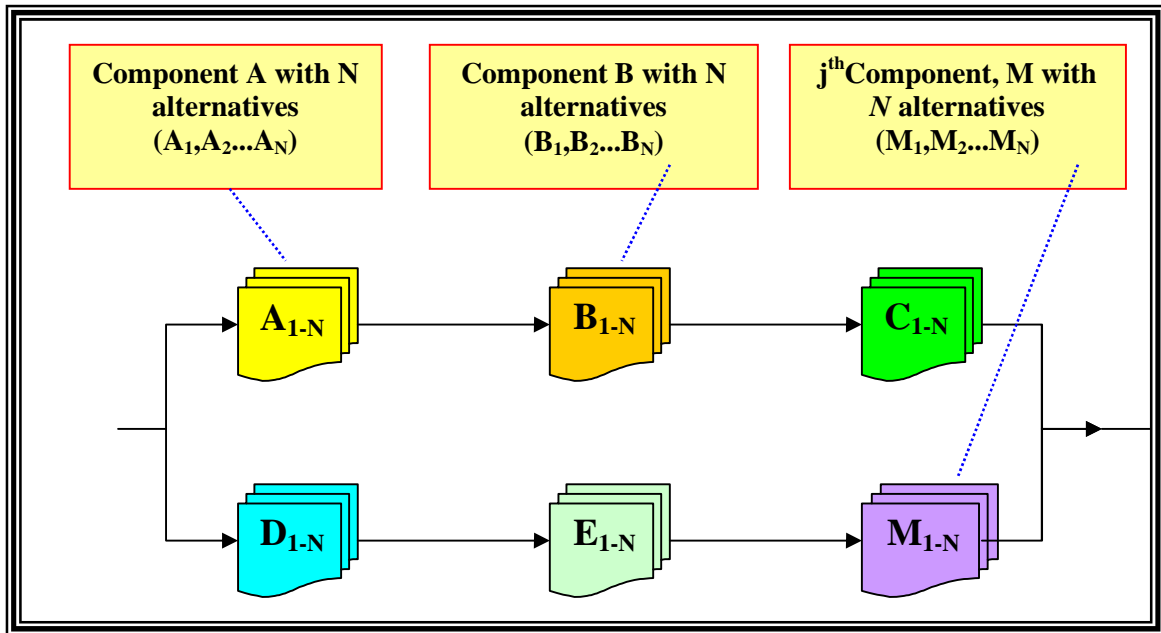


Figure 1.3 Before Optimisation: System with M Components with N alternatives

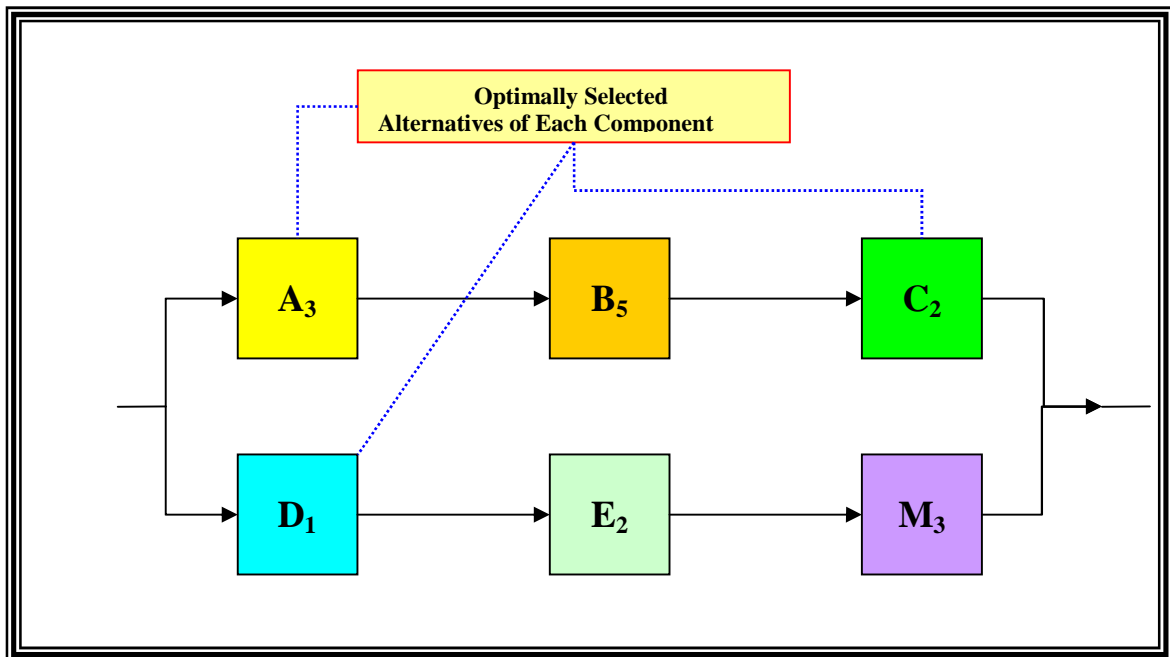


Figure 1.4 After Optimisation: System with M Components with N alternatives

## 1.4 RESEARCH AIMS

In view of the recently published work of Todinov (2004, 2006), it can be seen that the novel risk based reliability analysis offers a new generation of modelling technique for allocating system reliability particularly for industries associated with a high cost of failure. The aim of this research is to extend this work by demonstrating its application on complex, large scale non-repairable engineering systems. The types of systems which are considered in this research are assumed to be consisting of many components such that each component has a large choice of distinctive alternatives with different reliabilities and corresponding costs, which makes the application process to be a very difficult combinatorial optimisation problem.

Consequently, for a given reliability system, consisting of many subsystems, the problem is to allocate appropriate level of system reliability by selecting optimal combination of alternatives from the available choices such that the amount of total losses ' $T_L$ ', sum of the *reliability investment* ' $Q$ ' and the *risk of failure* ' $K$ ', associated with a non-repairable system failure, is minimum.

$$T_L = (Q + K) \quad (1.9)$$

The formal statement of the optimisation problem has already been presented in Chapter 1, Section 1.3.1. The mathematical formulation of the optimisation problem is, however, presented in the next section.

### 6.1.1 Mathematical Formulation

According to a commonly accepted equation (Henley & Kumamoto, 1981), the risk of failure ' $K$ ', has been defined as the product of the probability of failure ' $p_f$ ' and the loss ' $C$ ', given that failure has occurred.



$$K = p_f \bar{C} \quad (1.10)$$

For ‘ $M$ ’ components in a system with ‘ $N$ ’ alternatives for each component, the aim is to allocate the system reliability, ‘ $R_s$ ’ through the function,

$$Opt(R_s) = \xi \left( R_{1(\gamma_{1,1} \dots \gamma_{1,N})}, R_{2(\gamma_{2,1} \dots \gamma_{2,N})}, \dots, R_{M(\gamma_{M,1} \dots \gamma_{M,N})} \right) \quad (1.11)$$

by minimising the total loss function from equation (1.9).

In equation (1.11), the system reliability ‘ $R_s$ ’ is a function of ‘ $M$ ’ components, each with ‘ $N$ ’ number of given alternatives, such that ‘ $\gamma_{ij}$ ’ is the reliability of the ‘ $i^{th}$ ’ component with ‘ $j^{th}$ ’ alternative, ‘ $T_L$ ’ is the total loss from system failure before some specific time interval ‘ $a$ ’.

$$Q = f_1(c_{1,1}, c_{2,2}, c_{3,3}, \dots, c_{M,N}) \quad (1.12)$$

$$Q = \sum_i^M c_i \quad (1.13)$$

$$K = f_2(\gamma_{1,1}, \gamma_{2,2}, \gamma_{3,3}, \dots, \gamma_{M,N}) \quad (1.14)$$

In above equations, ‘ $Q$ ’ is the cost of reliability investment towards risk reduction and is a function of component costs for all selected alternatives (equation, 1.1), such that ‘ $c_{ij}$ ’ is the cost of the ‘ $i^{th}$ ’ component with ‘ $j^{th}$ ’ alternative, where  $i = (1, 2, 3, \dots, M)$ ,  $j = (1, 2, 3, \dots, N)$ , and ‘ $Q$ ’ is equal to the sum of the cost of selected alternatives - equation (1.13). ‘ $K$ ’ in equation (1.14) represents the risk of failure and is a function of the component reliabilities for all selected alternatives.

Using equation (1.14), we can rewrite equation. (1.10) as,

$$K = P_f(\gamma_{1,1}, \gamma_{2,2}, \gamma_{3,3}, \dots, \gamma_{M,N}) \times \bar{C} \quad (1.15)$$

$$K = [1 - R_s(\gamma_{1,1}, \gamma_{2,2}, \gamma_{3,3}, \dots, \gamma_{M,N})] \times \bar{C} \quad (1.16)$$

where,

$$\bar{C} = p_{f1} \bar{c}_1 + p_{f2} \bar{c}_2 + \dots + p_{fM} \bar{c}_M \quad (1.17)$$

and ‘ $p_f$ ’ is the probability of failure, ‘ $R_s$ ’ is the system reliability, ‘ $\bar{C}$ ’ is the expected cost given failure of the system & ‘ $p_{fi} \bar{c}_i$ ’ is the risk of failure of the ‘ $i^{th}$ ’ component.

## 6.1.2 Objective Function

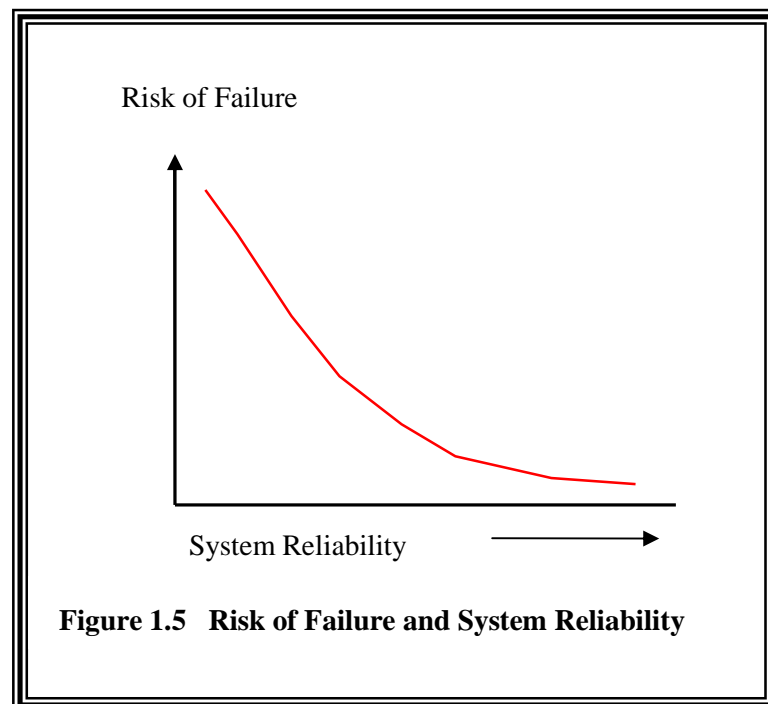
The objective function can thus be derived by substituting the values of equation (1.13) and equation (1.16) into equation (1.9). As a result, we get,

$$T_L = \text{Min} \left( \sum_i^M c_i + [1 - R_s(\gamma_{1,1}^*, \gamma_{2,2}^*, \dots, \gamma_{M,N}^*)] \times \bar{C} \right) \quad (1.18)$$

Where ‘ $\sum_{i=1}^M c_i$ ’ is the total cost of the selected alternatives, ‘ $R_s(\gamma_{1,1}^*, \gamma_{2,2}^*, \dots, \gamma_{M,N}^*)$ ’ is the reliability of the system with optimal (\*) set of selected alternatives and ‘ $\bar{C}$ ’ is the expected cost given failure before a specified time interval ‘ $a$ ’ associated with the selected alternatives.

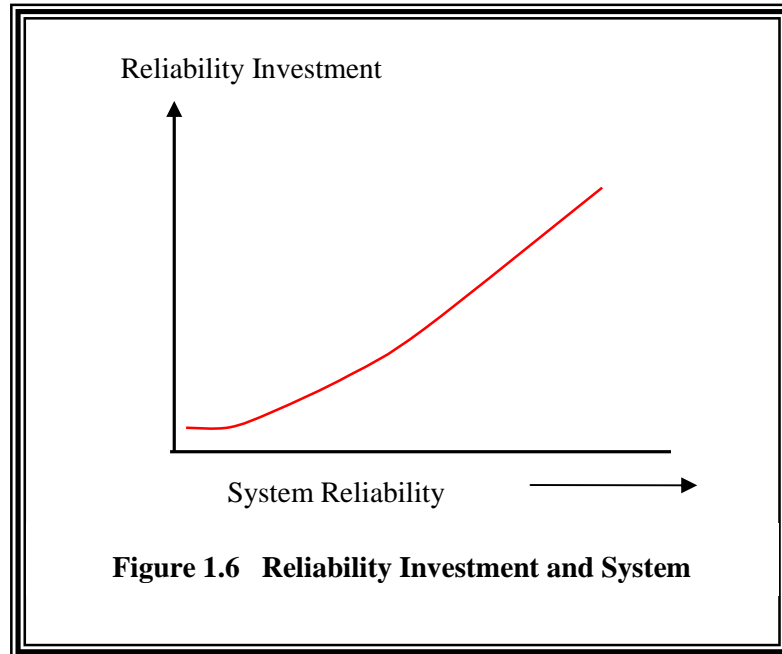
## 1.5 RESEARCH OBJECTIVES

The optimisation function in equation (1.18) shows that the total losses can be minimised, fundamentally by either decreasing the probability of failure ' $P_f$ ' of the system for reducing the risk of failure amount (equation 1.15. Fig. 1.5) or by using cheaper components in order to reduce the cost towards reliability investment, as shown in equation (1.12-1.13) and Fig. 1.6. The two options makes the optimisation process very interesting since reducing the probability of failure requires the use of components with higher reliabilities which automatically increases the cost towards reliability investment, in general. Similarly, opting for cheaper components would imply the complete opposite of the first option since it will increase the probability of failure due to a decrease in component reliability with the individual component cost (if a monotonically increasing function is assumed between cost and reliability).



The process of loss function minimisation becomes even more complex when the relationship between cost-reliability is considered to be non-monotonically increasing and when a very large choice of components alternatives is available. This

is because many combinations of components can exist which can produce minimum values of the loss function when selected together without showing any obvious pattern to the ones depicted in Fig, 1.5 and 1.6.



Since the aim of this research study is to demonstrate the risk based reliability allocation method at component level in non-repairable systems, consisting of many components with a large choice of available alternatives (irrespective of cost-reliability relationship), the primary objective is to develop an efficient optimisation technique which can be used effectively to minimise the loss function from equation (1.18) for such systems. This can be outlined in the following points:

- Allocate reliability at a component level in order to minimise the total losses: the sum of the cost of the component and the risk of premature failure.
- Be able to search for optimum solution in all instances, particularly when the evaluation of all possible solutions is impractical by using enumerative techniques, such as exhaustive search method.
- Perform loss function minimisation for all relationships between cost and reliability which may exist among the alternatives of the components.

- Perform optimisation of the topology of systems with respect to the risk based reliability allocation method.

The overall objective of the optimisation process is to ensure that the given system is both *economically reliable* and *reliably safe*. Economically reliable means the system's observed reliability has been established with consideration of the minimum total losses from premature failure. Reliably safe means designing sufficient reliability into the system to ensure that the probability of premature failure is within an acceptable limit. How much reliability should be designed into a system depends significantly on the level of the acceptable amount of total loss.

## 1.6 RESEARCH METHODOLOGY

For ' $M$ ' components with ' $N$ ' alternatives each, the search space ' $(\Omega)$ ' can be very large ( $N^M$ ) even for small and moderately sized problems. Additionally, the non-linear relationship between cost and reliability, as explained in Guikema and Pate-Cornell (2002) and in Chapter 6, demands efficient exploration of the solution space for these types of optimisation problems since any combination of component can produce optimal or near optimal solution without demonstrating any obvious and straightforward pattern. In the context of solving reliability optimisation problems, the method of using genetic search (detailed in Chapter 3 & 4) has been widely employed due to its robustness and capability to efficiently explore and exploit the search space (Gen and Cheng, 1997, 2000; Levitin, 2006; Smith, 2006). A solution for the optimisation problem studied in this research is therefore, proposed by using an efficient optimisation technique (detailed in chapter 5), which is inspired by the notion of the evolutionary algorithms (EA), detailed in chapter 3. Being a population based method with embedded variation operators mimicking the phenomenon of evolution of life in the natural science, the selected methodology provides a very useful tool in studying the very large search space of the optimisation process and evaluating the complex structures of the cost-reliability combinations for possible

exploration of the optimum solutions. While chosen method does not guarantee the absolute optimum, the quality of the results generally obtained deserve the title of 'optimal' or 'near optimal'. The core features of the proposed methodology are listed below:

- The methodology has the ability to estimate system reliability of complex reliability networks and risk of failures, by means of a software tool based predominantly on a powerful method designed by Todinov (2006, 2006a). Using the state of the art Monte Carlo simulation technique, the algorithm provides generic applicability for all types of complex systems and is fully capable of determining the reliability of any given system with associated amount of total losses. It is useful to point out that all optimisation problems studied in this research have also been tested with another reliability estimation method (Appendix I) which is similar to the method introduced by Todinov. However, it has not been tested for problems which exist outside the scope of this research but given the excellent quality of the comparative results, it is deemed as having great potential for future studies.
- The methodology is capable of skilfully selecting an optimal combination of alternatives which best minimises the total loss function. This process consists of randomly selecting possible solutions from the large search space and competently exploiting the good solutions for exploring even better solutions. Besides the optimum solution, the methodology is able to produce a list of other sub-optimal results which can be used to effectively undertake parametric studies for future studies.



# 2

## **SYSTEM RELIABILITY OPTIMISATION** *(CONCEPTS AND MODELS)*

---

This chapter reviews some of the important concepts used in this research along with the literature found in the field of reliability optimisation. The first half of the chapter details the fundamentals of system reliability, system configurations, methods for estimating system reliability and some widely known probability distributions. The remaining half of the chapter conducts a concise review of the reliability optimisation process using various models and techniques along with the specific literature review of risk based reliability allocation method. The final section of the chapter provides general review of some of the widely known areas of reliability optimisation.

## 2.1 SYSTEM RELIABILITY

The reliability of a system (or a component) is generally referred to as its ability to perform specified task, under specific conditions of use and during a specified interval of time, 't'. In other words, system reliability, being a function of time, 'R(t)', articulates the notion of dependability, successful operation or performance and most importantly, the absence of failures. Because the process of deterioration (e.g. crack occurrence and propagation in a surface of a component) leading to system failure arises in an uncertain manner, the notion of reliability necessitates a dynamic and probabilistic framework. Thus, it can be measured in terms of a probability, 'P(T > t)' that a system or a component will continue to work during the specified time interval, 't' before failing eventually in time, 'T', such that 'T', is a continuous random variable and 'T > t'; for this reason, it is also known as survival function (Kaufmann *et al.*, 1977).

$$R(t) = P(T > t) \quad (2.1)$$

For non-repairable systems, the above relationship is restricted to the time interval to the first failure of the system, whilst for those that can be repaired, all time intervals between successive failures must be considered. The reliability function is a monotone non-increasing function which is unity at start of life and gradually reduces towards zero as the time increases to infinity. The complement of 'R(t)' is the cumulative distribution function of failures 'F(t)' which is associated with reliability by the following relationship,

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

Using the similar approach as in equation (2.1) and continuous random variable 'T', the 'F(t)' gives the probability that the time to failure 'T' will be smaller than the specified time interval 't',

$$F(t) = P(T \leq t) \quad (2.3)$$

The probability density function of the time to failure is denoted by 'f(t)', which describes how the failure probability is spread over time. In the infinitesimal interval



' $t$ ', ' $t + dt$ ', the probability of failure is ' $f(t)dt$ '. For any specified time interval ' $t_1 \leq T \leq t_2$ ', the probability of failure can be estimated as,

$$P(t_1 \leq T \leq t_2) = \int_{t_1}^{t_2} f(t) dt \quad (2.4)$$

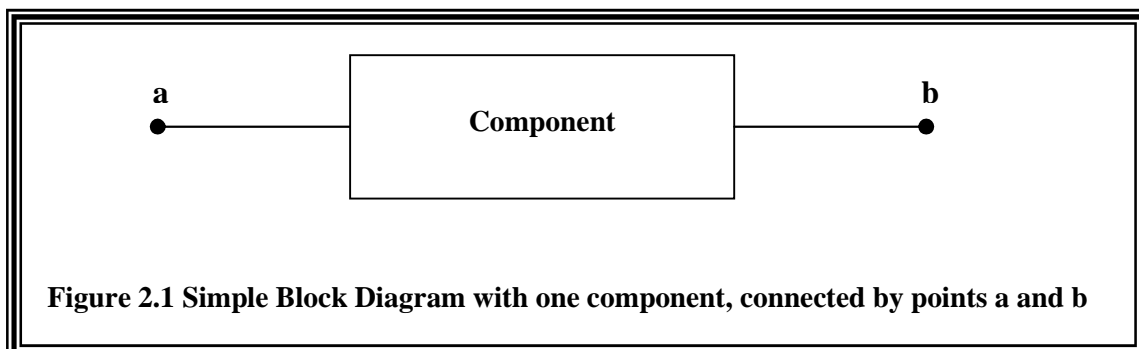
Because ' $f(t)$ ' is a probability distribution, the values of ' $f(t)$ ' are always non-negative and the total area beneath ' $f(t)$ ' is always equal to one. Also, it is related to the cumulative distribution function ' $F(t)$ ' by the following relationship,

$$f(t) = dF(t) / dt \quad (2.5)$$

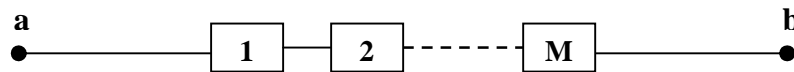
A comprehensive discussion related to the basic reliability functions has been provided by Kaufmann *et al.* (1977) and Grosh (1989). Also, the mathematically oriented explanations of the reliability theory are detailed in Barlow and Proschan (1965, 1975) and Catuneanu *et al.* (1989).

### 2.1.1 Basic Configurations of Reliability Systems

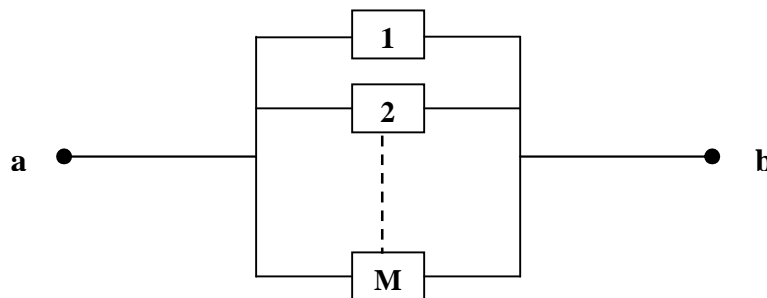
Most often, a system is found to be composed of more than one component (i.e. a multi-component system). In these cases, the reliability of the system becomes the function of the individual components reliabilities and the method in which these components are arranged for assembling the system. There are two fundamental configurations in which a system can be constructed. These are series configuration and parallel (or redundant) configuration. Each of these can be shown by means of a reliability block diagram (RBD). Each component in a RBD is represented by a block with two end points or nodes, ' $a$ ' (source) & ' $b$ ' (sink), which are connected only if the component is in its working state, otherwise this connection is broken, when the component is in failed state - Fig. 2.1.



A multi-component system can be represented as a network of such blocks with two end points. The system is considered in working state if there is a connection between the source and sink nodes, otherwise, it is considered in the failed state because of no path between the two nodes. Therefore, using RBDs, a system with ‘ $M$ ’ number of components in series configuration can be shown in Fig. 2.2, whereas, the parallel configuration of the system with the same number of components, is shown in Fig. 2.3.



**Figure 2.2 Reliability Block Diagram of a System in Series Configuration**



**Figure 2.3 Reliability Block Diagram of a System in Parallel Configuration**

In series configuration, all ‘ $M$ ’ components in a system must be in working state in order for the system to be in operation. If each component is represented by ‘ $A_i$ ’ such that  $1 \leq i \leq M$ , then the reliability of the system with series configuration can be represented as,  $R_s = \text{Probability}(\text{All components are working})$ .

Mathematically,

$$R_s = P(A_1 \cap A_2 \cap \dots \cap A_M)$$

$$R_s = \prod_{i=1}^M P(A_i) \quad (2.6)$$

Where ' $P(A_i)$ ' is the probability that component ' $A_i$ ' is in working state.

On the other hand, a system with parallel configuration will be in a failed state if all ' $M$ ' components in the system are not working (failed). Using the same notation of ' $A_i$ ' for components, the reliability of the system with parallel configuration can be represented as,  $R_s = \text{Probability}(\text{All components are failed})$ .

Mathematically,

$$R_s = P(A_1 \cup A_2 \cup \dots \cup A_M)$$

$$R_s = 1 - \prod_{i=1}^M [1 - P(A_i)] \quad (2.7)$$

Using these two fundamental configurations, a reliability system can also be consisting of the combined structures of both series-parallel and parallel-series configurations. The system reliability for such system is computed by breaking the system down to smaller units of series and parallel structures. Additionally, there are many other types of configurations (Kuo *et al.*, 2001) among which the two commonly known in the reliability literature are ' $k$ -out-of- $M$ ' systems and 'complex systems'. The former is a configuration in which a system is in working state only if at least ' $k$ ' of its ' $M$ ' components are operating without failure (similar to parallel systems). In 'complex configuration', the structure of the arrangement of components is neither in series nor in parallel. An example of such a system is shown in Fig. 2.4. The reliability of a complex system can be computed using various analytical techniques such as Inspection Method, Event Space Method, Path Tracing Method, Decomposition Method, Adjacency Matrix Method, Matrix Multiplication Method, Tree Based Method, and most commonly known Tie-Set and Minimal Cut-Set Methods. An excellent overview of these methods is provided by Billinton & Allan (1992), Ramakumar (1993), Ebeling (1997), Blischke & Murthy (2000) and

Todinov (2006a). Also, a practical demonstration of the Tie-Set and Cut-Set method has also been shown in Appendix I for an example bridge system and software using the algorithm from Fotuhi-Firuzabad *et al.* (2004), who also provide comparative review of various methods listed above.

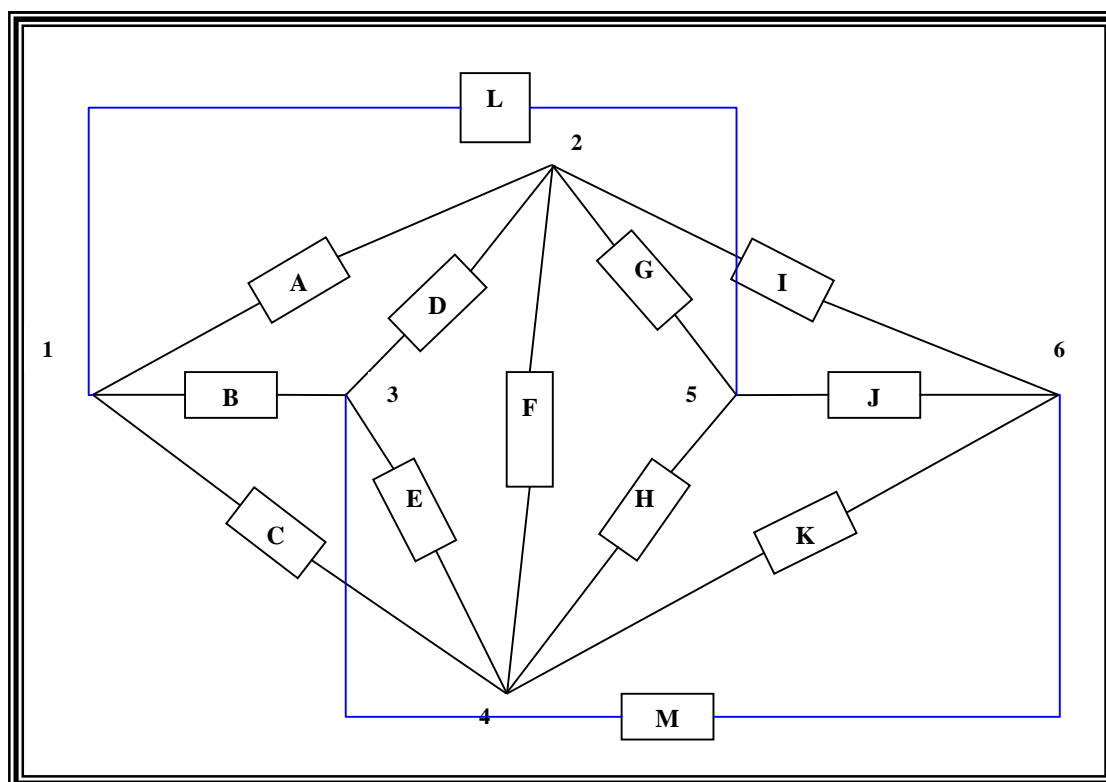


Figure 2.4 Example of a Complex System

### 2.1.2 System Reliability Computation of Complex System

For a large scale system with more complex configurations (than the system shown in Fig 2.4, for example) the analytical method for computing system reliability can be a very difficult and cumbersome process with a strong possibility that these methods may not even work with great success because of the computation times which can grow exponentially with the size of the reliability system. There are many alternative methods proposed by researchers for counteracting this problem which is found in the literature. Among these, are the methods of Hanzhong & Dongkui (1994) for computing complex system reliability using advancements in cut sets approach, Ball

& Van Slyke (1977) introduced modified cut-set enumeration method by backtracking to compute the system reliability, Mandaltsis & Kontolen (1987) provided a method for calculating system reliability using hierarchical routing strategies and Liu *et al.* (1995), who also suggested a different approach for computing system reliability.

Another interesting type of solution for computing reliability of complex systems adopted by the researcher in recent years is the use of Monte-Carlo (MC) simulation technique. Simulation methods are appropriate for large scale reliability systems because the computation time does not grow as steeply as observed for analytical methods for the same systems and also due to the latest advancements in computer technology; faster processing machines are available which can be used to optimise the runtime of the simulation algorithm. There are many publications in this field detailing the use of this simulation method with many variations in the individual approaches. Among many, some of the commonly found methods are Kamat & Riley (1975); used even based MC simulation for determining the system reliability, Kumamoto *et al.* (1977); suggested an evaluation method for determining the system reliability, Fishman (1986(a)(b)); studied the use of MC in depth and provided four methods for establishing the path between the source and sink nodes of a reliability system, Yeh *et al.* (1994); provided a new approach for using MC for evaluating system reliability and Cancela & Khadiri (1995) studied the communication network and suggested recursive variance-reduction technique with MC for computing system reliability. A simple yet very powerful method using MC for evaluating reliability of system consisting of components which are not arranged logically in series or parallel configuration is proposed by Todinov (2006, 2006a). The method is generic, uses adjacency matrix and node-stacking technique for exploring the valid path between the source and sink nodes and is very easy to program. In Appendix–I, another method for estimating system reliability is introduced in this research which is very similar to Todinov’s approach. Both methods use adjacency matrix for representing a reliability network however the process of navigating through the matrix are different in the two methods. While the second method has been used successfully for many complex reliability system evaluations during this research, it is important to point out that it is not thoroughly investigated in view of the generic application of this method across all networks. However, given the excellent results obtained using this method

to date, it is very probable that it can be used with greater success in other areas of the reliability system analysis. A comprehensive literature on various new methods is provided by Todinov (2006a).

### 2.1.3 Common Probability Distributions for Modelling Time to Failure

The computation of system reliability is about estimating the time to failure, ' $T$ ', which is a continuous random variable. The uncertainty associated with ' $T$ ' can be described by using an appropriate cumulative probability distribution function of system failures ' $F(t)$ ' which characterise the probability ' $P(T \leq t)$ ', as defined in section 2.1 above. There are many probability distributions which are found in the literature for modelling the descriptive characteristics of the continuous random variable ' $T$ '. Some of the most widely known distributions are detailed briefly in this section.

#### 2.1.3.1 Exponential Distribution

This distribution is also known as negative exponential distribution in reliability literature and used for modelling the assumption of constant failure rate or hazard rate, ' $\lambda$ ' which is an instantaneous rate of failure such that ( $\lambda > 0$ ). This means, that the probability that a system (or component), having survived time ' $t$ ' will fail within a short time interval, ' $t, t + \Delta t$ ' is constant. The probability of failure is therefore ' $\lambda \Delta t$ ' and is independent of the age of the system. The probability distribution modelling the life of the system can be shown to have the exponential distribution (negative) using this assumption of constant failure rate ( $\lambda \Delta t$ ).

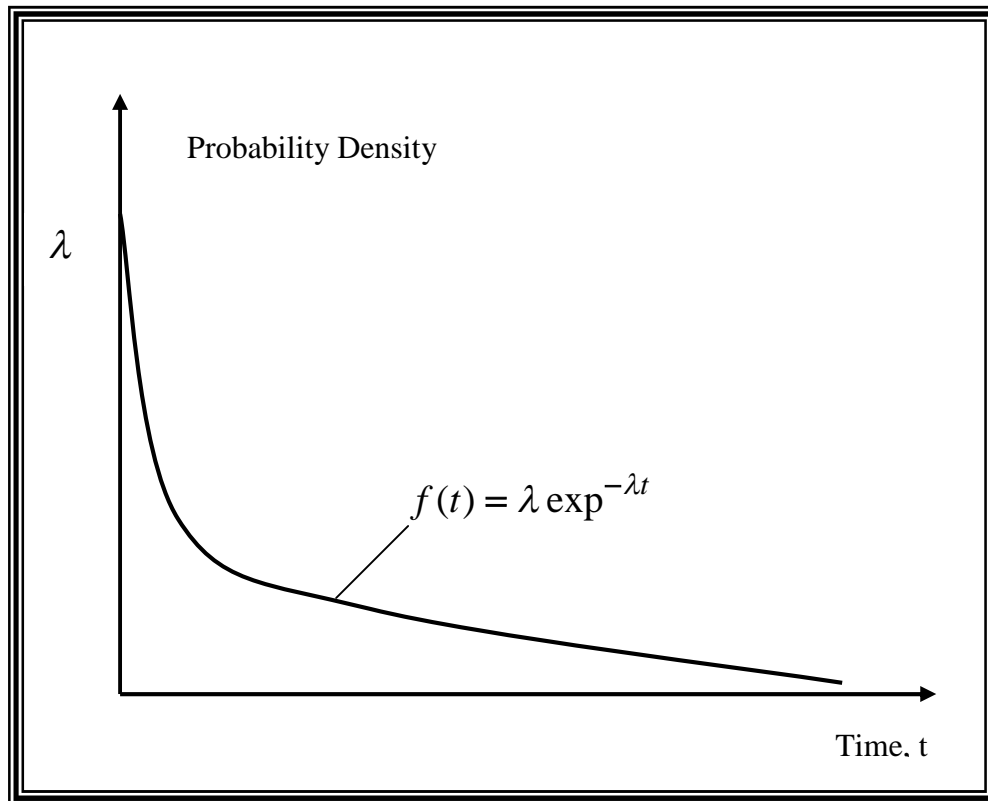
$$F(t) = 1 - \exp^{-\lambda t} \quad (2.8)$$

Using equation (2.2), the above relationship can also be shown for the survival function or the system reliability, ' $R(t)$ '

$$R(t) = \exp^{-\lambda t} \quad (2.9)$$

The probability density function of the time to failure using the equation (2.5) would therefore be equal to,

$$f(t) = \lambda \exp^{-\lambda t} \quad (2.10)$$



**Figure 2.5 Exponential Distribution**

The mean of the distribution is  $\mu = 1/\lambda$ , also known as mean time to failure (MTTF), for constant hazard rate and standard deviation is  $\sigma^2 = 1/\lambda^2$ . If the hazard rate increases with age of the system instead of being constant, the cumulative distribution function of the time to failure is referred to as increasing failure rate (IFR). For instances where the failure rate decreases with time, it is categorised as decreasing failure rate (DFR). Reliability can be increased by decreasing the failure rate which can also be presented in terms of the probability density function of time to failure, ' $f(t)$ '. From equation (2.10), it can be deduced that the hazard rate ' $\lambda$ ' can be interpreted as the ratio of the ' $f(t)$ ' and ' $R(t)$ '. This relationship also highlights the

fundamental difference between the hazard rate and the failure density,  $f(t)$ . The former is associated with the proportion of components in service that fail per unit interval while the latter is associated with the percentage of the initial number of components that fail per unit interval. The exponential distribution is one of the most widely used failure distributions and is appropriate for instances where the occurrence of failure is random and not age dependent. It is also mathematically tractable in most applications and found to be linked with the Poisson process and distribution as shown explicitly in Todinov (2005).

### 2.1.3.2 Weibull Distribution

The Weibull distribution was first proposed by Weibull (1951) and it is used universally for modelling the times to failure of systems which fail when the weakest component in the system fails. The cumulative distribution of the Weibull distribution is given by

$$F(t) = 1 - \exp^{-(t/\beta)^\alpha} \quad (2.11)$$

' $\beta$ ' is called the scale parameter or characteristic lifetime (Todinov, 2005) and ' $\alpha$ ' is called shape parameter such that both  $\beta, \alpha > 0$ . This distribution is known for its flexibility since by selecting different values of the shaper parameter ' $\alpha$ ' and by varying the scale parameter ' $\beta$ ', a number of shapes can be obtained to fit experimental data; 'shape' defines the graph of the failure rate function, for example, IFR, DFR, bath-tub curve etc. A detailed text on this subject is provided by Blischke & Murthy (2000).

When  $\alpha = 1$ , and  $\beta = 1/\lambda$ , the Weibull distribution transforms into exponential distribution, shown previously in equation (2.8). Also, when  $\alpha > 3$ , the distribution is approximately normal. Using equation (2.2), the above relationship can also be shown for the survival function or the system reliability, ' $R(t)$ '

$$R(t) = \exp^{-(t/\beta)^\alpha} \quad (2.12)$$



The probability density function of the time to failure by using the equation (2.5) and differentiating equation (2.11) with respect to 't' would therefore be equal to,

$$f(t) = \frac{\alpha t^{(\alpha-1)} \exp^{-(t/\beta)^\alpha}}{\beta^\alpha} \quad (2.13)$$

### 2.1.3.3 Other Distributions

The distributions briefly detailed in the previous sections are the two most widely known and used methods for estimating the probability of failure in the industry. Besides these there are many other distributions which are found in the literature. For example, binomial distribution, which is classified as a discrete distribution, extends the Bernoulli distribution (where a random variable takes only two values either '0' or '1' with equal probability) by providing the sum of a certain number (say 'n') of independent Bernoulli random variables (trials) so that its distribution can be obtained as the n-fold convolution of the Bernoulli distribution (Blischke & Murthy, 2000). While the modelling of the failed item in reliability analysis at component level (for example) can be carried out using the Bernoulli distribution such that the random variable is '0' when the component is failed and '1' otherwise for a single trial, the random variable represented by the binomial distribution is generally used to model the number of failed items in 'n' size Bernoulli trials. There are a fixed number of identical trials, which are statistically independent and each of which can result in either success or failure with equal probability. If 'X' is the discrete random variable representing the number of successes in 'n' trials with probability 'p' and failures with probability '1 - p', the binomial distribution can be stated as,

$$f(X = x) = \frac{n!}{x!(n-x)!} p^x (1-p)^{n-x} \quad (2.14)$$

$(x = 0, 1, 2, \dots, n)$

The mean and variance of the distribution are  $\mu = np$  and  $\sigma^2 = np(1-p)$ , respectively. Generally, if  $np > 5$  and  $n(1-p) > 5$ , the binomial distribution can be approximated by a normal distribution using the mean and variance detailed above.

Another commonly known distribution is Poisson distribution which is in fact a limiting case of a binomial experiment with parameters ' $n \rightarrow \infty$ ' and ' $p = \lambda t / n$ ' (' $\lambda$ ' is the number of occurrences (e.g. successes) per unit interval of time ' $t$ '); a binomial experiment with a large number of trials and a small probability ' $p$ ' in each trial can be approximated appropriately by a homogeneous Poisson process with intensity ' $\lambda = np/t$ ' (Todinov, 2005). The probability density function of the Poisson distribution can be derived from the binomial distribution stated in equation (2.14)

$$f(X = x) = \frac{n!}{x!(n-x)!} \left(\frac{\lambda t}{n}\right)^x (1-p)^{n-x} \quad (2.15)$$

Since ' $n \rightarrow \infty$ ' and ' $p$ ' is small, the above equation can be transformed into,

$$f(X = x) = \frac{(\lambda t)^x \exp(-\lambda t)}{x!} \quad (2.16)$$

$x = 0, 1, 2, \dots$

The Poisson distribution is used generally to model the random event such as random failures in non-overlapping intervals of times. The probability of the occurrence of the random event in intervals of the same length is identical and does not depend on the location. The mean and variance values of the distribution are the same in that  $\mu = np = \lambda t$  and  $\sigma^2 = np(1-p) = \lambda t$ , respectively because of the large value of ' $n$ ' and very small value of ' $p$ '.

Besides binomial and Poisson distribution, there are other well known and inter-related distributions which are used in the field of reliability engineering, such as uniform distribution, normal distribution, log normal distribution etc. Two very good sources which provide excellent descriptions of these distributions along with the ones highlighted in the previous section and many others are Blischke & Murthy (2000) and the recently published books of Todinov (2005, 2006a).

## 2.2 RELIABILITY OPTIMISATION

Reliability engineering came into view in the late 1940s and early 1950s and was first applied to communication and transportation systems (Kuo *et al.*, 2001). Much of the early work during those times was limited to the analysis of performance aspects of reliability systems. However, due to the advancement in technology (software and hardware) in the present era, the structure of reliability systems has become highly complex and reliability engineering has, therefore, become increasingly important. Generally, the primary goal of a reliability analyst is to find the best ways to increase system reliability in view of the commonly accepted principles such as,

- *Simplicity* - keeping the system as simple as is compatible with performance requirements
- *Component Reliability* - Increasing the reliability of the components in the system in order to improve the overall system reliability
- *Redundancy Allocation* - Using standby redundancy which is activated to replace the failed unit(s).
- *Repair* - Using repair maintenance where failed components are replaced but not automatically switched in, as in the redundancy allocation principle.
- *Maintenance* - Using preventive maintenance such that components are replaced by new ones whenever they fail, or at some fixed periodical interval, whichever comes first.

Besides these principles, using better improvement management programs and performing burn-in testing on components that have high infant-mortality also provide opportunities among possibly various other principles, for improving the level of system reliability. Because implementation of these principles normally consumes resources, a balance between system reliability and resource consumption is essential in order to increase profits while improving the stability of a given system. This process leads to the natural problem of system reliability optimisation, which has been a very popular subject among researchers in the past few decades. A comprehensive

source detailing the work in this field is provided by Tillman *et al.* (1977, 1980, 1985) and Kuo *et al.* (2000, 2001) along with Jensen (1970), Tzafestas (1980), Misra (1986, 1992), Aggarwal (1993), Gen & Cheng (1997, 2000), Cantoni & Zio (1999), Zio (2000), Kuo & Prasad (2000), Levitin (2007) and Smith (2007), are the most commonly known.

### 2.2.1 Modes of Optimisation

There are various modes of optimisation which are presented in the sizeable reliability literature for dealing with the main issues concerning reliability improvement and controlling the cost of system resources. Among them, below are the most commonly found modes which have been the centre of attention for most of the researchers in this field:

- Improve (maximise) system reliability by adding the redundant components in each specified subsystem.
- Maximise systems reliability by improving the individual components reliabilities in each specified subsystem.
- Minimise the cost of the system while meeting the minimum target of some specified level of system reliability.
- Similarly, minimise the cost of a multi-function system while meeting the minimum target of some specified level of individual components reliabilities.

The term ‘cost’ in the above points is used to represent individual constraints such as cost of sub-systems (components), weight, volume, or some combination of these which are the key factors imposed on systems with series, parallel or complex configurations for performing appropriate system reliability optimisation. A very good source of the general review is provided by Kuo *et al.* (2000, 2001). Each of the above cost constraints is generally considered as an increasing function of the component reliability and/or number of components.

## 2.2.2 Optimisation Models

In view of the optimisation modes listed above, there appear to be many optimisation models which are derived due to the given diversity of system configurations, resource constraints and growing demand for reliability improvements. The majority of these reliability optimisation models can be presented in the following general framework.

1. Allocation of continuous component reliabilities
2. Allocation of discrete and continuous component reliabilities
3. Redundancy allocation
4. Reliability-redundancy allocation
5. Allocation of discrete component reliabilities and redundancies
6. Redundancy allocation for cost minimisation
7. Component assignment
8. Multiple objective optimisation

The objective in all of the above models, except 6 and 8 is to maximise system reliability. These two models are different in that they deal with objectives such as cost, weight, volume etc. A comprehensive list of references for each of the eight optimisation models, along with many others, is provided exclusively in Kuo *et al.* (2001).

## 2.2.3 Classification of System Reliability Optimisation

The importance of the quantitative aspects of reliability arises from the increasing interest resulted from the growing need for highly reliable systems and components which are both safer and cheaper. Reliability experts have focused a great deal of their efforts in allocation of reliability and redundancy of components for maximising the overall system reliability. This approach is essential when there is no possibility for

replacement or repair of failed components during system operation. Nevertheless, in the event of a failure, the catastrophic financial and environmental impacts could easily have disastrous repercussions – hence the approach for ‘maximum reliability without taking into account the cost-of-failure’ would appear to be inappropriate.

Based on Tillman *et al.* (1980) and a recent review by Kuo and Prasad (2000), an overview of system-reliability optimisation can be presented quite comprehensively. In their review, Tillman *et al.* presented a classification of papers on reliability optimisation by system structure, problem type, and solution method. In Kuo and Prasad (2000), the contributions that have been made to the literature since the publication of Tillman *et al.* (1977) are discussed. With reference to Kuo *et al.* (2001), all of the articles on optimisation methods for system reliability can be categorised into three sections:

### **2.2.3.1      *Category One - by System configuration***

In solving reliability optimisation problems, ‘system configuration’ appears to influence a great deal. All of the articles on reliability optimisation, grouped by system configuration (Kuo *et al.* 2001) include systems such as series, parallel, series-parallel, standby & parallel-series. Also included are general network systems together with bridge networks, non-series-non-parallel structures, k-out-of-n system and other complex system configurations

### **2.2.3.2      *Category Two - by Problem Type***

The Bulk of the mathematical formulations for representing reliability optimisation problems are covered in the eight optimisation models, listed above. The methods developed for solving these system reliability optimisation problems target the underlying mathematical structure of the given problem. Each model serves a particular goal such as optimum reliability and/or redundancy allocation, maximisation of system reliability subject to cost constraints, cost minimisation subject to the minimum requirement of system reliability, maximisation of system profit etc. (Kuo *et al.*, 2001)

### 2.2.3.3 *Category Three - by Optimisation Techniques*

As indicated by Kuo *et al.* (2001), the development of heuristic methods and meta-heuristic algorithms for redundancy allocation problems appear to be the major objectives of the recent work of the researchers compared to the work which has been directed toward exact solution methodologies for such problems. Most, if not all, reliability systems considered in this area belong to the class of coherent systems (A system is coherent, when the component reliability improvement does not degrade the system reliability; e.g. a simple series or parallel system. A coherent system has a structure function that is monotonically increasing)

A detailed overview of each of the three categories along with the references of the research works is presented in the widely recognised literature of Kuo *et al.* (2001).

## 2.2.4 **Computational Methods of Optimisation**

There are a number of computational techniques which are used by the researchers in the field of reliability optimisation. Some of the most commonly known techniques are:

- Geometric Programming
- Integer Programming
- Dynamic Programming
- The Discrete Maximum Principle
- The Sequential Unconstrained Minimisation Technique (SUMT)
- The Generalised Reduced Gradient Method (GRG)
- Method Of Lagrange Multipliers And The Kuhn-Tucker Conditions
- The Generalised Lagrangian Functions Method
- Heuristic And Meta-Heuristic Approaches
- And Others (A Classical Approach, Parametric Method, Linear Programming And Separable Programming)

According to Kuo *et al.* (2001), the general set of assumptions made while using these optimisation techniques are:

- Each subsystem is considered essential for the overall operational success of the mission, if all the subsystems are operational in series.
- All the subsystems in series, parallel, or complex configuration are statistically independent. In parallel redundancy, all units have the same probability of failure whether they are spares or active.
- Before the requirement of linearisation for some specific optimisation techniques, the constraints of 'cost' do not need to be in a linear form.
- Good/bad is a sufficient description for each component, sub-system, and the whole system. In parallel cases, unless specified, only one component needs to be good for the subsystem to be good, this is considered to be a one out of m: G configuration. No assumptions are made about the hazard rates of the components, except that they are reflected in the reliability of the components
- Without the specific optimisation knowledge of the mission requirements, realistic decisions on redundancy, design change, and other aspects of reliability improvement cannot be reached. Tradeoffs can be considered only between optimal redundancy components and 'cost' measures.
- The constraints are additive between subsystems.
- The redundant models are based on the assumption that individual component or path failure has no effect on the operation of the surviving paths.
- The connection nodes may accrue some 'cost', but are assumed to function perfectly given the system is working.

Recent developments are based on the following methods as indicated by Kuo *et al.* (2001):

- Heuristics for redundancy allocation, special techniques developed for reliability problems.
- Meta-heuristic algorithms for redundancy allocation, perhaps the most attractive development in the last ten years.



- Exact algorithms for redundancy allocation or reliability allocation (most are based on mathematical programming techniques, e.g. the reduced gradient methods presented in Hwang *et al.* (1979).
- Heuristics for reliability–redundancy allocation, a difficult but realistic situation in reliability optimisation.
- Multiple objective optimisations in system reliability, an important problem in reliability optimisation.
- Optimal assignment of interchangeable components in reliability system, a unique scheme that often takes no effort.
- Others including decomposition, fuzzy apportionment and effort function minimisation.

Most of the system-reliability optimisation problems listed are nonlinear integer programming problems. They are more difficult to solve than general nonlinear programming problems because their solutions must be integer numbers. Several optimisation methods are available in the literature – see (Kuo *et al.*; 2001) – for solving such problems: each of the technique listed has had some success in solving particular reliability optimisation problems. It is impossible to select a single method applicable to solve all reliability optimisation problems since each method can be very different and indeed difficult to configure and customise for all types of optimisation problems. For example, dynamic programming has dimensionality constraints which increase with increasing the number of state variables, and it is hard to solve problems with more than three constraints. While integer programming yields integer solutions, transforming nonlinear objective functions and constraints into linear forms so that integer programming methods can be applied can be a very difficult task. Also, the various integer programming techniques do not guarantee that an optimal solution can be obtained in a reasonable time. Similarly, for branch-and-bound and other implicit enumeration techniques most often require significant computational effort to determine an exact optimal solution particularly for large scale optimisation problems. Discrete reliability optimisation problems are sometimes solved by continuous versions and rounding off the optimal values. Although many algorithms have been proposed for nonlinear programming problems, only a few, such as The Sequential

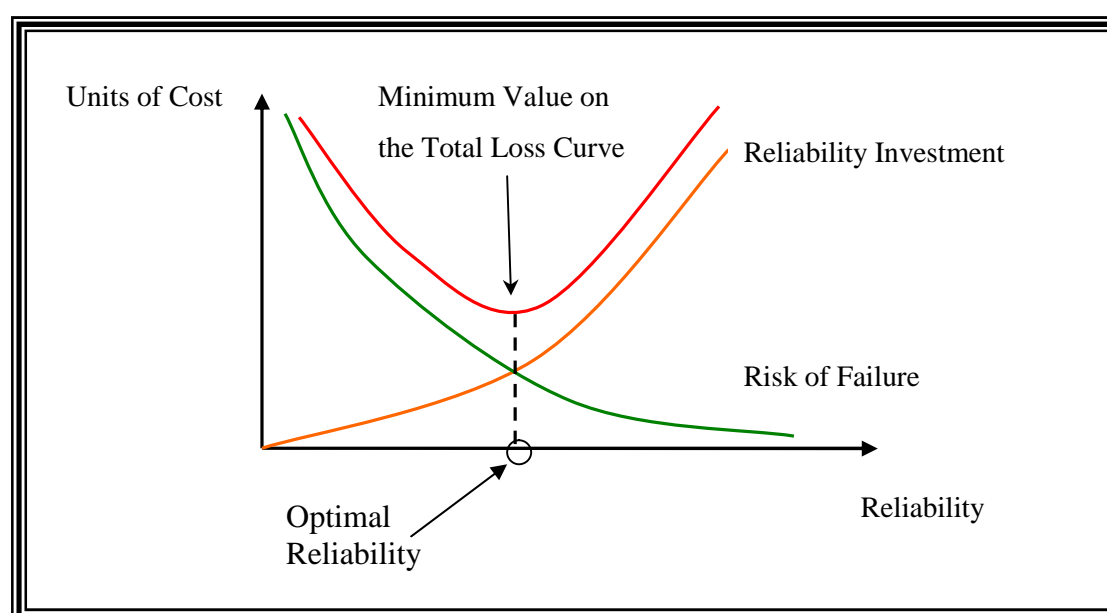
Unconstrained Minimisation Technique (SUMT), The Generalised Reduced Gradient Method (GRG), The Modified Sequential Simplex Pattern Search, and The Generalised Lagrangian Function Method, have proved to be effective when applied to large-scale nonlinear programming problems (Kuo *et al.*; 2001). The maximum principle generally has difficulty solving problems with more than three constraints. Likewise geometric programming is restricted to problems that can be formulated by polynomial functions. Meta-heuristics such as Genetic Algorithms, Tabu Search and Simulated Annealing methods can be used to solve complex discrete optimisation problems. These methods provide more flexibility and require fewer assumptions on the objective as well as the constraint functions with a little exception to the Tabu Search method (TS); since development of an effective TS method requires ingenuity and thorough understanding of the problem. They can be effective particularly when the objective function is not available in a closed form and the underlying mathematical model is very complex. While these methods are relatively easier to implement on computers they are also known for involving significant computational effort and providing only the heuristic solutions. Due to latest advancement in technology and introductions of powerful computer processors, the computational issue related to these methods is diminished to some extent and the efficiency of converging to better solutions is subsequently improved, particularly for genetic algorithms. For this reason, reliability literature has seen great proportion of contribution through genetic algorithm application on many optimisation problems in the last decade alone. A comprehensive literature on this is provided by Levitin (2007).

## **2.3 REVIEW OF OPTIMISATION PROCESSES USING THE RISK BASED RELIABILITY ALLOCATION METHOD**

This research extends the recently published work of Todinov (2004, 2006) on risk-based reliability allocation method, by demonstrating its application for selecting an optimal system configuration from a discrete choice of components. For a reliability system, consisting of many subsystems, the problem is to allocate system reliability

by selecting an optimal combination of components such that the total loss: the *sum of the reliability investment and risk of failure*, associated with a non-repairable system failure is minimum – Fig. 2.6. These alternatives have different reliabilities and costs hence resulting in a difficult combinatorial optimisation problem. The statement of this problem first appeared in Todinov (2005).

As such, the area of this research is new and relatively fresh since the type of optimisation problem studied here is derived from the recently published work of Todinov (2004, 2006) which introduces a novel and one of the most innovative methods of risk based reliability allocation. For this reason, no new literature currently exists outside Todinov's publications, particularly in the context of dealing with the optimisation problem studied in this research. However, the nature of this optimisation problem can be interpreted to fit all eight models of optimisation, discussed in section 2.2.2, specifically models 5-8 which are involved with discrete components, cost minimisation, component assignment and multi-objective optimisation problems. For this reason, some of the interesting work in this area based on reliability optimisation with cost constraints is reviewed in this section, despite having disparate characteristics to the risk based reliability allocation method.



**Figure 2.6 Risk Based Reliability Allocation Method**

In general, the conventional approach of reliability optimisation methods to-date appears to be driven mostly by the principle of setting the highest level of system reliability for a given cost. The objective of the risk-based reliability allocation technique is not about setting the highest level of system reliability. Instead, the reliability allocation is driven solely by the amount of 'total losses associated with a system failure'. Many of the popular reliability allocation strategies do not take into account the total losses from failures during reliability allocation. Since 1977, there have been a significant number of articles and books such as, Tillman *et al.* (1977, 1980), Jensen (1970), Tzafestas (1980), Misra (1986, 1992), Xu *et al.* (1990), Aggarwal (1993), Brown *et al.* (1997), Yang *et al.* (1999), Cantoni & Zio (1999), Kuo & Prasad (2000b), Zio (2000), Guikema and Pate-Cornell (2002), Elegbede *et al.* (2003), Pham (2003), Wattanapongsakorn & Levitan (2004), Meziane *et al.* (2005) and Yalaoui *et al.* (2004, 2005), related to reliability optimisation involving costs.

Most of the methods described in these sources, however, are related to either maximising the reliability of a system given an overall budget constraint (a maximum acceptable total cost of resources toward the reliability maximisation) or minimising the total cost of resources necessary to achieve a specified level of system reliability. For rare instances where a balance between cost and reliability is targeted, the objective has not been to minimise the risk of failure, as in the risk based reliability approach. In both Cantoni *et al.* (1999) and Zio (2000), an excellent methodology based on Monte Carlo simulation and genetic algorithm is proposed for solving complex plant (e.g. Shale oil) design problems. With choices on the type of components to be used and the assembly configurations, the optimisation process is subject to conflicting interaction of reliability/availability objectives with the economic costs associated to the design implementations, system construction and future operation. They introduce a profit function optimisation by taking into account various different costs into the optimisation process (for repairable systems) but no risk of failure is considered, as in the risk based reliability allocation method, for setting reliability targets. Similarly, Guikema and Pate-Cornell (2002) introduce three different models of risk-cost relationship for determining the values of the optimal allocation of funds among the various risk mitigation projects in order to minimise the probability of system failure (for a satellite system) but not the amount of total losses

associated with premature failures. According to Kuo *et al.* (2001), there exists no method dedicated to the problem of reliability and redundancy allocation in parallel-series systems where components must be chosen among a finite set. Yalaoui *et al.* (2005) address the reliability and redundancy allocation problem (optimisation model 4 from section 2.2.2) in parallel-series systems where the reliability of the components (from a finite set) and the redundancy levels have to be simultaneously determined in order to minimise the consumption of resources under a minimum reliability constraint. They developed a dynamic programming method for solving the cost minimisation problem, which could also be formulated as an integer linear programming problem. They have also shown that their method was equivalent to a one dimensional knapsack problem for optimising the profit under maximum volume constraint. While the method proposed in this article appears useful and flexible, the optimisation process undertaken does not take into account the losses from failure for allocating optimal level of system reliability by selecting appropriate components from the finite set. For series-parallel systems, Yalaoui *et al.* (2004) propose theoretical and practical results for reliability allocation problem, in which the reliability of the components have to be determined in order to minimise the consumption of a resource under a minimum reliability constraint. Once again, the allocation process is not driven with the view of losses from failure.

Using the optimisation model of reliability and redundancy allocation for parallel-series systems, Elegbede *et al.* (2003) present an efficient algorithm for allocating reliability and redundancy in each subsystem for achieving some target level of system reliability while minimising the cost of the system. In this paper, they prove that the components in each stage of a parallel-series system must have identical reliability under some non-restrictive condition on the component's reliability cost function. The algorithm appears to produce considerably better results than an already published algorithm. While considered as the increasing function of cost, the reliabilities of the components in Elegbede *et al.* (2003) are also assumed to be any real value between 0 & 1. In practice only a few types of components which have the same function in a system are available in the market. Very few studies have considered this assumption (Yalaoui *et al.* 2005). For example, Coit & Smith (1996) worked on the redundancy allocation problem in parallel-series systems in which each

subsystem was a k-out-of-n:G system, using methodology based on genetic algorithm when the components were chosen from a finite set, assuming different types of redundancy levels such as active, standby and k/n. They also considered the problem of minimising the cost, subject to a minimum requirement of system reliability and other constraints such as weight. Kuo & Prasad (2000a) propose an exact resolution method for similar problems for coherent systems.

In Meziane *et al.* (2005), the problem of electrical network reliability where redundant electrical devices are included for a multi-state system, is studied by using an ant colony algorithm (ACA) and the system reliability maximisation in their formulation is subject to performance and cost constraints. The optimisation process considered in their work belongs to the models 4 & 8 from section 2.2.4. The objective of the optimisation technique, ACA, is to select and evaluate the best configurations with maximum reliability under cost and performance constraints, which shows very interesting results for power system optimisation. However, the reliability allocation method is not driven by the total loss function as in this research.

A very interesting work has been communicated by Kumral (2005) regarding reliability optimisation of a mine production system using genetic algorithm. The optimisation process is required to estimate the minimum level of reliability for each sub-system along with incorporating a cost minimisation criterion for the risk associated with these uncertain estimates in order to avoid critical losses from the standpoints of safety, quality, health, environment and finance, as described by Kumral. Analogous to this approach, Yang *et al.* (1998) also use genetic algorithm for reliability allocation in nuclear power plant while minimising the total plant costs subject to the overall plant safety goal constraint with a different approach than Kumral by using fault trees and probabilistic safety assessments for evaluating target reliabilities of individual subsystems. The optimisation processes in the last two sources, despite being conceptually similar in some ways to the risk based reliability allocation method, are however, significantly different because of the minimum reliability requirements and no consideration of the amount of total losses from failure in allocating the optimal level of system reliability. Similarly, Brown *et al.* (1997)

provide useful information about designing an automated primary distribution system by optimising both cost and reliability. The objective function (total cost of reliability) is the sum to two costs, utility cost of reliability and customer cost of reliability. By using methods such as integer programming, genetic algorithm and simulated annealing along with some hybrid methods, the authors minimise the objective function for demonstrating its use as a tool for helping engineers design a reliable distribution system while minimising costs. The optimisation process however is different to risk based reliability allocation method in all aspects.

In Pham (2003), for a parallel system consisting of ‘ $n$ ’ components, the optimal subsystem size ‘ $n^*$ ’ was determined that minimises the average system cost which included the cost of the components and the cost of system failure. For parallel-series systems, the optimal subsystem size was determined that maximise the average system profit. Optimum reliability minimising the sum of cost of failure and cost of reliability has been discussed by Hecht (2004), who acknowledged that the total user cost has a minimum and the failure probability at which the minimum is reached represents the optimum reliability in economic terms. For systems characterised by a constant hazard rate, a model for determining the optimum hazard rate of the system at which the minimum total cost is attained has been proposed in Todinov (2004). For embedded systems, Wattanapongsakorn and Levitan (2004), for example, presented models for maximising reliability while meeting cost constraints and minimising system cost under multiple reliability constraints. Their optimisation method is based on simulated annealing meta-heuristic technique and the objective is to select both software and hardware components and the degree of redundancy to optimise the overall system reliability under cost constraints but no losses from failure is considered in their optimisation process. There exists also work related to reliability optimisation based on fuzzy techniques, dealing with the cost of the system and the costs of the separate components (Ravi *et al.*, 2000). The optimal redundancy allocation, however, is again oriented toward maximising the system reliability by minimising the system cost, not minimising the losses from failures. These models do not incorporate the losses from failures, and it is implicitly assumed that once reliability is maximised, the losses from failures will automatically be minimised.

---

## 2.4 GENERAL REVIEW OF RELIABILITY OPTIMISATION

The system-reliability optimisation literature was reviewed by Tillman *et al.* (1977) for the period before 1980 while Misra (1986) presented a survey of the literature on system-reliability design pre year 1986. Several interesting papers and, more recently, books on reliability optimisation have been published thereafter which are efficiently reviewed by Kuo *et al.* (2000, 2001). According to them, recent developments in system-reliability optimisation can be classified into seven categories:

### 2.4.1 Heuristics for Redundancy Allocation

It appears that the heuristic methods developed for optimisation model three, as mentioned in section 2.2.4, in the period before 1980 have the common approach where a solution in an iteration is obtained from the solution of a preceding iteration by increasing one of the variables (selected via sensitivity factor) by '1'. Nakagawa and Miyazaki (1981) numerically compared the heuristic methods of Nakagawa and Nakashima (1977), Kuo *et al.* (1978), Gopal *et al.* (1978) and Sharma and Venkateswaran (1971) for a redundancy allocation problem with nonlinear constraints. On the other hand, the heuristics presented after 1980 are based on distinct approaches. Dinghua Shi (1987) developed a heuristic method with separable, monotonic non-decreasing constraint functions following the approach of adjusting unit increment with time. Dinghua's method requires determination of all minimal path sets of the reliability system. Kohda and Inoue (1982) developed a method which was applicable even when the constraints did not involve all the non-decreasing functions. Kim and Yum (1993) developed a similar algorithm for redundancy allocation. The algorithm makes excursions to a bounded subset of infeasible solutions while improving a feasible solution. Based on the Branch-and-bound strategy and the Lagrange multiplier method, Kuo *et al.* (2001) also presented a heuristic method for redundancy allocation. The bound associated with any node is the optimal value of the corresponding optimisation problem and the nonlinear programming problem associated with each node is solved by the Lagrangian



multipliers method. Additionally, Jianping (1996) developed the bounded heuristic method for optimal redundancy allocation. It assumed that the constraint functions were increasing in each variable. The method has some similarity with the method of Kohda and Inoue (1982) in the sense that an addition and a subtraction are simultaneously done at two stages in some iterations.

### **2.4.2 Meta-heuristic Algorithms for Redundancy Allocation (Genetic Algorithms, Simulated Annealing and Tabu Search)**

In recent years, meta-heuristics have been selected and successfully applied to handle a number of reliability optimisation problems. The meta-heuristics based more on artificial reasoning than classical mathematics based optimisation, include genetic algorithms, simulated annealing and tabu search. Genetic algorithms (GA) seek to imitate the biological phenomenon of evolutionary production through the parent-children relationship. Holland (1975) and later Goldberg (1989) made pioneering contributions to the development of GA. Gen and Cheng (1997, 2000) described the application of GA to combinatorial problems including reliability optimisation problems – a good overview of genetic algorithms is provided in Chapter 5. Concerning the design of a personal computer, Painton and Campbell (1995) adopted a genetic algorithm approach to solve a reliability optimisation problem for a system with series-parallel configuration. Ida *et al.* (1994) and Yokota *et al.* (1995) designed a genetic algorithm for optimal redundancy allocation in a series system in which the components of each subsystem were also subject to two classes of failure modes. Majety and Rajagopal (1997) developed an evolution strategy based on an adoptive penalty function to solve some reliability optimisation problems. Dengiz *et al.* (1997) designed a genetic algorithm for cost-optimal network design. A similar algorithm was developed by Deeter and Smith (1998) for cost-optimal network design but without the assumptions used by Dengiz *et al.* (1997).

The concept of simulated annealing (SA) method is based on a physical process in metallurgy and used generally to solve combinatorial optimisation problems.

Metropolis *et al.* (1953) developed a method which was further modified by Cardoso *et al.* (1994). They introduced the non-equilibrium simulated annealing algorithm (NESA). The method was further developed by Ravi *et al.* (1997) who denoted this variant of NESA as I-NESA and applied to optimisation problems.

The process in tabu search (TS) guides the heuristic method to expand its search beyond the local optimality. It is an artificial intelligence technique which utilises memory at every stage to provide an efficient search for optimality. It was introduced by Fred Glover (Glover and Laguna 1997). Tabu search for any complex optimisation problem combines the merits of artificial intelligence with those of optimisation procedures. The most prominent feature of TS is the design and use of memory-based strategies for exploration by imposing restrictions on the search at every stage based on memory structures. Similar to GA and SA, TS is useful for solving large complex optimisation problems that are very difficult to solve by exact methods.

### **2.4.3 Exact Methods for Redundancy Allocation**

The purpose of exact methods is to obtain an exact optimal solution to a problem. Many exact methods were developed before 1980 and documented in Tillman *et al.* (1977). Nakagawa and Miyazaki (1981) adopted the surrogate constraints method to solve the optimisation model 3 (from section 2.2.4) with exactly two constraints. Misra (1972) has proposed an exact algorithm for optimal redundancy allocation. The method was later implemented by Misra and Sharma (1991) and Misra and Misra (1994) for solving various redundancy allocation problems. This algorithm does not always give an exact optimal solution (Kuo *et al.* (2001). For large systems with a good modular structure, Li and Haines (1992) proposed a three-level decomposition method for reliability optimisation subject to resource constraints.

### **2.4.4 Heuristics for Reliability-Redundancy Allocation**

This is the approach used in the model 4, mentioned previously in section 2.2.4. Tillman *et al.* (1977) were among the first to solve the problem using a heuristic and search technique. Gopal *et al.* (1980) developed a heuristic method using the stage sensitivity factor approach. The branch-and-bound method of Kuo *et al.* (1987) is

also useful for solving this type of optimisation problem. They demonstrated the method for a series system with five subsystems. Xu *et al.* (1990) offered a similar method with separable constraints. Hikita *et al.* (1992) developed a surrogate constraints method to solve model 4 with separable constraints. The method is based on the theory developed by Luenberger (1962) for minimising a quasi-convex function subject to convex constraints. In this method, a series of surrogate optimisation problems, each consisting of a single constraint, is solved. Chi and Kuo (1990) formulated mixed integer nonlinear programming problems for reliability-redundancy allocation in software systems and systems involving both software and hardware.

### 2.4.5 Multiple Objective Optimisation in Reliability Systems

While designing a reliability system, it is always desirable to simultaneously maximise system reliability and minimise resource consumption; a key concept behind the single object optimisation problems. However, when the limits on resource consumption are flexible or cannot be determined clearly, it is useful to employ an optimisation approach with multiple objectives. While such approach usually involves determination of all Pareto optimal (non-dominated) solutions, it is possible not to find a single solution which is optimal with respect to each objective. For example, an aircraft design engineer is often required to consider other objectives such as minimisation of cost, volume, weight etc. It may not be feasible to define limits on each objective, treated as constraints while maximising the highly desirable reliability. In such situations, the designer comes across the problem of optimising all objectives simultaneously. A general approach for solving multiple objective optimisation problems is to find a set of non-dominated feasible solutions and make interactive decisions based on this set (Kuo *et al.* (2001). Sakawa (1981) developed a large-scale multiple objective optimisation method to deal with the problem of determining optimal levels of component reliabilities and redundancies. He provided a theoretical framework for the sequential proxy optimisation technique (SPOT), which is an interactive, multiple objective decision-making technique for selection among a set of Pareto optimal solutions. Misra and Sharma (1991) adopted an

approach which involves the Misra integer programming algorithm and a multi-criteria optimisation method based on the min-max concept for obtaining Pareto optimal solutions. Misra and Sharma (1991) also presented a similar approach to solve multiple objective reliability redundancy allocation problems in reliability systems. Their methods take into account two objectives: maximisation of system reliability and minimisation of total cost subject to resource constraints. Dhingra (1992) used a goal programming approach and demonstrates the multiple objectives approach for a four-stage series system with constraints on cost, weight and volume. Similarly, Gen *et al.* (1990) also solve reliability optimisation using goal programming.

#### **2.4.6 Optimal Assignment of Interchangeable Components in Reliability Systems**

When a system has interchangeable components with different reliabilities, the system reliability depends on the assignment of such components to required positions. El-Neweihi *et al.* (1986, 1987) solved the problem analytically for series-parallel structures assuming that the component reliabilities were invariant of position. For parallel-series structures, they suggested a linear programming approach. Prasad *et al.* (1991) developed the algorithm to solve the problem for series-parallel structures by assuming the separability condition. They also provided two greedy algorithms for this problem. If both algorithms yield the same solution, then that solution is considered optimal. Prasad *et al.* (1991) also developed a heuristic method to solve the problem for series-parallel structures involving some of the classical assignment problems. Baxter and Harche (1992) presented a heuristic for optimal component assignment in parallel-series system showing that the system reliability calculated using their heuristic, converges to the optimal value since the number of components and subsystem sizes tend to infinity. Prasad and Raghavachari (1998) developed a heuristic method for parallel-series structures. Using important results of El-Neweihi *et al.* (1986), they approximated the problem as a mixed integer linear programming problem. The problem of allocating ' $m$ ' types of components to a general assembly of ' $n$ ' series system was considered by El-Neweihi *et al.* (1987). Under certain conditions, they derived an allocation that

stochastically maximises the number of functioning systems. As a consequence, this allocation also maximises the probability that at least k-out-of-n systems function. Malon (1990) presented a greedy rule to assemble modules of a coherent system out of a collection of available components. The greedy rule assembles modules one by one using best available components. Procedure using pair-wise interchange of components for obtaining optimal component assignment in coherent system was suggested by Boland *et al.* (1989). They introduced the notion of comparison of criticality of two positions in the system and used it to improve system reliability through pair-wise interchange of components. Lin and Kuo (1996) presented a greedy method for optimal component assignment in a general coherent structure when the component reliabilities are invariant of positions. Zuo and Kuo (1990) have summarised the results available for the invariant optimal design of consecutive k-out-of-n systems. They have also identified invariant optimal designs for such systems and proved that invariant optimal designs for other consecutive k-out-of-n systems do not exist. Zhang *at el.* (1991) have applied the invariant optimal design concept to a railway management system.

### **2.4.7 Effort Function Optimisation**

One of the standard approaches for enhancing system reliability is to increase the reliability of the components. However, an increase in component reliability requires some effort, which may be cost, volume, weight, power consumption etc and thus system-reliability enhancement also requires such effort. Assume that the effort to increase the reliability of any component from one level to another is measurable by a mathematical function. Such functions, called effort functions, are not necessarily explicit. Reliability engineers usually formulate the effort functions based on knowledge of the development process. The problem under consideration is to minimise the total effort required to increase the reliability of a general coherent system from an existing level to a desired level through incremental increases in component reliabilities. Albert (1958) solves this problem for series systems when the effort functions are the same for all components. Lloyd and Lipow (1962) provided a good description of this method. Dale and Winterbottom (1986) provide a solution approach for a general coherent structure.

The above listed general review of the reliability optimisation for the seven categories is only the brief extract from the widely accepted literature review provided by Kuo *et al.* (2001). For complete and thorough details of the literature in the field of reliability optimisation, it is suggested to consult all publications produced by Tillman *et al.* (1977, 1980,) , Kuo *et al.* (2000, 2001) and Kuo and Wan (2007).



# 3

# EVOLUTIONARY ALGORITHMS

---

This chapter details one of the most recognised optimisation techniques for solving complex scientific problems involving very large search spaces, called ‘Evolutionary Algorithms’. The chapter begins by first introducing the theory of evolutionary algorithms in section one, followed by its general structure which is defined in section two. The detailed overview of the features of an evolutionary algorithm including basic terminologies and concepts are explained in section three. With section four explaining the steps for designing evolutionary algorithms, the chapter concludes at section five which describes the various types of this optimisation technique.

### 3.1 INTRODUCTION

Similar to development of life in a natural system, an evolutionary process continuously changes the individuals of a population by varying their attributes and characteristics using the fundamental properties such as reproduction, recombination (crossover, mutation) and replacement. These properties of the evolution process motivated researchers from different fields to implement computer based algorithms (simulations) of evolution for solving difficult problems in their research areas. Evolutionary algorithms are therefore types of stochastic search algorithms which emulate the evolution properties and characteristics.

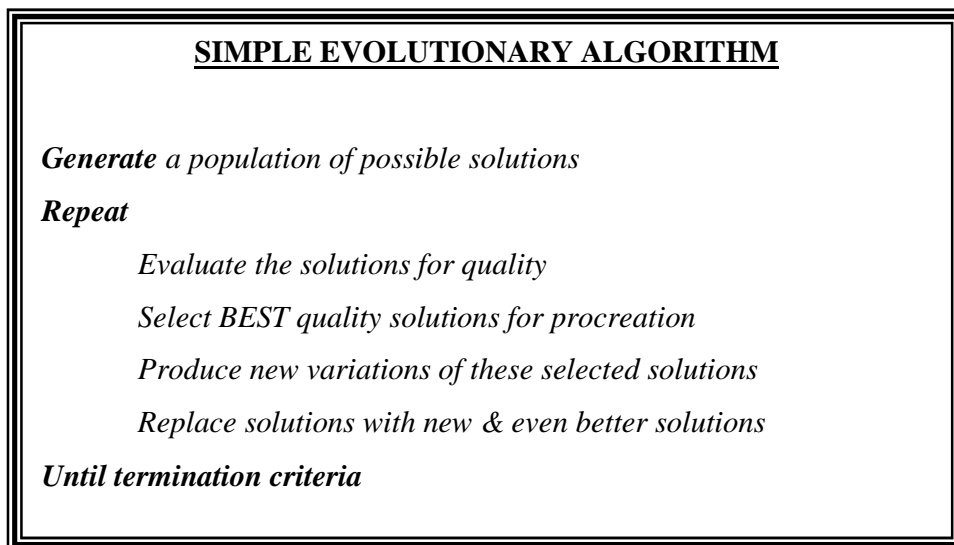
This idea of taking the techniques used by nature to produce diverse complex systems and use them as an algorithm for making scientific computation has been of interest to the researchers since 1950s. However, before the availability of large powerful computers, these biologically derived ideas could not be implemented since even extremely simplified versions of such evolutionary computations were too slow for most applications. By the late 1980s, computer power had vastly increased and combined with human intelligence, caused a sudden increase in research activities which resulted in a large number of publications in the area of computation based on evolutionary algorithms. For this reason and the interdisciplinary characteristics of computations based on evolutionary algorithms, there appear to be many originators and many names (types) associated with this method. The interdisciplinary characteristic means that different researchers from different fields had the idea independently of using evolution as an algorithm but they never read each others publications because of the immense diversities in the respective research fields. This weakness in the research platforms is somehow reduced by the arrival of the internet which can facilitate the review of all the latest publications based on research methodology (e.g. evolutionary algorithm) as well as the field it is implemented in (e.g. medical research, engineering research etc); therefore, allowing researchers from all different disciplines to at least search and compare for a possible duplication of their work. A concise summary of the origins of evolutionary computation can be found in Back *et al.* (1997). Another good source detailing the introduction to



evolutionary computation, its historical background and references of early papers in this area is available in Fogel (1998) along with the broad overview of evolutionary computation in Ashlock (2006).

## 3.2 GENERAL STRUCTURE

The general structure of an EA consists of generating an initial set of potential solutions, called population. Each individual in the population is called chromosome which represents a possible solution to a given problem in the form of a data structure. The population is generated at random (mostly) and the fitness of each chromosome in the population is evaluated with respect to an objective function (Fogel and Ghozeil, 1996). The measure of fitness determines the quality of the chromosome; i.e. increased fitness levels will correspond to better solutions. Based on these fitness levels, appropriate chromosomes are selected for reproduction as parents by using evolution operations (e.g. crossover, mutation) expecting to form new chromosomes (offspring) with better fitness levels. If a specified termination criterion is not reached, the next population is generated using the existing parents and new offspring depending on the probabilistic selection and fitness levels of the individuals. A simple evolutionary algorithm is shown in Fig.3.1.



**Figure 3.1 A simple evolutionary algorithm**

## 3.3 FEATURES OF EVOLUTIONARY ALGORITHM

### 3.3.1 Biological Overview of Evolutionary Algorithm

Evolutionary algorithms can be best understood with good knowledge of biological evolution and its fundamental concepts. A brief introduction to these can be started from ‘Deoxyribo-Nucleic-Acid’ (DNA) which forms the ‘*chromosomes*’ present in every living organisms and determines in many ways the properties of the organism that carries them. The DNA drives the highly complex physicochemical processes responsible for the growth of the organism from the fertilised egg up to the adult stage. In other words, DNA is composed of all the necessary instructions for forming an organism and is also referred to as ‘*genetic code*’. A chromosome is made up of ‘*genes*’ which are the sequences of DNA bases that code for the traits, e.g. eye colour, height, hair colour etc. Although genes are thought of as the basic units of information, in a pure biological context, each gene is formed of a number of amino acids from the four-member set ‘TCGA’ (i.e. Thymine, Cytosine, Guanine and Adenine). The value of each trait represented by the gene is called ‘*allele*’ and its position within the chromosome is called ‘*locus of the gene*’. A good reference for understanding genes can be found in Lewin (2000).

Each gene is responsible for a trait of the future individual; the acquired trait will depend on the locus of the gene with corresponding allele of the gene. For example, if a gene with locus ‘eye colour’ has the allele ‘blue’, and the gene with locus ‘hair colour’ has the allele ‘blonde’, the new organism will be created containing blue eyes and blonde hair. The complete information contained in a chromosome in the form of a genetic code is called the ‘*genotype*’. It is interpreted (decoded) by the various enzymes (in a biological context) in order to actually construct the particular organism it describes. That decoded value of the particular organism is called the ‘*phenotype*’ whose physical representation is contained in the DNA. Figure 4.1 shows the decoding process of a sample chromosome (genotype) into its visible appearance describing one of many frog species (phenotypes).

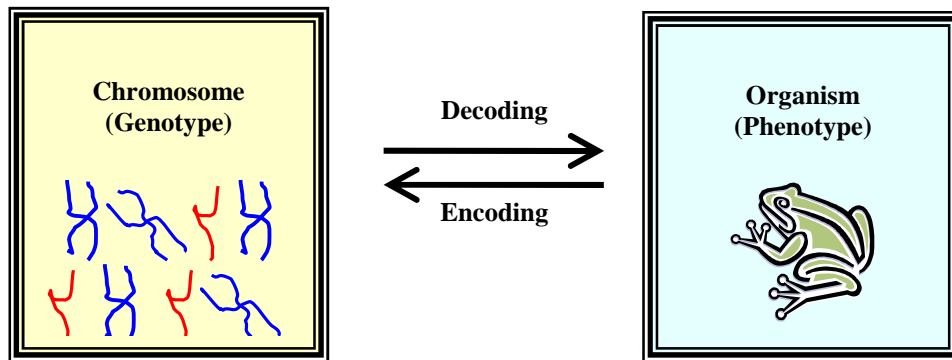


Figure 3.2 Genotype decoding into phenotype

A quick snap-shot of the biological terms used in the evolutionary computation is highlighted in Table 3.1.

Terms	Explanation
Chromosome	Coded structure (solution) containing full information about the organism and its properties
Genes	Units of information (trait)
Locus	Position of the gene in a chromosome
Alleles	Value of the gene
Genotype	Genetically coded solution
Phenotype	Decoded value of the solution
Model of Evolution	Method for selecting parents and inserting children in a population
Coadaptation of Genes	Noticeable impact formed by the combination of genes which is null otherwise
Disruption	The process of destroying the coadapted genes
Epsitais	Non-linearity factor in forming new traits

Table 3.1 Terminologies used in Evolutionary Algorithm

## 3.3.2 Terminologies and Concepts in Evolutionary Algorithms

### 3.3.2.1 Chromosome Representation

In evolutionary algorithms, chromosomes are used to represent solution of a problem in the form of data structures. Traditionally, chromosomes have been coded as binary strings or arrays containing '0' and '1' (Goldberg, 1989). For combinatorial optimisation problems, an encoding using integer values can be more efficient (Holland, 1975). For example, consider a reliability system consisting of six components such that for every component, there are ten available choices of alternatives each. These alternatives are different in terms of the reliability levels and associated cost. Having the discrete choice of available alternatives, the optimisation problem is to find an optimal combination of components with minimum system cost (i.e. total cost of all components) while satisfying some target level of system reliability. A chromosome representing one possible solution can be represented as:

<i>Number of Genes</i> = 6
<i>Number of possible alleles per gene</i> = 10
<i>Chromosome</i> = (7   8   4   6   1   3)

**Figure 3.3 Chromosome Representation**

The data structure in the form of a string used for the example chromosome above is consisted of six units of information (genes), each representing the individual component position (locus). For each unit in the chromosome, there are ten possible values (alleles) which can be selected as the choice of component alternatives. Therefore, every position in the chromosome string above represents the selected alternative value of the respective component. In other words, the encoded solution (genotype) can be decoded to a system (phenotype) containing alternative seven for component one, alternative eight for component two, alternative four for component three and so on – see Figure 3.4.

<p><i>Genotype (coded solution)</i> = (7   8   4   6   1   3)</p> <p><i>Phenotype (system configuration)</i> = <math>\{C_1^7, C_2^8, C_3^4, C_4^6, C_5^1, C_6^3\}</math></p> <p>Where, <math>C_i^j</math> represents <i>i</i>th component with <i>j</i>th alternative</p>
---

**Figure 3.4 Genotype and Phenotype Transition**

The representation of a chromosome depends on the nature of the problem in hand and can not be generalised for all problems. While this is the main reason for the recent advancements in the field of evolutionary algorithm, a good general theory regarding the representation of chromosome is yet to be specified (Ashlock, 2006). The type of chromosome representation selected drives the mechanism of placing the problem specific knowledge in designing the evolution algorithms which explains the reason for the advancement in the field of evolutionary computation. A good survey of successfully applied chromosome representations can be found in Back *et al.* (2000a) and Michaelwicz (1996). The latter also proposed using arbitrary data structures capable of giving a complete description of a problem solution without additional coding. More on chromosomes representation along with the issues surrounding the types of encoding and the cardinalities of the sets involved in the mapping between genotype and phenotype spaces are efficiently detailed in Gen and Cheng (1997, 2000) and Falkenauer (1998).

### 3.3.2.2 Global and Local Optimum Results

The general purpose of an optimisation process is to find the best value of a function after taking into account all relevant parameters and constraints. The best value achieved as a result of this optimisation process dominates all other possible values of the function depending on the nature of the optimisation process. If the purpose of an optimisation process is to minimise some function value, the best value achieved from optimisation is exceeded by all other values of the function. Similarly, if the goal of the optimisation process is to maximise a function, the best value derived from the optimisation process exceeds all other possible values which may exist on the solution space of the function.

Therefore, an outcome of an optimisation process which represents such extreme characteristics of being either minimum or maximum is referred to as '*global optimum*' of a function. In generic terms, a global optimum is a point in a search space where all other points are either worse or equal to this value. The latter part of the statement represents circumstances when a function might possibly have more than one global optimum. For very large scale optimisation problems, the search space is generally too large to be explored thoroughly. One way of exploring the search space like these, is to randomly select various regions of the search space and attempt to find acceptable solutions. An excellent literature on search techniques and methodologies can be found in Burke and Kendall (2005).

If a search space is divided into many regions, the global optimum for each region may differ from the global optimum of the other neighbouring regions. In order to simply this confusion, the best result for each region is termed as '*local optimum*' instead of global optimum. It is, therefore, a point in search space which represents the best current solution applicable only to the selected local region. However, the best found overall value of the local optimum results, by taking into account all regions of the divided search space, will be the true global optimum; it is this reason which creates the possibility of having more than one global optimum since more than one region of a search space can have similar values of the respective local optimum. Figure 4.5 demonstrates the types of optimum values for some function,  $F(X)$ . If the objective of the optimisation process is to find the maximum of this function, it would appear that the global optimum is found in three different regions of the search space, each with different values of the underlying variable,  $X$ . Similarly, if the goal of the optimisation process is to find the minimum of the function,  $F(X)$ , the figure shows that there are two instances of finding the required value for the global optimum. Also highlighted in the figure, are the various instances of local optimum results (optima) representing both maximum and minimum values of the optimised function for the respective search region.

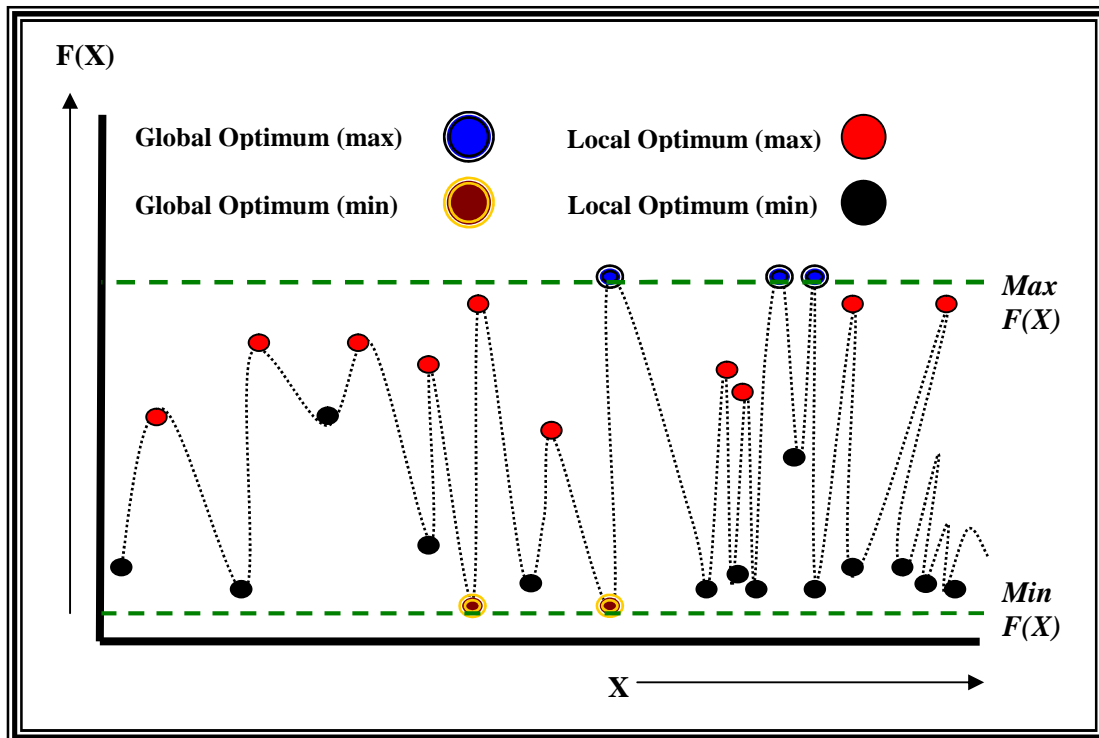


Figure 3.5 Global Optimum And Local Optimum of a Function,  $F(X)$

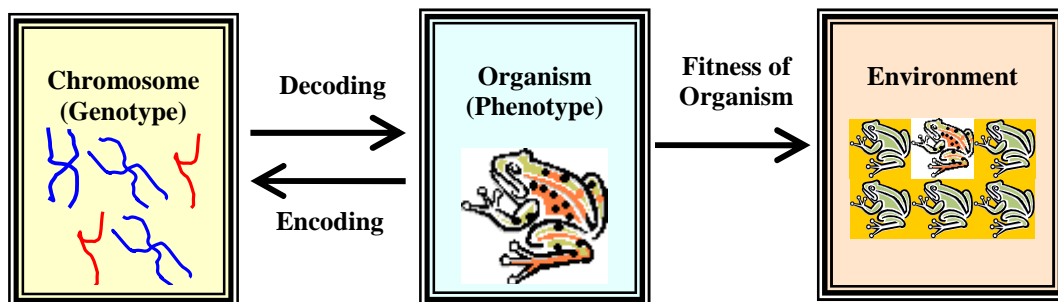
### 3.3.2.3 Population Structure

One of the main reasons which make evolutionary algorithms different from the conventional mathematical or heuristic optimisation methods is its ability to perform parallel search over a set of points (potential solutions) from the total search space. This set is called a *'population'* and the selection of this population in each iteration of the evolutionary algorithm is referred to as *'generation'*. An *'initial population'*, therefore, represents an initial set of chromosomes (encoded solutions) selected in the very first generation of the algorithm and the total of these chromosomes makes up the actual *'size of the population'*. These chromosomes represent the potential solutions of the problem in hand and are often selected randomly in order to provide a scattered sample over the available search space. The reason for the blind formation of the initial population is because at the early stage of the algorithm, no knowledge is generally available regarding the optimisation problem or the region of the search space where a global optimal may be found for the problem. Therefore, each chromosome in a population is required to be evaluated in order to establish its viability as a good solution. This process is carried out using a *'fitness function'*. The

size of the initial population generally depends on the scale of the optimisation problem and the method used for representing chromosomes in the population.

### 3.3.2.4 Fitness Function

In the context of biology, the ‘fitness’ of an organism is its ability to survive and produce progeny depending on the environment it is developed in. It is therefore, a measure of success with which an organism contends with its environment. In a given population, each individual organism possesses its own properties for surviving in the environment; an organism which flourishes in one environment could easily fail in another. Figure 3.6 demonstrates the fitness measure of frog specie with respect to its environment; although the developed organism in this case, is also a frog, it does not belong to the same class of frog species which is found in its developed environment, due to some of its different traits.



**Figure 3.6 Fitness of organism with respect to its environment**

In evolutionary computation, a fitness function, also known as ‘objective function’ in the literature, is an assessment of the potential solution based on a quantitative or (and) qualitative approach. In other words, it is used to establish the numerical quality of the competing solution (chromosome) and in some cases, it decides which of two chromosomes is better without assigning an actual numerical quality. For a given population of potential solutions, a fitness function value with respect to given system properties or conditions (similar to ‘environment’ in biological context) is evaluated for each individual in the population in order to identify the quality of the solution it represents. For example, the fitness of the potential solutions for the



problem stated in 3.3.2.1 is demonstrated in Figure 3.7. Each solution string from its encoded data structure is first decoded into its real life state showing the actual configuration of components in the reliability system. After identifying the components of the systems, the next step is to apply the fitness function by calculating the total system cost. It is achieved by simply adding the individual costs,  $C_i$  of all six components, and can be shown as:

$$\text{Fitness Function (Total System Cost)} = \sum_{i=1}^6 C_i .$$

This process is repeated for every chromosome in the population and the results are compared in order to identify the solutions with minimum system costs.

In large scale optimisation problems, the search space can be very large and finding the optimum solution becomes a very difficult task. For the same example of the reliability system above, the search space is consisted of  $10^6$  combinations of possible solutions for six components with ten alternatives each. Since the initial population is generated containing a very small proportion of the total search space (one million possible solutions in this case) selected randomly, it is crucial to identify which individuals in the population are nearer to the optimum with respect to the specified condition of minimum system cost.

Thus, fitness function evaluation separates all possible solutions which are better suited (strong fit) to the required environment from the rest of the other less suitable solutions (weak fit) in a given population. Based on these fitness evaluations, a 'selection' mechanism is applied which facilitates on average, a continuous breeding of the strong fit chromosomes while allowing the weaker individuals to drop out of the population.

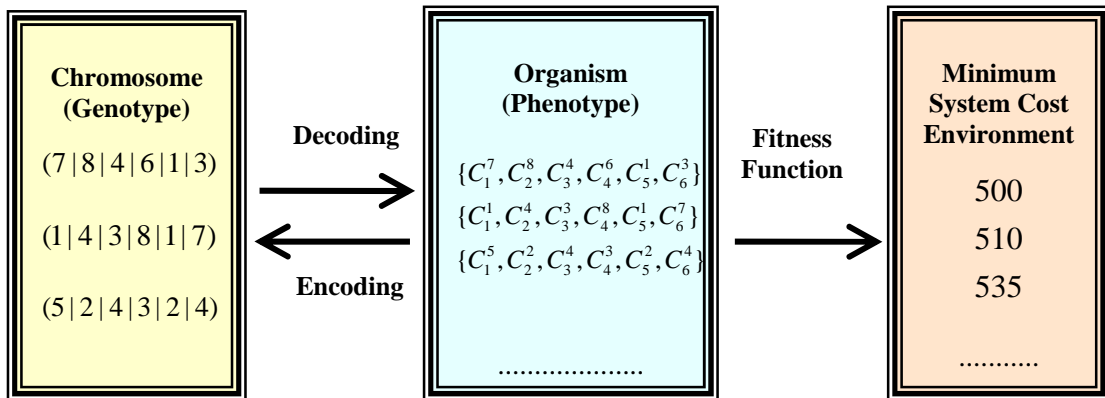


Figure 3.7 Fitness evaluation of chromosome in evolutionary algorithm

### 3.3.3 Processes of Variations in Evolutionary Algorithm

Evolution algorithms simulate the development of life in the natural system using the fundamental processes such as reproduction, recombination, mutation, and inversion.

#### 3.3.3.1 Recombination Process (Crossover Operation)

Recombination or crossover process operates generally on two parent chromosomes at a time and is also known as binary variation. This operation simulates sexual reproduction by mapping a pair of genes from one parent chromosome to the other.

According to the definition provided by Ashlock (2006),

‘A crossover operator for a set of genes  $G$  is a map

$$Cross: G \times G \rightarrow G \times G$$

or

$$Cross: G \rightarrow G \times G.$$

The point making up the pairs in the domain space of the crossover operator are termed parents, while the points either in or making up the pairs in the range space are termed children. The children are expected to preserve some part of the parents' structure'.

In other words, the crossover operation generates offspring by combining the features of both parents through exchange of genes. Since child chromosomes are expected to inherit some features of the parent chromosomes, the crossover process is applied in the hope of producing better versions of the existing parents with even stronger fitness (depending on the environment). For example, suppose, one parent has exceptional aptitude for mathematics whereas, the other parent has excellent drawing skills. When the chromosomes of these two parents are combined, the progeny will be produced consisting of the features from both parents and it would be expected that at least one of the children will have both mathematical and drawing skills as good as the parents. If the level of fitness is judged on the basis of these acquired skills (environment), the new child will be superior to either of its parents. In evolutionary algorithm, the inheritance of the promising genes from parents to child chromosomes drives the effective exploration of the search space since each new and better fit child chromosome is in fact, a new point (solution) in the total search space.

The transfer of traits from parents to child chromosome is one of the key features of natural evolution which motivates the use of evolutionary algorithm. Although the acquired traits in child chromosomes are not new since they already exist in either or both parents, the combination of these traits in a new chromosome is of most interest. This is one reason in Nature for the best (strong) fit chromosomes for living the longest and yielding the most progeny. As mentioned in the beginning of this section, a crossover process operates generally on two parents for producing offspring and not more which is similar to Nature. An interesting explanation is provided by Falkenauer (1998) regarding this statement, according to which the reason for selecting two chromosomes for parenting, is because of the 'epistasis' and 'coadaptation' of alleles of the genes inside the chromosomes of the living creatures.

The epistasis phenomenon describes the notion of a gene's impact which is influenced by one or many other genes of the future organism. In other words, the traits or the visible properties of a future organism are mostly results of a joint influence of many genes. In the context of evolutionary computation, this represents the non-linear behaviour of a function showing a complex and difficult to predict relationship of the function with its parameters (independent variables). With regards to coadaptation, Faulkenauer describes, 'two alleles are coadapted when the genes 'cooperate': some (possibly just one) combinations of their two alleles are beneficial, but changing one of them nullifies the effect of the other gene, i.e. the influence of the alleles is not additive.' For the crossover process applied for combining the genes from distinct parents (more than one, for example), there is a good chance that the coadapted alleles will be inherited in the new child however, if the parents do not agree on the alleles of those genes, the most likely outcome will be the destruction of the coadapted alleles which may produce undesirable consequences given the epistasis relative to those genes. The destruction of the coadapted set of alleles is called 'disruption'.

In general, a crossover process always carries the risk of disruption of coadapted sets of alleles which grows with the number of parents taking part in the gene recombination process for creating a new child; because the possibility of many parent chromosomes not agreeing with a set of different alleles increases, understandably. For this reason, it would be obvious to use just one parent, but this will mean choosing not to use information from diverse sources at all. Therefore, it appears that the best choice will be to use two individuals as parents for producing new versions (children) of these parents with reasonable diversity. While this explanation by Faulkenauer regarding the development of just two sexes (for most higher organisms) is based only on the computational aspects of the process, it is possible that there may be other possible justification for this development in Nature. Nevertheless, the most efficient approach to procreation which the Nature has settled to (leaping from asexual breeding) appears to be the sexual one involving two sexes, possibly explaining why it is also the case with the world of scientific computation using evolutionary algorithms.

There is a large variety of different types of crossover operations depending on the type of chromosome representation. For typical strings or array representation of chromosomes, the following types of crossover are most commonly used:

### *Single-point crossover*

This is the simplest type of crossover operation used for producing variations of parents. The process involves randomly generating a locus, called the crossover point over the length of the chromosome string and then swapping the genes from the parents from one side of the crossover point in order to produce two new child chromosomes. The information for each child comes from both parents before and after the generated crossover point.

Using the reliability system example introduced in section 3.3.2.1, consider two parent solutions representing a reliability system consisting of six components such that for every component, there are ten available choices of alternatives each.

*Parent 1:*

3	5	7	9	2	1
---	---	---	---	---	---

*Parent 2:*

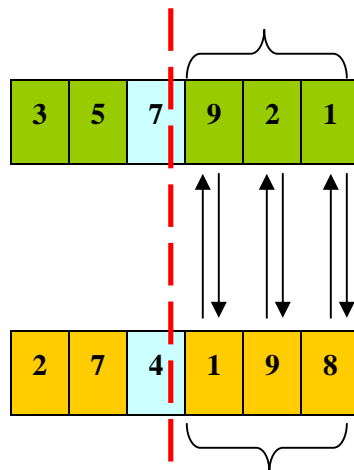
2	7	4	1	9	8
---	---	---	---	---	---

The data structure in the form of a string used for the example chromosomes above is consisted of six units of information (genes), each representing the individual component position (locus). For each unit in the chromosome, there are ten possible values (alleles) which can be selected as the choice of component alternatives. Therefore, every position in the chromosome string above represents the selected alternative value of the respective component.

The one point cross over operation can be applied on the two parents by first selecting a random crossover point. In this case, let this value be locus position three and then swapping all the genes from the right hand sides of both parents.

Therefore,

*Parent 1:*



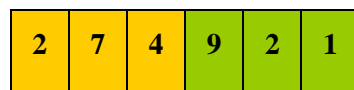
*Parent 2:*

This process creates two new versions of these chromosomes each of which containing information from both parents.

*Child 1:*



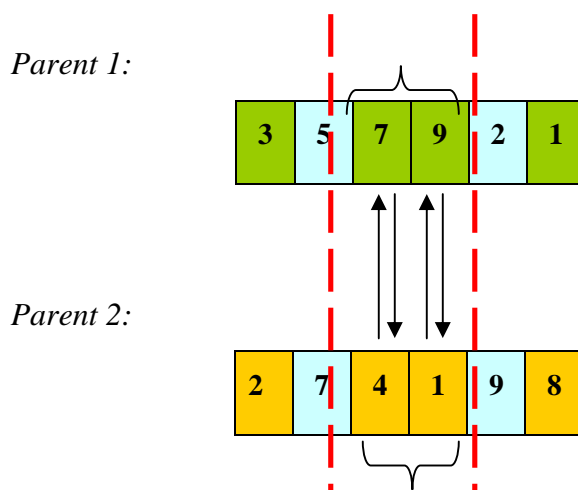
*Child 2:*



The phenomenon of disruption introduced at the beginning of this chapter can also be demonstrated using the two newly produced child chromosomes. Suppose in the first parent, the genes '3' and '2' form a promising combination when exist together but inefficient individually; i.e. these genes are coadapted. Because of the crossover at locus three, none of the two child chromosomes has inherited these coadapted genes together as a result the coadapted genes are destroyed.

### *Multi-point crossover*

The single point crossover exhibits stronger possibility of disruption since it does not appear to treat all loci of the genes equally. For instance, if all the promising genes are unevenly scattered such that their positions are located nearer to the two ends of the chromosomes string or further apart from each other (see example above), the probability of their disruption becomes higher with the single point crossover operation compared to the case where these genes are evenly spread across the length of the chromosome and are much closer together. A simple way of reducing this is to use multiple-point crossover operation. One common type of this crossover operation is a two point crossover which involves randomly generating two crossover points first and then exchanging the alleles of the selected parents which are located in between the two crossover points. The two child chromosomes created as a result of this process contain genes from both parents such that one child contains genes which are swapped between the two crossover points from the first parent while the second child chromosome is created from genes which are swapped from the second parent. Using the previous example of chromosome, the two point crossover can be demonstrated as below:

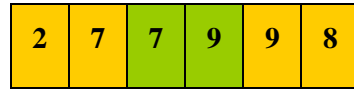


This process creates two new versions of these chromosomes each of which contains information from both parents.

Child 1:



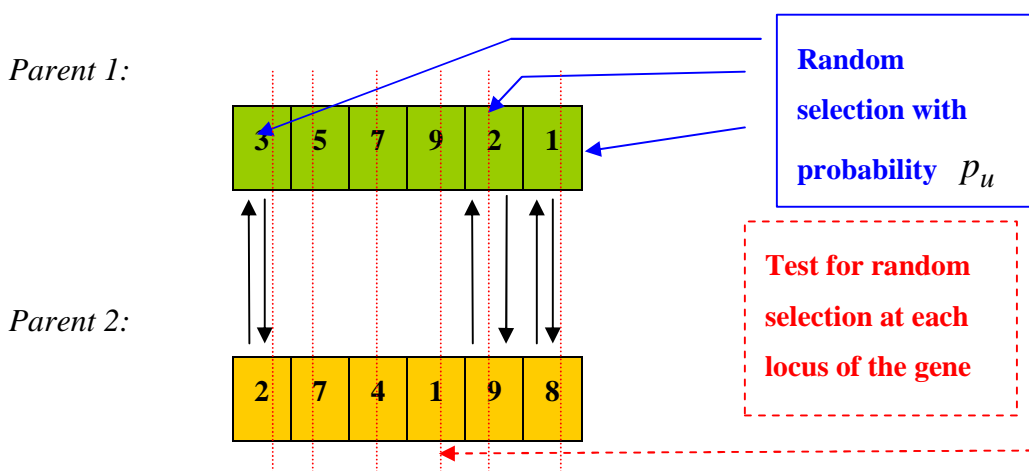
Child 2:



The multiple crossover operation can be extended to  $k$ -point crossover, where  $k$  crossover points are used similar to two point crossover detailed above.

### *Uniform crossover*

Uniform crossover is another common type of operation in which allele is exchanged between the two selected genes of the parents depending on the swapping probability,  $p_u$  for all loci. This probability is usually predefined and is taken to be 0.5. While this type of crossover effectively deals with the disruption process, it is however, computationally expensive because of the need for generating a large number of random crossover points. Further details of this method can be found in Syswerda (1989) and Spears and De Jong (1994).





*Child 1:*

2	5	7	9	9	8
---	---	---	---	---	---

*Child 2:*

3	7	4	1	2	1
---	---	---	---	---	---

A crossover operator can be further customised using any of the above mentioned types and depending on the nature of the optimisation problem in hand. It is for these reasons there exist many forms of this operator in the literature. A comprehensive survey can be found in Burke and Kendall (2005)

### **3.3.3.2      *Reproduction Process***

As the name suggests, reproduction is simply a copy of a chromosome from one generation to the next without any variation in the genes structure. In Nature, this process is demonstrated by the production of offspring through asexual breeding where each child is generally an exact copy of the parent. Similarly, a strong fit individual who is well adapted to its environment can survive and carry over to further generations its capacity to produce offspring. The process is applied in the field of evolutionary computation with the same objective as Nature for preserving the strong fit individuals for continued survival (i.e. search for optimal solution) by including them in the next population. However, like crossover operation, reproduction does not introduce new genes or traits in the next population despite demonstrating strong fitness to the given environment. Due to this reason, this process is used at a lesser extent (if at all anymore) in evolutionary algorithms, where the real objective is to find optimal solutions by exploiting the current best solutions for diverse exploration of the search space.

### 3.3.3.3 *Mutation Process*

As described in the previous section, an evolutionary algorithm strengthens its ability to effectively explore the total search space through crossover operations due to the variations introduced by these processes. These variations are however limited since recombining information (alleles) from parents produces offspring with the same alleles which are already present in either of the two (or both) parents. As a result of this, none of the individuals from the new set of offspring are expected to contain fresh alleles; in the context of evolutionary algorithm, this means, no solutions are found with new information which may assist in directing the search towards more promising solutions, a feature strongly required for problems with infinitely large search spaces. Another concerning issue is the loss of alleles which may be important for directing the search towards optimal solution but inadvertently destroyed during the recombination process. Because it is generally not possible to ascertain which alleles are parts of the best solution, it is necessary to implement another method for recovering the discarded alleles as well as some new ones for the purpose of streamlining the current solutions. This new method is called ‘*mutation*’ which performs random modifications on an individual by altering the alleles for producing newer versions of the same individual. According to the definition provided by Ashlock (2006),

‘A mutation operator on a population of genes  $G$  is a function

$$\textit{Mutation}: G \rightarrow G$$

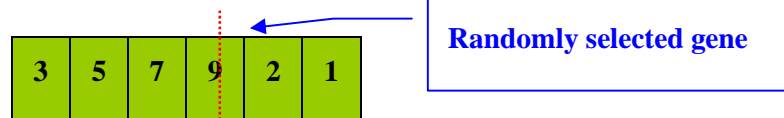
that takes a gene to another similar but different gene. Mutation operators are also called unary variation operators.’

The main purpose of the mutation is to facilitate local search and gradually introduce new ideas into the current population by making small changes in the individuals. Like crossover operations, mutation can also be performed in number of ways. Some of the most common types found in literature are detailed below:

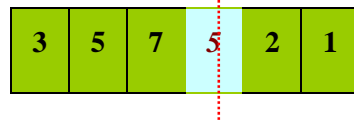
### *Single Point Mutation*

In this type of mutation, a random position of the gene in a chromosome is selected, called '*mutation point*' and the allele of this gene is altered. The gene alteration is generally carried out using Boolean operator of true or false especially for chromosome representation in binary '0' and '1' format. The allele is flipped from one binary value to other in order to create a new child chromosome. Therefore, single point mutation depends on the type of representation and the problem receiving attention. For example use in section 3.3.2.1, the single point mutation will involve changing any random gene (component) value with any other value from the given choice of ten alternatives for this component.

*Parent 1:*



*Child 1:*



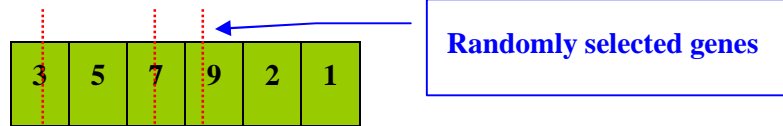
The new child is therefore, created with an alternative choice of five for the fourth component, which is different from the parent chromosome for which the alternative choice was in fact nine for the same component.

### *Multiple Point Mutations*

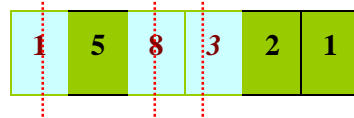
The single point mutation operation can be extended to  $k$ -point mutation, where  $k$  mutation sites are used for altering the gene alleles. For  $k = 3$  and using the same example above, the multiple points mutation process can be applied by selecting three random mutation points in the chromosome string and varying the genes (components) with respect to the given choice of available alternatives. The new child chromosome created using this process contains different combination of

components compared to its parent chromosome. Thus, the new chromosome is a new solution for the optimisation problem and the fitness of this solution can be evaluated as described in section 3.3.2.4.

*Parent 1:*



*Child 1:*



### *Uniform Mutation*

In this type of mutation operation, every gene of the chromosome is selected one at a time and the alteration is made with respect to some predefined level of probability similar to ‘swapping probability’ as in uniform crossover operation. Because of this reason, this type of mutation is also known as probabilistic mutation.

#### **3.3.3.4 Inversion Process**

A gene in a chromosome is recognised by means of its locus and corresponding allele (value). The inversion operation is yet another method along with crossover and mutation which is used to introduce variation in the chromosome structure. It is carried out by inverting the order of genes on a randomly selected segment of the chromosome. The important aspect of this process is the dual change in the structure of a parent chromosome due to the simultaneous changes in both positions (loci) and the alleles (values) of the inverted genes. This interesting feature of this type of variation operation can be observed easily in the structure of the newly created child chromosomes.

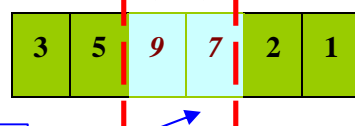
Using the same example of a reliability system with six components and ten alternatives each, the inversion process is applied on a sample parent chromosome and inverting the genes between loci two and five. The result of this process produces a new child chromosome containing genes which are identical to parent chromosome, however, the positions of these two genes are significantly different. At first, this type of variation appears pointless since no new information has appeared to be transferred in the child chromosome which is the requirement of a successful evolutionary algorithm. Looking closely, it will become clear that this type of variation is in fact very useful for some complex optimisation problems. Using the reliability system example above, it can be shown that the structure of the child chromosome is indeed different from its parent since at locus 'three', a new alternative 'nine' has been selected instead of alternative 'seven' which was the case with the parent chromosome. Similar observation can be made for locus four. Also worth mentioning is the phenomenon of disruption introduced in section 3.3.2.5.1, in the context of inversion operation. The disruption of the coadapted genes can be reduced by applying the inversion operator which alters the structures of the parent chromosome by only inverting the genes loci without removing them completely. Therefore, the child chromosome represents a new solution of the optimisation problem and can be tested for fitness.

*Parent 1:*



**Randomly selected  
segment in a chromosome**

*Child 1:*



**Inverted genes**

### 3.3.4 Selection & Replacement Process – The Model of Evolution

Evolutionary algorithms are also known as stochastic optimisation techniques for finding an optimum solution based on the probabilistic selection which increases with the fitness of the individual solutions in a population with respect to the given condition (environment). As detailed in the 3.3.2.4, the selection process is applied to update the current version of a population by making probabilistically biased decisions of selecting individual solutions with strong fitness compared to the solutions with weak fitness in the same population. In other words, a selection process is a mechanism consisting of two main steps: firstly, it is used for choosing individuals with strong fitness as parents from the given population and aspiring for new versions of these individuals (offspring) with even better fitness and secondly, it is used to replace weaker individuals of the population by inserting the new children back in to the population which are formed as a result of the genetic variation processes explained in section 3.3.2.5. This two step process is also known as the *model of evolution*. There are many ways in which a selection process can be applied for selecting strong individuals; some of the strategies commonly known in the literature are detailed below:

#### 3.3.4.1 *Proportional Selection or Stochastic Sampling*

This type of selection is widely known as ‘roulette-wheel’ strategy for selecting the strong individuals from a population and was introduced by Holland (1975). According to this strategy, an individual with higher levels of fitness subsequently, has higher chances of surviving in the next population. The selection process simulates a roulette wheel having one section allocated for each individual in the population and the size of each of this allocated section is proportional to the fitness of the corresponding individual. For a population consisting of ‘ $m$ ’ number of individuals, the roulette wheel is divided into ‘ $m$ ’ sections and if the size of each section is identical, the selection probability of each individual will be identical and uniformly distributed. On the other hand, if the size of the section allocated for one individual is twice the size of other, the selection probability of the first individual will be doubled as a result of covering wider area on the roulette wheel. If an

individual ' $i$ ' has fitness ' $\Omega_i$ ', its probability of selection, ' $P_i$ ' can be evaluated using the following equation:

$$P_i = \Omega_i / \sum_{j=1}^m \Omega_j$$

The selection of all individuals is probabilistic and having higher fitness does not guarantee an automatic selection. However, on average, it appears that the rates at which individuals are selected are generally proportional to fitness levels of these individuals. Because of this reason, there are possibilities of selecting individuals with weak fitness levels causing the evolutionary algorithm to slow down as this process requires extra computation and processing time especially for large scale problems. While this appears to be a valid shortcoming of this strategy, it is also an essential feature of the algorithms working with random search spaces; the uncertainty associated with the direction of the search using evolutionary algorithm towards optimum solution using only the strong individuals can be reduced with random inclusion of weaker individuals. This process diversifies the exploration of the search space and decreases the risk of premature convergence of the algorithm (this is explained in more detail later in the chapter).

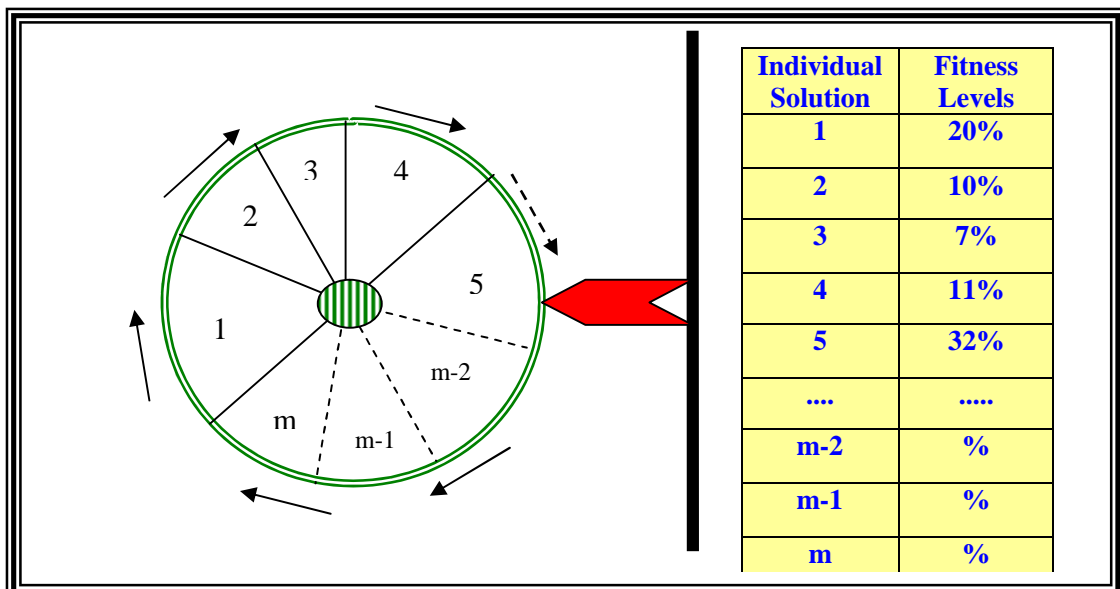


Figure 3.8 Roulette Wheel Selection Mechanism

### 3.3.4.2 *Deterministic Selection*

This strategy is similar to proportional selection and differs mainly because the selection of the individuals is completely biased towards the solutions with strong fitness levels. This type of selection allows the algorithm to perform faster without spending extra computing time on weaker solutions unlike in the previous strategy. While this appears to be an encouraging feature, this type of selection, however, possesses the high risk of converging prematurely to a local optimum instead of the required global optimum for the problem in hand. Since an initial population is created using blind search in the total solution space of an optimisation problem, it is possible for the deterministic selection strategy to select individuals from this population which are associated with strong fitness levels yet they represent points in the search space which are far away from the required global optimum. This phenomenon is called '*sampling error*' and it represents solutions which are located in the suboptimal region of the search space.

### 3.3.4.3 *Tournament Selection*

The tournament selection strategy, as the name suggests, simulates competitions among the individuals of a population during random encounters. This approach contains the features of both stochastic and deterministic selection strategies. The method randomly selects a set of potential solutions (chromosomes) and identifies the best one from the set for generating a new population by entering them into a tournament against each other. The total number of chromosomes in the set is called the '*tournament size*' with two chromosomes set commonly known as '*binary tournament*'. Using this selection scheme,  $n$  tournaments are needed for producing  $n$  individuals for the next population.

### 3.3.4.4 *Elitist Selection*

This type of selection process ensures that the best chromosomes are passed to the next population if not already selected by another mechanism; it is the reason for this method to be part of the deterministic selection process which also favours the best solutions. These members of the population which are guaranteed to survive are called the elite chromosomes. These chromosomes ensure that the fitness function of



the population remains high by producing more offspring with elite genes which increases the domination in the overall population. The latter can also be ineffective if the elite genes are part of the local optimum and considerably far away from the required global optimum.

In addition to the above, there are other less common selection strategies such as, *Truncation Selection*, *Block Selection*, *Rank Selection*, and *Selection by Normalisation*. The details of these methods are profusely documented in the literature of evolutionary computation; along with world wide web, some of the excellent resources are Holland (1975); Goldberg (1989), Faulkenauer (1998), Gen and Cheng (1997, 2000) and Ashlock (2006).

## **3.4 DESIGNING EVOLUTIONARY ALGORITHMS**

There are many ways in which an evolutionary algorithm can be designed, due to its interdisciplinary application in various research fields. However, there exist some basic steps which designers should carefully consider for developing robust evolutionary algorithms capable of efficiently exploring complex search spaces by exploiting all good solutions. These steps are explained below with a view of the optimisation problem (where necessary) introduced in section 3.3.2.1, in which an optimal configuration of components is required for a reliability system, such that the total cost of the selected component is minimum for a predefined target reliability level.

### **3.4.1 Genotype & Phenotype Representation**

Choosing the type of chromosome representation is an important step towards designing efficient evolutionary algorithms. Correct representation of chromosomes ensures accurate transfer of information between genotype and phenotype which essentially drives the search mechanism in evolutionary algorithms. Therefore, a chromosome should be simple and less computationally intensive with the ability to clearly detail the type of solution it represents taking into account all constraints of

the optimisation problem. For the above example, the objective is to find an optimal configuration of ‘ $m$ ’ components in a reliability system from the given discrete choice of ‘ $n$ ’ alternatives (for each component) with the constraint on minimum system cost for a target reliability level.

An acceptable representation of chromosome for this problem can be created using string based data structure containing ‘ $m$ ’ partitions (genes), each representing the component in the system. Also, for correctly displaying the appropriate alternative (allele) of each component (gene), the alleles are able to take any value between 1 and  $n$  from the given set of alternatives of the respective component – See Figure 3.3 for an example of this chromosome representation. Because, the constraint of the optimisation problem requires the sum of each configuration of components (genotype) for establishing the total system cost of the constituted system (phenotype), the selected representation of the chromosome is adequate for converting the genotype information into phenotype for evaluating this constraint.

### **3.4.2 Population Structure**

The productivity of an evolutionary algorithm depends significantly on the structure of its population involving its size and the method of its generation. The size of a population generally reflects the scale of the optimisation problem however, larger size introduces more diversity in the search space but it can be very computationally expensive. Similarly, having too few individuals in the population encourages the EA to converge prematurely on local optimum.

A general approach in the literature appears to be the use of population of size fifty, though it can be varied depending on the individual problem. Another important aspect of the population is to do with its maintenance during each generation. Depending on the model of evolution used in a given EA, it is possible to maintain more than one population, simultaneously in the search algorithm, as seen typically with Genetic Algorithms; one population is for randomly selected parent individuals while the second one is used for breeding offspring. This is different from the steady-

state population (also used for example above) where only one population is maintained throughout and individuals with weaker fitness are instantly replaced by the new progeny with better fitness.

### **3.4.3 Fitness Function**

A fitness function is a measure for evaluating the quality of each random solution found by the evolutionary algorithm. Having a simplified and clear version of this function reduces the complexity of an evolutionary algorithm while improving its ability to find optimum solution. If a fitness function is flawed with incomplete and inaccurate structure, it is very likely that the evaluation of the solution will also be incorrect which could easily lead to a premature convergence of evolutionary algorithm to a suboptimal solution.

The fitness function for the example above is relatively simple in nature since it is designed to compute the sum of all selected alternatives of the components represented in every instance of the random solution (chromosome) which satisfies the target reliability level. Correct evaluation of the chromosomes in each generation of the evolutionary algorithm determines the accurate selection of promising solutions requiring further attention, which generally leads to an optimum solution.

### **3.4.4 Variation Operators**

Given the enormous choice of variation operators found in the literature (section 4.3.2.5), it is important that the selected operators are pertinent to the nature of the optimisation problem under consideration. A common reason for the large choice of these operators is the problem specific nature of these operators; one type of such operator may not be suitable for two different problems. In the context of the example being discussed so far, the choice of uniform crossover and both single and multi-point mutation operators appear to be the best. This is because of the very complex and non-linear relationship between cost and reliability; having one point crossover will introduce greater variation than desired (due to disruption phenomenon) which will make the search for the optimum solution very difficult

because of the destruction of the coadapted genes. This is explained more intuitively in chapter 5.

### 3.4.5 Model of Evolution

The nature of the optimisation problem influences the type of model required for selecting individuals with stronger fitness for procreation and introducing the new found solutions back in to the population for further breeding. Therefore, the model of evolution selected while designing an evolutionary algorithm should closely investigate the quality of each random solution and make all possible efforts to streamline good solutions into even better ones. The latter can be achieved in many ways in the literature, some of which are exploiting the local optimum through hill-climbing using Lamarckian approach (Gen and Cheng, 1997, 2000; and Ashlock, 2006), introducing penalty function for corrupting the fitness of the similar solutions (Coit *et al.* 2000), using niche specialisation for reducing the fitness of the coevolving solutions (Goldberg, 1989) and using repair methods for correcting the infeasibility of the solution (Schonberger, 2005).

### 3.4.6 Termination Criteria

An evolutionary algorithm generally begins with a blind search in the total search space and utilizes various techniques for searching the optimum solution, as explained in section 3.3.2. However, the search can not guarantee the discovery of the global optimum and can continue ad infinitum particularly for very large scale optimisation problems with complex and infinite search spaces. It is therefore imperative to specify termination criteria for the algorithm such that it either finds the optimum solution within a reasonable length of time or ceases the search with the current best solution. Besides run time, other commonly used termination criteria are pre-defined tolerance level of the expected result, total number of generations and the structure of the population (showing no change in the fitness levels of the population members, for example).

### 3.5 TYPES OF EVOLUTIONARY ALGORITHMS

The most promising application of evolutionary algorithms appears to be the field of optimisation (Yao, 2002). It has proven to outperform conventional optimisation methods when applied to difficult real-world problems (Back *et al.*, 1997 & 2000(a) (b); Schwefel, 1994). In comparison with conventional mathematical or heuristic optimisation methods, the evolutionary algorithms are different in two ways; first they are population based and secondly they possess the feature of continuously exchanging the communication and information among the individual in the population (Schonberger, 2005). Despite continuous growth in the field of computation using evolutionary algorithms, there appear to be four main types of such algorithms commonly found in the literature, these are: *Evolutionary Programming (EP)*, *Evolution Strategies (ES)*, *Genetic Algorithms (GA)* and *Genetic Programming (GP)*.

Among the four types of evolutionary algorithm (EA), Evolutionary Programming (EP) tends to follow the general framework of a standard EA. The method of chromosome representation is usually a vector containing real numbers. Each chromosome represents a point in the search space (potential solution). In this type of evolutionary algorithm, the crossover functionality is not employed, instead all variations in the chromosomes are carried out using a mutation operator at random and the selection of parent individuals is made using a probabilistic selection process. A comprehensive literature on EP can be found in Porto (2000).

The structure of Evolution Strategies (ES) is similar to EP and the differences appear to be the use of the deterministic selection process for generating new individuals. Also, unlike in EP, in this method, both crossover and mutation operators are utilised for introducing variations in the parent chromosomes. A good literature on this method is available in Rudolph (2000).



# 4

# GENETIC ALGORITHMS

---

This chapter presents the basic framework of the most commonly known types of evolutionary algorithms called, ‘Genetic Algorithms (GAs)’ which are extensively recognized as one of the most powerful and broadly applicable stochastic search and optimisation techniques by researchers from various fields. The first half of the chapter introduces the notion of classical GA along with its general structure and common features. The later half of the chapter compiles the strengths and weaknesses of GAs, their various types currently found in literature and finally the review of their application in the field of system reliability optimisation.

## 4.1 INTRODUCTION

The idea of Genetic Algorithms (GAs) in computer science is inspired by the observations of the natural process of evolution of species including plants and animals. As described in the previous chapter, the mechanism for generating new creatures within an organism skilfully utilises the knowledge accumulated in the current population of living organism in order to produce new offspring with the same or even better fitness than the parents. The transmission of information (traits) from parent to offspring is commonly known as ‘heredity’ which is biologically performed by the complex structure of the DNA in all living creatures and is the cornerstone of Genetic Algorithms. The rules of inheritance of traits in plants were established in the beginning of the nineteenth century by Gregor Mendel. His observations were specific to the inheritance phenomenon in only one species which were further extended in 1859 by Charles Darwin through his highly controversial theory of new species which essentially described the human race on the same footing as animals. The theories of inheritance and speciation by both Mendel and Darwin were investigated by many researchers for many years and nearly a century later, the actual physical mechanism underlying those theories were identified by physicist Erwin Schrodinger in 1944 which led to the discovery of DNA in 1953 by James Watson and Frances Crick. The principles of natural selection and genetics were employed by Fraser (1957) and Bremermann (1958) in their research areas. However in scientific literature, the first formal introduction to Genetic Algorithms appear to be attributed to John Holland through his revolutionary book, *Adaptation in Natural and Artificial Systems* which was published in 1975 and the theory was further extended by Goldberg (1989). Due to this reason, the research world refers to the structure of the GA provided by Holland and Goldberg as the ‘classical’ GA.

GAs can easily be implemented on a computer for a wide spectrum of problems across various fields. These algorithms are computationally simple yet powerful in their search for improvement and are not fundamentally limited by restrictive assumption about the search space making them particularly useful for solving very complex optimisation problems which are normally cumbersome for direct



mathematical treatment. A comprehensive bibliography on Genetic Algorithms has been compiled by Alander (1999).

## 4.2 GENERAL FRAMEWORK OF GENETIC ALGORITHMS

Usually the structure of a GA varies with the scale of the optimisation problem it is applied to but on the whole it consists of the following generic steps:

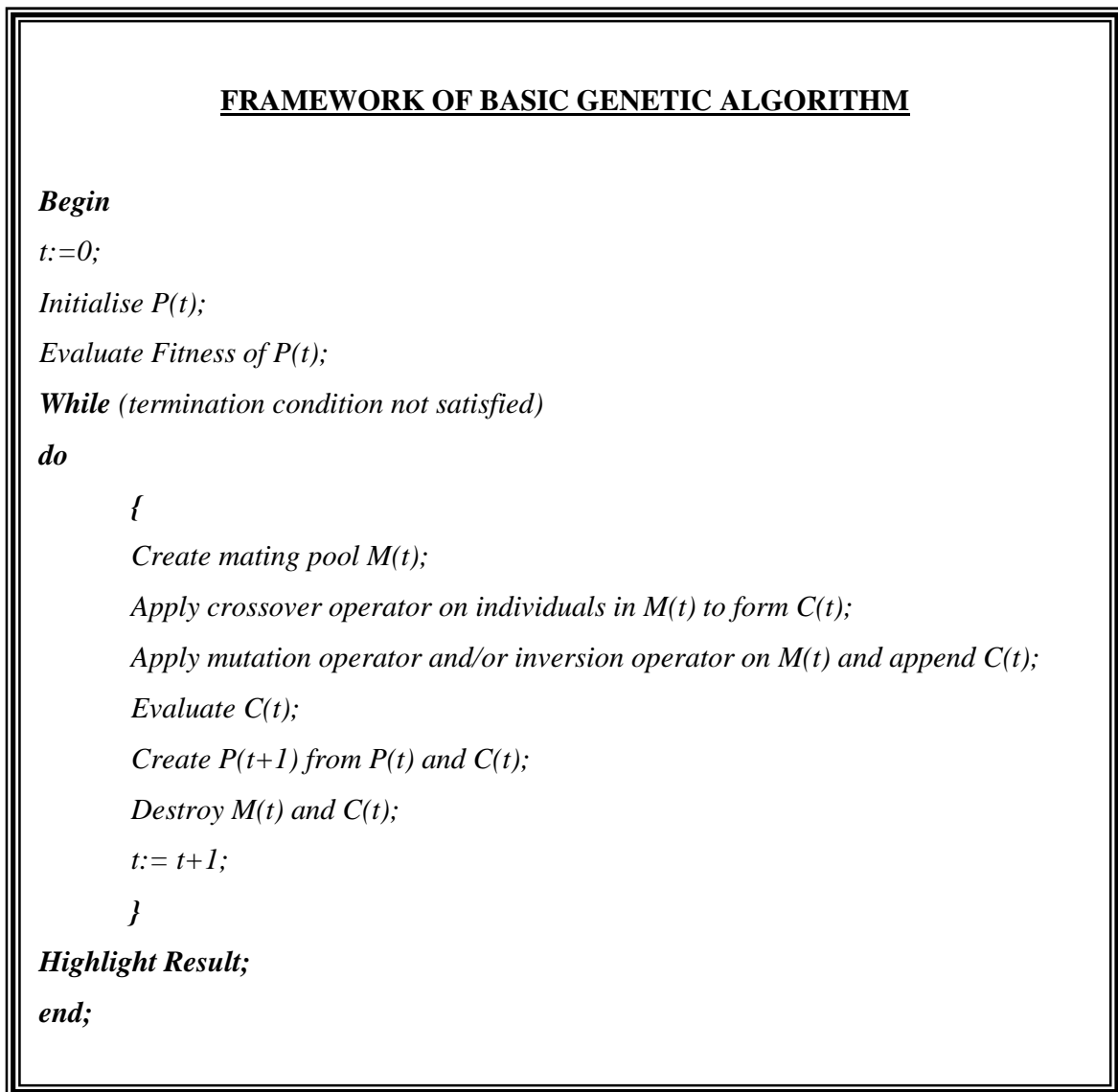
1. An appropriate chromosome representation is selected such that each point in the search space is represented by this chromosome. The search space of all possible solutions of the problem is mapped onto a set of finite chromosome strings over a finite length having fixed data structures. The GA works with the encoded solutions (chromosomes) rather than the actual solutions themselves.
2. Having chosen the chromosome structure, an initial population  $P(t)$ , of these chromosomes is selected at iteration stage ' $t = 0$ ' which is the  $t^{\text{th}}$  generation or population. The initial population  $P(0)$  is usually filled at random; this is because, unlike ordinary optimisation methods, GA performs a parallel search over a set of random points in the search space which reduces the probability of converging to a local optimum.
3. Each member in the population  $P(t)$  represents the potential solution of the problem in consideration and is subsequently evaluated for the level of its fitness using appropriate fitness function
4. For generating next population  $P(t + 1)$ , several individuals are selected as parents from the current population  $P(t)$ , using probabilistic selection which is biased towards higher fitness (e.g. roulette wheel) hence forming a separate population which is also known as mating pool,  $M(t)$ .
5. The parent individuals are expected to transmit their genetic information into the next population and therefore undergo stochastic transformation by means

of crossover operation to form two new individuals. The new individuals are also maintained in a separate population,  $C(t)$  which is different from both  $P(t)$  and  $M(t)$ .

6. In the next stage, another variation by means of mutation operator is applied with a very small probability on the parent individuals in  $M(t)$ . This process introduces very small random changes in a few randomly selected individuals hence forming new progeny of these individuals, which are maintained in  $C(t)$ . In some Genetic Algorithms, this small randomly chosen proportion of the population also endures an additional variation process called inversion which brings alteration in the chromosome string by changing the positions of all the genes. The reason for this step in the algorithm is to introduce diversity in the search space since crossover operator, despite producing new offspring, does not introduce any new traits in the offspring.
7. Depending on the fitness levels of individuals from  $M(t)$  and  $C(t)$ , a new population  $P(t+1)$  is formed by probabilistically selecting individuals with the best fitness levels. The main reason for this step is to replace weaker individuals from the current population  $P(t)$  with the offspring individuals with stronger fitness levels developed in  $C(t)$ .
8. Each offspring in the population  $C(t)$  contains genetic information of at least one parent from the mating pool,  $M(t)$  and is also evaluated for its fitness using the same fitness function which was used for the parent individuals.
9. The new population  $P(t+1)$  is formed at this stage by probabilistically selecting the best individuals from both  $M(t)$  and  $C(t)$  by replacing the weaker individuals from  $P(t)$ . This new population  $P(t+1)$  is ready to continue with the search from step (3). The temporal population  $M(t)$  and  $C(t)$  are destroyed and the iteration counter 't' is increased such that  $t = t + 1$ .
10. The above process continues until the iteration counter stops increasing due to any of the following reasons: if either the given number of iterations has

reached, or the given time span is passed or even if no improvements have been observed within the last few iterations. After several iterations, the population is expected to converge if the frequency of the solutions does not significantly change any more and no new solutions are produced. If the leading genotype contained in the converged population represents high quality phenotype, the Genetic Algorithm is deemed successful.

The general framework of Genetic Algorithms can also be demonstrated via the Figure 4.1.



**Figure 4.1 A simple Genetic Algorithm**

## 4.3 FEATURES OF GENETIC ALGORITHM

The principal use of Genetic Algorithms appears to be in the field of function optimisation however, they are also used effectively in many other fields. As mentioned in (Spall, 2003), GAs are used to study the social systems of human populations in order to investigate the evolution of societies, impact of government policies, resource shortages and human interaction with the environment. GAs can also be used to design simulation based methods for making policy recommendations, as stated in the preface of the 1992 update to the seminal Holland (1975), “Genetic Algorithms are a tool for investigating the phenomena generated by *complex adaptive systems*, a collective designation for nonlinear systems defined by interactions of large numbers of adaptive agents (economics, political, systems, ecologies, immune systems, developing embryos, brains and the like)”. Some of the main features of Genetic Algorithms are detailed in this section.

### 4.3.1 Terminologies and Concepts in Genetic Algorithm

The terminologies across Genetic Algorithms are similar to Evolutionary Algorithms and the details of these can be found in the previous chapter. Likewise, the concepts in GAs are also effectively the same however the application of these concepts is essentially what makes GAs different from any other forms of EAs. For instance, the crossover operation is one backbone feature of the GA and is therefore applied extensively in the algorithm as compared to the mutation operator which is implemented only slightly. The reasons for this approach will be explained later in the chapter. Despite sharing many similarities with EAs, there are some concepts which are found in the literature mainly in the context of the Genetic Algorithms. Some of these concepts are defined in this section.

#### 4.3.1.1 Types of Chromosome Coding

The types of chromosome encoding can be best explained in the context of minimising a loss function ‘ $L = L(\mathbf{v})$ ’; the optimisation problem is to find the best

values for vector ' $\mathbf{v}$ ' belonging to the domain of all permissible values, ' $\Psi$ ', which minimises the ' $L(\mathbf{v})$ '. This can be formulated as,

$$\Psi^* \equiv \left\{ \mathbf{v}^* \right\} = \min_{\mathbf{v} \in \Psi} L(\mathbf{v})$$

where, ' $L(\mathbf{v})$ ' defines the losses of the system for the given composition of the ' $\mathbf{v}^*$ ', which is a  $N$ -dimensional vector of parameters (components) with optimal combination of selected parameters, and  $\Psi \subseteq \mathbb{R}^N$  is the domain for ' $\mathbf{v}$ ' representing constraints on all acceptable values for this vector. The ' $\Psi^*$ ' is the set of values  $\mathbf{v} = \mathbf{v}^*$  that minimises  $L(\mathbf{v})$  subject to ' $\mathbf{v}^*$ ' satisfying the constraints in the set ' $\Psi$ '.

In order to apply the GA operation on the given optimisation problem, the important step is to decide the type of chromosome structure which can be used effectively for encoding and decoding the values of ' $\mathbf{v}$ '. Generally in GAs, the structure of the chromosome consists of string representation, in the form of a sequence of numbers each representing the corresponding value of the parameters. These strings of numbers can be selected in many ways for representing the given structure of vector ' $\mathbf{v}$ '. Among these, standard '*binary bit strings*' (0,1) appear to be the most common type of numbering found in the literature (Goldberg, 1989; Mitchell, 1996; Davis, 1996). The main reasons for extensive use of this type of coding, as indicated in Spall (2003), appear to be the continuation of the classical approach of Holland (1975), relative simplicity of implementing the genetic operations (such as selection, crossover, mutation etc), similarity of binary '0' & '1' manipulation with computer data processing, and the compatibility of binary coding when using with the popular schema theory (detailed in the next section). An excellent approach of binary bit string coding for a scalar  $\mathbf{v}$  is presented in Spall (2003). According to this, encoding/decoding procedures for a scalar  $\mathbf{v}$  can be applied to individual parameters of a vector  $\mathbf{v}$ , in which case the procedures will be associated with one gene in the chromosome. The sample approach also shows the standard feature of the GA where the number of bits representing each gene can also be different. The details of the approach presented by Spall are stated below:

Let ‘ $b$ ’ be the number of bits representing one of the scalar elements in vector  $\mathbf{v}$ , then the minimum value of this scalar  $\mathbf{v}$  is represented by all zeros in each of the bit positions as in  $[0,0,0,\dots,0]$  and the maximum value with ones in all bit positions, i.e.  $[1,1,1,\dots,1]$ . If the total number of bits in the chromosome are represented by ‘ $B$ ’, then ‘ $B$ ’ will be greater than ‘ $b$ ’ since it corresponds to all  $N$  elements in vector  $\mathbf{v}$ .

### *Steps for encoding (scalar $\mathbf{v}$ )*

1. Let  $\mathbf{v}_{\min}$  and  $\mathbf{v}_{\max}$  be such that  $\mathbf{v}_{\min} \leq \mathbf{v} \leq \mathbf{v}_{\max}$  and let ‘ $m$ ’ represent the maximum number of positions after the decimal point such that  $m < 0$  symbolises positions before the decimal. Select number of bits ‘ $b$ ’ such that it is the smallest number satisfying the relationship  $10^m (\mathbf{v}_{\max} - \mathbf{v}_{\min}) \leq 2^0 + 2^1 + 2^2 + \dots + 2^{b-1} = 2^{b-1} - 1$ , for the number of possible representation for a string of length  $b$  - bits.
2. Let  $d = (\mathbf{v}_{\max} - \mathbf{v}_{\min}) / (2^b - 1)$ . Each increase in  $\mathbf{v}$  by an amount  $d$  increases the bit representation by one unit.
3. Round off the given  $\mathbf{v}$  to the nearest integer using the operator  $\text{round}[(\mathbf{v} - \mathbf{v}_{\min}) / d]$  and represent it using the standard binary format  $[a_1, a_2, a_3, \dots, a_b]$  where the members  $a_i$  are either 0 or 1.

### *Steps for decoding (scalar $\mathbf{v}$ )*

1. Assuming a  $b$  -bit representation  $[a_1, a_2, a_3, \dots, a_b]$  derived as in the encoding steps above.
2. The value of  $\mathbf{v}$ , to specified accuracy ( $m$ ) is given by,

$$\mathbf{v} = \mathbf{v}_{\min} + \frac{\mathbf{v}_{\max} - \mathbf{v}_{\min}}{2^b - 1} \sum_{i=1}^b a_i 2^{i-1}$$

***Numerical Example by Spall***

The approach of using the binary bit representation is shown using a  $\mathbf{v}$  with two components (i.e.  $N = 2$ ).

$$\text{Let } \mathbf{v} = [t_1, t_2]^T$$

Such that

$$t_1 \in [-4.00, 10.00] \text{ \& } t_2 \in [1000, 4500].$$

For the first component of  $\mathbf{v}$ ,  $m = 2$  and  $b = 11$ , since

$$2^{10} - 1 = 1023$$

$$10^m (\mathbf{v}_{\max} - \mathbf{v}_{\min}) = 10^2 (10.00 - (-4.00)) = 1400$$

$$2^{11} - 1 = 2047$$

$$\therefore 1023 \leq 1400 \leq 2047$$

For the second element,  $m = -2$  and  $b = 6$ , since

$$2^5 - 1 = 31$$

$$10^m (\mathbf{v}_{\max} - \mathbf{v}_{\min}) = 10^{-2} (4500 - 1000) = 35$$

$$2^6 - 1 = 63$$

$$\therefore 31 \leq 35 \leq 63$$

if  $\mathbf{v} = [-2.31, 4300]^T$ , an encoding would be

[0 0 0 1 1 1 1 0 1 1 1 ; 1 1 1 0 1 1] where the semicolon separates the two genes for the two elements of  $\mathbf{v}$ . The value of  $d$  for the first gene is 0.00684 and integer value

$$\text{round}[(-2.31 - (-4.00)) / 0.00684] = 247$$

for encoding [0 0 0 1 1 1 1 0 1 1 1] with  $b=11$

$$2^0 + 2^1 + 2^2 + 0^3 + 2^4 + 2^5 + 2^6 + 2^7 + 0^8 + 0^9 + 0^{10} = 247$$

Similarly, the values of  $\mathbf{v}$  can be decoded using the steps above.

$$\mathbf{v} = \mathbf{v}_{\min} + \frac{\mathbf{v}_{\max} - \mathbf{v}_{\min}}{2^b - 1} \sum_{i=1}^b a_i 2^{i-1}$$

$$\mathbf{v} = -4.00 + (10.00 - (-4.00))(2^0 + 2^1 + 2^2 + 2^4 + 2^5 + 2^6 + 2^7) / (2^{11} - 1)$$

$$\mathbf{v} = -2.311$$

Similarly,

$$\mathbf{v} = 1000 + (4500 - 1000)(2^0 + 2^1 + 2^3 + 2^4 + 2^5) / (2^6 - 1)$$
$$\mathbf{v} = 4277.8$$

The decoded values of  $\mathbf{v}$  are identical to the target values with specified level of accuracy, expressed in terms of  $m$ .

An alternative to binary bit style coding is ‘gray coding’, which also uses the (0,1) alphabets in the string representation but differs in the way in which bits (i.e. **binary digit**) are arranged. As described in Spall (2003), it is an alternative coding scheme which attempts to closely match the bit representation with the natural characteristics of the optimisation problem space, particularly when the decimal accuracy between the adjacent values is required. Because the adjacent floating point values differ by only one bit in the chromosome string, it is expected that in gray coding, small changes in  $\mathbf{v}$  can be accomplished more easily compared to binary representation. The latter may have a very different representation when moving from one adjacent value to another, for example, if  $\mathbf{v}$  is an integer valued scalar quantity, then a move of one unit from  $\mathbf{v} = 7$  to  $\mathbf{v} = 8$  would require all four bits [0 1 1 1] to change to [1 0 0 0]. Therefore, the probability of simultaneously changing several bits to produce a small change will also be small since GA operates by flipping the individual bits in the chromosome string for carrying out genetic tasks such as crossover and mutation. There appears to be no strict criteria for forming gray code as indicated in Spall (2003). However, a good source for understanding the translation between binary and gray coding can be found in Michalewicz (1996). A short sample of this is demonstrated in Table 4.1 where it can be seen that the gray code changes more gradually than standard binary code with the changes in the integer representation. Also, in most cases, this change in gray code is limited to only one bit for each one unit change of the integer value.



Along with binary and gray coding, ‘*multiple character encoding*’, which contain more than two elements in the string alphabet is yet another type of chromosome encoding which is found in the literature. This type of encoding includes the complete ten character representation and is also known as ‘*real number coding*’ and works directly with the parameters of  $\mathbf{v}$  since each value of the parameter is represented as a real number in the string. Due to this feature, the real number encoding appears to be growing in use and is found in many successful numerical implementations (Spall, 2003).

<b>Integer Value</b>	<b>Binary Code Representation</b>	<b>Gray Code Representation</b>
0	0 0 0 0	0 0 0 0
1	0 0 0 1	0 0 0 1
2	0 0 1 0	0 0 1 1
3	0 0 1 1	0 0 1 0
4	0 1 0 0	0 1 1 0
5	0 1 0 1	0 1 1 1
6	0 1 1 0	0 1 0 1
7	0 1 1 1	0 1 0 0
8	1 0 0 0	1 1 0 0
9	1 0 0 1	1 1 0 1
10	1 0 1 0	1 1 1 1

**Table 4.1 Comparison of Binary and Gray Coding (Spall, 2003)**

### 4.3.1.2 Similarity Templates or Schemata

One of the most important concepts behind the theory of GAs is similarity templates or schemata (plural for schema). As defined by Holland (1975) and Goldberg (1989), a schema is a pattern matching device containing a subset of strings with similarities at certain string positions. In other words, a schema represents the constant values of some of the genes in the chromosome which are identified among highly fit strings in order to help guide the search towards best possible solution. For example, a chromosome string of length six using binary representation can contain either '0' or '1' as an allele for any of the six genes of the chromosome – see below.

1	0	1	1	1	0
---	---	---	---	---	---

Let's imagine, the chromosome strings producing promising solutions have the following alleles of the genes

1	0	1	1	1	0
0	1	1	1	0	0
1	1	1	1	0	1

Then it can be seen from the three chromosomes strings that loci three and four in each string has common allele of '1' for each respective gene. This information can therefore be utilised to form a template for evaluating all possible combinations of strings matching this pattern.

?	?	1	1	?	?
---	---	---	---	---	---

In the above template, the '?' or 'don't care symbol' represent any combination of '0' or '1' in the chromosome string with fixed values of '1' at locus three and four.

There are two important characteristics associated with the concept of schema. First is the *order* and the second is its *defining length*. The order of the schema represents the number of genes with fixed values of the alleles in a schema definition. Therefore, the order of the similarity template defined above is two since it has two genes out of six with fixed values of the respective alleles. The defining length of the schema represents the distance on the chromosome, measured in number of genes, between the first and the last gene that defines the schema. For the above example, the length of the schema is one which is the difference of position 4 and position three ( $4 - 3 = 1$ ).

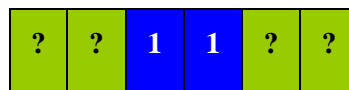
#### ***4.3.1.2.1 Use of Schemata in Genetic Algorithms***

In the literature, the schemata also appear to be referred to as the building blocks of Genetic Algorithms because of their ostensible ability to guide the search towards the optimal solution; a claim which is not commonly agreed by the researchers (Spall, 2003). In general, schemata are found to be associated with two primary theoretical results first of which is known as Schema Theorem while the second is cited as Implicit Parallelism (also known as intrinsic parallelism in Holland, 1975).

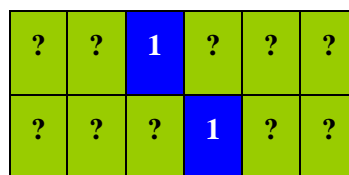
##### **4.3.1.2.1.1 Schema Theorem**

For a given optimisation problem, a search space is a collection of all possible solutions of this problem and thus an arbitrary solution can be considered as a point in this search space. A chromosome encoding of this point (genotype) contains genes which can be used to represent the dimension (identified by its locus) and its value (allele) to the coordinate along this dimension therefore, making this a vector in the multidimensional search space, where each dimension can be considered as a hyperplane in the search space. For the example chromosome structure above, the vector represents six genes each corresponding to a dimension in the total search space with their values representing the respective coordinates. Therefore, each point in the sample space lies at the intersection of many hyperplanes and constitutes a sample of all the hyperplanes that contain it – the quality of this solution will determine the quality of each of the genes representing the respective hyperplane.

For this reason, the search for optimum can be imagined as the search for the best intersection of hyperplanes. The concept can be extended to represent each hyperplane by a schema on a chromosome and vice-versa (Falkenauer, 1998) which leads to the hypothesis that searching for an optimum is about searching for the best schemata. Each chromosome can be a sample consisting of various schemata (with different schema orders). Thus, a GA can utilise this information for searching the optimum and by targeting only the high-order schemata where the defined area of the search space is smaller. This is because, a schema of the highest order with maximum number of coordinates (hyperplanes) of the search space, can almost certainly define a single point as the global optimum if it performs better than all other schemata. However, there may be substantially large numbers of high order schemata for a given optimisation problem and evaluating each one of them can be unfeasible given the size of the search space. It is important to mention that high order schemata are in fact comprised of lower-order schemata themselves. For example, the two order schema



contains the following two schemata each with order one:



If both of the lower order schemata perform better with the combination of genes made up of these templates, then the higher order schema would automatically contain all solutions derived from these lower order schemata. As a result, the higher order schema will be selected for further evaluation with a higher probability for searching the global optimum of the optimisation problem. Thus, lower order schemata provide the building blocks for constructing the promising higher order

schemata but once again, the total number of these schemata can be very large and certainly impractical to be evaluated rigorously (e.g. using an enumerative search). In GAs, this issue of finding the promising schemata is resolved through evaluation of the optimisation function using random trials and finding the best balance between identifying the promising schemata (exploration) and searching for the best solutions derived from these schemata (exploitation). A common known strategy for trial allocation found in the GA literature (Holland, 1975; Goldberg, 1989; Michaelwicz, 1996) is one that allocates an exponentially increasing number of trials to the best found schemata, which will dominate across generations, as depicted in Equation 5.4 below.

Under the basic GA operation with proportional selection (roulette wheel) with basic genetic operators (reproduction, crossover, mutation), the strategy for the trial allocation can be defined using the following assumptions, as defined in Falkenauer(1998). Let ‘H’ be a schema template, such that the members of this schema consistently perform better than the non-members by an amount ‘c’. Therefore, if the average performance of the whole of the population is ‘f’ then the average performance of the members of ‘H’ can be represented as,

$$f(H) = (1 + c)f \text{ or } \frac{f(H)}{f} = 1 + c \quad (4.1)$$

Also, if there are ‘ $m(H, k) \leq N$ ’ number of chromosomes at generation ‘k’, which are the instances of the template ‘H’ in the population of size ‘N’, then the expected number of these chromosomes in ‘k + 1’ generation can be estimated by,

$$m(H, k + 1) = m(H, k) \frac{f(H)}{f} = (1 + c)m(H, k) \quad (4.2)$$

which is the survival rate of this schema from one generation to the next, following a geometric progression with exponential rate of growth. The above relationship can be extended by adding the crossover and mutation operators and assuming zero selection of elite chromosomes:

$$m(H, k+1) \geq m(H, k) \frac{f(H)}{f} \left[ 1 - p_c \frac{\delta(H)}{l-1} - p_m o(H) \right] \quad (4.3)$$

In the above relationship, the ' $p_c$ ' is the probability of crossover such that  $0 \leq p_c \leq 1$ , ' $\delta(H)$ ' is the defining length of the schema 'H', ' $l$ ' is the length of the chromosome, ' $p_m$ ' is the mutation probability (which is generally very low ; i.e.  $p_m \approx 0$ ), and ' $o(H)$ ' is the order of the schema, 'H'. Using this relationship, the lower bound to the rate at which the domination will occur across generations under the schema theorem for the basic GA from Section 5.2, above, can be stated as,

$$\min \left\{ m(H, k) \frac{f(H)}{f} \left( 1 - p_c \frac{\delta(H)}{l-1} - p_m o(H) \right), N \right\} \leq m(H, k+1) \leq N \quad (4.4)$$

As mentioned in Spall (2003), the 'min' operator ensures that the expected number of chromosomes (which are the instances of the template 'H' in the population of size ' $N$ ') in ' $k+1$ ' generation are equal to ' $N$ ' if the bound would otherwise be larger than the population size. As the order of the schema 'H' increases, the number of 'don't care symbols', '?', decreases as a result of this, the products of the terms, ' $p_c \times \delta(H) / l - 1$ ' and ' $p_m \times o(H)$ ' will be nearly zero in most circumstances. Therefore, equation (5.4) implies that a schema 'H' with consistently higher fitness value than the possibly increasing average of all the chromosomes (i.e.  $f(H)/f > 1$ ) will dominate the new generations by continuously producing chromosomes based on 'H'. The Equation also shows that short, low-order with above average schemata are propagated from one generation to another, being represented by an exponentially growing number of chromosomes, which verifies the

use of trial allocation strategy used in the GA, to be optimal for exploring extremely large search spaces (Falkenauer, 1988). To all intents and purposes, this equation represents the schema theorem, also known as the fundamental theorem of generic algorithms and provides useful insight into the underlying mechanics of the classical GA described in Section 4.2, above.

#### **4.3.1.2.1.2 Implicit Parallelism**

This is the second most referenced schema result found in the GA literature and also referred as intrinsic parallelism in Holland (1975). As stated in Spall (2003), an implicit parallelism states that the number of schemas processed in one generation of the GA is much larger than the actual size of the population suggesting that the algorithm is capable of processing more information at each generation than would be suggested by the population size alone. What's more, this implicit information is available without additional storage and/or processing requirements. The reason for processing a greater number of schemata than the size of the population is because a given chromosome can be associated with various schemata alone. Therefore, each chromosome can be used to assess the quality of many specific schemata. Detailed review of many derivations of implicit parallelism bound on schemata can be found in Spall (2003).

To summarise the two schema results above, there appears to be a considerable controversy about the implications of these results on practical implementations of GAs, as indicated by Spall. Because many promising schemata are processed in a particular generation of a GA which may or may not be significant for reaching an optimum solution, the concept of schema is not absolutely appealing (and reliable) for solving large scale complex optimisation problems. Nevertheless, schema theory has historical significance in the development of evolutionary computation and to an extent, provides some intuitive justification for the good performance that is frequently observed. An excellent source for one of the most up to date literatures on genetic algorithm is provided in Burke and Kendall (2005).

#### **4.3.1.3 Niche Specialisation**

The concept of niche specialisation was first introduced by Goldberg (1989) and is inspired by the concept of biological niches. He suggests reducing (dividing) the

fitness of a member of an evolving population in proportion to the number of other solutions that are essentially the same. In a real function optimisation, this might be the number of solutions which are found in the very close proximity of each other in the domain space. The effect of this is to make solutions less promising once they have been discovered by several members of the population. As detailed in Ashlock (2006), niche specialisation reduces the accumulation of solutions onto a good, but suboptimal, solution found near the beginning of the GA search. In theory, once the niche is filled, it becomes hard for new species to enter the niche. This is because the existing members of the niche are already using the resources it contains. There are many in which niche specialisation can be applied depending on the details of how clearly and accurately similarity measure is defined among solutions and how the number of similar solutions is transformed into a fitness penalty. According to Ashlock, there exists two obvious similarity measures for real functions optimisation. First of these is called the *Domain Niche Specialisation (DNS)*, which measures similarity by comparing the chromosomes of population members. Whereas, the *Range Niche Specialisation (RNS)* is the second type which measures similarity by comparing fitness. The DNS is used to make the optimum less attractive by dividing this solution by a penalty function based on the number of solutions existing in the nearby population. The method employed in RNS is similar to DNS with exception to the way in which penalty function is applied – it measures the difference between the fitness of the optimum solution with respect to the fitness of the neighbouring solutions within a specific tolerance level (similarity radius). In other words, RNS simply computes the number of solutions that have found roughly the same function value. The two types of niche specialisation have different strengths and weaknesses; for this reason, it is sensible to use them sporadically. More details on the two methods along with their applications on real functions optimisation problems can be found in Ashlock (2006).

### **4.3.2 Strength and Weaknesses of Genetic Algorithms**

Genetic Algorithms are powerful stochastic search techniques which have been applied successfully in many optimisation problems across various fields. GAs are different from other search techniques mainly because they are population based



algorithms and are well suited for parallel processing. Working with the population of solutions instead of a single solution, strengthens the capabilities of the GAs because the search process begins by evaluating different initial points (solutions), selected randomly (generally) from the search space, and then gradually converges (if successful) towards optimal solution by eliminating the infeasible solutions using genetic operations. Also, the final population contains the best found solution along with other suboptimal solutions which can be considered alternatively if the best solution can not be implemented for some reason. GAs can easily be implemented on a computer for a wide spectrum of problems across various fields. These algorithms are computationally simple yet powerful in their search for improvement and are not fundamentally limited by restrictive assumption about the search space making them particularly useful for solving very complex optimisation problems which are normally cumbersome for direct mathematical treatment.

Despite many attractive features of genetic algorithms there appears to be some weaknesses associated with this method which need to be taken into consideration when applying the GA methodology. Few of the most commonly found weaknesses in the GA literature (Schonberger, 2005) are briefly explained below:

#### **4.3.2.1 Convergence of Genetic Algorithms**

Genetic Algorithms are population based methods and generally deemed successful when the population after a specific number of iterations, converges to an optimum solution. If the size of a chromosome is very large, the progress of GA can be adversely affected due to the large number of different genes requiring extra additional efforts for experiencing the key genetic operations such as crossover and mutation. Consequently, it increases the processing time of the GA computation which in turn decelerates the rate of convergence or causes the genetic search to miss the convergence completely. This is because the promising solutions are not able to dominate the population since the frequency of occurrence of such solutions gets too small for a statistical dominance and for undergoing genetic operations. Besides the large numbers of genes in a chromosome, the other important factor which causes GA to miss convergence is the epistasis phenomenon (Naudts *et al.*, 1997; Mattfeld,

1996). As mentioned in Schonberger, epistasis contradicts the building block hypothesis of Goldberg (1989) which assumes that most promising solutions are derived from low order with above average fitness schemata. Similar to missing convergence, another shortcoming of the GA is a premature convergence of the genetic search on a suboptimal solution. One of the reasons for this can be associated to an inadequate method for measuring the fitness function. Encoded solutions having similar phenotypes but with slightly different genotypes, are sometimes overestimated and subsequently selected more favourably for transmitting their genetic information into the next population. This deficiency in the fitness measure can promote the abundance of such genotypes in the population which can lead to a premature convergence of the genetic search due to reduced diversity in the search space. Another reason for premature convergence as explained by Schonberger, is the type of chromosome representation used in a GA. On occasions, it is not possible to find bijective string coding of solution instances due to one-to-many relationships between phenotype and genotype; different genotypes with the same phenotype structuring as explained in Falkenauer (1998) who referred to this type of coding as 'redundant encoding'. With this type of encoding structure, there is a high probability of assigning the offspring to the same phenotype and for a large level of such redundancy can potentially invalidate the use of genetic operators for making variations in the genetic code.

#### **4.3.2.2 Feasibility of Solutions**

In addition to convergence issues, other known weaknesses of genetic algorithms is to do with solutions which are found to be unacceptable with respect to the constraints of the optimisation problem in hand. According to Schonberger, these constraints separate the set of genotypes into two main categories: *feasible genotypes*, which can be decoded into feasible phenotype and *infeasible genotypes*, in which the decoded phenotype violates some or all of the constraints. A detailed overview of feasibility issues and ideas for handling infeasibilities including repairing and improving of the genetic code, is presented in Schonberge (2005) and Coello (2002).

### 4.3.3 Comparative Analysis of Genetic Algorithms with Other EC Methods

Genetic algorithms belong to the class of Evolutionary Algorithms which also has many other types among which the two most commonly known methods are, Evolutionary Strategy (ES) and Evolutionary Programming (EP). Like GA, both ES and EP are population based methods and despite having many similarities, all of the three approaches can be compared distinctively.

Evolutionary Strategies were originally designed by Rechenberg (1965) for dealing with constrained continuous variable optimisation problems whereas, GAs were essentially proposed to target machine learning and studying adaptive systems and subsequently proven to be equally useful for both discrete and continuous variable optimisation problems (Spall, 2003). ES are generally known for working directly with function parameters unlike in GAs where solutions are evaluated in the form of a genotype (encoded representation of parameters) and phenotype (decoded solution). There appears to be two commonly known notations associated with ES, which are found in the literature. The first of which is referenced as  $(N + \lambda) - ES$  while the second one is represented by the form,  $(N, \lambda) - ES$  where ‘ $\lambda$ ’ corresponds to the offspring produced in the initial population of the ES and ‘ $N$ ’ is the size of the population. The basic steps of ES for both of the two notations are similar in the sense that for both versions, the first step is to create an initial population of potential solutions (selected randomly, in general) and evaluating the objective function with constraints, the second step is to generate offspring ‘ $\lambda$ ’ from the current population size of ‘ $N$ ’ solutions which satisfy the objective function. However, it is the next step where the algorithm differs for the two versions of the ES. For  $(N + \lambda) - ES$ , the algorithm generates a new population by selecting  $N$  best solutions from the combined populations of old and new ( $\lambda$ -offspring) solutions. Similarly for  $(N, \lambda) - ES$ , the next population is generated by selecting  $N$  best solutions from the population of  $\lambda > N$  offspring only. The life of each

population member in this version of ES is limited to only one generation this is because the solutions for the next population are selected only from the  $\lambda$  offspring. The final step in the ES algorithm defines the appropriate stopping criteria. More details of the processes in ES can be found in Spall (2003) and Rudolph (2000).

Evolutionary Programming, introduced by L.Fogel *et al.* (1966), was intended for evolving artificial intelligence by creating finite-state machines that are adept at prediction (Spall, 2003). These machines can be represented as directed graphs for predicting the next symbol in the sequence of symbols. As detailed by Spall, if some real system has generated output ' $s_1, s_2, \dots, s_n$ ', then the machine can be used to predict the next sequence, ' $s_{n+1}$ '. In EP, the population is generally composed of these finite state machines, where each one of them is represented in matrix form. The fitness of each machine is evaluated by comparing the predictions from the machine with real outcomes using an appropriate fitness function. Using only the mutation operator, EP generates offspring and selects individuals for the next population until termination criteria is achieved. A comprehensive survey of literature on EP along with its applications in real function optimisation is available in Fogel (2000).

As discussed above, the comparison of GAs with other EC methods such as ES and EP can be done by highlighting the main differences in the three algorithms. These being: the method of chromosomes encoding (e.g. binary bit structure in GA) and the order in which the genetic operators are applied on the respective populations. In GA, parents are generally selected before applying the genetic operators such as crossover, mutation and reproduction unlike in ES where crossover operator is used first to create parent and then mutation is employed to produce  $(N + \lambda)$  or  $\lambda > N$  offspring. Another interesting comparison is outlined by Spall (2003) referring to the emphasis on general constrained problems in the ES and EP. According to him, 'these algorithms allow for a direct check on constraint violation and the exclusion of an offspring that violates the constraints. The coefficients of the algorithms may automatically be adjusted if the constraints are violated too frequently. In contrast,

the GA is largely used with simple hypercube constraints, although it is possible to modify the fitness function to include a penalty function as a way of handling more general constraints'. Spall also indicates that the differences between all EC methods are gradually reducing because of the introduction of many hybrid versions which combine attractive features from different evolutionary methods. A comprehensive literature in this area includes resources such as Schwefel (1995), Michalewicz (1996), and Fogel (2000).

## 4.4 TYPES OF GENETIC ALGORITHMS

There are many versions of GA which can be found in the literature and while they seem to possess the basic framework of a classical GA (Holland, 1975; Goldberg, 1989), there are slight variations depending on the research area of their application. These differences are, for example, the way genotype population is selected and maintained (Falkenauer, 1998), the probability of applying the genetic operators (Goldberg, 1989, 2002; Booker *et al.*, 1997; Spears, 1997; Syswerda, 1989; Spears and De Jong, 1994; Back *et. al.*, 2000(a) & (b); Goldberg and Sastry, 2002) and the methods of improving the infeasible solutions by employing kind of repair procedures on local optima.

In the literature, the latter modified versions of the classical GA are also known as, 'Memetic Algorithm' (Moscato, 1989, 1999, 2001; Krasnogor and Smith, 2005; Krasnogor *et al.*, 2004; Moscato and Cotta 2003; Schonberger, 2005; Burke *et al.* 1996, 1999, 2001) or more popularly, 'Hybrid GA (HGA)' (Joines and Kay, 2002; Louis and Mcdonnell, 2004; Burke and Newall, 1999 and Ibraki, 1997 who called this Genetic Local Search). An excellent review of genetic search can be found in Burke and Kendall (2005) and Ashlock (2006) along with a comprehensive bibliography on Genetic Algorithms, compiled by Alander (1999).

## 4.5 APPLICATION OF GENETIC ALGORITHMS IN RELIABILITY OPTIMISATION

In the context of solving reliability optimisation problems, the method of using genetic search has been widely employed due to its robustness and capability to efficiently explore and exploit the search space (see also, Chapter 2).

Painton and Campbell (1995) adopted a genetic algorithm approach to solve a reliability optimisation problem for a system with series-parallel configuration. The objective of the optimisation was to maximise system reliability for a linear cost constraint. For an additional constraint of weight along with cost, Coit & Smith (1996) worked on the redundancy allocation problem in parallel-series systems in which each subsystem was a k-out-of-n:G system, using methodology based on genetic algorithm when the components were chosen from a finite set, assuming different types of redundancy levels such as active, standby and k/n. Their method found feasible solutions for all 33 problems presented previously in the referenced article. The latter had managed 30 solutions. Additionally, the level of reliability for a given cost constraint was improved in most of the highlighted problems. With an objective of maximising the lowest percentile of the system time to failure, modelled by Weibull distribution, Coit and Smith (1998) also solved the redundancy allocation problem in the series-parallel system by using genetic algorithm along with bisection search method for searching the potential solution space. Their findings in this paper and also in 2002 show that the Weibull scale parameters are uniformly distributed random variables, different to the shape parameters which can be estimated exactly.

The Ida *et al.* (1994) and Yokota *et al.* (1995) designed a genetic algorithm for optimal redundancy allocation in a series system in which the components of each subsystem were also subject to two classes of failure modes. Majety and Rajagopal (1997) developed an evolution strategy based on an adoptive penalty function to solve some cost optimisation problems for a minimum level of required system reliability. They applied this strategy on both series-parallel and parallel-series

systems. Dengiz *et al.* (1997) designed a genetic algorithm for cost-optimal network design. A similar algorithm was developed by Deeter and Smith (1998) for cost-optimal network design but unlike Dengiz *et al.* (1997), they assumed that multiple choices for each link in a network exist. The problem considered was to design a network using all available links to minimise the total cost of the links for a given constraint of a minimum reliability.

In both Cantoni *et al.* (1999) and Zio (2000), an excellent methodology based on Monte Carlo simulation and genetic algorithm is proposed for solving complex plant (e.g. Shale oil) design problems. With choices on the type of components to be used and the assembly configurations, the optimisation process is subject to conflicting interaction of reliability/availability objectives with the economic costs associated to the design implementations, system construction and future operation.

A very interesting work has been communicated by Kumral (2005) regarding reliability optimisation of a mine production system using genetic algorithm. The optimisation process is required to estimate the minimum level of reliability for each sub-system along with incorporating a cost minimisation criterion for the risk associated with these uncertain estimates in order to avoid critical losses from the standpoints of safety, quality, health, environment and finance, as described by Kumral.

Analogous to this approach, Yang *et al.* (1998) also use genetic algorithm for reliability allocation in nuclear power plant while minimising the total plant costs subject to the overall plant safety goal constraint with a different approach than Kumral by using fault trees and probabilistic safety assessments for evaluating target reliabilities of individual subsystems. The optimisation processes in the last two sources, despite being conceptually similar in some ways to the risk based reliability allocation method, are however, significantly different because of the minimum reliability requirements and no consideration of the amount of total losses from failure in allocating the optimal level of system reliability. Similarly, Brown *et al.*

(1997) provide useful information about designing an automated primary distribution system by optimising both cost and reliability. The objective function (total cost of reliability) is the sum to two costs, utility cost of reliability and customer cost of reliability. By using methods such as integer programming, genetic algorithm and simulated annealing along with some hybrid methods, the authors minimise the objective function for demonstrating its use as a tool for helping engineers design a reliable distribution system while minimising costs.

Genetic algorithms are also generating a great deal of interest in optimisations of multistate systems (MSS). Characterised by availability, cost and nominal performance rate, the state of the components in such systems facilitate various performance levels at which these systems can carry out their operations. Lisnianski *et al.* (2000), solves structured optimisation of MSS in reference to time redundancy using a GA based strategy. Another strategy, combined with GA, is also used to solve survivability optimisation by Levitin & Lisnianski (2001) in a series-parallel system with a constraint on separation cost for separating the elements of the system. Levitin (2007) also provide a comprehensive list of all GA based publication in the filed of reliability on his website.

An interesting recent development in the application of GAs is the combination of these algorithms with other optimisation methods (heuristics) for producing even more powerful search techniques. These are referred as hybrid GAs and are generally combined with some local search methods in order to improve both solution quality and computational efficiency while preserving the major properties of classical GA such as robustness and feasibility. The optimisation algorithm produced in this research also designed on the same platform, which combines the exploration capabilities of genetic search with exploiting hill climbing procedures in order to resolve the optimisation problem presented in this thesis. Among various publications, the Hsieh and Hsieh (2003) use GA with steepest decent method to optimise system cost during the period of task execution for a cycle-free computer distribution system. Using hybrid GA, Hsieh (2003) also solves similar optimisation problem based on the constraints on the hardware redundancy level. By



incorporating neural networks, fuzzy logic and local search with classical GA, Lee *et al.* (2001, 2002a, 2002b), show the reliability design optimisation which considerably improves the computational time.

An excellent resource for the review of reliability optimisation with the view of genetic algorithms can be found in the two books published by Gen and Cheng in 1997 and 2000. Their work described many GA approaches for solving reliability design problems in the areas such as network reliability design, tree-based network reliability design, bi-criteria reliability design (multi-objective optimisation) of redundant system formulated as nonlinear integer programming problems and problems with fuzzy goals. Most of these problems with constraints on reliability can be applied in the fields of telecommunications and computer networking along with other important networks such as gas, power and sewer networks. Additional resources detailing the application of genetic algorithms in the field of reliability optimisation include Kuo *et al.* (2001), Smith (2006), Kuo & Wan (2007), Ken & Kim (1999) and an outstanding list of all GA based publication in the field of reliability by Levitin (2007).



# 5

## **OPTIMISATION ALGORITHM (THE RESEARCH METHODOLOGY)**

---

This chapter details the proposed optimisation algorithm (OA) used as the research methodology for optimising engineering systems using the risk-based reliability allocation approach. The algorithm is a member of a class of evolutionary algorithms and is specific to the reliability optimisation problems examined in this research. The OA employs a different model of evolution compared to classical GA in order to ensure quick and efficient convergence to an optimal or near optimal solution. The chapter details the optimisation algorithm, its main features and finally the process diagram detailing the full cycle of operation.

## 5.1 INTRODUCTION

The optimisation algorithm (OA) presented in this chapter is used as the research methodology for solving the risk based reliability allocation problem detailed in the earlier chapters. The OA is yet another modified member of the class of evolutionary algorithms and resembles the approach of ‘Memetic Algorithms’ or some ‘Hybrid Genetic Algorithms’ for improving the local optimum (Chapter 4, *Section 4*). However, the processes of solution improvement as well as the selection of the genotype populations are significantly different in this methodology.

The optimisation algorithm combines the exploration abilities of genetic search with skilful exploitation of hill climbing procedures and is specific to the reliability optimisation problems considered in this research. In the context of solving these types of reliability optimisation problems, the developed algorithm (OA) introduces a different model of evolution compared to classical GA. The main features of this model are the generation of populations with unique chromosomes, working exclusively with the elite chromosomes and introducing genetic variations in the elite chromosomes using prudently designed genetic operators for ensuring rapid and efficient convergence to optimum or near optimum region of the search space. The two main reasons for implementing these notions in the optimisation algorithm are detailed below:

### 5.1.1 Epistasis Phenomenon

The effect of the combined influence of the genes in a chromosome on the visible trait of an organism is described by the ‘epistasis phenomenon’ (Chapter 3, *Section 3*). In other words, epistasis portrays the impact of one or more genes on the appearance of a particular property (or solution) which may not occur if the relevant genes are existed separately. In evolutionary computation, epistasis can be interpreted to explain the very complex and non-linear relationship between parameters. For example, the relationship between cost and reliability is non-linear and very difficult to predict. While cost is generally considered as the monotonically increasing function of reliability, there are possibilities where increased reliability does not incur higher costs (non-monotonic) and

can also appear to have a discontinuous relationship – For example, a plain and simple version of a particular component may be cheaper in cost yet it may offer greater reliability compared to a version with the same or lower reliability having many extra ‘nice to have’ features hence making it more expensive. Another example can be derived from the fiercely growing competition among retailers for dominating the respective market by means of offering greater discounts and choice of alternatives for many off-the-shelf products hence, attracting a large proportion of consumers. Thus, given the availability, a product with identical or similar specifications may be purchased at different costs from different retailers. While this may offer greater flexibility in choosing from the available varieties of the same product, the actual selection can be very challenging and cumbersome significantly for the reliability optimisation problems of large scale complex systems because of the difficult to predict overall effects of the selected product (or combination of these products) on the loss function (introduced in the preceding chapters) due to the non-linear cost-reliability relationship. An excellent survey of relationship is presented in Guikema and Pate-Cornell (2002) and Majety *et al.* (1996). A comprehensive list of articles in the field of warranty analysis by studying various cost-reliability models can be found in Pham (2007).

### **5.1.2 Extremely Large Search Space**

In the context of the optimisation problems examined in this research, the absolute combination of components appears to closely dictate the optimal reliability allocation process. For a large system with many choices of available alternatives for each component, the possibility of reaching the optimum can be very difficult due to the sheer size of the search space. Therefore, in order to converge to the optimal region of the total search space, a guiding process exploring large search areas with efficient exploitation of both feasible and infeasible solutions is pivotal. This concept is implemented in the OA by continuously loading the genotype population with unique, non-duplicating chromosomes in each iteration of the algorithm and retaining only the two best solutions from each generation (elitist selection). Working with the two best

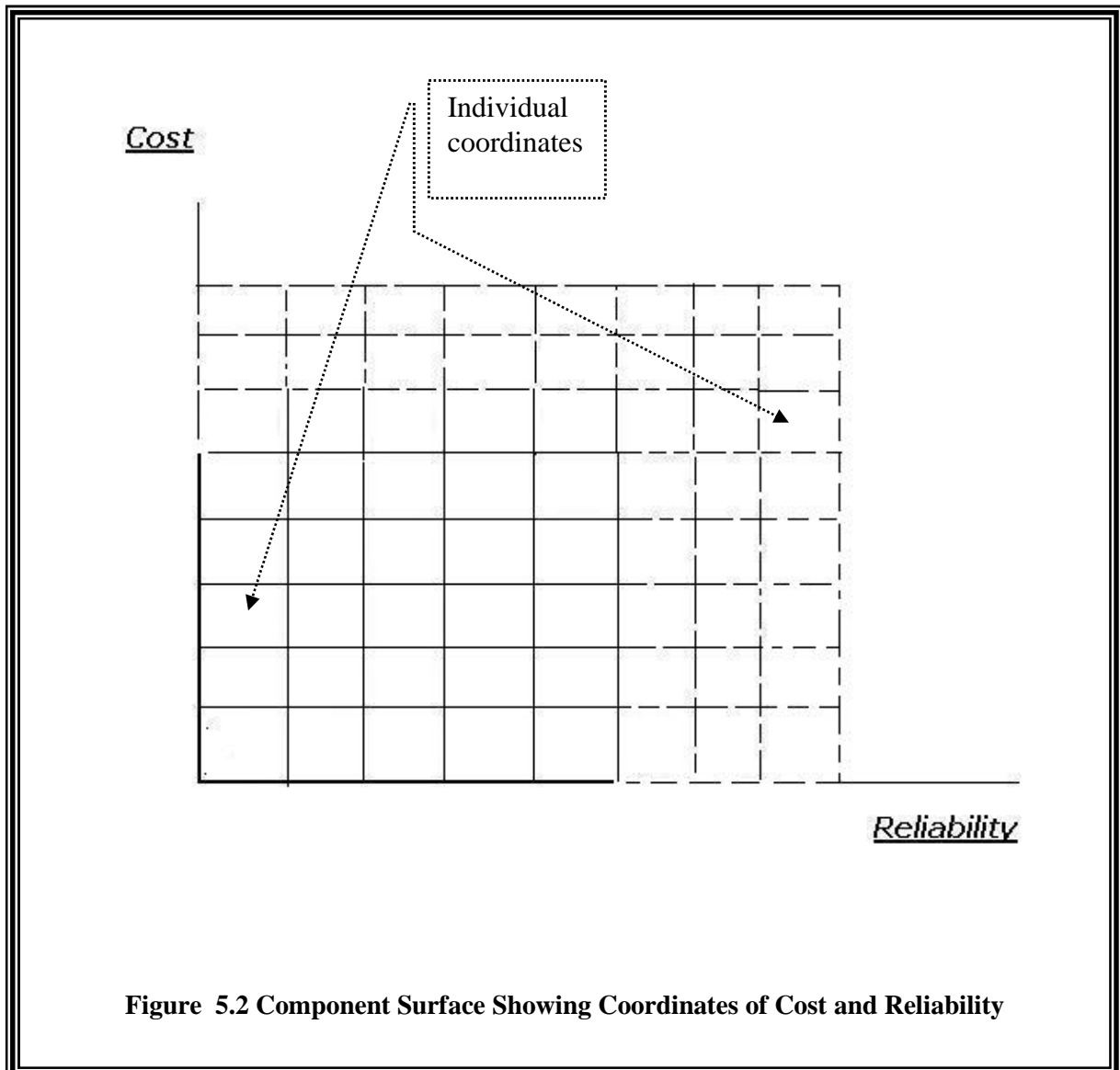
solutions along with a unique selection of non-duplicated chromosomes in each generation, introduces increased diversity in the search space. This weakens the possibilities of premature convergence and increases the performance of the computer program at the same time (because of running a smaller number of generations).

For the reasons described above, it is extremely difficult to minimise the loss function in the absence of any obvious correlation between cost and reliability particularly when cost is not considered as a monotonically increasing function of reliability; any combination of components from the infinitely large search space will have equal probability of producing the optimum solution and will subsequently require an enumerative search method for exhausting all combinations. The latter is, however, impractical for systems with a large choice of components and respective alternatives. For example, a system consisting of ' $M$ ' components with ' $N$ ' alternatives each, as represented in Fig. 5.1, there will exist ' $N^M$ ', combination of components.

$\gamma_{1,1}, C_{1,1}$	$\gamma_{1,2}, C_{1,2}$	$\gamma_{1,3}, C_{1,3}$	$\gamma_{1,4}, C_{1,4}$	.....	.....	$\gamma_{1,N}, C_{1,N}$
$\gamma_{2,1}, C_{2,1}$	$\gamma_{2,2}, C_{2,2}$	$\gamma_{2,3}, C_{2,3}$	$\gamma_{2,4}, C_{2,4}$	.....	.....	$\gamma_{2,N}, C_{2,N}$
$\gamma_{3,1}, C_{3,1}$	$\gamma_{3,2}, C_{3,2}$	$\gamma_{3,3}, C_{3,3}$	$\gamma_{3,4}, C_{3,4}$	.....	.....	$\gamma_{3,N}, C_{3,N}$
$\gamma_{4,1}, C_{4,1}$	$\gamma_{4,2}, C_{4,2}$	$\gamma_{4,3}, C_{4,3}$	$\gamma_{4,4}, C_{4,4}$	.....	.....	$\gamma_{4,N}, C_{4,N}$
.....	.....	.....	.....	.....	.....	.....
.....	.....	.....	.....	.....	.....	.....
.....	.....	.....	.....	.....	.....	.....
.....	.....	.....	.....	.....	.....	.....
.....	.....	.....	.....	.....	.....	.....
$\gamma_{M,1}, C_{M,1}$	$\gamma_{M,2}, C_{M,2}$	$\gamma_{M,3}, C_{M,3}$	$\gamma_{M,4}, C_{M,4}$	.....	.....	$\gamma_{M,N}, C_{M,N}$

**Figure 5.1 A System Consisting of M Components With N Alternatives Each**

If each row of the table in Fig 5.1, is considered as a surface defined by an individual component of the system, then the surface can be divided into many co-ordinates

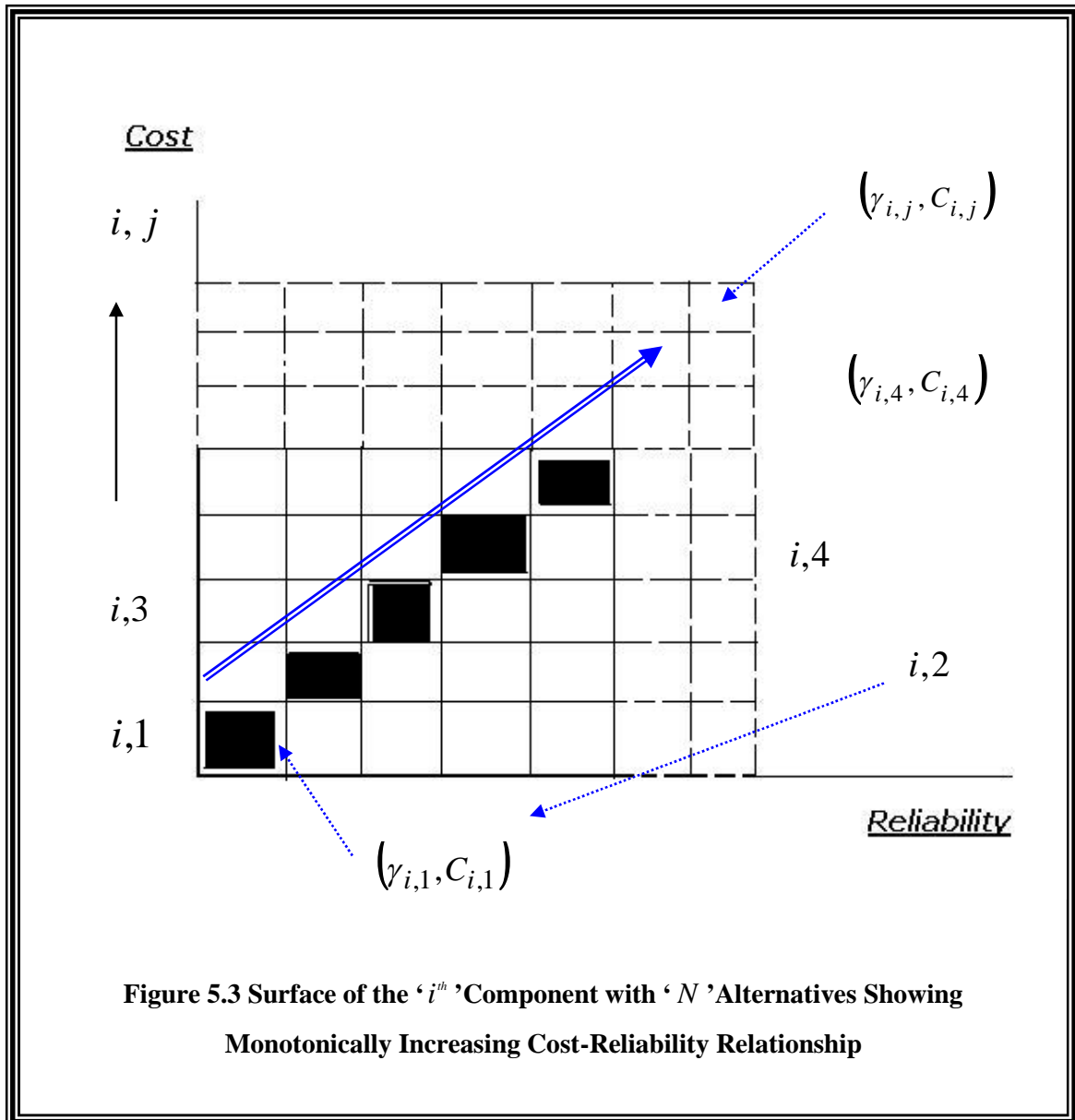


**Figure 5.2 Component Surface Showing Coordinates of Cost and Reliability**

representing individual values of cost with a corresponding value for reliability – see Fig. 5.2.

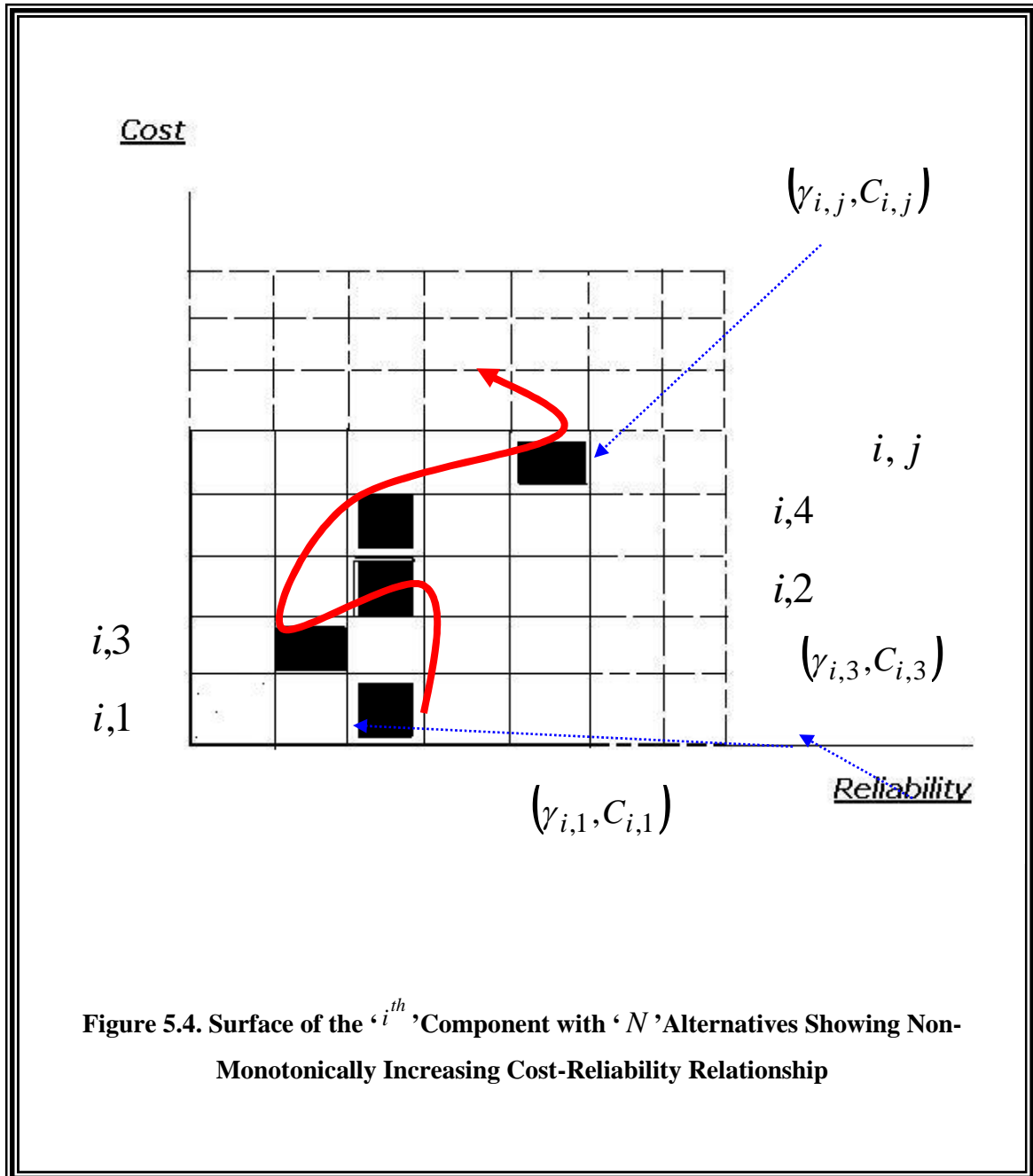
For the purpose of representing all available alternatives of a given component (i.e. ' $i^{th}$ ' row of the table from Fig. 5.1), the coordinates of the component surface can, therefore, be populated in accordance with the individual value of the cost and reliability of the ' $j^{th}$ ' alternative. If the cost of the alternatives is monotonically

increasing with the reliability, then the surface of the component can be populated using the pattern shown in Fig. 5.3.



**Figure 5.3 Surface of the ' $i^{\text{th}}$ ' Component with ' $N$ ' Alternatives Showing Monotonically Increasing Cost-Reliability Relationship**

On the other hand, if the cost-reliability relationship is not monotonically increasing among the alternatives of the component, then the plotting of the component surface will show a pattern similar to the one demonstrated in Fig 5.4.



Using the concept of a multi-coordinated surface for representing the ' $i^{th}$ ' individual component with ' $N$ ' available alternatives, the search for an optimal combination of ' $M$ ' components from the system (shown in Fig. 5.1) which defines the best (minimum) value of the loss function, can be understood as finding an optimal point in the search space comprising of ' $M$ ' coordinates where each coordinate is selected from the ' $j^{th}$ ' coordinate of the ' $i^{th}$ ' component surface. This process is depicted in Fig. 5.5



for ‘ $M$ ’ component surfaces each showing monotonically increasing cost-reliability relationship in the given choice of alternatives.

In order to effectively search the solution space particularly for a complex reliability optimisation problem involving a large number of components with an even larger number of alternatives for each component, an effective strategy for dealing with the non-linear relationship between cost and reliability is *de rigueur*. This may involve cleverly examining a large number of components combinations for feasibility without wasting too much computational effort in exploring infeasible solutions and at the same time, exploiting good solutions (e.g. local optimum) in the hope of transforming them into even better solutions. This approach is developed in the research methodology by introducing two ‘improvement procedures’ (section 5.3.5), each demonstrating the capability of efficiently testing a large number of samples with a good mixture of component combinations. The mixing process is carried out by means of skilfully structured crossover and mutation operations and using a uniform random number generator. These operations are detailed in sections 5.3.5.1.1 and 5.3.5.1.2, respectively.

For clarification, the reason for referring to the research methodology as an ‘optimisation algorithm’ or as an ‘evolutionary algorithm’ instead of ‘Hybrid GA’ or ‘Memetic Algorithm’ is because the population in the optimisation algorithm is not subject to any probabilistic alteration imposed by any of the genetic operators for mating and selection. In other words, there are no parameters such as crossover probability or mutation probability. Even the selection and maintenance of the populations, in each iteration of the algorithm, is not probabilistic as in the conventional GA, where the common approach is to use a roulette wheel strategy (see Chapter 3 & 4 for more details). Additionally, while retaining the benefits of the crossover operation (e.g. Falkenauer, 1998), extra emphasis is wielded on the mutation operation which is applied more frequently than crossover, unlike in the classical GA where it is usually applied to a very small proportion of the individuals in the population (i.e. low mutation probability).

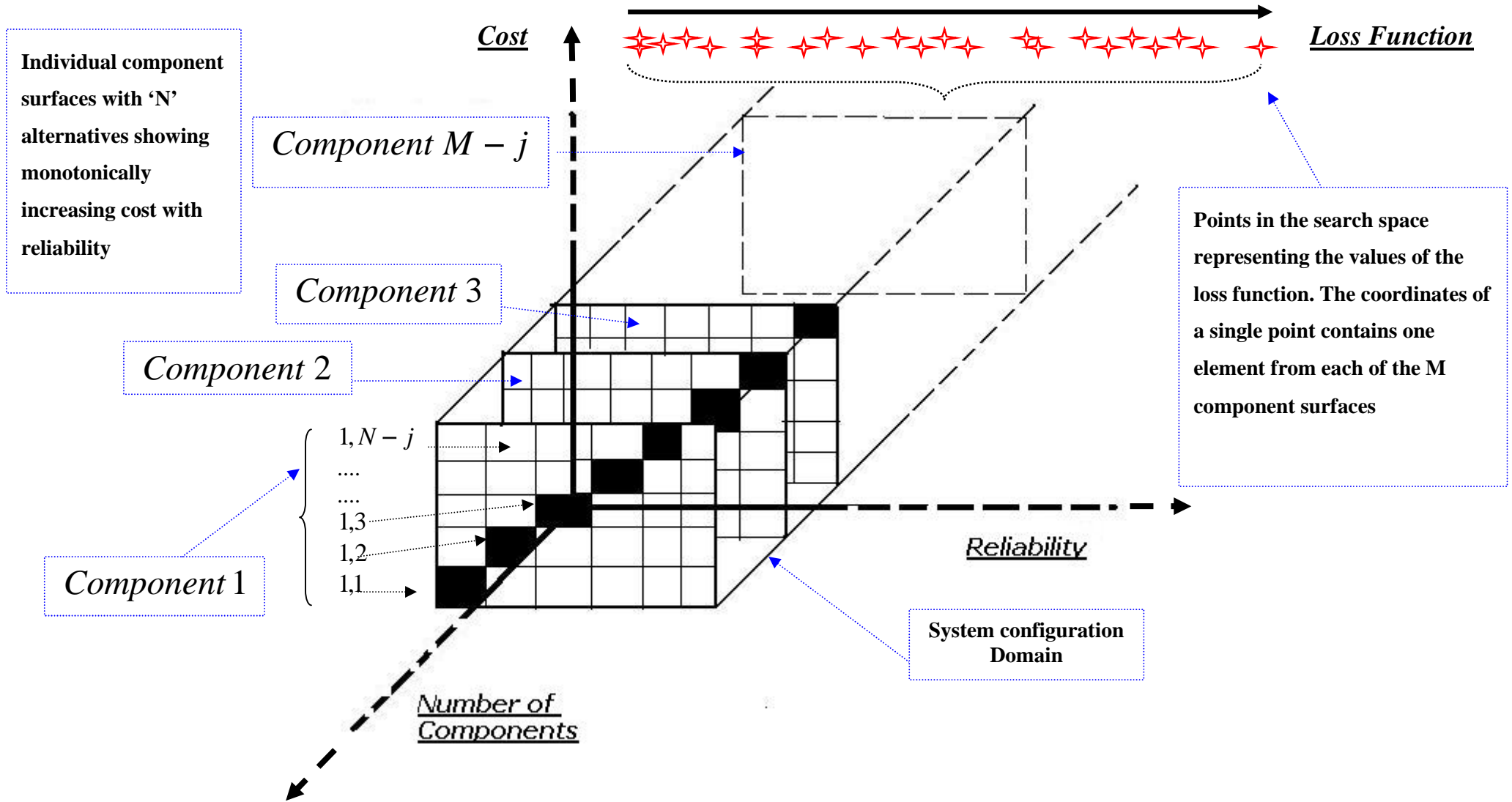
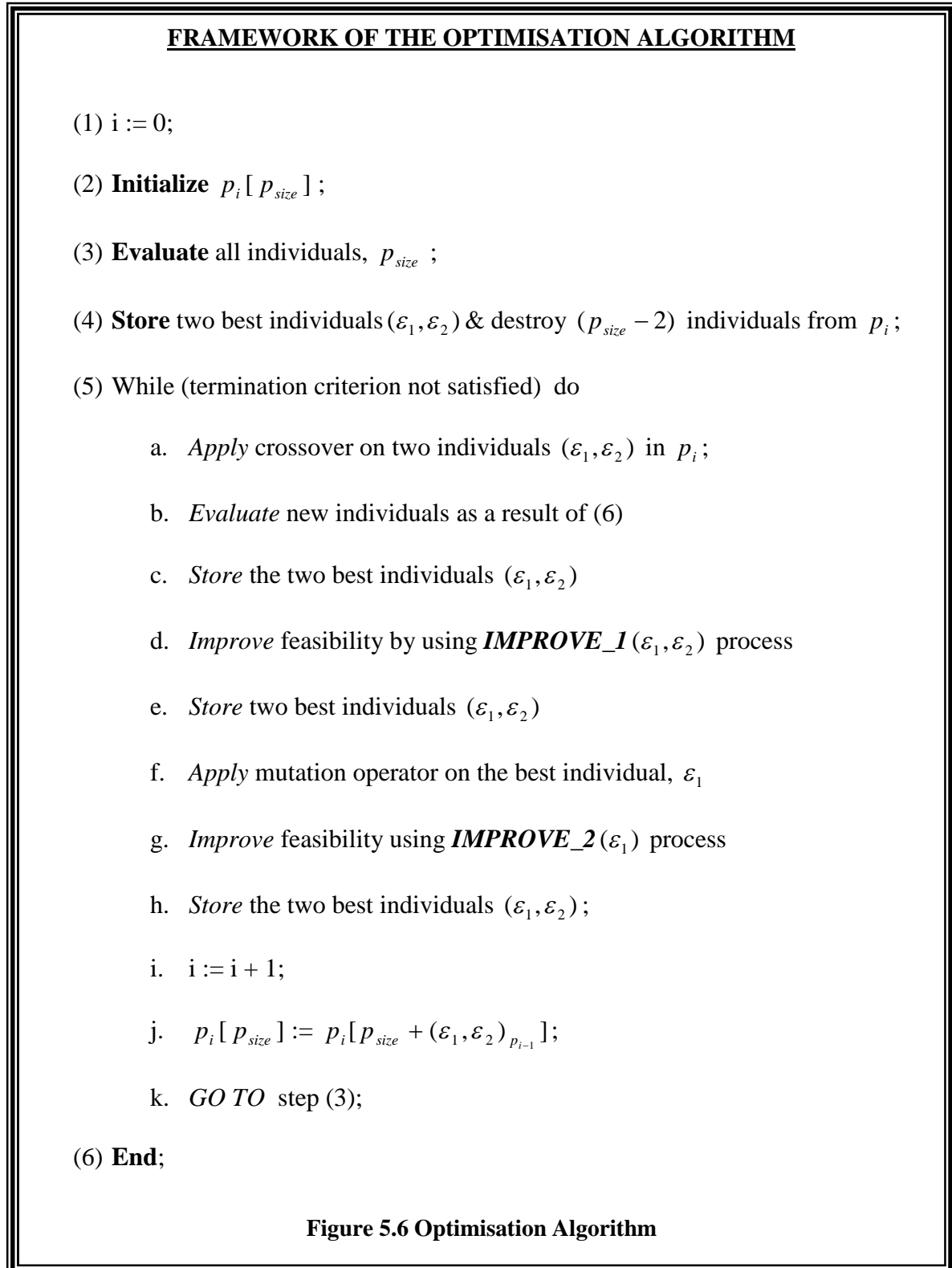


Figure 5.5 System Configuration Domain With Choice of M Components, Represented As Surfaces. Also Shown Are the Points In the Search Space Representing The Values of The Loss Function With Coordinates Located In Each of the M Component Surfaces.

## 5.2 THE OPTIMISATION ALGORITHM

The framework of the optimisation algorithm along with the improvement procedures is shown in Fig.6.6, below.



Where,

$P_k$	=	Population Number, $k = (1,2,\dots,P_{RUN})$
$P_{RUN}$	=	Total number of populations
$P_{size}$	=	Size of the population
$\varepsilon_1$	=	Optimal solution
$\varepsilon_2$	=	Near optimal solution
$C_{RUN}$	=	Total number of crossover runs
$M_{RUN}$	=	Total number of mutation runs
$IMPROVE\_1()$	=	Improvement function for crossover operation
$IMPROVE\_2()$	=	Improvement function for mutation operation

## 5.3 FEATURES OF THE OPTIMISATION

### ALGORITHM

The objective of this research is to solve complex systems consisting of ‘ $M$ ’ components with ‘ $N$ ’ alternatives each, by using the risk-based reliability allocation method, detailed in the first chapter. The alternatives have different reliabilities and costs hence resulting in difficult combinatorial optimisation problem. For a given system, the problem reduces to selecting  $M$  optimal (\*) alternatives from each row of the reliability matrix,  $(\gamma_{1,1}^*, \gamma_{2,2}^*, \dots, \gamma_{M,N}^*)$  with corresponding cost values,  $(C_{1,1}^*, C_{2,2}^*, \dots, C_{M,N}^*)$  from the cost matrix, such that the total loss function, ‘ $T_L$ ’ is minimised. The reliability and cost matrices along with the loss function ‘ $T_L$ ’ have already been defined explicitly in the earlier chapters. However, a brief overview of these is shown in the equations below, since this will be used in explaining the key concepts of the proposed optimisation algorithms (OA) such as chromosome encoding, crossover operation and mutation operation.

$$\gamma = \begin{bmatrix} \gamma_{11}, \gamma_{12}, \gamma_{13} \dots \dots \dots, \gamma_{1N} \\ \gamma_{21}, \gamma_{22}, \gamma_{23} \dots \dots \dots, \gamma_{2N} \\ \gamma_{31}, \gamma_{32}, \gamma_{33} \dots \dots \dots, \gamma_{3N} \\ \dots \dots \dots \\ \dots \dots \dots \\ \gamma_{M1}, \gamma_{M2}, \gamma_{M3} \dots \dots \dots, \gamma_{MN} \end{bmatrix} \quad (5.1)$$

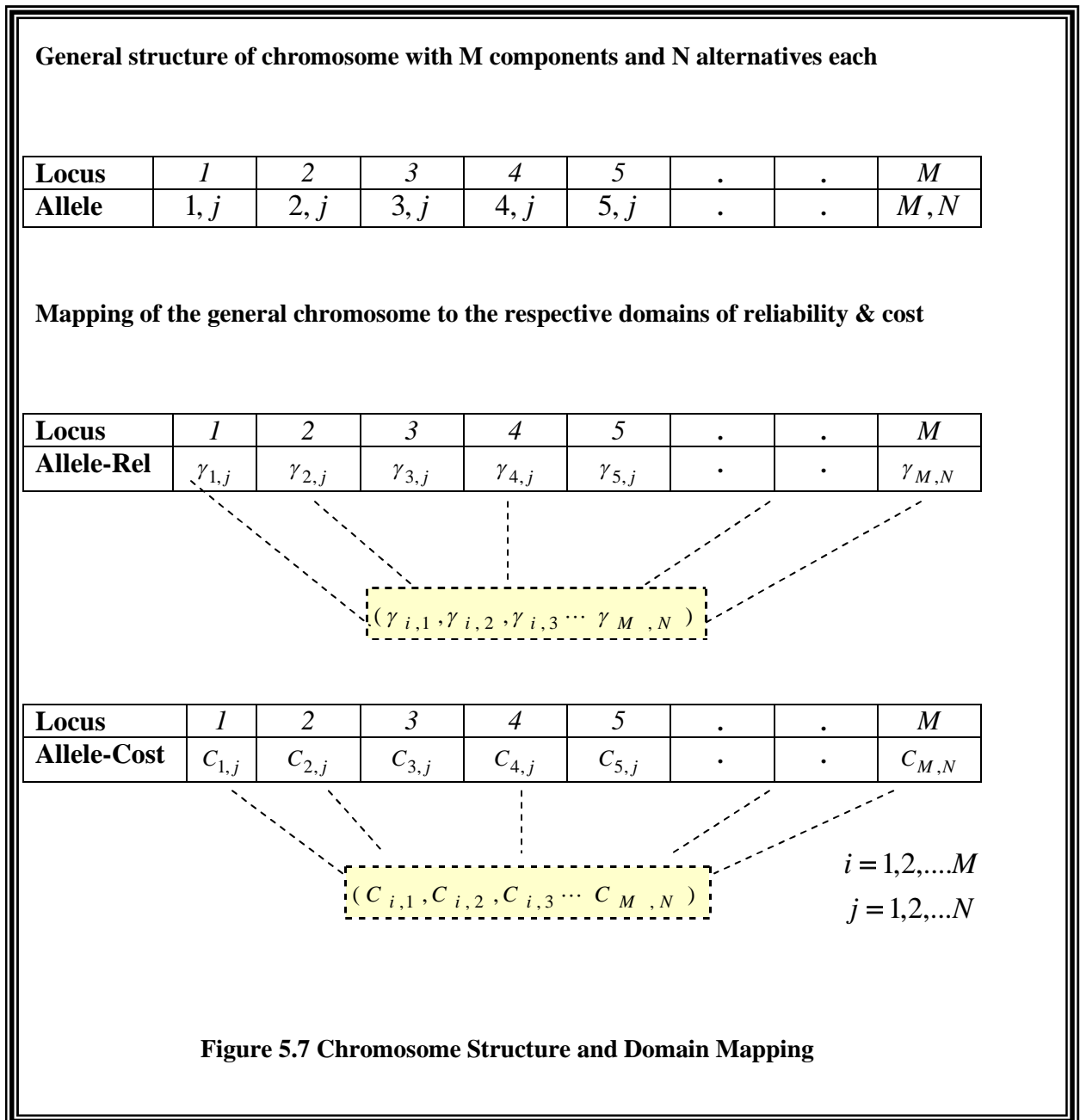
$$C = \begin{bmatrix} C_{11}, C_{12}, C_{13} \dots \dots \dots, C_{1N} \\ C_{21}, C_{22}, C_{23} \dots \dots \dots, C_{2N} \\ C_{31}, C_{32}, C_{33} \dots \dots \dots, C_{3N} \\ \dots \dots \dots \\ \dots \dots \dots \\ C_{M1}, C_{M2}, C_{M3} \dots \dots \dots, C_{MN} \end{bmatrix} \quad (5.2)$$

$$T_L = \text{Min} \left( \sum_i^M c_i + \left[ 1 - R_s(\gamma_{1,1}^*, \gamma_{2,2}^*, \dots, \gamma_{M,N}^*) \right] \times \bar{C} \right) \quad (5.3)$$

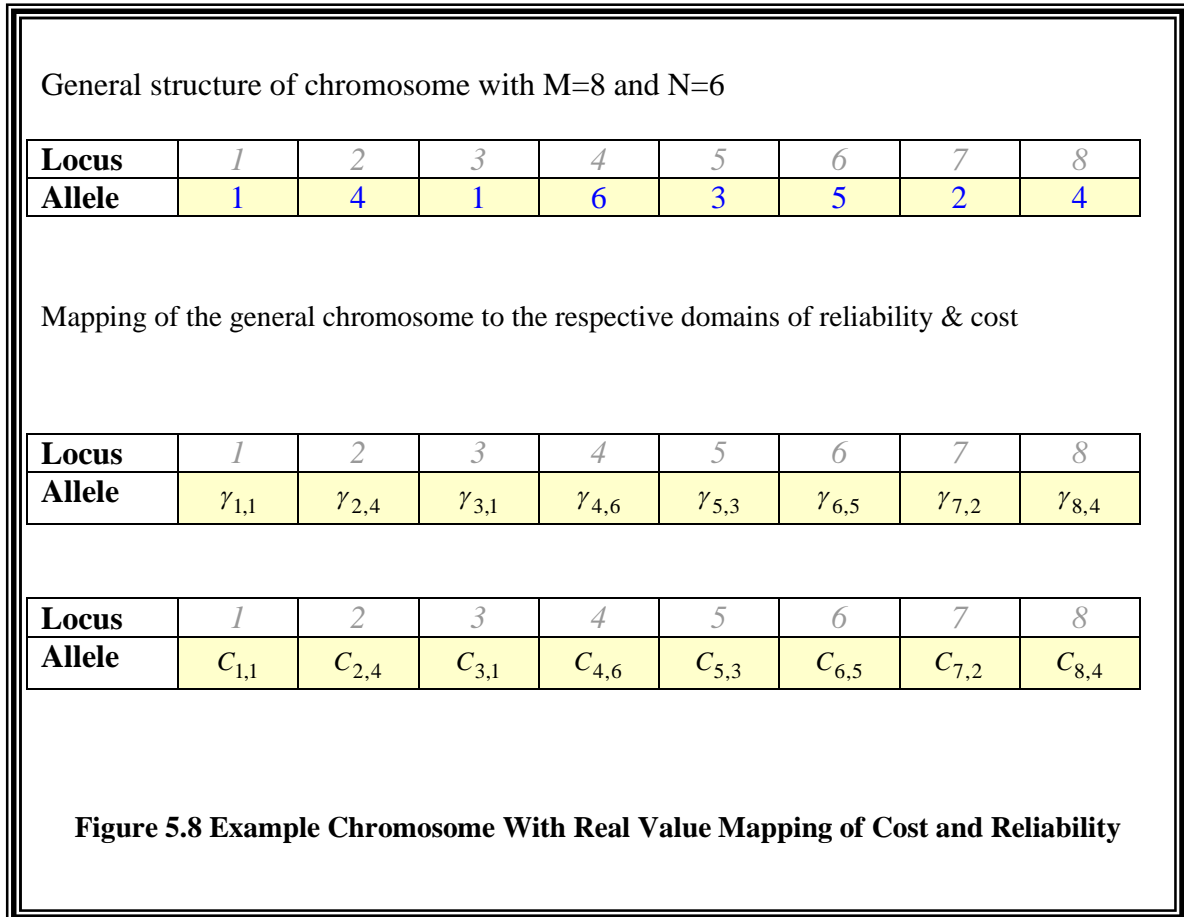
### 5.3.1 Structure of the Chromosome

Traditionally, chromosomes have been coded as binary strings (Goldberg, 1989) but for combinatorial optimisation problems, an encoding using integer values can be more efficient (Holland, 1975; Spall 2003). The optimisation algorithm therefore uses the same approach of representing the chromosomes with a real number encoding method as introduced in section 3 of the previous chapter. The general structure of the chromosome in this algorithm therefore consists of a string containing ‘ $M$ ’ loci for genes (equal to the total number of components) with ‘ $N$ ’ alleles representing the integer value up to the available number of alternatives, for corresponding genes (component). The value (allele) of each component (gene) in the encoding mechanism is filled randomly by using a uniformly distributed random numbers generator and in

accordance with the parameters ‘ $M$ ’ and ‘ $N$ ’. The general structure of the chromosome can be further explained in terms of the real number values of the randomly selected alternatives by mapping the alleles to the domain of reliability and cost, depicted in equation 5.1 and equation 5.2. This process is shown in the Fig. 5.7, where two subsets of the general chromosomes are decoded in terms of the corresponding values of reliability and associated cost from the encoded genotype.



Using the general structure of the chromosome defined above, an example system consisting of eight components with six alternatives is shown in Fig.5.8.



Each position in the chromosome string above represents the selected alternative of the corresponding component. Thus, at locus one, the allele of the gene is one which represents the first alternative of the first component. Similarly, at locus two, the gene represents the fourth alternative for the second component and so on.

### 5.3.2 Population Structure

The initial population, ' $p_0$ ' is constructed randomly with unique ' $p_{size}$ ' number of genotypes (chromosomes) selected for genetic search. Although the chromosomes are selected randomly, it is ensured that only unique chromosomes are part of the

population and no duplicates are chosen. This introduces greater diversity in the search mechanism of the algorithm by sampling more potential solutions. This feature of the OA also eliminates the possibilities of examining the same individual chromosome multiple numbers of times, which saves the extra computational efforts of the algorithm. Once the population is formed, each member of the population is evaluated for the feasible solution of the phenotype using the objective function (for example, equation 5.3). Only the two best (elite) chromosomes,  $(\varepsilon_1 \& \varepsilon_2)$  with minimum values of the loss function, ' $T_c$ ', are stored from the whole population. These two elite chromosomes are ordered according to the lowest value of the loss cost function (if being minimised) such that  $(\varepsilon_1 < \varepsilon_2)$ . This method of selection is similar to tournament selection sampling approach (Goldberg *et al.*, 1989), discussed in chapter four.

The two elite chromosomes are automatically selected as the parents for undergoing the genetic operations (explained in the next sections). These chromosomes also represent the local optimum (Chapter 3) and local sub-optimum solutions, respectively. This is a good enough reason for justifying their selection as the parents for breeding in the next population and allowing them to contribute their genetic information with the hope of producing progeny with an even better fitness value of the objective function. If no better solutions are found after these genetic operations and termination criteria is not reached, the same pair of the elite chromosomes is injected in the next population for competing with the new group of potential solutions. On the other hand, if better versions of these parents are formed as a result of the genetic operations, the offspring instantly replace the parents with the view of the respective fitness level and enter the next population for competing once again. For example, if a new chromosome is not as good as the first parent, ' $\varepsilon_1$ ' but better than the second parent ' $\varepsilon_2$ ', it instantly replaces the second parent.

### 5.3.3 Crossover Operation

The essence of the crossover operation is similar to what has previously been explained in the preceding chapters. However, in this optimisation algorithm, the format of this



operation is considerably different. Detailing this, there are three different stages in which the crossover operation is applied on the parent individuals. Each of these stages is explained below:

### 5.3.3.1 First Stage Crossover Operation (FSCO)

The first stage crossover operation proceeds by initially selecting a crossover site (locus of the gene) using a uniform random number generator between '1' and ' $M$ ', where  $M$  is the length of the chromosome string, representing the total number of genes (components) in the solution string. The same crossover site is used for both parents and the FSCO continues by swapping the alleles of the genes (components) from the two parent chromosome, at the randomly selected crossover site. This process produces two new offspring and the genes from the parent chromosomes are inherited in the new chromosomes. The structure of the chromosome along with the domain of its reliability and cost values used in operation is already shown in Fig. 5.7 above. The first stage crossover process is demonstrated in Fig.5.9 where the genes of the two best chromosomes ( $\varepsilon_1$  &  $\varepsilon_2$ ), also in different colour coding, are exchanged at a random crossover site, using an example reliability system with eight components ( $M = 8$ ) with six alternatives each ( $N = 6$ ), see Fig. 5.8. The position of the locus selected in the FSCO example, shows the crossover site to be the fifth gene or the fifth component of the reliability system in consideration (Fig. 5.9). After the completion of the first stage crossover operation, the figure details the two offspring showing the structure of the inherited genes from the two parents; the colour coding scheme embellishes the contributions of the parent genes. The alleles exchanged at the random crossover point, ' $\hat{X}$ ' are represented by the symbols, ' $\hat{X}_{\gamma_{i,j}}^{\varepsilon_k}$ ' and ' $\hat{X}_{C_{i,j}}^{\varepsilon_k}$ ' for the respective reliability and cost values from the ' $k^{th}$ ' parent, with ' $i^{th}$ ' component (gene) and ' $j^{th}$ ' alternative (allele), where ' $k = \{1,2\}$ ', depending on either of the two parents.

The progeny resulting from the first stage crossover operation is evaluated for the level of their fitness with respect to the objective function. If the fitness of either or both of the new chromosomes is better than the two parents, the two elite chromosomes ' $(\varepsilon_1, \varepsilon_2)$ ' are updated accordingly and consequently, advance to the second stage of the

crossover operation. All other individuals are discarded. The crossover only swaps alleles of the selected gene from the two parents instead of exchanging the values of all genes either sides of the crossover point as seen in the conventional evolutionary algorithms (e.g. GA). Selecting only one gene at a time for crossover reduces the risk of loosing the good solution given the complex, nonlinear relationship between cost & reliability and the high epistasis, which is generally found among the respective genes. The risk would have been higher if all alleles either sides of the crossover point were swapped simultaneously.



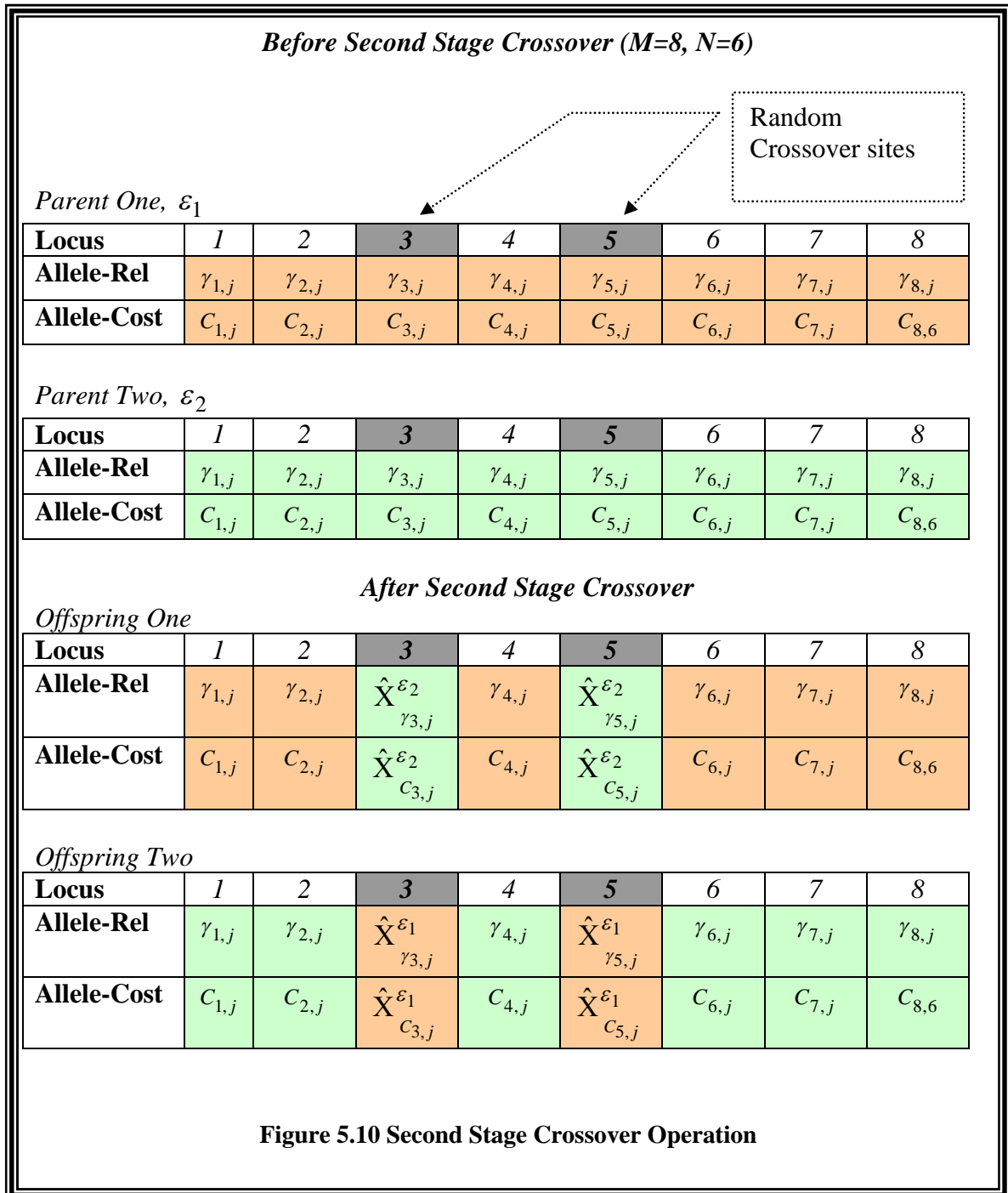
### 5.3.3.2 Second Stage Crossover Operation (SSCO)

Similar to the FSCO, the second stage crossover operation carries on by selecting two crossover sites (loci of the genes) using a uniform random number generator between '1' and ' $M$ ', where  $M$  is the length of the chromosome string, representing the total number of genes (components) in the solution string. The two crossover sites are identical for both parents and the SSCO continues by swapping the alleles of the genes (components) from the two parent chromosome at each of the two crossover sites. This process produces two new offspring and the genes from the parent chromosomes are inherited in the new chromosomes as shown in Fig.5.10. The figure shows the exchange of the genes between the two best chromosomes ( $\varepsilon_1$  &  $\varepsilon_2$ ) at two random crossover sites by using the same example reliability system with eight components ( $M = 8$ ) & six alternatives each ( $N = 6$ ) as in FSCO.

The positions of the locus selected in the figure 5.10, shows the crossover sites to be at the third and the fifth genes. After the completion of the second stage crossover operation, the figure details the two offspring showing the structure of the inherited genes from the two parents. The alleles exchanged at each random crossover point, ' $\hat{X}$ ', are represented by the symbols, ' $\hat{X}_{\gamma_{i,j}}^{\varepsilon_k}$ ', and ' $\hat{X}_{C_{i,j}}^{\varepsilon_k}$ ', for the respective reliability and cost values from the ' $k^{th}$ ' parent, with ' $i^{th}$ ' component (gene) and ' $j^{th}$ ' alternative (allele), where ' $k = \{1,2\}$ ', depending on either of the two parents. The progeny resulting from this operation is evaluated for the level of their fitness with respect to the given objective function. If the fitness of either or both of the new chromosomes is better than the two parents, the two elite chromosomes ' $(\varepsilon_1, \varepsilon_2)$ ' are updated accordingly and progress to the third stage of the crossover operation. All other individuals are discarded.

The main purpose of the two stage crossover operation is to gradually increase the complexity of the FSCO by swapping only two alleles of the selected genes from the two parents instead of exchanging the values of all genes either side of the crossover points as seen in the conventional evolutionary algorithms. Selecting two genes at a time in the crossover process carefully introduces the variations in the elite solutions

while lowering the risk of loosing the good solution because of the nonlinear relationship between cost & reliability and the high epistasis. Understandably, the risk will be higher if all alleles either side of the crossover points are swapped concurrently.



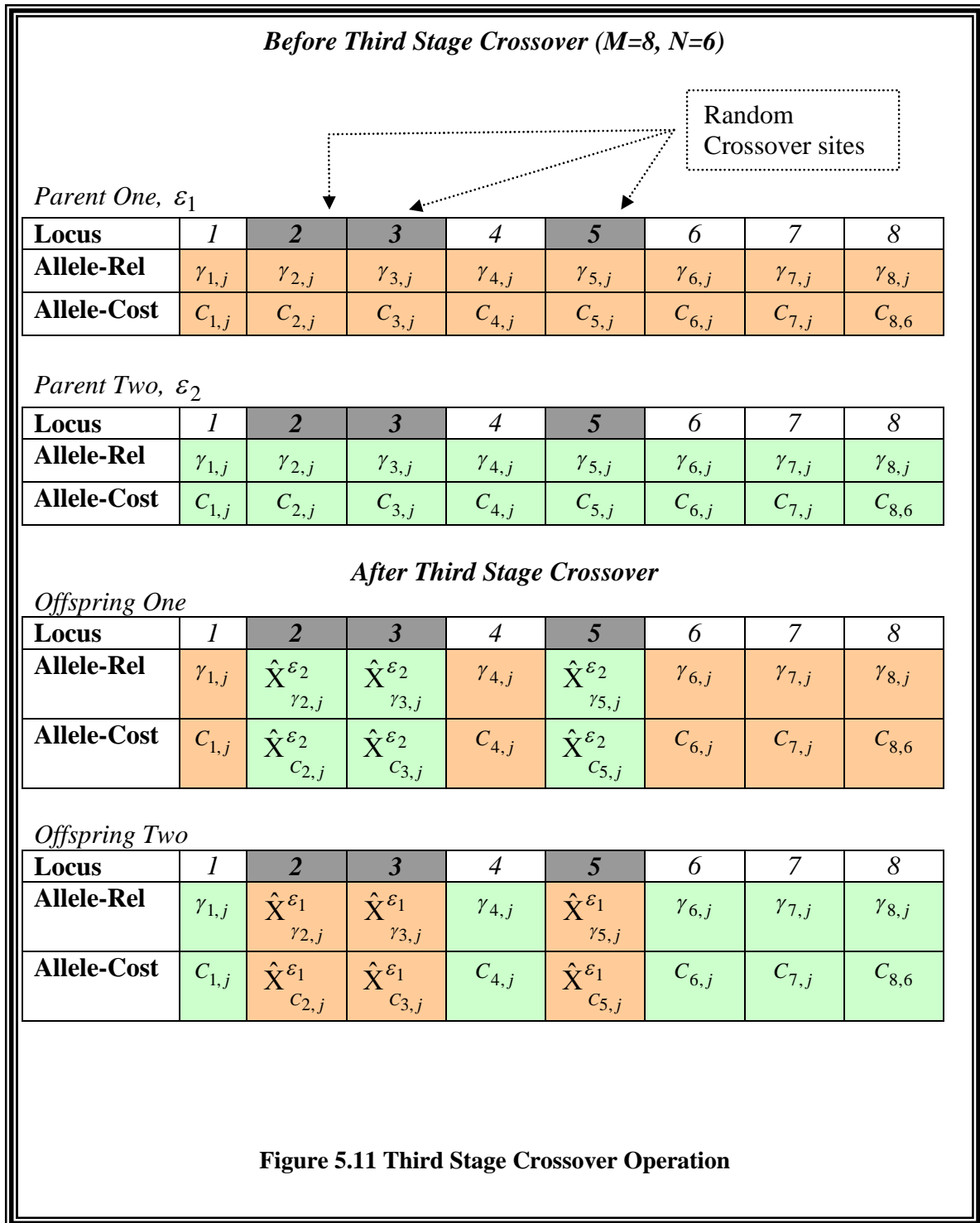
### 5.3.3.3 Third Stage Crossover Operation (TSCO)

The TSCO is the final stage of the crossover operation in which the crossover operation selects three random crossover sites by using a uniform random number generator between '1' and ' $M$ ', where  $M$  is the length of the chromosome string, representing the total number of genes (components) in the solution string. The three crossover sites are common for both parents and the TSCO progresses by swapping the alleles of the genes (components) from the two parent chromosome at each of the three crossover sites. This process produces two new offspring and the genes from the parent chromosomes are inherited in the new chromosomes as shown in Fig.5.11. The figure shows the exchange of the genes between the two best chromosomes ( $\varepsilon_1$  &  $\varepsilon_2$ ) at three random crossover sites by using the same example reliability system with eight components ( $M = 8$ ) & six alternatives each ( $N = 6$ ) as in the previous section.

The random positions selected in the figure shows the crossover sites to be at the second, third and the fifth genes of the chromosome string. After the end of the third stage crossover operation, the figure also shows the two offspring displaying the structure of the inherited genes from the two parents. The alleles exchanged at each random crossover point, ' $\hat{X}$ ', are represented by the symbols, ' $\hat{X}_{\gamma_{i,j}}^{\varepsilon_k}$ ', and ' $\hat{X}_{C_{i,j}}^{\varepsilon_k}$ ', for the respective reliability and cost values from the ' $k^{th}$ ', parent, with ' $i^{th}$ ', component (gene) and ' $j^{th}$ ', alternative (allele), where ' $k = \{1,2\}$ ', depending on either of the two parents. The progeny resulting from this operation is evaluated for the level of their fitness with respect to the given objective function. If the fitness of either or both of the new chromosomes is better than the two parents, the two elite chromosomes ' $(\varepsilon_1, \varepsilon_2)$ ' are updated accordingly. The best of the two elite chromosomes, ' $\varepsilon_1$ ', is then selected to undergo mutation operation while ' $\varepsilon_2$ ' is stored and updated accordingly. All other individuals are discarded.

The objective of the three stage crossover operation is to continue with the steady increase in the complexity of the SSCO by exchanging only three alleles of the selected genes from the two parents. This way, the two elite solutions (parents) are prudently

examined without disrupting the combined effect of the genes (epistasis) which constitute the good solution as a whole. Of course, achieving this objective will be very difficult if the entire subset of the chromosome string from either side of the crossover points is exchanged, causing bigger disruption in the gene structure.



### 5.3.3.4 Numerical Example

The crossover stages defined above can be shown by a numerical example. For a reliability system consisting of eight components ( $M = 8$ ) & six alternatives each ( $N = 6$ ), as in the previous section, let's assume the cost and reliability data as defined in equation (5.4) and equation (5.5).

$$\gamma = \begin{bmatrix} 0.50 & 0.60 & 0.70 & 0.80 & 0.90 & 0.95 \\ 0.50 & 0.6 & 0.75 & 0.80 & 0.90 & 0.95 \\ 0.50 & 0.65 & 0.75 & 0.80 & 0.90 & 0.95 \\ 0.50 & 0.60 & 0.70 & 0.85 & 0.90 & 0.95 \\ 0.55 & 0.65 & 0.70 & 0.85 & 0.90 & 0.95 \\ 0.55 & 0.60 & 0.75 & 0.80 & 0.90 & 0.95 \\ 0.55 & 0.60 & 0.75 & 0.80 & 0.90 & 0.95 \\ 0.50 & 0.65 & 0.70 & 0.80 & 0.90 & 0.95 \end{bmatrix} \quad (5.4)$$

$$C = \begin{bmatrix} 100 & 200 & 300 & 400 & 700 & 1000 \\ 100 & 250 & 300 & 400 & 650 & 1100 \\ 110 & 205 & 310 & 405 & 800 & 950 \\ 100 & 200 & 300 & 400 & 750 & 850 \\ 105 & 210 & 310 & 385 & 650 & 775 \\ 100 & 215 & 305 & 400 & 690 & 1000 \\ 115 & 100 & 300 & 300 & 750 & 1100 \\ 100 & 200 & 270 & 300 & 890 & 1115 \end{bmatrix} \quad (5.5)$$

Using the chromosome structure as depicted in Fig. 5.3, let's imagine the two best chromosomes ' $(\varepsilon_1, \varepsilon_2)$ ' are:

Chromosome ' $\varepsilon_1$ ':

Locus	1	2	3	4	5	6	7	8
Allele	1	4	1	6	3	5	2	4

Chromosome ' $\varepsilon_2$ ':

Locus	1	2	3	4	5	6	7	8
Allele	2	6	3	2	4	1	5	6

The mapping of the two best chromosomes to the respective domains of reliability & cost can be carried out by using equations (5.4) and (5.5).

Chromosome ' $\varepsilon_1$ ':

Locus	1	2	3	4	5	6	7	8
Allele-Rel	0.5	0.8	0.5	0.95	0.7	0.9	0.6	0.8
Allele-Cost	100	400	110	850	310	690	100	300

Chromosome ' $\varepsilon_2$ ':

Locus	1	2	3	4	5	6	7	8
Allele-Rel	0.6	0.95	0.75	0.6	0.85	0.55	0.9	0.95
Allele-Cost	200	1100	310	200	385	100	750	1115

Having ascertained the two best chromosomes, they can be selected as parents for undergoing all three stages of the crossover operations. The steps of these processes are shown below:

### First Stage Crossover Operation

If random crossover site = 3, then,

Chromosome ' $\varepsilon_1$ ':

Locus	1	2	3	4	5	6	7	8
Allele-Rel	0.5	0.8	0.5	0.95	0.7	0.9	0.6	0.8
Allele-Cost	100	400	110	850	310	690	100	300

Chromosome ' $\varepsilon_2$ ':

Locus	1	2	3	4	5	6	7	8
Allele-Rel	0.6	0.95	0.75	0.6	0.85	0.55	0.9	0.95
Allele-Cost	200	1100	310	200	385	100	750	1115



The new offspring chromosomes produced in the first stage mutation operation will be,

Chromosome ' $\varepsilon_1$ ':

Locus	1	2	3	4	5	6	7	8
Allele-Rel	0.5	0.8	0.75	0.95	0.7	0.9	0.6	0.8
Allele-Cost	100	400	310	850	310	690	100	300

Chromosome ' $\varepsilon_2$ ':

Locus	1	2	3	4	5	6	7	8
Allele-Rel	0.6	0.95	0.5	0.6	0.85	0.55	0.9	0.95
Allele-Cost	200	1100	110	200	385	100	750	1115

### Second Stage Crossover Operation

If random crossover sites = 2 and 6, then,

Chromosome ' $\varepsilon_1$ ':

Locus	1	2	3	4	5	6	7	8
Allele-Rel	0.5	0.8	0.5	0.95	0.7	0.9	0.6	0.8
Allele-Cost	100	400	110	850	310	690	100	300

Chromosome ' $\varepsilon_2$ ':

Locus	1	2	3	4	5	6	7	8
Allele-Rel	0.6	0.95	0.75	0.6	0.85	0.55	0.9	0.95
Allele-Cost	200	1100	310	200	385	100	750	1115

The new offspring chromosomes produced in the first stage mutation operation will be,

Chromosome ' $\varepsilon_1$ ':

Locus	1	2	3	4	5	6	7	8
Allele-Rel	0.5	0.95	0.5	0.95	0.7	0.55	0.6	0.8
Allele-Cost	100	1100	110	850	310	100	100	300

Chromosome ' $\varepsilon_2$ ':

Locus	1	2	3	4	5	6	7	8
Allele-Rel	0.6	0.8	0.75	0.6	0.85	0.9	0.9	0.95
Allele-Cost	200	400	310	200	385	690	750	1115

**Third Stage Crossover Operation**

If random crossover sites = 4, 7 and 8, then,

Chromosome ' $\varepsilon_1$ ':

Locus	1	2	3	4	5	6	7	8
Allele-Rel	0.5	0.8	0.5	0.95	0.7	0.9	0.6	0.8
Allele-Cost	100	400	110	850	310	690	100	300

Chromosome ' $\varepsilon_2$ ':

Locus	1	2	3	4	5	6	7	8
Allele-Rel	0.6	0.95	0.75	0.6	0.85	0.55	0.9	0.95
Allele-Cost	200	1100	310	200	385	100	750	1115

The new offspring chromosomes produced in the first stage mutation operation will be,

Chromosome ' $\varepsilon_1$ ':

Locus	1	2	3	4	5	6	7	8
Allele-Rel	0.5	0.8	0.5	0.6	0.7	0.9	0.9	0.95
Allele-Cost	100	400	110	200	310	690	750	1115

Chromosome ' $\varepsilon_2$ ':

Locus	1	2	3	4	5	6	7	8
Allele-Rel	0.6	0.95	0.75	0.95	0.85	0.55	0.6	0.8
Allele-Cost	200	1100	310	850	385	100	100	300

### 5.3.4 Mutation Operation

The best found local optimum solution ' $\varepsilon_1$ ' from the previous step of the algorithm is selected exclusively to undergo the mutation operation. The sub-optimum solution, ' $\varepsilon_2$ ', on the other hand, is only updated if the new variations of the ' $\varepsilon_1$ ' are better than the current values of the ' $\varepsilon_2$ '. Like crossover operation, the fundamental nature of the mutation operation is similar to what has previously been explained in the context of evolutionary algorithms. However, in this optimisation algorithm, the configuration of this operation is considerably different. Detailing this, there are three different stages in which the mutation operation is applied on the best found solution, ' $\varepsilon_1$ ' such that the number of genes undergoing the mutation process are increased gradually in each stage. This process is similar to the crossover process and indeed employed for the same reasons as explained previously. All three stages of mutation are described below:

#### 5.3.4.1 First Stage Mutation Operation (FSMO)

The first stage mutation operation is carried out by initially selecting a mutation site (locus of the gene) by means of a uniform random number generator between '1' and ' $M$ ', where  $M$  is the length of the chromosome string, representing the total number of genes (components) in the solution string. Having established the gene (component)

location in the chromosome string, the allele is varied by replacing the current value of this gene with the ' $j^{th}$ ' alternative from the given choice of ' $N$ ' alternatives for the selected component. The selection of the ' $j^{th}$ ' alternative is performed by using a uniform random number generator between '1' and ' $N$ ', where ' $N$ ' is the total number of available alternatives for a particular component, ' $i$ ' as defined in equations 5.1 and 5.2 above.

The first stage mutation operation along with the structure of the chromosome and the domain of its reliability and cost values are demonstrated in Fig.5.12. As can be seen in the figure, the gene value of the best chromosomes ' $\varepsilon_1$ ', is altered at a randomly selected mutation site, using an example reliability system with eight components ( $M = 8$ ) with six alternatives each ( $N = 6$ ), see Fig. 5.8. The position of the locus selected in the FSMO example, shows the mutation site to be the fifth gene or the fifth component of the reliability system in consideration. After the completion of the first stage mutation operation, the figure reveals the new offspring with details of the inherited genes from its parent chromosome.

In general, the alleles which are exchanged at the random mutation point, ' $\widehat{M}$ ' are represented by the symbols, ' $\widehat{M}_{\gamma_{i,j}}^{\varepsilon_k}$ ' and ' $\widehat{M}_{C_{i,j}}^{\varepsilon_k}$ ' for the respective reliability and cost values from the ' $k^{th}$ ' parent, with ' $i^{th}$ ' component (gene) and ' $j^{th}$ ' alternative (allele), where ' $k = 1$ ', since only the best of the two elite solutions is selected to endure the mutation operation..

The mutation operation is applied in the hope of creating a new offspring with better fitness than the current best solutions, ' $(\varepsilon_1, \varepsilon_2)$ ' with particular emphasis on ' $\varepsilon_1$ '. If the new chromosome is better in fitness than either ' $\varepsilon_1$ ' or ' $\varepsilon_2$ ', it instantly replaces them in accordance with the respective fitness levels. Otherwise, it is discarded for being a poor solution. At the end of the first stage mutation operation, the current best solution ' $\varepsilon_1$ ' is deemed as the local optimum and considered for the second stage of the mutation operation; details of this process is explained in the next section.

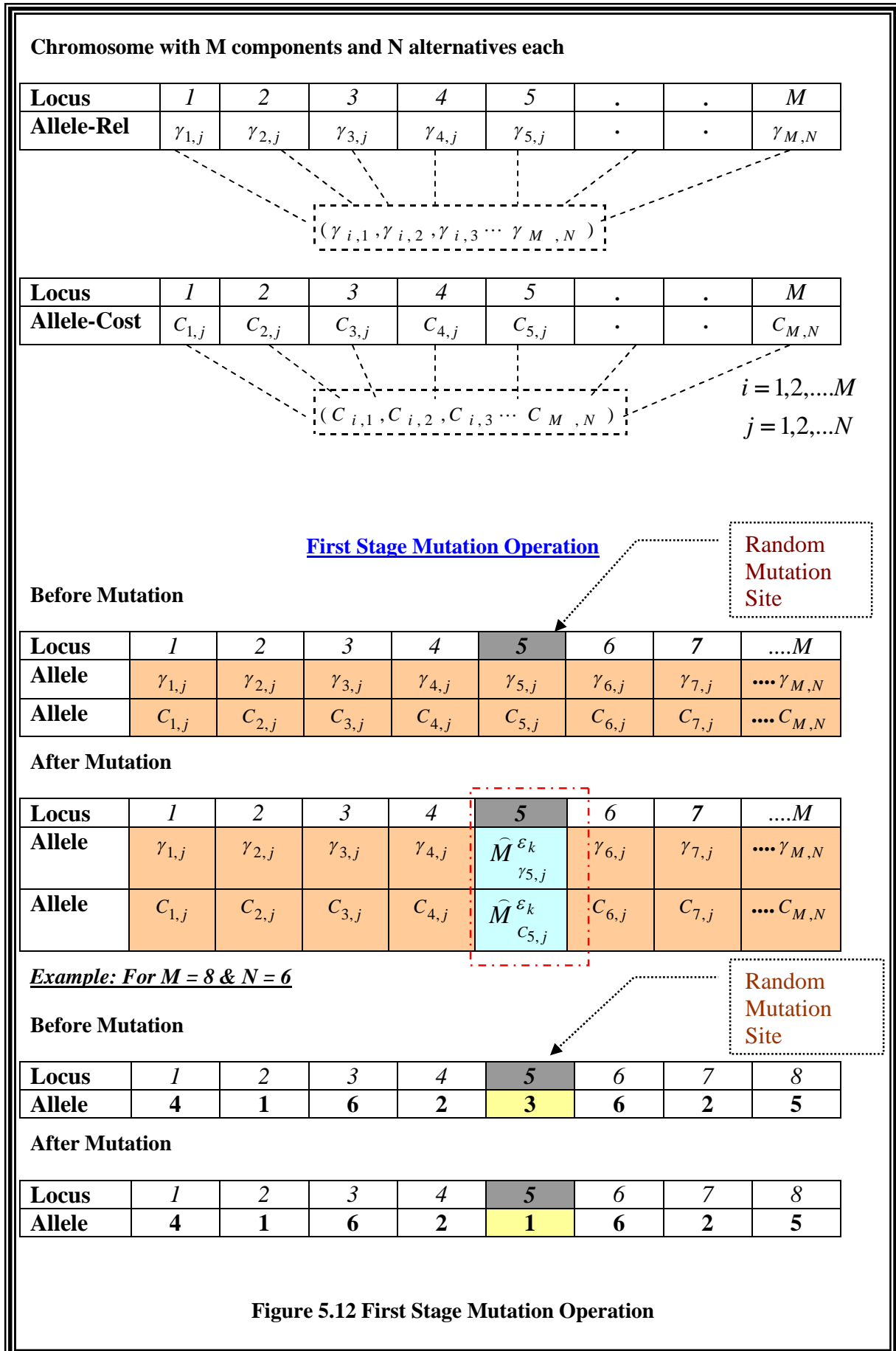


Figure 5.12 First Stage Mutation Operation

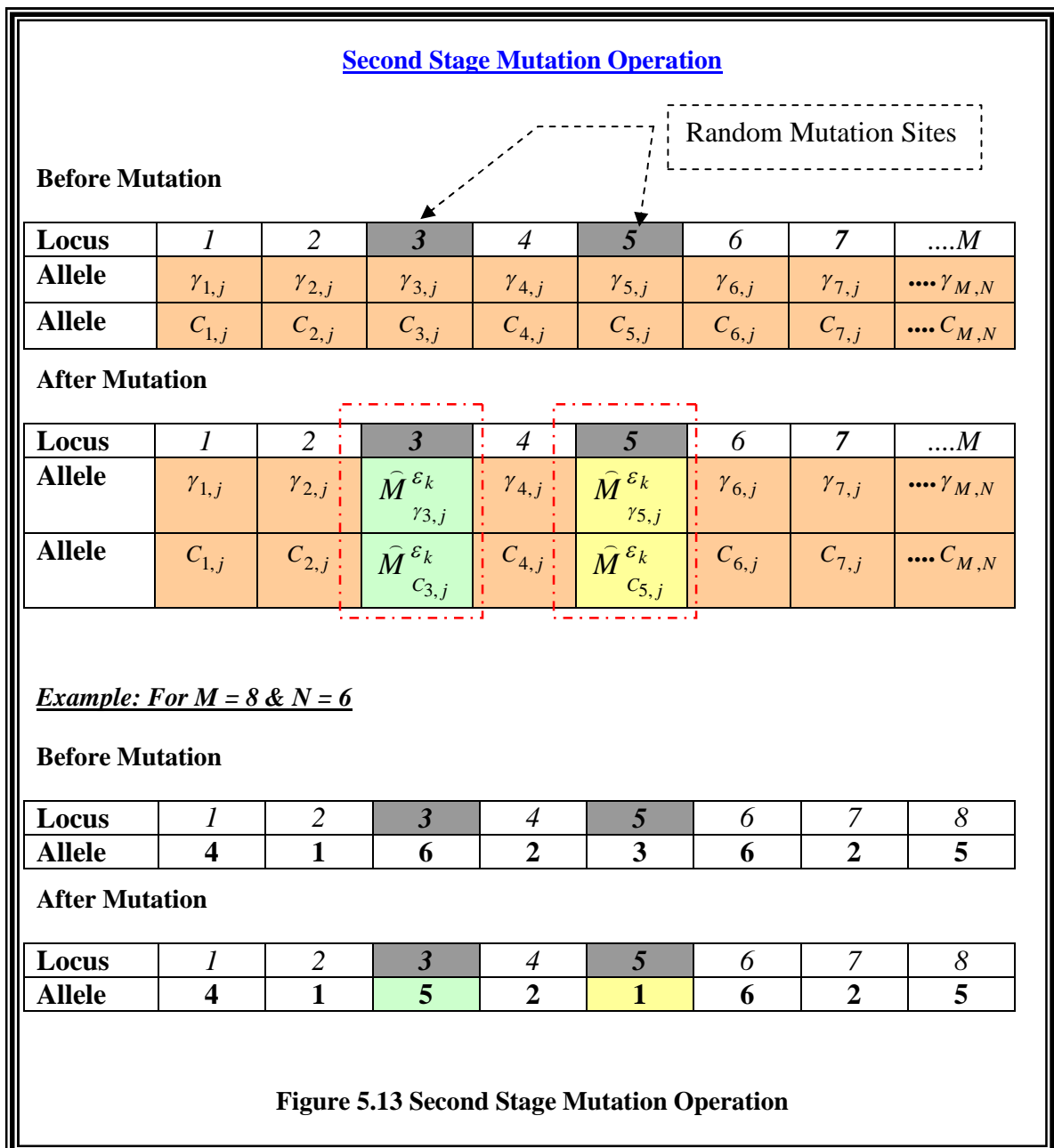
### 5.3.4.2 Second Stage Mutation Operation (SSMO)

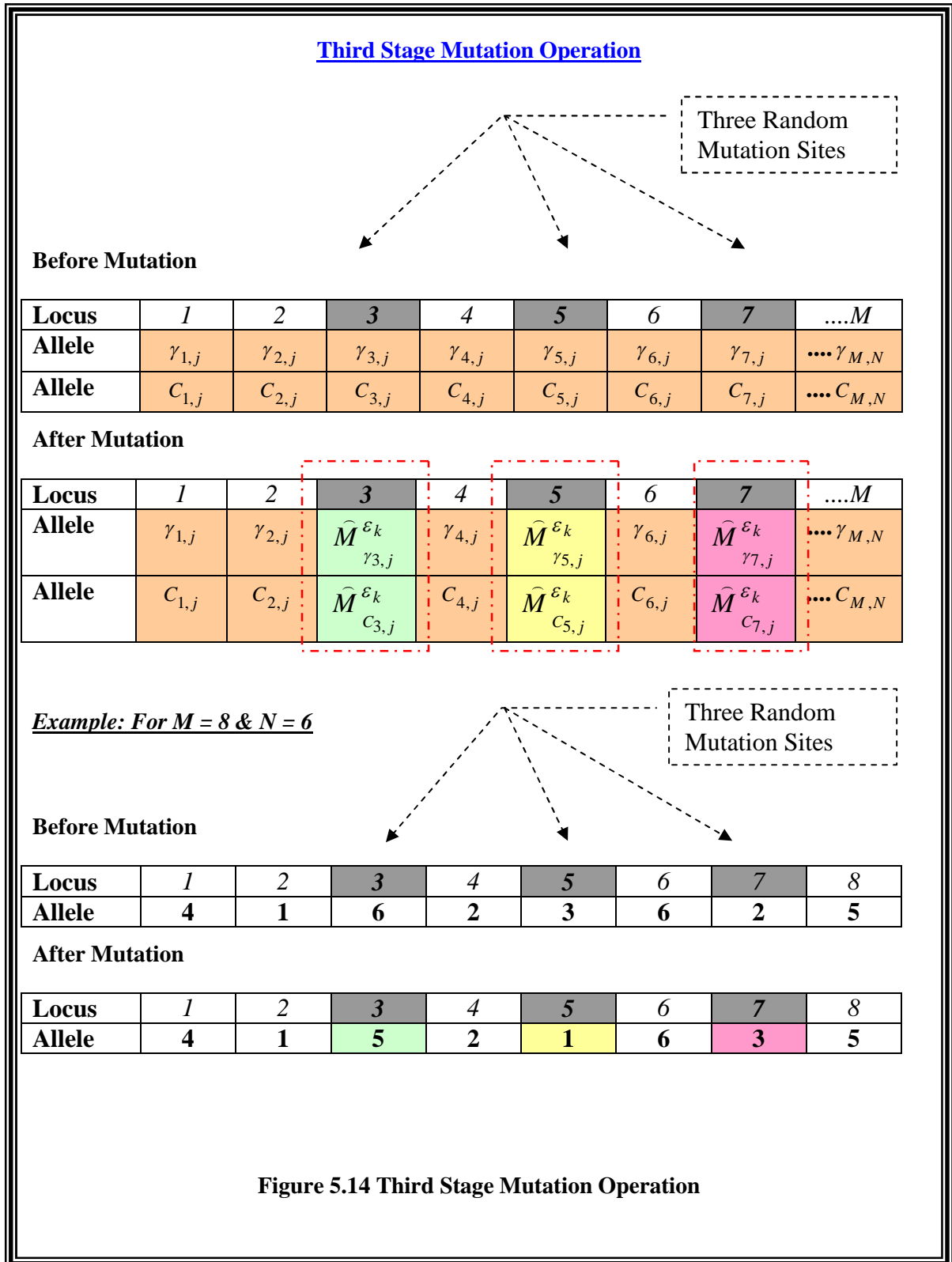
In the second stage mutation operation, two mutation sites (loci of the genes) are selected by using a uniform random number generator between '1' and ' $M$ ', where  $M$  is the length of the chromosome string, representing the total number of genes (components) in the solution string. Having selected the two genes (components) locations for performing the mutation operation in the chromosome string, the alleles of ' $\varepsilon_1$ ' are varied by replacing the current values of the selected genes with the ' $j^{th}$ ', alternatives from the given choice of ' $N$ ' alternatives for the respective components. The selection of the ' $j^{th}$ ' alternative is performed by using a uniform random number generator between '1' and ' $N$ ', where ' $N$ ' is the total number of available alternatives for a particular component ' $i$ ', as defined in the equations 5.1 and 5.2. The process of second stage mutation operation is shown in Fig. 5.13, by using the same chromosome structure as highlighted previously in Fig. 5.12 above. If the new chromosomes produced in the SSMS are weaker in fitness than either ' $\varepsilon_1$ ' or ' $\varepsilon_2$ ', they are immediately discarded for being inadequate solutions. Otherwise, they replace either of the two best solutions, depending on their fitness levels. At the end of the second stage mutation operation, the current best solution ' $\varepsilon_1$ ' is considered as the local optimum and subsequently presented for the final stage of the mutation operation.

### 5.3.4.3 Third Stage Mutation Operation (TSMO)

In the third stage of the mutation operation, three mutation sites (loci of the genes) are selected by using a uniform random number generator between '1' and ' $M$ ', where  $M$  is the length of the chromosome string, representing the total number of genes (components) in the solution string. Having selected the three genes (components) locations for performing the mutation operation in the chromosome string, the alleles of the best chromosome, ' $\varepsilon_1$ ', are varied by replacing the current values of the three selected genes with the ' $j^{th}$ ', alternatives from the given choice of ' $N$ ' alternatives for the respective components. The selection of the ' $j^{th}$ ' alternative is performed by using a uniform random number generator between '1' and ' $N$ ', where ' $N$ ' is the total

number of available alternatives for a particular component ‘ $i$ ’, as defined previously in the equations 5.1 and 5.2. The process of third stage mutation operation is shown in Fig. 5.14. Similar to the previous stages of mutations, if the new chromosomes produced in the TSMS are better in fitness than either ‘ $\varepsilon_1$ ’ or ‘ $\varepsilon_2$ ’, they immediately replace the respective best solutions for being more promising solutions. Otherwise, these new offspring are culled because of the weak fitness levels. At the end of this final stage of the mutation operation, the current best solution ‘ $\varepsilon_1$ ’ is regarded as the local optimum and subsequently presented as the best solution of the optimisation problem in hand, if the termination criteria is reached.







### 5.3.4.4 Numerical Example

The mutation stages defined above can be shown by a numerical example, using the same reliability system as used in section 5.3.3.4, which consists of eight components ( $M = 8$ ) & six alternatives each ( $N = 6$ ). The cost and reliability data is defined in equations (5.4) and (5.5), above.

#### *First Stage Mutation Operation*

From the chromosome structure as depicted in Fig. 5.7, let's assume the best chromosomes ' $\varepsilon_1$ ' is given by:

Chromosome ' $\varepsilon_1$ ':

Locus	1	2	3	4	5	6	7	8
Allele	4	1	6	2	3	6	2	5

The mapping of the best chromosome to the respective domains of reliability & cost can be carried out using equations (5.4) and (5.5).

Chromosome ' $\varepsilon_1$ ' (Decoded Value):

Locus	1	2	3	4	5	6	7	8
Allele-Rel	0.80	0.50	0.95	0.60	0.70	0.95	0.60	0.90
Allele-Cost	100	400	950	200	310	1000	100	890

If random mutation site = 5, then,

Chromosome ' $\varepsilon_1$ ' before FSMO:

Locus	1	2	3	4	5	6	7	8
Allele	4	1	6	2	3	6	2	5

Chromosome ' $\varepsilon_1$ ' after FSMO:

Locus	1	2	3	4	5	6	7	8
Allele	4	1	6	2	1	6	2	5

The mapping of the new chromosome to the respective domains of reliability & cost can also be carried out using equations (5.4) and (5.5).

*New Chromosome (Decoded Value):*

Locus	1	2	3	4	5	6	7	8
Allele-Rel	0.80	0.50	0.95	0.60	0.55	0.95	0.60	0.90
Allele-Cost	100	400	950	200	105	1000	100	890

### ***Second Stage Mutation Operation***

Similar to the first stage, the second stage mutation can be demonstrated using the elitist chromosome, ' $\varepsilon_1$ '.

If random mutation site = 3 and 5, then,

*Chromosome ' $\varepsilon_1$ ' before SSMO:*

Locus	1	2	3	4	5	6	7	8
Allele	4	1	6	2	3	6	2	5

*Chromosome ' $\varepsilon_1$ ' after SSMO:*

Locus	1	2	3	4	5	6	7	8
Allele	4	1	5	2	1	6	2	5

*Chromosome ' $\varepsilon_1$ ' (Decoded Value):*

Locus	1	2	3	4	5	6	7	8
Allele-Rel	0.80	0.50	0.95	0.60	0.70	0.95	0.60	0.90
Allele-Cost	100	100	950	200	310	1000	100	890

*New Chromosome (Decoded Value):*

Locus	1	2	3	4	5	6	7	8
Allele-Rel	0.80	0.50	0.90	0.60	0.55	0.95	0.60	0.90
Allele-Cost	100	100	800	200	105	1000	100	890

### ***Third Stage Mutation Operation***

Similarly, the third stage mutation operation can be demonstrated using the elitist chromosome, ' $\varepsilon_1$ '.

If random mutation site = 3, 5, and 7, then,

Chromosome ' $\varepsilon_1$ ' before TSMO:

Locus	1	2	3	4	5	6	7	8
Allele	4	1	6	2	3	6	2	5

Chromosome ' $\varepsilon_1$ ' after TSMO:

Locus	1	2	3	4	5	6	7	8
Allele	4	1	5	2	1	6	3	5

Chromosome ' $\varepsilon_1$ ' (Decoded Value):

Locus	1	2	3	4	5	6	7	8
Allele-Rel	0.80	0.50	0.95	0.60	0.70	0.95	0.60	0.90
Allele-Cost	100	100	950	200	310	1000	100	890

New Chromosome (Decoded Value):

Locus	1	2	3	4	5	6	7	8
Allele-Rel	0.80	0.50	0.90	0.60	0.55	0.95	0.75	0.90
Allele-Cost	100	100	800	200	105	1000	300	890

### 5.3.5 Improvement Procedures

An 'improvement process' as mentioned in the framework of the optimisation algorithm (Fig. 5.6), combines the exploration abilities of genetic search with hill-climbing procedures and is introduced in the hope of further streamlining the fitness of the best chromosomes,  $(\varepsilon_1, \varepsilon_2)$ . Given the nature of the optimisation problem discussed in this research and the complex relationship between cost and reliability (section 5.1), the objective of the improvement function is to explore large search areas with skilful exploitation of all feasible solutions. This involves testing large samples of component combinations quickly and efficiently and at the same time, examining the local region of the current best solution (local optimum) for possibly discovering an improved version of this solution; for this reason, the improvement procedure can be considered as the corner stone of the optimisation algorithm.

### 5.3.5.1 Types of Improvement Procedures

There are two types of improvement procedures implemented in the optimisation algorithm, depending on the steps in which they are used in the algorithm. The first instance of the procedure is introduced in step 5(a) of the algorithm and is called ‘*IMPROVE\_1()*’ whereas, the second procedure is implemented in step 5(g) and is called ‘*IMPROVE\_2()*’. The details of the two procedures are described below.

#### 5.3.5.1.1 First Improvement Procedure (*IMPROVE\_1()*)

This improvement procedure is applied after the first stage crossover operation as defined in section 5.3.3.1 above and made up of the two additional crossover operations (Fig. 5.15), which are defined in section 5.3.3.2 and 5.3.3.3, respectively.

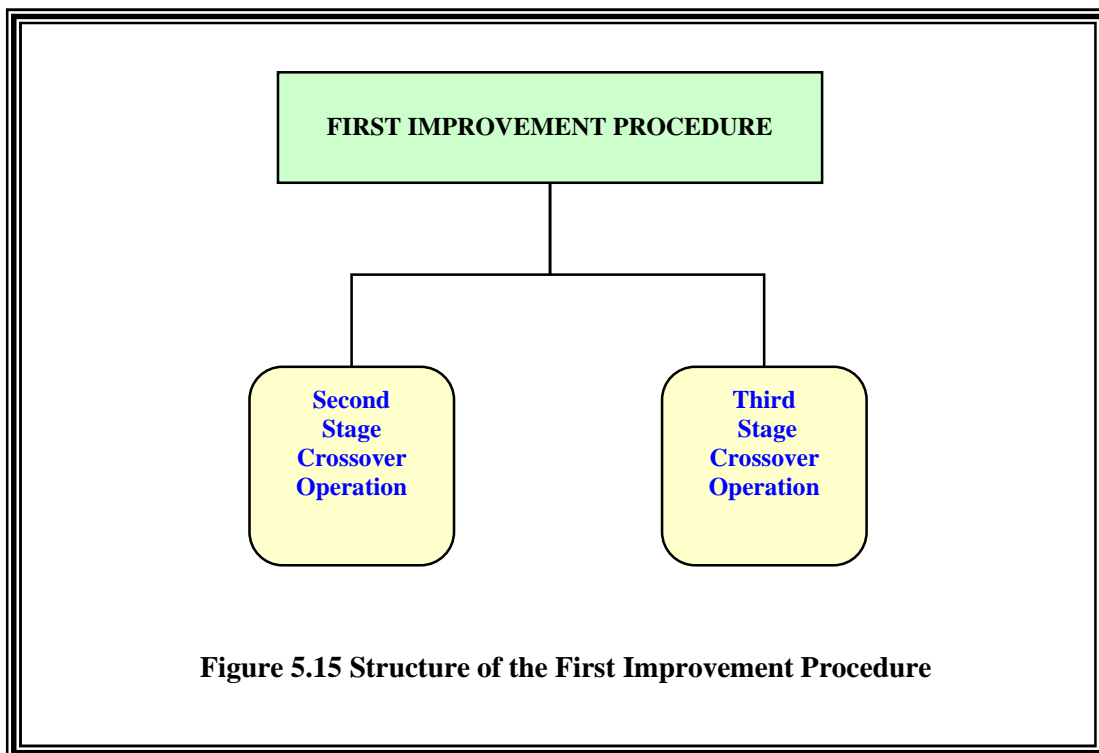


Figure 5.15 Structure of the First Improvement Procedure

The reason for designing additional stages of the crossover operation is to develop an efficient technique for exploring the search space while exploiting the current two best solutions. Thus, the paradigm of the first improvement procedure is to skilfully use the genetic information of the current best solutions  $(\varepsilon_1, \varepsilon_2)$  by introducing additional

crossover operations. These operations exhausts these solutions by randomly generating multiple crossover points (loci of the genes) and producing progeny by swapping the corresponding alleles of only the selected genes in the hope of finding even better solutions. For a system consisting of eight components with six alternatives each, lets imagine the two best solutions ( $\varepsilon_1, \varepsilon_2$ ) found in a particular iteration of the algorithm are:

1- Optimum Solution ( $\varepsilon_1$ )

1	2	3	4	5	6	7	8
$\gamma_{1,4}, C_{1,4}$	$\gamma_{2,2}, C_{2,2}$	$\gamma_{3,4}, C_{3,4}$	$\gamma_{4,6}, C_{4,6}$	$\gamma_{5,4}, C_{5,4}$	$\gamma_{6,2}, C_{6,2}$	$\gamma_{7,4}, C_{7,4}$	$\gamma_{8,5}, C_{8,5}$

$\gamma_{1,1}, C_{1,1}$	$\gamma_{1,2}, C_{1,2}$	$\gamma_{1,3}, C_{1,3}$	$\gamma_{1,4}, C_{1,4}$	$\gamma_{1,5}, C_{1,5}$	$\gamma_{1,6}, C_{1,6}$
$\gamma_{2,1}, C_{2,1}$	$\gamma_{2,2}, C_{2,2}$	$\gamma_{2,3}, C_{2,3}$	$\gamma_{2,4}, C_{2,4}$	$\gamma_{2,5}, C_{2,5}$	$\gamma_{2,6}, C_{2,6}$
$\gamma_{3,1}, C_{3,1}$	$\gamma_{3,2}, C_{3,2}$	$\gamma_{3,3}, C_{3,3}$	$\gamma_{3,4}, C_{3,4}$	$\gamma_{3,5}, C_{3,5}$	$\gamma_{3,6}, C_{3,6}$
$\gamma_{4,1}, C_{4,1}$	$\gamma_{4,2}, C_{4,2}$	$\gamma_{4,3}, C_{4,3}$	$\gamma_{4,4}, C_{4,4}$	$\gamma_{4,5}, C_{4,5}$	$\gamma_{4,6}, C_{4,6}$
$\gamma_{5,1}, C_{5,1}$	$\gamma_{5,2}, C_{5,2}$	$\gamma_{5,3}, C_{5,3}$	$\gamma_{5,4}, C_{5,4}$	$\gamma_{5,5}, C_{5,5}$	$\gamma_{5,6}, C_{5,6}$
$\gamma_{6,1}, C_{6,1}$	$\gamma_{6,2}, C_{6,2}$	$\gamma_{6,3}, C_{6,3}$	$\gamma_{6,4}, C_{6,4}$	$\gamma_{6,5}, C_{6,5}$	$\gamma_{6,6}, C_{6,6}$
$\gamma_{7,1}, C_{7,1}$	$\gamma_{7,2}, C_{7,2}$	$\gamma_{7,3}, C_{7,3}$	$\gamma_{7,4}, C_{7,4}$	$\gamma_{7,5}, C_{7,5}$	$\gamma_{7,6}, C_{7,6}$
$\gamma_{8,1}, C_{8,1}$	$\gamma_{8,2}, C_{8,2}$	$\gamma_{8,3}, C_{8,3}$	$\gamma_{8,4}, C_{8,4}$	$\gamma_{8,5}, C_{8,5}$	$\gamma_{8,6}, C_{8,6}$

2- Optimum Solution ( $\varepsilon_2$ )

1	2	3	4	5	6	7	8
$\gamma_{1,1}, C_{1,1}$	$\gamma_{2,2}, C_{2,2}$	$\gamma_{3,4}, C_{3,4}$	$\gamma_{4,2}, C_{4,2}$	$\gamma_{5,1}, C_{5,1}$	$\gamma_{6,4}, C_{6,4}$	$\gamma_{7,1}, C_{7,1}$	$\gamma_{8,4}, C_{8,4}$

$\gamma_{1,1}, C_{1,1}$	$\gamma_{1,2}, C_{1,2}$	$\gamma_{1,3}, C_{1,3}$	$\gamma_{1,4}, C_{1,4}$	$\gamma_{1,5}, C_{1,5}$	$\gamma_{1,6}, C_{1,6}$
$\gamma_{2,1}, C_{2,1}$	$\gamma_{2,2}, C_{2,2}$	$\gamma_{2,3}, C_{2,3}$	$\gamma_{2,4}, C_{2,4}$	$\gamma_{2,5}, C_{2,5}$	$\gamma_{2,6}, C_{2,6}$
$\gamma_{3,1}, C_{3,1}$	$\gamma_{3,2}, C_{3,2}$	$\gamma_{3,3}, C_{3,3}$	$\gamma_{3,4}, C_{3,4}$	$\gamma_{3,5}, C_{3,5}$	$\gamma_{3,6}, C_{3,6}$
$\gamma_{4,1}, C_{4,1}$	$\gamma_{4,2}, C_{4,2}$	$\gamma_{4,3}, C_{4,3}$	$\gamma_{4,4}, C_{4,4}$	$\gamma_{4,5}, C_{4,5}$	$\gamma_{4,6}, C_{4,6}$
$\gamma_{5,1}, C_{5,1}$	$\gamma_{5,2}, C_{5,2}$	$\gamma_{5,3}, C_{5,3}$	$\gamma_{5,4}, C_{5,4}$	$\gamma_{5,5}, C_{5,5}$	$\gamma_{5,6}, C_{5,6}$
$\gamma_{6,1}, C_{6,1}$	$\gamma_{6,2}, C_{6,2}$	$\gamma_{6,3}, C_{6,3}$	$\gamma_{6,4}, C_{6,4}$	$\gamma_{6,5}, C_{6,5}$	$\gamma_{6,6}, C_{6,6}$
$\gamma_{7,1}, C_{7,1}$	$\gamma_{7,2}, C_{7,2}$	$\gamma_{7,3}, C_{7,3}$	$\gamma_{7,4}, C_{7,4}$	$\gamma_{7,5}, C_{7,5}$	$\gamma_{7,6}, C_{7,6}$
$\gamma_{8,1}, C_{8,1}$	$\gamma_{8,2}, C_{8,2}$	$\gamma_{8,3}, C_{8,3}$	$\gamma_{8,4}, C_{8,4}$	$\gamma_{8,5}, C_{8,5}$	$\gamma_{8,6}, C_{8,6}$

The improvement procedure exploits the two best chromosomes by gradually interchanging the genes and steadily exploring the local search region of both of the good solutions. Consequently, the algorithm is able to perform multi-directional search while retaining the best solution in the population. The shaded area in the table below represents the components which can be used to construct random combinations in each iteration; combinations can have one component from every row of the matrix.

$\gamma_{1,1}, C_{1,1}$	$\gamma_{1,2}, C_{1,2}$	$\gamma_{1,3}, C_{1,3}$	$\gamma_{1,4}, C_{1,4}$	$\gamma_{1,5}, C_{1,5}$	$\gamma_{1,6}, C_{1,6}$
$\gamma_{2,1}, C_{2,1}$	$\gamma_{2,2}, C_{2,2}$	$\gamma_{2,3}, C_{2,3}$	$\gamma_{2,4}, C_{2,4}$	$\gamma_{2,5}, C_{2,5}$	$\gamma_{2,6}, C_{2,6}$
$\gamma_{3,1}, C_{3,1}$	$\gamma_{3,2}, C_{3,2}$	$\gamma_{3,3}, C_{3,3}$	$\gamma_{3,4}, C_{3,4}$	$\gamma_{3,5}, C_{3,5}$	$\gamma_{3,6}, C_{3,6}$
$\gamma_{4,1}, C_{4,1}$	$\gamma_{4,2}, C_{4,2}$	$\gamma_{4,3}, C_{4,3}$	$\gamma_{4,4}, C_{4,4}$	$\gamma_{4,5}, C_{4,5}$	$\gamma_{4,6}, C_{4,6}$
$\gamma_{5,1}, C_{5,1}$	$\gamma_{5,2}, C_{5,2}$	$\gamma_{5,3}, C_{5,3}$	$\gamma_{5,4}, C_{5,4}$	$\gamma_{5,5}, C_{5,5}$	$\gamma_{5,6}, C_{5,6}$
$\gamma_{6,1}, C_{6,1}$	$\gamma_{6,2}, C_{6,2}$	$\gamma_{6,3}, C_{6,3}$	$\gamma_{6,4}, C_{6,4}$	$\gamma_{6,5}, C_{6,5}$	$\gamma_{6,6}, C_{6,6}$
$\gamma_{7,1}, C_{7,1}$	$\gamma_{7,2}, C_{7,2}$	$\gamma_{7,3}, C_{7,3}$	$\gamma_{7,4}, C_{7,4}$	$\gamma_{7,5}, C_{7,5}$	$\gamma_{7,6}, C_{7,6}$
$\gamma_{8,1}, C_{8,1}$	$\gamma_{8,2}, C_{8,2}$	$\gamma_{8,3}, C_{8,3}$	$\gamma_{8,4}, C_{8,4}$	$\gamma_{8,5}, C_{8,5}$	$\gamma_{8,6}, C_{8,6}$

**Table 5.1. Domain of Reliability and Cost for A Sample System Showing Selected Number of Components Which Can Be Used To Form Combinations Per Iteration in the First Improvement Procedure**

The improvement process is configured according to the complexity of the optimisation problem. In the context of this research, the improvement function is configured to perform at first, a two-stage crossover operation (selecting two random genes for swapping alleles) and secondly, a three-stage crossover operation which selects three random genes for swapping corresponding alleles in the two best fit chromosomes (Fig. 5.10 & Fig. 5.11). All crossover operations run for ' $C_{RUN}$ ' number of times. During these operations, the objective is to evaluate the progeny from each iteration (up to a maximum of ' $C_{RUN}$ ') and anticipating improvement of fitness levels in the existing best solutions ( $\varepsilon_1, \varepsilon_2$ ), by means of recombining their features and forming new versions of these solutions. If the new found solution is better than either of the two parent chromosomes, ( $\varepsilon_1, \varepsilon_2$ ), it instantly replaces the corresponding parent and the improvement procedure continues. At the end of this process, only the two best

chromosomes survive and the rest are discarded. If either or both  $(\varepsilon_1, \varepsilon_2)$  are updated in the improvement process, they are available for inheritance (Lamarckian Evolution – (Jones and Kay, 2002)) and the only the best of the two,  $(\varepsilon_1)$ , also the local optimum until the termination criterion is met) is used for the second improvement procedure.

### 5.3.5.1.2 Second Improvement Procedure (*IMPROVE\_2* ( ))

Although the crossover operation is a very powerful technique for exploring the search space, it also has a significant weakness. Since it proceeds by recombining information (alleles) from parents, the progeny produced ideally contains only alleles that were already present in either or both of the parents. In other words, it never produces new alleles which can be a big problem for genetic search involving very large search space, (Falkenauer, 1998). This is where mutation operation is applied in order to introduce diversity in the solution space and avoid a premature convergence of the optimisation algorithm. For a system consisting of eight components with six alternatives each, let's imagine the best solutions ' $\varepsilon_1$ ' found in a particular iteration of the algorithm is:

*Optimum Solution* ( $\varepsilon_1$ )

1	2	3	4	5	6	7	8
$\gamma_{1,4}, C_{1,4}$	$\gamma_{2,2}, C_{2,2}$	$\gamma_{3,4}, C_{3,4}$	$\gamma_{4,6}, C_{4,6}$	$\gamma_{5,4}, C_{5,4}$	$\gamma_{6,2}, C_{6,2}$	$\gamma_{7,4}, C_{7,4}$	$\gamma_{8,5}, C_{8,5}$

$\gamma_{1,1}, C_{1,1}$	$\gamma_{1,2}, C_{1,2}$	<del><math>\gamma_{1,3}, C_{1,3}</math></del>	$\gamma_{1,4}, C_{1,4}$	$\gamma_{1,5}, C_{1,5}$	$\gamma_{1,6}, C_{1,6}$
$\gamma_{2,1}, C_{2,1}$	$\gamma_{2,2}, C_{2,2}$	<del><math>\gamma_{2,3}, C_{2,3}</math></del>	$\gamma_{2,4}, C_{2,4}$	$\gamma_{2,5}, C_{2,5}$	$\gamma_{2,6}, C_{2,6}$
$\gamma_{3,1}, C_{3,1}$	$\gamma_{3,2}, C_{3,2}$	$\gamma_{3,3}, C_{3,3}$	$\gamma_{3,4}, C_{3,4}$	<del><math>\gamma_{3,5}, C_{3,5}</math></del>	$\gamma_{3,6}, C_{3,6}$
$\gamma_{4,1}, C_{4,1}$	$\gamma_{4,2}, C_{4,2}$	$\gamma_{4,3}, C_{4,3}$	$\gamma_{4,4}, C_{4,4}$	<del><math>\gamma_{4,5}, C_{4,5}</math></del>	$\gamma_{4,6}, C_{4,6}$
$\gamma_{5,1}, C_{5,1}$	$\gamma_{5,2}, C_{5,2}$	<del><math>\gamma_{5,3}, C_{5,3}</math></del>	$\gamma_{5,4}, C_{5,4}$	$\gamma_{5,5}, C_{5,5}$	$\gamma_{5,6}, C_{5,6}$
$\gamma_{6,1}, C_{6,1}$	$\gamma_{6,2}, C_{6,2}$	<del><math>\gamma_{6,3}, C_{6,3}</math></del>	$\gamma_{6,4}, C_{6,4}$	$\gamma_{6,5}, C_{6,5}$	$\gamma_{6,6}, C_{6,6}$
$\gamma_{7,1}, C_{7,1}$	$\gamma_{7,2}, C_{7,2}$	<del><math>\gamma_{7,3}, C_{7,3}</math></del>	$\gamma_{7,4}, C_{7,4}$	<del><math>\gamma_{7,5}, C_{7,5}</math></del>	$\gamma_{7,6}, C_{7,6}$
$\gamma_{8,1}, C_{8,1}$	$\gamma_{8,2}, C_{8,2}$	$\gamma_{8,3}, C_{8,3}$	$\gamma_{8,4}, C_{8,4}$	$\gamma_{8,5}, C_{8,5}$	$\gamma_{8,6}, C_{8,6}$

The improvement procedure exploits the best chromosome by gradually altering the genes and steadily exploring the local search region of the solution. Consequently, the

algorithm is able to perform multi-directional search while retaining the best solution in the population. This process effectively deals with the epistasis found in the cost-reliability relationship since the local region of the best solution is evaluated by introducing gradual and minimum variation in the genes in the successive iterations.

The shaded area in the table below represents all the new genes which can be selected randomly by moving either side of the selected genes (mutation sites) in the chromosome string of the best solution. The optimisation algorithm therefore introduces greater diversity in the search space while directing the search towards optimum solution.

$\gamma_{1,1}, C_{1,1}$	$\gamma_{1,2}, C_{1,2}$	$\gamma_{1,3}, C_{1,3}$	$\gamma_{1,4}, C_{1,4}$	$\gamma_{1,5}, C_{1,5}$	$\gamma_{1,6}, C_{1,6}$
$\gamma_{2,1}, C_{2,1}$	$\gamma_{2,2}, C_{2,2}$	$\gamma_{2,3}, C_{2,3}$	$\gamma_{2,4}, C_{2,4}$	$\gamma_{2,5}, C_{2,5}$	$\gamma_{2,6}, C_{2,6}$
$\gamma_{3,1}, C_{3,1}$	$\gamma_{3,2}, C_{3,2}$	$\gamma_{3,3}, C_{3,3}$	$\gamma_{3,4}, C_{3,4}$	$\gamma_{3,5}, C_{3,5}$	$\gamma_{3,6}, C_{3,6}$
$\gamma_{4,1}, C_{4,1}$	$\gamma_{4,2}, C_{4,2}$	$\gamma_{4,3}, C_{4,3}$	$\gamma_{4,4}, C_{4,4}$	$\gamma_{4,5}, C_{4,5}$	$\gamma_{4,6}, C_{4,6}$
$\gamma_{5,1}, C_{5,1}$	$\gamma_{5,2}, C_{5,2}$	$\gamma_{5,3}, C_{5,3}$	$\gamma_{5,4}, C_{5,4}$	$\gamma_{5,5}, C_{5,5}$	$\gamma_{5,6}, C_{5,6}$
$\gamma_{6,1}, C_{6,1}$	$\gamma_{6,2}, C_{6,2}$	$\gamma_{6,3}, C_{6,3}$	$\gamma_{6,4}, C_{6,4}$	$\gamma_{6,5}, C_{6,5}$	$\gamma_{6,6}, C_{6,6}$
$\gamma_{7,1}, C_{7,1}$	$\gamma_{7,2}, C_{7,2}$	$\gamma_{7,3}, C_{7,3}$	$\gamma_{7,4}, C_{7,4}$	$\gamma_{7,5}, C_{7,5}$	$\gamma_{7,5}, C_{7,5}$
$\gamma_{8,1}, C_{8,1}$	$\gamma_{8,2}, C_{8,2}$	$\gamma_{8,3}, C_{8,3}$	$\gamma_{8,4}, C_{8,4}$	$\gamma_{8,5}, C_{8,5}$	$\gamma_{8,5}, C_{8,5}$

**Table 5.2 Domain of Reliability and Cost for A Sample System Showing Selected Number of Components Which Can be Used to Form Combinations Per Iteration in the Second Improvement Procedure**

The second improvement procedure is applied after the first stage mutation operation as defined in section 5.3.4.1 above and composed of the two additional mutation operations (Fig. 5.16), which are defined in section 5.3.4.2 and 5.3.4.3, respectively. The reason for designing additional stages of the mutation operation is to develop an effective technique for exploring the search space while exploiting the local region of the current best solution. Thus, the paradigm of the second improvement procedure is to competently use the genetic information of the current best solution ' $\varepsilon_1$ ' by introducing additional mutation operations. These operations exhaust the best solution by randomly



generating multiple mutation points (loci of the genes) and producing progeny by altering only the corresponding allele of the selected genes in the hope of discovering an even better version of the current solution.

The '*IMPROVE\_2()*' procedure introduces new ideas for better solutions by accommodating a multidirectional search in the search space and is therefore applied more frequently than the first improvement procedure. After the standard mutation process as in step 5(f) of the algorithm, the '*IMPROVE\_2()*' procedure is applied which exhausts ' $\varepsilon_1$ ' by randomly generating various loci of genes, and altering only the corresponding alleles. This process produces modified versions of the best chromosome, ' $\varepsilon_1$ ', which are evaluated in the hope of finding even better solutions. The improvement process is configured according to the complexity of the optimisation problem. In the context of this research, the improvement function is configured to perform at first, a two-point mutation operation (selecting two random genes for altering alleles), Fig. 5.13 and secondly, a three-point mutation operation, Fig. 5.14, which selects three random genes for altering corresponding alleles in the best fit chromosome ( $\varepsilon_1$ ), while running both operations for ' $M_{RUN}$ ' number of times each.

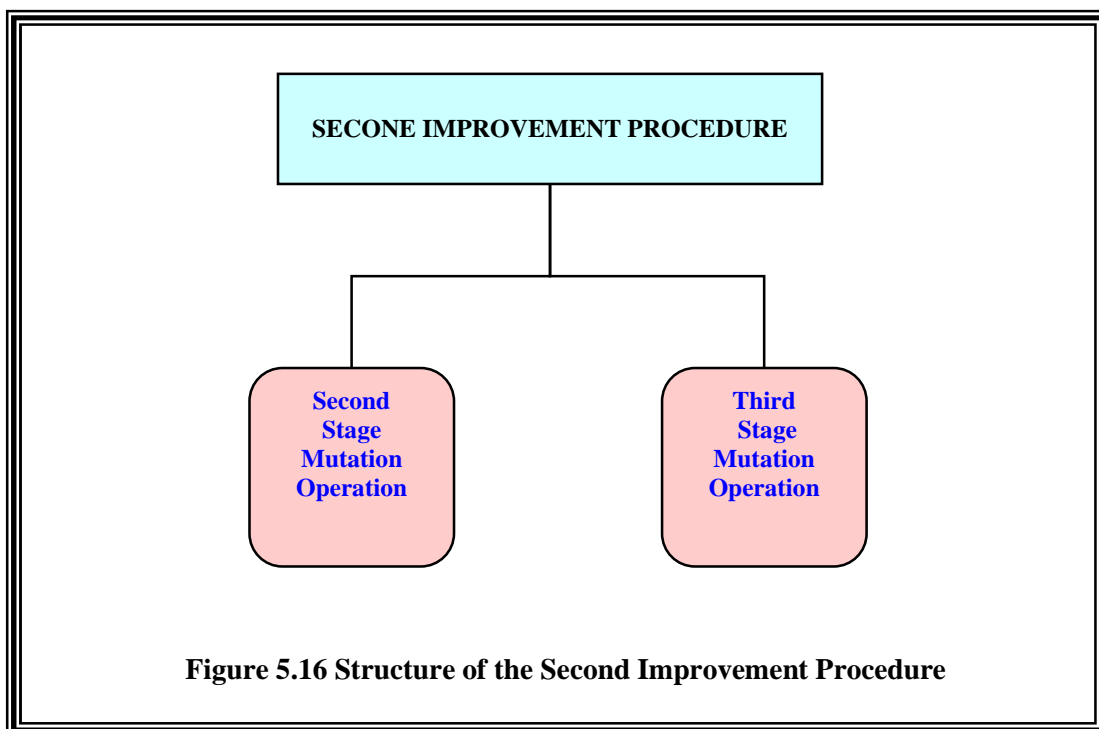


Figure 5.16 Structure of the Second Improvement Procedure

During these two mutation operations, the objective is to evaluate the progeny in each iteration (up to a maximum of ' $M_{RUN}$ ' for each operation) expecting to reveal more promising solutions by streamlining the existing best chromosome ' $(\varepsilon_1)$ '. At the end of the improvement process, only the two best chromosomes survive and the rest are discarded. If the termination criterion is achieved, the best of the two solutions,  $(\varepsilon_1)$  is highlighted as the optimum solution of the optimisation problem being considered. Otherwise, both  $(\varepsilon_1, \varepsilon_2)$  are automatically selected to join the next population.

### 5.3.6 Termination Criteria

The algorithm is repeated for ' $p_{RUN}$ ' number of times depending on the values of ' $p_{size}$ ', ' $C_{RUN}$ ' and ' $M_{RUN}$ '. For higher values of these parameters, the total number of generations, ' $p_{RUN}$ ', can be very small as more sampling of the search space will be carried out without having to run a large number of population cycles (generations). This is because, more potential solutions will be examined due to the large number of genetically modified solutions, produced as the result of the crossover, mutation and the two improvement procedures. Generally, good quality solutions are found with smaller value of the ' $p_{RUN}$ ' but higher values of ' $C_{RUN}$ ' and in particular ' $M_{RUN}$ '. However, the absolute values of these parameters will change in line with the scale and complexity of the optimisation problem in hand.

### 5.3.7 Process Diagram

A pictorial demonstration of the algorithm is also outlined in Fig. 5.17.

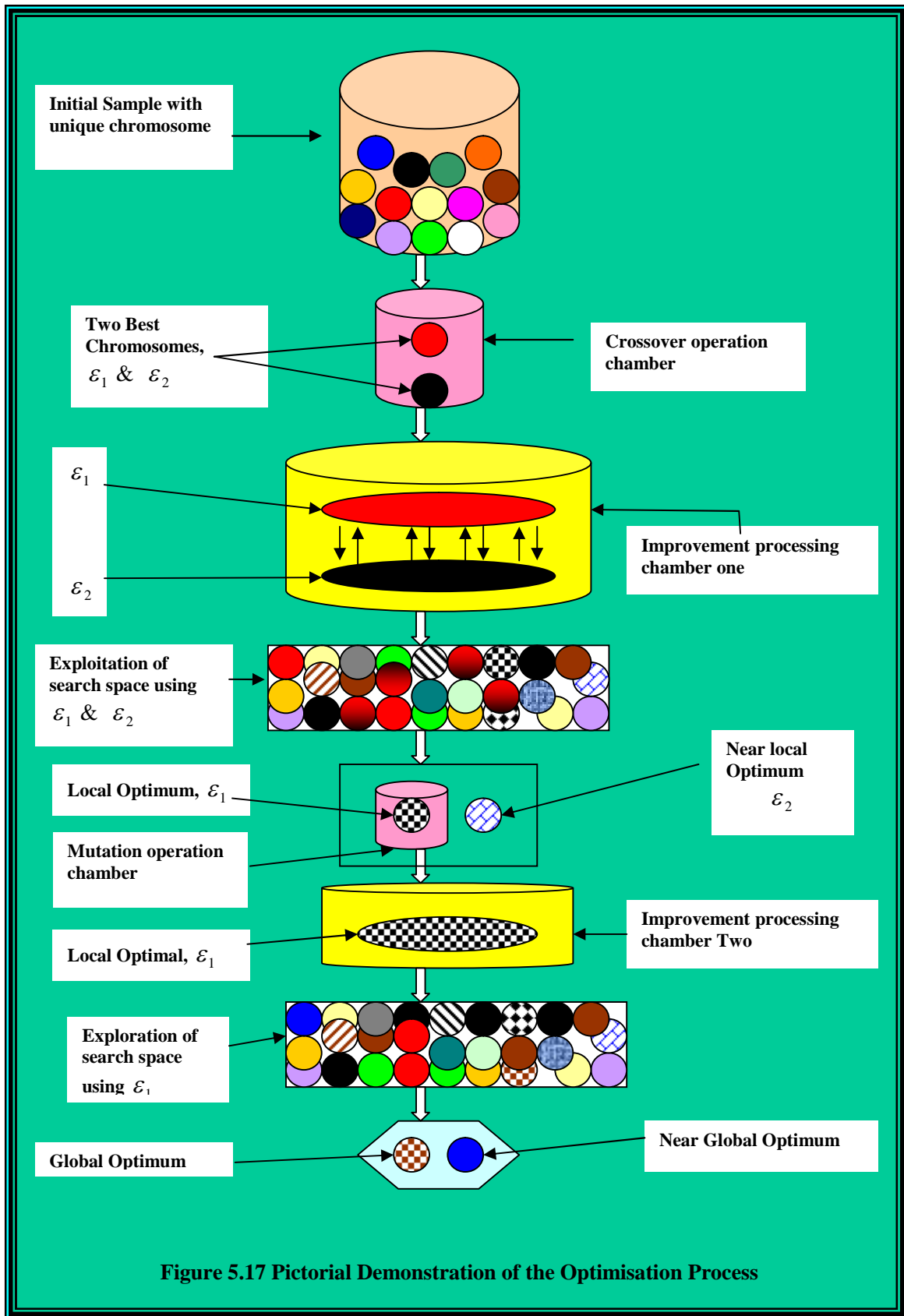


Figure 5.17 Pictorial Demonstration of the Optimisation Process

# 6

## **APPLICATIONS & RESULTS** *(RESEARCH METHODOLOGY)*

---

The optimisation algorithm (OA) introduced in this research has been applied to study several non-repairable engineering systems, using the risk-based reliability allocation method. The details of these systems and the results of the algorithm are described in this chapter. The first half demonstrates the validity of the optimisation algorithm by applying it to solve an example system from an already published article. The later half of this chapter shows the application of the optimisation algorithm on a number of reliability systems using a discrete choice of components exhibiting both monotonically and non-monotonically increasing relationships between costs and reliabilities among the alternatives. The algorithm is also used to study the effect of different levels of cost of failure for each system and the results are presented in the form of graphs and tables. The detailed discussion on the observations and findings of the optimisation algorithm by comparing systems individually and as a whole will be discussed in the next chapter.

## **6.1 METHODOLOGY DEVELOPMENT**

An optimisation algorithm has been developed as the research methodology which provides a structured approach for allocating optimal system reliability by using the risk based reliability method when a large choice of component alternatives is available. The approach is based on the realisation that the relationship between the component cost and reliability is generally very complex and unpredictable particularly for estimating the amounts of total loss (sum of reliability investment and risk of failure) associated with non-repairable system failures.

The optimisation problem studied in this research belongs to the class of combinatorial optimisations, which are well known for their complexity when the number of parameters involved in the underlying problem is large. For systems consisting of a large number of components with many choices of alternatives for each component, finding the optimal combination of components alternatives which minimises the loss function is therefore a very challenging problem, significantly in the presence of a non-linear and unpredictable relationship between cost and reliability parameters. Based on this understanding, the methodology developed in this research focuses attention on the need to skilfully evaluate this relationship between the two parameters when performing the optimisation process and choosing the optimal combination of alternatives. The underlying thoughts in designing the evaluation process can be explained in the following steps:

### **6.1.1 Component Characteristic**

Assuming availability, many versions of a particular component can be found at different cost and reliability levels. Generally, the cost is considered as the monotonically increasing function of reliability but this is not always the case as explained in Chapter 5; a simplified version of the same component may be even cheaper while meeting the required reliability level, for example. Selecting the best component from the given choices is already a very difficult task for large scale

optimisation problems, assuming the non-monotone structure in the cost-reliability relationship, therefore, makes this optimisation problem even harder even for the non-analytical methods.

### **6.1.2 System Reliability**

Increasing the reliability of the components in a system generally increases the overall system reliability along with the cost of the system (sum of the cost of all components). While increasing the reliability reduces the probability of failure and subsequently, the risk of failure, the loss function will not be minimised efficiently since the increased cost of components will reduce the effect of decrease in the risk of failure. This process will be even more convoluted for non-monotonically increasing cost-reliability relationship among the alternatives of the system due to the presence of extreme uncertainty; cheaper components might be available in higher reliability or vice-versa. Besides this intricacy, the other difficulty is associated with the actual estimation of system reliability for large scale problems with complex configurations (e.g. combination of series and parallel configuration). The general practice of using the cut-set and tie-sets (Appendix –I) obtained from the reliability block diagrams in estimating the system reliability, can be very cumbersome due to the large quantity of these sets. Using commercial software is another approach which is expensive, impractical for the purpose of the system optimisation due to the large number of possible combinations of components and it does not guarantee the correct estimation of reliability for all systems.

### **6.1.3 Exploring the Search Space**

For a large number of components and respective alternatives, finding the optimal combination of alternatives which minimise the loss function will require evaluation of all combinations using the enumerative search method. This exhaustive search is however, unfeasible despite using the new and powerful computation technology (if available) due to the sheer size of the search space. An effective technique is therefore required to explore the search space efficiently for these combinatorial optimisation problems.

### **6.1.4 Exploiting the Search Space**

Similar to exploring the search space, utilising the information of the current best solution in the hope of finding an even better version of this solution can be very effective in the optimisation process carried out by using the risk based reliability allocation method. This requires skilfully searching the local region of the current best found solution.

### **6.1.5 Complexity of the Method**

The method for optimisation based on risk based reliability allocation approach is required to be practical, programmable in a computer language and most importantly with linear complexity; a solution taking hours of computer processing time is not deemed as an efficient solution for the type of reliability optimisation considered in this research. Additionally, the method is expected to be generic for all types of systems (non-repairable) using the commonly known exponential distribution for estimating the failure probability.

Based on the principal of genetic search, the research methodology is therefore designed by taking into account all of the above points in order to successfully carrying out the reliability optimisation of systems with a large choice of components by using the risk based reliability allocation method. It combines the exploration abilities of genetic search with skilful exploitation of hill climbing procedures and introduces a different model of evolution compared to classical GA. The main features of this model are the generation of populations with unique chromosomes, working exclusively with the elite chromosomes and introducing genetic variations in the elite chromosomes using prudently designed genetic operators for ensuring rapid and efficient convergence to optimum or near optimum region of the search space. The two main reasons for implementing these notions in the optimisation algorithm are the non-linear cost-reliability relationship and the extremely large search space. A comprehensive detail of this method is presented in Chapter 5. The application of the methodology is demonstrated in this chapter ( along with appendix II-IX) by using

four types of system configurations which are commonly found in the reliability literature and the results are discussed in detail in the next chapter.

## 6.2 APPLICATION OF THE METHODOLOGY

The research methodology developed in the previous chapter can be applied to optimise various engineering systems along with the risk based reliability allocation method. The mathematical model of the reliability allocation method is shown in equation (6.1). For validation purposes, the optimisation algorithm is first applied to solve an already published reliability optimisation problem with constraint on some target level of system reliability, which is required to be achieved at a minimum system cost (i.e. total cost of all components in the system). This process is detailed in section 6.3. The risk based reliability allocation method is demonstrated in the next section by using the optimisation algorithm on four different reliability systems with discrete choice of component data set (Table 6.1), showing monotonically increasing cost and reliability relationship among the alternatives, and a fixed amount of cost associated with a given failure of a system (cost of failure).

For the purpose of undertaking parametric studies, the same four systems are studied individually, in section 6.4, with two variations in the optimisation process. The first of which involves using a different choice of discrete data set of components (Table 6.13) and allocating optimal system reliability with minimum total loss. The new data set is different because it exhibits a non-monotonically increasing relationship between cost and reliability among the alternatives. The second variation in the optimisation process is introduced by means of a different cost of failure amount, associated with a given system failure. Using this amount, the risk based reliability allocation is performed on all four systems using Table 6.1. The results obtained from the applications of the optimisation algorithm on all four systems with two different data sets and cost of failure amounts, will be discussed in the next chapter for accentuating the findings.



There are three principles which act as the main constituents of the text detailed in this chapter and the knowledge of these is an important requisite for making the most of the described information. These three principles being, the model of risk based reliability allocation, the research methodology (optimisation algorithm) and the distinctive yet most innovative Monte-Carlo simulation method for determining system reliability and estimating the total amounts, associated with a given system failure. The first two principles are explained comprehensively in Chapter 1 and Chapter 5, respectively, whereas, the third principle is described in Appendix I. For the benefit of the reader, a brief explanation of the three principles is detailed next.

## 6.2.1 Reliability Allocation Model

The model of the risk based reliability allocation defines a loss function, which consists of the sum of reliability investment and risk of failure:

$$T_L = Q + K \quad (6.1)$$

where,

$$Q = f_1(c_{1,1}, c_{2,2}, c_{3,3}, \dots, c_{M,N}) \quad (6.2)$$

$$K = [1 - R_s(\gamma_{1,1}, \gamma_{2,2}, \gamma_{3,3}, \dots, \gamma_{M,N})] \times \bar{C} \quad (6.3)$$

such that,

$$Q = \sum_i^M c_i \quad (6.4)$$

$$\bar{C} = p_{f1} \bar{c}_1 + p_{f2} \bar{c}_2 + \dots + p_{fM} \bar{c}_M \quad (6.5)$$

and

$$R_s = \xi \left( R_1(\gamma_{1,1} \dots \gamma_{1,N}), R_2(\gamma_{2,1} \dots \gamma_{2,N}), \dots, R_M(\gamma_{M,1} \dots \gamma_{M,N}) \right) \quad (6.6)$$

Where, ' $R_s$ ' is a function of ' $M$ ' components, each with ' $N$ ' number of given alternatives, such that ' $\gamma_{ij}$ ' is the reliability of the ' $i^{th}$ ' component with ' $j^{th}$ ' alternative ' $T_L$ ' is the total loss from system failure before some specific time interval ' $a$ ', ' $Q$ ' is the cost of reliability investment towards risk reduction and is a function of component costs for all selected alternatives (equation 6.2), such that ' $c_{ij}$ ' is the cost of the ' $i^{th}$ ' component with ' $j^{th}$ ' alternative where  $i=(1,2,3...M)$  ,  $j=(1,2,3...N)$ , therefore, ' $Q$ ' is equal to the sum of the cost of selected alternatives, as in equation (6.4). Also, ' $p_f$ ' is the probability of failure, ' $R_s$ ' is the system reliability, ' $\bar{C}$ ' is the expected cost given failure of the system & ' $p_{fi} \bar{c}_i$ ' is the risk of failure of the ' $i^{th}$ ' component. Therefore, the model of risk based reliability allocation can be formulated in the form of an objective function below,

$$T_L = \text{Min} \left( \sum_{i=1}^M c_i + \left[ 1 - R_s(\gamma_{1,1}^*, \gamma_{2,1}^*, \dots, \gamma_{M,N}^*) \right] \times \bar{C} \right) \quad (6.7)$$

Where  $\sum_{i=1}^M c_i$  represents the reliability investment and consists of the total cost of the selected alternatives, ' $R_s(\gamma_{1,1}^*, \gamma_{2,1}^*, \dots, \gamma_{M,N}^*)$ ' is the reliability of the system with optimal (\*) set of selected alternatives and ' $\bar{C}$ ' is the expected cost given failure before some specified time interval, associated with the selected alternatives.

## 6.2.2 The Optimisation Algorithm (OA)

The optimisation process in the OA begins by randomly generating an initial population, ' $p_k$ ', (where  $i=1$ ) containing ' $p_{size}$ ' many solutions of the given optimisation problem and then evaluating each solution from the population for the

respective level of fitness in accordance with the objective function defined in equation (6.7). The two best found solutions from the whole population are stored and then selected to contribute their characteristics in the new solutions through genetic operations such as crossover and mutation. The new solutions obtained as the result of these operations, are also evaluated for their fitness levels and if they perform better in the fitness test then they instantly replace either of the two best solutions, previously stored. An important feature of the optimisation algorithm is to introduce improvement procedures during each of the two genetic operations. The object of these procedures is to search the local region of the two best solutions by randomly altering the structures of these solutions in the hope of discovering an even better version of these solutions. If the objective is achieved, the new solutions instantly replace the current best solutions depending on the level of their fitness. After the improvement procedures the two best solutions are automatically included in the next population, ' $p_{k+1}$ '. The process continues until ' $p_{RUN}$ ' many generations are executed and the best found result is highlighted as the optimum solution. The optimisation algorithm is also shown in Fig. 6.1. (See Chapter 5 for full details on OA).

### 6.2.3 Monte Carlo Method For Determining System Reliability And Total Loss

During the optimisation process, the OA is required to compute the system reliability and total loss amount for each combination of component evaluated for fitness using the given objective function. This is achieved by using either of the two Monte Carlo (MC) simulation methods explained in Appendix I. The first of these two methods was originally introduced by Todinov (2006, 2006a) and involves randomly simulating the number of failures using a MC based 'node-stacking' technique in an adjacency matrix. The matrix represents the reliability system and during each sample run of the MC simulation, the objective is to find a valid path between the start node and the end node of the adjacency matrix. The search of the path begins by checking the immediate neighbouring nodes of the start node and continuing in the direction of the nodes where the connection between the two corresponding nodes exists

uninterrupted until the end node is found. If no path exists, the system is deemed in a failure mode and the cost of failures of each of the failed component is determined. The number of system failures and the total cost of failures of all components are aggregated across all sample runs. At the end of the MC simulations, the system reliability and risk of failure are obtained from the total number of failures and the cumulative cost of failures.

The second method for determining the system reliability and risk of failure is very similar to Todinov's method and differs in the sense that the path between start node and end node is established without using the node-stacking technique. Also, the search of the path in the adjacency matrix is performed in reverse order by checking the existence of the immediate valid path between start and end node and gradually moving backward until a full connection is established. If no connection is found, the system is considered in a fail state. The calculation method of system reliability and risk of failure amount is similar to the first method.

It is important to point out that all the calculations performed in the next sections are conducted using the second method as listed above. This MC based method for calculating system reliability and total loss is programmed in C/C++ language and the code is also provided in Appendix X. The results obtained are based on fewer numbers of Monte Carlo generations because of large number of analysis conducted through out. However, increasing the sample size can further refine results, such as system reliability estimation, which appears to be acceptable even with the smaller number of samples used in this research. However, it has not been tested for problems which exist outside the scope of this research but given the excellent quality of the comparative results with the first method, it is deemed as a great potential for future studies.

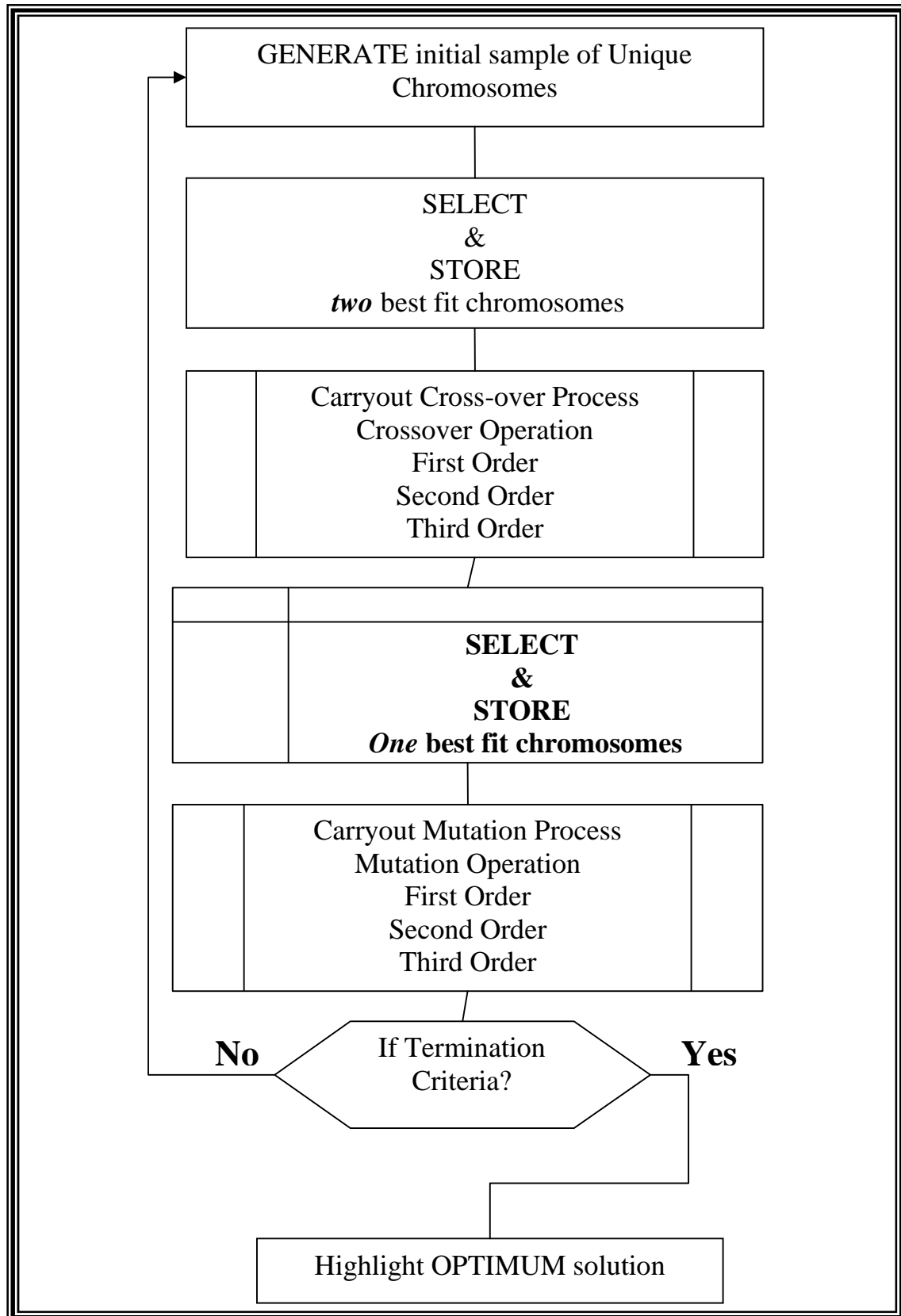


Figure 6. 1 Structure of the Optimisation Algorithm

### 6.3 RELIABILITY OPTIMISATION PROBLEM

The application of the optimisation algorithm is first validated against a reliability optimisation problem, previously solved by using the Simulated Annealing (SA) method in Majety *et al.* (1996). The optimal reliability allocation problem is NP-hard and focuses on a situation where a system with a certain configuration is required to be assembled from a given choice of components with different levels of cost and reliability. Therefore, the same level of system reliability can be achieved by using different combinations of components with various levels of associated system costs. For this reason and given the discrete choice of available components, the process of finding the optimal combination of components with minimum system cost for achieving some target level of system reliability understandably develops into a difficult combinatorial optimisation problem. The example reliability system from the referenced optimisation problem consists of nine components with twelve alternatives each. Additionally, the configuration of the reliability system portrays three subsystems in series arrangement, each containing three, four and two components, connected in parallel, respectively. The system is depicted in Fig.6.2 and cost-reliability data used in the example is detailed in Table 6.1.

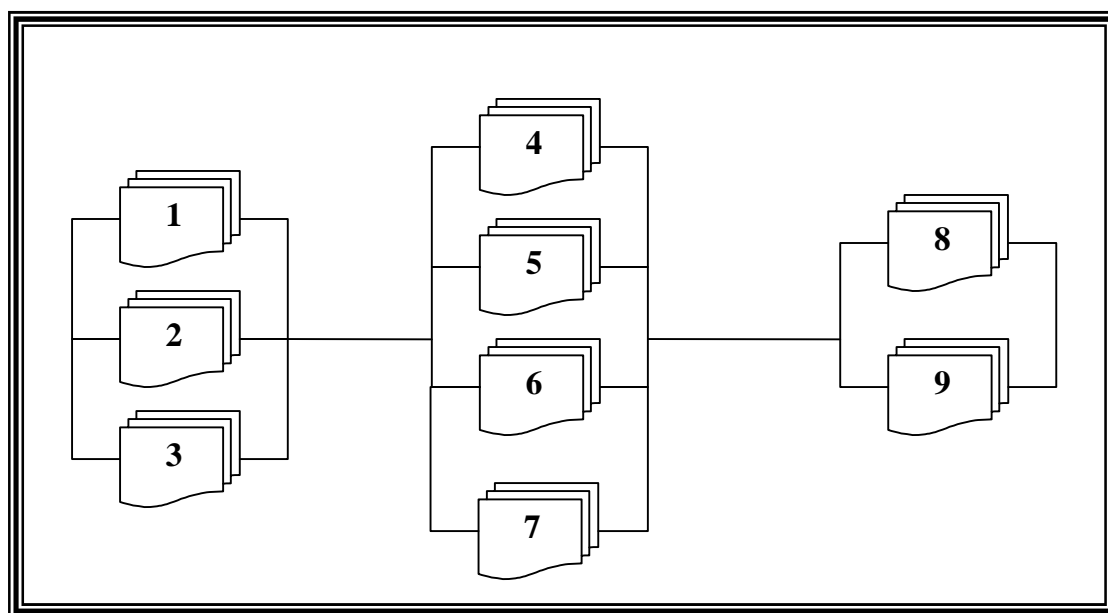


Figure 6. 2 Reliability System with Nine Components

Component No.	Reliability											
	0.001	0.5	0.55	0.6	0.65	0.7	0.75	0.8	0.85	0.9	0.95	0.99
1	0	4.05	16.3	40.4	67.35	95.7	135.75	186.05	251.05	339.8	440.45	598
2	0	3.65	17.75	36	59.35	78.5	169.6	224.45	303.8	391.95	505.3	654
3	0	9.1	22.35	44.45	71.55	105.1	148.85	198.1	276.75	374.25	496.8	634
4	0	4.35	14.1	29.15	50.45	78.2	117.55	170.9	248.55	347.9	663.75	609
5	0	3.15	10.8	31.95	52	183.25	222.1	278.8	350.3	434.15	539.3	699
6	0	7.8	22.85	43.85	70.8	101.45	143.7	202.05	276.7	370.2	495.15	629
7	0	8.75	18.8	42.8	72.05	106.25	151.2	210.95	289.95	370	482.75	637
8	0	5.45	16.45	36.45	60.7	191.2	230.75	282	354	449.5	572.75	703
9	0	2.05	7.67	23.87	101.9	128.81	164.35	207	271.25	362.8	480.95	623

Table 6.1 Cost and Reliability Data (Majety *et al.*, 1996)

In view of the given data set for the available components, the search space of the reliability optimisation problem consists of  $12^9$  possible solutions. The objective of the problem is to find the best combination of alternatives for the given series-parallel system, such that the system reliability is 85% and the total cost of the alternatives selected to compose the system is at a minimum. The objective function in equation (6.7), can be modelled for this example problem by assuming the value of ‘ $K$ ’ equal to zero and finding an optimal combination of alternatives for ‘ $Q$ ’ such that the total loss, ‘ $T_L$ ’ is minimum for the target level of reliability (85%).

Mathematically, the statement of the problem can be expressed as:

$$Opt(R_s) = \xi \left( R_{1(\gamma_{1,1} \dots \gamma_{1,12})}, R_{2(\gamma_{2,1} \dots \gamma_{2,12})}, \dots, R_{9(\gamma_{9,1} \dots \gamma_{9,12})} \right) = 85\% \quad (6.8)$$

by minimising the total cost function,

$$T_L = Min \left( \sum_i^9 c_i \right) \quad (6.9)$$

### 6.3.1 Application of the Optimisation Algorithm

The global optimum of this problem is actually known from enumeration search, which takes approximately six hours of computer processing time, as reported in the published paper of Majety et. al. (1996) and the best result they found for this particular problem was 533.90 by using only 20,000 out of  $12^9$  possible solutions, in 30 different independent runs. This is a very good result, within 6.65% of the global optimum solution of 500.60, obtained by enumeration. The optimisation algorithm introduced in this research, however, calculates an optimal solution of 506.70 units. This is only 1.22% within the global optimum solution obtained by enumeration method. Also, the OA used only 4469 solutions from the total search space before



converging to the optimal result. Additionally, there are many suboptimal values calculated by the OA which satisfy the minimum reliability requirement of 85%, as detailed in Table 6.2.

Results	Total Loss	System Reliability	Best Configurations		
1	506.70	85.02%	3-6-5	3-4-2-3	5-8
2	508.03	85.02%	3-6-5	4-2-3-3	5-8
3	508.85	85.05%	4-6-4	5-3-2-2	5-8
4	509.50	85.03%	5-6-4	3-3-2-3	5-8
5	510.45	85.01%	2-6-4	3-3-2-3	5-9
6	509.85	85.17%	3-6-4	5-3-3-3	5-8
7	516.70	85.12%	2-6-5	3-5-2-2	5-8
8	517.10	85.02%	3-6-4	5-2-2-2	4-9
9	522.20	85.09%	3-6-4	5-2-3-3	3-9
10	523.80	85.20%	3-6-3	5-4-2-2	4-9
11	525.90	85.13%	3-6-4	5-4-2-2	3-9

**Table 6.2 Quick Snap-Shot of the Results from the OA**

This information can be used to understand the complexity of the optimisation problem driven by the variation in the system cost for the same level of system reliability with different combinations of system configurations. It is interesting to note that the configuration of the reliability system with global optimum (Table 6.3) is not much different from the configuration of the optimal solution found by the OA (Table 6.2). In the context of the optimisation problem discussed here, this observation can be used to further streamline the OA's optimum result since swapping the fourth and fifth components in the second sub-system will not change the system

reliability, however, it will decrease the total system cost hence making the result identical to the global optimal found by the enumeration method.

Results	Total Loss	System Reliability	Best Configurations		
From SA	533.9	85.03%	4-5-4	2-5-3-4	5-8
From Enumeration	500.6	85.02%	3-6-5	4-3-2-3	5-8

Table 6.3 Results from the Published Article

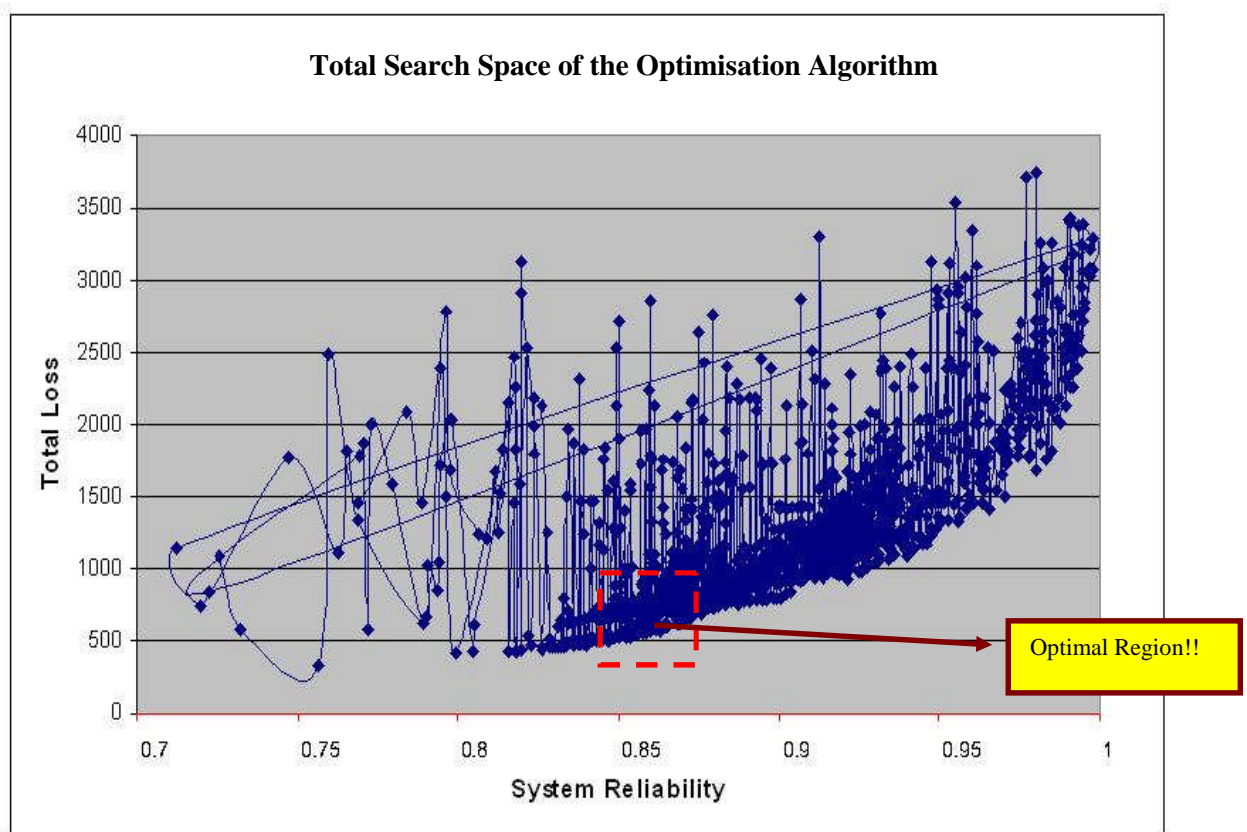


Figure 6. 3 Total Search Area Explored By the Optimisation Algorithm

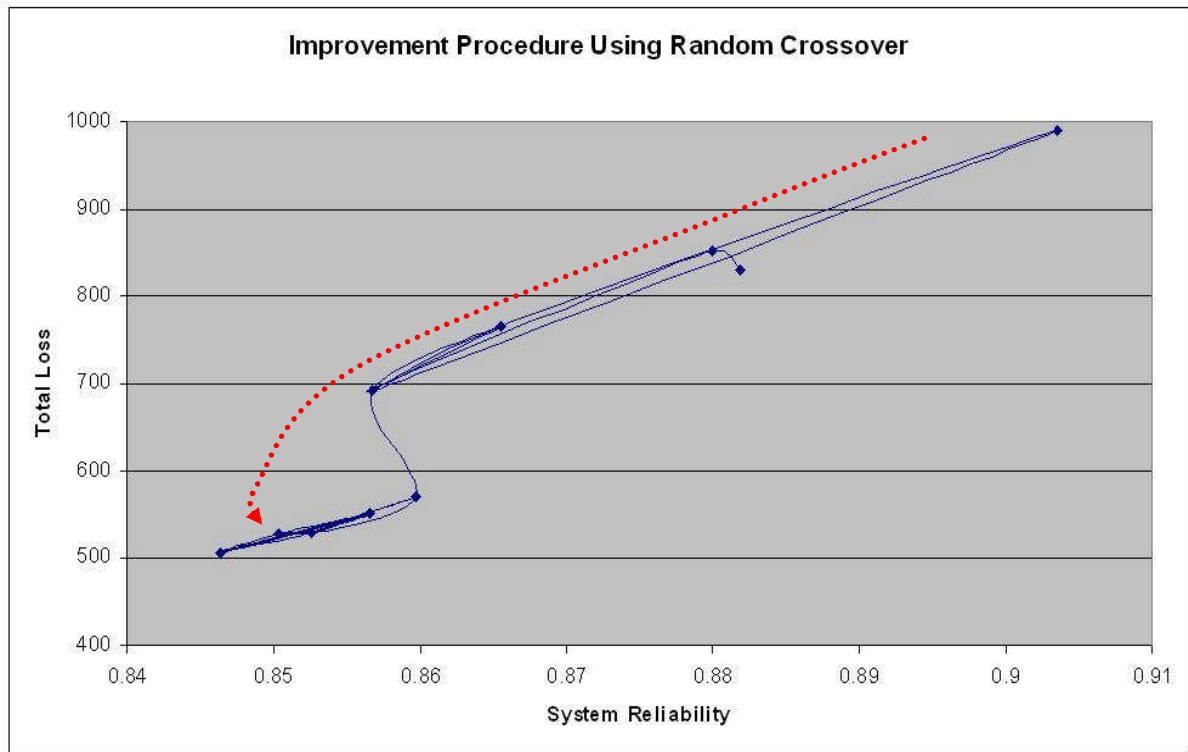


Figure 6. 4 Effect of the Improvement Procedure In Crossover Process

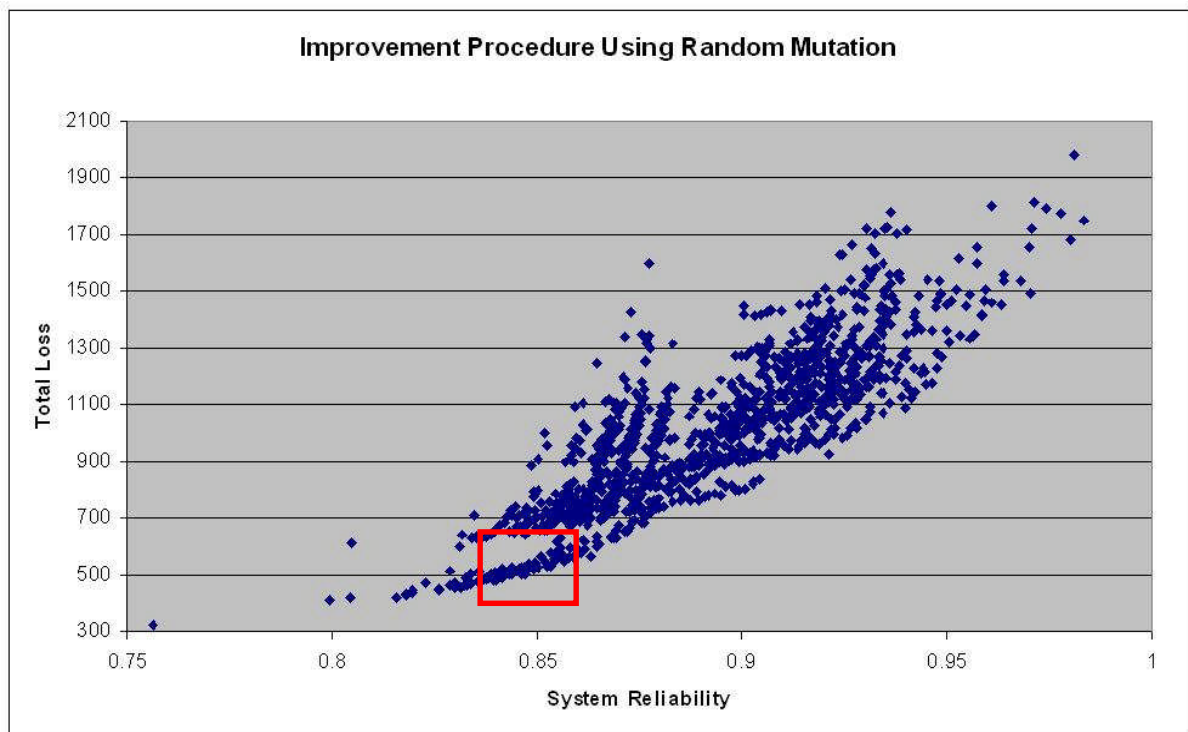
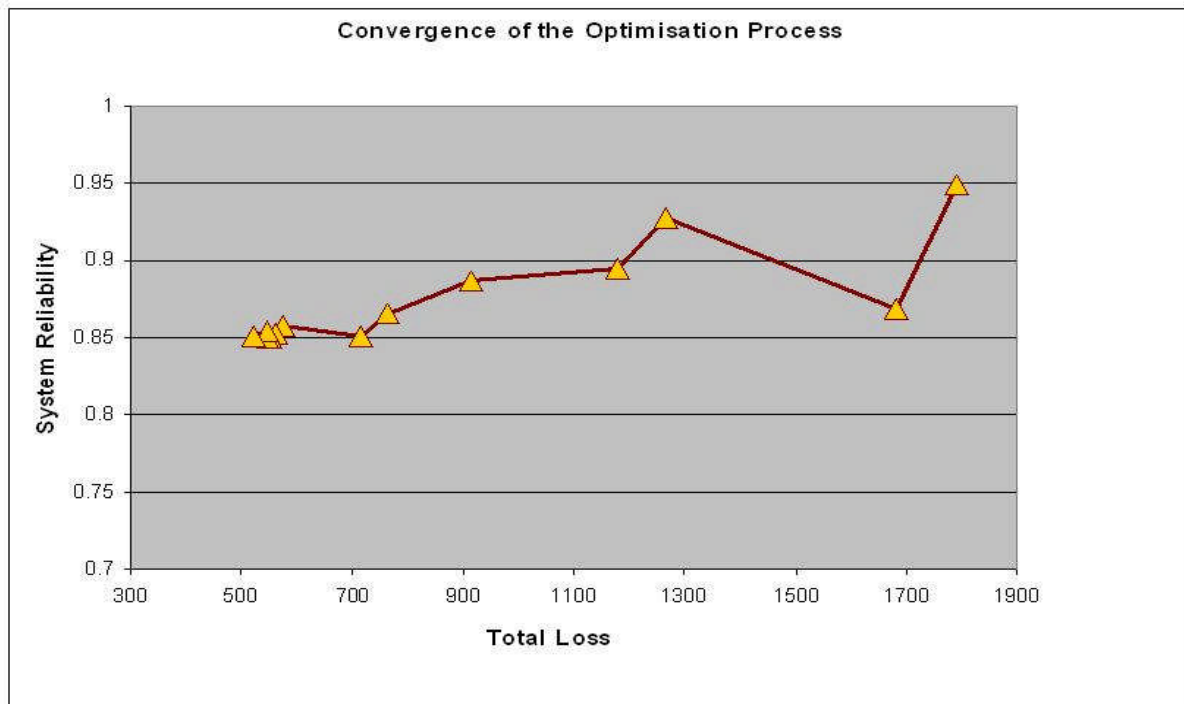
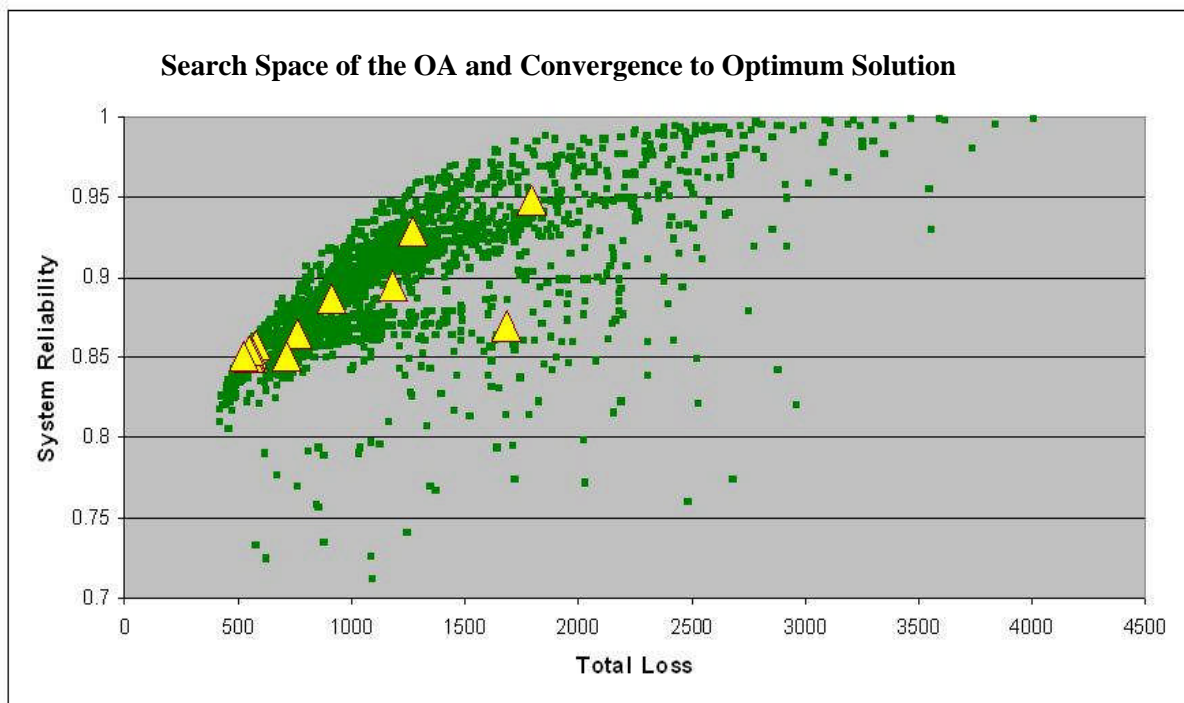


Figure 6. 5 Area Search by the Improvement Procedure Based On Random Mutations



**Figure 6. 6 Convergence of the Optimisation Toward Optimum Solution Using Improvement Procedures**



**Figure 6. 7 Total Search Space And the Optimisation Process**

## 6.4 RISK-BASED RELIABILITY ALLOCATION

In this section, the application of the optimisation algorithm using the risk-based reliability allocation method is demonstrated on four example systems. These systems are characterised as ‘A’, ‘B’, ‘C’ and ‘D’; every system is comprised of nine components with twelve choices of alternatives each and possesses a unique configuration, detailed in Table 6.4 and graphically in Fig 6.9. The configurations of these systems are commonly known in the engineering world because of the embedded series-parallel and parallel-series connection among the sub-systems. The data set used for the optimisation process is identical to the one detailed in Table 6.1 with components generally possessing a monotonically increasing relationship between cost and reliability parameters (Fig. 6.8).

System No	Sub system 1	Sub system 2	Sub system 3
<b>A</b>	3-Components (in parallel)	4-Components (in parallel)	2-Components (in parallel)
<b>B</b>	2-Components (in parallel)	5-Components (in parallel)	2-Components (in parallel)
<b>C</b>	3-Components (in parallel)	3-Components (in parallel)	3-Components (in parallel)
<b>D</b>	2-Components (in parallel)	2-Components (in series) + 2-Components (in parallel)	2-Components (in series) + 1-Component (in parallel)

**Table 6.4 System Configuration of Four Example Systems**

The objective of the optimisation process is to allocate an optimal level of system reliability by selecting an appropriate combination of component alternatives from the given choices, such that the total loss from system failure, ‘ $T_L$ ’, is minimum by using the relationship specified in equation (6.7). The risk of failure amount, ‘ $K$ ’ is derived from equation (6.3) by using the probability of system failure, ‘ $p_f$ ’ and the cost

given failure, ' $\bar{C}$ ', which is assumed to be a fixed cost ' $\bar{C}_1$ ', (2500 units) plus the random replacement cost of each failed component. The latter is imagined to be 25% of the cost of each failed component and is accumulated together with ' $\bar{C}$ ', in every event of a system failure (equation 6.5) during Monte Carlo simulations. The probability of failure is computed from the system reliability which is a function of the individual reliabilities of all selected alternatives in a given combination (equation 6.6). The reliability investment, ' $Q$ ', towards risk reduction is the total cost of components selected for building the system and is computed from equation (6.4) by taking into account the cost of all individual alternatives selected in a given combination.

The optimal reliability allocation process is detailed for each of the four systems in the next sub-sections.

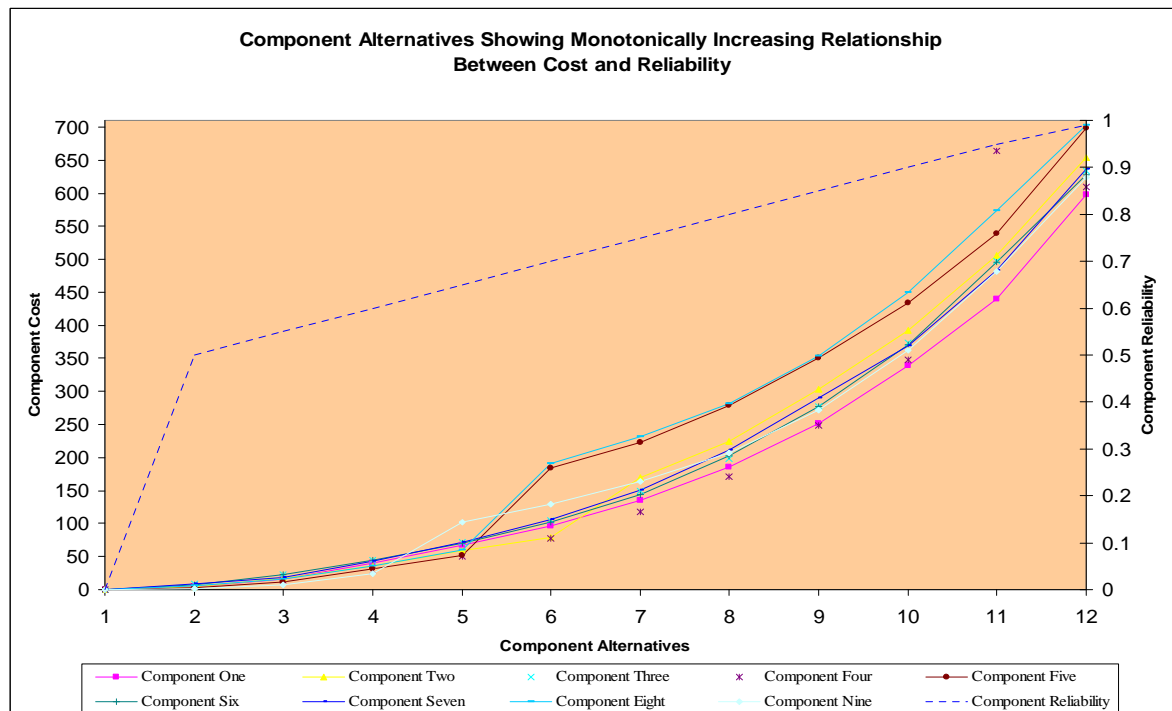


Figure 6. 8 Monotonically increasing relationship of cost-reliability from Table 6.1

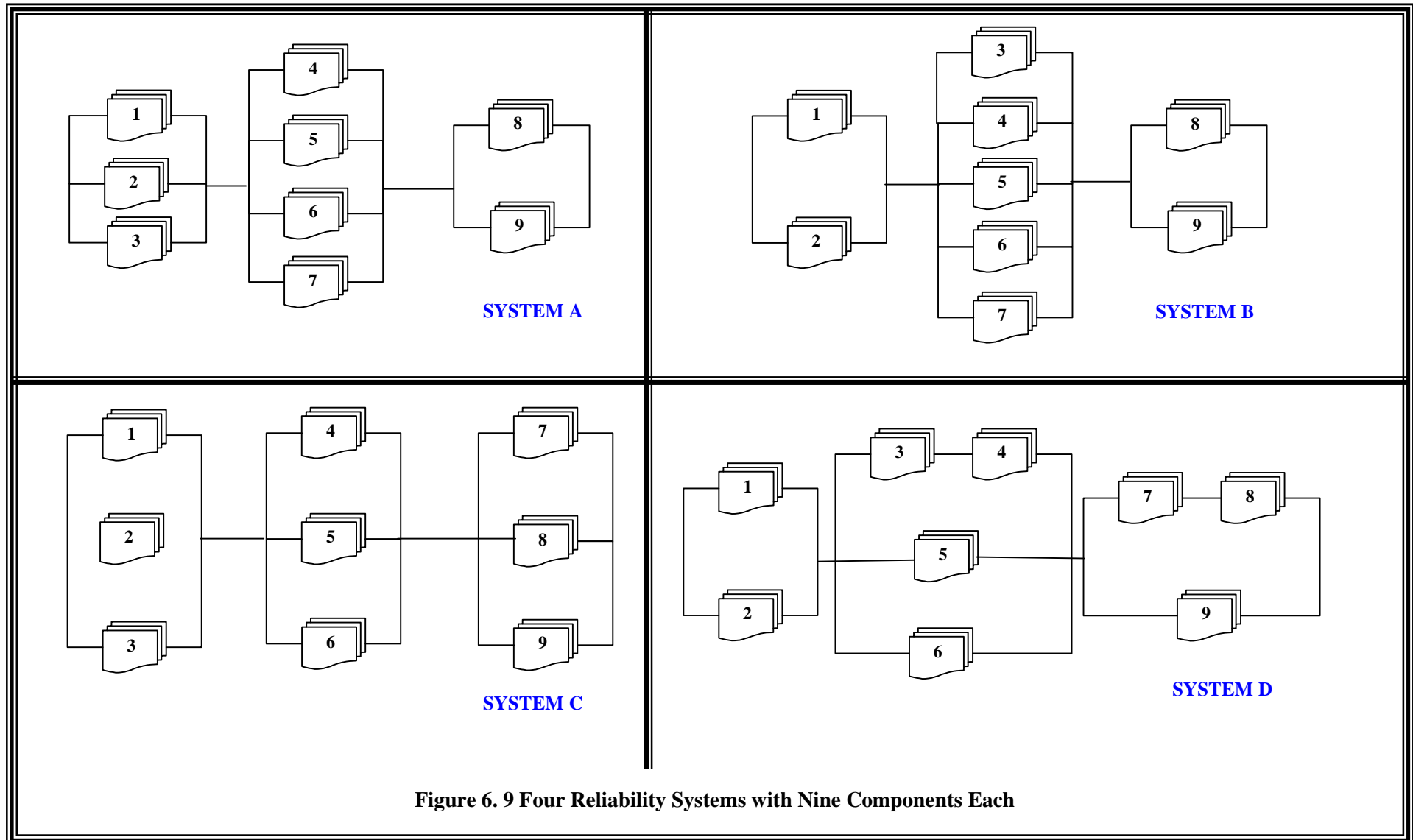


Figure 6. 9 Four Reliability Systems with Nine Components Each

### 6.4.1 Optimisation Results For System A

The results for system A are detailed in **Table 6.5**. In light of these results, it is found that the **reliability of 80.5% is optimum** for this system when using the cost-reliability data from **Table 6.1** since it is associated with the lowest amount of total loss, '**L<sub>Opt</sub>**' which is **877 units** found by the optimisation algorithm. Also interesting to note are the types of other observations detailed in the result table. The maximum value of the total loss or the loss function, '**L<sub>Max</sub>**' is **associated with the maximum level of system reliability, 'R<sub>Max</sub>'** and vice-versa. This is expected but not always the case as it will be demonstrated in the results from the other systems. The most found value, '**L<sub>Mode</sub>**', **of the loss function is very close to the optimum value, 'L<sub>Opt</sub>'**.

System Type	A
Optimum value of total loss - L <sub>Opt</sub>	<b>877</b>
Optimum value of system reliability	<b>80.5%</b>
Maximum value of total loss - L <sub>Max</sub>	3482
Maximum value of system reliability associated with L <sub>Max</sub>	99.9%
Mode value of total loss - L <sub>Mode</sub>	879
Mode value of System Reliability for L <sub>Mode</sub>	82.1%
Maximum value of system reliability - R <sub>Max</sub>	99.9%
Maximum value of total loss associated with R <sub>Max</sub>	3482
Average value of total loss - L <sub>Avg</sub>	1305
Standard Deviation of total loss	276
Coefficient of Variance total loss	21.2%
Average value of System Reliability	87.1%
Standard Deviation of System Reliability	4.9%
Coefficient of Variance System Reliability	5.6%

**Table 6.5 Optimisation Results for System A**



The snap-shot of the loss function optimisation process, showing the **hill climbing ability of the optimisation algorithm**, is highlighted in **Fig. 6.10** while the changes in the values of reliability investment and risk of failure with respect to system reliability and associated total loss is shown in **Fig. 6.11**.

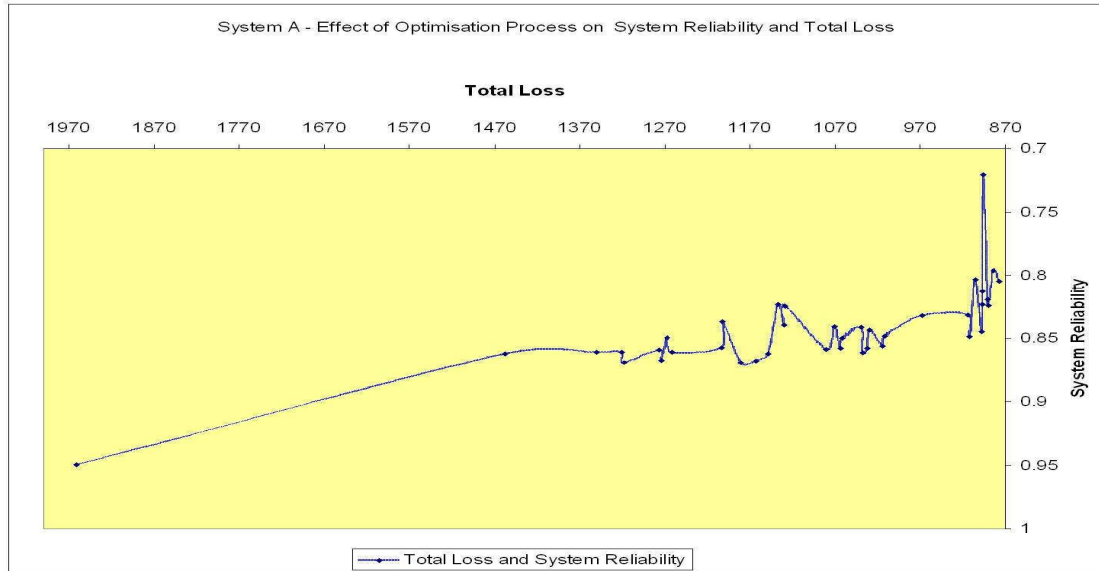


Figure 6. 10 Optimisation Process of System Reliability and Total Loss in System A

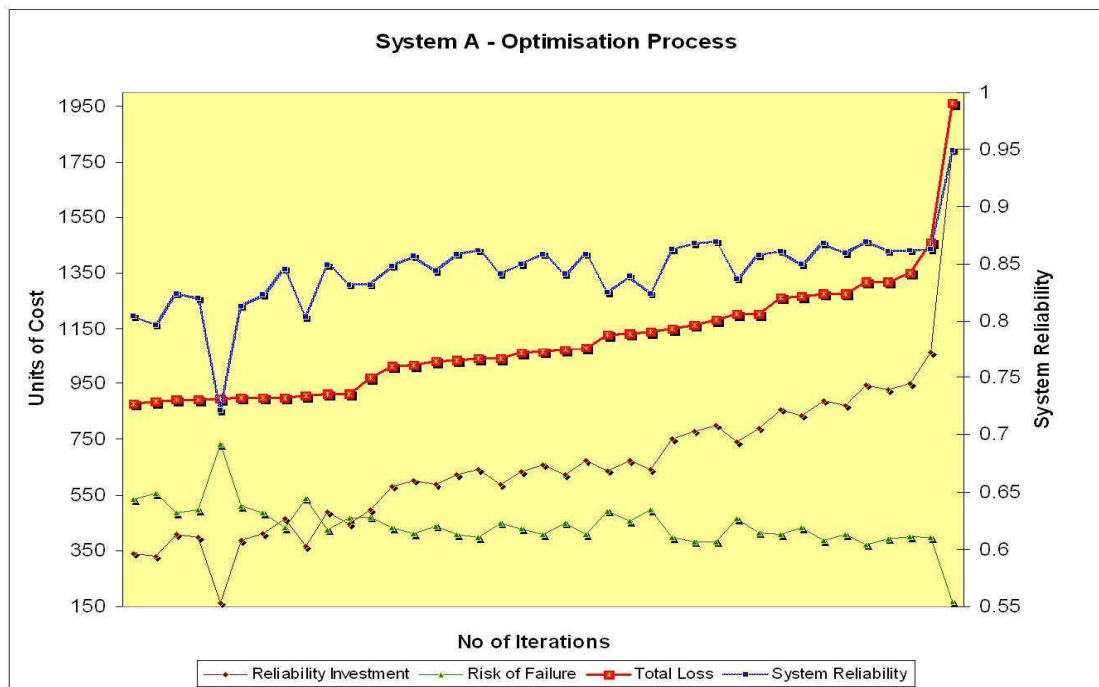


Figure 6. 11 Total Effect of Optimisation Process on System A

The **improvement processes** introduced in the optimisation algorithm (Chapter 5) using the crossover and mutation operations, **enhance the capability of the optimisation algorithm to converge towards optimal results while efficiently exploiting the search space**. These effects of the improvement processes are demonstrated in **Fig. 6.12 & 6.13** for crossover and mutation processes, respectively.

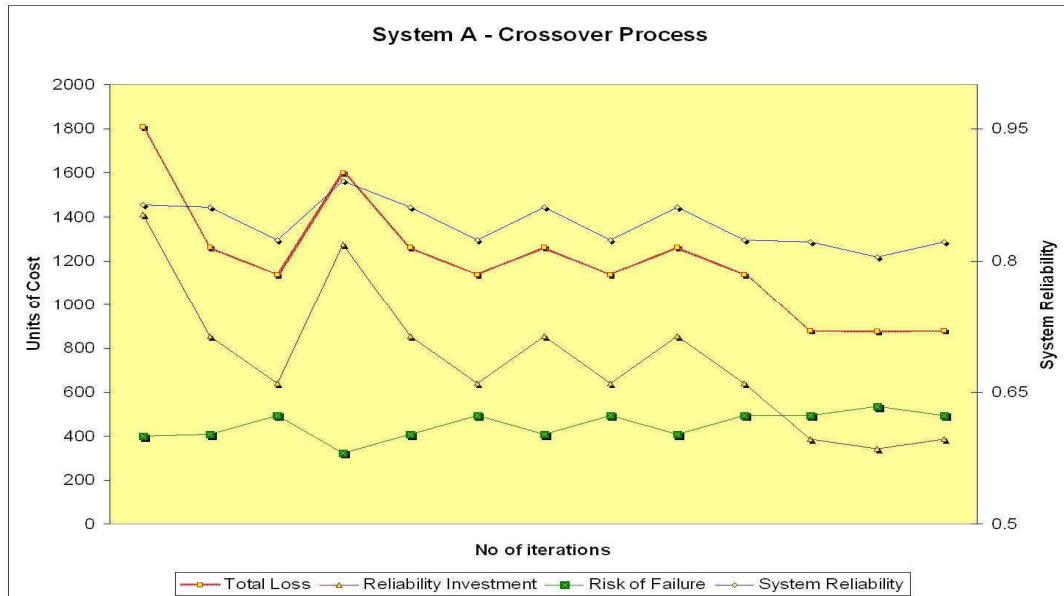


Figure 6. 12 Crossover Process of System A

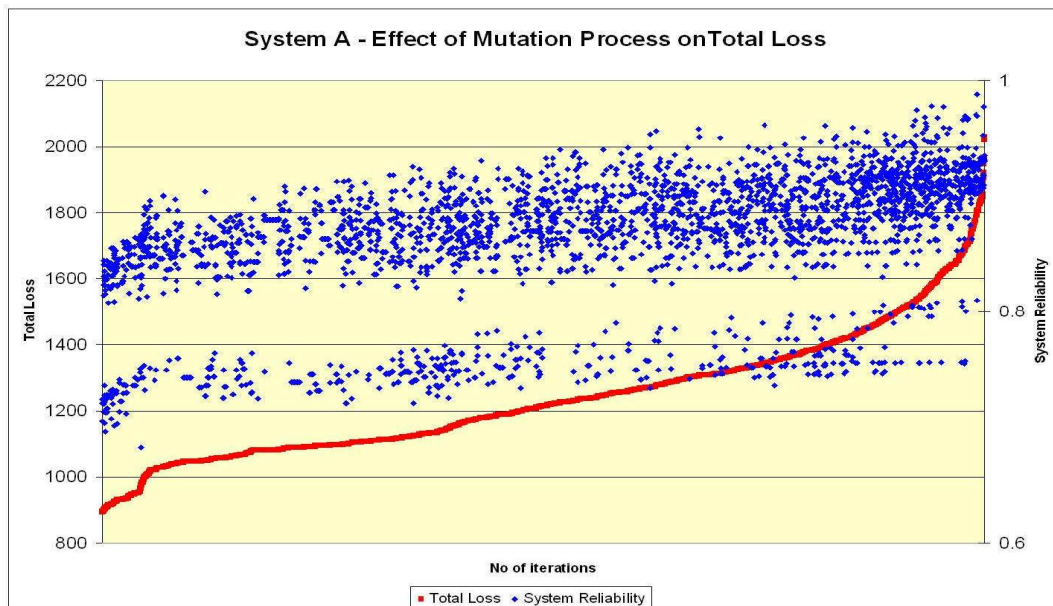
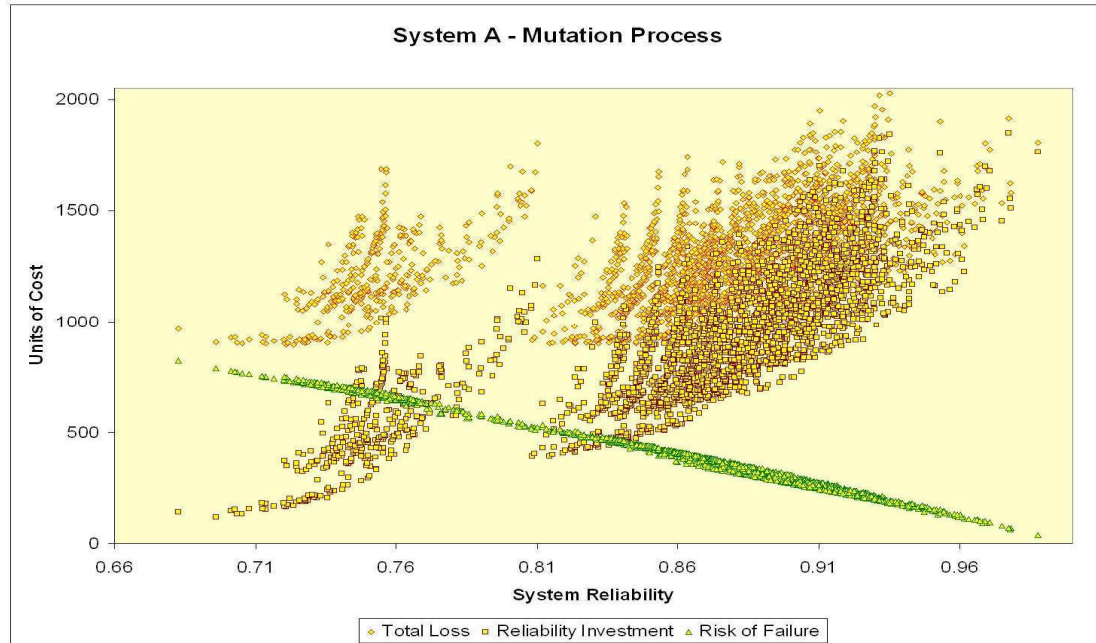
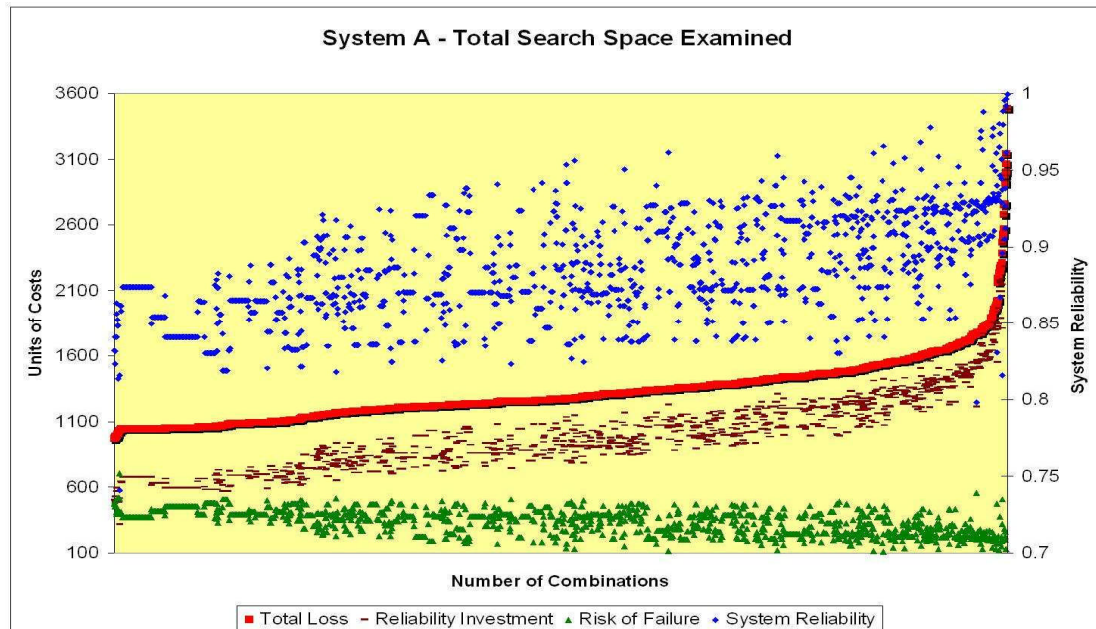


Figure 6. 13 Effect of Mutation Process on Total Loss in System A

The region of the search space **exploited** by the mutation process with the view of reliability investment and risk of failure is shown in **Fig. 6.14**. Also, the **total** search space **examined** by the optimisation algorithm in accordance with the objective function from equation (6.7) is detailed explicitly in **Fig. 6.15**.



**Figure 6. 14** Region of the search space examined by the Mutation Process in System A



**Figure 6. 15** Total Search Space examined by the optimisation algorithm for System A

While **Table 6.5** highlights the optimum result for system A, the optimisation algorithm, being a population based search method, also produces a **list of other sub-optimal solutions** which can be of great interest to a reliability analyst, see **Table 6.6**. The results detailed in this table are helpful in studying **different levels of reliability for respective values of the total loss function** along with the important information concerning the **actual configuration of the system** by illustrating the precise description of the chosen alternatives of all components. A quick overview of these observations can be categorised in the following points:

***I) Decreasing Amount of Total Loss with Increasing Level of System Reliability***

It is generally expected for an engineering system to have a higher reliability level which seems appropriate as seen in the results table where higher reliability has lower values of total loss (results 1- 3 compared to result 4). However, it can also be seen from the same table that **increasing the reliability further produces a drastic increase in the total loss** (e.g. results 15 onwards).

***II) Increasing Amount of Total Loss with Constant and/or Increasing Level of System Reliability***

The amount of total loss can vary tremendously for the same level of system reliability when an inappropriate combination of components is selected. For example, *results 3, 11 and 13*: the **total loss increases around 27%** for the system reliability of 82%, *results 7 & 14*: the **total loss increases 24%** for system reliability of 83%, *results 5, 9, 10 & 12*: the **total loss increases 25%** for the same system reliability of 84% and so on. Similarly, it can be seen from the result table that **higher levels of system reliabilities are associated with larger values of total loss function**.

***III) Increasing Amount of Total Loss with Decreasing Level of System Reliability***

It is understandable to expect higher losses with decreasing level of system reliability as shown in Table 6.6 where results 2, 4, 6, 9, 11, 13, 16 & 18 shown **increases in loss values with relative decreases in the system reliability**. However, this is not

always the case as stated previously in points - **I & II** above, where the loss also increased with system reliability.

#### ***IV) Decreasing Amount of Total Loss with Decreasing Level of System Reliability***

Similar to point **III**, it can be surmised that the **loss function also decreases with the diminution of system reliability** as seen from the relative positions of results **19, 16, 14, 13, 11, 7, 6, 4** and **2** by moving in the upward direction of **Table 6.6**.

In view of these observations, the **importance** of the **improvement procedures** designed within the optimisation process (Chapter 5, section 5.3.5) appears more **significant for risk based reliability optimisation problems** involving a large choice of component alternatives. Since these procedures operate on one gene (component) at a time for applying the genetic operations such as crossover and mutation and then gradually increasing to two and three components later in the optimisation cycle, the chances for evaluating good solutions with better fitness can be carefully organised. This process is shown in Table 6.6, when reading it backwards. For example, the total **loss amount in result no.18 is improved by 7%** from the amount in **result no. 19** (found by the random search) through first stage mutation operation at position seven. The **result no. 17** is an improved version of **result no. 18** obtained by performing a **two stage mutation operation** (Chapter 5, section 5.3.4) at position **nos. 2 and 7** (mutation sites). Similarly, **result no.16** is obtained by the combination of a **two stage crossover operation** (Chapter 5, section 5.3.3) at **position 2 and 6** from results **18 & 17** and then a **single stage mutation operation** at **position 7** (result no. 16 could have also been found by a **three stage mutation** alone on these positions also), and so on.

The above optimisation process is **repeated** for **systems B, C and D** in the next sections and only the statistical results are shown for each of the three systems. The graphical results, similar to **Fig. 6.10 – 6.15**, showing the optimisation process by improving the local result, the effects of optimisation process on total loss, effects of improvement procedures using genetic operators such as crossover and mutation and

the total area search by the optimisation algorithm for all three systems are presented in Appendix II, III and IV.

No.	Total Loss	System Reliability	Reliability Investment	Risk of Failure	Sub-System One			Sub-System Two				Sub-System Three	
<b>1</b>	<b>877</b>	<b>80.5%</b>	<b>342</b>	<b>535</b>	<b>3</b>	<b>4</b>	<b>3</b>	<b>3</b>	<b>2</b>	<b>2</b>	<b>3</b>	<b>3</b>	<b>8</b>
2	884	79.7%	330	554	2	4	3	3	2	2	3	3	8
3	890	82.4%	406	484	3	4	3	3	2	2	3	3	9
4	896	72.1%	164	732	3	4	3	4	2	2	2	3	4
5	898	84.5%	464	434	3	6	3	4	2	2	3	3	9
6	906	80.4%	366	540	2	4	3	5	2	2	3	3	8
7	968	83.2%	497	471	3	4	2	4	2	4	5	3	9
8	1015	85.7%	603	411	3	4	4	4	2	7	4	3	9
9	1030	84.4%	589	441	3	4	3	2	2	2	8	3	9
10	1039	84.1%	589	450	3	4	2	5	2	7	4	3	9
11	1130	82.5%	637	492	2	4	3	5	2	2	9	3	8
12	1130	84.0%	674	457	4	4	3	5	2	2	9	3	8
13	1137	82.4%	641	496	4	2	3	5	2	2	9	3	8
14	1203	83.7%	739	463	3	3	2	5	2	7	8	3	9
15	1275	86.8%	887	387	5	4	2	5	2	7	9	3	9
16	1277	85.9%	869	408	5	3	2	5	2	7	9	3	9
17	1318	87.0%	944	374	5	4	2	5	2	2	11	3	9
18	1351	86.1%	949	402	5	3	2	5	2	7	10	3	9
19	1458	86.3%	1062	396	5	3	2	5	2	7	11	3	9
<b>20</b>	<b>1961</b>	<b>95.0%</b>	<b>1792</b>	<b>168</b>	<b>3</b>	<b>5</b>	<b>10</b>	<b>9</b>	<b>5</b>	<b>2</b>	<b>11</b>	<b>10</b>	<b>5</b>

**Table 6.6 List of Various Results for System A**



## 6.4.2 Optimisation Results For System B

The results for **system B** are detailed in **Table 6.7**. In light of these results, it is found that the **reliability** of **73.7%** associated with **992 units** towards the '**L<sub>Opt</sub>**' is **optimum** for this system when using with the cost-reliability data from **Table 6.1**. Also interesting to note are the types of other observations detailed in the result table. Unlike system A, **the maximum value of the total loss or the loss function, 'L<sub>Max</sub>'** found in the optimisation process, **is not associated with the maximum level of system reliability, 'R<sub>Max</sub>'**. The latter is associated with a lower amount of total loss (2979 units) which is still **three times the 'L<sub>Opt</sub>'** amount for being **35% more than the optimum reliability**. The most found value, '**L<sub>Mode</sub>**', of the loss function is although higher than the optimum value, '**L<sub>Opt</sub>**', it is around **8% better than the optimum reliability**.

System Type	B
Optimum value of total loss - L <sub>Opt</sub>	992
Optimum value of system reliability	73.7%
Maximum value of total loss - L <sub>Max</sub>	3683
Maximum value of system reliability associated with L <sub>Max</sub>	93.0%
Mode value of total loss - L <sub>Mode</sub>	1008
Mode value of System Reliability for L <sub>Mode</sub>	79.2%
Maximum value of system reliability - R <sub>Max</sub>	99.7%
Maximum value of total loss associated with R <sub>Max</sub>	2979
Average value of total loss - L <sub>Avg</sub>	1436
Standard Deviation of total loss	263
Coefficient of Variance total loss	18.4%
Average value of System Reliability	81.2%
Standard Deviation of System Reliability	6.3%
Coefficient of Variance System Reliability	7.8%

**Table 6.7 Optimisation Results for System B**

The list of other sub-optimal results is enclosed in Table 6.8 which can be used for the comparative analysis similar to section 6.3.1. and are discussed in the points below:

***I) Decreasing Amount of Total Loss with Increasing Level of System Reliability***

It can be seen from the result table above that some combinations of components produce **higher reliability with lower values of total loss**. For instance, results **18, 14, 13, 11, 9, 8, 7, 5, 4 & 3** relatively **increase in reliability** but **decrease in total loss**. However, it can also be seen from the same table that **increasing the reliability** (with respect to 'L<sub>Opt</sub>') **through inappropriate selection of component alternatives** can also cause a **significant increase in the total loss** (e.g. results 16 onwards).

***II) Increasing Amount of Total Loss with Constant and/or Increasing Level of System Reliability***

The amount of total loss **varies** enormously for the **same level of system reliability** when an **inappropriate combination of components** is selected. For example, *results 3, 11 and 20*: the total loss **increases over two and a quarter times** for the **same level of system reliability (81%)** *results 13, 17 & 18*: the total loss **increases 21%** for system reliability of **85%** and so on. Similarly, it can be seen that results such as, **2, 3, 7, 11, 13, 16, 17 & 20** **increase in reliability** with a corresponding **increase in total loss values**.

***III) Increasing Amount of Total Loss with Decreasing Level of System Reliability***

The results **4, 6, 8, 9, 10, 12, 14, 15 & 19** shown **increases in loss values** with relative **decrease in system reliability**. However, this is **not** always the case as stated in the points above, where the **loss also increased with system reliability**.

***IV) Decreasing Amount of Total Loss with Decreasing Level of System Reliability***

Similar to point **III**, it can be surmised that the loss function also **decreases** with the **diminution** of system reliability as seen from the relative positions of results **19, 16, 15, 12, 10, 6 and 2** by looking upwards in **Table 6.8**.



No.	Total Loss	System Reliability	Reliability Investment	Risk of Failure	Sub-System One		Sub-System Two					Sub-System Three	
<b>1</b>	<b>992</b>	<b>73.7%</b>	<b>283</b>	<b>708</b>	<b>5</b>	<b>6</b>	<b>2</b>	<b>2</b>	<b>4</b>	<b>3</b>	<b>2</b>	<b>4</b>	<b>4</b>
2	1008	80.2%	450	558	7	5	3	2	3	2	2	4	7
3	1019	81.6%	500	520	5	6	2	2	4	3	2	2	9
4	1037	76.8%	394	643	3	6	2	2	4	4	2	4	7
5	1133	72.0%	366	767	3	4	4	2	4	4	2	3	7
6	1161	70.0%	348	813	3	3	4	2	4	4	2	3	7
7	1193	85.3%	772	420	11	3	4	2	4	4	2	3	7
8	1234	79.5%	641	593	8	4	2	2	6	4	2	2	7
9	1259	77.8%	623	636	5	6	2	2	6	6	2	2	7
10	1267	75.4%	572	696	3	6	2	2	6	6	2	2	7
11	1279	81.7%	741	537	8	6	2	2	6	6	2	2	7
12	1291	74.0%	559	731	2	6	2	2	6	6	2	2	7
13	1343	85.7%	924	419	11	3	4	2	6	4	2	3	7
14	1387	74.3%	660	727	2	6	2	2	6	8	2	2	7
15	1456	69.8%	617	839	2	4	2	2	6	8	2	2	7
16	1457	74.4%	735	723	2	6	2	2	6	9	2	2	7
17	1590	85.7%	1171	419	11	6	2	2	6	9	2	2	7
18	1624	85.9%	1206	418	11	6	4	2	6	9	2	2	7
19	1658	79.0%	1062	596	5	3	2	5	2	7	11	3	9
<b>20</b>	<b>2327</b>	<b>81.8%</b>	<b>1792</b>	<b>535</b>	<b>3</b>	<b>5</b>	<b>10</b>	<b>9</b>	<b>5</b>	<b>2</b>	<b>11</b>	<b>10</b>	<b>5</b>

Table 6.8 List of Various Results for System B

### 6.4.3 Optimisation Results For System C

The results for **system C** are detailed in **Table 6.9**. In light of these results, it is found that the **system reliability of 79.8%** is **optimal** for this system since it is associated with the lowest value of the total loss, **782 units**, when using with the cost-reliability data from **Table 6.1**. Unlike **system B** and similar to **system A**, the maximum value of the total loss function, '**L<sub>Max</sub>**' found in the optimisation process, **is associated with the maximum level of system reliability, 'R<sub>Max</sub>'**. The former is approximately four and a half times **more than the 'L<sub>Opt</sub>'** amount with **25% more** associated reliability than the **optimum reliability**. The most found value, '**L<sub>Mode</sub>**', of the loss function is **25% higher than the 'L<sub>Opt</sub>'** with around **8% higher** associated reliability value than the **optimum**.

System Type	C
Optimum value of total loss - L <sub>Opt</sub>	<b>782</b>
Optimum value of system reliability	<b>79.8%</b>
Maximum value of total loss - L <sub>Max</sub>	3482
Maximum value of system reliability associated with L <sub>Max</sub>	99.9%
Mode value of total loss - L <sub>Mode</sub>	971
Mode value of System Reliability for L <sub>Mode</sub>	85.9%
Maximum value of system reliability - R <sub>Max</sub>	99.9%
Maximum value of total loss associated with R <sub>Max</sub>	3482
Average value of total loss - L <sub>Avg</sub>	1175
Standard Deviation of total loss	295
Coefficient of Variance total loss	25.2%
Average value of System Reliability	86.4%
Standard Deviation of System Reliability	3.6%
Coefficient of Variance System Reliability	4.1%

**Table 6.9 Optimisation Results for System C**

The list of other sub-optimal results is enclosed in Table 6.10 which can be used for a similar kind of analysis as detailed in section 6.3.1 and 6.3.2 for systems A and B.

NO.	Total Loss	System Reliability	Reliability Investment	Risk of Failure	Sub-System One			Sub-System Two			Sub-System Three		
1	782	79.8%	242	540	3	4	3	4	3	4	4	3	4
2	798	81.3%	293	505	3	6	2	5	2	3	5	3	4
3	825	79.9%	285	540	3	4	3	3	2	7	2	3	4
4	909	86.5%	529	379	3	6	3	3	2	7	8	3	4
5	916	85.9%	520	396	3	6	3	2	2	7	8	3	4
6	961	86.6%	582	379	3	6	3	5	3	4	5	3	9
7	968	85.7%	568	400	3	6	2	6	2	3	5	3	9
8	969	86.0%	574	395	3	6	3	5	2	4	5	3	9
9	970	83.5%	515	455	3	3	4	5	2	3	5	3	9
10	974	83.6%	520	453	3	3	3	6	2	3	5	3	9
11	974	86.9%	602	372	3	6	3	6	2	4	5	3	9
12	979	81.5%	475	504	3	4	3	3	2	3	5	3	9
13	981	87.4%	624	356	3	4	4	5	2	7	9	3	4
14	989	85.7%	589	400	3	4	2	5	2	7	9	3	4
15	1000	85.6%	595	404	3	4	3	3	2	7	5	3	9
16	1001	86.1%	610	391	3	4	3	4	2	7	5	3	9
17	1146	88.9%	833	313	5	4	2	5	2	7	11	3	4
18	1378	90.1%	1093	285	5	4	3	5	2	7	11	3	9
19	1388	88.5%	1062	327	5	3	2	5	2	7	11	3	9
20	1920	96.0%	1792	128	3	5	10	9	5	2	11	10	5

**Table 6.10 List of Various Results for System C**

### 6.4.4 Optimisation Results For System D

The results for **system D** are detailed in **Table 6.11**. In light of these results, it is found that the **system reliability of 74.3%** is **optimal** for this system since it is associated with the **lowest value of the total loss, 1356 units**. Like system B, the maximum value of the total loss function, ' $L_{Max}$ ' found in the optimisation process, is **higher compared to the corresponding amount associated with the ' $R_{Max}$ '**. The total loss amount of ' $R_{Max}$ ' is **approximately just under two and a half times more than the ' $L_{Opt}$ ' amount with 33% more associated reliability than the optimum reliability**. The most found value, ' $L_{Mode}$ ', of the loss function is **11% higher than the ' $L_{Opt}$ ' with only 3% improvement in associated reliability value than the optimum**.

System Type	D
Optimum value of total loss - $L_{Opt}$	<b>1356</b>
Optimum value of system reliability	<b>74.3%</b>
Maximum value of total loss - $L_{Max}$	3964
Maximum value of system reliability associated with $L_{Max}$	97.0%
Mode value of total loss - $L_{Mode}$	1511
Mode value of System Reliability for $L_{Mode}$	77.1%
Maximum value of system reliability - $R_{Max}$	98.9%
Maximum value of total loss associated with $R_{Max}$	3166
Average value of total loss - $L_{Avg}$	1647
Standard Deviation of total loss	249
Coefficient of Variance total loss	15.1%
Average value of System Reliability	75.9%
Standard Deviation of System Reliability	4.9%
Coefficient of Variance System Reliability	6.4%

**Table 6.11 Optimisation Results for System D**

The list of other sub-optimal results is enclosed in Table 6.12 which can be used for the similar kind of analysis as detailed in section 7.3.1 & 7.3.2 for systems A and B.

NO.	Total Loss	System Reliability	Reliability Investment	Risk of Failure	Sub-System One		Sub-System Two				Sub-System Three		
					5	6	2	2	3	7	3	3	9
1	1356	74.3%	620	736	5	6	2	2	3	7	3	3	9
2	1362	66.4%	434	928	3	6	2	2	2	6	2	2	8
3	1364	69.8%	513	850	6	6	2	2	2	6	2	2	8
4	1368	75.5%	669	699	6	6	2	2	2	6	2	2	10
5	1377	73.8%	639	738	6	6	2	2	2	5	2	2	10
6	1379	70.3%	559	820	3	6	2	2	2	5	2	2	10
7	1399	74.3%	659	740	5	6	2	5	2	7	3	3	9
8	1411	75.2%	694	718	6	6	2	2	3	6	6	3	9
9	1426	66.7%	517	909	3	4	2	2	2	5	2	2	10
10	1448	64.9%	498	950	3	3	2	2	2	5	2	2	10
11	1451	74.9%	725	727	5	6	2	4	2	7	6	3	9
12	1478	63.0%	484	994	3	2	2	2	2	5	2	2	10
13	1521	66.9%	616	906	3	2	2	2	2	8	2	2	10
14	1535	87.3%	1158	377	11	2	2	2	2	8	2	2	11
15	1560	87.8%	1193	366	11	2	4	2	2	8	2	2	11
16	1592	89.5%	1268	324	11	2	4	2	2	9	2	2	11
17	1627	76.1%	930	697	5	6	2	5	2	7	9	3	9
18	1704	76.5%	1026	678	11	6	4	2	2	9	2	2	7
19	1843	72.4%	1062	781	5	3	2	5	2	7	11	3	9
<b>20</b>	<b>2467</b>	<b>77.3%</b>	<b>1792</b>	<b>675</b>	<b>3</b>	<b>5</b>	<b>10</b>	<b>9</b>	<b>5</b>	<b>2</b>	<b>11</b>	<b>10</b>	<b>5</b>

**Table 6.12 List of Various Results for System D**

---

## 6.5 PARAMETRIC STUDY USING THE RISK BASED RELIABILITY ALLOCATION METHOD AND OPTIMISATION ALGORITHM

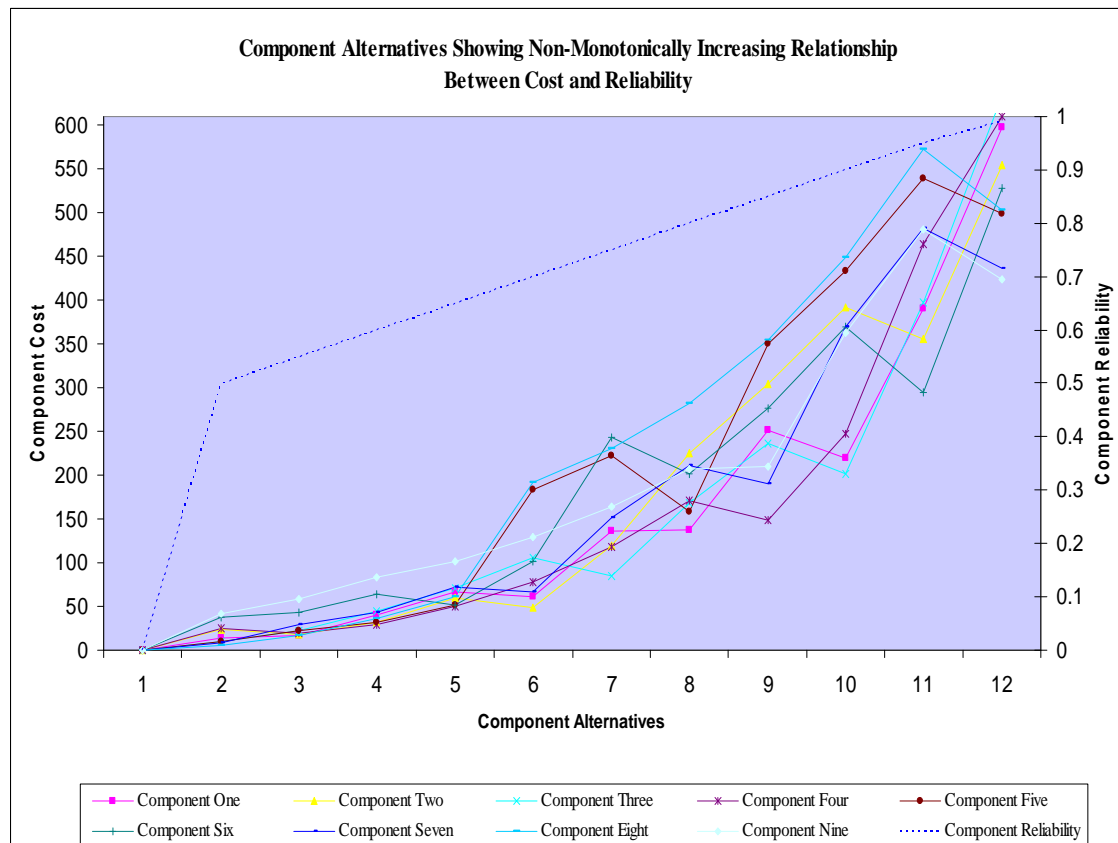
The optimisation algorithm along with the risk-based reliability allocation method is applied on all four systems detailed in Table 6.4 and in Fig 6.9 with two variations in the overall optimisation process from section 6.3. These variations are introduced in order to perform parametric studies on all four systems individually and as a whole.

The first variation involves using a different choice of a discrete data set of components, depicted in Table 6.13, and allocating optimal system reliability with minimum total loss on each of the four systems. The new data set is different because it exhibits a non-monotonically increasing relationship between cost and reliability among all alternatives of the nine components, which is shown in Fig 6.16. The two sets of data are used in order to expound the high level of complexity in selecting an optimal combination of components, from the large choice of available alternatives with the aim of efficiently minimising the loss function from equation (6.7). The level of this already high complexity increases further when the components choice exhibits a non-monotonically increasing relationship between cost-reliability parameters, as the results will demonstrate later in the chapter.

The second variation in the optimisation process is introduced by means of using a different value of the cost of failure amount, ' $\bar{C}$ ', associated with a given system failure; it is assumed to be a fixed cost 'C2' of 1000 units, which is a reduced value from section 6.3. Using this amount, the risk based reliability allocation is performed on all four systems by utilising the data from Table 6.1. This type of parametric study is carried out in the hope that it will benefit engineering systems associated with lower cost of system failure by closely evaluating the effects of the risk based reliability allocation process on individual systems, when a large choice of component data is at hand.

Component No.	Reliability											
	0.001	0.5	0.55	0.6	0.65	0.7	0.75	0.8	0.85	0.9	0.95	0.99
1	0	14.05	16.3	40.4	67.35	60.7	135.75	137.75	251.05	219.8	390.45	598
2	0	23.65	17.75	32	59.35	48.5	119.6	224.45	303.8	391.95	355.3	554
3	0	9.1	22.35	44.45	71.55	105.1	84.85	168.1	236.75	201.25	396.8	634
4	0	24.35	19.1	29.15	50.45	78.2	117.55	170.9	148.55	247.9	463.75	609
5	0	9.15	21.8	31.95	52	183.25	222.1	158.8	350.3	434.15	539.3	499
6	0	37.8	42.85	63.85	50.8	101.45	243.7	202.05	276.7	370.2	295.15	529
7	0	8.75	28.8	42.8	72.05	66.25	151.2	210.95	189.95	370	482.75	437
8	0	5.45	16.45	36.45	60.7	191.2	230.75	282	354	449.5	572.75	503
9	0	42.05	57.67	83.87	101.9	128.81	164.35	207	210.25	362.8	480.95	423

Table 6.13 Cost and Reliability Data with non-monotonically increasing values



**Figure 6.16 Monotonically Increasing Relationship of Cost-Reliability for Data in  
Table 6.13**

### 6.5.1 Optimisation Process Using Data Set Showing A Non-Monotonically Increasing Relationship Between Cost And Reliability

For each of the four systems, only the core optimisation results are detailed in this section. All other results including the graphical representation of the effects of optimisation algorithm on system reliability, total loss, reliability investment and risk of failure along with the list of sub-optimal results are shown in Appendix V – VIII.



### 6.5.1.1 Optimisation Results For System A

The results for **system A** are detailed in Table 6.14. In light of these results, it is found that the **reliability of 86.8% is optimum** for this system when using the cost-reliability data from Table 6.13 since it is associated with the lowest amount of total loss, '**L<sub>Opt</sub>**' which is **815 units** found by the optimisation algorithm. The maximum value of the total loss or the loss function, '**L<sub>Max</sub>**' is **slightly higher** than the corresponding value associated with the **maximum level of system reliability, 'R<sub>Max</sub>'** found by the algorithm. The most found value, '**L<sub>Mode</sub>**', of the loss function is very close to the optimum value, '**L<sub>Opt</sub>**' with a very **little difference** in the respective level of system reliabilities.

System Type	A
Optimum value of total loss - L <sub>Opt</sub>	<b>815</b>
Optimum value of system reliability	<b>86.8%</b>
Maximum value of total loss - L <sub>Max</sub>	3008
Maximum value of system reliability associated with L <sub>Max</sub>	99.0%
Mode value of total loss - L <sub>Mode</sub>	845
Mode value of System Reliability for L <sub>Mode</sub>	86.2%
Maximum value of system reliability - R <sub>Max</sub>	99.5%
Maximum value of total loss associated with R <sub>Max</sub>	2901
Average value of total loss - L <sub>Avg</sub>	1079
Standard Deviation of total loss	241
Coefficient of Variance total loss	22.3%
Average value of System Reliability	89.3%
Standard Deviation of System Reliability	3.5%
Coefficient of Variance System Reliability	3.9%

**Table 6.14 Optimisation Results for System A**

### 6.5.1.2 Optimisation Results For System B

The results for **system B** are detailed in Table 6.15. In light of these results, it is found that the **reliability of 85.5%** associated with **905 units** towards the ' $L_{Opt}$ ' is **optimum** for this system when using with the cost-reliability data from Table 6.13. Also interesting to note are the types of other observations detailed in the result table. The maximum value of the total loss function, ' $L_{Max}$ ' found in the optimisation process, is **significantly higher (16%)** than the corresponding value associated with ' $R_{Max}$ '. The latter is associated with total loss of **2588 units**, which is still nearly three times the ' $L_{Opt}$ ' amount for being 16% more than the optimum reliability. The most found value, ' $L_{Mode}$ ', of the loss function is about **13% higher** than the optimum value, ' $L_{Opt}$ ', and is associated with around **6% worse reliability** than the **optimum** solution.

System Type	B
Optimum value of total loss - $L_{Opt}$	<b>905</b>
Optimum value of system reliability	<b>85.5%</b>
Maximum value of total loss - $L_{Max}$	3096
Maximum value of system reliability associated with $L_{Max}$	93.1%
Mode value of total loss - $L_{Mode}$	1031
Mode value of System Reliability for $L_{Mode}$	80.0%
Maximum value of system reliability - $R_{Max}$	99.7%
Maximum value of total loss associated with $R_{Max}$	2588
Average value of total loss - $L_{Avg}$	1259
Standard Deviation of total loss	246
Coefficient of Variance total loss	19.5%
Average value of System Reliability	85.9%
Standard Deviation of System Reliability	5.6%
Coefficient of Variance System Reliability	6.5%

Table 6.15 Optimisation Results for System B

### 6.5.1.3 Optimisation Results For System C

The results for **system C** are detailed in Table 6.16. In light of these results, it is found that the system **reliability** of **84.2%** is **optimal** for this system since it is associated with the **lowest** value of the total loss, **813 units**, when used with the cost-reliability data from Table 6.13. Similar to **system A**, the maximum value of the total loss function, '**L<sub>Max</sub>**' found in the optimisation process, is **slightly higher (4%)** than the corresponding value of the loss function for '**R<sub>Max</sub>**'. In comparison with '**L<sub>Opt</sub>**', the amount of '**L<sub>Max</sub>**' is **around four times higher with 16% increased in associated reliability than the optimum**. The most found value, '**L<sub>Mode</sub>**', of the loss function is only **4% higher** than the '**L<sub>Opt</sub>**' with around **3% lower** associated reliability value than the optimum.

System Type	C
Optimum value of total loss - L <sub>Opt</sub>	<b>813</b>
Optimum value of system reliability	<b>84.2%</b>
Maximum value of total loss - L <sub>Max</sub>	3038
Maximum value of system reliability associated with L <sub>Max</sub>	97.9%
Mode value of total loss - L <sub>Mode</sub>	851
Mode value of System Reliability for L <sub>Mode</sub>	81.5%
Maximum value of system reliability - R <sub>Max</sub>	99.9.0%
Maximum value of total loss associated with R <sub>Max</sub>	2901
Average value of total loss - L <sub>Avg</sub>	1076
Standard Deviation of total loss	241
Coefficient of Variance total loss	22.4%
Average value of System Reliability	87.7%
Standard Deviation of System Reliability	3.9%
Coefficient of Variance System Reliability	4.5%

Table 6.16 Optimisation Results for System C

### 6.5.1.4 Optimisation Results For System D

The results for **system D** are detailed in Table 6.17. In light of these results, it is found that the system **reliability** of **86.2%** is **optimal** for this system since it is associated with the **lowest** value of the total loss, **1184 units**. The maximum value of the total loss function, '**L<sub>Max</sub>**' found in the optimisation process, is **higher** compared to the corresponding amount associated with the '**R<sub>Max</sub>**'. The **total loss amount of 'R<sub>Max</sub>'** is approximately just under **two and a half times more than the 'L<sub>Opt</sub>'** amount with **15% more** associated **reliability** than the **optimum reliability**. The most found value, '**L<sub>Mode</sub>**', of the loss function is very close to '**L<sub>Opt</sub>**' and **the associated reliability is not very different from the optimum**.

System Type	D
Optimum value of total loss - L <sub>Opt</sub>	1184
Optimum value of system reliability	86.2%
Maximum value of total loss - L <sub>Max</sub>	3293
Maximum value of system reliability associated with L <sub>Max</sub>	76.3%
Mode value of total loss - L <sub>Mode</sub>	1209
Mode value of System Reliability for L <sub>Mode</sub>	85.0%
Maximum value of system reliability - R <sub>Max</sub>	99.5%
Maximum value of total loss associated with R <sub>Max</sub>	2751
Average value of total loss - L <sub>Avg</sub>	1504
Standard Deviation of total loss	266
Coefficient of Variance total loss	17.7%
Average value of System Reliability	82.5%
Standard Deviation of System Reliability	8.3%
Coefficient of Variance System Reliability	10.0%

**Table 6.17 Optimisation Results for System D**

## 6.5.2 Optimisation Process Using Lower Cost Of Failure

In this section, the results from the optimisation process are presented using a **lower cost of failure amount**, 'C<sub>2</sub>' (1000 units) for all four systems by utilising the data from Table 6.1. For each of the four systems, the core optimisation results are detailed jointly in this section (Table 6.18). All other results including the graphical representation of the effects of optimisation algorithm on total loss and system reliability along with the list of sub-optimal results are shown in Appendix IX, for each of the four systems.

System Type	A	B	C	D
Optimum value of total loss - $L_{Opt}$	453	553	413	682
Optimum value of system reliability	70.2%	64.3%	70.0%	44.2%
Maximum value of total loss - $L_{Max}$	3699	3579	3481	3584
Maximum value of system reliability associated with $L_{Max}$	99.3%	93.1%	100.0%	92.8%
Mode value of total loss - $L_{Mode}$	522	644	1215	1429
Mode value of System Reliability for $L_{Mode}$	68.8%	63.1%	88.5%	72.4%
Maximum value of system reliability - $R_{Max}$	99.9%	99.7%	100.0%	99.2%
Maximum value of total loss associated with $R_{Max}$	3481	2975	3481	2648
Average value of total loss - $L_{Avg}$	966	1085	988	1189
Standard Deviation of total loss	353	323	336	310
Coefficient of Variance total loss	36.5%	29.8%	34.1%	26.1%
Average value of System Reliability	80.1%	74.9%	83.3%	62.8%
Standard Deviation of System Reliability	7.5%	7.5%	4.0%	9.3%
Coefficient of Variance System Reliability	9.4%	10.0%	4.8%	14.8%

**Table 6.18 Optimisation Results for All Systems Using Lower Cost of Failure**

### 6.5.3 Parametric Study

The results obtained from the applications of the optimisation algorithm on all four systems with the two variations, as stated above, are discussed together with the set of results from section 6.3. The structure of the parametric study is therefore based on the following grounds:

#### 6.5.3.1 Comparisons Of Results From The Two Cost-Reliability Data Tables

The results for all four systems are compared from the two data sets in Table 6.19. Each column represents the proportional change in the values of a system by taking into account the values from the appropriate results tables (detailed in the second row) for this system.

System Type	A	B	C	D
Source Tables	6.5 & 6.14	6.7 & 6.15	6.9 & 6.16	6.11 & 6.17
Optimum value of total loss - $L_{Opt}$	-7.1%	-8.7%	4.1%	-12.7%
Optimum value of system reliability	7.9%	16.1%	5.5%	16.1%
Mode value of total loss - $L_{Mode}$	-3.9%	2.2%	-12.4%	-20.0%
System Reliability for $L_{Mode}$	5.0%	1.1%	-5.1%	10.2%
Average value of total loss- $L_{Avg}$	-17.3%	-12.3%	-8.4%	-8.7%
Standard Deviation of total loss	-13.0%	-6.8%	-18.4%	7.0%
Coefficient of Variance total loss	5.2%	6.2%	-10.9%	17.2%
Average value of System Reliability	2.5%	5.7%	1.5%	8.7%
Standard Deviation of System Reliability	-28.4%	-11.9%	10.9%	69.1%
Coefficient of Variance System Reliability	-30.1%	-16.7%	9.3%	55.6%

**Table 6.19 Proportional Changes in Results of All Systems from the Two Data Sets**

The proportional changes specified in the above table are interpreted in Table 6.20 in order to explain the underlying changes in the values of each of the four systems.

System Type	A	B	C	D
Optimum value of total loss - $L_{Opt}$	Reduction in Total Loss	Reduction in Total Loss	Increment in Total Loss	Reduction in Total Loss
Optimum value of system reliability	Improvement in Reliability	Improvement in Reliability	Improvement in Reliability	Improvement in Reliability
Mode value of total loss - $L_{Mode}$	Reduction in $L_{Mode}$	Increment in $L_{Mode}$	Reduction in $L_{Mode}$	Reduction in $L_{Mode}$
System Reliability for $L_{Mode}$	Improvement in Reliability	Improvement in Reliability	Deterioration in Reliability	Improvement in Reliability
Average value of total loss- $L_{Avg}$	Reduction in $L_{Avg}$	Reduction in $L_{Avg}$	Reduction in $L_{Avg}$	Reduction in $L_{Avg}$
Standard Deviation of total loss	Reduction in Value	Reduction in Value	Reduction in Value	Reduction in Value
Coefficient of Variance total loss	Increment in Value	Increment in Value	Reduction in Value	Increment in Value
Average value of System Reliability	Improvement in Reliability	Improvement in Reliability	Improvement in Reliability	Improvement in Reliability
Standard Deviation of System Reliability	Reduction in Value	Reduction in Value	Increment in Value	Increment in Value
Coefficient of Variance System Reliability	Reduction in Value	Reduction in Value	Increment in Value	Increment in Value

**Table 6.20 Explanations of the Proportional Change in the Values of each System**



### 6.5.3.2 Comparisons Of Results From Two Different Costs Of Failure Amount

The results for all four systems generated from two different costs of failure amounts (section 6.4.1-4 and section 6.5.2) compared in Table 6.21. Each column represents the proportional change in the values of a system by taking into account the values from the appropriate results tables (detailed in the second row) for this system.

System Type	A	B	C	D
Source Tables	6.5 & 6.18	6.7 & 6.18	6.9 & 6.18	6.11 & 6.18
Optimum value of total loss - $L_{Opt}$	-48.3%	-44.2%	-47.2%	-49.7%
Optimum value of system reliability	-12.8%	-12.8%	-12.3%	-39.1%
Mode value of total loss - $L_{Mode}$	-40.6%	-36.1%	25.2%	-5.4%
System Reliability for $L_{Mode}$	-16.2%	-20.3%	3.0%	-6.1%
Average value of total loss- $L_{Avg}$	-26.0%	-24.4%	-15.9%	-27.8%
Standard Deviation of total loss	27.6%	22.6%	13.8%	24.8%
Coefficient of Variance total loss	72.4%	62.1%	35.4%	72.8%
Average value of System Reliability	-8.0%	-7.8%	-3.6%	-17.3%
Standard Deviation of System Reliability	54.4%	18.9%	12.9%	90.5%
Coefficient of Variance System Reliability	67.8%	29.0%	17.1%	130.2%

**Table 6.21 Proportional Changes in Results of All Systems from the Two Cost of Failure Amounts**

An interesting observation is the proportional change in the optimum result with respect to the mode and average values of the total loss, depicted in Table 6.22. All results from this section are discussed in detail in the next chapter..

		C1	C2
<b>A</b>	L <sub>Mode</sub>	-0.18%	-13.14%
	L <sub>Avg</sub>	-32.76%	-53.05%
<b>B</b>	L <sub>Mode</sub>	-1.66%	-14.14%
	L <sub>Avg</sub>	-30.93%	-49.02%
<b>C</b>	L <sub>Mode</sub>	-19.51%	-66.03%
	L <sub>Avg</sub>	-33.46%	-58.20%
<b>D</b>	L <sub>Mode</sub>	-10.25%	-52.30%
	L <sub>Avg</sub>	-17.65%	-42.68%

**Table 6.22 Proportional Changes in Optimum with respect to Mode and Average Values**

### 6.5.3.3 Comparisons Of Results For Establishing The Best System Topology

Given a choice of various system designs and large selection of components for each sub-system, the risk-based reliability allocation method together with the optimisation algorithm can be used to select optimal system design (topology) from all available choices. This section provides analysis of such nature using the optimisation results acquired for all four systems (Fig. 6.9) by utilising the two different cost-reliability data tables (Table 6.1 & 6.13) and cost of failure amounts (section 6.5.2) for determining the optimal system topology.

With the view of the structure of this chapter, the optimal topology selection process is presented in the following three steps:

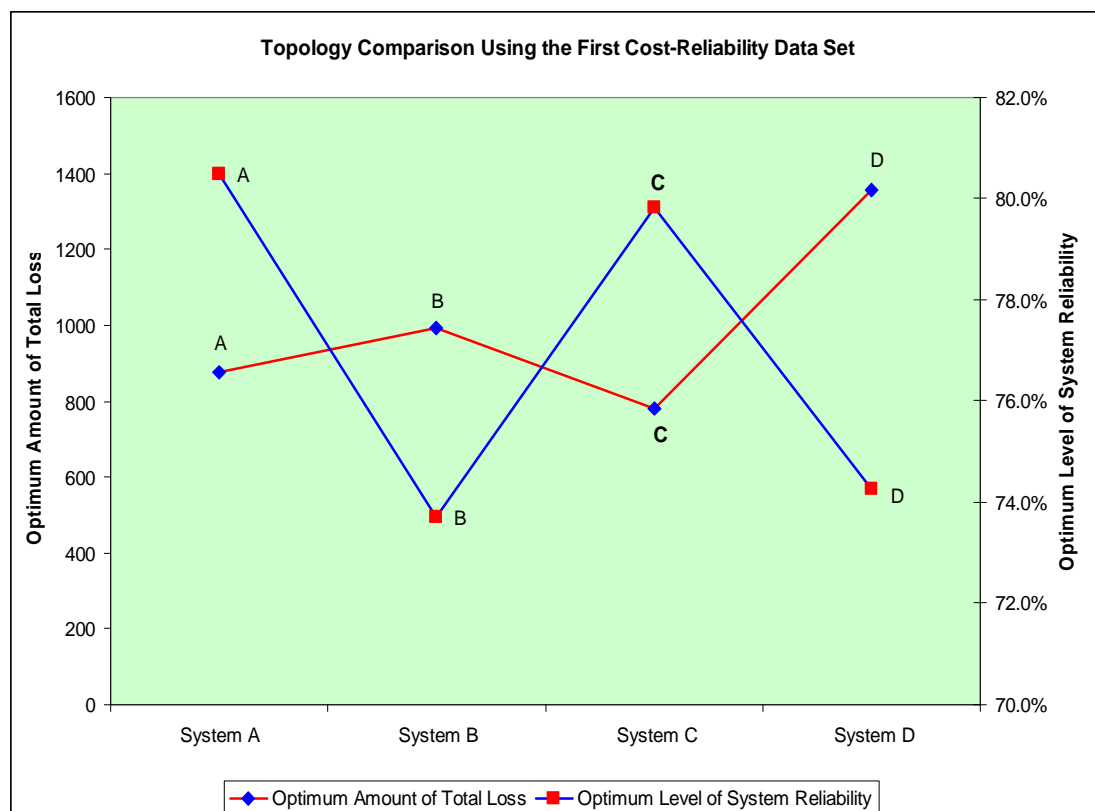
- Optimal Topology Selection Using First Data Table
- Optimal Topology Selection Using Second Data Table
- Optimal Topology Selection Using Lower Cost of Failure

### Optimal Topology Selection Using First Data Table

The results for each system highlighted in **Table 6.25** and in **Fig. 6.17** show that the **system C is associated with the least amount of total loss** in comparison to all other systems in the table. Therefore, if a choice is available, system C can be selected as the **optimal topology**.

System Type	A	B	C	D
Optimum value of total loss	877	992	<b>782</b>	1356
Optimum value of system reliability	80.5%	73.7%	<b>79.8%</b>	74.3%

**Table 6.23 Optimal Topology Selection Using First Data Table**



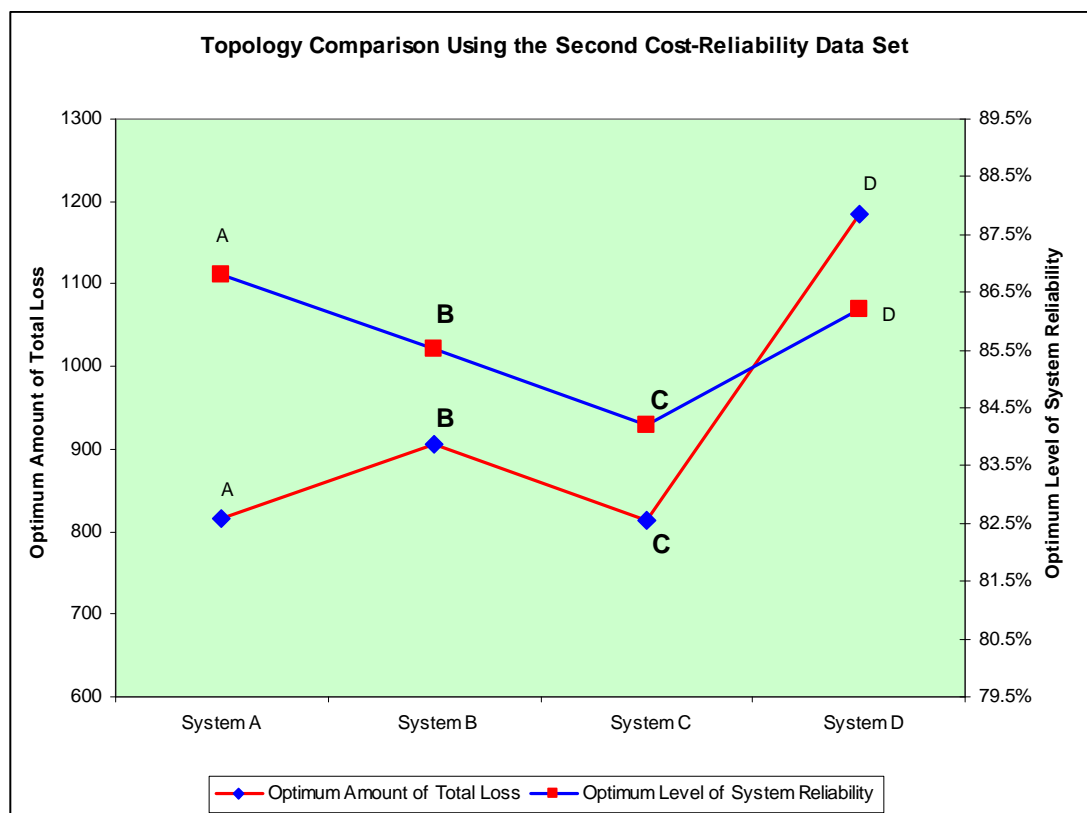
**Figure 6. 17 Topology Comparison Using the First Cost-Reliability Data Set**

### *Optimal Topology Selection Using Second Data Table*

The results for each system highlighted in **Table 6.24** and in **Fig. 6.18** show that both **system A** and **system C** are competing very closely for the **optimal** spot. The amount of **total loss associated with each of the two systems is lower** than the corresponding **amounts of system B and system D** and differs only slightly from each other. Using this information only, either of the two systems can be selected as optimal with bias decision toward **system A for being slightly better in the level of reliability**.

System Type	A	B	C	D
Optimum value of total loss	815	905	813	1184
Optimum value of system reliability	86.8%	85.5%	84.2%	86.2%

**Table 6.24 Optimal Topology Selection Using the Second Data Table**



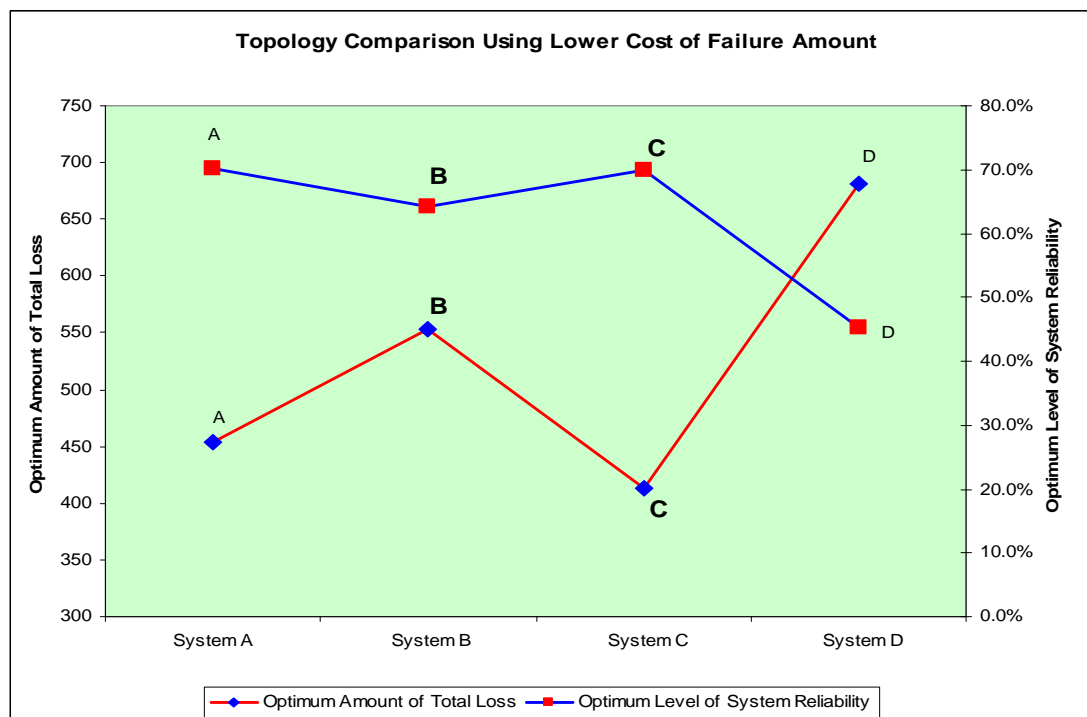
**Figure 6. 18 Topology Comparison Using the Second Cost-Reliability Data Set**

### *Optimal Topology Selection Using Lower Cost Of Failure Amount*

The results for each system highlighted in **Table 6.25** and in **Fig. 6.19** show that **system C** is associated with the **least amount of total loss** in comparison to all other systems in the table. Therefore, if a choice is available, **system C can be selected as the optimal topology**.

System Type	A	B	C	D
Optimum value of total loss	453	553	413	682
Optimum value of system reliability	70.2%	64.3%	70.0%	45.2%

**Table 6.25 Optimal Topology Selection Using Lower Cost of Failure Amount**



**Figure 6. 19 Topology Comparison Using the Lower Cost of Failure Amount**

The topology optimisation can be further extended by assuming a situation where a selection of the best system design across all available results (derived from the two

data sets and the different cost of failure amounts), is required. Understandably, the solution for this would also be system C. With this view in mind, **Table 6.26** is presented with yet **more useful information about the different levels of total cost associated with various system designs**, right to the point of the **selected alternatives for all components** with corresponding levels of system reliability. This information can be of great interest to reliability engineers in **analysing various competing topologies** with the view of system reliability levels with **corresponding total cost associated with system failures**. Various adaptation of Table 6.26, which is specific to the results from section 6.5.2, can be generated using the research methodology including all combinations of the parametric studies discussed in this section. For example, Table 6.27 which is created for set of results in section 6.5.1.

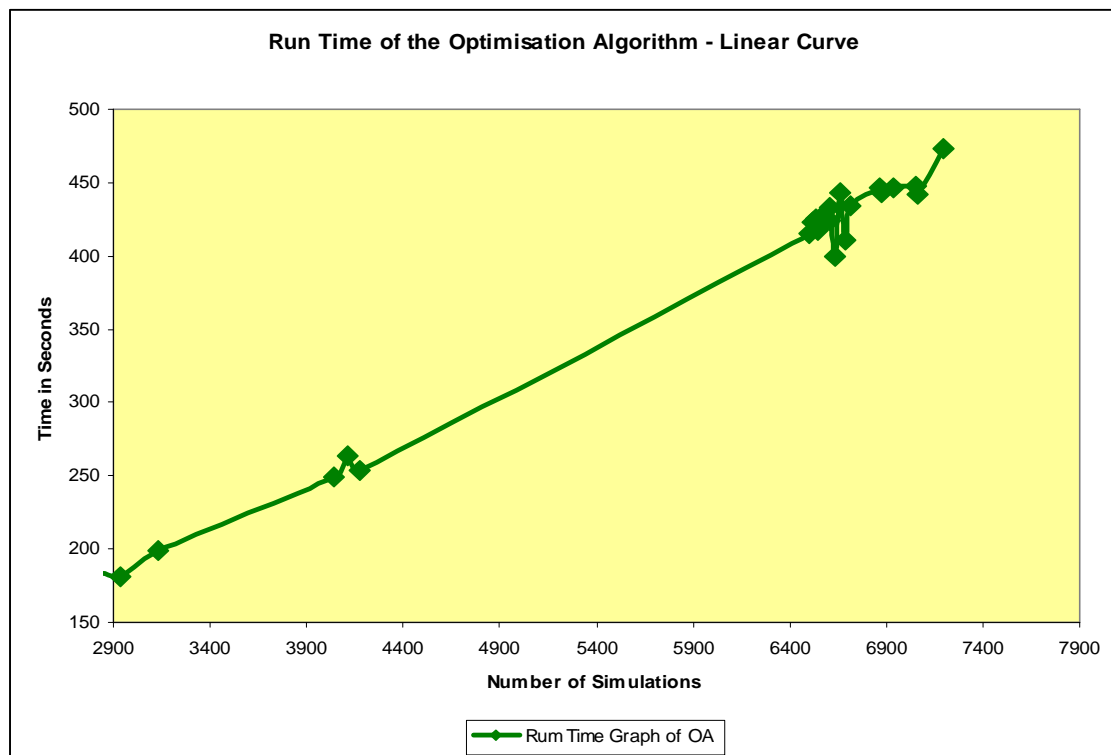
No. Results	Topology Type	System Reliability	Total Loss	Best Configuration		
1	C	70.0%	412.8	2-2-2	2-2-2	4-3-2
2	C	74.4%	435.6	3-2-2	5-2-2	4-3-2
3	C	77.8%	464.5	2-4-3	3-2-5	3-3-4
4	C	79.6%	507.4	3-2-4	3-2-6	4-3-4
5	C	81.3%	707.4	3-2-2	3-2-6	4-3-9
6	A	83.8%	742.1	2-6-2	4-3-4-6	4-8
7	C	84.3%	771.9	3-3-2	5-2-7	4-3-9
8	C	86.3%	804.0	5-3-2	5-2-7	4-3-9
9	C	88.5%	948.0	6-3-5	5-2-8	4-6-5
10	C	89.0%	960.8	6-4-5	5-2-8	4-6-5
11	A	90.1%	1056.9	7-5-2	3-2-2-3	6-11
12	C	95.9%	1859.8	3-5-10	9-5-2	11-10-15
13	A	94.9%	1885.4	3-5-10	9-5-2-11	10-5

**Table 6.26. Best results for risk-based reliability allocation on all topologies**

Result No.	System Type	Total Loss	System Reliability	Reliability Investment	Risk of Failure	System Configuration									
						6	6	3	5	4	2	6	3	2	
1	C	813	84.2%	377	437	6	6	3	5	4	2	6	3	2	
2	A	815	86.8%	446	369	6	6	3	3	2	5	2	3	9	
3	C	821	87.9%	479	342	6	6	3	4	4	6	6	3	5	
4	C	829	86.0%	436	393	6	6	3	5	4	2	6	3	5	
5	C	841	83.5%	384	457	6	6	3	4	4	4	2	3	5	
6	A	843	87.8%	498	345	3	6	7	3	2	5	4	3	9	
7	C	843	88.3%	513	330	6	6	3	5	4	4	2	3	9	
8	C	843	87.5%	492	351	6	6	3	4	4	4	2	3	9	
9	C	849	80.7%	321	528	3	6	3	4	4	4	2	3	4	
10	C	849	83.3%	387	463	6	6	3	5	4	4	2	3	4	
11	A	854	84.7%	427	427	6	4	3	4	2	2	2	3	9	
12	C	856	84.6%	424	432	6	6	3	5	4	2	5	3	4	
13	A	859	86.8%	484	374	6	6	3	5	2	2	3	3	9	
14	A	876	85.5%	468	408	6	4	3	5	2	2	3	3	9	
15	A	885	84.3%	447	438	3	4	4	2	2	5	4	3	9	
16	C	886	81.1%	363	523	3	4	3	5	4	2	5	3	4	
17	A	886	81.9%	389	497	3	3	3	5	2	2	2	3	9	
18	A	888	82.8%	411	476	3	3	4	5	2	2	2	3	9	
19	C	890	77.5%	280	609	3	3	3	3	4	4	2	3	4	
20	C	895	78.3%	304	591	4	3	3	3	4	4	2	3	4	
21	B	905	85.5%	496	409	8	6	2	4	2	2	2	2	9	
22	C	914	76.5%	278	635	2	3	3	3	4	4	2	3	4	
23	B	943	92.1%	722	221	8	6	2	4	3	2	2	2	12	
24	B	955	84.7%	518	437	6	7	2	2	2	5	3	2	9	
25	B	955	91.4%	717	238	8	6	2	2	3	2	2	2	12	

**Table 6.27 Best results for risk-based reliability allocation on all topologies from section 6.5.1**

It is useful to point out that the results produced in this chapter are a snap-shot of only five executions on average per system, of the computer program with the optimisation algorithm. Each time the program was executed, it ran with different and at times increasing values of the OA parameters. For example, the final run in system A was executed with parameter values:  $p_{RUN} = 4$ ; (No. of generations)  $p_{size} = 100$ ; (No. of unique solutions in a population)  $C_{RUN} = 100$ ; (No. of crossover cycles) and  $M_{RUN} = 400$ ; (No. of mutation cycles), performing 6000 simulations on average (out of  $12^9$  possible solutions) in around seven minutes. In all the five executions per system, the run time increased linearly with the increased complexity of the optimisation algorithm which demonstrates the efficiency of this algorithm. Figure 6.20 details the execution times of the computer program using all four systems.



**Figure 6. 20 Run time of the optimisation algorithm**



# 7

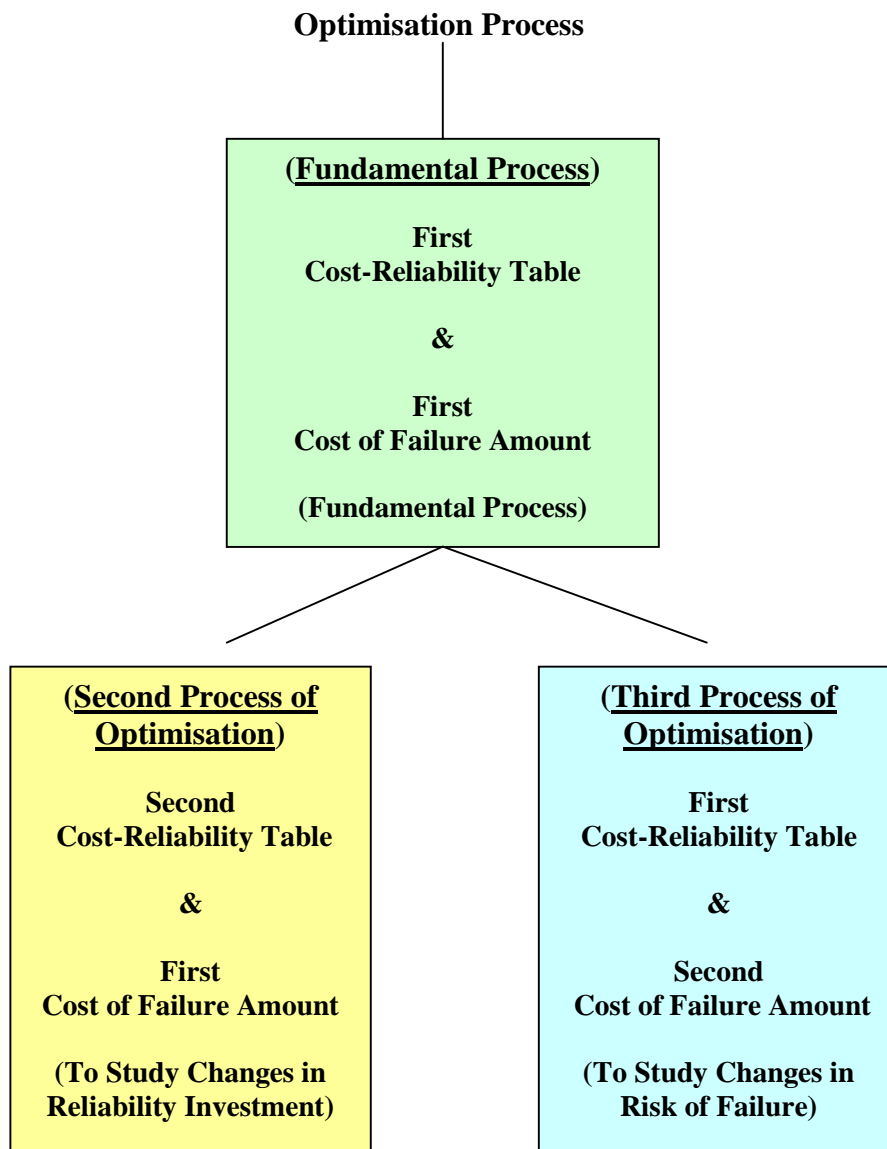
## DISCUSSION

---

This chapter provides discussion on the methodology developed in this research along with its application and results, as detailed in the previous chapter. Based on the principal of genetic search, the research methodology combines the exploration abilities of genetic search with skilful exploitation of hill climbing procedures and introduces a different model of evolution compared to classical GA. The main features of this model are the generation of populations with unique chromosomes, working exclusively with the elite chromosomes and introducing genetic variations in the elite chromosomes using prudently designed genetic operators for ensuring rapid and efficient convergence to optimum or near optimum region of the search space. The two main reasons for implementing these notions in the optimisation algorithm are the non-linear cost-reliability relationship and the extremely large search space. A comprehensive detail of this method is presented in Chapter 5. The application of the methodology is demonstrated in Chapter 6 ( along with appendix II-IX) by using four types of system configurations which are commonly found in the reliability literature and the results are discussed in detail in this chapter.

## 7.1 OBSERVATIONS FROM THE OPTIMISATION PROCESS

The results obtained in the previous chapter from the application of the research methodology together with the risk based reliability allocation method, on the four systems, from Fig. 6.9, are discussed in this section. There are three main categories of these results depending on the amount assumed towards the cost of failure and the type of cost-reliability data set used (Table 6.1 and Table 6.13), as shown in Fig 7.1.



## 7.1.1 Fundamental Process of Optimisation

The first optimisation process using Table 6.1 and cost of failure amount, ' $\bar{C}_1$ ', is the fundamental process for showing the viability of the research methodology for optimising the four systems using the risk based reliability allocation method in the presence of a large choice of component alternatives, available for each system. The optimisation process, described in section 6.3, therefore, details the optimal combination of alternatives selected for each system associated with the minimum total cost, ' $L_{Opt}$ '.

Tables 6.5, 6.7, 6.9 and 6.11 are the main results table showing the statistic of the optimisation process for all systems, respectively. The values such as maximum total loss, ' $L_{Max}$ ', mode value of the total loss, ' $L_{Mode}$ ', and average total loss, ' $L_{Avg}$ ', along with the corresponding values of the reliabilities, are detailed in each of the four tables in order to highlight the variations in these results for important reasons such as:

- To show that maximum reliability can be associated with maximum total loss (system A and C) and maximum total loss is not always associated with maximum system reliability; systems such as B and D reveal that maximum system reliability, ' $R_{Max}$ ' is in fact linked with lower values of the total loss compared to ' $L_{Max}$ '.
- To understand how close is the value of ' $L_{Opt}$ ' identified by the optimisation algorithm, in proportion with the most found value of the total loss, ' $L_{Mode}$ '. This is useful in working out the complexity of the system and also reflects the efficiency of the algorithm in exploring the optimal solution.

- For a general expectation of the total loss associated with the system reliability, the average values of the total loss, ' $L_{Avg}$ ' and system reliability are also computed in the optimisation process for each system. These results along with the standard deviations for the two values can be used for performing many statistical analyses on the sample population and for generating probabilistic distributions. These analysis are not part of this research however, the value of ' $L_{Avg}$ ' for each system is taken into account for comparing the corresponding value of the ' $L_{Opt}$ ' (Table 6.21 – 6.24).

While Tables 6.5, 6.7, 6.9 and 6.11 highlight the optimum result for systems A – D respectively, the optimisation algorithm, being a population based search method, also produces a list of other sub-optimal solutions which can be of great interest to a reliability analyst for gaining better understanding of the current system. The results, shown in Tables 6.6, 6.8, 6.10 & 6.12, are useful in studying different levels of reliability for respective values of the total loss function along with the important information concerning the actual configuration of the system by illustrating the precise description of the chosen alternatives of all components. Hence, with a possible flexibility in the overall budget constraint, the required level of system reliability can be adjusted with the view of the associated total loss from system failure.

However, it is very important for a reliability analyst to understand the complex nature of cost-reliability relationship before choosing the right combination of components from the given alternatives. As can be seen in these result tables (e.g. Table 6.6), the amount of total loss is different for different levels of system reliability and both values fluctuate throughout without showing any obvious trend. Furthermore, it can be observed from the same tables that highest levels of system reliability can also be associated with the highest amount of total loss (systems A & B); increasing the reliability inappropriately can also increase the losses from failures despite reducing the probability of failure, as mentioned in Todinov (2004). It is, therefore, very difficult and challenging to analytically estimate the optimum level of

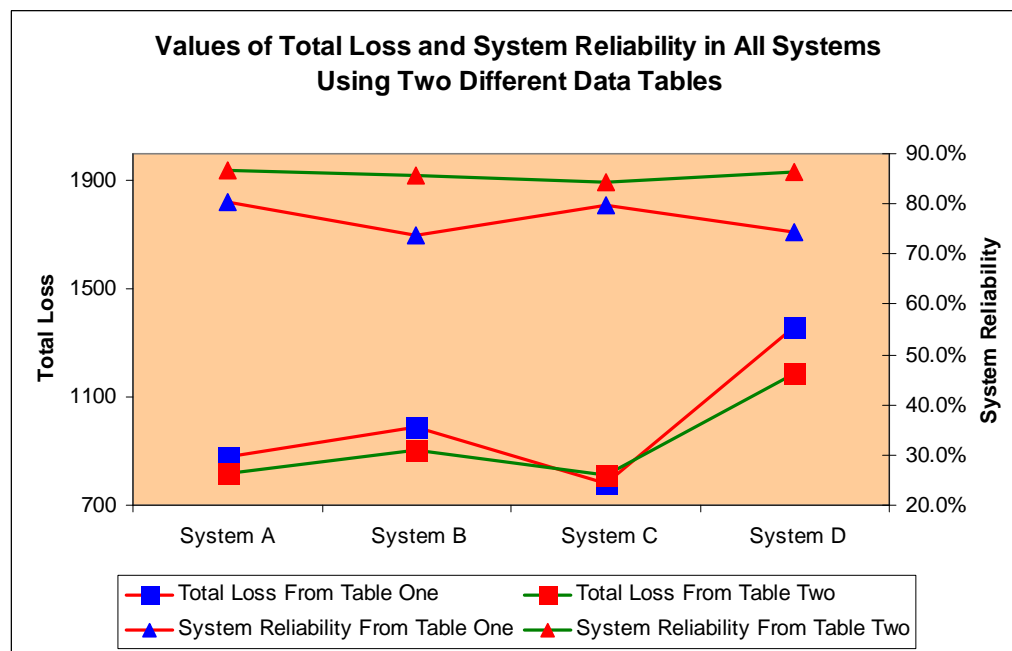
system reliability while keeping the amount of total loss down to a minimum possible value for a system consisting of many components with a large choice of available alternatives.

### 7.1.2 Second Process of Optimisation

The observations from the first optimisation process emphasise the difficulty in estimating the minimum value of the total loss function because of the complexity in the relationship between cost and reliability. In order to have a closer look at the level of difficulty in minimising the loss function, the optimisation process is repeated twice, with two variations as detailed in section 6.4.

The first variation involves using a different data set (Table 6.13), showing a non-monotonically increasing relationship between cost and reliability, among the given alternatives of the components. The table is one example of many arbitrary ways in which such data can be created for experimental purposes. The objective of using such data is to observe the impact of reliability investment, ' $Q$ ', on the loss function, as stated in equation (6.7). Because ' $Q$ ' is the total cost of the selected alternatives of the components which is added to the risk of failure amount, ' $K$ ', for producing the total loss figure, using the second data set creates more opportunities for the optimisation process for producing better values of the loss function since cheaper components with higher reliabilities can be selected with higher probabilities when using the second data table. As a result, the total cost of the selected components will generally not increase in the same proportion as it would if a first data set is used. At the same time, the system reliability will be improved in most cases, because of the increase in the individual reliability of the components, which will subsequently decrease the risk of failure amount; the total loss figure will therefore, be smaller in more occurrences than observed previously in the first optimisation process.

The results generated in the parametric study detailed in section 6.4.1, reflect this change in the values of the loss function. Table 6.19 shows that the values of the optimum losses for most of the systems from section 6.3 are decreased when the second cost-reliability data is used (Table 6.13). This is also shown graphically in Fig. 7.2. The corresponding values of the system reliabilities also appeared to be improved with significant reduction in the values of ' $L_{Mode}$ ' and ' $L_{Avg}$ ', in general.



**Figure 7.2 Effects of the Two Cost-Data Tables on Total Loss and System Reliability Values in All Four Systems**

Because of the non-linear relationship between cost and reliability, the coefficient of variances for both average total loss and average system reliability appears to fluctuate consistently. A possible reason for this may be the slightly less cost of extremely reliable alternatives in the second data table which allows more flexibility in choosing better components (more reliable) without increasing the cost associated towards reliability investment ( $Q$ ) in the same proportion as seen in the results from the first data table. As a result, the total loss function from equation (6.7) can have a number of good solutions for higher levels of system reliabilities, as observed in the comparison table. An interesting observation is the proportional change in the

optimum result with respect to the mode and average values of the total loss as shown in Tables 7.1 & 7.2.

System Type	A	B	C	D
Proportion change in $L_{Opt}$ w.r.t $L_{Mode}$	-0.18%	-1.66%	-19.51%	-10.25%
Proportion change in $L_{Opt}$ w.r.t $L_{Avg}$	-32.76%	-30.93%	-33.46%	-17.65%

**Table 7.1 Proportional Change in Optimum Result from the First Data Table**

System Type	A	B	C	D
Proportion change in $L_{Opt}$ w.r.t $L_{Mode}$	-3.48%	-12.18%	-4.37%	-2.06%
Proportion change in $L_{Opt}$ w.r.t $L_{Avg}$	-24.43%	-28.10%	-15.86%	-39.82%

**Table 7.2 Proportional Change in Optimum Result from the Second Data Table**

These results show that it is very difficult to predict optimal combination of component for minimising the loss function as the optimum value is generally too far below the average values of the total loss in all systems using the two different cost-reliability data tables. However, the mode values of the loss function are generally closer to the optimum value, which reflects the efficiency of the optimisation algorithm for effectively searching the search space for promising solutions.

### 7.1.3 Third Process of Optimisation

The second variation in the fundamental optimisation process of section 6.3 is introduced by means of using a different value of the cost of failure amount, ' $\bar{C}_2$ ',

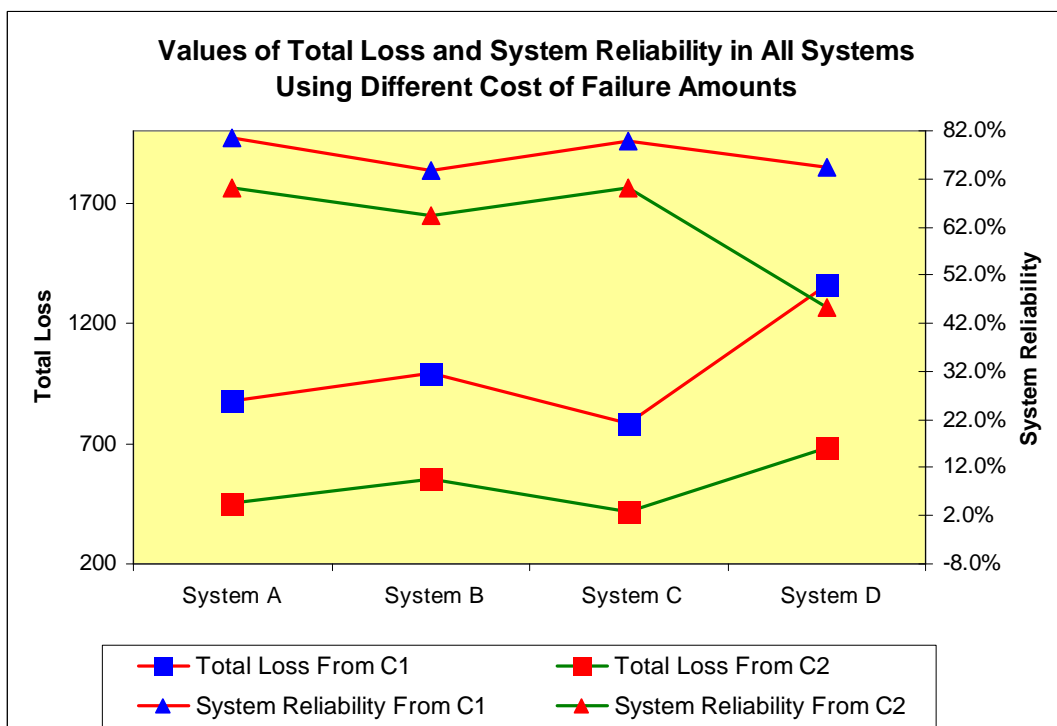
associated with a given system failure; it is assumed to be 1000 units, which is a reduced value from the one used in section 6.3 ( $\bar{C}_1 = 2500$  units). The objective of using different cost of failure amounts is to observe the impact of risk of failure, 'K', on the loss function stated in equation (6.7). From equation (6.3), it can be seen that 'K' is derived from the product of the probability of failure and the cost given failure ' $\bar{C}_2$ '. Therefore, the reduction in the ' $\bar{C}_2$ ' amount will reduce the size of 'K' and since the latter constitutes to the value of the total loss directly, along with the reliability investment, 'Q', the effect of reduced cost associated with a given failure will reflect the effect of 'K' on the loss function. In view of this notion, using ' $\bar{C}_2$ ' will increase the chances of using cheaper components with lower levels of reliability (Table 6.1) for minimising the objective function from equation (6.7). This is because lower cost of the components along with the lower risk of failure will produce lower values of the loss functions.

The result Table 6.21 from section 6.4.3.2 is produced by taking into account the combined results of section 6.3 and 6.4.2. The table shows that using a lower value of the cost of failure amount indeed decreases the total losses from system failures, also presented graphically in Fig. 7.3. All four systems, therefore show significant reduction in the optimum values of the total loss compared to the corresponding values calculated using a higher cost of failure amount. The other noticeable result is the reduction in the values of system reliabilities associated with the optimum values of total loss for each of the four systems. This can be explained by the same notion as stated above; cheaper less reliable component alternatives have relatively higher chances of selection because of the reduced amount. ' $\bar{C}_2$ ', associated with a given system failure, which reduces the overall system reliability.

The same observations generally hold for the values of ' $L_{\text{Mode}}$ ' and ' $L_{\text{Avg}}$ ' and the associated reliabilities. The coefficient of variances for both average total loss and average system reliability appears to be very high in the above result table. This can be justified by the explanation that the average values of these variables are lower when lower cost of failure is used but the corresponding standard deviations are proportionally higher. The average values, ' $L_{\text{Avg}}$ ', of loss function are lower in all



systems because of the reduced risk of failure amount associated with a given failure; the use of cheaper components alternatives with lower reliability levels offers more flexibility in producing lower values of the loss function without increasing the cost associated with the reliability investment ( $Q$ ) and in particular the risk of failure. The latter will be in lower proportion compared to the corresponding amount used in the results from section 6.3 because of less cost of failure associated with each failure. As a result, the total loss function can have a number of good solutions with lower levels of system reliabilities, as observed in the comparison Table 6.21.



**Figure 7.3 Effects of the Two Cost of Failure Amounts on Total Loss and System Reliability Values in all Four Systems**

An interesting observation is the proportional change in the optimum result with respect to the mode and average values of the total loss, depicted in Table 6.22. These results show that it is very difficult to select an optimal combination of component for minimising the loss function as the optimum value is generally two far below both the average and mode values of the total loss in all systems using two

different values for cost of failure amounts ( $\bar{C}_1 > \bar{C}_2$ ). The problem is compounded for optimisation process using ' $\bar{C}_2$ ', with the view of the mode values of the loss function; for systems C and D, the optimum value is well below the commonly found solution of the loss function, for example. The optimisation of such reliability systems using a large selection of component alternatives can therefore be deemed as a very difficult process, requiring efficient techniques for exploring the optimum solution in a complex search space; for that reason, the methodology presented in this research provides this opportunity.

## **7.2 COMPARATIVE ANALYSIS OF THE OPTIMISATION ALGORITHM**

The aim of this research has been to extend the recently published work of Todinov (2004, 2006) on risk based reliability allocation, by demonstrating its application on complex, large scale non-repairable engineering systems. Therefore, being part of a relatively new research, it is very difficult to perform the comparative analysis of the results obtained in the preceding sections with other published work in the field of risk based reliability allocation. The optimisation performed here is fundamentally composed of two blocks.

### **7.2.1 Risk Based Reliability Allocation**

First of which is to estimate components reliability in view of the risk based reliability allocation approach for a given system. In general, the conventional approach of reliability optimisation methods to-date appears to be driven mostly by the principle of setting the highest level of system reliability for a given cost. However, the objective of the risk-based reliability allocation technique is driven solely by the amount of 'total losses associated with a system failure'. Many of the popular reliability allocation strategies do not take into account the total losses from

failures during reliability allocation. Since 1977, there have been a significant number of articles and books such as, Tillman *et al.* (1977, 1980), Jensen (1970), Tzafestas (1980), Misra (1986, 1992), Xu *et al.* (1990), Aggarwal (1993), Brown *et al.* (1997), Yang *et al.* (1999), Cantoni & Zio (1999), Kuo & Prasad (2000b), Zio (2000), Guikema and Pate-Cornell (2002), Elegbede *et al.* (2003), Pham (2003), Wattanapongsakorn & Levitan (2004), Meziane *et al.* (2005), Yalaoui *et al.* (2004, 2005) and Kuo & Wan (2007), related to reliability optimisation involving costs. Most of the methods described in these sources, however, are related to either maximising the reliability of a system given an overall budget constraint (a maximum acceptable total cost of resources toward the reliability maximisation) or minimising the total cost of resources necessary to achieve a specified level of system reliability. For occasional instances where a balance between cost and reliability is targeted, the objective has not been to minimise the risk of failure, as in the risk based reliability approach. A comprehensive comparison of the risk based reliability allocation method with other published work is provided in section 2.3.

## **7.2.2 Optimisation using Evolutionary Algorithm**

The second block is to use an evolutionary algorithm as the research methodology which provides a structured approach for allocating optimal system reliability by using the risk based reliability method when a large choice of component alternatives is available. The approach is based on the realisation that the relationship between the component cost and reliability is generally very complex and unpredictable particularly for estimating the amounts of total loss (sum of reliability investment and risk of failure) associated with non-repairable system failures. As explained in section 4.5, genetic search has been gaining wide interests from the researchers in solving difficult optimisation problems especially in the field of reliability engineering and system designs. Being based on the genetic search principle, the method of optimisation presented in this research is fundamentally similar to the articles listed in section 4.5; however, there are some significant variations which make it considerably different. These are explained below:

### 7.2.2.1 Population Structure

There is only one population maintained in this method at all times unlike traditional genetic algorithms where at least two populations are used. Examples of some of these populations are, an initial population of randomly (in general) selected chromosomes, separate populations for holding parent and offspring chromosomes, population in the form of a breeding pool and so on (Goldberg, 1989; Alander, 1999). The format of the population used in OA is similar to the model of steady state population used by Faulkenauer (1998) for solving difficult grouping problems. He also used the tournament selection method for determining the order of replacing the worst solutions with the newly formed offspring with better fitness, instead of using the common approach of proportional selection (*see* Chapter 3). Despite possessing similarities with the population structure used by Faulkenauer, the population structure in the research methodology is still considerably different in that it begins with a predefined number of chromosomes (population size) and by the end of an iteration, it is left with only two best solutions. Both members of the population experience genetic variations unlike Faulkenauer where only some of the randomly selected members of the population are genetically modified. The similarities of the population structure of the OA can also be compared with the method used by Syswerda (1989) because of the steady state format utilised in both methods. However, the approach used by Syswerda is different in the sense that in his method, proportional selection is used for selecting parents and replacing worst solutions with new offspring which are found as the result of applying genetic operation on only two individuals per generation. The key benefit of using just one population in the OA besides simplicity is the reduction in computational effort required by the algorithm for processing the search in shorter span of time with less memory requirements.

### 7.2.2.2 Chromosome Structure

The optimisation algorithm uses the approach of representing the chromosomes with a real number encoding method. Traditionally, chromosomes have been coded as binary strings (Goldberg, 1989) but for combinatorial optimisation problems, an

encoding using integer values can be more efficient (Holland, 1975; Spall 2003). The general structure of the chromosome in OA consists of a string containing ' $M$ ' loci for genes (equal to the total number of components) with ' $N$ ' alleles representing the integer value up to the available number of alternatives, for corresponding genes (component). This approach is similar to Coit and Smith (1996) however, the interpretation of the chromosome string is different. The components are ordered and not divided into subsystems and a particular allele greater than ' $N$ ' is not used to represent the empty selection. The value (allele) of each component (gene) in the encoding mechanism of OA is filled randomly by using a uniformly distributed random numbers generator and in accordance with the parameters ' $M$ ' and ' $N$ '. This structure is specific to the nature of the optimisation problem presented in this research and simplifies both coding and decoding of the string. See section 5.4.2 for an example.

### **7.2.2.3 Embedded Improvement Procedures**

As shown in section 5.2, the application of the improvement procedures in step 5(d) and 5(g) of the optimisation algorithm strengthens the capability of the genetic search for comprehensively exploring the search space and exploiting many feasible solutions of the given optimisation problem with a built in diversity mechanism. The diversity in the search space is further complemented by the selection of the next population containing unique non-duplicate random genotypes, not evaluated in any of the previous generations; the addition of the fresh set of unique chromosomes in the next population boost the multi-directional search ability of the OA.

Unlike the usual practices of using crossover and mutation probabilities shown in the literature (Alander, 1999; Kendall, 2005; Ashlock, 2006 and Goldberg, 1989, 2002) the embedded improvement procedures of the OA, operate on one gene (component) at a time for applying the genetic operations such as crossover and mutation and then gradually increasing to two and three components later in the optimisation cycle, the chances for evaluating good solutions with better fitness can be carefully organised. An instance of this process is shown in Table 6.6, when reading it backwards. For

example, the total loss amount in result no.18 is improved by 7% from the amount in result no. 19 (found by the random search) through first stage mutation operation at position seven. The result no. 17 is an improved version of result no. 18 obtained by performing a two stage mutation operation at position nos. 2 and 7 (mutation sites). Similarly, result no.16 is obtained by the combination of a two stage crossover operation at position 2 and 6 from results 18 & 17 and then a single stage mutation operation at position 7 (result no. 16 could have also been found by a three stage mutation alone on these positions also), and so on.

The improvement procedures effectively deal with the epistatis phenomenon found in the cost and reliability relationship using simple hill-climbing process carried out by randomly sampling the local regions of the feasible solutions and turning the good solutions into even better ones. Consequently, it increases the chances of converging to a promising solution significantly while avoiding a premature convergence and other basic weaknesses of classical GAs such as limited ability to exploit promising regions of the genetic search space and effective treatment of infeasibilities originating from restrictions belonging to (combinatorial) optimisation problems (Schoneberger, 2005).

Similar approaches of using hybrid GAs are appearing to be very popular among researchers. For example, Hsieh and Hsieh (2003) use GA with steepest decent method to optimise system cost during the period of task execution for a cycle-free computer distribution system. Using hybrid GA, Hsieh (2003) also solves similar optimisation problem based on the constraints on the hardware redundancy level. By incorporating neural networks, fuzzy logic and local search with classical GA, Lee *et al.* (2001, 2002a, 2002b), show the reliability design optimisation which considerably improves the computational time. It is important to point out the improvement procedures introduced in OA are designed in view of the risk based reliability allocation problem introduced in this research and for the reasons explained in section 7.1.

#### 7.2.2.4 Software Implementation

The optimisation algorithm is simple to program and does not contain confusing mathematical calculations. This is because there are no parameters such as crossover probability or mutation probability as commonly found in the literature. Even the selection and maintenance of the populations, in each iteration of the algorithm, is not probabilistic as in the conventional GA, where the common approach is to use a roulette wheel strategy (see Chapter 3 & 4 for more details).

Another interesting statistic associated with the performance of the optimisation algorithm is the value of the standard error. For every execution of the program, the standard error has appeared to be very similar for each system. Additionally, not all statistical results highlighted in the respective result tables in sections 6.3 to 6.5 are directly related to the work undertaken in this research. For example, results such as the mean, standard deviation and coefficient of variance are only provided to show the outstanding performance of the optimisation algorithm and the viability of the embedded improvement procedures based on crossover and mutation operations. While these results are useful in evaluating the OA, it is currently outside the scope of this research to analyse them in details.

Since the objective of this research is to demonstrate the risk-based reliability allocation method on the specific problem presented in this research, and given the excellent quality of the results already obtained, it was not deemed necessary to increase the number of executions of the optimisation algorithm or even the complexity of the OA parameters in the computer program. Nevertheless, through the extension of this research in future, it is aimed that such modifications will be explored in the hope of improving the solutions and carrying out more interesting parametric studies with in-depth statistical analysis of all the results.



# 8

## CONCLUSIONS AND FUTURE RECOMMENDATIONS

---

### 8.1 CONCLUSIONS

In light of the methodology developed and used in this research, the following conclusions are made:



- Risk-based reliability allocation method together with the proposed optimisation algorithm can be used as an excellent decision making tool for estimating optimal level of system reliability by selecting an appropriate combination of components, from a given choice of alternatives, for a given non-repairable reliability system.
- Risk-based reliability allocation method together with the optimisation algorithm can be used to highlight various levels of system reliability with associated total cost of system failure. This information can help reliability engineers in streamlining system design and total cost of failure.
- Given a choice of various system designs and large selection of components for each sub-system, the risk-based reliability allocation method together with the optimisation algorithm can be used to select optimal system topology from all available choices. Information similar to Table 6.27, can further assist reliability engineers in analysing various competing topologies with the view of system reliability levels with corresponding total cost of system failure.
- With reference to the optimisation algorithm, every new population in the algorithm is a fresh sample of randomly generated but unique non-duplicate chromosomes. This introduces multi-directional search diversity in the solution space and also improves the chances of rectifying the main deficiencies of the classical GA method. The population does not converge in any region of the search space prematurely, thereby, producing a feasible solution.
- The proposed optimisation algorithm effectively deals with the epistasis phenomenon found in the cost-reliability relationship by appropriately employing improvement procedures for searching the local region of the best

solutions, during the crossover and mutation operations. As a result, the optimisation process converges rapidly towards the optimal solution without running too many generations.

- The proposed optimisation algorithm is simple to program and does not contain confusing mathematical calculations. It conducts more detailed search in each generation than the classical genetic algorithm which gradually reaches the optimum, when successful.
- The run time of the optimisation algorithm increases linearly with the complexity of the algorithm.
- The optimisation algorithm possesses a generic structure which can be configured for very large systems with a complex arrangement of components and is able to estimate minimum total loss with a corresponding level of reliability.

## **8.2 FUTURE RECOMMENDATIONS**

In view of the excellent quality of statistical results obtained by the optimisation algorithm, there appears a lot of scope for enhancing this method and configuring this for a number of other combinatorial optimisation problems. During this research activity, below are some of the key areas identified for future development for this work:

- The optimisation process can be extended for a repairable system by taking into account the cost associated with given random failures in the optimisation algorithm. For a multivariate system consisting of many

alternatives, the risk based reliability allocation can be a very difficult optimisation process which will require in-depth analysis of the failure processes and sophisticated means for modelling risk. A good source of information on this subject is provided by Todinov (2006a).

- The optimisation process can be configured to deal with multiple objectives optimisation problems for both repairable and non-repairable systems by taking into account large number of individual constraints while allocating optimum level of system reliability.
- The optimisation algorithm together with the risk based reliability can be applied to solve component assignment problems in large scale reliability systems with complex structures and many choices of available alternatives.
- The model of evolution (method of selecting parents and children) in the proposed optimisation algorithm can be analysed in greater details for streamlining the variation operators such as crossover and mutation. It may be possible to design yet more sophisticated technique for searching the local region of the best solution by integrating a more structured process of genes evaluations. This, however, can be an intricate process as it may involve testing large number of different optimisation problems and introducing complex calculations possibly using computer programming in the optimisation algorithm.
- The optimisation algorithm can be configured to solve many other widely known combinatorial optimisation problems such as ‘Knapsack Problem’, ‘Travelling Sales Man Problem’ and more importantly, in the supply-chain and demand-chain environments where optimisation of the process life-cycle is pivotal for the future growth and profitability. The algorithm can be used for optimal selection of suppliers and designing distribution networks.

Similarly for a finance industry, the structure of the optimisation algorithm renders the potential of solving complex portfolio optimisation problems where a portfolio is consisting of large number of trades belonging to various different asset classes with different levels of associated risks. The optimisation technique along with Monte Carlo simulations can also be configured to possibly price complex financial products which are used in derivative trading environment. For example, a CDO (Collateralised Debt Obligation) consisting of a pool of various individual securities (bonds, loans etc.), which is a very difficult instrument to price and hedge due to its complex structure which resembles the type of optimisation problems solved in this research.

The conjectures listed above as the future recommendations of the research work require detailed analysis for their evaluations. The interdisciplinary application of the research methodology provides great flexibility in assessing these conjectures and it is hoped that many useful and interesting results based on the proposed optimisation method, will be highlighted in the future publications.



# APPENDIX

# I

## GENERAL RESULTS & ALGORITHMS

---

### I.1 RELIABILITY & RISK ALGORITHMS

#### I.1.1 Method One

The algorithm which is used principally in this research for the purpose of evaluating system reliability and associated amount of total losses (sum of reliability investment and risk of failure) is designed by Todinov (2006, 2006a). Based on the sophisticated technique of Monte Carlo sampling, the algorithm provides powerful approach for dealing generically with all kinds of complex reliability systems. Besides, it is very easy to program and runs in non-exponential time with linear complexity.

##### I.1.1.1 Main Features

- The reliability system which is studied using this algorithm is first transformed from a reliability block diagram into an adjacency matrix, which details the number of nodes and the types of connection which may be associated with these nodes. The accuracy of the algorithm depends significantly on the correct construction of this matrix.
- A very clever technique of using node-stacking is implemented which orchestrates the navigation through the adjacency matrix in searching

for a valid path. The process provides control by keeping the log of all nodes which have already been visited and those which are still in a queue for examination.

- A Monte Carlo simulation is conducted which estimates the failure of components, risk of failure and the total loss amount, in each sample run. During each sample run of the MC simulation, the objective is to find a valid path between the start node and the end node of the adjacency matrix. The search of the path begins by checking the immediate neighbouring nodes of the start node and continuing in the direction of the nodes where the connection between the two corresponding nodes exists uninterrupted until the end node is found. If no path exists, the system is deemed in a failure mode and the cost of failures of each of the failed component is determined. The number of system failures and the total cost of failures of all components are aggregated across all sample runs. At the end of the MC simulations, the system reliability and risk of failure are obtained from the total number of failures and the cumulative cost of failures. The latter is also used to estimate the amount of total losses by simply adding the cost of reliability investment into this amount.

### **I.1.2 Method Two**

The second method, found incidentally, for determining the system reliability and total losses is very similar to the method provided by Todinov. However it is different in the sense that the path between start node and end node is established without using the node-stacking technique. Also, the search of the path in the adjacency matrix is performed in reverse order by checking the existence of the immediate valid path between start and end node and gradually moving backward until a full connection is established. If no connection is found, the system is considered in a fail state. The procedures for computing the system reliability and total loss amount are similar to the first method. The detail of this method is provided in the next section.

### I.1.3 The Algorithm

The method for tracing path is explained below:

Let  $i = \text{Row}$ ,  $j = \text{Column}$  of the adjacency matrix such that  $i = 1, 2, \dots, Z$  and  $j = 1, 2, \dots, Z$  where  $Z = \text{Total number of nodes in a system}$  .

Set  $START\_NODE = 1, END\_NODE = Z$  .

Step (1): **Initialise**  $Sample\_Run = 0$ ;

Step (2): **Make** Copy of the original Adjacency Matrix

Step (3): **Start** by initialising,  $i = START\_NODE$ ; and  $J = END\_NODE$ ;

Step(4): **Set**  $Sample\_Run = Sample\_Run + 1$ ;

Step(5): **Store**  $VarOne = Matrix(i, j)$ ;

**IF**  $VarOne = 0$ ; (*No connection Found*)

{  $j = j + 1$ ;

**IF** ( $j = START\_NODE$ ) & ( $i = START\_NODE$ )

{System Failure Counter ++;

Terminate and return Zero (System Failure);

Goto step (6) ;}

**ELSE IF** ( $j = START\_NODE$ )

{  $j = 1$ ; (*To search other nodes connected to start node*)

$i = START\_NODE$ ;

Goto step (5) ;}

**ELSE**

Goto step (5);

}

---

```

ELSE
{
    SET  $Matrix(i, j) = 0$ ; ; (setting the current node to zero)
    SET  $Matrix(j, i) = 0$ ; (setting the opposite node to zero)
    CHECK Component Failure at Current Node ( $Matrix(i, j)$ );
    IF (FAILED) (Remove inactive node)
        {SET  $Matrix(i \rightarrow Z, j) = 0$ ; (set all value of column j to zero)
           $j = j - 1$ ; ;
          Goto step (5) ;}
    IF ( $j = END\_NODE$ ) (Path is found)
        {System Success Counter ++;
          Terminate and return One (System Success);
          Goto step (6) ;}
    ELSE
        {  $i = j$ ;
           $j = END\_NODE$ ;
          Goto step (5) ;}
}

```

Step (6): Repeat Simulation Run at (2) until Sample Size (total number of simulations)

Step (7): Deduce system reliability from the failure or success counters.

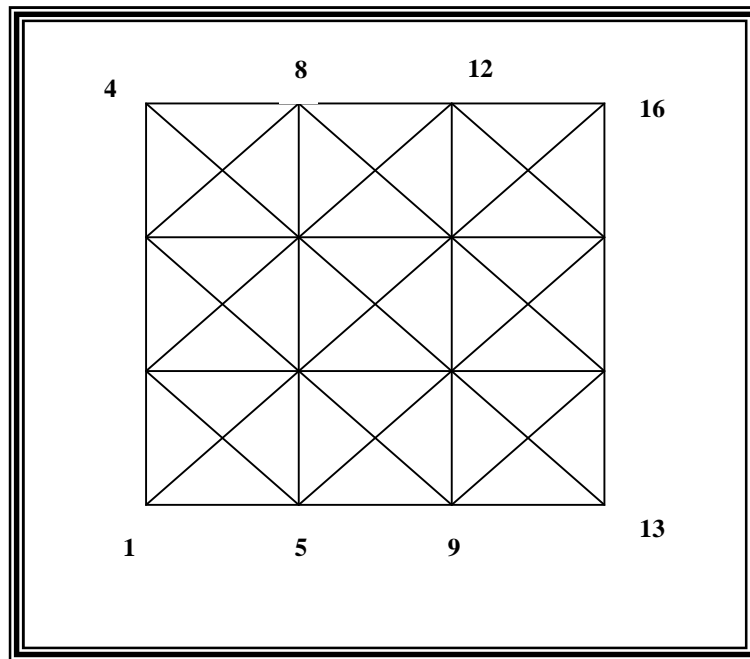
It is useful to point out that all optimisation problems studied in this research have also been tested with this method. However, it has not been tested for problems which exist outside the scope of this research but given the excellent quality of the comparative results with the first method, it is deemed as a great potential for future studies.



### I.1.3.1. Reliability of a complex lattice

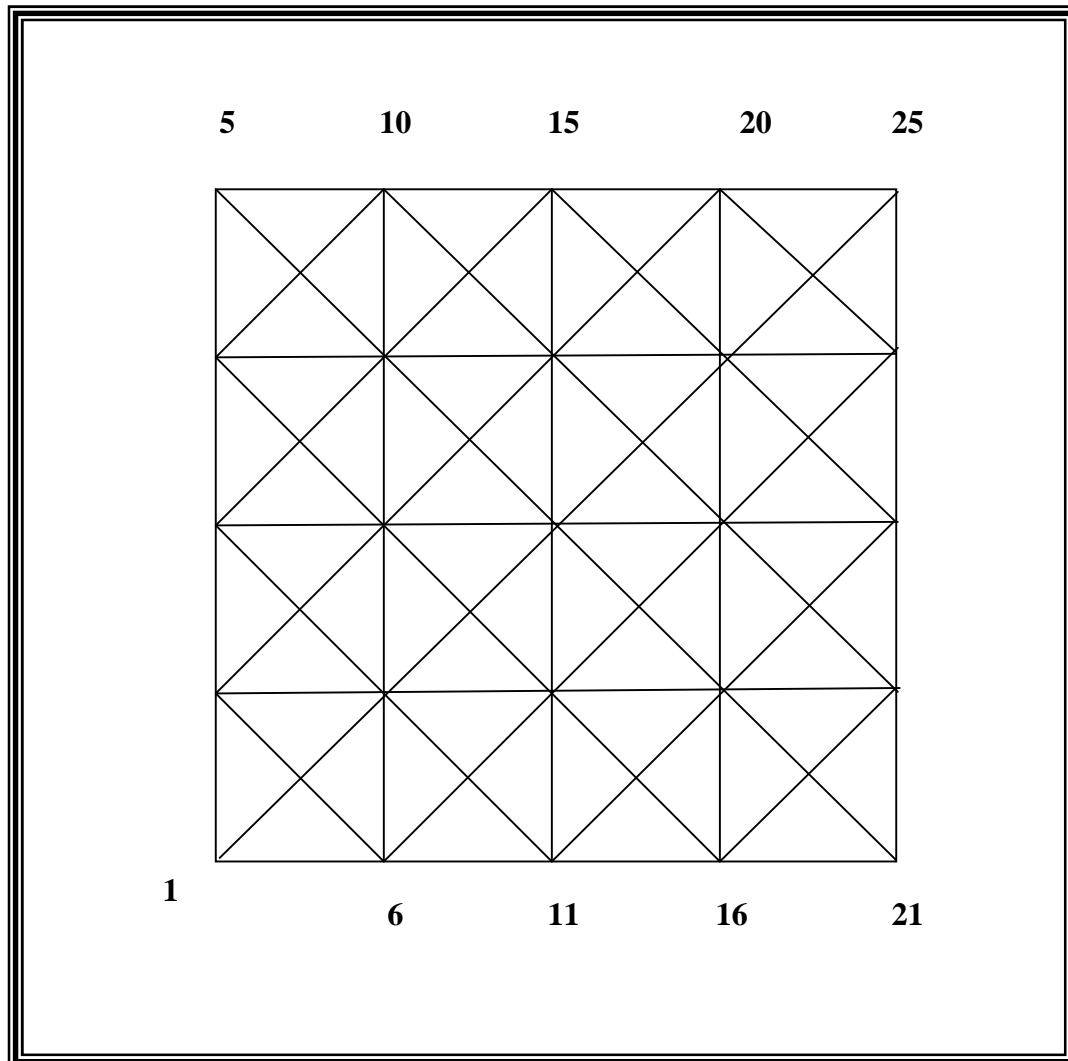
The algorithm detailed above has been applied on a complex lattice for verifying the system reliability values. Detailing this, for a given hazard rate of 0.5/yr and reliability period of 2 yr, system reliability of a complex lattices is calculated as below,

#### Lattice with 16 Nodes



#### SYSTEM CALCULATIONS

Sample Size	Number of Nodes	Hazard Rate (Per Years)	Reliability Period	System Reliability	Failure Probability
100,000	16	0.5	2.0 years	39.6%	60.4%

**Lattice with 25 Nodes****SYSTEM CALCULATIONS**

Sample Size	Number of Nodes	Hazard Rate (Per Years)	Reliability Period	System Reliability	Failure Prob
100,000	25	0.5	2.0 years	37.7%	62.3%

These results have been verified by TOTAL Ltd (France). An excellent text on various system reliability computation methods is provided in Todinov (2006a).

## I.2 APPLICATION OF THE CUT-SET AND TIE-SET SOFTWARE

### I.2.1 Basic Concepts

A typical system not having a series/parallel structure is the bridge-type network as shown in Fig.I.1. It is a common system that is frequently used to demonstrate techniques for complex systems and one that can occur often in many engineering applications (Billinton *et al.*, 1992; Ramakumar, 1993)

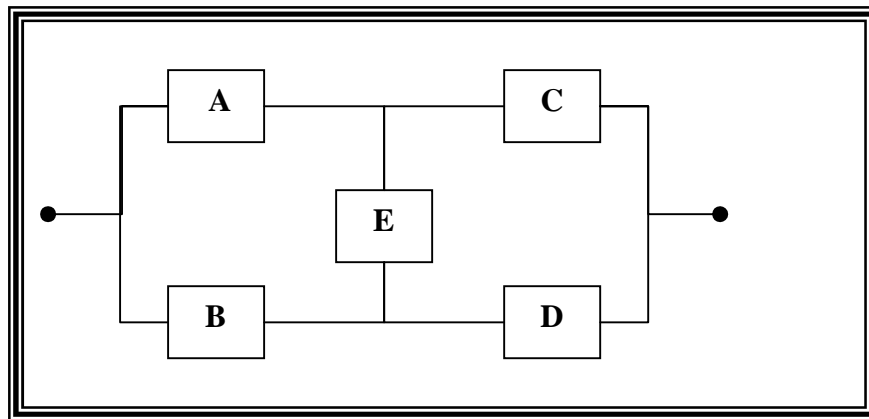


Figure I.1 Bridge network

Clearly, the components are not connected in a simple series/parallel arrangement. In order to determine the reliability of this type of network, there are a number of techniques available such as conditional probability approach, cut and tie set analysis, tree diagrams, logic diagrams and connection matrix techniques (Billinton *et al.*, 1992; Ramakumar, 1993; Todinov, 2006a).

The software developed in this research uses the cut and tie set analysis approach based on the algorithms of Fotuhi-Firuzabad *et al* (2004) and Allan *et al.* (1976), respectively.

### I.2.1.1. Cut Set and Minimal Cut Set

A cut set is a set of system components in a given reliability network or block diagram which, when failed, causes failure of the system. The minimum subset of any given set of components which causes system failure is known as a 'minimal cut set'. Therefore, a minimal cut set is a set of system components which, when failed, causes failure of the system but when any one component of the set has not failed, failure does not occur.

The cut set method is a powerful method for evaluating the reliability of a system. The main advantages being:

- It can be programmed for fast and efficient solution of many general networks but can be computationally intensive for very large systems with complex structures.
- Most importantly, many distinct ways in which a system could fail (modes of failure) can be evaluated using the cut sets approach

Assuming component statistical independence, and denoting the probability of failure of a cut set ' $c_i$ ' by ' $P(\overline{c_i})$ ', the probabilities of the system failure for ' $m$ ' minimal cut sets can be expressed as:

$$P_f = P(\overline{c_1} \cup \overline{c_2} \cup \overline{c_3} \cup \dots \cup \overline{c_m}) \quad (\text{I. 1})$$

And the reliability is

$$R = 1 - P_f = 1 - P(\overline{c_1} \cup \overline{c_2} \cup \overline{c_3} \cup \dots \cup \overline{c_m}) \quad (\text{I. 2})$$

### I.2.1.2. Minimal Tie Set

A minimal tie set is a group of components which forms a connection between the input and the output node of a reliability network, when traversed in the direction of the flow, with no node encountered more than once. The tie set method is essentially the complement of the cut set method. It is used less frequently, as it does not directly identify the failure modes of the system. Assuming component statistical independence and denoting the probability of occurrence of the tie set ' $T_i$ ' by ' $P(\overline{T_i})$ ', the reliability of the system with ' $n$ ' minimal tie-sets can be expressed as:

$$R = P(\overline{T_1} \cup \overline{T_2} \cup \overline{T_3} \cup \dots \cup \overline{T_n}) \quad (\text{I. 3})$$

### I.2.1.3. Connection Matrix (Adjacency Matrix)

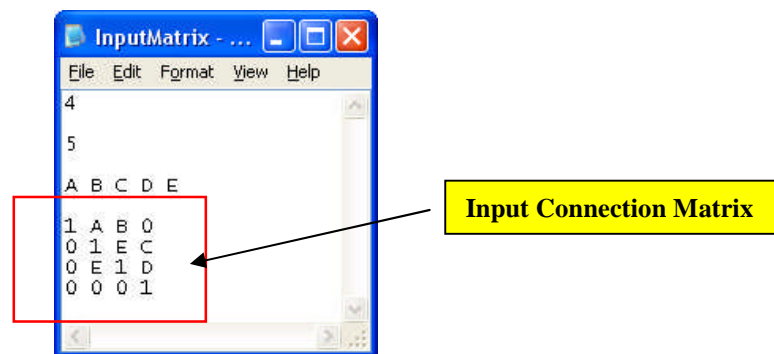
A connection matrix (adjacency matrix) is a formal method of representing reliability network or block diagram. It defines which components are connected between the nodes of the network. A zero in the matrix indicates no connection between corresponding nodes, and unity represents a connection between a node and itself, this being the value of the elements on the principal diagonal. For the network above, the connection matrix can be constructed as below:

		Nodes To			
		1	2	3	4
Nodes From	1	1	$A$	$B$	0
	2	0	1	$E$	$C$
	3	0	$E$	1	$D$
	4	0	0	0	1

The software developed in this research, takes such a connection matrix as an input in order to produce the required tie-sets and cut-sets of the system. For comprehensive literature on the above topics, refer to Billinton *et al.*, (1992), Ramakumar, (1993) and Andrews & Moss (2002).

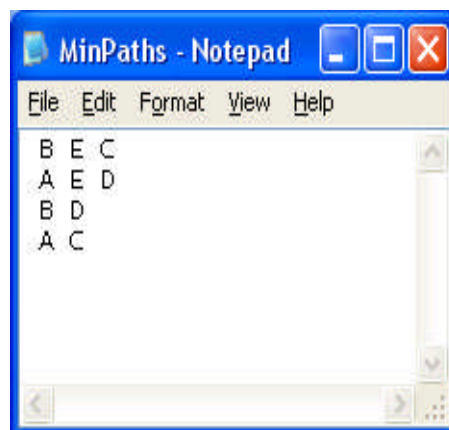
#### I.2.1.4. Reliability Evaluation of Bridge Network Using Cut Sets and Tie Sets

Using the bridge network from Fig. I.1, the software tool can be used to calculate system tie-sets and cut-sets. Firstly, a connection matrix is constructed using the reliability block diagram of the given system. The matrix is then entered in a file which is read by the software program.

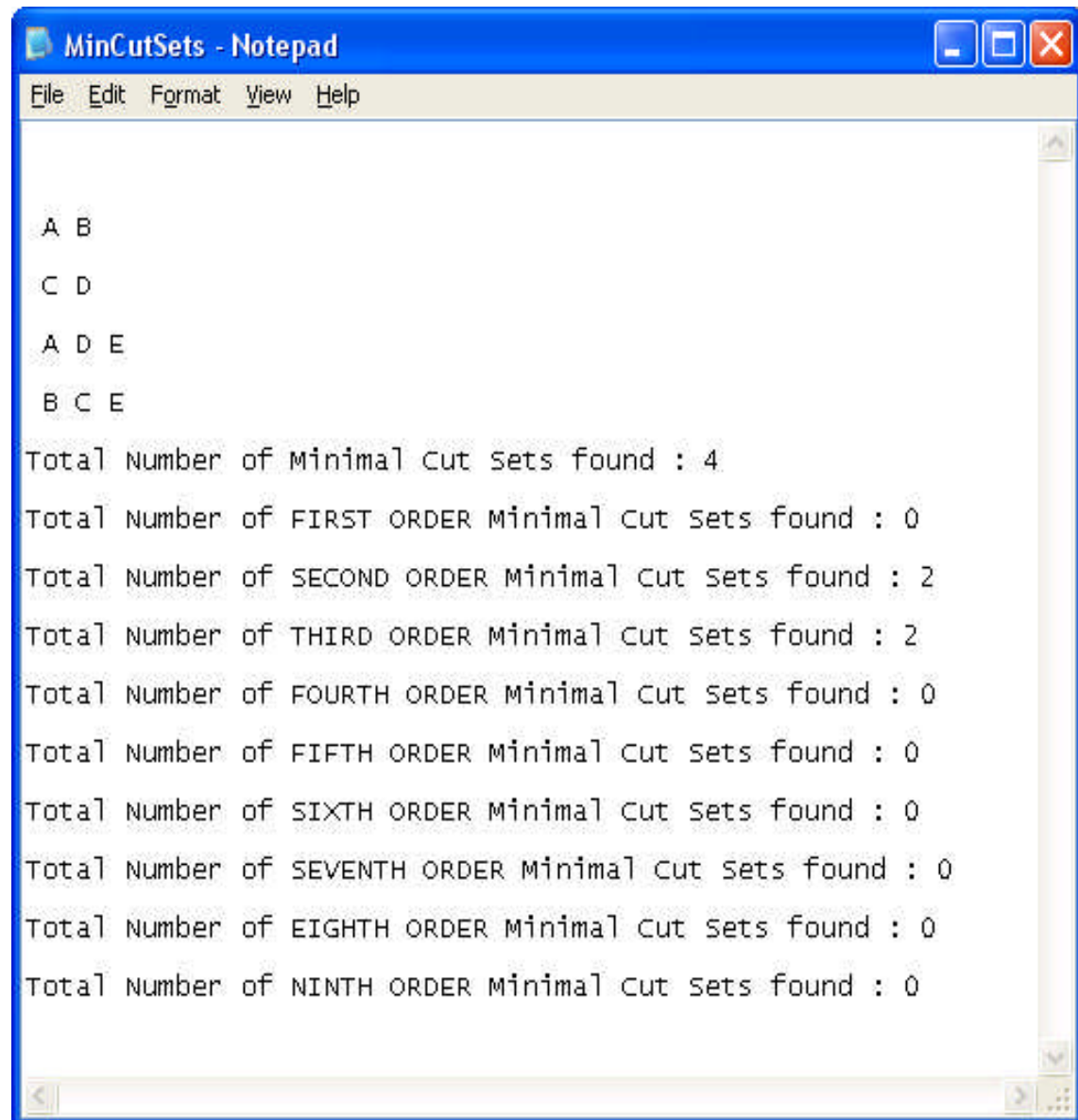


Upon successfully reading the input file, the program outputs the following data:

- File containing the tie sets of the system



- File detailing the full summary of the cut sets, up to and including order nine

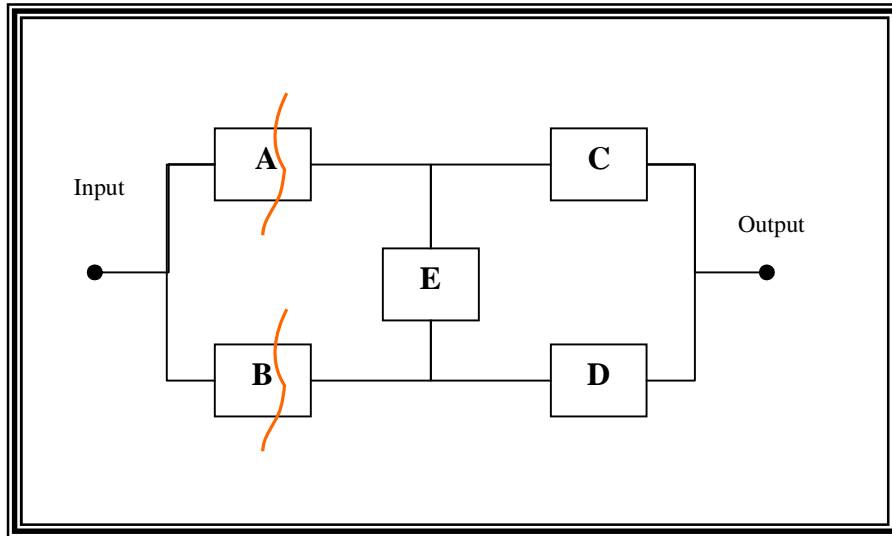


```
MinCutSets - Notepad
File Edit Format View Help

A B
C D
A D E
B C E

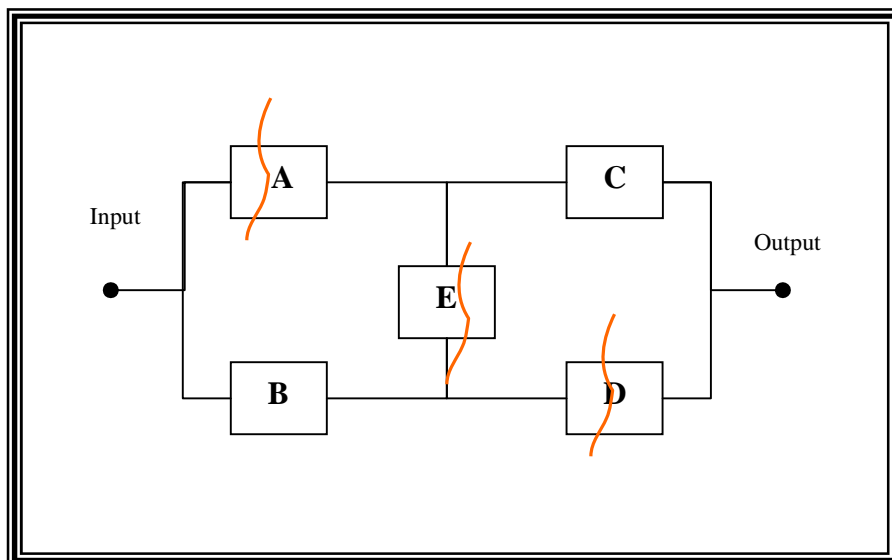
Total Number of Minimal Cut Sets found : 4
Total Number of FIRST ORDER Minimal Cut Sets found : 0
Total Number of SECOND ORDER Minimal Cut Sets found : 2
Total Number of THIRD ORDER Minimal Cut Sets found : 2
Total Number of FOURTH ORDER Minimal Cut Sets found : 0
Total Number of FIFTH ORDER Minimal Cut Sets found : 0
Total Number of SIXTH ORDER Minimal Cut Sets found : 0
Total Number of SEVENTH ORDER Minimal Cut Sets found : 0
Total Number of EIGHTH ORDER Minimal Cut Sets found : 0
Total Number of NINTH ORDER Minimal Cut Sets found : 0
```

The output result can be easily verified manually from the given network. There appear to be two second order minimal cut sets. In other words, there are two cut sets, each with two components. If both components of any of the two cut sets, fail at once, the system will also stop working. For example, for a cut set, 'A B', if both A and B, fail simultaneously, there will be no possibility of traversing from input node to the output node.



**Figure I.2 System Failure – Second Order Cut Set**

Similarly, if all of the components in the third order cut set 'A D E' are failed, the system will also encounter the failure state.

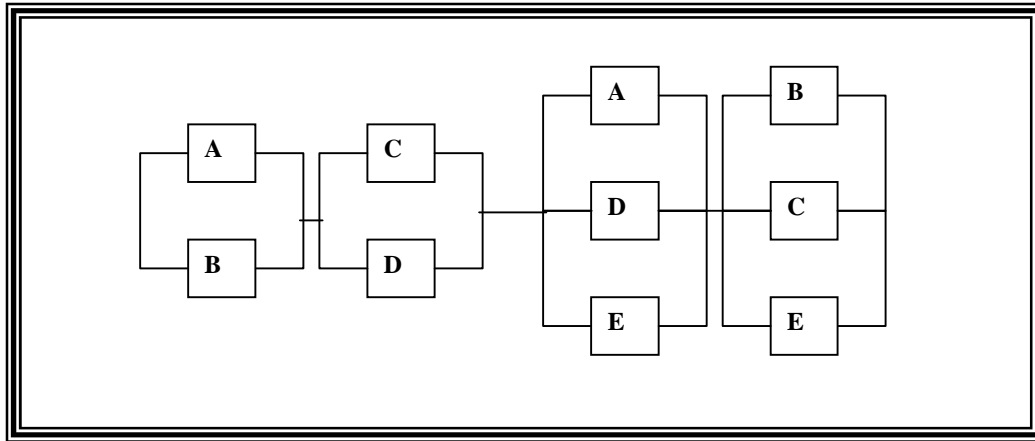


**Figure I. 3 System Failure – Third Order Cut Set**



### ***Reliability Calculation – Cut Set Method***

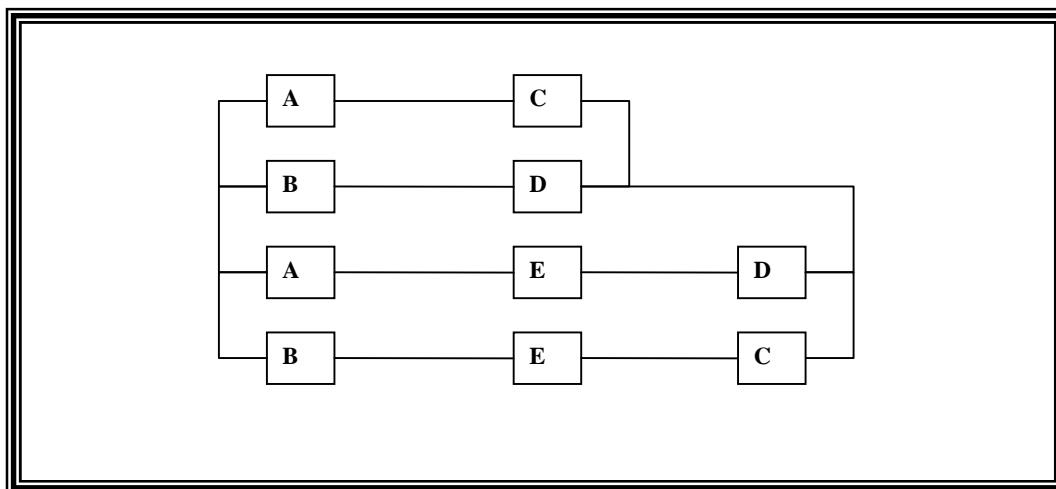
Having obtained the cut set of the given system, the reliability of the given system can now be calculated easily.



**Figure I.4 Minimal Cut Sets of Bridge Network**

### ***Reliability Calculation – Tie Set Method***

Similarly, the reliability of the bridge network can also be calculated using Eq. (9) and the minimal tie sets, obtained from the computer program.



**Figure I.5 Minimal Tie Sets of Bridge Network**

### I.2.1.5. A Real Life Production System

The software tool has also been successfully used to deduce minimal paths and minimal cut sets of a real production system. The reliability block diagram is sketched below:

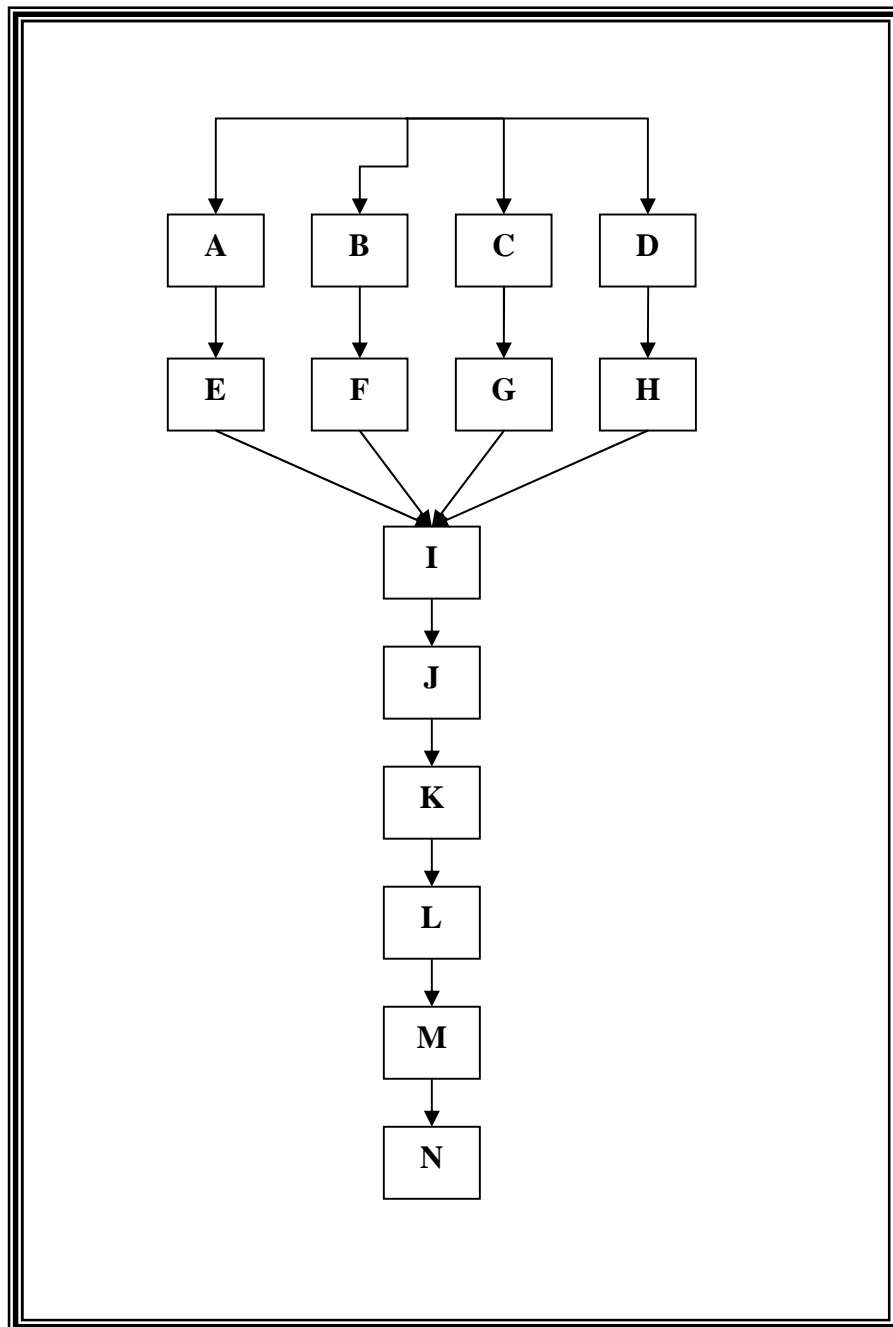
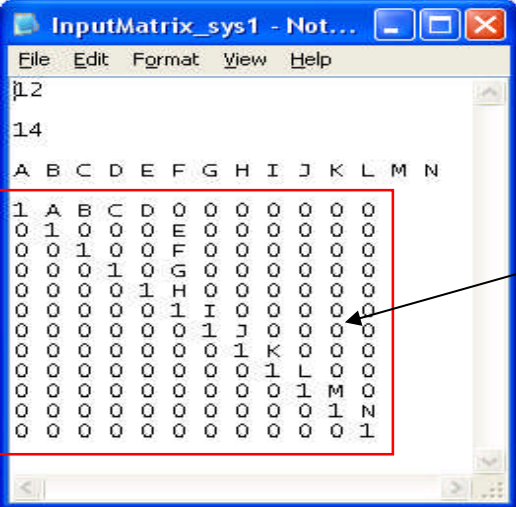


Figure I.6 Reliability Network of a Real Life Production System

The connection matrix of the system is below



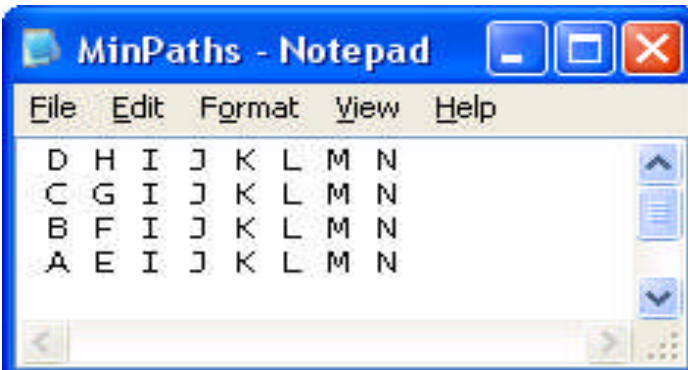
```

InputMatrix_sys1 - Notepad
File Edit Format View Help
12
14
A B C D E F G H I J K L M N
1 1 0 0 0 0 0 0 0 0 0 0 0 0
0 1 0 0 0 0 0 0 0 0 0 0 0 0
0 0 1 0 0 0 0 0 0 0 0 0 0 0
0 0 0 1 0 0 0 0 0 0 0 0 0 0
0 0 0 0 1 0 0 0 0 0 0 0 0 0
0 0 0 0 0 1 0 0 0 0 0 0 0 0
0 0 0 0 0 0 1 0 0 0 0 0 0 0
0 0 0 0 0 0 0 1 0 0 0 0 0 0
0 0 0 0 0 0 0 0 1 0 0 0 0 0
0 0 0 0 0 0 0 0 0 1 0 0 0 0
0 0 0 0 0 0 0 0 0 0 1 0 0 0
0 0 0 0 0 0 0 0 0 0 0 1 0 0
0 0 0 0 0 0 0 0 0 0 0 0 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 1
0 0 0 0 0 0 0 0 0 0 0 0 0 1
  
```

**Figure I.7 Connection Matrix of the Real Life Reliability System**

After reading the connection matrix, the software program produces the following data set:

- File containing the minimal tie sets of the system

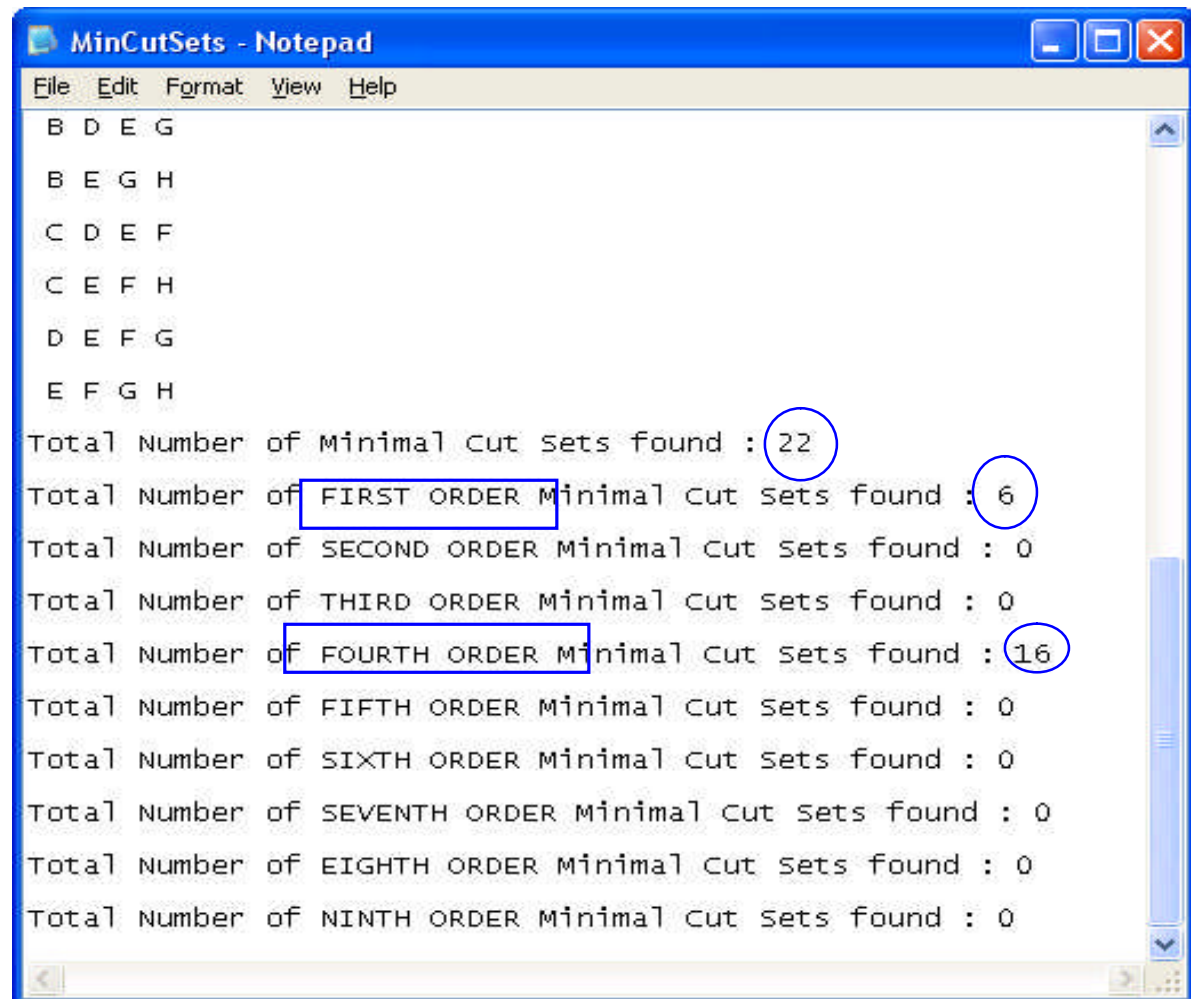


```

MinPaths - Notepad
File Edit Format View Help
D H I J K L M N
C G I J K L M N
B F I J K L M N
A E I J K L M N
  
```

**Figure I.8 Minimal Tie Sets of the Real Life Reliability System**

- File containing the summary of the minimal cut sets of the system



```
MinCutSets - Notepad
File Edit Format View Help
B D E G
B E G H
C D E F
C E F H
D E F G
E F G H
Total Number of Minimal Cut Sets found : 22
Total Number of FIRST ORDER Minimal Cut Sets found : 6
Total Number of SECOND ORDER Minimal Cut Sets found : 0
Total Number of THIRD ORDER Minimal Cut Sets found : 0
Total Number of FOURTH ORDER Minimal Cut Sets found : 16
Total Number of FIFTH ORDER Minimal Cut Sets found : 0
Total Number of SIXTH ORDER Minimal Cut Sets found : 0
Total Number of SEVENTH ORDER Minimal Cut Sets found : 0
Total Number of EIGHTH ORDER Minimal Cut Sets found : 0
Total Number of NINTH ORDER Minimal Cut Sets found : 0
```

**Figure I.9 Minimal Cut Sets of the Real Life Reliability System**

---

### **I.3 RESEARCH PUBLICATIONS**

**Publication One:** *Reliability Optimization Based on Minimizing the Total Losses*, International Conference on Reliability and Safety Engineering (*INCRESE 2005*), Indian Institute of Technology Kharagpur (India) in December 2005.

**Publication Two:** *Reliability Optimisation based on minimising the total cost using genetic algorithm*, Accepted for publication in the 17<sup>th</sup> AR<sup>2</sup>TS (Advances in Risk and Reliability Technology Symposium), Burleigh Court Conference Centre, Loughborough University, April 2007.

**Publication Three:** *An Efficient Evolutionary Algorithm for Solving Complex Reliability Optimisation Problems with Cost Constraint and Discrete Choice of Alternative Components*, International Journal of Evolutionary Optimisation (Accepted - awaiting publication)

**Publication Four:** *Risk Based Reliability Allocation in a complex system using evolutionary algorithm*, International Journal of Computers and Industrial Engineering. (Being revised)

**Publication Five:** *Optimal Topology Analysis Using Risk Based Reliability Allocation Method*, International Journal of Quality and Reliability Management. (Being revised)

.



# APPENDIX II

## OPTIMISATION RESULTS OF SYSTEM-B (FROM SECTION 6.3.2)

---

The topology of System B consists of three subsystems each containing two, five and two components, connected in parallel, respectively – Fig. II.1. The application of the optimisation algorithm using the data from Table 6.1 is described by means of various graphs showing the actual optimisation process, the effect of the genetic operations (crossover and mutation) on total loss associated with the optimal reliability allocated for this system and table detailing various sub-optimal results found along with the optimum solution.

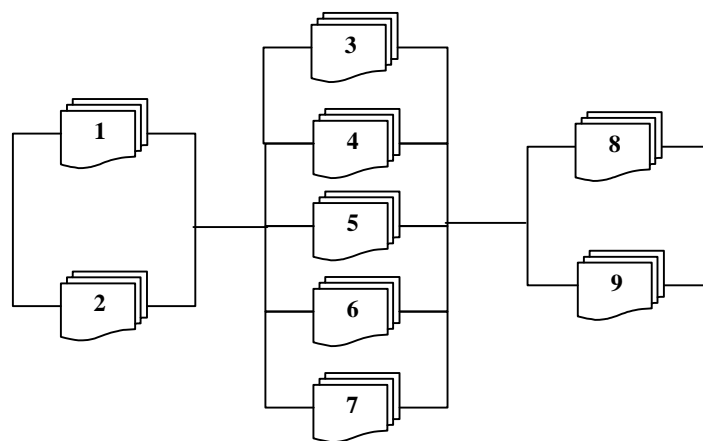
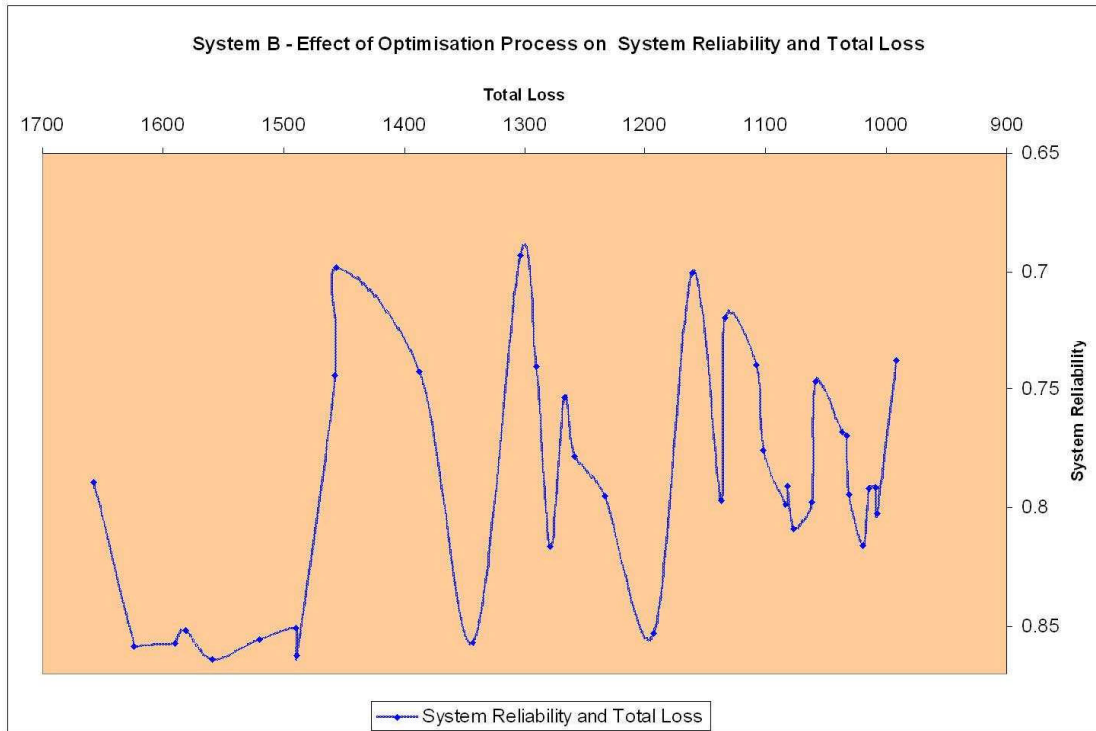


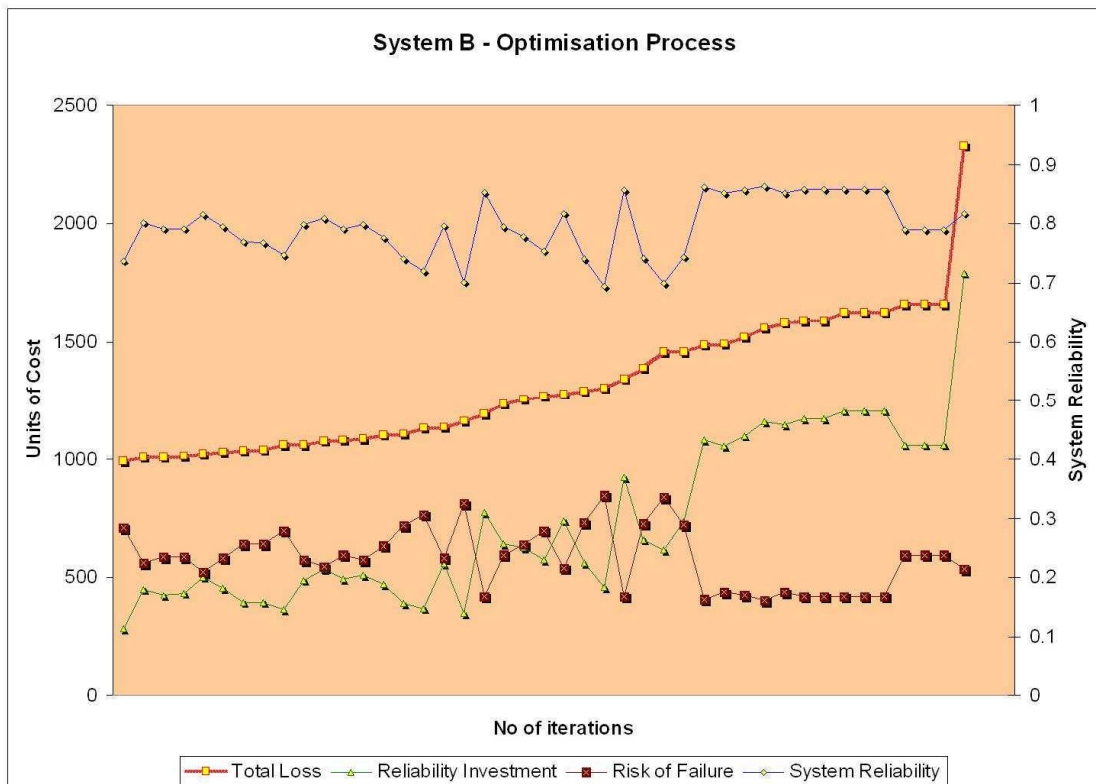
Figure II.1 Structure of System B

No.	Total Loss	System Reliability	Reliability Investment	Risk of Failure	Sub-System One		Sub-System Two					Sub-System Three	
<b>1</b>	<b>992</b>	<b>73.7%</b>	<b>283</b>	<b>708</b>	<b>5</b>	<b>6</b>	<b>2</b>	<b>2</b>	<b>4</b>	<b>3</b>	<b>2</b>	<b>4</b>	<b>4</b>
2	1008	80.2%	450	558	7	5	3	2	3	2	2	4	7
3	1019	81.6%	500	520	5	6	2	2	4	3	2	2	9
4	1037	76.8%	394	643	3	6	2	2	4	4	2	4	7
5	1133	72.0%	366	767	3	4	4	2	4	4	2	3	7
6	1161	70.0%	348	813	3	3	4	2	4	4	2	3	7
7	1193	85.3%	772	420	11	3	4	2	4	4	2	3	7
8	1234	79.5%	641	593	8	4	2	2	6	4	2	2	7
9	1259	77.8%	623	636	5	6	2	2	6	6	2	2	7
10	1267	75.4%	572	696	3	6	2	2	6	6	2	2	7
11	1279	81.7%	741	537	8	6	2	2	6	6	2	2	7
12	1291	74.0%	559	731	2	6	2	2	6	6	2	2	7
13	1343	85.7%	924	419	11	3	4	2	6	4	2	3	7
14	1387	74.3%	660	727	2	6	2	2	6	8	2	2	7
15	1456	69.8%	617	839	2	4	2	2	6	8	2	2	7
16	1457	74.4%	735	723	2	6	2	2	6	9	2	2	7
17	1590	85.7%	1171	419	11	6	2	2	6	9	2	2	7
18	1624	85.9%	1206	418	11	6	4	2	6	9	2	2	7
19	1658	79.0%	1062	596	5	3	2	5	2	7	11	3	9
<b>20</b>	<b>2327</b>	<b>81.8%</b>	<b>1792</b>	<b>535</b>	<b>3</b>	<b>5</b>	<b>10</b>	<b>9</b>	<b>5</b>	<b>2</b>	<b>11</b>	<b>10</b>	<b>5</b>

**Table II.1 List of Results Found by the Optimisation Algorithm for System A**



**Figure II.2 Effect of Optimisation Process on System Reliability and Total Loss in System B**



**Figure II.3 Optimisation Process of System B**



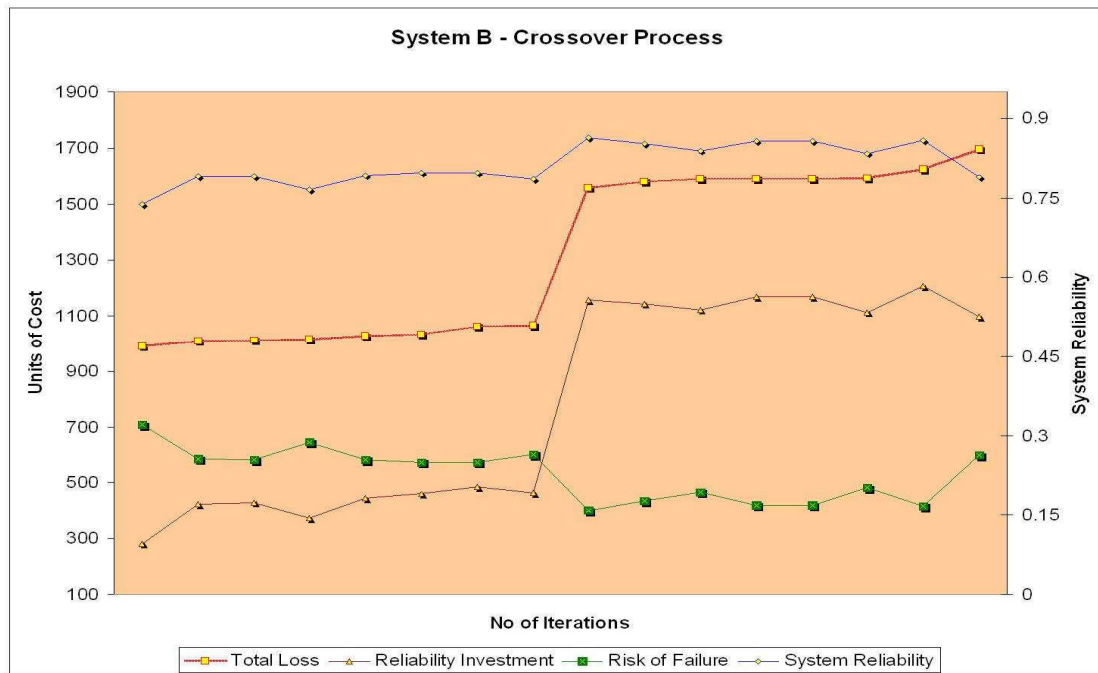


Figure II.4 Crossover Process of System B

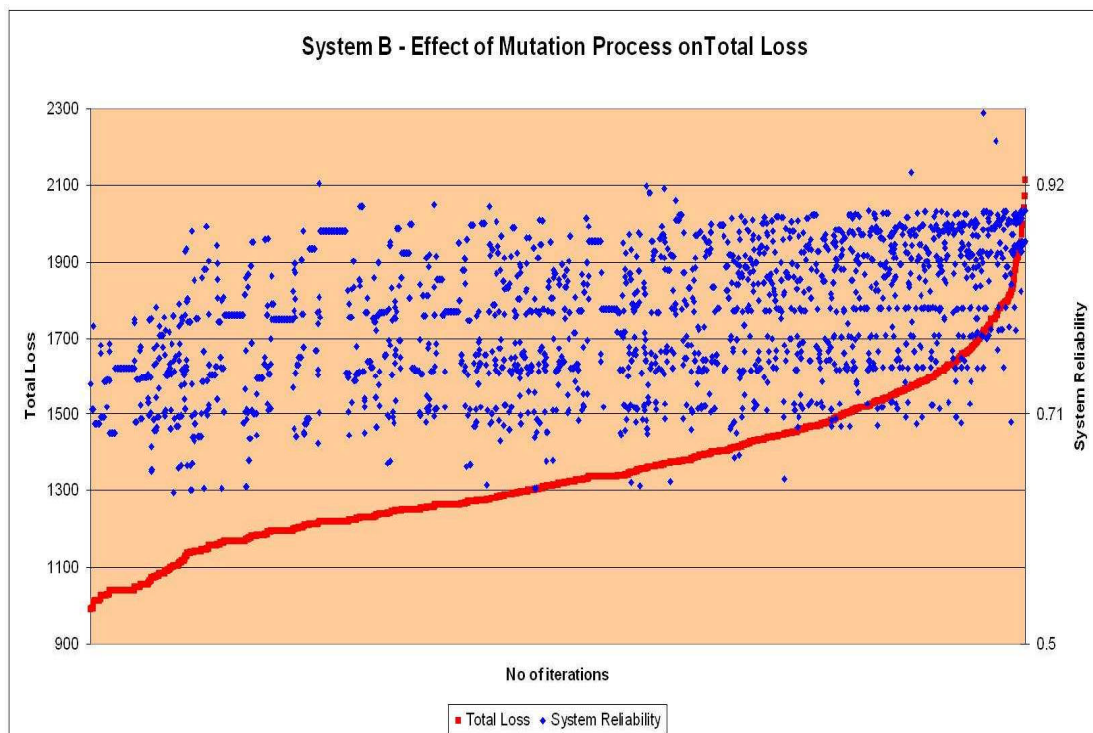


Figure II.5 Effect of Mutation Process on Total Loss in System B

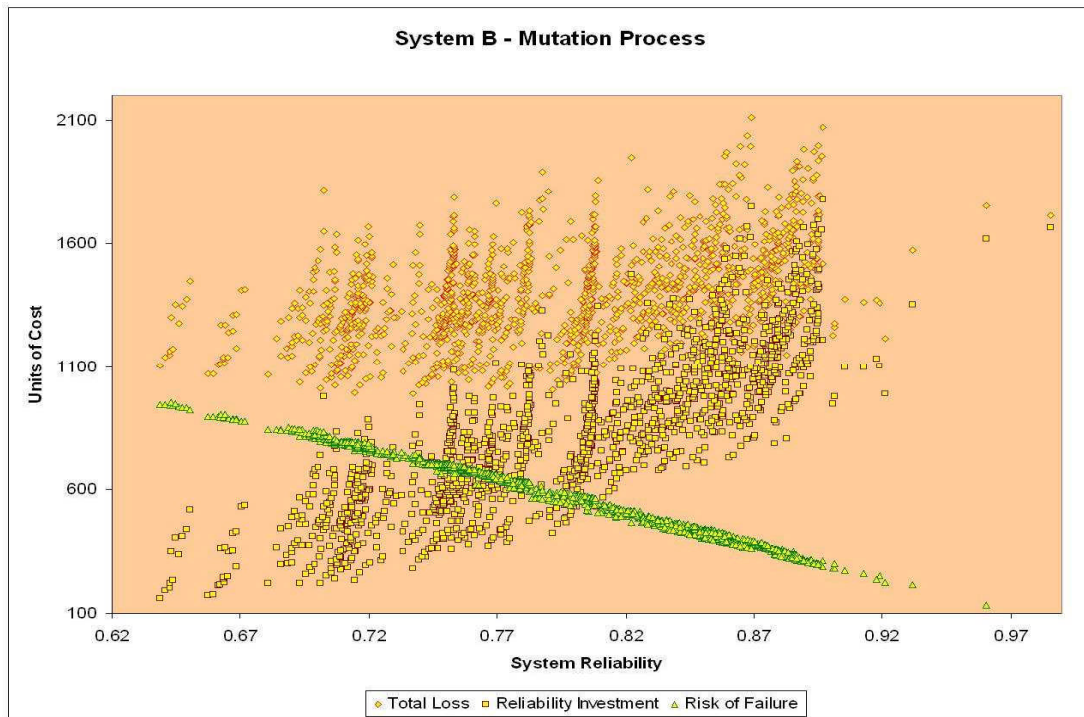


Figure II. 6 Mutation Process in System B

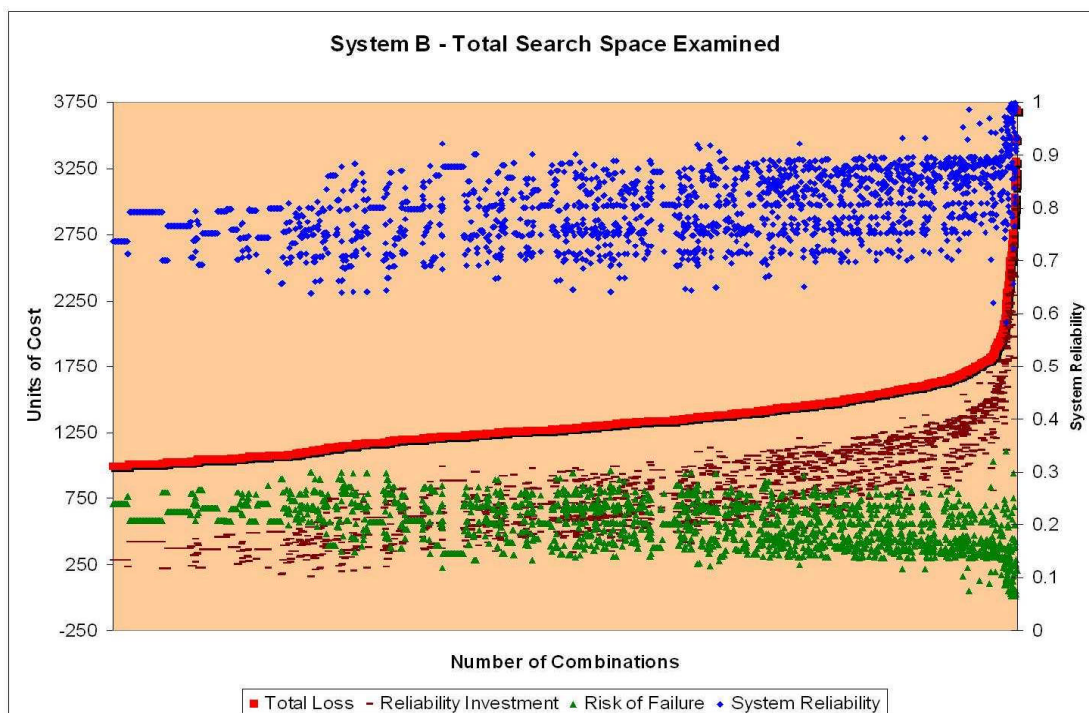


Figure II.7 Total Search Space examined By the Optimisation Algorithm for System B

# APPENDIX

# III

## OPTIMISATION RESULTS OF SYSTEM-C (FROM SECTION 6.3.3)

---

The topology of System C consists of three subsystems each containing two, five and two components, connected in parallel, respectively – Fig. III.1. The application of the optimisation algorithm using the data from Table 6.1 is described by means of various graphs showing the actual optimisation process, the effect of the genetic operations (crossover and mutation) on total loss associated with the optimal reliability allocated for this system and table detailing various sub-optimal results found along with the optimum solution.

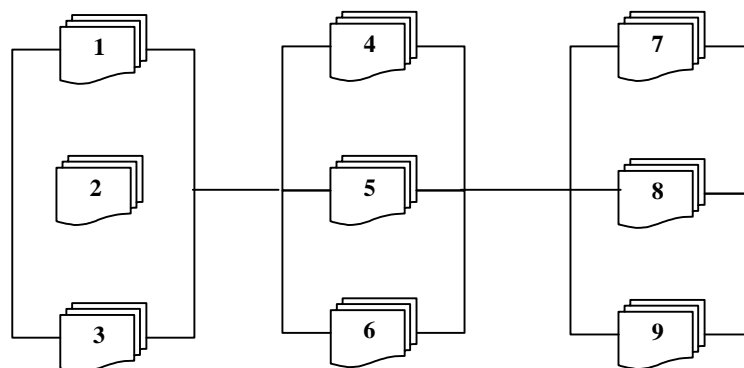


Figure III.1 Structure of System C

NO.	Total Loss	System Reliability	Reliability Investment	Risk of Failure	Sub-System One			Sub-System Two			Sub-System Three		
1	782	79.8%	242	540	3	4	3	4	3	4	4	3	4
2	798	81.3%	293	505	3	6	2	5	2	3	5	3	4
3	825	79.9%	285	540	3	4	3	3	2	7	2	3	4
4	909	86.5%	529	379	3	6	3	3	2	7	8	3	4
5	916	85.9%	520	396	3	6	3	2	2	7	8	3	4
6	961	86.6%	582	379	3	6	3	5	3	4	5	3	9
7	968	85.7%	568	400	3	6	2	6	2	3	5	3	9
8	969	86.0%	574	395	3	6	3	5	2	4	5	3	9
9	970	83.5%	515	455	3	3	4	5	2	3	5	3	9
10	974	83.6%	520	453	3	3	3	6	2	3	5	3	9
11	974	86.9%	602	372	3	6	3	6	2	4	5	3	9
12	979	81.5%	475	504	3	4	3	3	2	3	5	3	9
13	981	87.4%	624	356	3	4	4	5	2	7	9	3	4
14	989	85.7%	589	400	3	4	2	5	2	7	9	3	4
15	1000	85.6%	595	404	3	4	3	3	2	7	5	3	9
16	1001	86.1%	610	391	3	4	3	4	2	7	5	3	9
17	1146	88.9%	833	313	5	4	2	5	2	7	11	3	4
18	1378	90.1%	1093	285	5	4	3	5	2	7	11	3	9
19	1388	88.5%	1062	327	5	3	2	5	2	7	11	3	9
20	1920	96.0%	1792	128	3	5	10	9	5	2	11	10	5

Table III.1 Optimisation Results of System C

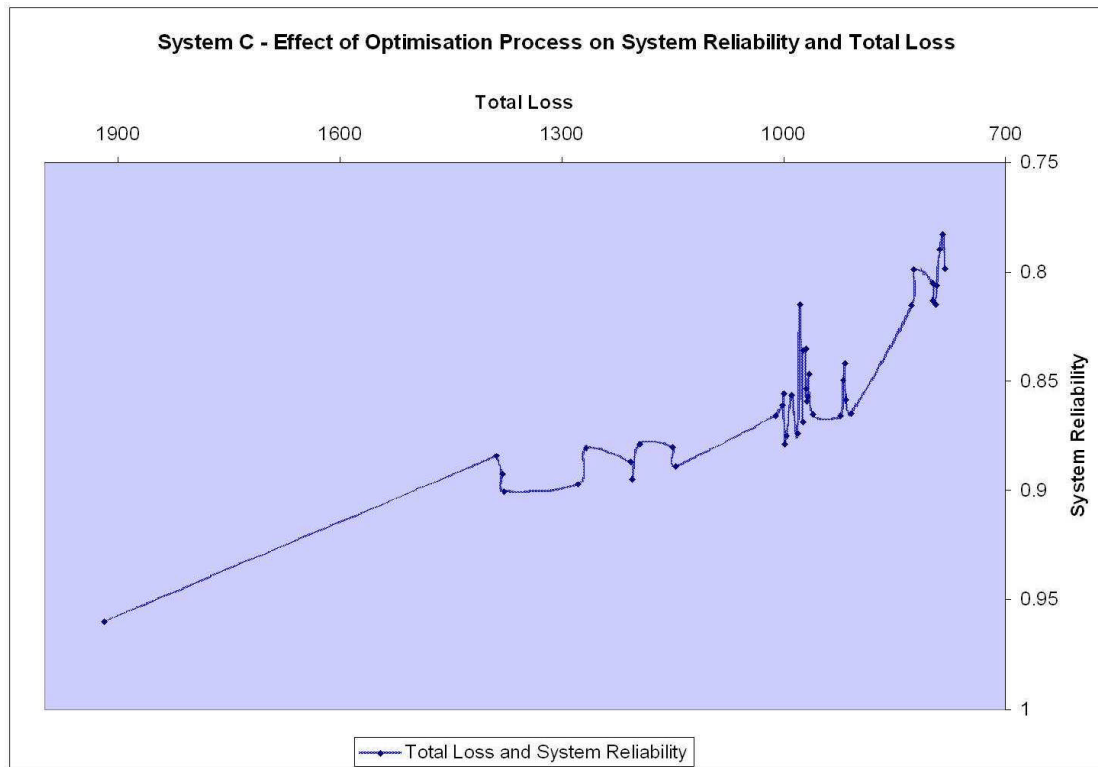


Figure III. 2 Effect of Optimisation Process on System Reliability and Total Loss in System C

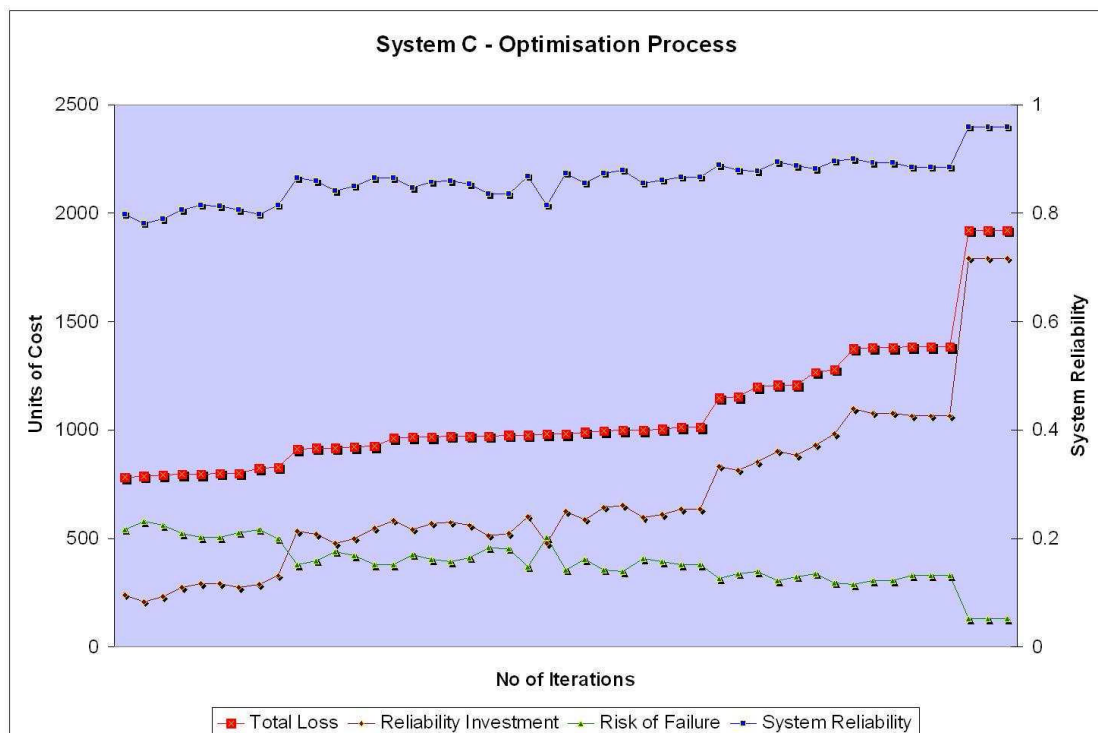


Figure III.3 Optimisation Process of System C

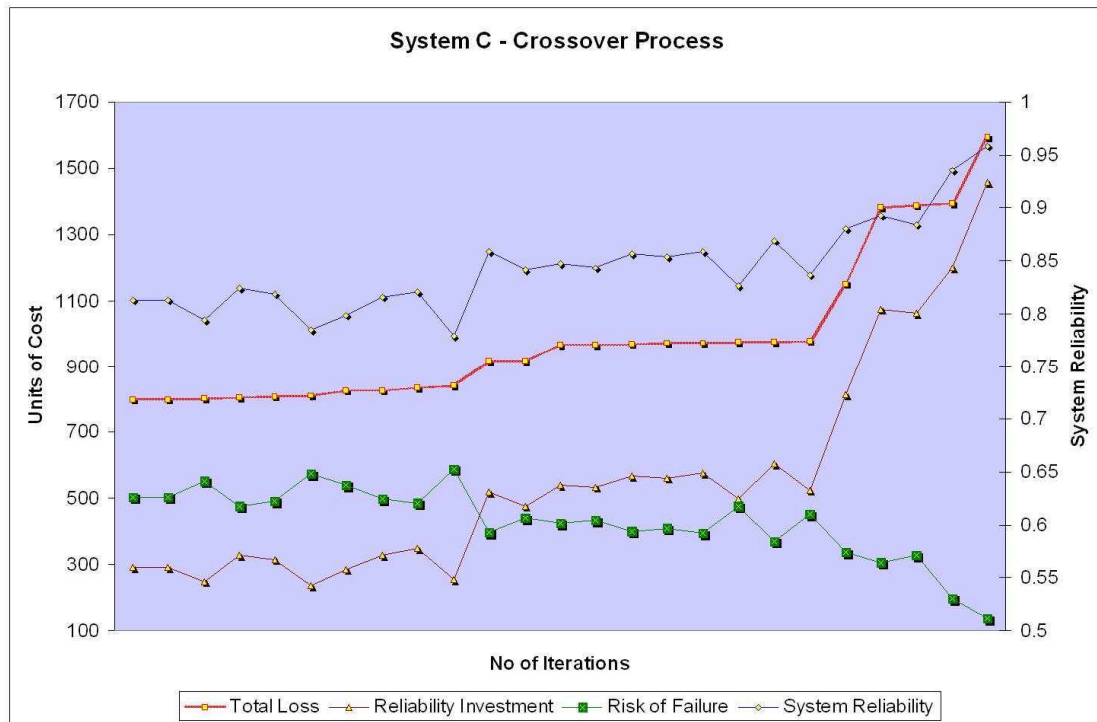


Figure III.4 Crossover Process of System C

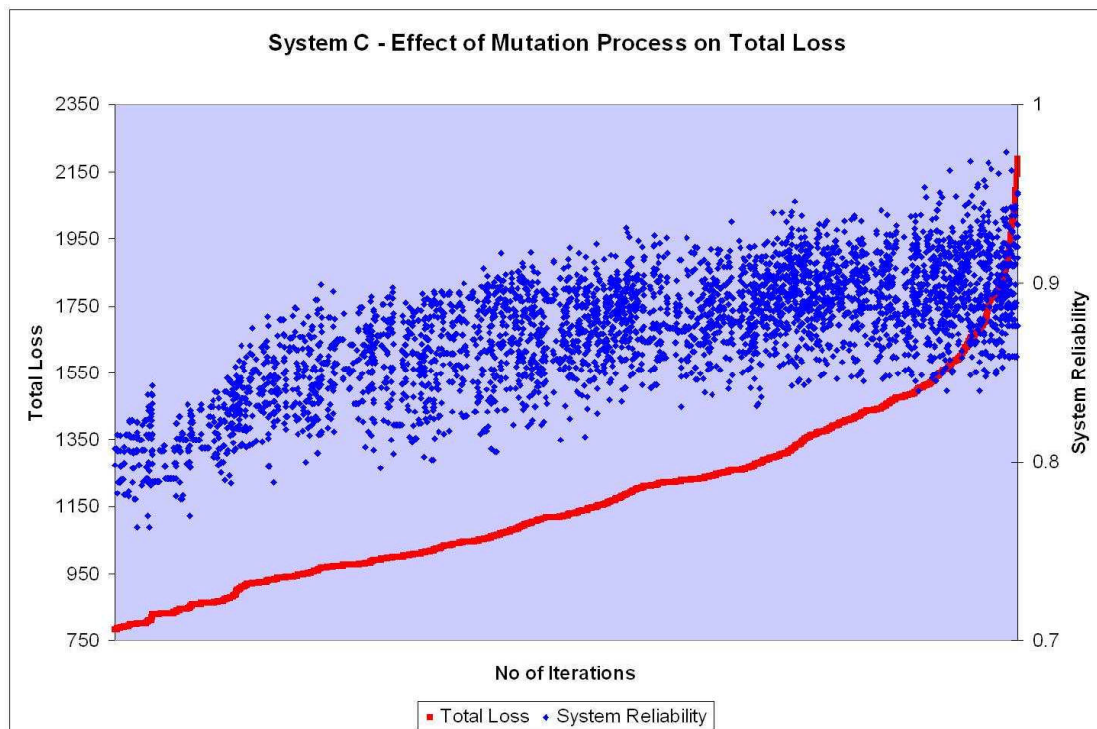


Figure III.5 Effect of Mutation Process on Total Loss in System C



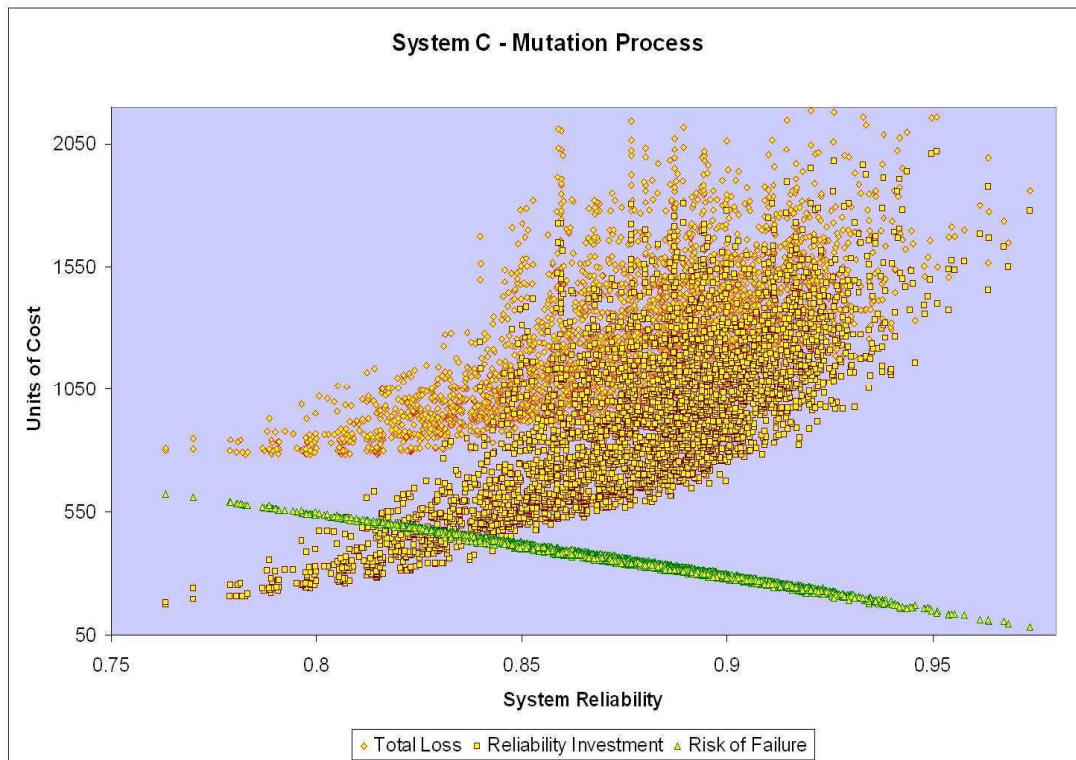


Figure III.6 Mutation Process in System

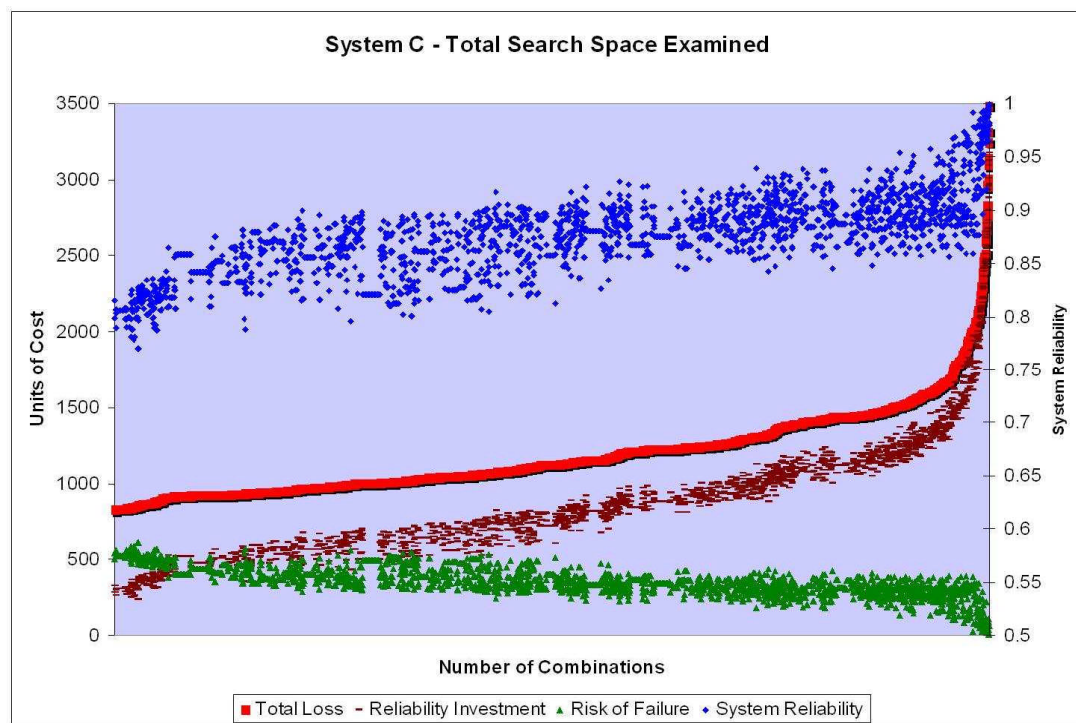


Figure III.7 Total Search Space Examined By the Optimisation Algorithm for System C

# APPENDIX IV

## OPTIMISATION RESULTS OF SYSTEM-D (FROM SECTION 6.3.4)

---

The topology of System D consists of three subsystems each containing two, five and two components, connected in parallel, respectively – Fig. IV.1. The application of the OA using the data from Table 6.1 is described by means of various graphs showing the actual optimisation process, the effect of the genetic operations (crossover and mutation) on total loss associated with the optimal reliability allocated for this system and table detailing various sub-optimal results found along with the optimum solution.

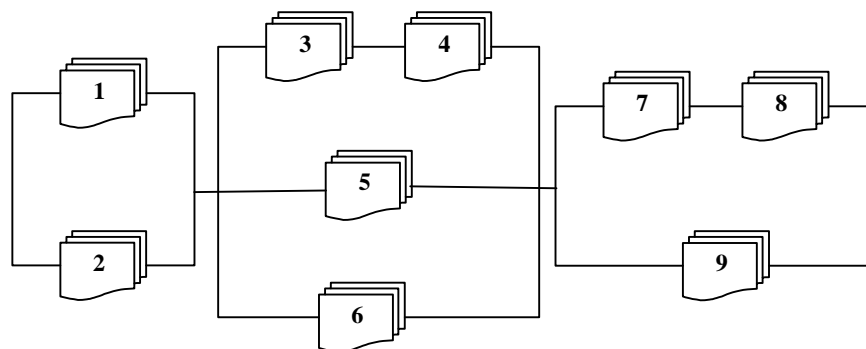


Figure IV.1 Structure of System D



NO.	Total Loss	System Reliability	Reliability Investment	Risk of Failure	Sub-System One		Sub-System Two				Sub-System Three		
1	1356	74.3%	620	736	5	6	2	2	3	7	3	3	9
2	1362	66.4%	434	928	3	6	2	2	2	6	2	2	8
3	1364	69.8%	513	850	6	6	2	2	2	6	2	2	8
4	1368	75.5%	669	699	6	6	2	2	2	6	2	2	10
5	1377	73.8%	639	738	6	6	2	2	2	5	2	2	10
6	1379	70.3%	559	820	3	6	2	2	2	5	2	2	10
7	1399	74.3%	659	740	5	6	2	5	2	7	3	3	9
8	1411	75.2%	694	718	6	6	2	2	3	6	6	3	9
9	1426	66.7%	517	909	3	4	2	2	2	5	2	2	10
10	1448	64.9%	498	950	3	3	2	2	2	5	2	2	10
11	1451	74.9%	725	727	5	6	2	4	2	7	6	3	9
12	1478	63.0%	484	994	3	2	2	2	2	5	2	2	10
13	1521	66.9%	616	906	3	2	2	2	2	8	2	2	10
14	1535	87.3%	1158	377	11	2	2	2	2	8	2	2	11
15	1560	87.8%	1193	366	11	2	4	2	2	8	2	2	11
16	1592	89.5%	1268	324	11	2	4	2	2	9	2	2	11
17	1627	76.1%	930	697	5	6	2	5	2	7	9	3	9
18	1704	76.5%	1026	678	11	6	4	2	2	9	2	2	7
19	1843	72.4%	1062	781	5	3	2	5	2	7	11	3	9
20	2467	77.3%	1792	675	3	5	10	9	5	2	11	10	5

Table IV.1 Optimisation Results for System D

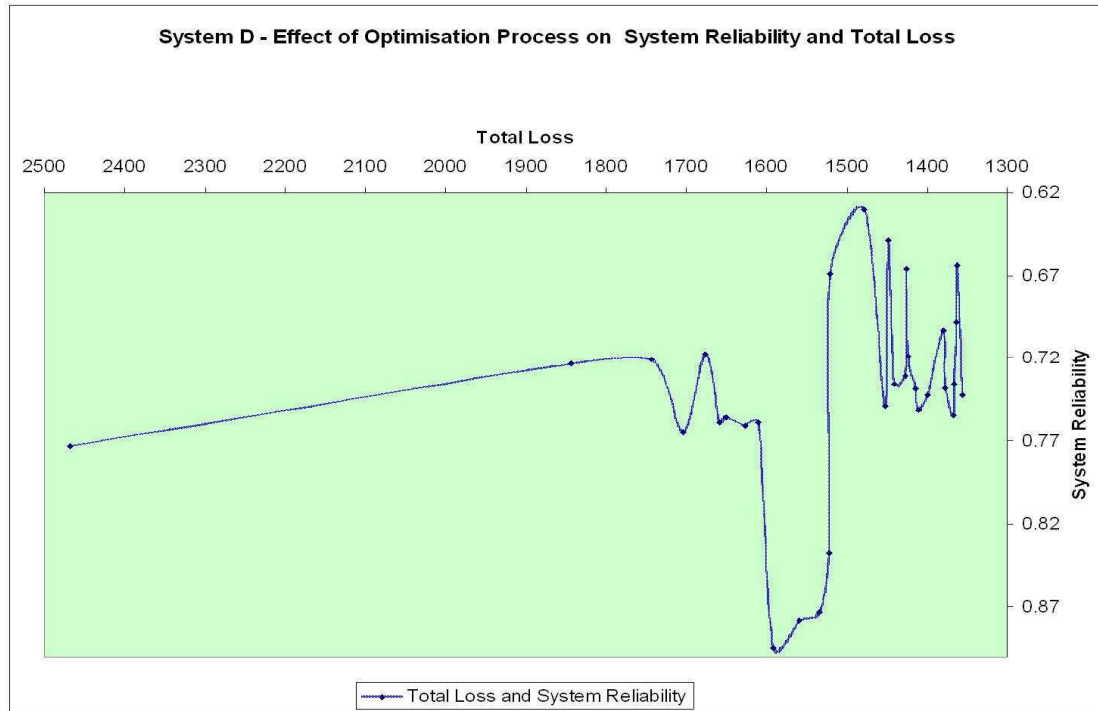


Figure IV. 1 Effect of Optimisation Process on System Reliability and Total Loss in System D

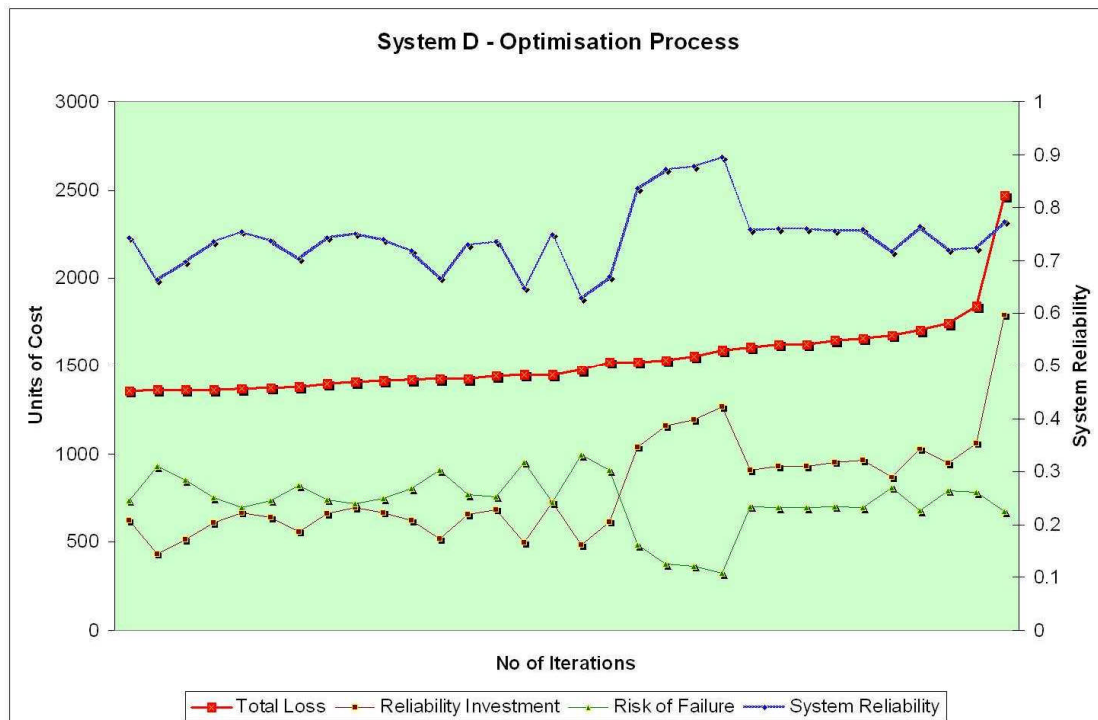


Figure IV.2 Optimisation Process of System D

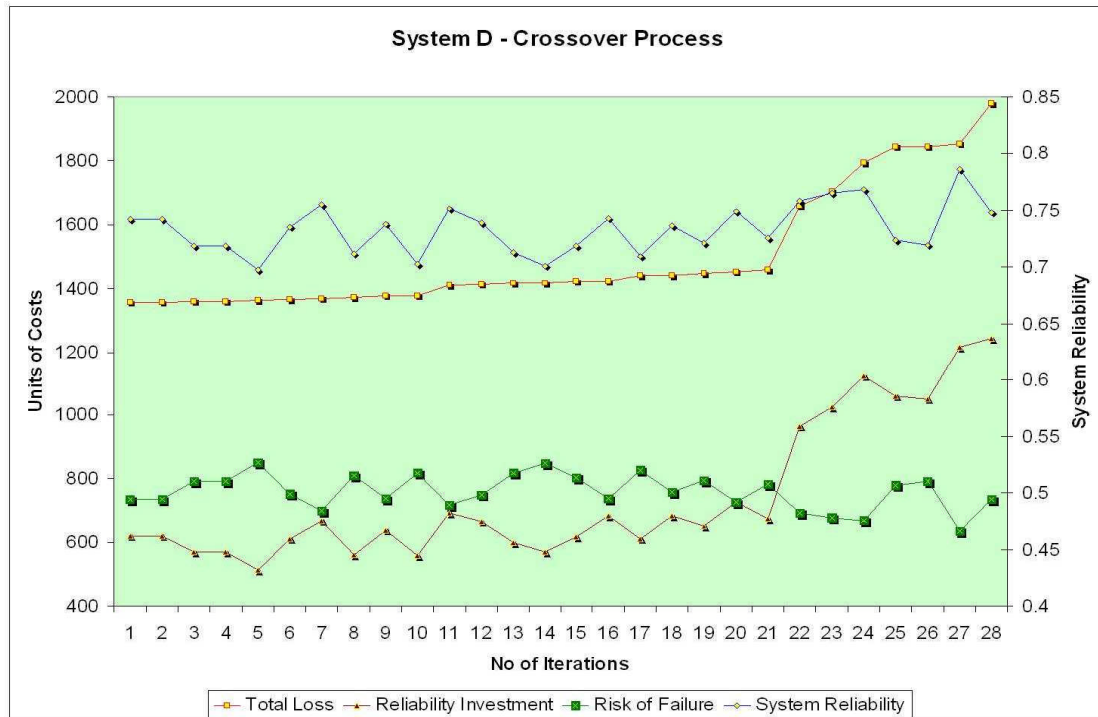


Figure IV.3 Crossover Process of System D

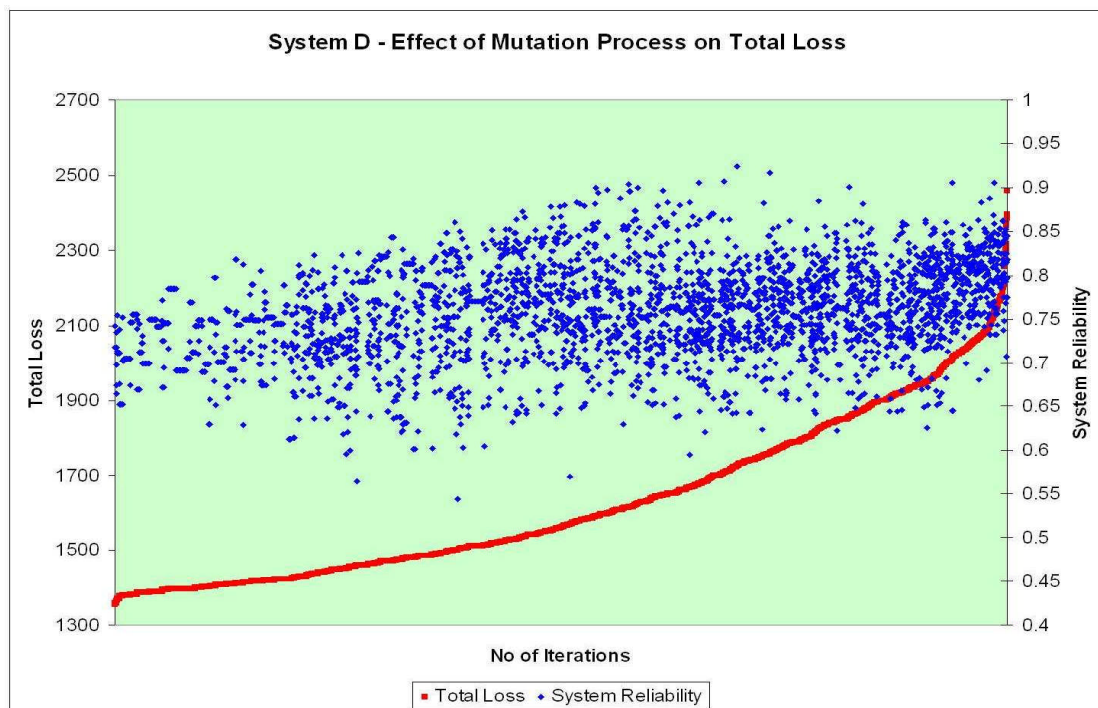


Figure IV.4 Effect of Mutation Process on Total Loss in System D

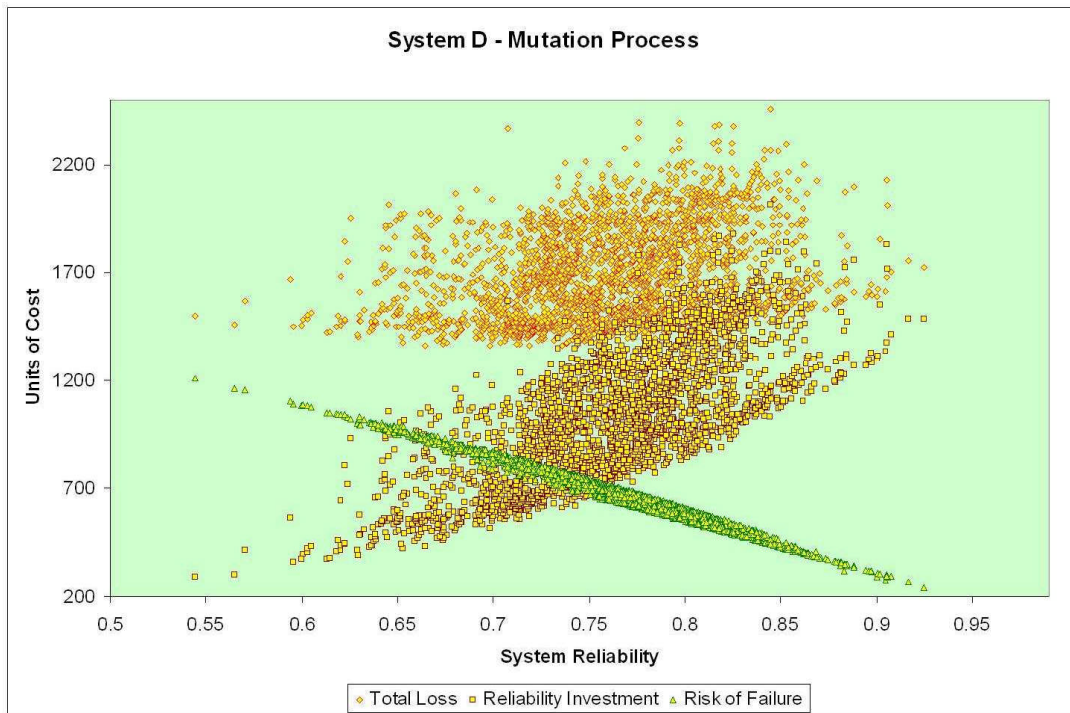


Figure IV.5 Mutation Process in System D

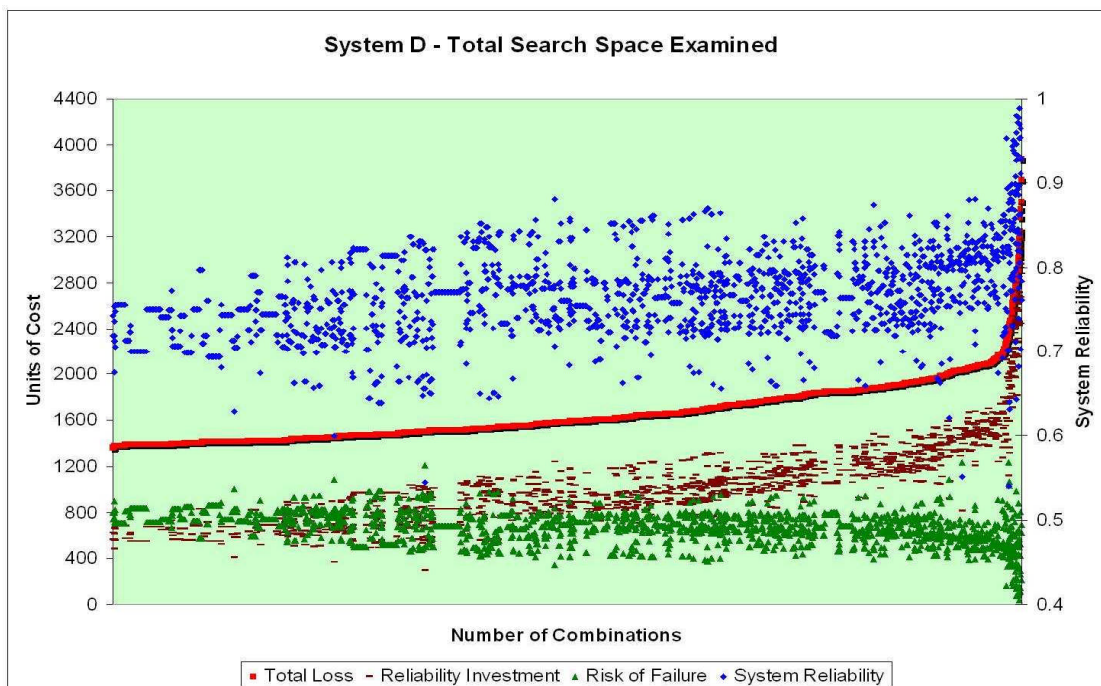


Figure IV.6 Total Search Space Examined By the Optimisation Algorithm for System D

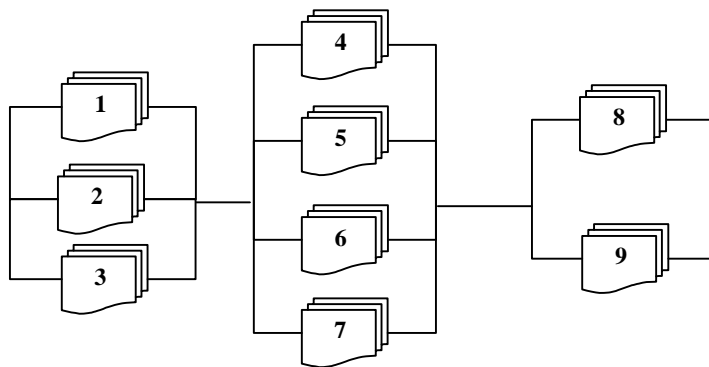
## APPENDIX

## V

# OPTIMISATION RESULTS OF SYSTEM-A (FROM SECTION 6.4.1.1, USING TABLE 6.13)

---

The topology of System A consists of three subsystems each containing three, four and two components, connected in parallel, respectively – Fig. V.1. The application of the optimisation algorithm using the data from Table 6.13 is described by means of various graphs showing the actual optimisation process, the effect of the genetic operations (crossover and mutation) on total loss associated with the optimal reliability allocated for this system and table detailing various sub-optimal results found along with the optimum solution.



**Figure V.1 Structure of System A**

No	Total Loss	System Reliability	Reliability Investment	Risk of Failure	Sub-System One			Sub-System Two				Sub-System Three	
1	815	86.8%	446	369	6	6	3	3	2	5	2	3	9
2	843	87.8%	498	345	3	6	7	3	2	5	4	3	9
3	854	84.7%	427	427	6	4	3	4	2	2	2	3	9
4	859	86.8%	484	374	6	6	3	5	2	2	3	3	9
5	876	85.5%	468	408	6	4	3	5	2	2	3	3	9
6	885	84.3%	447	438	3	4	4	2	2	5	4	3	9
7	886	81.9%	389	497	3	3	3	5	2	2	2	3	9
8	888	82.8%	411	476	3	3	4	5	2	2	2	3	9
9	1013	83.0%	540	473	3	3	2	5	2	8	2	3	9
10	1015	85.0%	591	424	5	3	2	5	2	8	2	3	9
11	1023	85.7%	614	410	5	4	2	2	2	8	4	3	9
12	1071	84.7%	633	438	5	3	2	5	2	7	2	3	9
13	1089	85.9%	681	407	5	4	2	5	2	7	4	3	9
14	1147	86.0%	749	398	3	4	3	5	2	8	9	3	9
15	1208	87.5%	842	366	5	4	3	5	2	7	9	3	9
16	1389	86.9%	1007	382	5	3	3	5	2	7	10	3	9
17	1394	87.8%	1036	358	5	5	2	5	2	7	10	3	9
18	1496	87.0%	1120	376	5	3	3	5	2	7	11	3	9
19	1503	86.3%	1107	396	5	3	2	5	2	7	11	3	9
20	1714	95.0%	1549	165	3	5	10	9	5	2	11	10	5

**Table V.1 List of Results Found By the Optimisation Algorithm for System A**

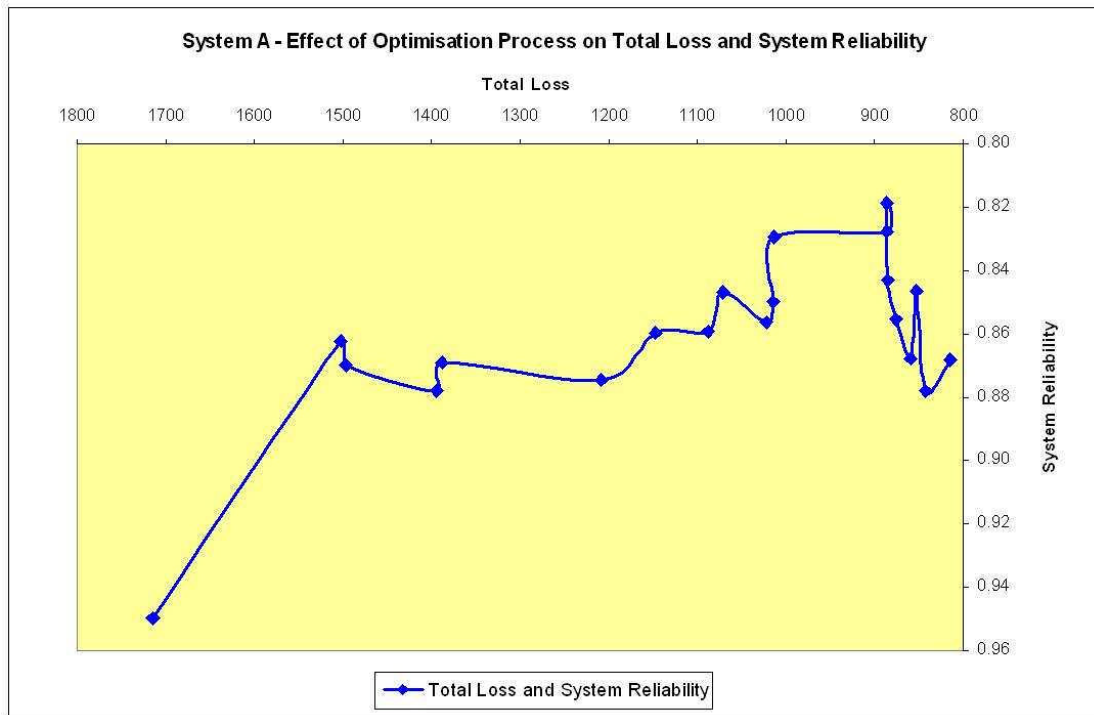


Figure V.2 Effect of Optimisation Process on System Reliability and Total Loss

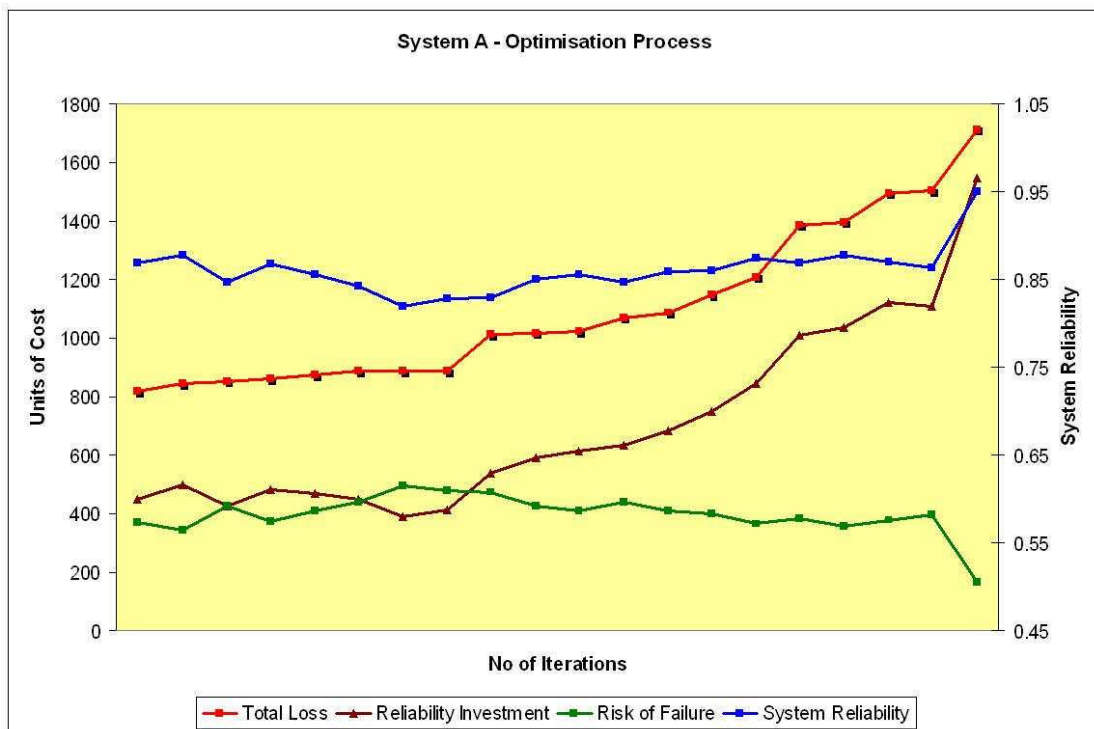


Figure V.3 Optimisation Process of System A



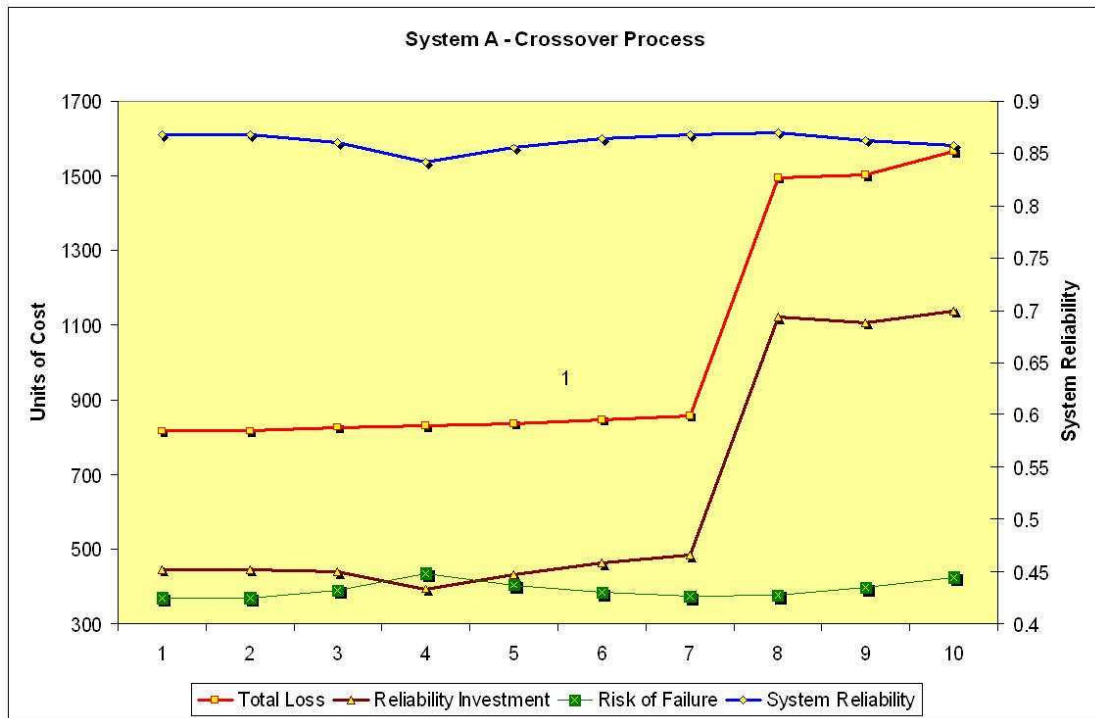


Figure V.4 Crossover Process of System A

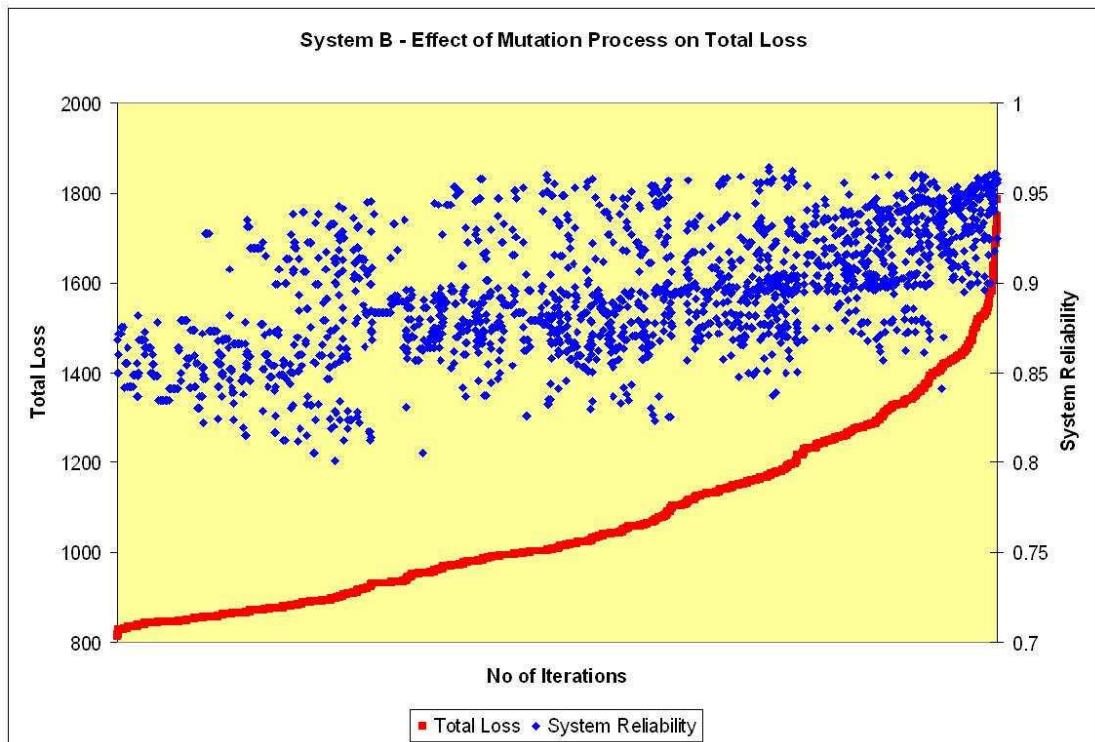


Figure V.5 Effect of Mutation Process on Total Loss in System A



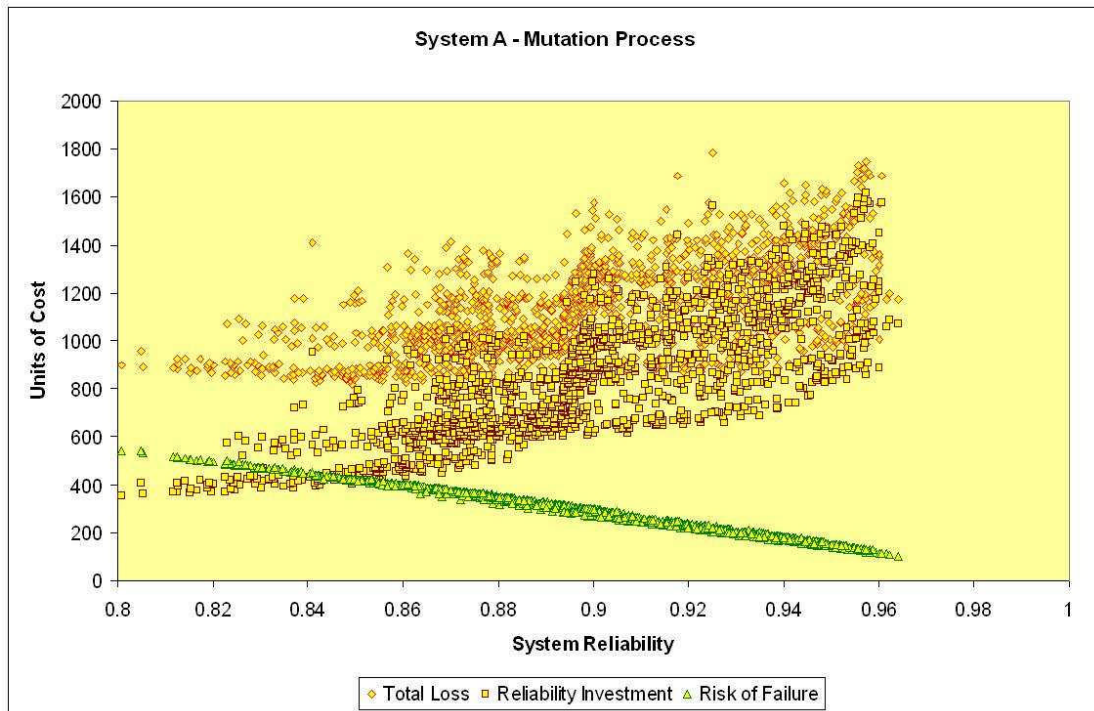


Figure V.6 Mutation Process in System A

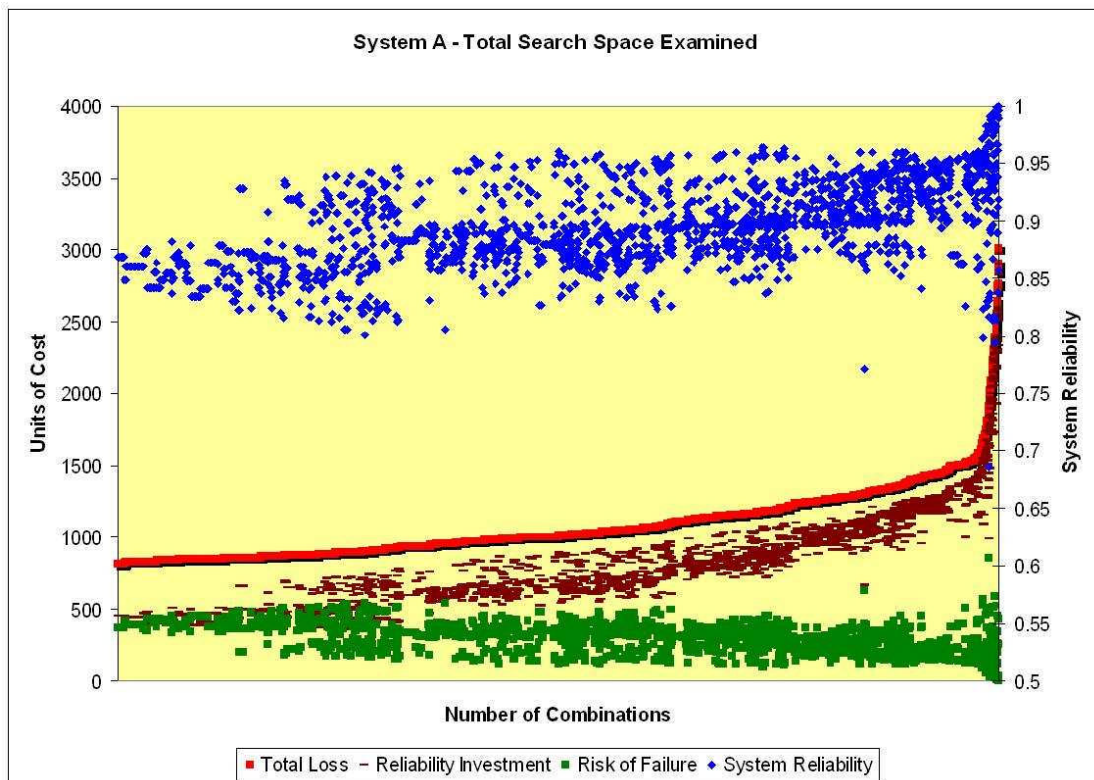


Figure V.7 Total Search Space Examined By the Optimisation Algorithm for System A

# APPENDIX VI

## OPTIMISATION RESULTS OF SYSTEM-B (FROM SECTION 6.4.1.2, USING TABLE 6.13)

---

The topology of System B consists of three subsystems each containing two, five and two components, connected in parallel, respectively – Fig. VI.1. The application of the optimisation algorithm using the data from Table 6.13 is described by means of various graphs showing the actual optimisation process, the effect of the genetic operations (crossover and mutation) on total loss associated with the optimal reliability allocated for this system and table detailing various sub-optimal results found along with the optimum solution.

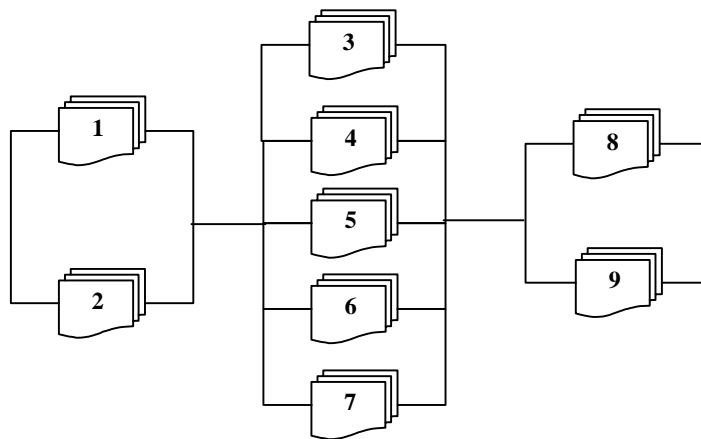
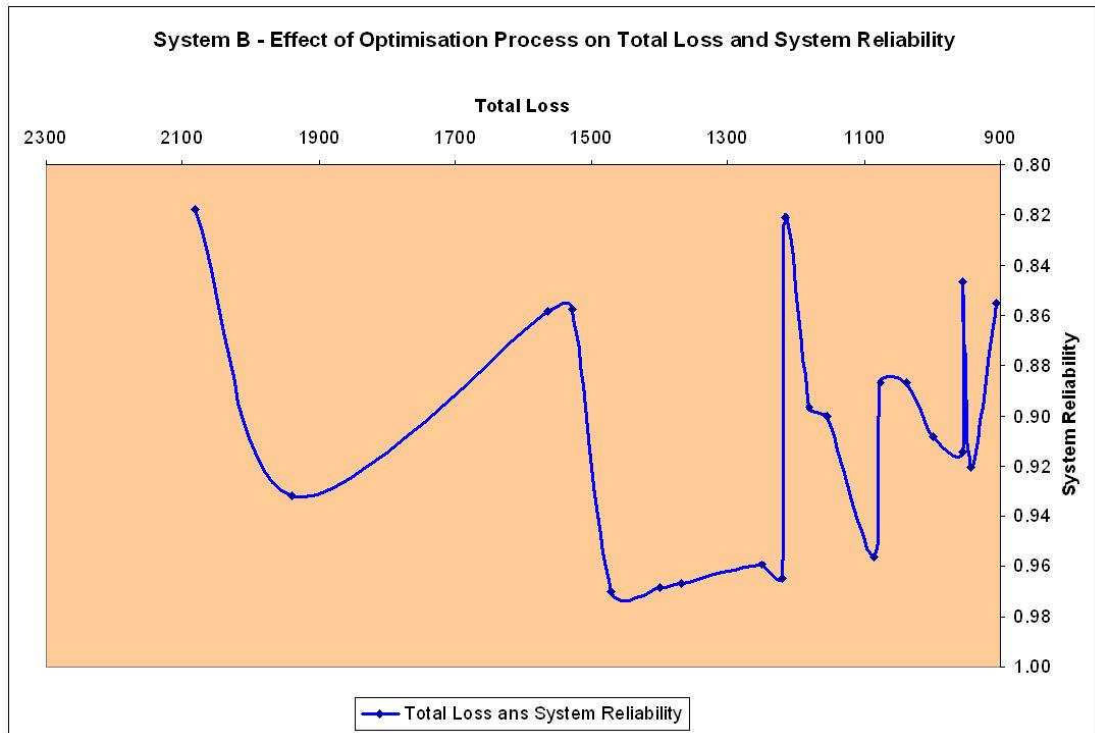


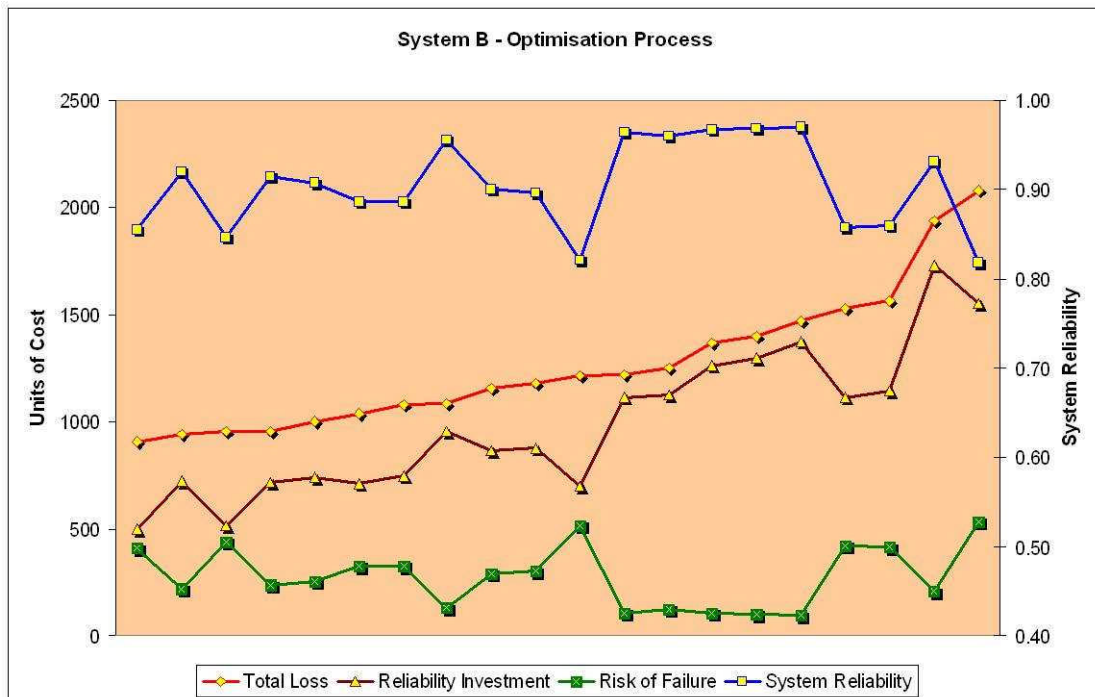
Figure VI. Structure of System B

No.	Total Loss	System Reliability	Reliability Investment	Risk of Failure	Sub-System One		Sub-System Two				Sub-System Three		
1	905	85.5%	496	409	8	6	2	4	2	2	2	2	9
2	943	92.1%	722	221	8	6	2	4	3	2	2	2	12
3	955	84.7%	518	437	6	7	2	2	2	5	3	2	9
4	955	91.4%	717	238	8	6	2	2	3	2	2	2	12
5	999	90.8%	741	258	6	7	2	2	5	2	2	2	12
6	1038	88.7%	713	326	3	11	3	2	4	2	2	2	9
7	1076	88.7%	749	327	11	3	3	2	4	2	2	2	9
8	1085	95.6%	956	129	2	11	2	2	5	4	2	2	12
9	1156	90.1%	865	291	7	11	2	2	5	4	2	2	9
10	1180	89.7%	876	304	11	3	3	2	8	2	2	2	9
11	1215	82.1%	700	515	3	10	2	2	2	5	3	2	7
12	1219	96.5%	1111	108	11	3	4	2	8	2	2	2	12
13	1250	96.0%	1126	124	11	3	2	2	6	4	2	2	12
14	1368	96.7%	1265	104	11	3	2	2	6	8	2	2	12
15	1400	96.8%	1300	100	11	3	4	2	6	8	2	2	12
16	1470	97.0%	1375	95	11	3	4	2	6	9	2	2	12
17	1529	85.7%	1111	418	11	6	2	2	6	9	2	2	7
18	1563	85.9%	1146	417	11	6	4	2	6	9	2	2	7
19	1939	93.2%	1733	206	8	11	5	6	11	11	2	4	9
20	2080	81.8%	1549	531	3	5	10	9	5	2	11	10	5

Table VI.1 List of Results Found by the Optimisation Algorithm for System B



**Figure VI.2 Effect of Optimisation Process on System Reliability and Total Loss in System B**



**Figure VI.3 Optimisation Process of System B**

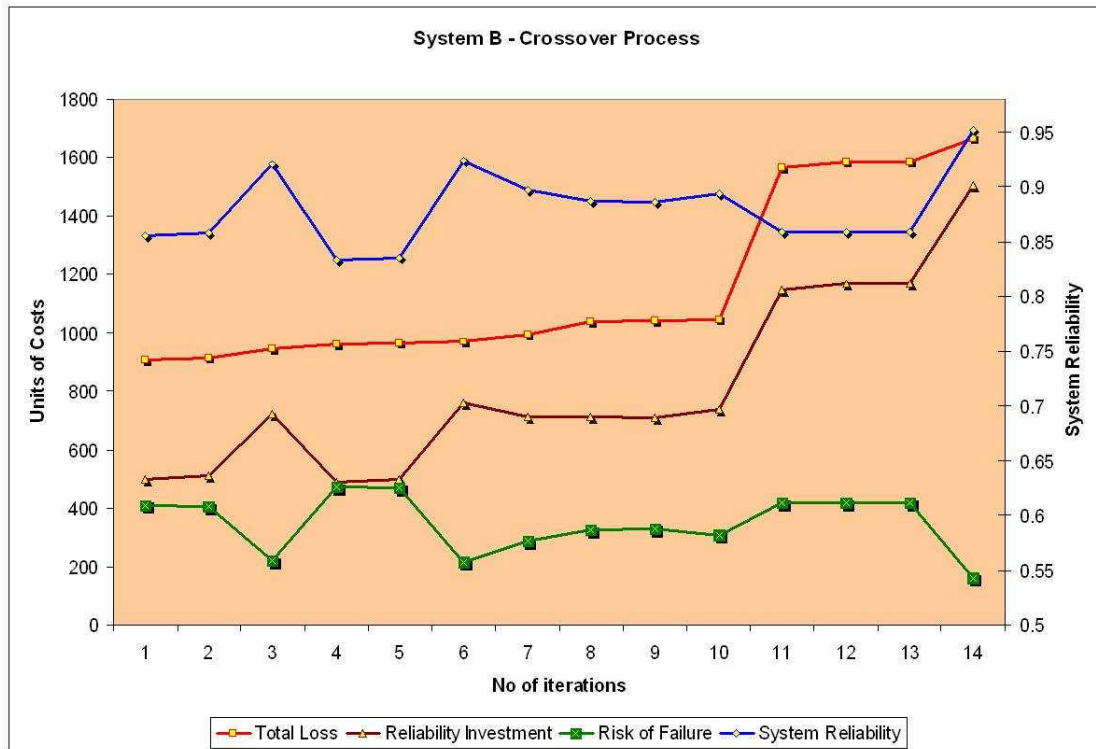


Figure VI. 4 Crossover Process of System B

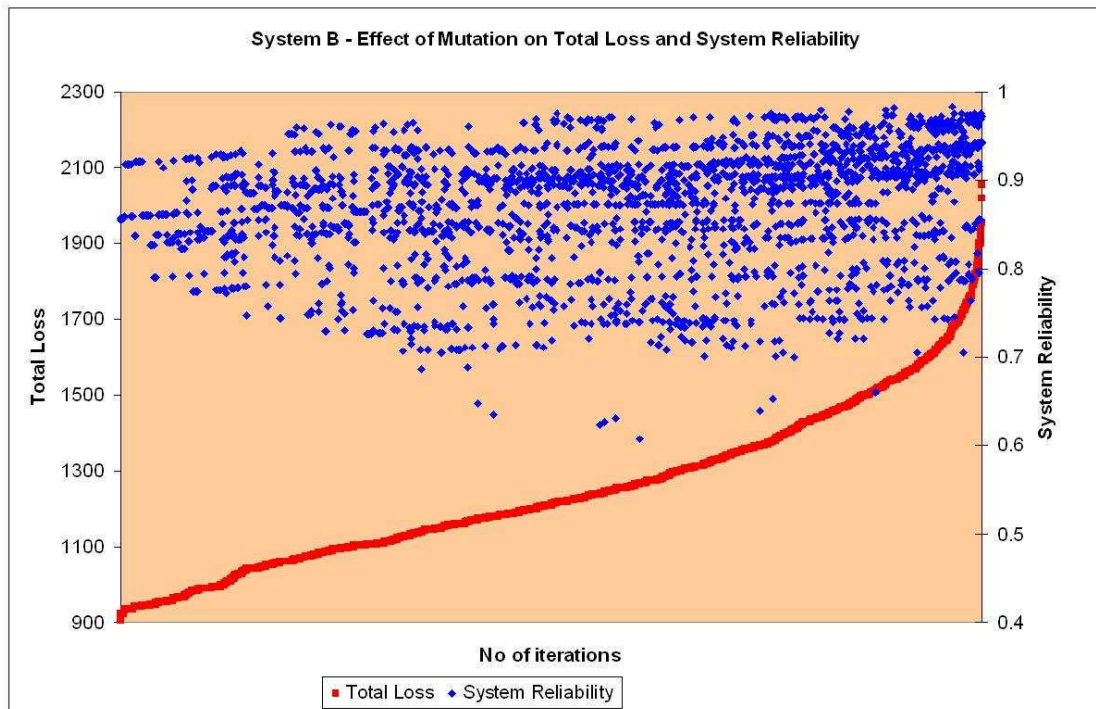


Figure VI. 5 Effect of Mutation Process on Total Loss in System B



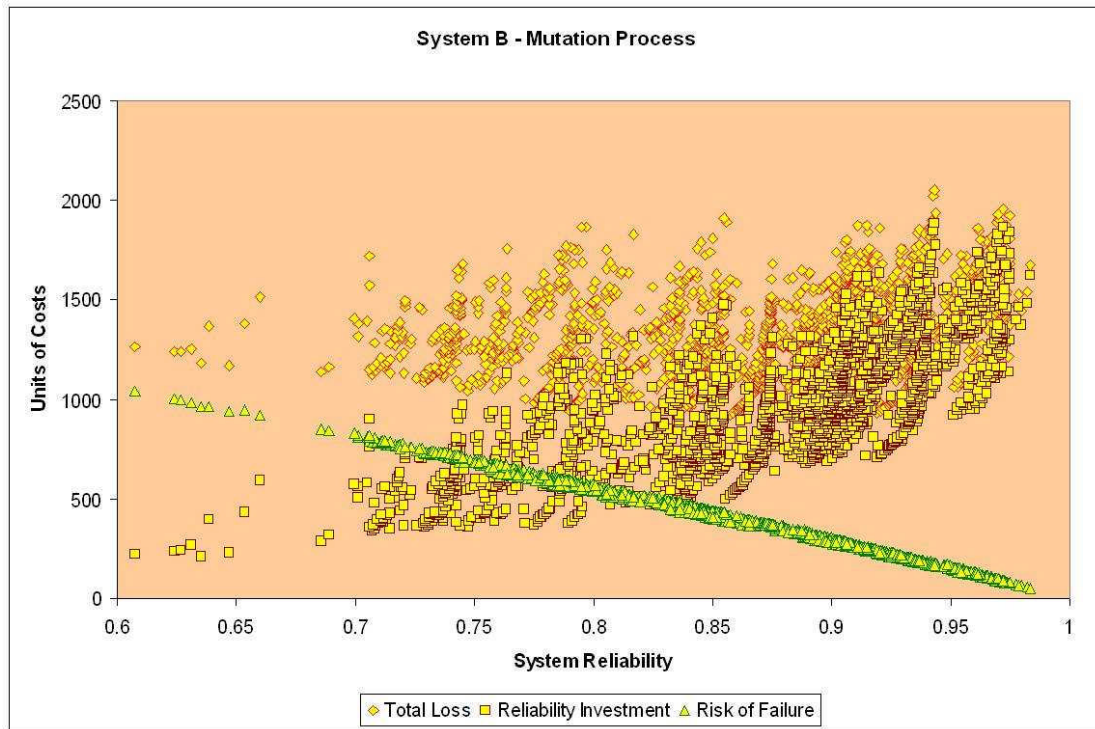


Figure VI. 6 Mutation Process in System B

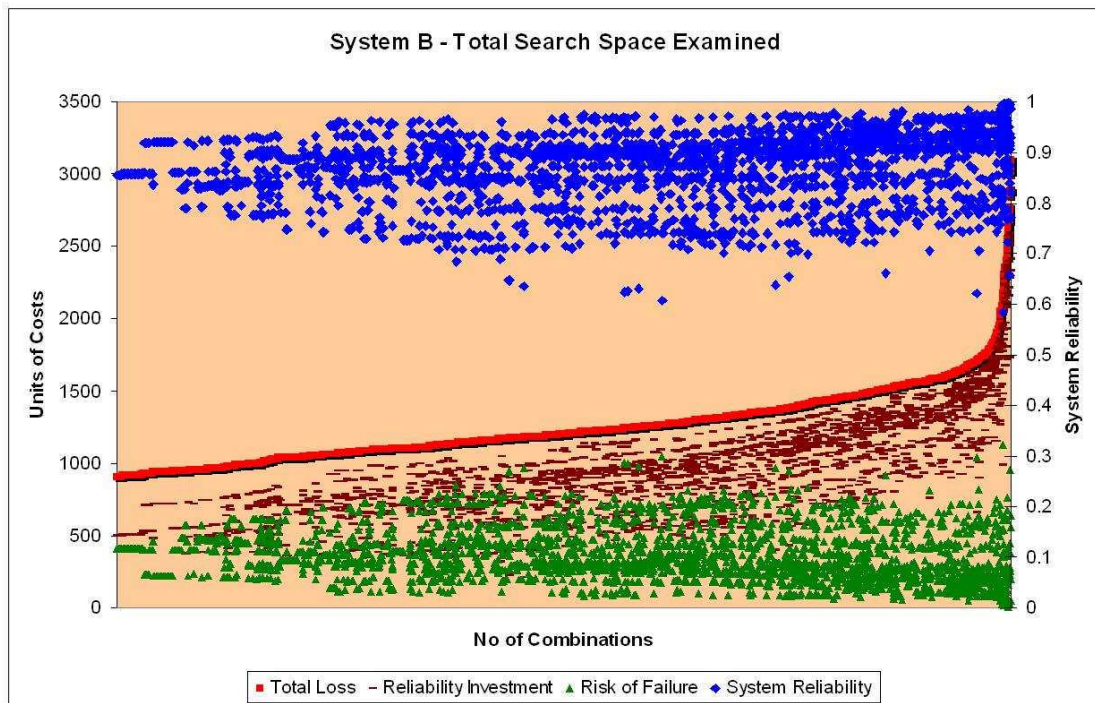


Figure VI. 7 Total Search Space Examined By the Optimisation Algorithm for System B

# APPENDIX VII

## OPTIMISATION RESULTS OF SYSTEM-C (FROM SECTION 6.4.1.3, USING TABLE 6.13)

---

The topology of System C consists of three subsystems each containing two, five and two components, connected in parallel, respectively – Fig. VII.1. The application of the optimisation algorithm using the data from Table 6.13 is described by means of various graphs showing the actual optimisation process, the effect of the genetic operations (crossover and mutation) on total loss associated with the optimal reliability allocated for this system and table detailing various sub-optimal results found along with the optimum solution.

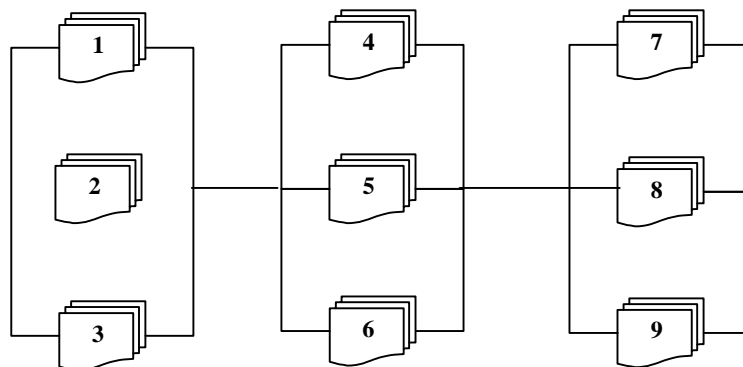


Figure VII.1 Structure of System C

No.	Total Loss	System Reliability	Reliability Investment	Risk of Failure	Sub-system One			Sub-System Two			Sub-System Three		
1	813	84.2%	377	437	6	6	3	5	4	2	6	3	2
2	821	87.9%	479	342	6	6	3	4	4	6	6	3	5
3	829	86.0%	436	393	6	6	3	5	4	2	6	3	5
4	841	83.5%	384	457	6	6	3	4	4	4	2	3	5
5	843	88.3%	513	330	6	6	3	5	4	4	2	3	9
6	843	87.5%	492	351	6	6	3	4	4	4	2	3	9
7	849	80.7%	321	528	3	6	3	4	4	4	2	3	4
8	849	83.3%	387	463	6	6	3	5	4	4	2	3	4
9	856	84.6%	424	432	6	6	3	5	4	2	5	3	4
10	886	81.1%	363	523	3	4	3	5	4	2	5	3	4
11	890	77.5%	280	609	3	3	3	3	4	4	2	3	4
12	895	78.3%	304	591	4	3	3	3	4	4	2	3	4
13	914	76.5%	278	635	2	3	3	3	4	4	2	3	4
14	974	87.0%	611	362	3	4	4	3	4	11	5	3	4
15	1010	88.6%	693	317	3	3	3	3	4	11	9	3	4
16	1037	87.5%	691	346	2	3	3	3	4	11	9	3	4
17	1276	93.8%	1086	190	2	10	3	3	4	11	8	3	4
18	1409	91.2%	1146	263	11	6	4	2	6	9	2	2	7
19	1440	88.5%	1107	333	5	3	2	5	2	7	11	3	9
20	1672	96.0%	1549	122	3	5	10	9	5	2	11	10	5

Table VII.1 Optimisation Results of System C found by the optimisation algorithm



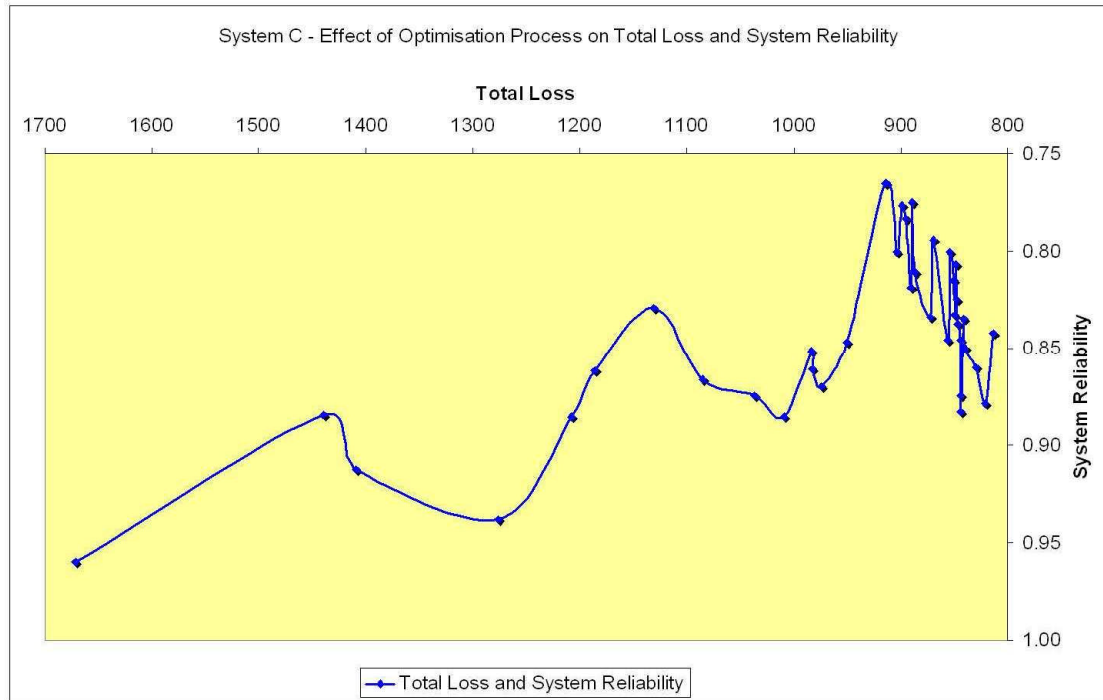


Figure VII.2 Effect of Optimisation Process on System Reliability and Total Loss in System C

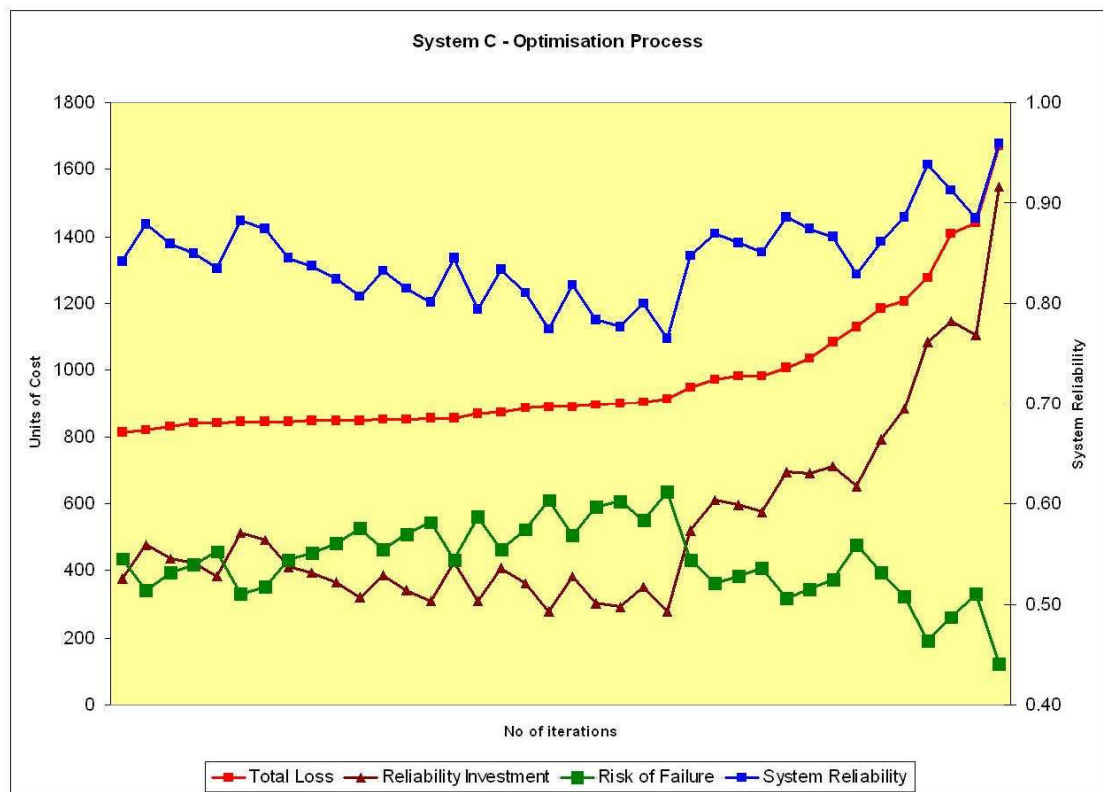


Figure VII.3 Optimisation Process of System C

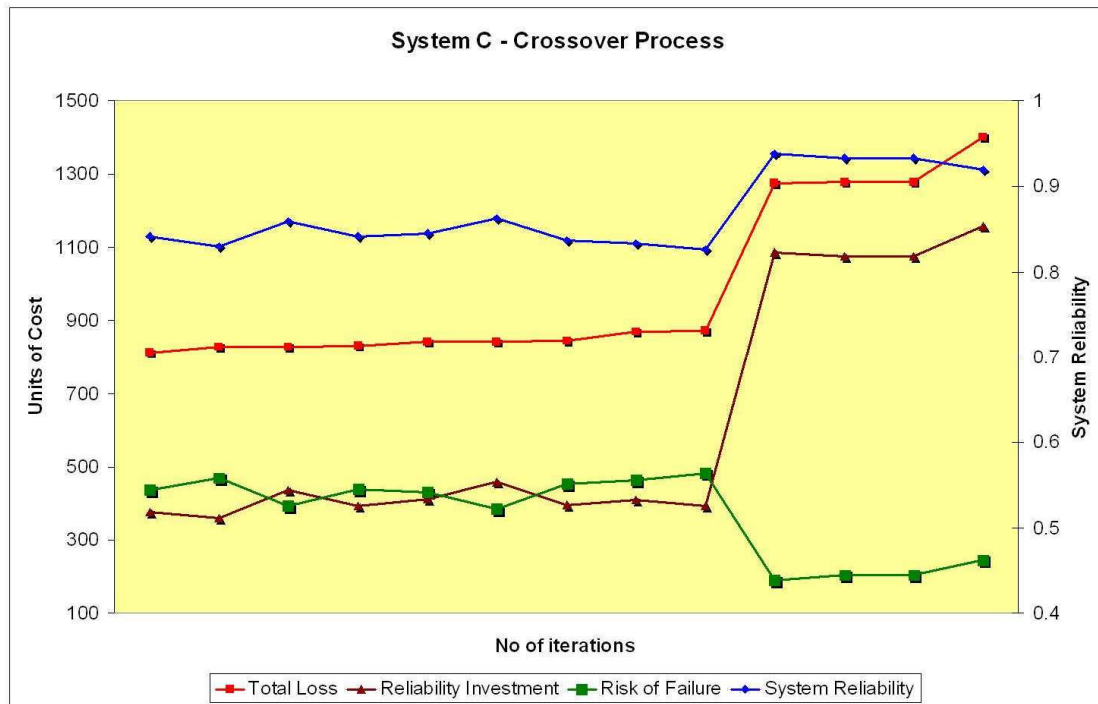


Figure VII.4 Crossover Process of System C

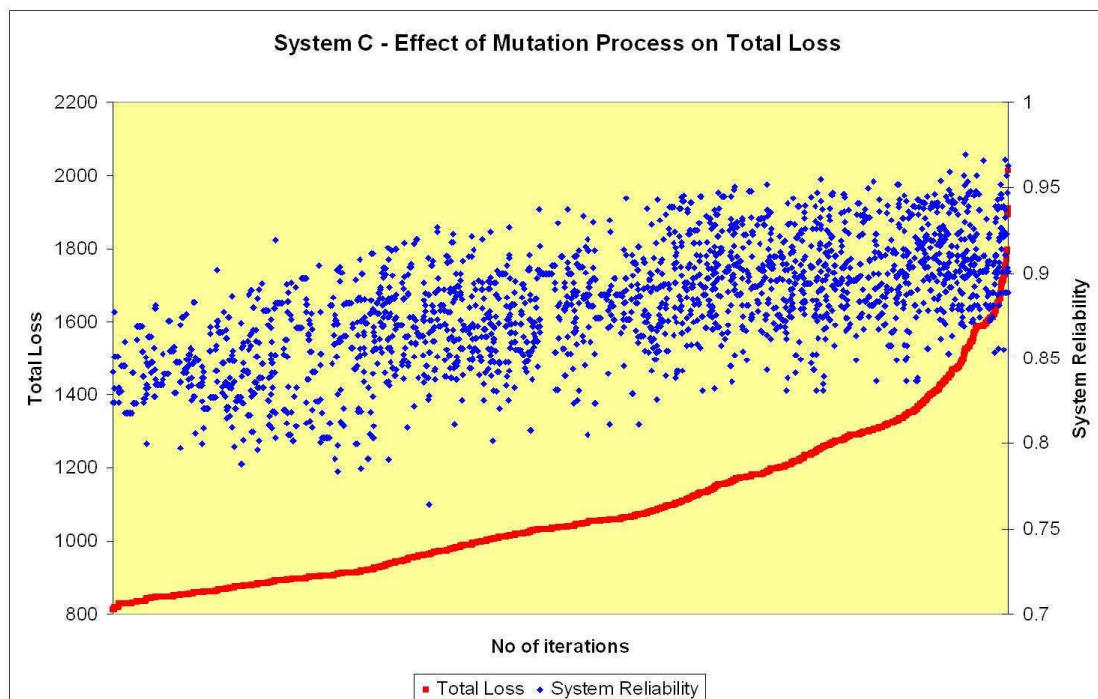


Figure VII.5 Effect of Mutation Process on Total Loss in System C

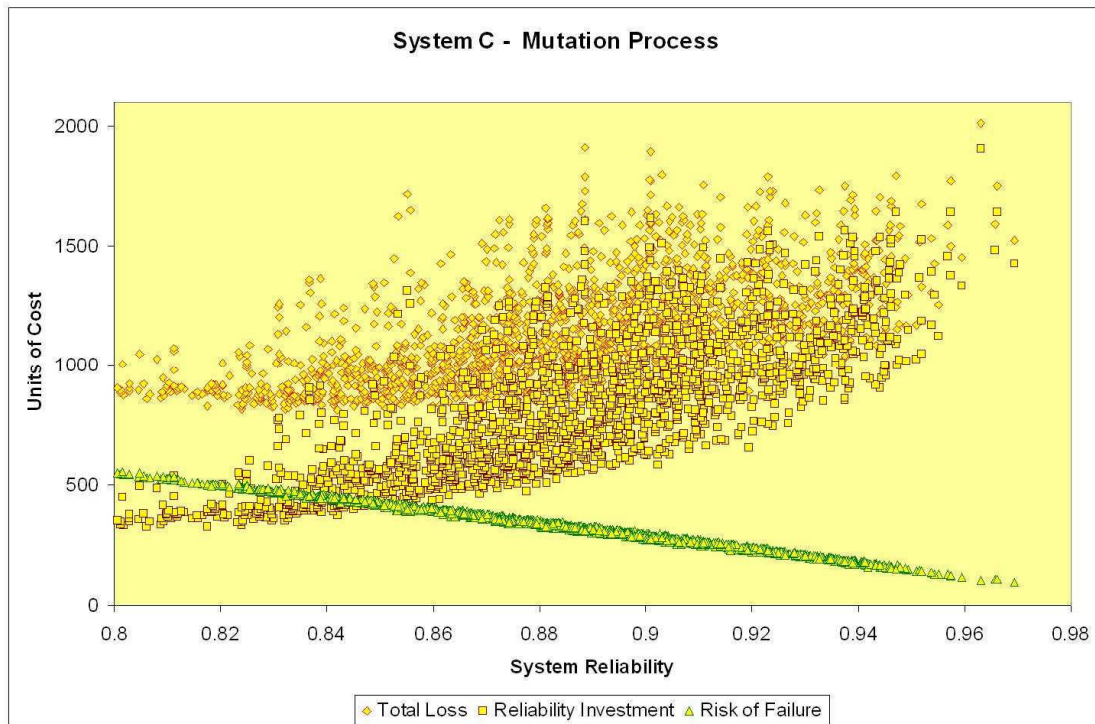


Figure VII.6 Mutation Process in System

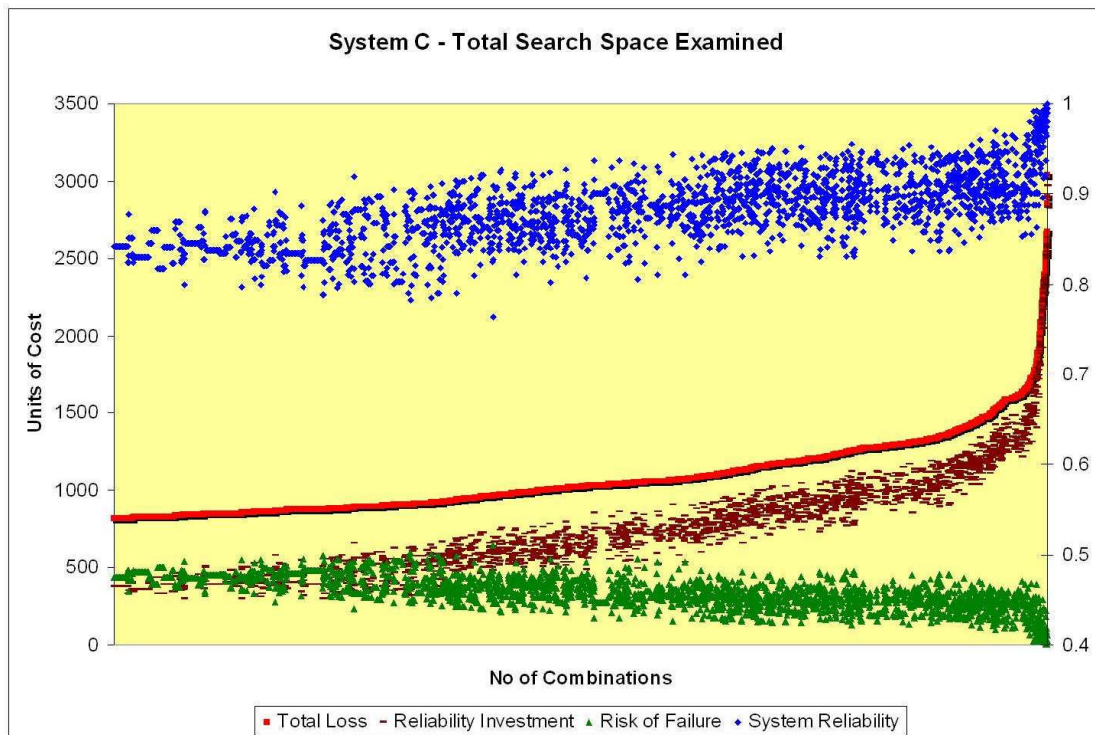


Figure VII.7 Total Search Space Examined By the Optimisation Algorithm for System C

# APPENDIX VIII

## OPTIMISATION RESULTS OF SYSTEM-D (FROM SECTION 6.4.1.4, USING TABLE 6.13)

---

The topology of System D consists of three subsystems each containing two, five and two components, connected in parallel, respectively – Fig. VIII.1. The application of the OA using the data from Table 6.13 is described by means of various graphs showing the actual optimisation process, the effect of the genetic operations (crossover and mutation) on total loss associated with the optimal reliability allocated for this system and table detailing various sub-optimal results found along with the optimum solution.

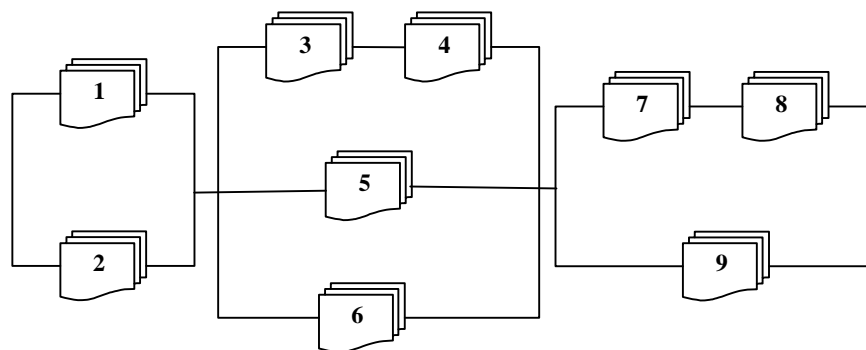
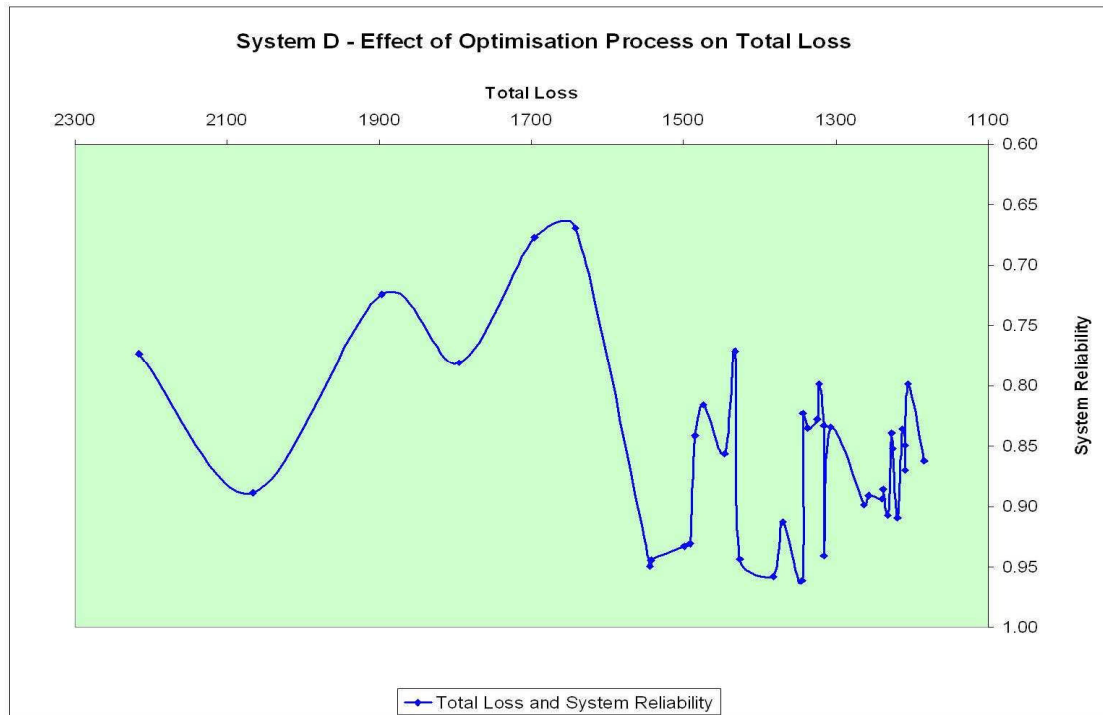


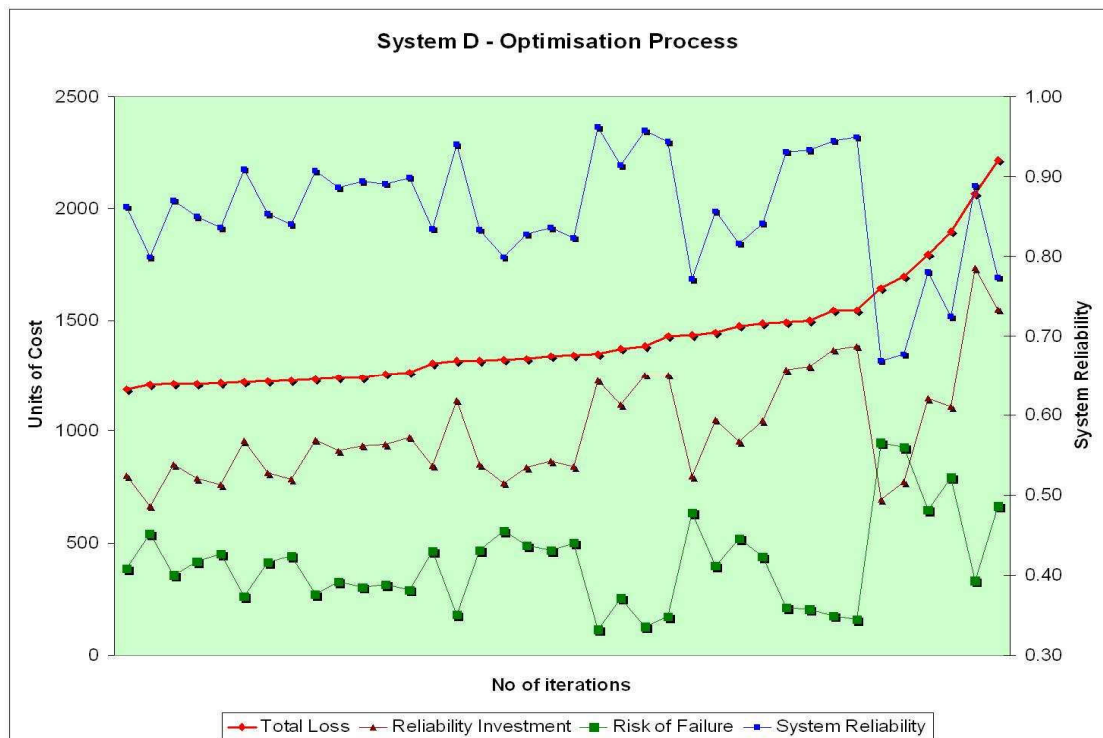
Figure VIII.1 Structure of System D

NO.	Total Loss	System Reliability	Reliability Investment	Risk of Failure	Sub-System One		Sub-System Two				Sub-System Three		
1	1184	86.2%	799	386	8	6	3	3	4	6	2	2	12
2	1205	79.9%	663	542	6	6	3	3	4	3	2	2	12
3	1209	87.1%	850	359	10	6	2	3	5	4	2	2	12
4	1219	90.9%	957	262	10	6	2	3	8	4	2	2	12
5	1225	85.2%	812	413	6	6	4	3	8	3	2	2	12
6	1231	90.7%	962	269	10	6	2	2	8	4	2	2	12
7	1238	88.6%	910	328	8	6	4	3	8	4	2	2	12
8	1263	89.9%	971	292	10	6	4	2	8	2	2	2	12
9	1316	94.1%	1136	180	10	6	4	2	8	8	2	2	12
10	1316	83.3%	850	467	4	6	3	6	8	4	2	2	12
11	1345	96.2%	1229	116	10	6	4	2	8	11	2	2	12
12	1370	91.3%	1115	255	11	3	3	2	8	4	2	2	12
13	1382	95.8%	1253	129	10	6	4	2	6	11	2	2	12
14	1427	94.4%	1253	173	11	3	3	2	8	8	2	2	12
15	1447	85.7%	1050	397	3	6	4	2	6	11	2	2	12
16	1499	93.3%	1292	207	11	4	3	2	6	8	2	2	12
17	1544	94.9%	1383	161	11	6	3	2	6	9	2	2	12
18	1642	66.9%	695	947	2	6	4	2	6	8	2	2	7
19	2066	88.9%	1733	333	8	11	5	6	11	11	2	4	9
20	2215	77.3%	1549	666	3	5	10	9	5	2	11	10	5

Table VIII.1 Optimisation Results for System D

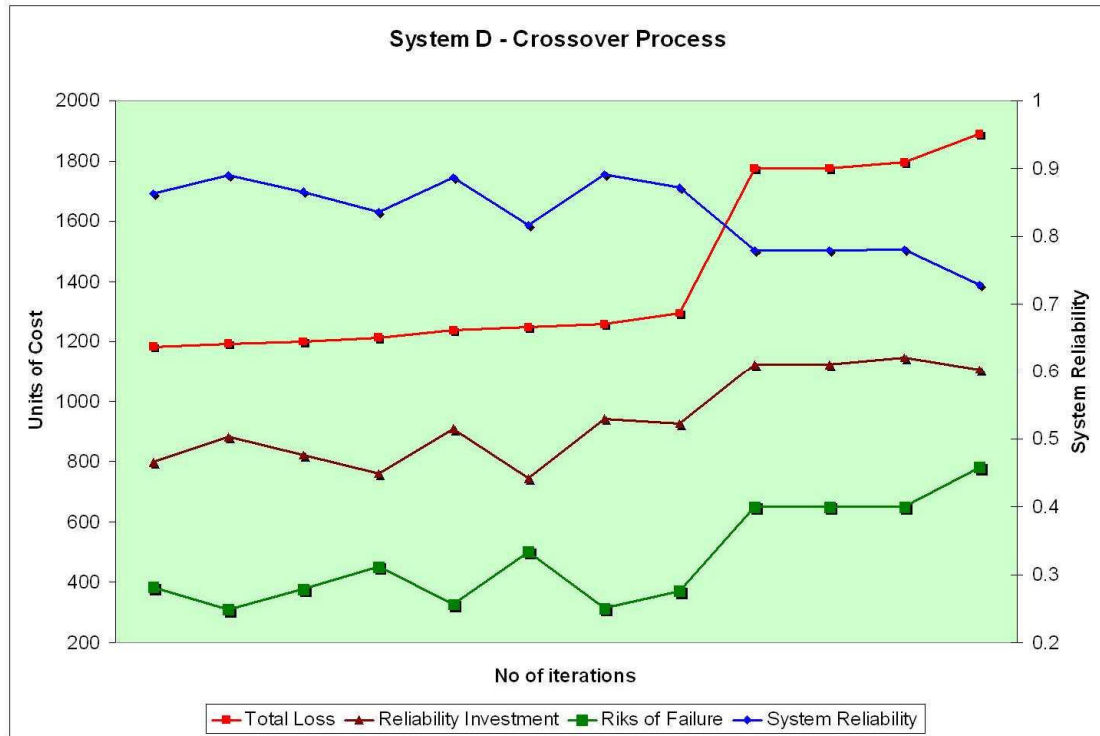


**Figure VIII.2 Effect of Optimisation Process on System Reliability and Total Loss in System D**

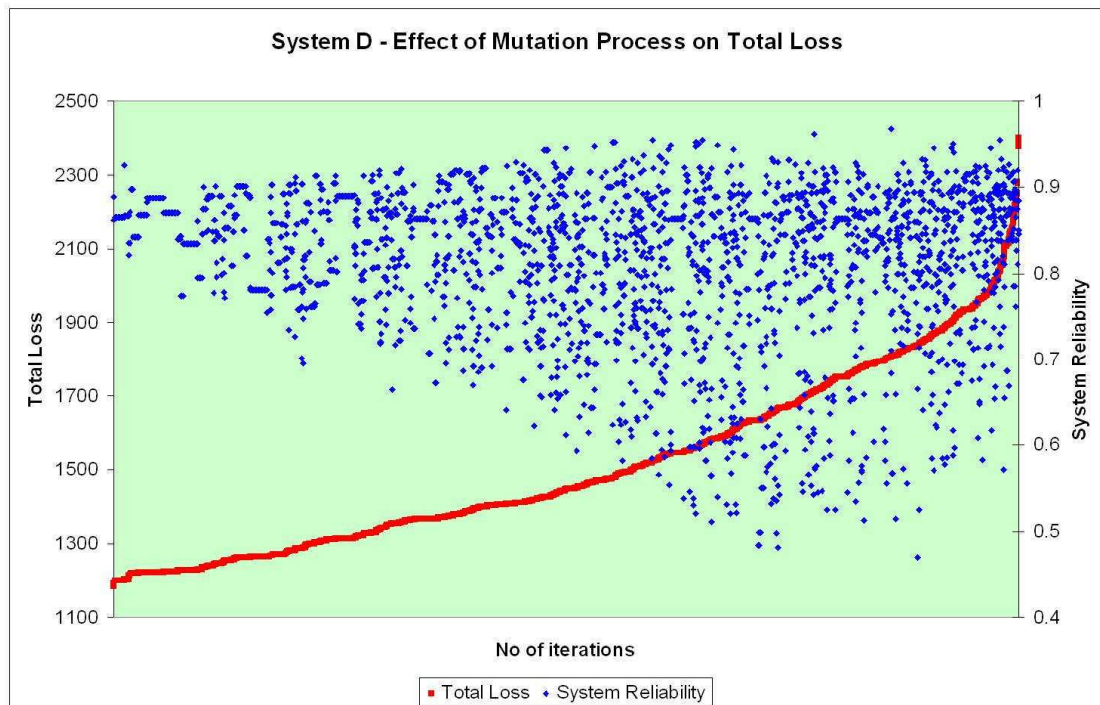


**Figure VIII.3 Optimisation Process of System D**





**Figure VIII.4 Crossover Process of System D**



**Figure VIII.5 Effect of Mutation Process on Total Loss in System D**

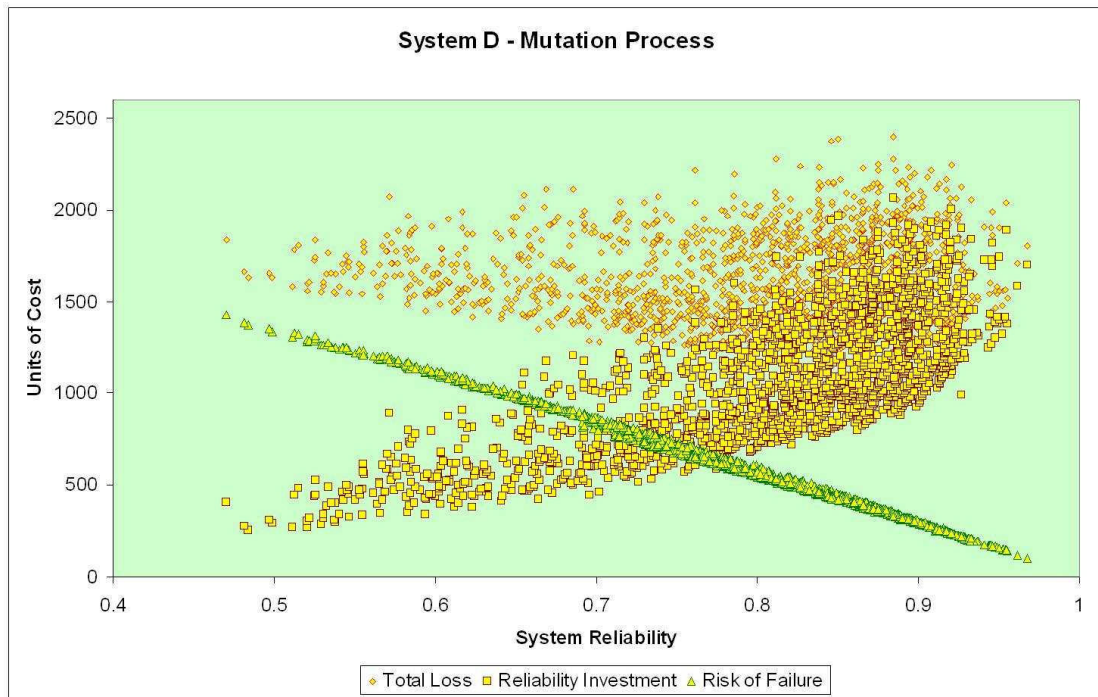


Figure VIII.6 Mutation Process in System D

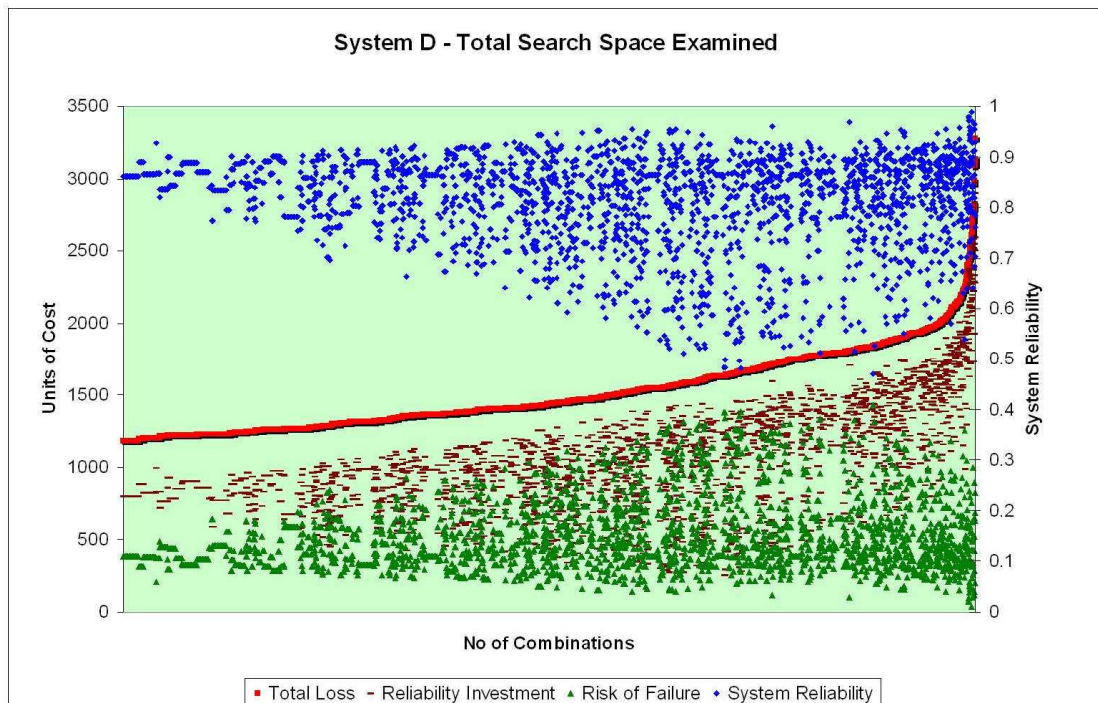


Figure IV.7 Total Search Space Examined By the Optimisation Algorithm for System D



# APPENDIX IX

## OPTIMISATION RESULTS OF SYSTEMS USING LOWER COST OF FAILURE

**(FROM SECTION 6.4.2, USING TABLE 6.1)**

---

The results from the optimisation process detailed in section 6.4.2 are presented in this appendix, using a lower cost of failure amount, 'C<sub>2</sub>' (1000 units) for all four systems by utilising the data from Table 6.1. For each of the four systems, the optimum solution along with a list of various sub-optimal solutions is presented in a result table, showing also the configurations of the selected components for each system.

## System A

The structure of System A, consists of three subsystems each containing three, four and two components, connected in parallel, respectively, as shown in Fig. IX.1.

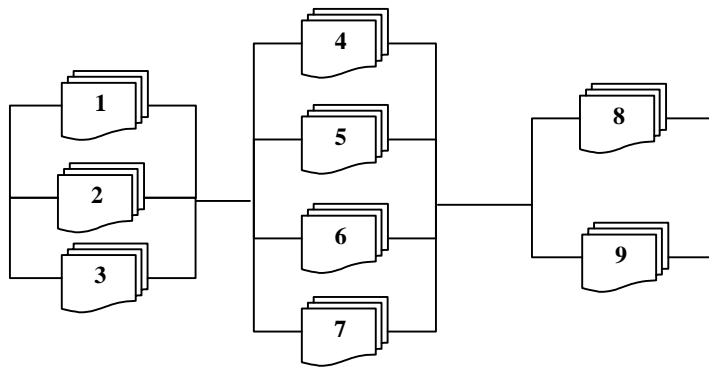


Figure IX.1 Structure of System A

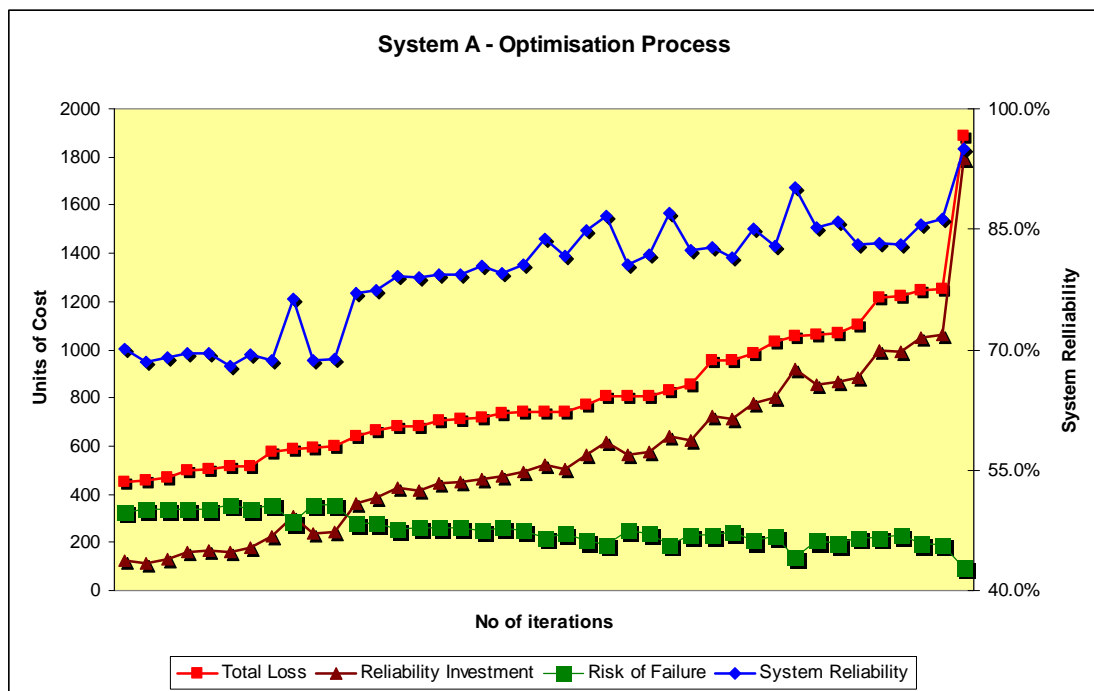


Figure IX.2 Optimisation Process of System A

No	Total Loss	System Reliability	Reliability Investment	Risk of Failure	Sub-System One			Sub-System Two				Sub-System Three	
1	453	70.2%	127	327	2	2	4	2	2	2	3	3	4
2	468	69.0%	129	339	2	2	2	3	2	2	4	4	3
3	577	68.6%	223	354	2	2	2	3	2	4	6	4	2
4	590	76.3%	307	283	2	2	2	2	3	4	3	2	8
5	600	68.9%	245	354	2	2	2	4	3	4	6	4	2
6	641	77.0%	360	281	2	2	2	2	3	4	5	2	8
7	737	79.6%	477	259	2	2	2	4	3	5	6	4	8
8	741	80.6%	491	250	2	3	2	4	3	5	6	4	8
9	742	83.8%	525	217	2	6	2	4	3	4	6	4	8
10	771	84.9%	564	207	3	6	2	4	3	5	6	4	8
11	807	86.6%	617	190	6	6	2	4	3	4	6	4	8
12	809	80.6%	562	247	2	2	2	2	2	7	6	3	9
13	831	86.9%	644	187	6	6	2	4	3	5	6	4	8
14	954	82.8%	725	229	3	2	2	5	2	7	8	3	9
15	1057	90.2%	920	137	7	5	2	3	2	2	3	6	11
16	1067	85.9%	869	197	5	3	2	5	2	7	9	3	9
17	1106	83.1%	884	222	3	2	2	5	2	7	10	3	9
18	1215	83.2%	997	218	3	2	2	5	2	7	11	3	9
18	1246	85.5%	1048	198	5	2	2	5	2	7	11	3	9
20	1885	95.0%	1792	93	3	5	10	9	5	2	11	10	5

Table IX.1 List of Results Found by the Optimisation Algorithm for System A

## System B

The topology of System B consists of three subsystems each containing two, five and two components, connected in parallel, respectively – Fig. IX.3.

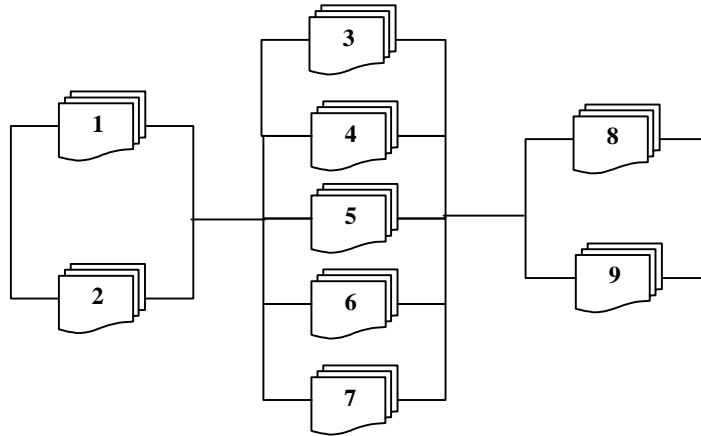


Figure IX.3 Structure of System B

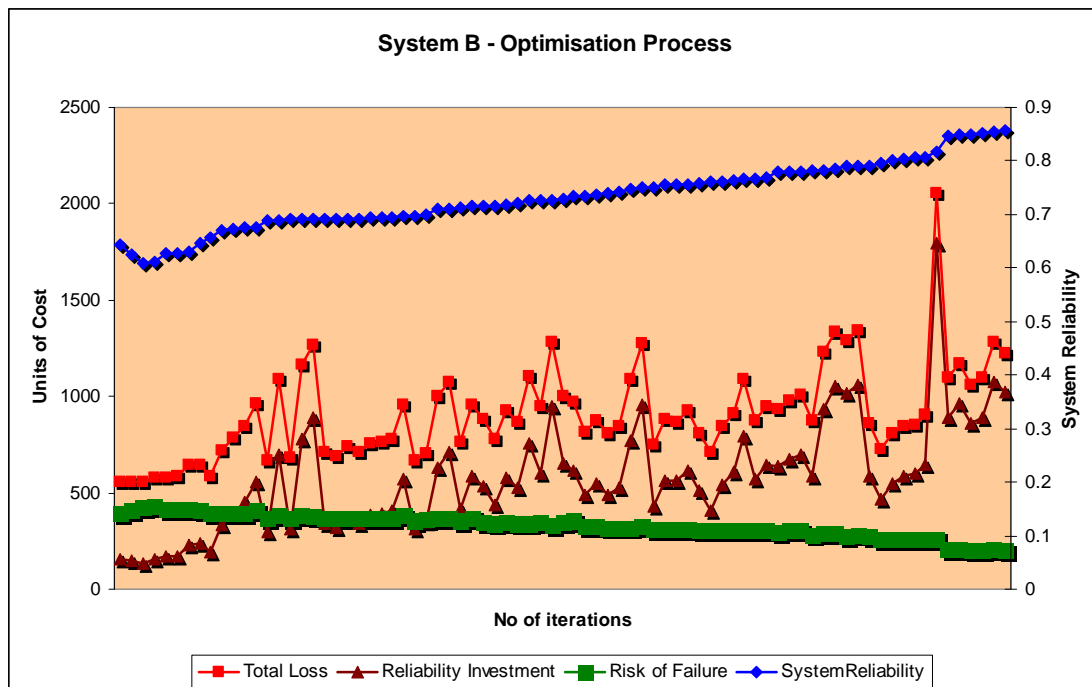


Figure IX.4 Optimisation Process of System B

No.	Total Loss	System Reliability	Reliability Investment	Risk of Failure	Sub-System One		Sub-System Two					Sub-System Three	
1	553	64.3%	157	396	3	3	3	2	3	4	2	2	4
2	648	64.7%	238	410	3	3	2	2	3	5	2	2	5
3	671	69.8%	311	360	3	3	2	2	3	5	2	2	5
4	708	70.1%	346	362	3	3	4	2	3	5	2	2	5
5	725	79.6%	465	261	3	6	2	2	6	3	2	2	3
6	747	75.1%	434	313	3	6	2	2	6	7	2	2	7
7	810	75.9%	509	301	3	4	2	2	4	7	2	2	7
8	855	80.5%	598	258	3	6	2	5	6	7	5	5	7
9	877	76.8%	574	303	3	6	2	2	6	5	2	2	5
10	928	75.7%	613	314	4	3	2	5	3	7	5	5	7
11	934	77.9%	640	294	5	3	2	5	3	7	5	5	7
12	951	76.9%	647	304	3	6	2	2	6	7	2	2	7
13	982	78.0%	678	304	5	6	3	2	6	7	2	2	7
14	1096	85.2%	893	203	11	6	4	2	6	7	2	2	7
15	1173	84.9%	965	207	11	3	4	2	3	9	2	2	9
16	1226	85.6%	1026	200	11	6	4	2	6	9	2	2	9
17	1271	69.1%	892	380	2	3	2	5	3	7	5	5	7
18	1281	85.6%	1073	208	11	6	4	2	6	7	2	2	7
19	1292	78.9%	1016	276	5	3	2	2	3	7	2	2	7
20	2053	81.8%	1792	261	3	5	10	9	5	2	9	9	2

Table IX.2 List of Results Found by the Optimisation Algorithm for System B

## System C

The topology of System C consists of three subsystems each containing two, five and two components, connected in parallel, respectively as shown in Fig. IX.5.

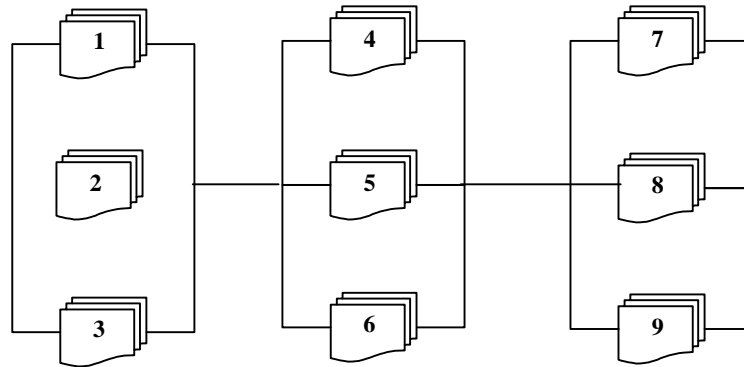


Figure IX.5 Structure of System C

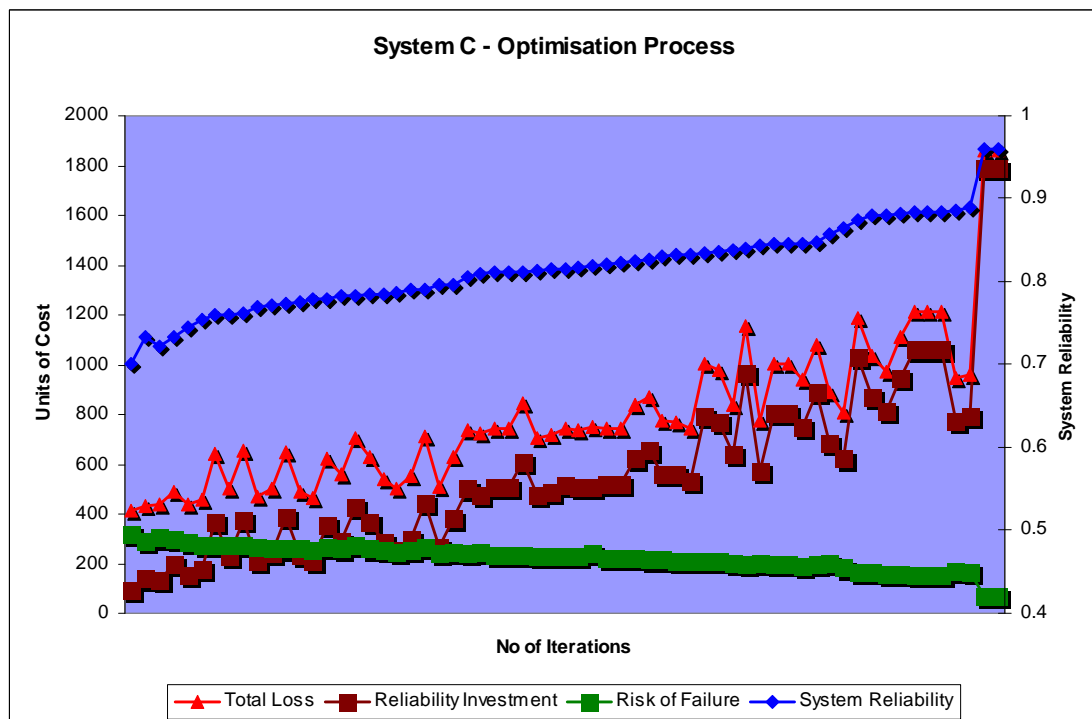


Figure IX.6 Effect of Optimisation Process on System Reliability and Total Loss in System C

No.	Total Loss	System Reliability	Reliability Investment	Risk of Failure	Sub-system One			Sub-System Two			Sub-System Three		
1	413	70.1%	93	319	2	2	2	2	2	2	4	3	2
2	436	74.4%	152	284	3	3	3	3	3	3	4	3	3
3	465	77.9%	210	255	2	2	2	2	2	2	3	3	2
4	491	77.6%	232	259	2	2	2	2	2	2	4	3	2
5	504	78.7%	254	250	2	2	2	2	2	2	4	3	2
6	621	77.9%	357	264	2	2	2	2	2	2	3	2	2
7	714	81.4%	487	228	3	3	3	3	3	3	3	3	3
8	804	86.4%	622	182	5	5	5	5	5	5	4	3	5
9	840	83.8%	639	201	2	2	2	2	2	2	10	3	2
10	888	85.6%	685	203	2	2	2	2	2	2	4	6	2
11	939	84.6%	751	187	2	2	2	2	2	2	11	3	2
12	948	88.6%	777	172	6	6	6	6	6	6	4	6	6
13	978	83.6%	770	208	2	2	2	2	2	2	9	3	2
14	1004	84.6%	806	198	2	2	2	2	2	2	9	3	2
15	1033	87.9%	869	164	5	5	5	5	5	5	9	3	5
16	1080	84.7%	886	194	2	2	2	2	2	2	10	3	2
17	1158	84.1%	962	196	2	2	2	2	2	2	11	3	2
18	1190	87.4%	1026	164	5	5	5	5	5	5	11	3	5
19	1215	88.5%	1062	153	5	5	5	5	5	5	11	3	5
20	1860	96.0%	1792	67	3	3	3	3	3	3	11	10	3

Table IX.3 Optimisation Results of System C found by the optimisation algorithm

## System D

The topology of System B consists of three subsystems each containing two, five and two components, connected in parallel, respectively – Fig. IX.7.

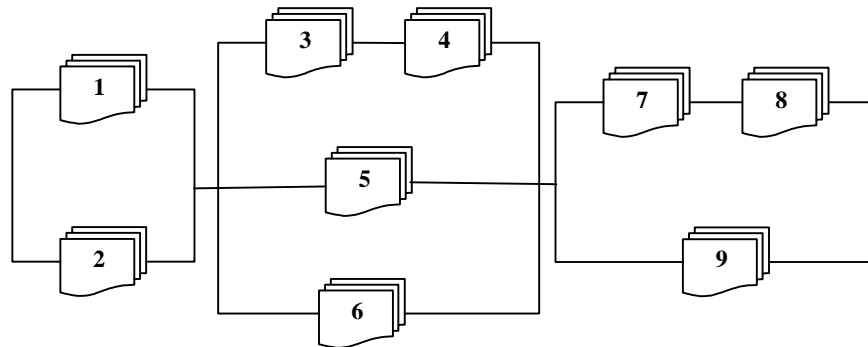


Figure IX.7 Structure of System D

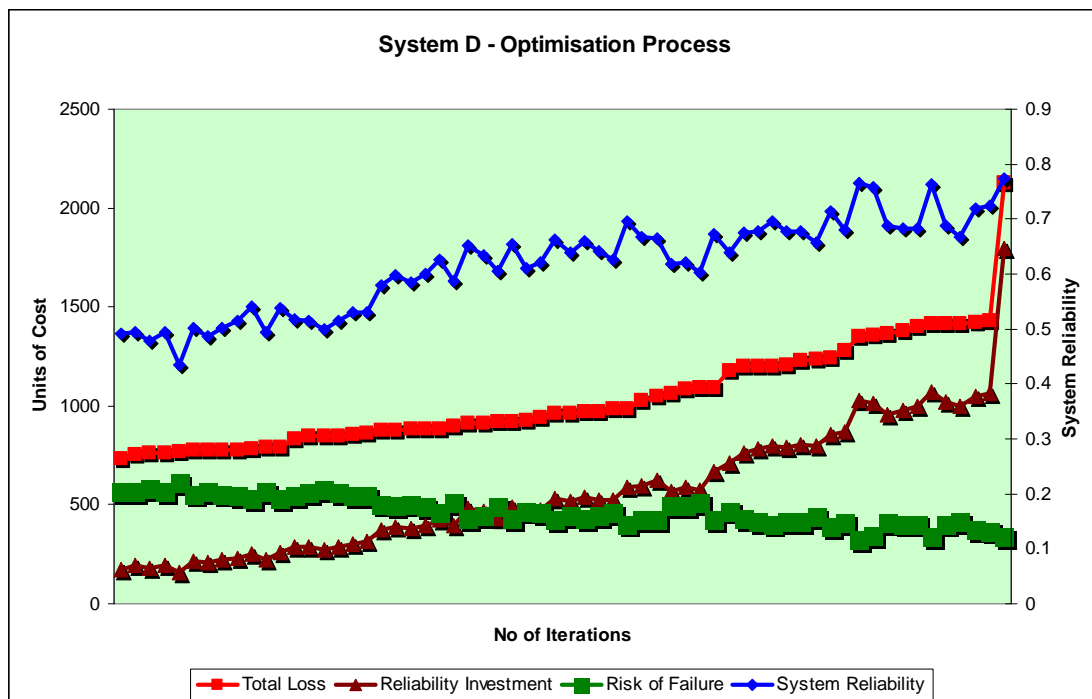


Figure IX.8 Optimisation Process of System D



NO.	Total Loss	System Reliability	Reliability Investment	Risk of Failure	Sub-System One		Sub-System Two			Sub-System Three			
1	682	44.2%	91	591	2	2	2	3	2	2	2	3	4
2	734	50.4%	180	554	3	3	2	3	2	5	2	3	4
3	777	51.5%	232	545	3	3	2	2	2	7	2	2	4
4	876	63.1%	428	449	3	3	2	3	2	5	2	3	9
5	879	59.7%	386	492	3	3	3	2	2	7	2	2	7
6	911	65.1%	480	431	3	3	2	2	2	7	2	2	9
7	915	63.3%	468	447	2	3	2	2	2	7	2	2	9
8	930	65.6%	501	430	3	3	2	3	2	7	2	3	9
9	961	63.9%	514	447	2	3	2	5	2	7	2	2	9
10	967	65.9%	537	430	3	3	2	5	2	7	2	3	9
11	971	64.1%	525	447	2	3	2	5	2	7	2	3	9
12	988	69.6%	588	400	5	3	2	5	2	7	2	3	9
13	1062	61.9%	566	496	3	3	3	2	6	7	2	2	7
14	1085	62.1%	588	497	3	3	4	2	6	7	2	2	7
15	1232	65.8%	795	438	2	3	2	5	2	7	9	2	9
16	1245	71.4%	858	387	5	3	2	5	2	7	9	2	9
17	1359	75.7%	1013	347	11	3	4	2	6	7	2	2	7
18	1414	76.4%	1073	341	11	6	4	2	6	7	2	2	7
19	1429	72.4%	1062	367	5	3	2	5	2	7	11	3	9
20	2127	77.3%	1792	334	3	5	10	9	5	2	11	10	5

Table IX.4 Optimisation Results for System D

## APPENDIX

## X

COMPUTER PROGRAM

---

```
#include <stdio.h>
#include <string.h>
#include <ctype.h>
#include <stdlib.h>
#include <math.h>
#include <time.h>
#include "math_cla.h"
class random_generator rg;
#define sizeComp 25
#define ctrSearchValue 5 //2
#define size 200
#define FACTOR 0.25
#define CrossOverRun 10
#define MutationRun 10
define InitialSampleSize 20 //2.1
define PRINTDATA 0
void GetData(void);
void GetComponentData(void);void GetAlternativesData(void);void ComponentSelection(void);int
BuildCombinations(void);void ProcessData(void);
```

---

```

double CalculateReliability(double SelectionMatrixRel[sizeComp][sizeComp-23],double
SelectionMatrixCost[sizeComp][sizeComp-23]) ;
void PrintData(void);
int CrossOverProcess(void);int FirstOrderCrossOverProcess(void);int
SecondOrderCrossOverProcess(void);
int ThirdOrderCrossOverProcess(void);int MutationProcess(void);int FirstOrderMutation(void);
int SecondOrderMutation(void);int ThirdOrderMutation(void);double GenerateRandomNumber(void);
int GetCrossOverSite(void);int GetDiffCrossOverSiteTwo(int a, int b);int GetDiffCrossOverSite(int a);
void PrintParameters(double Rel[sizeComp], double Cost[sizeComp], double HRate[sizeComp]);
void PrintParametersToFile(double Rel[sizeComp], double Cost[sizeComp], double HRate[sizeComp],
double Loss_Fn,double REL);
void PrintDataBeforeCrossOver(void);void PrintDataAfterCrossOver(void);void
PrintDataBeforeMutation(void);
void PrintDataAfterMutation(void);void PrintOptimalResult(void);
int CompareChromosomes(void);
FILE *file1, *file2, *file3, *file4, *file5, *file6, *fileXoverOprt, *fileMutOprt;
char InputData[] = text file to provide input parameters,
AllComb[] = text file for dumping out all combination built in the program,
UniqComb[] = text file for dumpoing out all unique combinations built in the program,
InputData2[] = text file to input cost data of the reliability system,
UniqCostComb[]= text file for dumping out cost data produced by the program,
OptProcess[] = text fiel for dumping out the optimisation process,
XoverProcess[] = text fiel for dumping out crossover process,
MutProcess[] = text fiel for dumping out mutation process,
char startTime[9], endTime[9], todayDate[9];
double rand_num, CompDataReliability[sizeComp][sizeComp - 13],
CompDataCost[sizeComp][sizeComp - 13], SelectionMatrixRel[sizeComp][sizeComp-23],
SelectionMatrixCost[sizeComp][sizeComp-23], rel_time;
time_t start_time, end_time, processing_time;
float rand_value;
int ValueNumVar, AltData[sizeComp][sizeComp-23], numAlt=0, iRowAltData=0,
CompDataRow=0, CompDataCol=0, CompDataColLimit, SelectionMatrixRow=0, simRun=0,
sample_size, numVar, comb_ctr=0, FILEcomb_ctr=0, fileRow, fileCol, rowMatch,
EXPcomb_ctr=0, No_Nodes=0,,START_node=1, END_node=0, S_size=0,No_Cost=2, s_run=0,
ctrSearch=0, ChromSource=0, TRACEindicator=0;
unsigned long int EXPcomb_ctr=0;
int hr=0,row=0,col=0,
SYSTEM_FAILURE=0,SYSTEM_SUCCESS=0,CRITICAL_failure=0,CLEAN_FCompNo_
ctr=0, CompNo=0,FCompNo_ctr=0,LatticeRow=0,LatticeCol=0,
LatticeValue=0,NodeMatrixRow=0, NodeMatrixCol=0,
NodeMatrixCol2=0,NodeMatrixValue=0,FCompNo=0,FCompValue=0,
NodeCompFailCtr=0,SUCCESS_CTR=0,MaxLINK=0,MaxLINK2=0,
NodeMatrix[sizeComp][sizeComp][sizeComp],LatticeCOPY[sizeComp][sizeComp],
Lattice[sizeComp][sizeComp];
double CostVec[size],CHROMOSOME_1_R[sizeComp],CHROMOSOME_2_R[sizeComp],
CHROMOSOME_1_C[sizeComp],CHROMOSOME_2_C[sizeComp],RepCost,COF_temp=0.
0,
COF_Prod_Interv=0.0,COF_CompCost=0.0,COF_CompCost_OPT=0.0,COF=0.0,COF_OPT

```

---

```

=0.0,
    COF_perComp=0.0,COF_AllComp=0.0,COF_sys=0.0,REL=0.0,REL2=0.0,REL_OPT=0.99,
        Loss_Fn=0.0,TARGET_RELIABILITY=0.85,LOSS_OPT=20000.0,
        LOSS_OPT2=5000000.0;

int    ChromRow,CrossOverRow,CrossOverSite,a,CrossOverSite2,CrossOverSite3,retVal,
        iTEST=0,iTestVal=0,RelCtr=0;

double CHROMOSOME_1_R_MASTER[sizeComp][sizeComp-23],
        CHROMOSOME_1_C_MASTER[sizeComp][sizeComp-23],
        CHROMOSOME_2_R_MASTER[sizeComp][sizeComp-23],
        CHROMOSOME_2_C_MASTER[sizeComp][sizeComp-23],
        CHROMOSOME_1_R_COPY[sizeComp][sizeComp-23],
        CHROMOSOME_1_C_COPY[sizeComp][sizeComp-23],
        CHROMOSOME_2_R_COPY[sizeComp][sizeComp-23],
        CHROMOSOME_2_C_COPY[sizeComp][sizeComp-
23],rand_num2=0.0,rand_value,iTestVal=1.0,    CrossOverRelVal1,    CrossOverRelVal2,
        CrossOverCostVal1,CrossOverCostVal2,RelTest1,    RelTest2,    RelTest3,
        RelTest4;

int main(void)
{
    start_time = time(NULL);
    _strtime(startTime);
    _strdate(todayDate);
        if ((file1 = fopen(InputData,"r"))==NULL)
            printf("Error opening DataInput.TXT\n");
        else if ((file4 = fopen(InputData2,"r"))==NULL)
            printf("Error opening CostData.TXT\n");
        else if ((file2 = fopen(AllComb,"w+t"))==NULL)
            printf("Error opening AllCombination.TXT\n");
        else if ((file3 = fopen(UniqComb,"w+t"))==NULL)
            printf("Error opening UniqueReliabilityData.TXT\n");

        else if ((file5 = fopen(UniqCostComb,"w+t"))==NULL)
            printf("Error opening UniqueCostData.TXT\n");
        else if ((file6 = fopen(OptProcess,"w+t"))==NULL)
            printf("Error opening OptimisationProcess.TXT\n");
        else if ((fileXoverOprt = fopen(XoverProcess,"w+t"))==NULL)
            printf("Error opening XoverProcess.TXT\n");
        else if ((fileMutOprt = fopen(MutProcess,"w+t"))==NULL)
            printf("Error opening MutationProcess.TXT\n");
    else
    {
        GetData();
        printf("\n\tSYSTEM OPTIMISATION IN PROGRESS.....\n");
        for(ctrSearch=0;ctrSearch<ctrSearchValue;ctrSearch++)
        {
            comb_ctr=0;

```

```

        if(PRINTDATA)
        printf("\n Acquiring Sample Population: %d\n", ctrSearch+1);
        if((ctrSearch==0)||(ctrSearch==24)||(ctrSearch==49)||(ctrSearch==74))
        printf("\n Counter Search Number: %d\n", ctrSearch+1);
        ProcessData();iTestVal = time(NULL);    srand(iTestVal);iTEST =
rand();

        rg.set_new_seed(iTEST);
        retVal = CrossOverProcess();//cmm-cm-ccmmm
        MutationProcess();
    }
    PrintOptimalResult();PrintData();getchar();getchar();
}
printf("\n**** Finishing Program****\n ");return 0;
}
void GetData(void)
{
    int i;

    fscanf(file1,"%d", &sample_size);
    printf("\n **** Sample size ****: %d\n\n", sample_size);
    fscanf(file1,"%d", &numVar);
    printf("\n Number of Components: %d\n\n", numVar);
    GetAlternativesData();GetComponentData();
    printf("\n variables have been added \n");
    fseek(file1,0L,SEEK_CUR);    fscanf(file1,"%d", &No_Nodes);
    fscanf(file1,"%lf",&rel_time);fscanf(file1,"%d",&S_size);
    for(i=0;i<No_Cost;i++)
    fscanf(file1,"%lf",&CostVec[i]);
    for(LatticeRow=0;LatticeRow<No_Nodes;LatticeRow++)
        for(LatticeCol=0;LatticeCol<No_Nodes;LatticeCol++)
            {fscanf(file1,"%d",&Lattice[LatticeRow][LatticeCol]);}
    fscanf(file1,"%d",&MaxLINK);
    //NodeMatrix[NodeMatrixRow][NodeMatrixCol][NodeMatrixCol2]=0;
    for(NodeMatrixRow=0;NodeMatrixRow<No_Nodes;NodeMatrixRow++)
    for(NodeMatrixCol=0;NodeMatrixCol<No_Nodes;NodeMatrixCol++)
        {fscanf(file1,"%d",&MaxLINK2);
            MaxLINK = Lattice[NodeMatrixRow][NodeMatrixCol];
            if(!MaxLINK)
                {fscanf(file1,"%d",&MaxLINK2);continue;}
            else
                {
                    for(NodeMatrixCol2=0;NodeMatrixCol2<MaxLINK;NodeMatrixCol2++)

```

```

        { fscanf(file1, "%d", &MaxLINK2);
          if(!MaxLINK2)
            {NodeMatrixCol2=0;break;}

          else{
NodeMatrix[NodeMatrixRow][NodeMatrixCol][NodeMatrixCol2] = MaxLINK2;
                //NodeMatrixCol2++;}
          }}}
}

void GetAlternativesData(void)
{for(iRowAltData=0;iRowAltData<numVar;iRowAltData++)
  {numAlt=0;fscanf(file1, "%d", &AltData[iRowAltData][numAlt]);}
  for(iRowAltData=0;iRowAltData<numVar;iRowAltData++)
  {numAlt=0;printf("\nNumber of Alternatives for Component: %d = ", iRowAltData+1);
    printf("\t%d", AltData[iRowAltData][numAlt]);}
  printf("\n");EXPcomb_ctr=1;
  for(iRowAltData=0;iRowAltData<numVar;iRowAltData++)
  {numAlt=0;EXPcomb_ctr = EXPcomb_ctr * AltData[iRowAltData][numAlt];}}

void GetComponentData(void)
{for(CompDataRow=0;CompDataRow<numVar;CompDataRow++)
  {CompDataColLimit = AltData[CompDataRow][0];
    for(CompDataCol=0;CompDataCol<CompDataColLimit;CompDataCol++)
      {fscanf(file1, "%lf", &CompDataReliability[CompDataRow][CompDataCol]);
        fscanf(file4, "%lf", &CompDataCost[CompDataRow][CompDataCol]);}
    for(CompDataRow=0;CompDataRow<numVar;CompDataRow++)
      {CompDataColLimit = AltData[CompDataRow][0];
        printf("\n*** Set of Alternatives for Component: %d ***", CompDataRow+1);
        printf("\n\n");
        for(CompDataCol=0;CompDataCol<CompDataColLimit;CompDataCol++)
          {printf("%lf\t", CompDataReliability[CompDataRow][CompDataCol]);

//fscanf(file1, "%lf", &CompDataReliability[CompDataRow][CompDataCol]);
          //fscanf(file1, "%lf", &CompDataCost[CompDataRow][CompDataCol]);}
        printf("\n");
        for(CompDataCol=0;CompDataCol<CompDataColLimit;CompDataCol++)
          {printf("%lf\t", CompDataCost[CompDataRow][CompDataCol]);}
        printf("\n");}}

void ProcessData(void)
{//printf("\n Acquiring Sample Population\n");int iBCresult;double mathValue;
iTEST = rand();rg.set_new_seed(iTEST);simRun=0;
  do{ComponentSelection();iBCresult = BuildCombinations();if(iBCresult==99)
    {simRun++;continue;}

```

---

```

        mathValue = pow(numVar,InitialSampleSize);}
while (comb_ctr<InitialSampleSize);//while (comb_ctr<pow(numVar,InitialSampleSize));}
void ComponentSelection(void)
{int    SelectionRow,AltValue;
    for(SelectionRow=0;SelectionRow<numVar;SelectionRow++)
        {rand_num=GenerateRandomNumber();//rg.real_random();numAlt=0;AltValue=AltData[Sele
ctionRow][numAlt];    rand_value=(AltValue * rand_num) + 1;ValueNumVar =rand_value;
        SelectionMatrixRel[SelectionRow][numAlt]=
CompDataReliability[SelectionRow][ValueNumVar-1];
        SelectionMatrixCost[SelectionRow][numAlt]    =
CompDataCost[SelectionRow][ValueNumVar-1];}
int BuildCombinations(void)
{int TempValueRow=0,CheckCtr=0,CheckRow=0,NoMatchFound=0;
double  TempValue[sizeComp][sizeComp-22],    CheckTempValue,
        CheckSelectionValue,dRel;
/*    TRACEindicator=1;
        dRel = CalculateReliability(SelectionMatrixRel,SelectionMatrixCost);
        comb_ctr++;
*/    if(!comb_ctr)    { //printf("\n selection no: %d \n",simRun+1);
        TRACEindicator=1;dRel =
CalculateReliability(SelectionMatrixRel,SelectionMatrixCost);
        for(SelectionMatrixRow=0;SelectionMatrixRow<numVar;SelectionMatrixRow++)
            {
                fprintf(file3,"%lf ",SelectionMatrixRel[SelectionMatrixRow][numAlt]);
                fprintf(file5,"%lf ",SelectionMatrixCost[SelectionMatrixRow][numAlt]);
            }
            //fprintf(file3,"%lf ",REL);    fprintf(file3,"\n");fprintf(file5,"\n");
FILEcomb_ctr++;
        comb_ctr++;return 99;}fclose(file3);
if ((file3 = fopen(UniqComb,"r+w+t"))==NULL)
    printf("Error opening UniqueCombination.TXT\n");
else
fseek(file3,0L,SEEK_SET);
//for(fileRow=0;fileRow<comb_ctr;fileRow++)
for(fileRow=0;fileRow<FILEcomb_ctr;fileRow++)
    {numAlt=0;
    for(TempValueRow=0;TempValueRow<numVar;TempValueRow++)
        { //fscanf(file3,"%lf",&TempValue[fileRow][numAlt]);
        fscanf(file3,"%lf",&TempValue[TempValueRow][numAlt]);}
        CheckCtr=0;
        for(CheckRow=0;CheckRow<numVar;CheckRow++)
            {CheckTempValue
            =TempValue[CheckRow][numAlt];CheckSelectionValue=SelectionMatrixRel[CheckRow][nu
mAlt];
                if(CheckTempValue==CheckSelectionValue)

```

---

---

```

        CheckCtr++;
    else
        break;}
    if(CheckCtr >= numVar)
        break;
    if(fileRow==FILEcomb_ctr-1)
        {NoMatchFound=1;break;}
    else continue; }
    if(NoMatchFound)
        {NoMatchFound=0;TRACEindicator=1;
        dRel = CalculateReliability(SelectionMatrixRel,SelectionMatrixCost);
        //CalculateLosses(); fseek(file3,0L,SEEK_END);
    fseek(file5,0L,SEEK_END);
        fprintf(file3,"\n");fprintf(file5,"\n");
    for(SelectionMatrixRow=0;SelectionMatrixRow<numVar;SelectionMatrixRow++)
        { fprintf(file3,"%f
",SelectionMatrixRel[SelectionMatrixRow][numAlt]);
        fprintf(file5,"%f
",SelectionMatrixCost[SelectionMatrixRow][numAlt]); }
        //fprintf(file3,"%f ",REL); fprintf(file3,"\n");fprintf(file5,"\n");
    fclose(file3);fclose(file5);comb_ctr++; FILEcomb_ctr++;
    if ((file3 = fopen(UniqComb,"r+w+t"))==NULL)
        printf("Error opening UniqueCombination.TXT\n");
    else
        fseek(file3,0L,SEEK_CUR);
    if ((file5 = fopen(UniqCostComb,"r+w+t"))==NULL)
        printf("Error opening UniqueCostComb.TXT\n");
    else
        fseek(file5,0L,SEEK_CUR); }

    simRun++;return 1;}

void PrintData(void)
{
    if(comb_ctr==EXPcomb_ctr)
        {printf("\n Results found in '%d' simulation runs from total sample size of '%d'\n",
    simRun,sample_size);
        fseek(file2,0L,SEEK_END);
        numAlt=0;}

    printf("\nExpected Number of unique combinations (Analytical): %f\n", (float)
    EXPcomb_ctr);
    printf("\nTotal number of unique combinations found (Monte-Carlo Simulation): %d\n",
    FILEcomb_ctr);
    printf("\n Total amount of search space used: %d\n", RelCtr);
    fseek(file3,0L,SEEK_END); fseek(file5,0L,SEEK_END);
    numAlt=0;

```

---



---

```

    fprintf(file3, "\nNumber of Alternatives for '%d' Components are: ", numVar);
    for(iRowAltData=0; iRowAltData<numVar; iRowAltData++)
        fprintf(file3, "\t%d", AltData[iRowAltData][numAlt]);
    fprintf(file3, "\nTotal number of expected unique combinations (Analytical) : %ld\n",
EXPcomb_ctr);
    fprintf(file3, "\nTotal number of actual unique combinations found (Monte-Carlo Simulation):
%d\n", comb_ctr);
    fprintf(file3, "\nAllocated Sample size was '%d', but results were found in '%d' runs",
sample_size, simRun);    fcloseall(); end_time = time(NULL); _strtime(endTime);
    printf("\n Start time of the program : %s\t%s", todayDate, startTime);
    printf("\n End time of the program : %s\t%s", todayDate, endTime);
    processing_time = end_time - start_time; printf("\n Processing time of the program : %ld
seconds or %.3lf minutes \a", processing_time, (double)processing_time/60);}
double CalculateReliability(double SelectionMatrixRel[sizeComp][sizeComp-23], double
SelectionMatrixCost[sizeComp][sizeComp-23])
{
double
    Rel[sizeComp], Cost[sizeComp], HRate[sizeComp], F_time[sizeComp], RelVal, rand_num,
        SYSTEM_Fp, SYSTEM_RELIABILITY; CompOne, CompTwo,
        CompThree, CompFour, CompFive, CompSix, CompSeven, CompEight,
        CompNine, RelSubSysOne, RelSubSysTwo, RelSubSysThree;
int i, FAIL_comp, TESTctr=0, Comp_Failed[sizeComp];
    RelCtr++;
    for (hr=0; hr<numVar; hr++)
    {
        Rel[hr] = SelectionMatrixRel[hr][0];
        Cost[hr] = SelectionMatrixCost[hr][0];
        RelVal = Rel[hr]; HRate[hr] = -log(RelVal)/rel_time;}
    CompOne = Rel[0]; CompTwo = Rel[1]; CompThree = Rel[2]; CompFour = Rel[3];
    CompFive = Rel[4]; CompSix = Rel[5];    CompSeven = Rel[6]; CompEight =
Rel[7];
    CompNine = Rel[8]; RelSubSysOne = 1-((1-CompOne)*(1-CompTwo)*(1-
CompThree));
    RelSubSysTwo = 1-((1-CompFour)*(1-CompFive)*(1-CompSix)*(1-
CompSeven));
    RelSubSysThree = 1-((1-CompEight)*(1-CompNine));
    //production cost and intervention cost
    COF_Prod_Interv=0;
    for(i=0; i<No_Cost; i++)
    {COF_temp=CostVec[i]; COF_Prod_Interv=COF_Prod_Interv+COF_temp;}
    //cost of REPLACEMENT for each component = FACTOR * cost of
component
    COF_CompCost=0.0;
    for(i=0; i<numVar; i++)
    {COF_temp=Cost[i]; COF_CompCost =COF_CompCost
+COF_temp;}

```

---

---

```

        SUCCESS_CTR=0;SYSTEM_FAILURE=0;rg.set_new_seed(1651);
        SYSTEM_FAILURE = 0;CRITICAL_failure =0;   TESTctr=0;
        for(s_run=0;s_run<S_size;s_run++)
        {
r++) for(CLEAN_FCompNo_ctr=0;CLEAN_FCompNo_ctr<FCompNo_ctr;CLEAN_FCompNo_ctr++)
        {
            Comp_Failed[CLEAN_FCompNo_ctr]=NULL;   FCompNo_ctr=0;
            for(row=0;row<No_Nodes;row++)
                for(col=0;col<No_Nodes;col++)
                    {LatticeCOPY[row][col]=Lattice[row][col];}
            for (CompNo=1;CompNo<=numVar;CompNo++)
                {rand_num = rg.real_random();
                F_time[CompNo-1] = -log(rand_num)/HRate[CompNo-1];
                if(F_time[CompNo-1] < rel_time)
                    {Comp_Failed[FCompNo_ctr] = CompNo;
                    FCompNo_ctr++;}
                if(!FCompNo_ctr)
                    {SYSTEM_SUCCESS = 1;}
                else
                    {TESTctr++;LatticeRow=0;
                    for(LatticeCol=No_Nodes-1;LatticeCol>=0;LatticeCol--)
                        {LatticeValue = LatticeCOPY[LatticeRow][LatticeCol];
                        if (!LatticeValue)
                            continue;
                            if(LatticeValue)
                                { LatticeCol;
                                LatticeCOPY[LatticeRow][LatticeCol]=0;
                                LatticeCOPY[LatticeCol][LatticeRow]=0;NodeMatrixRow = LatticeRow;
                                NodeMatrixCol = LatticeCol;   NodeCompFailCtr=0;
                                for(NodeMatrixCol2=0;NodeMatrixCol2<LatticeValue;NodeMatrixCol2++)
                                    if(!FCompNo_ctr)
                                        break;
                                        NodeMatrixValue =
NodeMatrix[NodeMatrixRow][NodeMatrixCol][NodeMatrixCol2];
                                    if(!NodeMatrixValue)
                                        break;
                                    for(FCompNo=0;FCompNo<FCompNo_ctr;FCompNo++)
                                        FCompValue = Comp_Failed[FCompNo];
                                        if(FCompValue ==NodeMatrixValue)
                                            {
                                            NodeCompFailCtr++;break;   }
                                            else
                                                continue;   }}}
                                    if(NodeCompFailCtr>=LatticeValue)
                                        continue;   if(LatticeCol == No_Nodes - 1)
                                        {SYSTEM_SUCCESS = 1;break;}
                                        else{LatticeRow = LatticeCol;LatticeCol = No_Nodes ;continue;}} }
                                        if(SYSTEM_SUCCESS){SUCCESS_CTR++;;SYSTEM_SUCCESS = 0;}

```

---

```

each failure
else{SYSTEM_FAILURE++;COF_AllComp = 0;//cost of

for(i=0;i<FCompNo_ctr;i++)
    {FAIL_comp = Comp_Failed[i];
    RepCost = Cost[FAIL_comp-1] * FACTOR;
    COF_perComp = Cost[FAIL_comp-1] + RepCost;
    COF_AllComp = COF_AllComp + COF_perComp;}
COF_sys = COF_sys + COF_AllComp + COF_Prod_Interv; }
SYSTEM_RELIABILITY = (double)SUCCESS_CTR / (double) S_size;
SYSTEM_Fp = (double) SYSTEM_FAILURE / (double)S_size;
REL = 1 -SYSTEM_Fp; COF = COF_sys/S_size;COF_sys=0.0;
Loss_Fn = COF_CompCost + COF;
REL2 = RelSubSysOne*RelSubSysTwo*RelSubSysThree;
fprintf(file2, "\nReliability, %lf,Reliability(Analytical), %lf, Loss Fn, %lf,
Comp Cost, %lf, Risk(K), %lf, Counter No, %d\n", REL,
REL2, Loss_Fn, COF_CompCost, COF, ctrSearch+1);
if(TRACEindicator)
{if(Loss_Fn<LOSS_OPT)
    {printf("\n\t\t***** close match found - Reliability: %lf AND Loss Fn: %lf(Counter No:
%d)\n", REL, Loss_Fn, ctrSearch+1);fprintf(file6, "\nReliability, %lf, AND Loss Fn, %lf, Q, %lf, K, %lf,
Counter No, %d\n", REL, Loss_Fn, COF_CompCost, COF, ctrSearch+1);
    PrintParametersToFile(Rel, Cost, HRate, Loss_Fn, REL);LOSS_OPT2=LOSS_OPT;
    for(i=0;i<numVar;i++)
        {CHROMOSOME_2_R[i]= CHROMOSOME_1_R[i];
        CHROMOSOME_2_C[i] = CHROMOSOME_1_C[i];}
    LOSS_OPT=Loss_Fn;REL_OPT = REL;COF_OPT=COF;
    COF_CompCost_OPT = COF_CompCost;
    for(i=0;i<numVar;i++)
        {CHROMOSOME_1_R[i]=Rel[i];
        CHROMOSOME_1_C[i]=Cost[i];}
    if((Loss_Fn<LOSS_OPT2)&&(Loss_Fn>LOSS_OPT)) //if(Loss_Fn>LOSS_OPT)
        {LOSS_OPT2=Loss_Fn;for(i=0;i<numVar;i++)
            {CHROMOSOME_2_R[i]=Rel[i];CHROMOSOME_2_C[i]=ost[i];} }
    TRACEindicator=0;}
else{if(Loss_Fn<LOSS_OPT)
    {if(ChromSource==1)
        {printf("\n\t\t F.O.C. close match found - Reliability: %lf AND Loss_Fn:
%lf (Counter No: %d)\n", REL, Loss_Fn, ctrSearch+1);fprintf(file6, "\nReliability, %lf, AND Loss_Fn,
%lf, Q, %lf, K, %lf, Counter No, %d\n", REL,
Loss_Fn, COF_CompCost, COF, ctrSearch+1);PrintParametersToFile(Rel, Cost, HRate, Loss_Fn, REL);}
        if(ChromSource==2){printf("\n\t\t S.O.C. close match found - Reliability:
%lf AND Loss_Fn: %lf (Counter No: %d)\n", REL, Loss_Fn, ctrSearch+1);fprintf(file6, "\nReliability,
%lf, AND Loss_Fn, %lf, Q, %lf, K, %lf, Counter No, %d\n", REL,
Loss_Fn, COF_CompCost, COF, ctrSearch+1);
        PrintParametersToFile(Rel, Cost, HRate, Loss_Fn, REL);}
        if(ChromSource==3)
            {printf("\n\t\t T.O.C. close match found - Reliability: %lf AND Loss_Fn:
%lf (Counter No: %d)\n", REL, Loss_Fn, ctrSearch+1);fprintf(file6, "\nReliability, %lf, AND Loss_Fn,

```

---

```

%lf, Q,%lf,K,%lf, Counter No, %d\n", REL, Loss_Fn,COF_CompCost,COF,ctrSearch+1);
    PrintParametersToFile(Rel,Cost,HRate,Loss_Fn,REL);}

    if(ChromSource==4)

    {printf("\n\t\t F.O.M. close match found - Reliability: %lf AND Loss_Fn: %lf (Counter No: %d)\n",
REL, Loss_Fn,ctrSearch+1);fprintf(file6, "\nReliability, %lf, AND Loss_Fn, %lf, Q,%lf,K,%lf,
Counter No, %d\n", REL, Loss_Fn,COF_CompCost,COF,ctrSearch+1);
    PrintParametersToFile(Rel,Cost,HRate,Loss_Fn,REL);}

    if(ChromSource==5)

    {printf("\n\t\t S.O.M. close match found - Reliability: %lf AND Loss_Fn:
%lf (Counter No: %d)\n", REL, Loss_Fn,ctrSearch+1);fprintf(file6, "\nReliability, %lf, AND Loss_Fn,
%lf, Q,%lf,K,%lf, Counter No, %d\n", REL, Loss_Fn,COF_CompCost,COF,ctrSearch+1);
    PrintParametersToFile(Rel,Cost,HRate,Loss_Fn,REL);}

    if(ChromSource==6)

    {printf("\n\t\t T.O.M. close match found - Reliability: %lf AND Loss_Fn: %lf (Counter No: %d)\n",
REL, Loss_Fn,ctrSearch+1);fprintf(file6, "\nReliability, %lf, AND Loss_Fn, %lf, Q,%lf,K,%lf,
Counter No, %d\n", REL, Loss_Fn,COF_CompCost,COF,ctrSearch+1);
    PrintParametersToFile(Rel,Cost,HRate,Loss_Fn,REL);}

    ChromSource=0; LOSS_OPT2=LOSS_OPT;

    for(i=0;i<numVar;i++)

    {CHROMOSOME_2_R[i]=CHROMOSOME_1_R[i];CHROMOSOME_2_C[i]=CHROMOSOME_1_
C[i];}  LOSS_OPT=Loss_Fn;REL_OPT = REL;COF_OPT=COF;  COF_CompCost_OPT =
COF_CompCost;

REL,SYSTEM_Fp,LOSS_OPT,COF_CompCost,COF);for(i=0;i<numVar;i++)

    {CHROMOSOME_1_R[i]=Rel[i];CHROMOSOME_1_C[i]=Cost[i];  }

    fprintf(file2, "\n");}

    if((Loss_Fn<LOSS_OPT2)&&(Loss_Fn>LOSS_OPT))//<=?

    {LOSS_OPT2=Loss_Fn;

    for(i=0;i<numVar;i++)

    {CHROMOSOME_2_R[i]=Rel[i];CHROMOSOME_2_C[i]=Cost[i];  }}}

return REL;}

int CrossOverProcess(void)

{int retValue=0;retValue = FirstOrderCrossOverProcess();

    if(retValue==999)

        return retValue;

    SecondOrderCrossOverProcess();

    if(retValue==999)

        return retValue;

    ThirdOrderCrossOverProcess();

    if(retValue==999)

        return retValue;

    return 0;}

int FirstOrderCrossOverProcess(void)

```

---

```

{
int iCOS, cosTEST, IDENTICAL_CROSSOVERSITE=0, cosLIST[size];
    iTEST = rand();
    for(ChromRow=0;ChromRow<numVar;ChromRow++)

        {CHROMOSOME_1_R_MASTER[ChromRow][0]=CHROMOSOME_1_R[ChromRow];
        CHROMOSOME_1_C_MASTER[ChromRow][0]=CHROMOSOME_1_C[ChromRow];
        CHROMOSOME_2_R_MASTER[ChromRow][0]=CHROMOSOME_2_R[ChromRow];
        CHROMOSOME_2_C_MASTER[ChromRow][0]=CHROMOSOME_2_C[ChromRow]; }
    for(CrossOverRow=0;CrossOverRow<CrossOverRun;CrossOverRow++)
    {for(ChromRow=0;ChromRow<numVar;ChromRow++)

        {CHROMOSOME_1_R_COPY[ChromRow][0]=CHROMOSOME_1_R[ChromRow];
          CHROMOSOME_1_C_COPY[ChromRow][0]=
        CHROMOSOME_1_C[ChromRow];
        CHROMOSOME_2_R_COPY[ChromRow][0]=  CHROMOSOME_2_R[ChromRow];
          CHROMOSOME_2_C_COPY[ChromRow][0]=
        CHROMOSOME_2_C[ChromRow];}

        if(PRINTDATA)
            PrintDataBeforeCrossOver();
        iTEST = rand();rg.set_new_seed(iTEST);
        CrossOverSite =  GetCrossOverSite();//rand_value;
        if(PRINTDATA)
            printf("\n CrossOver Site: %d\n", CrossOverSite);
        for(iCOS=0;iCOS<numVar;iCOS++)
            {cosTEST = cosLIST[iCOS];
            if(cosTEST)
                {if(CrossOverSite==cosTEST)
                    {IDENTICAL_CROSSOVERSITE = 1;break;}}
                else
                    break;}

        if(IDENTICAL_CROSSOVERSITE)
            {IDENTICAL_CROSSOVERSITE=0;    continue;    }
        else
            cosLIST[CrossOverRow]=CrossOverSite;
        CrossOverRelVal1    =    CHROMOSOME_1_R_COPY[CrossOverSite-
1][0];
        CrossOverRelVal2    =    CHROMOSOME_2_R_COPY[CrossOverSite-
1][0];

        CHROMOSOME_1_R_COPY[CrossOverSite-1][0] =    CrossOverRelVal2;
        CHROMOSOME_2_R_COPY[CrossOverSite-1][0] =    CrossOverRelVal1;
        if(CrossOverRelVal1==CrossOverRelVal2)
            {    continue;    }

```

---

```

CrossOverCostVal1      =      CHROMOSOME_1_C_COPY[CrossOverSite-
1][0];
CrossOverCostVal2      =      CHROMOSOME_2_C_COPY[CrossOverSite-
1][0];
CHROMOSOME_1_C_COPY[CrossOverSite-1][0] =      CrossOverCostVal2;
CHROMOSOME_2_C_COPY[CrossOverSite-1][0] =      CrossOverCostVal1;
if(PRINTDATA)
    PrintDataAfterCrossOver();

ChromSource = 1;
RelTest1 = CalculateReliability(CHROMOSOME_1_R_COPY,CHROMOSOME_1_C_COPY);
    fprintf(fileXoverOprt, "\nReliability, %f, AND Loss_Fn, %f, Q,%f,K,%f,Counter
No, %d, X-over Type, %d, Generaton NO,%d\n", RelTest1,
Loss_Fn,COF_CompCost,COF,ctrSearch+1,1,RelCtr);
    ChromSource = 1;

    RelTest2 =
CalculateReliability(CHROMOSOME_2_R_COPY,CHROMOSOME_2_C_COPY);
    fprintf(fileXoverOprt, "\nReliability, %f, AND Loss_Fn, %f,Q,%f,K,%f,Counter
No, %d, X-over Type, %d, Generaton NO,%d\n", RelTest2,
Loss_Fn,COF_CompCost,COF,ctrSearch+1,1,RelCtr);
    if(LOSS_OPT==LOSS_OPT2)
        {return 999;}
    if (CompareChromosomes())
        return 999;}return 0;}

int SecondOrderCrossOverProcess(void)
{
    int noCHECK1=0,noCHECK2=0;
    for(CrossOverRow=0;CrossOverRow<CrossOverRun;CrossOverRow++)
    {for(ChromRow=0;ChromRow<numVar;ChromRow++)
        {CHROMOSOME_1_R_COPY[ChromRow][0]
        =CHROMOSOME_1_R[ChromRow];
        CHROMOSOME_1_C_COPY[ChromRow][0]
        =CHROMOSOME_1_C[ChromRow];
        CHROMOSOME_2_R_COPY[ChromRow][0]
        =CHROMOSOME_2_R[ChromRow];
        CHROMOSOME_2_C_COPY[ChromRow][0]
        =CHROMOSOME_2_C[ChromRow];    }
        if(PRINTDATA)
            PrintDataBeforeCrossOver();
        iTEST = rand();rg.set_new_seed(iTEST);
        CrossOverSite =      GetCrossOverSite();//rand_value;
        if(PRINTDATA)
            printf("\n CrossOver Site: %d\n", CrossOverSite);
        CrossOverRelVal1      =      CHROMOSOME_1_R_COPY[CrossOverSite-
1][0];
        CrossOverRelVal2      =      CHROMOSOME_2_R_COPY[CrossOverSite-
1][0];

```

---

---

```

        CHROMOSOME_1_R_COPY[CrossOverSite-1][0] = CrossOverRelVal2;
        CHROMOSOME_2_R_COPY[CrossOverSite-1][0] = CrossOverRelVal1;
        if(CrossOverRelVal1==CrossOverRelVal2)
            noCHECK1=1;
        else
            noCHECK1=0;

        CrossOverCostVal1 = CHROMOSOME_1_C_COPY[CrossOverSite-
1][0];
        CrossOverCostVal2 = CHROMOSOME_2_C_COPY[CrossOverSite-
1][0];

        CHROMOSOME_1_C_COPY[CrossOverSite-1][0] = CrossOverCostVal2;
        CHROMOSOME_2_C_COPY[CrossOverSite-1][0] = CrossOverCostVal1;
        CrossOverSite2 = GetCrossOverSite();//rand_value;
        if(CrossOverSite2==CrossOverSite)
            CrossOverSite2=GetDiffCrossOverSite(CrossOverSite);
        if(PRINTDATA)
            printf("\n CrossOver Site: %d\n", CrossOverSite2);
        CrossOverRelVal1 = CHROMOSOME_1_R_COPY[CrossOverSite2-
1][0];
        CrossOverRelVal2 = CHROMOSOME_2_R_COPY[CrossOverSite2-
1][0];

        CHROMOSOME_1_R_COPY[CrossOverSite2-1][0]= CrossOverRelVal2;
        CHROMOSOME_2_R_COPY[CrossOverSite2-1][0]= CrossOverRelVal1;
        if(CrossOverRelVal1==CrossOverRelVal2)
            noCHECK2=1;
        else
            noCHECK2=0;

        CrossOverCostVal1 = CHROMOSOME_1_C_COPY[CrossOverSite2-
1][0];
        CrossOverCostVal2 = CHROMOSOME_2_C_COPY[CrossOverSite2-
1][0];

        CHROMOSOME_1_C_COPY[CrossOverSite2-1][0]= CrossOverCostVal2;
        CHROMOSOME_2_C_COPY[CrossOverSite2-1][0]= CrossOverCostVal1;
        if(PRINTDATA)
            PrintDataAfterCrossOver();
        if(noCHECK1==noCHECK2)
            {noCHECK1=0; noCHECK2=0; continue; }
        ChromSource = 2;

        RelTest3 =
        CalculateReliability(CHROMOSOME_1_R_COPY,CHROMOSOME_1_C_COPY);
        fprintf(fileXoverOprt, "\nReliability, %lf, AND Loss_Fn, %lf,Q,%lf,K,%lf, Counter
No, %d, X-over Type, %d , Generaton NO,%d\n", RelTest3,
        Loss_Fn,COF_CompCost,COF,ctrSearch+1,2,RelCtr);

```

---

---

```

ChromSource = 2;

    RelTest4 =
CalculateReliability(CHROMOSOME_2_R_COPY,CHROMOSOME_2_C_COPY);
        fprintf(fileXoverOprt, "\nReliability, %f, AND Loss_Fn, %f,Q,%f,K,%f, Counter
No, %d, X-over Type, %d, Generaton NO,%d\n", RelTest4,
Loss_Fn,COF_CompCost,COF,ctrSearch+1,2,RelCtr);
if (CompareChromosomes())
        return 999;}return 0;}

int ThirdOrderCrossOverProcess(void)
{
    int noCHECK1=0,noCHECK2=0,noCHECK3=0;
    for(CrossOverRow=0;CrossOverRow<CrossOverRun;CrossOverRow++)
    {for(ChromRow=0;ChromRow<numVar;ChromRow++)
        {CHROMOSOME_1_R_COPY[ChromRow][0]
        =CHROMOSOME_1_R[ChromRow];
        CHROMOSOME_1_C_COPY[ChromRow][0]      =CHROMOSOME_1_C[ChromRow];
        CHROMOSOME_2_R_COPY[ChromRow][0]
        =CHROMOSOME_2_R[ChromRow];
        CHROMOSOME_2_C_COPY[ChromRow][0]      =CHROMOSOME_2_C[ChromRow];}
        if(PRINTDATA)
            PrintDataBeforeCrossOver();
        CrossOverSite =      GetCrossOverSite();//rand_value;
        if(PRINTDATA)
            printf("\n CrossOver Site: %d\n", CrossOverSite);
        CrossOverRelVal1      =      CHROMOSOME_1_R_COPY[CrossOverSite-
1][0];
        CrossOverRelVal2      =      CHROMOSOME_2_R_COPY[CrossOverSite-
1][0];
        CHROMOSOME_1_R_COPY[CrossOverSite-1][0] =      CrossOverRelVal2;
        CHROMOSOME_2_R_COPY[CrossOverSite-1][0] =      CrossOverRelVal1;
        if(CrossOverRelVal1==CrossOverRelVal2)
            noCHECK1=1;
        else
            noCHECK1=0;
        CrossOverCostVal1      =      CHROMOSOME_1_C_COPY[CrossOverSite-
1][0];
        CrossOverCostVal2      =      CHROMOSOME_2_C_COPY[CrossOverSite-
1][0];
        CHROMOSOME_1_C_COPY[CrossOverSite-1][0] =      CrossOverCostVal2;
        CHROMOSOME_2_C_COPY[CrossOverSite-1][0] =      CrossOverCostVal1;
        CrossOverSite2 =      GetCrossOverSite();//rand_value;
        if(CrossOverSite2==CrossOverSite)
            CrossOverSite2=GetDiffCrossOverSite(CrossOverSite);
        if(PRINTDATA)

```

---



---

```

        printf("\n CrossOver Site: %d\n", CrossOverSite2);
CrossOverRelVal1      =      CHROMOSOME_1_R_COPY[CrossOverSite2-
1][0];
CrossOverRelVal2      =      CHROMOSOME_2_R_COPY[CrossOverSite2-
1][0];
CHROMOSOME_1_R_COPY[CrossOverSite2-1][0]=      CrossOverRelVal2;
CHROMOSOME_2_R_COPY[CrossOverSite2-1][0]=      CrossOverRelVal1;
if(CrossOverRelVal1==CrossOverRelVal2)
        noCHECK2=1;
else
        noCHECK2=0;
CrossOverCostVal1     =      CHROMOSOME_1_C_COPY[CrossOverSite2-
1][0];
CrossOverCostVal2     =      CHROMOSOME_2_C_COPY[CrossOverSite2-
1][0];
CHROMOSOME_1_C_COPY[CrossOverSite2-1][0]=      CrossOverCostVal2;
CHROMOSOME_2_C_COPY[CrossOverSite2-1][0]=      CrossOverCostVal1;
CrossOverSite3 =      GetCrossOverSite();//rand_value;
if((CrossOverSite3==CrossOverSite)||((CrossOverSite3==CrossOverSite2))
        CrossOverSite3=GetDiffCrossOverSiteTwo(CrossOverSite,
CrossOverSite2);
if(PRINTDATA)
        printf("\n CrossOver Site: %d\n", CrossOverSite3);
CrossOverRelVal1      =      CHROMOSOME_1_R_COPY[CrossOverSite3-
1][0];
CrossOverRelVal2      =      CHROMOSOME_2_R_COPY[CrossOverSite3-
1][0];
CHROMOSOME_1_R_COPY[CrossOverSite3-1][0]=      CrossOverRelVal2;
CHROMOSOME_2_R_COPY[CrossOverSite3-1][0]=      CrossOverRelVal1;
if(CrossOverRelVal1==CrossOverRelVal2)
        noCHECK3=1;
else
        noCHECK3=0;
CrossOverCostVal1     =      CHROMOSOME_1_C_COPY[CrossOverSite3-
1][0];
CrossOverCostVal2     =      CHROMOSOME_2_C_COPY[CrossOverSite3-
1][0];
CHROMOSOME_1_C_COPY[CrossOverSite3-1][0]=      CrossOverCostVal2;
CHROMOSOME_2_C_COPY[CrossOverSite3-1][0]=      CrossOverCostVal1;
if(PRINTDATA)
        PrintDataAfterCrossOver();
if((noCHECK1==noCHECK2)&&(noCHECK2==noCHECK3))
        {noCHECK1=0; noCHECK2=0; noCHECK3=0;continue;}
ChromSource = 3;

```

---

```

        RelTest3 =
CalculateReliability(CHROMOSOME_1_R_COPY,CHROMOSOME_1_C_COPY);
        fprintf(fileXoverOprt, "\nReliability, %lf, AND Loss_Fn, %lf,Q,%lf,K,%lf, Counter
No, %d, X-over Type, %d, Generaton No, %d\n", RelTest3,
Loss_Fn,COF_CompCost,COF,ctrSearch+1,3,RelCtr);
        ChromSource = 3;

        RelTest4 =
CalculateReliability(CHROMOSOME_2_R_COPY,CHROMOSOME_2_C_COPY);
        fprintf(fileXoverOprt, "\nReliability, %lf, AND Loss_Fn, %lf, Q,%lf,K,%lf, Counter No, %d,
X-over Type, %d, Generaton No, %d\n", RelTest4,
Loss_Fn,COF_CompCost,COF,ctrSearch+1,3,RelCtr);
        if (CompareChromosomes())
                return 999;}return 0;}

int MutationProcess(void)
{
        FirstOrderMutation();
        SecondOrderMutation();
        ThirdOrderMutation(); return 0;}

int FirstOrderMutation(void)
{int AltValue=0, ctr=0;double chrmValue, DataValue;
        for(CrossOverRow=0;CrossOverRow<MutationRun;CrossOverRow++)
        {for(ChromRow=0;ChromRow<numVar;ChromRow++){
                CHROMOSOME_1_R_COPY[ChromRow][0] =CHROMOSOME_1_R[ChromRow];
                CHROMOSOME_1_C_COPY[ChromRow][0]
                =CHROMOSOME_1_C[ChromRow];}
        if(PRINTDATA)
                PrintDataBeforeMutation();
                iTEST = rand();rg.set_new_seed(iTEST);
        CrossOverSite = GetCrossOverSite();
        if(PRINTDATA)
                printf("\n MUTATION Site: %d\n", CrossOverSite);
        numAlt = 0; AltValue=AltData[CrossOverSite-1][numAlt];
        rand_value=(AltValue * rand_num) + 1; ValueNumVar = rand_value;
        chrmValue= CHROMOSOME_1_R_COPY[CrossOverSite-1][0];
        DataValue= CompDataReliability[CrossOverSite-1][ValueNumVar-1];
        if(chrmValue==DataValue)
        { for(ctr=0;ctr<=MutationRun;ctr++)
                { if(chrmValue==DataValue) {
                        rand_num = GenerateRandomNumber();
                        rand_value = (AltValue * rand_num) + 1;
                        ValueNumVar = rand_value;
                        chrmValue = CHROMOSOME_1_R_COPY[CrossOverSite-1][0];
                        DataValue = CompDataReliability[CrossOverSite-1][ValueNumVar-
1];}
                else

```

```

        break; } }

    CHROMOSOME_1_R_COPY[CrossOverSite-1][0]= CompDataReliability[CrossOverSite-1][ValueNumVar-1];
    CHROMOSOME_1_C_COPY[CrossOverSite-1][0]= CompDataCost[CrossOverSite-1][ValueNumVar-1];
    if(PRINTDATA)
        PrintDataAfterMutation();
    ChromSource = 4;
    RelTest1 = CalculateReliability(CHROMOSOME_1_R_COPY,CHROMOSOME_1_C_COPY);
    fprintf(fileMutOprt, "\nReliability, %lf, AND Loss_Fn, %lf,Q,%lf,K,%lf, Counter No, %d, Mutation Type, %d, Generation No, %d\n", RelTest1, Loss_Fn,COF_CompCost,COF,ctrSearch+1,1,RelCtr);
    } return 0;}

int SecondOrderMutation(void)
{int AltValue=0,ctr=0;double chrmValue, DataValue;
for(CrossOverRow=0;CrossOverRow<MutationRun;CrossOverRow++)
    {for(ChromRow=0;ChromRow<numVar;ChromRow++)
        {CHROMOSOME_1_R_COPY[ChromRow][0]
        =CHROMOSOME_1_R[ChromRow];
        CHROMOSOME_1_C_COPY[ChromRow][0]      =CHROMOSOME_1_C[ChromRow];
        }
        if(PRINTDATA)
            PrintDataBeforeMutation();
            iTEST = rand();rg.set_new_seed(iTEST);
            CrossOverSite = GetCrossOverSite();
            if(PRINTDATA)
                printf("\n MUTATION Site: %d\n", CrossOverSite);
            numAlt = 0;
            AltValue = AltData[CrossOverSite-1][numAlt];
            rand_value = (AltValue * rand_num) + 1;
            ValueNumVar = rand_value;
            chrmValue = CHROMOSOME_1_R_COPY[CrossOverSite-1][0];
            DataValue = CompDataReliability[CrossOverSite-1][ValueNumVar-1];
            if(chrmValue==DataValue)
                { for(ctr=0;ctr<=MutationRun;ctr++) { if(chrmValue==DataValue)
                    { rand_num = GenerateRandomNumber(); rand_value = (AltValue *
                    rand_num) + 1;
                    ValueNumVar = rand_value;
                    chrmValue = CHROMOSOME_1_R_COPY[CrossOverSite-1][0];
                    DataValue = CompDataReliability[CrossOverSite-1][ValueNumVar-1]; }
                    else break; } }
            CHROMOSOME_1_R_COPY[CrossOverSite-1][0]= CompDataReliability[CrossOverSite-1][ValueNumVar-1];
            CHROMOSOME_1_C_COPY[CrossOverSite-1][0]= CompDataCost[CrossOverSite-1][ValueNumVar-1];

```

---

```

CrossOverSite2 =      GetCrossOverSite();
if(CrossOverSite2==CrossOverSite)
    CrossOverSite2=GetDiffCrossOverSite(CrossOverSite);
if(PRINTDATA)
    printf("\n MUTATION Site: %d\n", CrossOverSite2);
AltValue      =      AltData[CrossOverSite2-1][numAlt];
rand_value    =      (AltValue * rand_num) + 1;
ValueNumVar =      rand_value;
chrnValue     =      CHROMOSOME_1_R_COPY[CrossOverSite2-1][0];
DataValue     =      CompDataReliability[CrossOverSite2-1][ValueNumVar-1];
if(chrnValue==DataValue)
{ for(ctr=0;ctr<=MutationRun;ctr++)
    { if(chrnValue==DataValue)
        {      rand_num = GenerateRandomNumber();
            rand_value = (AltValue * rand_num) + 1;
            ValueNumVar =      rand_value;
            chrnValue     =      CHROMOSOME_1_R_COPY[CrossOverSite2-1][0];
            DataValue     =      CompDataReliability[CrossOverSite2-
1][ValueNumVar-1];
        }
        else
            break; } }
CHROMOSOME_1_R_COPY[CrossOverSite2-1][0]= CompDataReliability[CrossOverSite2-
1][ValueNumVar-1]; CHROMOSOME_1_C_COPY[CrossOverSite2-1][0] =
CompDataCost[CrossOverSite2-1][ValueNumVar-1];
if(PRINTDATA)
    PrintDataAfterMutation();
ChromSource = 5;
RelTest1 = CalculateReliability(CHROMOSOME_1_R_COPY,CHROMOSOME_1_C_COPY);
fprintf(fileMutOprt, "\nReliability, %lf, AND Loss_Fn, %lf,Q,%lf,K,%lf, Counter No, %d, Mutation
Type, %d, Generation No, %d\n", RelTest1, Loss_Fn,COF_CompCost,COF,ctrSearch+1,2,RelCtr);
}
return 0;}

int ThirdOrderMutation(void)
{int AltValue=0,ctr=0;double chrnValue, DataValue;
    for(CrossOverRow=0;CrossOverRow<MutationRun;CrossOverRow++)
        {for(ChromRow=0;ChromRow<numVar;ChromRow++)
            {CHROMOSOME_1_R_COPY[ChromRow][0]
            =CHROMOSOME_1_R[ChromRow];
            CHROMOSOME_1_C_COPY[ChromRow][0]      =CHROMOSOME_1_C[ChromRow];
            }
        if(PRINTDATA)
            PrintDataBeforeMutation();

```

---

---

```

        iTEST = rand(); rg.set_new_seed(iTEST);
CrossOverSite = GetCrossOverSite();
if(PRINTDATA)
    printf("\n MUTATION Site: %d\n", CrossOverSite);
numAlt = 0;
AltValue=AltData[CrossOverSite-1][numAlt];
rand_value=(AltValue * rand_num) + 1;
ValueNumVar = rand_value;
    if(chrmValue==DataValue)
    { for(ctr=0;ctr<=MutationRun;ctr++)
        { if(chrmValue==DataValue)
            { rand_num = GenerateRandomNumber(); rand_value = (AltValue *
rand_num) + 1;
                ValueNumVar = rand_value;
                chrmValue = CHROMOSOME_1_R_COPY[CrossOverSite-1][0];
                DataValue = CompDataReliability[CrossOverSite-1][ValueNumVar-
1]; }
            else
                break; } }
        CHROMOSOME_1_R_COPY[CrossOverSite-1][0]= CompDataReliability[CrossOverSite-
1][ValueNumVar-1];
        CHROMOSOME_1_C_COPY[CrossOverSite-1][0]= CompDataCost[CrossOverSite-
1][ValueNumVar-1];
        CrossOverSite2=GetCrossOverSite();
        if(CrossOverSite2==CrossOverSite)
            CrossOverSite2=GetDiffCrossOverSite(CrossOverSite);
        if(PRINTDATA)
            printf("\n MUTATION Site: %d\n", CrossOverSite2);
        AltValue =AltData[CrossOverSite2-1][numAlt];
        rand_value=(AltValue * rand_num) + 1;
        ValueNumVar = rand_value;
        chrmValue= CHROMOSOME_1_R_COPY[CrossOverSite2-1][0];
        DataValue= CompDataReliability[CrossOverSite2-1][ValueNumVar-1];
        if(chrmValue==DataValue)
        { for(ctr=0;ctr<=MutationRun;ctr++)
            { if(chrmValue==DataValue)
                {
                    rand_num = GenerateRandomNumber();
                    rand_value = (AltValue * rand_num) + 1;
                    ValueNumVar = rand_value;
                    chrmValue = CHROMOSOME_1_R_COPY[CrossOverSite2-1][0];
                    DataValue = CompDataReliability[CrossOverSite2-
1][ValueNumVar-1];
                }
            }
        }

```

---

```

else
    break; } }

CHROMOSOME_1_R_COPY[CrossOverSite2-1][0]= CompDataReliability[CrossOverSite2-
1][ValueNumVar-1]; CHROMOSOME_1_C_COPY[CrossOverSite2-1][0] =
CompDataCost[CrossOverSite2-1][ValueNumVar-1]; CrossOverSite3 =
    GetCrossOverSite();//rand_value;

if((CrossOverSite3==CrossOverSite)||(CrossOverSite3==CrossOverSite2))
    CrossOverSite3=GetDiffCrossOverSiteTwo(CrossOverSite, CrossOverSite2);
if(PRINTDATA)
    printf("\n MUTATION Site: %d\n", CrossOverSite3);
AltValue =AltData[CrossOverSite3-1][numAlt];
rand_value=(AltValue * rand_num) + 1;
ValueNumVar = rand_value; chrnValue =
CHROMOSOME_1_R_COPY[CrossOverSite3-1][0];
DataValue = CompDataReliability[CrossOverSite3-1][ValueNumVar-1];
if(chrnValue==DataValue)
{ for(ctr=0;ctr<=MutationRun;ctr++)
    { if(chrnValue==DataValue)
        { rand_num = GenerateRandomNumber();
          rand_value = (AltValue * rand_num) + 1;
          ValueNumVar = rand_value;
          chrnValue = CHROMOSOME_1_R_COPY[CrossOverSite3-1][0];
          DataValue = CompDataReliability[CrossOverSite3-
1][ValueNumVar-1];
        }
    else
        break; } }

    CHROMOSOME_1_R_COPY[CrossOverSite3-1][0] =
CompDataReliability[CrossOverSite3-1][ValueNumVar-1];
CHROMOSOME_1_C_COPY[CrossOverSite3-1][0]= CompDataCost[CrossOverSite3-
1][ValueNumVar-1];
if(PRINTDATA)
    PrintDataAfterMutation();
ChromSource = 6; RelTest1 =
aculateReliability(CHROMOSOME_1_R_COPY,CHROMOSOME_1_C_COPY);
    fprintf(fileMutOprt, "\nReliability, %lf, AND Loss_Fn, %lf, Q,%lf,K,%lf, Counter No, %d, Mutation
Type, %d, Generation No, %d\n", RelTest1, Loss_Fn,COF_CompCost,COF,ctrSearch+1,3,RelCtr);
    } return 0;}

double GenerateRandomNumber(void)
{int a;for(a=1;a<25;a++)
    rand_num = rg.real_random();return rand_num;}

int GetCrossOverSite(void)
{rand_num=GenerateRandomNumber();rand_value=(numVar * rand_num) + 1;return rand_value;}

int GetDiffCrossOverSite(int a)

```

---

```

{int rnd_val;iTEST = rand();      rg.set_new_seed(iTEST);
    rand_num=GenerateRandomNumber();    rnd_val=(numVar * rand_num) + 1;
    if(rnd_val==a)
        {iTEST = rand();rg.set_new_seed(iTEST);
            GetDiffCrossOverSite(a);return rnd_val;}
    else
        return rnd_val;}

int GetDiffCrossOverSiteTwo(int a, int b)
{
    int rnd_val2;iTEST = rand();      rg.set_new_seed(iTEST);
    rand_num=GenerateRandomNumber();
    rnd_val2=(numVar * rand_num) + 1;
    if((rnd_val2==a)||(rnd_val2==b))
        {iTEST = rand();rg.set_new_seed(iTEST);GetDiffCrossOverSiteTwo(a,b);}
    else
        return rnd_val2;}

void PrintParameters(double Rel[sizeComp], double Cost[sizeComp], double HRate[sizeComp])
{printf("\n");for (hr=0;hr<numVar;hr++)
    {printf("%f\t", Rel[hr]);}
    printf("\n");
    for (hr=0;hr<numVar;hr++)
        {printf("%f\t", Cost[hr]);}
    printf("\n");
    for (hr=0;hr<numVar;hr++)
        {printf("%f\t", HRate[hr]);}}

void PrintParametersToFile(double Rel[sizeComp], double Cost[sizeComp], double
HRate[sizeComp],double Loss_Fn,double REL)
{double RelValueTEST,RelAnalytical=1.0;
fprintf(file6,"%f, %f, %f, %f, %f, %f, %f, %f, %f, %f,
%f,\n",LOSS_OPT,Loss_Fn,REL,Rel[0],Rel[1],Rel[2],Rel[3],Rel[4],Rel[5],Rel[6],Rel[7],Rel[8]);
fprintf(file6,"%f, %f, %f, %f, %f, %f, %f, %f, %f, %f, %f,
%f,\n",LOSS_OPT,Loss_Fn,REL,Cost[0],Cost[1],Cost[2],Cost[3],Cost[4],Cost[5],Cost[6],Cost[7],Co
st[8]);}

void PrintDataBeforeCrossOver(void)
{int i;printf("\n Before Cross Over\n");
    for(i=0;i<numVar;i++)
        printf("%f\t",CHROMOSOME_1_R_COPY[i][0]);printf("\n");
    for(i=0;i<numVar;i++)
        printf("%f\t",CHROMOSOME_2_R_COPY[i][0]);printf("\n");
    for(i=0;i<numVar;i++)
        printf("%f\t",CHROMOSOME_1_C_COPY[i][0]);printf("\n");
    for(i=0;i<numVar;i++)

```

---

---

```

printf("%f\t",CHROMOSOME_2_C_COPY[i][0]);printf("\n");}

void PrintDataAfterCrossOver(void)
{int i;printf("\n After Cross Over\n");
  for(i=0;i<numVar;i++)
    printf("%f\t",CHROMOSOME_1_R_COPY[i][0]);printf("\n");
  for(i=0;i<numVar;i++)
    printf("%f\t",CHROMOSOME_2_R_COPY[i][0]);printf("\n");
  for(i=0;i<numVar;i++)
    printf("%f\t",CHROMOSOME_1_C_COPY[i][0]);printf("\n");
  for(i=0;i<numVar;i++)
    printf("%f\t",CHROMOSOME_2_C_COPY[i][0]);printf("\n");}

void PrintDataBeforeMutation(void)
{
  int i;printf("\n Before Mutation\n");
  for(i=0;i<numVar;i++)
    printf("%f\t",CHROMOSOME_1_R_COPY[i][0]);
  printf("\n");for(i=0;i<numVar;i++)
    printf("%f\t",CHROMOSOME_1_C_COPY[i][0]);printf("\n");}

void PrintDataAfterMutation(void)
{int i;printf("\n After Mutation\n");for(i=0;i<numVar;i++)
  printf("%f\t",CHROMOSOME_1_R_COPY[i][0]);printf("\n");
  for(i=0;i<numVar;i++)
    printf("%f\t",CHROMOSOME_1_C_COPY[i][0]);printf("\n");}

void PrintOptimalResult(void)
{int i; printf("\n*****\n");
  printf("\n Optimal Selection of Components is Highlighted Below:\n");
  for(i=0;i<numVar;i++)
    printf("%f\t",CHROMOSOME_1_R[i]);printf("\n");
  for(i=0;i<numVar;i++)
    printf("%f\t",CHROMOSOME_1_C[i]);printf("\n");
  printf("\n Optimal System Reliability : %f\n Optimal Losses :%f\n Optimal (Q) : %f\n Optimal (K)
: %f\n", REL_OPT,LOSS_OPT,COF_CompCost_OPT,COF_OPT);
  printf("\n*****\n");}

int CompareChromosomes(void)
{int cmpGENE=0,cmpCTR=0;
  double cmpCHROM1, cmpCHROM2;
  for(cmpCTR=0;cmpCTR<numVar;cmpCTR++)

    {cmpCHROM1=CHROMOSOME_1_R[cmpCTR];cmpCHROM2=CHROMOSOME_2_R[c
mpCTR]; if(cmpCHROM1==cmpCHROM2){cmpGENE++;continue;}
      else break;}
  if(cmpGENE>=numVar)

```

---



---

```
{printf("\n\t IDENTICAL CHROMOSOMES\n");return 999;}  
else return 0;}
```



---

## References & Bibliography

---

- Ahuja, S. (1997), Performance based reliability optimisation for computer networks, 'Southeastcon 97 Engineering new New Century', *Proceedings. IEEE*, pages 121-125.
- Alander, J. (1994), *An indexed bibliography of Genetic Algorithms*, Art of CAD Ltd., Espoo, Finland, 1957-1993.
- Albert, A. (1958), *A Measure of the Effort Required to Increase Reliability*, Technical Report 43, Applied Mathematics and Statistic Laboratory; Stanford University, Stanford
- Allan, R., Billinton, R. and De Oliveira, M. (1976), An efficient algorithm for deducing the minimal cuts and reliability indices of a general network configuration, *IEEE Transactions on Reliability*, Vol. 25(4), pages 226-232.
- Amstadter, B.L. (1971), *Reliability Mathematics*, New York: McGraw-Hill.
- Andrews, J.D. and Moss, T.R. (2002), *Reliability and Risk Assessment*, Professional Engineering Publishing.
- Ashlock, D. (2006), *Evolutionary Computation for Modeling and Optimisation*, Springer.
- Back, T., Fogel, D. and Michalewicz, Z. (1997), *The handbook of evolutionary computation*, Publishing and Oxford University Press, Philadelphia, PA.
- Back, T., Fogel, D. and Michalewicz, Z..(2000a), *Evolutionary Computation 1 - Basic Algorithms and Operators*, Institute of Physics Publishing.
- Back, T., Fogel, D. and Michalewicz, Z. (2000b), *Evolutionary Computation 2 - Basic Algorithms and Operators*, (Eds.), Institute of Physics Publishing.
- Back, T. Hammel, U and Schewefel, H (1997), Evolutionary Computation: Comments on the history and current state. *IEEE Transaction on Evolutionary Computation*, Vol.1(1), pages 3– 17.
- Bai, D., Yun W. and Chung, S. (1991), Redundancy optimization of k-out-of-n:G system with common-cause failures, *IEEE Transactions on Reliability*, Vol.40, pages 56-59.
- Barlow, R. and Proschan, F. (1965), *Mathematical Theory of Reliability*, Wiley.

- Barlow, R. and Proschan, F. (1975), *Statistical Theory of Reliability and Life Testing*, Rinehart and Winston, Inc.
- Baxter, L. & Harche, F. (1992), On the optimal assembly of series-parallel systems, *Operations Research Letters*, Vol.11, pages 153-157.
- Billinton, R. and Allan, R. (1992), *Reliability Evaluation of Engineering Systems – concepts and technique*, Plenum Press, New York & London.
- Blischke, W. and Murthy, D. (2000), *Reliability Modelling, Prediction and Optimisation*, Wiley.
- Boland, P., Proschan, F. and Tong, Y. (1989), Optimal arrangement of components via pairwise rearrangements, *Naval Research Logistics*, Vol.36, pages 807-815.
- Booker, D., Fogel, B., Whitley, D. and Angeline, P. J. (1997), *Recombination In Back et al.* (1997), Chapter E3.3, IOP Publishing and Oxford University Press, Philadelphia, PA. pp. C3.3:1-C3.3:27.
- Bremermann, H.. (1958), *The evolution of intelligence. The nervous system as a model of its environment*, Technical Report No. 1, Department of Mathematics, University of Washington, Seattle, WA.
- Brown, R., Gupta, S., Christie, R., Venkata, S. and Fletcher, R. (1997), Automated Primary Distribution System Design: Reliability and Cost Optimisation, *IEEE Transactions on Power Delivery*, Vol. 12 (2), pages1017-1022.
- Burke, E. and Smith, A. (1999), A memetic algorithm to schedule planned maintenance, *ACM journal of Experimental Algorithm*, Vol. 41.
- Burke, E. and Kendall, G. (2005), *Search Methodologies - Introductory Tutorials in Optimisation and Decision Support Techniques*, Springer.
- Burke, E. and Newall, J. P. (1999), A multi-stage evolutionary algorithm for the timetable problem, *IEEE Transactions on Evolutionary Computation*, Vol. 3, pages 63 - 74.
- Burke, E., Newall, J. P. and Weare, R. F. (1996), A memetic algorithm for university exam timetabling, In Burke, E. and Ross, P. *The Practice and Theory of Automated Timetabling 1, Lecture Notes in Computer Science*, Vol. 1153, Springer, Berlin, pages 241-250.
- Burke, E., Cowling, P., De Causmaecker, P. and Vanden Berghe, G. (2001), A memetic approach to the nurse rostering problem, *Appl. Intell.*, Vol. 15, pages 199-214.
- Cancela, H. and Khadiri, M. (1995), A recursive variance-reduction algorithm for estimating communication-network reliability, *IEEE Transactions on Reliability*, Vol. 44, pages 595-602.
- Cantoni, M., Marseguerra, M. and Zio, E. (1999), Genetic algorithms and monte carlo simulation for optimal plant design, *Reliability Engineering and System Safety*, Vol. 68(2000), pages 29-38.
- Catuneanu, V. and Mihalache, A. (1989), *Reliability fundamentals*, Elsevier.

- Chen M.S. (1992), On the computational complexity of reliability redundancy allocation in a series system, *Operations Research Letters*, Vol.11, pages 309-15.
- Chi, D. and Kuo, W. (1990), Optimal design for software reliability and development cost, *IEEE Journal on Selected Areas in Communications*, Vol.8, pages 276-281.
- Cho, N., Papazoglou, I. and Bari, R. (1986), A methodology for allocating reliability and risk, *Technical Report: NUREG/CR -4048*, Brookhaven National Lab., Upton, NY (USA).
- Coello, C.C. (2002), Theoretical and numerical constraint-handling techniques used with evolutionary algorithms: a survey of the state of the art. *Computational methods in applied mechanics and engineering* 191, 1245-1287.
- Coit, D.W. & Smith, A.E. (1996), Reliability optimization of series-parallel system using genetic algorithm, *IEEE Transactions on Reliability*, Vol.45, pages 254 - 260.
- Coit, D.W. & Smith, A.E. (1996), Solving the redundancy allocation problem using a combined neural network/genetic algorithm approach, *Computers and Operations Research*, Vol.23, pages 515 - 526.
- Coit, D.W. and Smith, A.E. (1996), Penalty guided genetic search for reliability design optimization, *Computers and Industrial Engineering*, Vol.30, pages 895-904.
- Coit, D.W. & Smith, A.E. (1998), Redundancy allocation to maximise a lower percentile of the system time to failure distribution, *IEEE Transactions on Reliability*, Vol 47(1), pages 79-87.
- Coit, D.W. and Smith, A.E. (2002), Genetic algorithm to estimate a lower bound for system time to failure with uncertain component Weibull parameters , *Computers and Industrial Engineering*, Vol.41(4), pages 423-440.
- Dale, C.J. & Winterbottom, A. (1986), Optimal allocation of effort to improve system reliability, *IEEE Transactions on Reliability*, Vol.35, pages 188-191.
- Davis, L. (1996), *Handbook of Genetic Algorithms*, International Thomson Computer Press, London.
- De Jong and Spears. (1994) On the virtues of parameterized uniform crossover, *Proc. 4th Int. Conf. on Genetic Algorithms*.
- Dimitris, N. and Chorafas, P. (1960), *Statistical Processes and Reliability Engineering*, D. Van Nostrand Company Princeton, NJ.
- Dengiz, B., Altıparmak, F., and Smith, A.E. (1997), Efficient optimization of all-terminal reliable networks using an evolutionary approach, *IEEE Transactions on Reliability*, Vol.46, pages 18-26.
- Dhillon, B. (2005), *Reliability, Quality and Safety for Engineers*, CRC Press.
- Dhingra, A.K. (1992), Optimal apportionment of reliability and redundancy in series systems under multiple objectives, *IEEE Transactions on Reliability*, Vol.41, pages 576-582.
- Doty, L. (1985), *Reliability for the technologies*, Industrial Press Inc.

- 
- Ebeling C.E. (1997), *An Introduction To Reliability and Maintainability Engineering*, McGraw-Hill Inc.
- Elegbede, A.O.C., Chu, C., Adjallah, K.H. & Yalaoui, F. (2003), Reliability allocation through cost minimisation, *IEEE Transactions on Reliability*, Vol.52, pages 106-111.
- El-Newehi, E., Proschan, F. and Sethuraman, J. (1986), Optimal allocation of components in parallel-series and series-parallel systems, *Journal of Applied Probability*, Vol.23, pages 770-777.
- Eshelman, L. (2000), *Genetic Algorithms*, In Back et al. (2000a), pages 64-80.
- Falkenauer, E. (1998), *Genetic Algorithms and Grouping Problems*, Wiley.
- Fishman, G.S. (1986), A comparison of four Monte Carlo methods for estimating the probability of s-t connectedness, *IEEE Transaction on Reliability*, Vol. 35(2), pages 145-55.
- Fishman, G.S. (1986) A Monte Carlo sampling plan for estimating network reliability, *Annals of Operations Research*, Vol. 34, pages 581-584.
- Fogarty, T. (1994), *Evolutionary Computing*, Springer-Verlag, Berlin.
- Fogel, D. and Ghozeil, A. (1996), Using fitness distributions to design more efficient evolutionary computations, In: Fogel,D. *Proceedings of the Third IEEE Conference on Evolutionary Computation*, IEEE Press, Nagoya, Japan, pages 11-19.
- Fogel, D., (1998), *Evolutionary Computation, the Fossil Record*. IEEE Press, Piscataway, New Jersey.
- Fogel, D.B. (2000), *Evolutionary Computation: Towards a New Philosophy of Machine Intelligence (2<sup>nd</sup> ed.)*, IEEE Press, Piscataway, NJ.
- Fogel, L.J., Owens, A.J. and Walsh, M.J. (1996), *Artificial Intelligence through Simulated Evolution*, Wiley, New York.
- Fotuhi-Firuzabad, M., Munian, T.S. and Vinayagam, B. (2004), A novel approach to determine minimal tie-sets of complex network, *IEEE Transactions on Reliability*, Vol. 53 (1), pages 61-70.
- Fraser, A.S. (1957), Simulation of genetic systems by automatic digital computers. II: Effects of linkage on rates under selection, *Australian Journal of Biological Science*, Vol. 10, pages 492-499.
- Gen, M. and Cheng, R. (1997), *Genetic Algorithms and Engineering Design*, Wiley.
- Gen, M. and Cheng, R. (2000), *Genetic Algorithms and Engineering Optimisation*, Wiley.
- Gen, M., Ida, K. and Lee, J.U. (1990), A computational algorithm for solving 0-1 goal programming with GUB structures and its application for optimization problems in system reliability, *Electronics and Communications in Japan*, Vol.73, pages 88-96.
- Gen, M. and Kim, J. (1999), GA-based reliability design: State-of-the-art survey, *Comp. Ind. Eng.*, Vol.37(1), pages 151-155.
- Glover, F. and Laguna, M. (1997), *Tabu Search*, Kluwer Academic Publishers, Boston, MA.

- Goldberg, D. and Sastry, K. (2001), A practical schema theorem for genetic algorithm design and tuning, *Proceedings of the Genetic and Evolutionary Computation Conference*, pages. 328-335.
- Goldberg, D., Korb, B. and Deb, K. (1989), Messy Genetic Algorithms: motivation, analysis and first results, *Complex Systems*, Vol.3, pages 493-530.
- Goldberg, D. (1989) *Genetic Algorithms in Search, Optimisation and Machine Learning*, Addison-Wesley.
- Goldberg, D. (2002) *Design of Innovation: Lessons From and For Competent Genetic Algorithms*, Kluwer, Boston, MA.
- Gopal, K., Aggarwal, K.K. and Gupta, J.S. (1980), A new method for solving reliability optimization problem, *IEEE Transactions on Reliability*, Vol.29, pages 36-38.
- Gopal, K., Aggarwal, K.K. and Gupta, J.S. (1978), An improved algorithm for reliability optimization, *IEEE Transactions on Reliability*, Vol.-27, pages 325-328.
- Guikema, S. and Pate-Cornell, E. (2002), Component choice for managing risk in engineered systems with generalized risk/cost functions, *Reliability Engineering and System Safety*, Vol.78, pages 227-238.
- Hecht, H. (2004), *Systems Reliability and Failure Prevention*, Boston, MA: Artech House.
- Henley E.J. and Kumamoto, H. (1981), *Reliability Engineering and Risk Assessment*, Prentice-Hall.
- Holland, J. (1975), *Adaptation in Natural and Artificial Systems*, In Ann Arbor, University of Michigan Press.
- Hwang, C.L., Tillman, F.A. and Kuo, W. (1979), Reliability optimization by generalized Lagrangian-function and reduced-gradient methods, *IEEE Transaction on Reliability*, Vol.28, pages 316-319.
- Hussain, A. and Murthy, D. (2003), Warranty and optimal reliability improvement through product development, *Journal of Mathematical and Computer Modelling*, Vol.38, pages 1211-1217.
- Hsieh, C. and Hsieh, Y. (2003), Reliability and cost optimisation in distributed computing systems, *Computers and Operations Research*, Vol.30(8), pages 1103-1119.
- Hsieh, C. (2003), Optimal task allocations and hardware redundancy policies in distributed computing system, *European Journal of Operational Research*, Vol. 147(2), pages 430-447.
- Ibaraki, T. (1997), *Combination with other optimisation methods*, In Bristol and New York, Institute of Physics Publishing and Oxford University Press.
- Ida, K., Gen, M. and Yokota, T. (1994), System reliability optimization with several failure modes by genetic algorithm'. In Gen, M. and Kobayashi, T., editors, *Proceedings of the 16<sup>th</sup> International Conference on Computers and Industrial Engineering*, Ashikaga, Japan, pages, 349-352.

- Jianping, L. (1995), A bound heuristic algorithm for solving reliability redundancy optimisation, *Microelectronics and Reliability*, Vol.36, pages 335-339
- Jianping, L. (1996), A bound dynamic programming for solving reliability redundancy optimisation, *Microelectronics and Reliability*, Vol.36(10), pages 1515-1520.
- Jianping, L. (1996), A bound heuristic algorithm for solving reliability redundancy optimization, *Microelectronics and Reliability*, Vol.36, pages 335-339.
- Jianping, L. (1996), A new partial bound enumeration technique for solving reliability redundancy optimisation, *Microelectronics and Reliability*, Vol.37, pages 237-242.
- Joines, J. and Kay, M. (2002), Utilizing hybrid genetic algorithms. In Sarker *et al.*, pages. 199-228.
- Kamat, S and Riley, M (1975), Determination of reliability using event based Monte Carlo simulation, *IEEE Transactions on Reliability*, Vol. 24 (1), pages 73–75.
- Kaufmann, A., Grouchko, D. and Cruon, R. (1977), *Mathematical models for the study of the reliability systems*, Academic press.
- Kim, J.H.& B.J.Yum. (1993), A heuristic method for solving redundancy optimization problems in complex systems, *IEEE Transactions on Reliability*, Vol.24, pages 572-578.
- Kohda, T. & Inoue, K. (1982), A reliability optimization method for complex systems with the criterion of local optimality, *IEEE Transactions on Reliability*, Vol.31, pages 109-111.
- Kleyner, A. and Sandborn, P. (2005), A warranty forecasting model based on piecewise statistical distributions and stochastic simulation, *Reliability Engineering and System Safety*, Vol.88, pages 207-214.
- Krasnogor, N. and Smith, A.J. (2005) *A tutorial for competent memetic algorithms: model, taxonomy and design issues*, *IEEE Transactions on Evolutionary Computation*.
- Krasnogor, N., Hart, W. and Smith, J. (2004), Recent Advances in Memetic Algorithms, *Studies in Fuzziness and Soft Computing*, Vol.166, Springer.
- Kumamoto, H., Tanaka, K. and Inoue, K. (1977), Efficient evaluation of system reliability by Monte Carlo method, *IEEE Transactions on Reliability*, Vol. 26(5), pages 311–315.
- Kumral, M. (2005), Reliability-based optimisation of a mine production system using genetic algorithms, *Journal of Loss Prevention*, Vol.18, pages 186-189.
- Kuo, W., Prasad, V. R., Tillman, F.A. and Hwang, C. (2001), *Optimal Reliability Design - fundamental and applications*, Cambridge University Press.
- Kuo, W. & Prasad, V.R. (2000), An annotated overview of system reliability optimization, *IEEE Transactions on Reliability*, Vol.49, pages176-191.
- Kuo, W., Hwang, C.L. and Tillman, F.A. (1978), A note on heuristic methods in optimal system reliability, *IEEE Transactions on Reliability*, Vol.27, pages 320-324.



- Kuo, W., Lin, H., Xu, Z. and Zhang, W. (1987), Reliability optimization with the Lagrange multiplier and branch-and-bound technique, *IEEE Transactions on Reliability*, Vol.36, pages 624-630.
- Kuo, W., and Wan, R. (2007), Recent advances in optimal reliability allocation, *IEEE Transactions on Systems*, Vol.37(2), pages 143-156.
- Lee, C., Yun, Y. and Gen, M., (2002a), Reliability optimisation design using hybrid genetic algorithm with a neural network technique, *IEICE Transaction on Fund. Electr. Comm. Comp. Sc.*, Vol. E84-A(2), pages 627-637.
- Lee, C., Gen, M., and Kuo, W. (2001), Reliability optimisation design using hybrid NN-GA with fuzzy logic controller, *IEICE Transaction on Fund. Electr. Comm. Comp. Sc.*, Vol. E85-A(2), pages 432-447.
- Lee, C., Gen, M., and Tsujimura, Y. (2002b), Reliability optimisation design for complex systems by hybrid GA with fuzzy logic controller and local search, *IEICE Transaction on Fund. Electr. Comm. Comp. Sc.*, Vol. E85-A(4), pages 880-891.
- Levitin, G. (2007), *Genetic Algorithm in Reliability Engineering* [Homepage of Dr. Gregory Levitin]. <http://iew3.technion.ac.il/~levitin/GA+Rel.html>.
- Levitin, G and Lisnianski, A., Optimal separation of elements in vulnerable multi-state systems, *Reliability Engineering and System Safety*, Vol. 73(1), pages 55-66.
- Lewin, B. (2000), *Genes VII*. Oxford University Press, New York.
- Li, D. & Haimes, Y.Y. (1992), A decomposition method for optimization of large-system reliability, *IEEE Transactions on Reliability*, Vol.41, pages 183- 188.
- Lin, F. & Kuo, W. (1996), Reliability Importance and Invariant Optimal Allocation, *Technical Report, Texas A&M University, College Station, TX*.
- Lloyd, D.K and Lipow, M. (1962), *Reliability: Management, Methods and Mathematics*, Prentice-Hall, Englewood Cliffs, NJ.
- Louis, S. and McDonnell, J. (2004), Learning with case injected genetic algorithms, *IEEE Transactions on Evolutionary Computation*, Vol. 8, pages. 316-328.
- Luenberger, D.G. (1962), Quasi-convex programming, *SIAM Journal of Applied Mathematics*, Prentice-hall, Englewood Cliffs, NJ.
- Hikita, M., Nakagawa, Y., Nakashima, K. and Narihisa, H. (1992), Reliability optimization of systems by a surrogate-constraints algorithm, *IEEE Transactions on Reliability*, Vol. 41, pages 473-480.
- Majety, S., Venkatasubramanian, S. and Smith, A.E. (1996), Optimal Reliability Allocation in series-parallel systems from components' discrete cost-reliability data sets: A nested simulated annealing approach, *Proceedings of the Fifth International Industrial Engineering Research Conference*, pp. 435-440.



- 
- Majety, S. & Rajagopal, J. (1997), Dynamic Penalty Function for Evolutionary Algorithms with an Application to Reliability Allocation, *Technical Report, Department of Industrial Engineering, University of Pittsburgh, Pittsburgh, PA.*
- Malon, D.M. (1990), When is greedy module assembly optimal?, *Naval Research Logistics Quarterly*, Vol.37, pages 847-854.
- Mattfeld, D. (1996), *Evolutionary Search and the Job Shop*, Physica-Verlag.
- Metropolis, N., Rosenbluth, A.W., Rosenbluth, M.N., Teller, A. and Teller, E. (1953), Equation of state calculations by fast computing machines', *Journal of Chemical Physics*, Vol.21, pages 1087-1092.
- Meziane, R., Massim, Y., Zeblah, A., Ghoraf, A. and Rahili, R. (2005), Reliability optimisation using ant colony algorithm under performance and cost constraints, *Electric Power System Research*, Vol. 76, pages 1-8.
- Michalewicz, Z. (1996), *Genetic Algorithms + Data Structures = Evolution Programs*, 3rd Edition, Springer.
- Michalewicz, Z. (2000a), *Decoders*. In: Back *et al.* (2000b), pages. 49-55.
- Michalewicz, Z. (2000b), *Repair Algorithms*. In: Back *et al.* (2000b), pages 56-61.
- Misra, K.B. (1972), A simple approach for constrained redundancy optimisation problems, *IEEE Transactions on Reliability*, Vol.21, pages, 30-34.
- Misra K.B. (1986), On optimal reliability design: a review, *System Science*, Vol.12, pages 5-30.
- Misra, K.B. (1991), An algorithm to solve integer programming problems: an efficient tool for reliability design', *Microelectronics and Reliability*, Vol.31, pages 285-294.
- Misra, K.B.(1992) '*Reliability analysis and predictions*', Elsevier.
- Misra, K. & Misra, V. (1994), A procedure for solving general integer programming problems, *Microelectronics and Reliability*, Vol.34, pages 157-163.
- Misra, K.B. & Sharma, U. (1991), Multicriteria optimization for combined reliability and redundancy allocation in systems employing mixed redundancies, *Microelectronics and Reliability*, Vol.31, pages 323-335.
- Misra, K.B. & Sharma, U. (1991), An efficient approach for multiple criteria redundancy optimization problems, *Microelectronics and Reliability*, Vol.31, pages 303-321.
- Mitchell, M. (1996), *An introduction to Genetic Algorithms*, MIT Press, Cambridge, MA.
- Mohan, C. & Shanker, K. (1988), Reliability optimization of complex systems using random search technique, *Microelectronics and Reliability*, Vol.28, pages 513-518.
- Moscato, P. and Cotta, C. (2003), *A gentle introduction to memetic algorithms*, In Norwell, MA, Kluwer.
- Moscato, P. (1989), On evolution, search, optimisation, genetic algorithms and martial arts, *Technical Report C3P 826*, California Institute of Technology, Pasadena, CA.

- 
- Moscato, P. (1999), Part 4: Memetic Algorithms, In: *New ideas in optimisation*, Corne, D, Dorigo, M, and Glover, F, eds., McGraw-Hill, New York, pages. 217-294.
- Moscato. (2001) Memetic algorithms, In: section 3.6.4, *Handbook of applied optimisation*, Pardalos, P. and Resende, M., eds., Oxford University Press.
- Nakagawa Y. & Nakashima, K. (1977), A heuristic method for determining optimal reliability allocation, *IEEE Transactions on Reliability*, Vol.26, pages 156-161.
- Nakagawa, Y. and Miyazaki, S. (1981), Surrogate constraints algorithm for reliability optimization problem with two constraints, *IEEE Transactions on Reliability*, Vol.30, pages 175-180.
- Nakagawa, Y. & Miyazaki, S. (1981), An experimental comparison of the heuristic methods for solving reliability optimization problems, *IEEE Transactions on Reliability*, Vol.30, pages 181-184.
- Naudts, B., Suys, D., Verschoren, A. (1997), Epistasis as a basic concept in formal landscape analysis. In: Back, T. (Ed.), *Proceedings of the Seventh International Conference on Genetic Algorithms*, Morgan Kaufmann.
- O'Connor, P. (2002), *Practical Reliability Engineering*, Wiley.
- Pham, H. (2007), list of articles in the field of cost-reliability analysis and warranty, <http://www.rci.rutgers.edu/~hopham/recentpublications.html>
- Pham, H. (2003), *Handbook of Reliability Engineering*, London: Springer.
- Painton, L. & Campbell, J. (1995), Genetic algorithms in optimization of system reliability, *IEEE Transactions on Reliability*, Vol.44, pages 172-178.
- Park, K.S. (1987), Fuzzy apportionment of system reliability, *IEEE Transactions on Reliability*, Vol.36, pages 129-132.
- Porto, V. (2000), Evolutionary Programming, In: Back *et al.*, pages 89-102.
- Prasad, V.. & Raghavachari, M. (1998), Optimal allocation of interchangeable components in a series-parallel system, *IEEE Transactions on Reliability*, Vol.47, pages 255-260.
- Prasad, V.. & Kuo, W. (2000), Reliability optimization of coherent systems, *IEEE Transactions on Reliability*, Vol.49, pages 176-187.
- Prasad, V., Nair, K. and Aneja, Y. (1991), Optimal assignment of components to parallel-series and series-parallel systems, *Operations Research*, Vol.39, pages 407-414.
- Prasad, V., Nair, K. and Aneja, Y. (1991), A heuristic approach to optimal assignment of components to a parallel-series network, *IEEE Transactions on Reliability*, Vol.40, pages 555-558.
- Radcliffe, N. and Surry, P. (1994), Formal Memetic Algorithms, In: Fogarty, pages. 1-16.
- Ravi, V. Murty, V. and Reddy, P. (1997), Nonequilibrium simulated-annealing algorithm applied to reliability optimization of complex systems, *IEEE Transactions on Reliability*, Vol.46, pages 233-239.
- Ravi, V., Reddy, P. and Zimmermann, H. (2000), Fuzzy global optimisation of complex system reliability, *IEEE Transaction on Fuzzy Systems*, Vol.8, pages 241-248.

- Rudolph, G. (2000a), *Evolution Strategies*, In: Back *et al.*, pages 81-88.
- Sakawa, M. (1981), Optimal reliability-design of a series-parallel system by a large-scale multi-objective optimization method, *IEEE Transactions on Reliability*, Vol.30, pages 173-174.
- Sanker, V. and Prasad, V (1993), Comment on: Enumeration of all minimal cutsets for a node pair in graph, *IEEE Transaction on Reliability*, Vol.42(1), pages 44-45.
- Sarker, R., Mohammadian, M. and Yao, X. (2002) *Evolutionary Optimisation*. Kluwer Academic Publishers.
- Schaffer, D. (1989) *Proceedings of the Third International Conference on Genetic Algorithms*, San Mateo, CA, Morgan Kaufmann.
- Schonberger, J. (2005) *Operational Freight Carrier Planning- Basic Concepts, Optimisation Models and Advanced Memetic Algorithms*, Springer.
- Schwefel, H.P. (1995), *Evolution and Optimum Seeking*, Wiley, New York.
- Sharma, J. and Venkateswaran, K. (1971), A direct method for maximising the system reliability, *IEEE Transactions on Reliability*, Vol.20, pages 256-259.
- Sharma, U., Misra, K. and Bhattacharjee, A. (1991), Application of an efficient search technique for optimal design of a computer communication network, *Microelectronics and Reliability*, Vol.31:2-3, pages 337-341.
- Shi, D.H. (1987), A new heuristic algorithm for constrained redundancy-optimization in complex system, *IEEE Transactions on Reliability*, Vol.36, pages 621-623.
- Smith, A. (2006), *Journal articles on reliability* (Homepage of Dr A. Smith), [http://www.eng.auburn.edu/~aesmith/index\\_files/page0004.htm](http://www.eng.auburn.edu/~aesmith/index_files/page0004.htm).
- Dengiz, B., Altıparmak, F. and Smith, A. (1997) *Local Search Genetic Algorithm for optimal design of reliability networks*, *IEEE Transactions on Evolutionary Computation*, Vol.1, pages 179-183.
- Smith, D. (2005), *Reliability Maintainability and Risk*, Elsevier.
- Smith, C.O. (1976), *Introduction to Reliability in Design*, New York, McGraw-Hill.
- Spears, W. (1997), *Recombination parameters*, In Back *et al.* (1997), Chapter E1.3, pages E1.3:1-E1.3:13.
- Syswerda, G. (1989), *Uniform Crossover in genetic algorithms*, In Schaffer, pages 2-9.
- Tillman F.A., Hwang, C. and Kuo, W. (1980), *Optimization of system reliability*, Marcel Dekker, New York.
- Tillman F.A., Hwang, C. and Kuo, W. (1977), Optimization techniques for system reliability with redundancy – a review, *IEEE Transactions on Reliability* Vol.26, pages 148 - 155.
- Tillman, F.A. (1969), Optimization by integer programming of constrained reliability problems with several modes of failure, *IEEE Transactions on Reliability*, Vol.18, pages 47-53.

- Tillman F.A., Hwang, C. and Kuo, W. (1977), Determining component reliability and redundancy for optimum system reliability', *IEEE Transactions on Reliability*, Vol.26, pages 162-165.
- Todinov M.T. (2004), Reliability analysis and setting reliability requirements based on the cost of failure, *International Journal of Reliability, Quality and Safety Engineering*, Vol.11, pages 1-27.
- Todinov M.T. (2005), *Reliability and Risk Models*, Wiley.
- Todinov, M.T. (2006), Reliability analysis based on the losses from failures, *Risk Analysis*, Vol. 26(2), pages. 311-335.
- Todinov, M.T. (2006a), *Risk based reliability analysis and generic principles for risk reductions*, Elsevier.
- Todinov M.T. (2006b), Reliability analysis of the complex systems based on the losses from failure, *International Journal of Reliability, Quality and Safety Engineering*, Vol.13(2), pages 1-22.
- Tzafestas, S.G. (1980), Optimisation of system reliability: a survey of problems and techniques, *International Journal of Systems and Science*, Vol.11, pages 455-486.
- Wattanapongsakorn, N. & Levitan, S.P. (2004), Reliability optimisation models for embedded systems with multiple applications, *IEEE Transactions on Reliability*, Vol.53 pages, 406-410.
- Winter, G., Periaux, J., Galan, M. and Cuesta, P. (1995) *Genetic Algorithms in Engineering and Computer Science*, Wiley.
- Xu, Z., Way Kuo, and H.Lin. 'Optimization limits in improving system reliability', *IEEE Transactions on Reliability*, Vol.39 (1990) 51-60.
- Yalaoui, A., Chatelet, E. and Chu, C. (2005), A new dynamic programming method for reliability and redundancy allocation in a parallel-series system, *IEEE Transaction on Reliability*, Vol. 54(2), pages 254-261.
- Yalaoui, A., Chatelet, E. and Chu, C. (2005) Reliability allocation problem in a series-parallel system, *Reliability Engineering and System Safety*, Vol.90, pages 55-61.
- Yang, J., Hwang, M., Sung, T and Jin, Y. (1999), Application of genetic algorithm for reliability allocation in nuclear power plants, *Reliability Engineering and System Safety*, Vol. 65 , pages 229-238.
- Yao, X. (2002), *Evolutionary Computation*. In Sarker *et al.*, pages 27-53.
- Yeh, M.S., Lin, J.S. and Yeh, W.C. (1994), A new Monte Carlo method for estimating network reliability, In: Gen, M. and Kobayashi, T. *Proceedings of the 16<sup>th</sup> International Conference on Computers and Industrial Engineering*, Ashikaga, Japan.
- Yokota, T., Gen, M. and Ida, K. (1995) System reliability of optimization problems with several failure modes by genetic algorithm, *Japanese Journal of Fuzzy Theory and Systems*, Vol.7, pages 117-135.

- Zhang, W., Miller, C. and Kuo, W. (1991), Application and analysis for consecutive k-out-of-n: G structure, *Reliability Engineering and System Safety*, Vol.33, pages 189-197.
- Zio, E. (2000), System design optimisation by genetic algorithm, *IEEE Proceedings, Annual Reliability and Maintainability Symposium*, pages 222-227.
- Zuo, M. and Kuo, W. (1990), Design and performance analysis of consecutive k-out-of-n structure, *Naval Research Logistics Quarterly*, Vol.37, pages 203-230.