ABSTRACT

Title of Dissertation: OPTIMIZATION OF TEST/DIAGNOSIS/REWORK LOCATION(S) AND CHARACTERISTICS IN ELECTRONIC SYSTEMS ASSEMBLY

Zhen Shi, Doctor of Philosophy, 2004

Dissertation directed by: Associate Professor Peter A. Sandborn
Department of Mechanical Engineering

For electronic systems it is not uncommon for 60% or more of the recurring cost to be associated with testing. Performing tradeoffs associated with where in a process to test and what level of test, diagnosis and rework to perform are key to optimizing the cost and yield of an electronic system's assembly. In this dissertation, a methodology that uses a real-coded genetic algorithm has been developed to minimize the yielded cost of electronic products by optimizing the locations of test, diagnosis and rework operations and their characteristics.

This dissertation presents a test, diagnosis, and rework analysis model for use in electronic systems assembly. The approach includes a model of functional test operations characterized by fault coverage, false positives, and defects introduced in test; in addition, rework and diagnosis operations (diagnostic test) have variable success rates and their own defect introduction mechanisms. The model accommodates multiple rework attempts on a product instance. For use in practical assembly processes, the

model has been extended by defining a general form of the relationship between test cost and fault coverage.

The model is applied within a framework for optimizing the location(s) and characteristics (fault coverage/test cost and rework attempts) of Test/Diagnosis/Rework (TDR) operations in a general assembly process. A new search algorithm called Waiting Sequence Search (WSS) is applied to traverse a general process flow to perform the cumulative calculation of a yielded cost objective function. Real-Coded Genetic Algorithms (RCGAs) are used to perform a multi-variable optimization that minimizes yielded cost. Several simple cases are analyzed for validation and general complex process flows are used to demonstrate the applicability of the algorithm. A real multichip module (MCM) manufacturing and assembly process is used to demonstrate that the optimization methodology developed in this dissertation can find test and rework solutions that have lower yielded cost than solutions calculated by manually choosing the test strategies and characteristics. The optimization methodology with Monte Carlo methods included for the process flow under uncertain inputs is also addressed in this dissertation.

It is anticipated that this research will improve the ability of manufacturing engineers to place TDR operations in a process flow. The ability to optimize the TDR operations can also be used as a feedback to a Design for Test (DFT) analysis of the electronic systems showing which portion of the system should be redesigned to accommodate testing for a higher level of fault coverage, and where there is less need for test.

OPTIMIZATION OF TEST/DIAGNOSIS/REWORK LOCATION(S) AND CHARACTERISTICS IN ELECTRONIC SYSTEMS ASSEMBLY

By

Zhen Shi

Dissertation submitted to the Faculty of the Graduate School of the
University of Maryland, College Park in partial fulfillment
of the requirements for the degree of
Doctor of Philosophy
2004

Advisory Committee:

Associate Professor Peter A. Sandborn, Chair/Advisor
Professor Gnanalingam Anandalingam
Professor Shapour Azarm
Associate Professor Jeffry W. Herrmann
Associate Professor F. Patrick McCluskey

TABLE OF CONTENTS

Section **Page**

List of Figures **vi**

List of Tables **xiii**

Nomenclature **xvi**

Chapter 1 Introduction……………………………………………………………..1

1.1 Background……………………………………………………………3

1.1.1 Process-Flow Modeling……………………………………..7

1.1.2 Test/Diagnosis/Rework Operations………………………..10

1.1.3 Characteristics of Process Steps……………………………15

1.1.4 Other Relevant Topics………………………………………18

1.2 Dissertation Scope and Problem Statement……………….…………..22

1.3 Overview of the Solution Strategy……………………………………..24

1.4 Organization of Dissertation ……………………………………………..27

Chapter 2 The Test/Diagnosis/Rework Model ………………..……………….. 29

2.1 Development of Test/Diagnosis/Rework Model………………………..30

2.2 Trichy *et al*. Model Formulation………………………………………34

2.3 Variable Test and Rework Cost Model………….……………………..37

2.4 Application of the TDR Model when Multiple Faults are Present……..42

2.5 Relationship of the TDR Model to Loop Number………..……………44

2.6 An Alternative Model of TDR Operation Cost ……………………......46

2.7 Single TDR Process Flow Calculation……………..……………….....48

2.8 Calculation of the Yielded Cost……………………………………51

2.9 Summary………………………………………………………………56

Chapter 3 Search Algorithm for Use in the Process Flow……………………58

3.1 Graphical Representation of a Process Flow…..…………………………59

3.2 Definitions of Basic Types of Vertices (Process Steps)……………60

3.3 Representations of Graphs — Adjacency Matrix…………………61

3.4 Waiting Sequential Search (WSS)…………………………………63

3.5 The Multi-Variable Optimization Function………………………69

3.6 Data Structures and Pseudo-Code for WSS Implementation…..……72

3.7 Summary………………………………………………………………74

Chapter 4 TDR Optimization with Real-coded Genetic Algorithms (RCGAs)…………………………………………………………………………75

4.1 Introduction to Real-Coded Genetic Algorithms………………………78

4.2 The Procedure of RCGAs Applied to the Process Flow………………..79

4.2.1 Chromosomal Representation……………………………..……80

4.2.2 Initial Population……………………………………………81

4.2.3 Selection of Mating Pairs…………………………………...81

4.2.4 Crossover……………………………………………………82

4.2.5 Mutation………………………………………………….....83

4.2.6 Elitist Strategy……………………………………………...84

4.3 Verification of the Optimization Model………………………………...85

4.3.1 Validation from the Simple Cases…………………………...86

4.3.1.1 One-Test Case (Test Case 1)……………………….....87

4.3.1.2 One-TDR Case (Test Case 2)………………………92

4.3.2 Optimization in a General Process Flow……………………97

4.3.2.1 Optimization in a Sequential Process Flow (Test Case 3) ……………………………………………………97

4.3.2.2 Optimization in a Branched Process Flow (Test Case 4)………………………………………………104

4.4 Optimization of Rework Attempts…..…………………………….....111

4.5 Summary………………………………………………………113

Chapter 5 Optimization of TDR in Multichip Module (MCM) Assembly…...114

5.1 Introduction to Multichip Module…………...………………………..114

5.2 Testing an MCM…………………………………………………....116

5.3 The Test/Diagnosis/Rework (TDR) of an MCM ……………………117

5.4 Optimization of Fault Coverage in the MCM Assembly Process Flow………………………………………………………124

5.5 Optimization of Rework Attempts for MCM Assembly Process Flow…………………………………………...……………152

5.6 Summary………………………………………………………148

Chapter 6 Optimization of TDR Under Uncertain Inputs…………….………149

6.1 Uncertain Inputs ……………….……………..…………………149

6.2 Monte Carlo Methods…………………………………………….150

6.3 Application of Monte Carlo Method in Optimization of TDR Operations for Uncertain Inputs……………………………………………………151

6.3.1 Estimation of the Number of Samples……………………156

6.3.2 Monte Carlo Implementation…………………………158

6.4 Real Case Analysis………………………………………………164

6.4.1 MCM Assembly Process Flow with Single TDR and Uncertain Inputs……………………………………………164

6.4.2 MCM Assembly Process Flow with Multiple TDRs and Uncertain Inputs……………………………………………176

6.5 Summary……………………………………………………………180

Chapter 7 Conclusions and Contributions……………………………………181

References………………………………………………………………189

List of Figures

Figure 1.1 Comparison of manufacturing and testing cost of per transistor……...…2

Figure 1.2 The lifecycle of an electronic system……………………………………4

Figure 1.3 Location(s) of test/diagnosis/rework operation…………………………..7

Figure 1.4 Genetic process flow and step…………………………………………...8

Figure 1.5 A simple non-test process flow…………………………………………..9

Figure 1.6 A simple test/diagnosis/rework process……………..…………………..…..10

Figure 1.7 MCM fractional yield as a function of number of chips in the MCM……………………..………………………………………………..…14

Figure 1.8 Outgoing yield versus fault coverage…………………………………..….16

Figure 1.9 The percentage of module fractional yield improvements as a functional of the number of repairs for MCMs containing 9, 25, 49 and 81 chips, where each chip has a yield of 0.95……………………………….………….18

Figure 1.10 The comparison of yielded cost of process flow with four different test cases ………………………………………………………………………….23

Figure 1.11 The framework of optimization of TDRs in process flow……………..27

Figure 2.1 Example test/diagnosis/rework models currently in use for technical cost modeling……………………..…………………………………………30

Figure 2.2 Organization of the test/diagnosis/rework model………………….…….31

Figure 2.3 Typical curve of untested faults versus the number of tests……….…….39

Figure 2.4 Example relationship between the test cost and fault coverage……..…..40

Figure 2.5 Example relationship between the rest cost and fault coverage with 10% threshold……………………………………………………………41
Figure 2.6 Parallel test operations……………………………………………..43
Figure 2.7 Single pass rework numerical example……………………………..….49
Figure 2.8 A single non-test process flow……………………………………..52
Figure 2.9 A single-TDR process flow………………………………………….52
Figure 2.10 An example process flow with two assembly steps and a TDR package……………………………………………………………54
Figure 2.11 An example process flow with two incoming branches……………….55
Figure 3.1 A portion of a complex process flow………………………………...59
Figure 3.2 Graphical representation of process flow given in Figure 3.1…………..60
Figure 3.3 Waiting Sequential Search (WSS) algorithm flowchart………………..65
Figure 3.4 Applying a Waiting Sequential Search (WSS) to the process flow in Figure 3.1………………………………………………………………67
Figure 3.5 Data exchange in the implementation of WSS to the process flow……..69
Figure 4.1 An example of binary and floating-point representations……………….78
Figure 4.2 GA optimization process ....…………………………………..…79
Figure 4.3 Roulette-wheel selection……………………………………………..82
Figure 4.4 One-point crossover for floating-point representation…………………..83
Figure 4.5 The framework of process flow cost optimization modeling system…..84
Figure 4.6 Simple case with just one test step……………………………………86
Figure 4.7 Yielded cost of process flow in Figure 4.6 versus fault coverage of test……………………………………………………………………88

Figure 4.8 Optimization of yielded cost in the single-test case (Test Case 1) at the high-yield level……………………………………………………91
Figure 4.9 Process flow with one test/diagnosis/rework operation (Test Case 2)….92
Figure 4.10 Optimization histories of fault coverage in one-TDR case (Test Case 2)…………………………………………………………………93
Figure 4.11 Optimization histories of rework yield in one-TDR case (Test Case 2)…………………………………………………………………94
Figure 4.12 Sequential process flow……………………………..…..…………97
Figure 4.13 Computed optimum yielded cost of process flow in Figure 4.12……..98
Figure 4.14 Computed cost of process flow in Figure 4.12………………………...99
Figure 4.15 Computed yield of process flow in Figure 4.12………………………100
Figure 4.16 Computed optimum fault coverage for TDR Operations when test cost coefficient (p_t) is 0.02 in the process flow shown in Figure 4.12……101
Figure 4.17 Computed optimum fault coverage for TDR Operations when test cost coefficient (p_t) is 0.04 in the process flow shown in Figure 4.12……101
Figure 4.18 Computed optimum fault coverage for TDR Operations when test cost coefficient (p_t) is 0.08 in the process flow shown in Figure 4.12……102
Figure 4.19 Computed optimum fault coverage for TDR Operations when test cost coefficient (p_t) is 0.16 in the process flow shown in Figure 4.12……102
Figure 4.20 Computed optimum fault coverage for TDR Operations when test cost coefficient (p_t) is 0.32 in the process flow shown in Figure 4.12……103
Figure 4.21 Computed optimum fault coverage for TDR Operations when test cost coefficient (p_t) is 0.64 in the process flow shown in Figure 4.12……103

Figure 4.22 A complex process flow with all possible TDR operations…………104

Figure 4.23 (a) Computed optimum fault coverage for TDR operations in the process flow shown in Figure 4.22 (C_{ft}= \$1 and C_{fr}= \$1)……………………..107

Figure 4.23 (b) Applicable rework operations for TDR operations in the process flow shown in Figure 4.22 (C_{ft}= \$1 and C_{fr}= \$1)…………………….........107

Figure 4.24 (a) Computed optimum fault coverage for TDR operations in the process flow shown in Figure 4.22 (C_{ft}= \$1 and C_{fr}= \$100)…………………..108

Figure 4.24 (b) Applicable rework operations for TDR operations in the process flow shown in Figure 4.22 (C_{ft}= \$1 and C_{fr}= \$100)………………………..108

Figure 4.25 Computed optimum fault coverage for TDR operations in the process flow shown in Figure 4.22 (C_{ft}= \$1 and no rework)…………………109

Figure 4.26 (a) Computed optimum fault coverage for TDR operations in the process flow shown in Figure 4.22 (C_{ft}= \$50 and C_{fr}= \$100)…………………110

Figure 4.26 (b) Applicable rework operations for TDR operations in the process flow shown in Figure 4.22 (C_{ft}= \$50 and C_{fr}= \$100)………………………110

Figure 4.27 Optimization of yielded cost for various levels of fixed cost of test and rework operations……………………………………………………111

Figure 5.1 A typical multichip module…………………………………………115

Figure 5.2 A simple MCM manufacturing and test model………………………...118

Figure 5.3 Functional relationship between fault coverage and test cost applied to the MCM example……………………………………………………………127

Figure 5.4 Cost of MCM after TDR versus incoming yield of MCM for the tree test scenarios plus the optimum solution…………………………………..130

Figure 5.5 Output yield of MCM after TDR versus incoming yield of MCM for the tree test scenarios plus the optimum solution....131
Figure 5.6 Yielded Cost of MCM after TDR versus incoming yield of MCM for the tree test scenarios plus the optimum solution....132
Figure 5.7 GA-based optimum fault coverage versus incoming yield of MCM.....133
Figure 5.8 Yielded cost of the MCM (after TDR) versus fault coverage (die cost = $2 and die yield =80%)....135
Figure 5.9 Yielded cost of the MCM (after TDR) versus fault coverage (die cost = $7 and die yield =99%)....137
Figure 5.10 Yielded cost of the MCM (after TDR) versus fault coverage (die cost = $8 and die yield =99.5%)....139
Figure 5.11 Comparison of minimum yielded cost generated by optimization of various rework attempts and fixed rework attempts....142
Figure 5.12 Comparison of fault coverage at the minimum yielded cost at various rework attempts and fixed rework attempts....144
Figure 5.13 Rework attempts at the minimum yielded cost for the single TDR MCM process flow....145
Figure 6.1 A simple process flow example with uncertain inputs....150
Figure 6.2 Application of Monte Carlo methods in the optimization of TDR operations for uncertain inputs....152
Figure 6.3 Monte Carlo sampling inside the GAs optimization of TDR operations....154
Figure 6.4 Confidence level for Monte Carlo size estimation....157

Figure 6.5 Estimation of sample size for a given confidence level………………..158

Figure 6.6 MCM assembly process flow with uncertain incoming yield of single die……………………………………………………………………….159

Figure 6.7 Monte Carlo sampling size estimation for the normal distribution of die yield………………………………………………………………………160

Figure 6.8 Monte Carlo sampling size estimation for the triangular distribution of die yield………………………………………………………………………162

Figure 6.9 Comparison of the minimum yielded cost for single value and a normal distribution of incoming die yield……………………………………..165

Figure 6.10 Distributions of die yield……………………………………………..…168

Figure 6.11 Comparison of the optimum fault coverage for single value input and a normal distribution of incoming die yield…………………………..….169

Figure 6.12 Mean of yielded cost varies with the various fault coverage………..170

Figure 6.13 Minimum yielded cost when the single die yield is represented by a triangular distribution…………………………………………………172

Figure 6.14 Comparison of the optimum fault coverage for single value input and a triangular distribution of incoming die yield……………………….173

Figure 6.15 Mean of yielded costs varies with the various fault coverage under the triangular distribution input of die yield………………………………174

Figure 6.16 Comparison of the mean of yielded cost for different input distributions……………………………………………………….……175

Figure 6.17 MCM assembly process flow with multiple TDRS…………….…….176

Figure 6.18 Minimum yielded cost of MCM assembly process with multi-TDR and a normal distribution of input die yield……………………………………177

Figure 6.19 Optimum fault coverages of various TDRs in the process flow in Figure 6.17...……..179

List of Tables

Table 2.1 Nomenclature used in Figure 2.2 and throughout the discussion in this section……………………………………………………………………33

Table 2.2 Example values of factors in (2.17) and (2.18)…………………………..40

Table 3.1 Adjacency matrix of complex process flow given in Figure 3.1………...62

Table 3.2 Search index of the vertices in the process of WSS to the process flow…68

Table 4.1 Values of input parameters………………………………………………86

Table 4.2 Results of RCGAs applied in the one-TDR case at high-yield inputs (Test Case 2)……………………………………………………………………………..95

Table 4.3 Results of RCGAs applied in the one-TDR case at low-yield inputs (Test Case 2)……………………………………………………………………………..96

Table 4.4 Characteristics of the process steps in Figure 4.10………………………97

Table 4.5 Characteristics of the process steps in Figure 4.22……………………105

Table 4.6 Optimized TDR locations in the process flow shown in Figure 4.22 for various fixed test cost and rework cost……………………………..112

Table 5.1 Chip yield versus MCM yield, cost and yielded cost …………………..120

Table 5.2 Values of parameters used in the MCM manufacturing and test flow in Figure 5.1……………………………………………………………………………121

Table 5.3 Fault coverage with its corresponding testing cost………….…………125

Table 5.4 Comparison of GA optimum yielded cost with results from test strategies (die cost = $2)…………………………………………………………………128

Table 5.5 Comparison of GA optimum yielded cost with results from test strategies (die cost = $3)..128
Table 5.6 Comparison of GA optimum yielded cost with results from test strategies (die cost = $4)..128
Table 5.7 Comparison of GA optimum yielded cost with results from test strategies (die cost = $5)..128
Table 5.8 Comparison of GA optimum yielded cost with results from test strategies (die cost = $6)..129
Table 5.9 Comparison of GA optimum yielded cost with results from test strategies (die cost = $7)..129
Table 5.10 Comparison of GA optimum yielded cost with results from test strategies (die cost = $8)..129
Table 5.11 Comparison of GA optimum yielded cost with results from test strategies (die cost = $9)..129
Table 5.12 Optimization of fault coverage in the MCM assembly process flow with the cost of incoming die is $2...136
Table 5.13 Optimization of fault coverage in the MCM assembly process flow with the cost of incoming die is $7...138
Table 5.14 Optimization of fault coverage in the MCM assembly process flow with the cost of incoming die is $8...140
Table 5.15 Cost reduction efficiency of minimum yielded cost with various input of die yield...141

Table 5.16 Optimization of the MCM assembly process flow with the rework attempts fixed at one (die cost is $3)......................................146

Table 5.17 Optimization of the MCM assembly process flow with the rework attempts optimized (die cost is $3)......................................147

Table 6.1 Comparison between RCGAs for certain and uncertain inputs….....…156

Table 6.2 Monte Carlo sampling for a normal distribution of die yield…………161

Table 6.3 Monte Carlo sampling for a triangular distribution of die yield………..163

Table 6.4 Optimization of fault coverage in single TDR MCM assembly process flow in Figure 6.6 with the cost of incoming die yield is 90%.............166

Table 6.5 Mean of yielded costs with the corresponding fault coverage calculated using Trichy *et al.* model with Monte Carlo method included..........171

Table 6.6 Optimization of fault coverage in multi-TDR MCM assembly process flow in Figure 6.17...178

Nomenclature

a	Exponential constant for a given logic structure.
b_r	Coefficient of rework characteristic.
b_t	Coefficient of test characteristic.
B	Average number of non-equivalent single stuck faults per gate in the circuit.
c	Confidence interval.
$C_{assembly}$	Cost of assembly process.
C_{diag}	Cost of diagnosis operation.
CDR	Cost of diagnosis and rework operations.
C_{fr}	Fixed cost of rework operation.
C_{ft}	Fixed cost of test operation.
C_{in}	Incoming cost of assemblies to a process flow.
C_{rew}	Cost of rework operation.
$C_{R.Prog}$	Cost of initial reflow controller programming.
CTDR	Cost of test/diagnosis/rework package.
C_{test}	Cost of test operation.
C_{out}	Outgoing cost of assemblies following a process flow.
C_Y	Yielded cost.
E	Set of edges in the graph.
f_c	Fault coverage of a test operation.
f_d	Fraction of assemblies determined to be reworkable.

f_r	Fraction of assemblies actually reworked.
f_p	False positives fraction, or the probability of testing a good assembly as bad.
FPY	Fraction of PWA's which pass the first circuit test.
f_T	Fault coverage threshold.
F_0	Number of undetected faults at the beginning of Phase I of testing.
G	Number of gates in the combinational circuit.
id	indegree of the step node in graph.
k	Number of faults detected per test in Phase II of testing.
l	Number of process steps in the process flow.
m	Number of feature parameters to be optimized.
M_{rew}	Total cost of rework equipment.
n	Number of rework iterations allowed.
N	Number of years in the payback period.
N_d	Number of assemblies entering the test/diagnosis/rework operation.
N_{d1}	$N_d - N_{gout}$.
N_{gout}	Number of no fault found assemblies.
N_{in}	Number of assemblies entering the test/diagnosis/rework operation.
N_r	Number of assemblies to be reworked.
N_{rout}	Number of assemblies actually reworked
N_{s1}	Number of assemblies scrapped by diagnosis operation.
N_{s2}	Number of assemblies scrapped during rework.
N_{out}	Number of assemblies exiting the test/diagnosis/rework operation.

od	Outdegree of a step node in the graph.
PFB	Probability of a part being faulty or the proportion of boards that contain one or more faults.
PFB_b	Probability of a failing part having both faults A and B.
p_t	Cost coefficient of test operation.
p_r	Cost coefficient of rework operation.
r	Sample size for the Monte Carlo analysis.
r_a	Number of rework attempts at a specific rework operation before scrapping.
r_t	Fault ratio.
s	Number of trials of Monte Carlo sampling.
S	Number of shifts worked.
S_{total}	Fraction of assemblies that are scrapped by the process flow.
S_b	Chromosomes made up of binary numbers.
S_r	Chromosomes made up of real numbers.
t	Number of tests.
T_0	Number of tests required to achieve a maximum possible test coverage.
T_I	Number of tests at the end of Phase I of testing.
t_{rew}	Average time to rework a faulty PWA.
u	Number of samples to be used for the Monte Carlo analysis.
v	Step node in the graph representation of process flow.
V	Set of step nodes.

W_{rew}	Wage paid to rework technician(s).
x_c	Proportion of reworked printed wiring assemblies.
$Y_{assembly}$	Yield of assembly process.
$Y_{beforetest}$	Yield of processes that occur entering the test.
$Y_{aftertest}$	Yield of processes that occur exiting the test.
Y_{rew}	Yield of a rework operation.
Y_{in}	Incoming yield of assemblies to a process flow.
Y_{out}	Outgoing yield of assemblies following a process flow.
Z	Random variable with unit normal distribution.
μ	Mean of sampled values.
σ	Standard deviation.
ω	Desired confidence level.

Chapter 1

Introduction

At a fundamental level, system design is a tradeoff analysis activity. This tradeoff includes factors such as size and performance, but often the most important factor in the tradeoff is cost. The various recurring costs that affect the manufacture of a system are the fabrication/assembly cost, test/inspection[1] cost, rework cost, and waste disposition cost.

For many types of systems, functional test is an important driver that significantly affects the total cost of manufacturing. In electronic systems, for example, it is not uncommon that greater than 60% of a product's recurring cost can be attributed to testing [1], for integrated circuits, recurring testing costs are approaching 50% of the total product cost [2]. The International Technology Roadmap for Semiconductors has predicted that it will cost nearly as much to test a transistor than it costs to manufacture the transistor by 2015, Figure 1.1.

[1] In this dissertation we are concerned with recurring functional (pass/fail) and diagnostic testing only, not environmental testing (i.e., qualification testing).

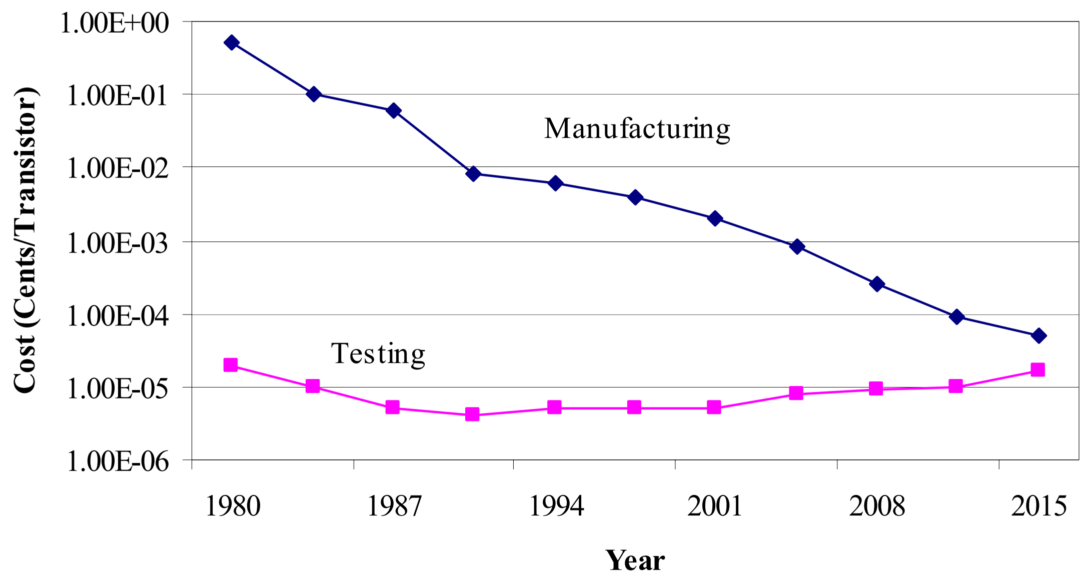


Figure 1.1: Comparison of manufacturing and testing cost of per transistor, [3].

When the products that result from a manufacturing process are imperfect, four costs are potentially involved, first the cost of determining whether a given instance of the product is good or bad (recurring functional testing), second the cost of determining what defect caused product to be faulty and where the defect is located (diagnosis), third fixing the defect (rework), and fourth eliminating the causes of the defect (continuous improvement). Depending on the maturity of the product, its placement in the market, and the profit associated with selling it, all, some or none of the four activities listed above may be involved. Understanding the test/diagnosis/rework costs may determine the extent to which the system designer can control and optimize the manufacturing cost, and the extent to which it makes sense to do so.

The ultimate goal of any functional test strategy is the determination of: 1) When should a system be tested? At what point(s) in the manufacturing process? 2) How much

testing should be done, i.e., how thorough should the test be? A test that detects 10% of the defects in a product may cost a small fraction of a test that identifies 95% of the defects, so, if I have multiple tests in a process, what is the optimum fault coverage to buy for each one? and 3) How much time and money should be spent to make the product more testable? These goals would be easy to realize if we had unlimited time, resources, and money. We could stop after every step in the manufacturing process and perform a full function test, and add structures to our system such that every critical element could be accessed and tested. These measures are unfortunately far from practical and we are usually faced with determining how to obtain the best possible coverage from our tests for the least cost.

So, how does the test influence the cost, yield and how can the parameters that characterize testing be optimized to minimize the cost and maximize the yield of the product simultaneously? These tradeoffs are the key questions that will be discussed in this dissertation. Section 1.1 provides reviews on the basic process of testing an electronic system.

1.1 Background

The specific goal of testing is to minimize the cost of discarding good product and the cost of shipping bad product. Test is a step in the manufacturing process that ensures that the physical device has no manufacturing defects, which means that the product functions and exhibits the properties and capabilities it was designed for. Testing is an important activity in the lifecycle of an electronic product shown in Figure 1.2.

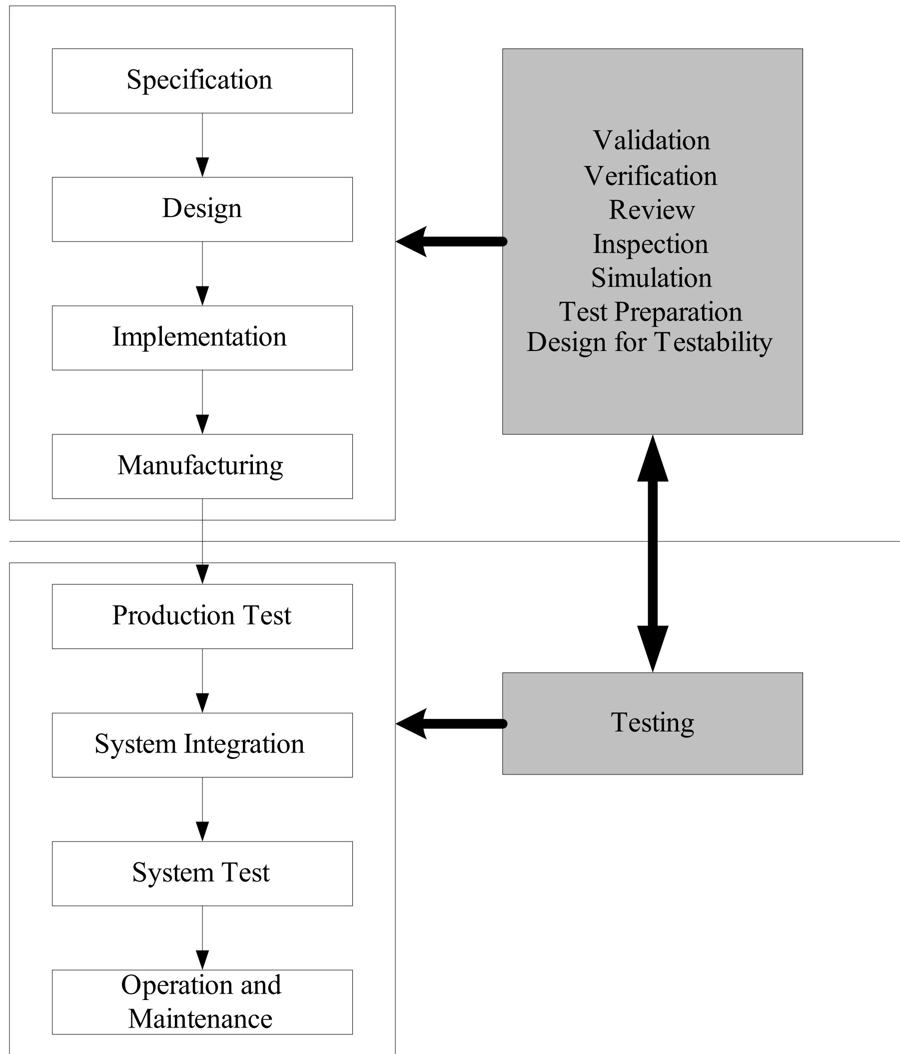


Figure 1.2: The lifecycle of an electronic system, [4].

Following the lifecycle flow, tests have been integrated in the manufacturing process of the electronic products. Production (manufacturing) test refers to the test for the individual products to check whether faults are introduced during manufacturing. Alternatively, system test is used to test a product in its operating environment to ensure that it works correctly when interconnected with other components. Tests are not only applied after the products have been produced but also considered in the design phase to check whether there are possible defects existing in the products or corresponding

manufacturing processes via the functional verifications and simulations. This type of test results in the correction or improvement of design.

Tests can be categorized as the following:

(1) In-Circuit tests, measuring the performance of individual components by electrically isolating them from the surrounding circuitry [5], which is best described as testing the functionality of each component in the product, with the inference that the overall circuit functionality can be verified by the fact that each component functions correctly and that all the components are wired together properly.

(2) Functional tests, verifying the overall board operation by applying stimuli to existing onboard connectors, measuring the response parameters and comparing these with design rules [6]. Functional testing concentrates on the question of the functionality requirements of the product: "Does the product do what it is supposed to do?" this usually entails testing the product at the critical ends of its specifications [7].

(3) Diagnostic tests, focusing on the fault location and identification, failure analysis design and/or process debugging and improvement.

(4) Parametric tests, measuring electrical properties of the system — delay, voltages, currents, etc., which puts emphasis on issues such as the frequency of operation, acceptable tolerances on power supply, temperature ranges, and power dissipation.

(5) Environmental tests, evaluating the validity of designs and reliability of materials, components and interconnections under various environmental stresses, e.g., shock, vibration and temperature.

Testing plays a very important role in increasing the yield and ensuring the quality of an electronic system. However, tests cost money, so one can't practically place a test step between every process activity. On the other hand, lack of tests means that the process steps may be wasting money processing modules that already contain defects that can't be repaired. Therefore, some balance of spending money on testing and risking spending money on processing modules that will be scrapped later is necessary. It is an imperative task to determine where in a manufacturing process to perform test and what are the optimum test characteristics. Figure 1.3 shows a possible placement of a test/diagnosis/rework operation in an example process for electronic systems assembly. Every arrow in the general manufacturing process in the top half of Figure 1.3 is a possible placement of a test/diagnosis/rework operation. The objective of the research in this dissertation is to develop a methodology that enables intelligent optimization of the location(s) of test/diagnosis/rework in general process flows.

Before the aspects of the test/diagnosis/rework operation are analyzed, a procedural model that can determine the process steps and represent the associated sequence necessary to fabricate a product is needed. The following subsections provide the background necessary to describe process flows, and test, diagnosis, and rework process steps.

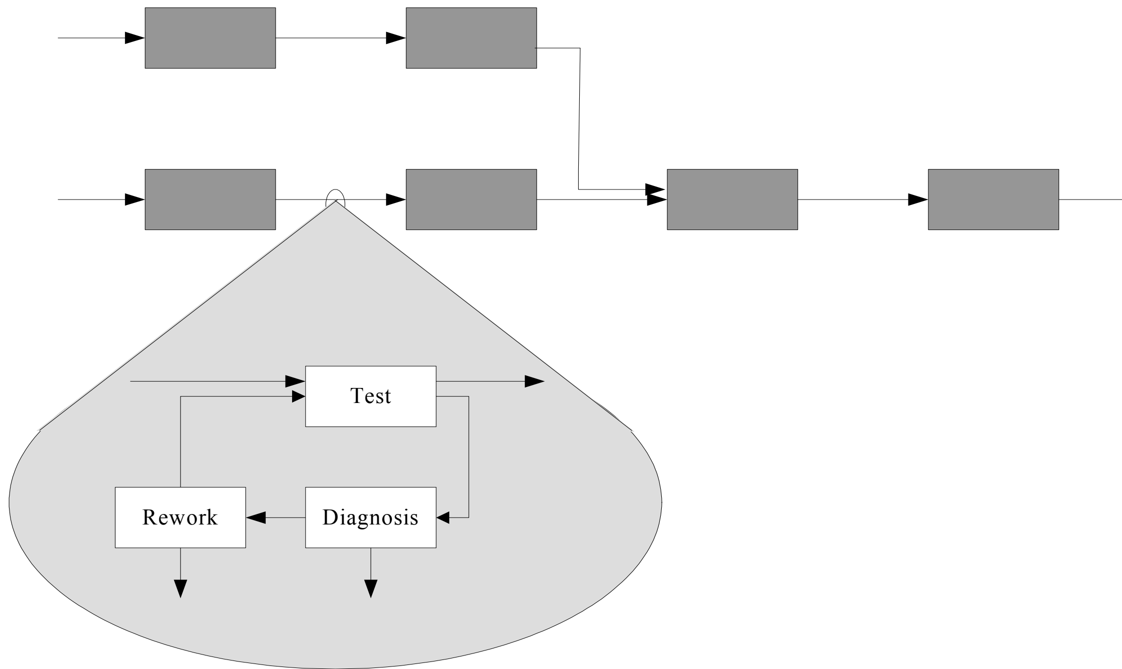


Figure 1.3: Location(s) of test/diagnosis/rework operation.

1.1.1 Process-Flow Modeling

Real life is a sequence of process step, things happen in a specific order. The steps and their order comprise a "process flow". Process-flow analysis attempts to emulate real life. The manufacturing cost and yield can be determined by modeling the fabrication process as a sequence of process steps. Process identification is the determination of the process steps and their associated sequence necessary to fabricate a product. For each process step, the cost and yield associated with the step is determined. Process identification is the key enabling activity for the evaluation of manufacturability, process compatibility, cost and yield.

Process-flow modeling is performed by providing a set of process step objects that may be combined to describe a process flow. Figure 1.4 shows a generic process flow and step. The form of the inputs to a process step should match the form of the outputs so that the process steps can be easily sequenced. However, the inputs may be modified by the process step. The fundamental quantities that are carried from step to step are the cost, quality, and time. Optionally, some processes may inventory materials (used and wasted) and/or energy requirements.

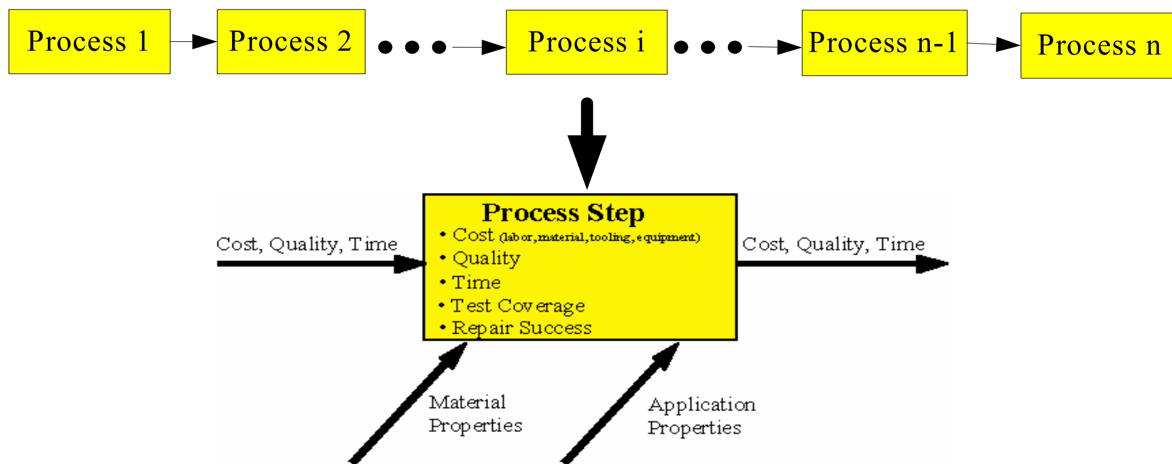


Figure 1.4: Generic process flow and step.

Process steps can be defined at a high level (cost, yield), or a detailed level (time, labor rates, material costs, equipment costs, tooling costs, etc.). Cost (recurring and non-recurring), quality (defect density or yield), and time are accumulated through the process flow. When executed, each step in the process flow uses its local data to modify the cumulative cost, quality, and time associated with the object being manufactured.

There are different kinds of process steps: Fabrication or assembly (generic), Test/inspection, Rework, Waste Disposition, and Insertion. Some steps (test and inspection) remove instances of the product from the process. In addition to the inputs/outputs described above, the test steps may be characterized by fault coverage, and false positives. The objects "scrapped" by a step may re-enter later after rework (characterized by a rework success rate).

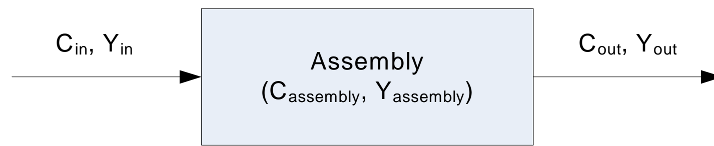


Figure 1.5: A simple non-test process flow.

To illustrate the cumulative calculations of the output yield and cost of products, consider the simple non-test process flow is shown in Figure 1.5. Every component coming to the assembly step has some cost and yield associated with it. The assembly process has its own cost ($C_{assembly}$) and yield ($Y_{assembly}$). In the simplist case the output cost and yield of the assembly process are given by (1.1) and (1.2) respectively:

$$C_{out} = C_{in} + C_{assembly} \tag{1.1}$$

$$Y_{out} = Y_{in} Y_{assembly} \tag{1.2}$$

$C_{assembly}$ and $Y_{assembly}$ could be computed quantities that depend on application-specific propertities.

The next sections define some of the terminology associated with a process flow that contains test, diagnosis and rework steps.

1.1.2 Test/Diagnosis/Rework Operations

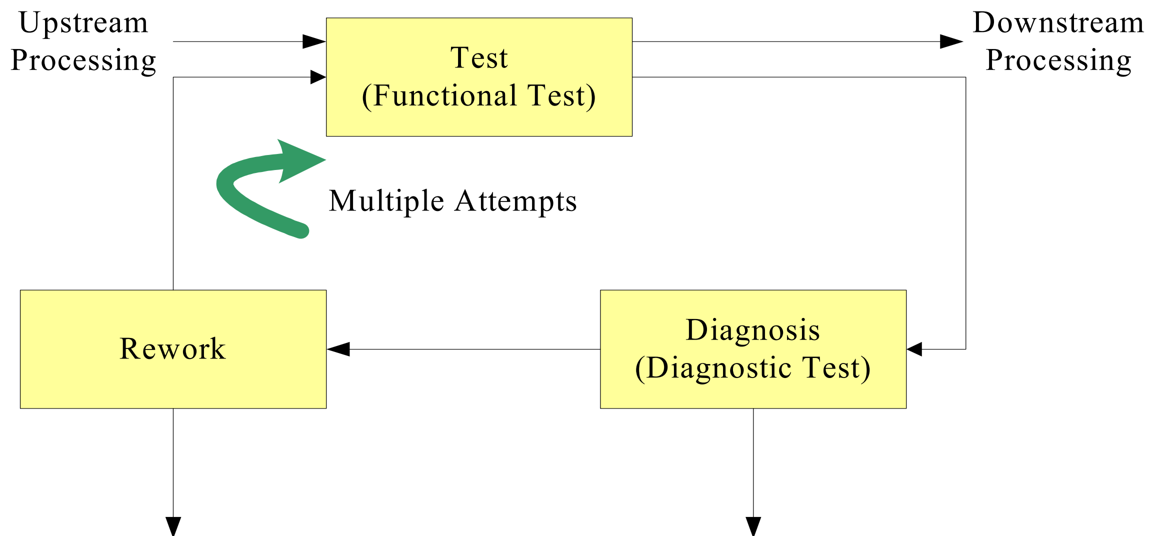


Figure 1.6: A simple test/diagnosis/rework process.

When a test or inspection activity is performed, the product that does not pass the test can be either scrapped (disposed of), salvaged (all or part of the product is recovered for reuse in the same or another product), recycled (broken down to its constituent materials), or the non-passing product can be reworked. The first activity that takes place after a product is failed by a test activity is to determine why it failed, this activity is called diagnosis. Once diagnosis is completed, a decision can be made as to whether a particular product should be reworked, i.e., repaired and sent back into the test, or scrapped. A simple view of test, diagnosis and rework is shown in Figure 1.6.

In the example test/diagnosis/rework process shown in Figure 1.6, all of the product coming from production is tested; a more detailed diagnostic test (diagnosis) is applied to all the product that is identified as defective during the test; and all reworkable the product is retested. Note, diagnosis and rework are not perfect (they introduce defects, make mis-diagnoses, and fail to correctly rework defective products), therefore, a product may go through test, diagnosis and rework multiple times (multiple “attempts”).

The goal of analyzing the diagnosis and rework process (coupled with test) is to determine which products should be reworked (as opposed to scrapped), and to determine the optimum number of times to attempt to rework a product before giving up and scrapping it. At a broader level, the challenge in a manufacturing process is to determine where in the process to test and when to diagnose and rework test rejects. In some cases it may be more economical to simply scrap product that does not pass tests then to pay to diagnose them. Chapter 2 will discuss a detailed test/diagnosis/rework model developed by [8, 9], which includes a model of test operations characterized by fault coverage, false positive, and defects introduced in test, in addition to rework and diagnosis operations that have varaiable success rates and their own defect introduction mechanisms. The model can also accmodate an arbitrary number of rework attempts on any given assembly and can be used to obtain a cost-effective fault coverage and rework investment during system tradeoff analyses.

Diagnosis

Diagnosis, also known as fault isolation, refers to determining the type of defect that caused a specific fault and the location of that defect within the faulty product. Before

any decisions are made regarding the disposition of product that the test step deems faulty, a diagnosis must be performed. The outcome of the diagnosis will be one of the following:

- No fault found, i.e., a false positive from the test activity — In the case of no fault found, the product is sent back for retesting without any rework. Note, even if no fault is found, the product still incurs the cost of the diagnosis and is subject to any defects that may be inserted into the product by the test and diagnosis processes.
- Product is faulty but the cause of the fault cannot be pinpointed — In this case a decision must be made to either expend additional resources to diagnose the problem or scrap the product.
- Defect type and location successfully identified — In this case a decision is made as to whether the defect is repairable or not, and whether it is worth repairing or not. If the defect is not worth repairing, then the product will be scrapped.

Several cost impacts are associated with diagnosis. First, the creation of, and correlation to fault dictionaries or trees is a non-trivial and very resource consuming activity. Existing fault dictionaries and trees are rarely directly applicable to a specific application and require considerable resources to become useful in the diagnosis process. Simply performing the diagnosis process itself consumes resources (labor, tooling, and capital). Second, the diagnosis impacts the throughput of the entire test/diagnosis/rework process.

Rework

Rework is the process of correcting defects in a product during the production process. Rework is differentiated from repair, which is the process of correcting defects in a product that has failed at some point in time after manufacturing was completed. In the case of repair, the defect could be due to undetected manufacturing defects or damage accumulated during field use. Rework generally plays a more important role when large costs have been invested into products prior to testing. While rework is common for board assembly, it is also performed during some types of integrated circuit fabrication.

Rework is one of the most unpredictable and variable parts of the electronic systems assembly process. In fact, no other single activity in the assembly process negatively affects profitability more than rework [10]. Unfortunately, most electronic assemblers treat rework as an afterthought, clinging to the notion that they can perfect their process to eliminate rework, which is often not the case.

The economic viability of the product lines is usually forced into one of two conditions: either we build modules so inexpensively that they are throwaways (given a high-yielding process with a large-commodity market) or we must be able to rework defective assemblies, which is effectively a way to improve the yield of system. Petek [11] did research on the impact of rework on multichip module (MCM) cost. For example, a 25-chip MCM with an individual chip yield of 90% has a rework potential of $(1-0.9^{25})$=0.928, that means approximately 92.8% of the modules have the potential to fail the test or require rework. Thus the ability to repair or rework MCMs is extremely

important, especially as the number of chips or the complexity of systems increases (see Figure 1.7).

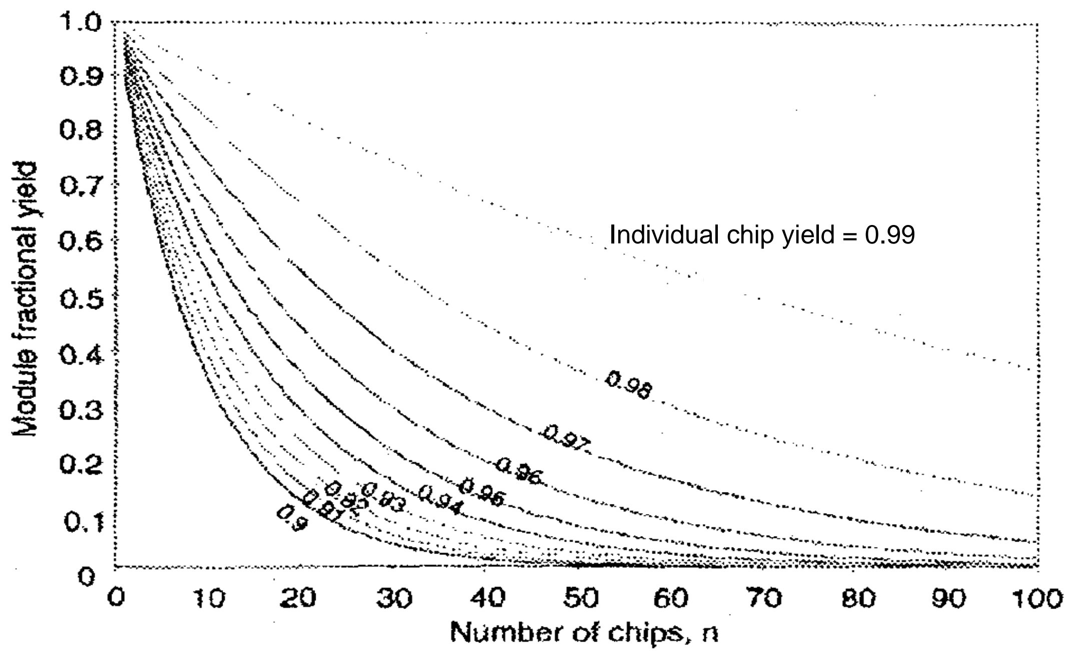


Figure 1.7: MCM fractional yield as a function of number of chips in the MCM, [11].[2]

So far we have reviewed the concepts of design, manufacturing, test, diagnosis and rework—a series of processes in the lifecycle of products. Of the manufacturing costs involved, the test, diagnosis and rework costs significantly affect the total cost of manufacturing of electronic systems. Consequently, it becomes very important to accurately model the characteristics of the process flow which determines the test/diagnosis/rework costs in order to control and reduce the manufacturing cost. The next section will focus on the parameters that drive the tradeoff analyses of test/diagnosis/rework operations in process flow.

[2] Figure 1.7 is a description of the Known Good Die (KGD) problem, i.e., MCMs have an extremely low first pass yield if KGD are not used.

1.1.3 Characteristics of Process Steps

There are several characteristic variables associated with the various types of process steps — fault coverage, rework yield, false positive, rework attempts and so on, which determine the cost and yield of products that pass through then, hereafter there will be called the *feature parameters* of process flow. As the main objective of the dissertation, these parameters are varied to obtain the optimum of a multi-variable function that represents the yielded cost of products that result from the process flow.

Fault Coverage

Fault Coverage (f_c) refers to the fraction of faults detected by a test activity.[3] Fault coverage is also referred to as Fault Cover, Test Coverage, or Test Efficiency [12].

$$f_c = P\,(\text{fault detection/fault existance}) = \frac{\#(\text{detected faults})}{\#(\text{possible faults})} \tag{1.3}$$

Fault coverage is a measure of the ability of a set of tests (a collection of test vectors) to detect a given class of faults that may occur in a device under test. The fault coverage attained with a test is dependent on the number of test vectors exercised, which determines the test time and thereby the test cost.

[3] This definition is sometimes referred as "raw coverage." Related metrics that could also be defined include:

$$\text{Testable Coverage} = \frac{\text{Number of detected faults}}{\text{Number of total faults} - \text{Number of untestable faults}} \tag{1.4}$$

$$\text{Fault Efficiency} = \frac{\text{Number of detected faults - Number of untestable faults}}{\text{Number of total faults}} \tag{1.5}$$

The yield of electronic products can be affected by the testing. (1.6) is the fundamental result from Williams and Brown [13] that relates the yield of product passed by a test to the fault coverage of the test. This relation forms the basis for much of test economics and the modeling of test process steps.

$$Y_{out} = Y_{in}^{1-f_c} \tag{1.6}$$

where Y_{in} is the yield of parts entering the test activity, Y_{out} is the yield of parts that have been passed by the test activity and f_c is the fault coverage associated with the test activity.

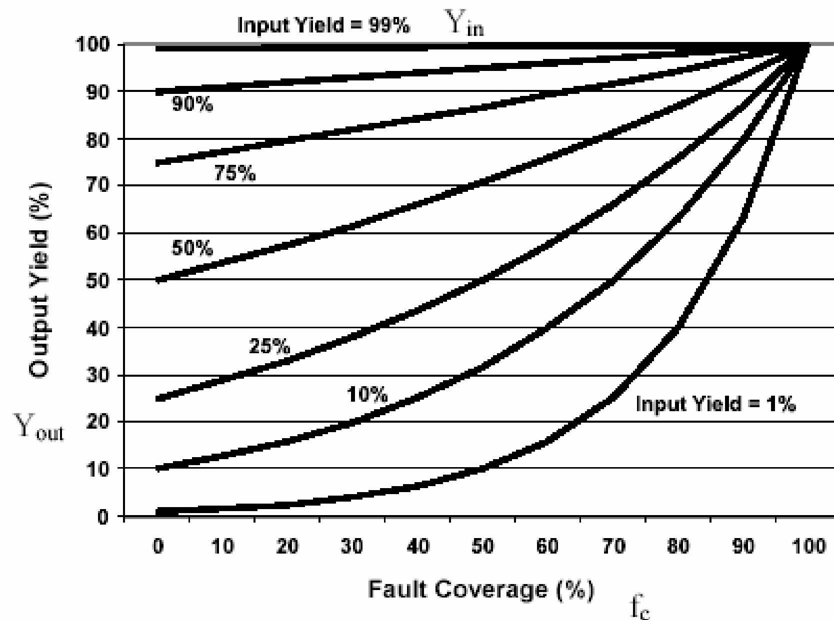


Figure 1.8: Outgoing yield versus fault coverage.

Figure 1.8 shows outgoing yield versus fault coverage for various values of incoming yield. In Figure 1.8 as fault coverage approaches 100%, outgoing yield is 100% independent of the incoming yield. This makes sense because at 100% fault coverage the

test step successfully scraps every defective part (irregardless of the fraction of parts are defective coming into the test) only letting good parts pass. When fault coverage drops to 0, the outgoing yield should equal the incoming yield (the test is not doing anything). When the incoming yield is 100%, every incoming part is good and therefore every outgoing part is also good regardless of fault coverage. As the incoming yield becomes small, the output yield is also small for all but fault coverages that approach 100%.

Rework Yield and Rework Attempts

Rework yield (Y_{rew}) is defined as the fraction of faults repaired by a rework operation [14]. The factor is used to remove the assumption that the rework would be performed correctly and completely for all defects on each visit to the rework step. This rarely happens in practice. Rework attempts (r_a) is the number of times an instance of a product is reworked before giving up and scrapping it. With the increasing of rework attempts, the yield of products would also rise, but the cost increases too. For a given module size, the first rework makes the most significant improvement, with each succeeding rework diminishing in importance, which is illustrated by Figure 1.9.

Yielded Cost

Yielded cost (C_Y) refers to the cost per good product instance passed by the test step, which is a ratio of the final cost and yield. If the C_{out} is the cost of a single part after some process and Y_{out} is the yield of the part, then the yielded cost is given by $C_Y = \frac{C_{out}}{Y_{out}}$ [15].

In this dissertation, the yielded cost of a process flow is the final objective

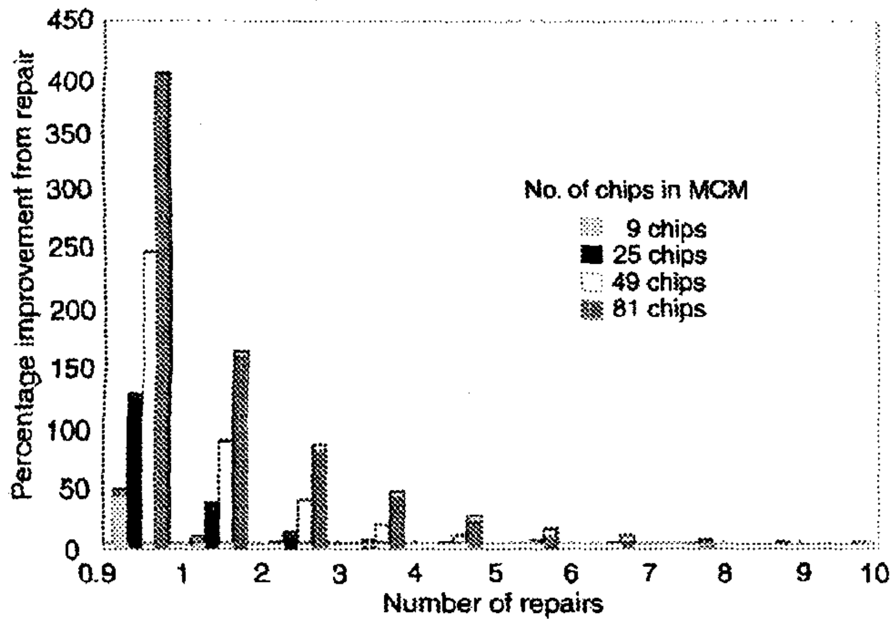


Figure 1.9: The percentage of module fractional yield improvements as a function of the number of repairs for MCMs containing 9, 25, 49 and 81chips, where each chip has a yield of 0.95, [11].

that needs to be minimized by choosing the optimal values of variables in the optimization function. As demonstrated in [8, 16], some of the feature parameters would definitely affect the cost and yield of the process flow and consequently affect the yielded cost. For example, Figure 1.8 shows that high fault coverage leads to high yield but the cost of obtaining the high fault coverage may be substantial (the detailed analysis of fault coverage versus cost of test will be discussed in Chapter 2).

1.1.4 Other Relevant Topics

Test cost per unit and test equipment capital cost considerations dominate manufacturing test methodology decisions for electronic systems. Most of the test problems causing major yield losses and cost increases are related to the fact that automatic test equipment (ATE) speeds have not been able to keep up with improving device speed, i.e., the

situation shown in Figure 1.1. Thus, low-cost solutions such as Design-for-Test (DFT) enabled designs have recently generated significant industry momentum. The concepts of ATE and DFT are presented in the next two sections.

Automatic Test Equipment (ATE)

Automatic test equipment provides tremendous flexibility, allowing many different types of devices to be tested without changes to the test hardware. In addition to the versatility that this equipment provides, it enables electronic testing of very complex systems.

For most ATE the main costs can be grouped into the following areas:

1. The purchase price (the investment).
2. Site preparation, training and other initial set-up costs.
3. Test programming and test fixture preparation costs.
4. Testing, diagnosis and rework costs.
5. Maintenance costs (hardware and software).
6. The field-service associated with escaping defects.

These will be the main cost areas that make up the total cost of ownership, or more accurately the lifecycle cost, of the test strategies and the testers that are being considered.

As technology evolves, functional test equipment costs have decreased over time for a constant performance window. Test will continue to leverage the functional test methodology as one opportunity to obtain the coverage required to guarantee outgoing product quality. However, it is expected that Design for Testability (DFT) will be used

when needed to limit the functional test performance envelope in production by reducing input/output (I/O) data rate requirements, enabling low pin count testing, and reducing the dependence on expensive instruments. DFT will enable manufacturers to step off the test equipment technology treadmill associated with functional test.

Design for Testability (DFT)

Design-for-testability techniques increase fault coverage and reduce test application and development time. DFT is an approach in which the electronic system is designed from the start in such a way that testing problems are minimized, which refers to hardware or software design styles or added hardware that reduces test generation complexity. The key to DFT is the ability to control and observe directly the internal states of the system under test.

DFT techniques like scan and Built-in Self Test (BIST) either enable test content to be generated automatically or reduce the test content generation effort, thus drastically reducing the manual test writing task. For highly integrated devices, DFT is required to provide re-use of test collateral and avoid a geometric or exponential growth of the test development and validation effort. Testability can be represented as:

$$\text{Testability} = \text{Observability} + \text{Controllability}$$

Observability is the ability to observe a gate output directly at external pins and controllability refers to apply any and all desired inputs to a gate via external pins.

DFT has been a much-discussed topic for several decades. “To DFT or not to DFT?” The answer depends on who is asking and who is answering. Designers might view DFT

as preventing them from achieving the best design with high performance and minimum hardware complexity [17]. Test engineers also still talk about “the wall” that is from the traditional separation of design and test development activities [18]. A significant body of literature has been devoted to address the tradeoffs of doing DFT or not. Many economic test models exist that justify DFT for specific companies, products and markets by examining DFT’s impact on cost [19-22]. Dislis *et al*. [23] have proposed a test planning system that compares different test strategies for a given design based on circuit properties, intended market and company resources. Similar work has been done for full partial scan designs in [24]. A key example is DFT’s improvement in yield learning through the enhancement of diagnosis — a DFT benefit that translates to an improved time-to-market and time-to-volume [25-27]. Nag [17, 28] proposed the analysis of DFT aimed at an end product that maximizes the cost-to-benefit ratio, with the decision to use DFT ultimately based on its impact on profit. They developed the Test Cost Model to studied DFT’s impact on the various IC cases. The debate about the economics of extra silicon for DFT will taper off or end altogether as predicted by Turino [18] ― simply because without DFT, future devices will be impossible to test to required quality levels.

Optimization of the test location(s) and characteristics of test/diagnosis/rework operations can be used as the feedback to DFT and provide suggestions from the view of the yielded cost to answer where we need to put the test and correspondingly, what portions of the system should be redesigned to accommodate the complex testing for a higher level of fault coverage (i.e., DFT) and where there is less need for test to decrease the cost of products.

1.2 Dissertation Scope and Problem Statement

It is essential to perform the tradeoffs among the feature parameters of test/rework operations to minimize the cost of the process flow and reduce the defects in the systems. The tradeoff process is the core of the allocation of test/rework operations. The tradeoff process has the purpose of making the best choice between various versions of given tests and reworks to be allocated, and to suggest the values of variables of the test steps to minimize an objective function associated with the process flow.

The Figure 1.10 helps to understand why we need to find the optimum of the feature parameters. Consider the four different scenarios shown: no test steps, all tests with 100% fault coverage, only one test at the end of the process and the optimum case (the data is from the example process flow in Figure 4.22 computed using the algorithms described in Chapter 4), the comparison of cost and yielded cost is given to demonstrate the best case. The optimum configuration of fault coverage in Figure 1.10 (case 4) has the smallest yielded cost but does not have the lowest cost or highest yield.

The methodology development in this dissertation enables answering the following four questions on an application-specific basis:

(1) How much fault coverage do I need to buy?

(2) Where (in a process) should I put the test locations?

(3) When do I need to do rework?

(4) How many rework attempts should be made?

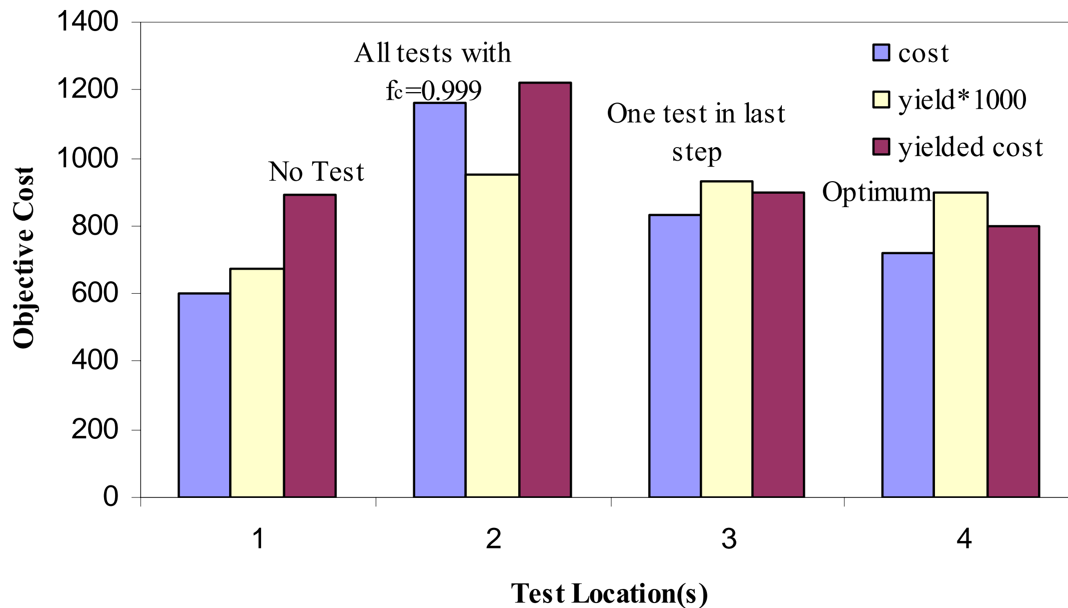


Figure 1.10: The comparison of yielded cost of process flow with four different test cases.

The key issues involved in this dissertation are how to obtain the solution to an optimum allocation of test and rework operations in various scenarios and how to determine the corresponding values of feature parameters in a general process flow. The main objective of this dissertation is to develop an optimization approach to minimize the yielded cost of systems by optimum placement of TDR operation and its characteristics. The optimization can be used during the manufacturing and assembly stages and even in the design of products providing the feedback to a Design for Test (DFT) analysis of the electronic systems to indicate which portion should be redesigned to accommodate the testing for a higher level of fault coverage and where there is less need for test to decrease the cost of products.

To achieve these goals, the following five items will be investigated:

Goal 1: Computations of a single Test/Diagnosis/Rework operation.

Goal 2: Development of a searching algorithm to traverse all process steps in a complex process flow.

Goal 3: Performing multi-variable optimization using real-coded genetic algorithms (RCGAs) to solve the TDR location optimization problem.

Goal 4: Validation of optimization algorithms using the real case with comparison to the calculation results from manually choosing test strategies (without optimization techniques involved).

Goal 5: Exploration of extensions of the TDR optimization for uncertain input(s).

1.3 Overview of the Solution Strategy

This section provides and overview of the general algorithm used to solve the test/diagnosis/rework optimization problem. Chapters 2-4 develop the detailed models and methodologies needed to implement this algorithm, Chapter 5 provides a detailed example of the application of the algorithm in a real system, and Chapter 6 discusses extensions to the algorithm to accommodate uncertain data inputs.

The assumed starting point for the optimization problem described in this dissertation is a manufacturing process for a product (described as a process flow comprised of a sequence of process steps). The goal of the algorithm is to determine where the TDR (test/diagnosis/rework) operations should be placed within the process and the characteristics of the diagnosis and rework operations.

The following algorithm has been developed:

1. **Insert a TDR "package" at every possible location in the process flow.** The possible TDR operations are located between each pair of adjacent process steps, before initial steps and after the final step. The number of test locations that need to be optimized depends on how many separate process steps are included in a process flow.
2. **Vary the fault coverage of all test steps continuously from 0% to 100%.** The recurring test costs are computed as a function of fault coverage using a variable test cost model. High fault coverage always leads to expensive testing.
    a) **Fault coverage threshold is embedded in the optimization algorithm.** The optimization methodology developed in this dissertation chooses the optimum values from the range of fault coverage for each possible test operation that is placed in the process flow to minimize a yielded cost objective function. If the value of fault coverage of a specific test operation falls below a predefined threshold — the minimum nonzero fault coverage that we can practically purchase the test operation will be removed from the process flow, i.e., there is no test present in this location. Instead of separately finding test locations and fault coverage, the optimization algorithm merges the two into a single problem by defining the fault coverage threshold.
    b) **Inclusion of rework is considered concurrent with fault coverage.** The rework operation is associated with the test operation when the

fault coverage of testing is above the threshold, i.e., the rework cannot be done without a test present at the location.

3. **Use overall yielded cost as the objective function to be minimized.** The overall yielded cost is the accumulated cost divided by the final yield of the whole process. The characteristics of all possible TDR operations included in the process flow are treated as the variables that need to be optimized.
4. **A Waiting Sequential Search (WSS) algorithm is used to determine the order for evaluating process steps and controlling the objective function calculation.** This graph-based search algorithm traverses all the steps nodes in the direct graph representation of process flow and does the accumulated computations of yields and cost of products. The search process performs the sequential calculations from the start to the end of the process flow and the single TDR model is used when it meets the step nodes of TDR operations.
5. **Real-coded Genetic Algorithms (RCGAs) are used to manage the optimization**. Genetic algorithm (GA) — a natural optimization method is used as the main optimization technique to solve the minimization problem in this dissertation, in which the variables needs to be optimized are represented as genes. RCGAs find the minimized value of the objective function by performing three kinds of genetic operations — selection, crossover, and mutation.

The entire solution strategy is described by the framework shown in Figure 1.11.

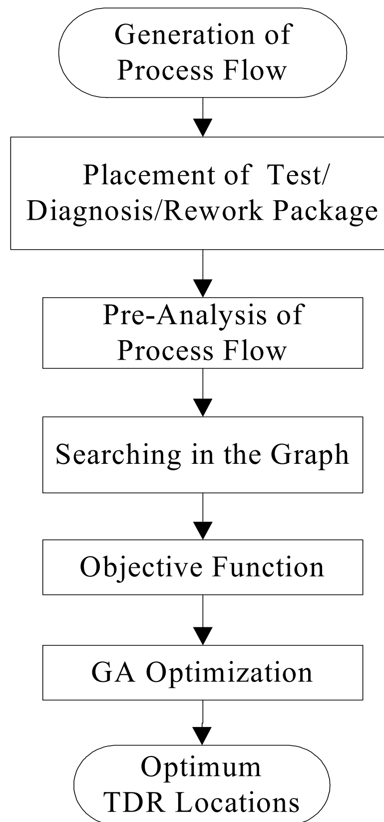


Figure 1.11: The framework of optimization of TDRs in process flow.

1.4 Organization of Dissertation

There are seven chapters in this dissertation. The contents of each chapter are summarized below.

Chapter 2 describes the representation of the test/diagnosis/rework operation using technical cost model. The first section reviews the development of a comprehensive TDR model that integrates the computations of cost and yield of systems into the process flow with the considerations of TDR characteristics — fault coverage, false positives, etc. A variable test cost model is developed to define the functional relationship between fault

coverage and test cost. The extension of the model to include the multiple fault types and rework loops and an alternative TDR cost model are also discussed.

Chapter 3 extends the single TDR model to the computation of the yielded cost of product for multiple TDRs in a process flow. A graph-based search method described in the fourth section, named “Waiting Sequential Search”, is developed to traverse all the step nodes in the graphical representation of the process flow with all possible TDR operations included. A multi-variable cumulative objective function to minimize the cumulated yielded cost of the process flow is derived.

Chapter 4 discusses the implementation of Real-Coded Genetic Algorithms (RCGAs) to optimize the TDR locations and the characteristics associated with them in the process flow. Simple Test cases for four different scenarios are provided to verify the correctness and demonstrate the feasibility of the optimization algorithms.

Chapter 5 applies the optimization methodology to a real case — the multichip module (MCM) assembly process. Results from the optimization methodology developed in this dissertation are compared to manually determined test locations.

Chapter 6 extends the optimization algorithm to process flows with uncertain input(s). Monte Carlo methods are integrated inside the RCGAs to sample the values from the input probability distribution(s). Optimization results of the real MCM assembly process flow under uncertain input(s) are presented.

Chapter 7 summarizes the main research accomplishments of this dissertation and describes the contributions made.

Chapter 2

The Test/Diagnosis/Rework Model

The objective of the test economics model is to accommodate the test/diagnosis/rework (TDR) effects relevant to the electronic system assembly processes. In these processes, defect insertion during test and rework operations is not un-common (e.g., from handling and/or probes making physical contact with the board), false positives[4] can be a significant problem, multiple rework attempts are made when dealing with expensive systems such as multichip modules, and complex rework operations that may include reassembly of significant portions of the system are performed.

This chapter describes a comprehensive test/diagnosis/rework model that includes a detailed formulation of the feature parameters for the single TDR operation. Based on this model, a multi-objective function for optimization can be constructed via the accumulation of cost and yield in a general process flow that is made up of manufacturing process steps and TDR operations. Performing the accumulation through

---

[4] A false positive is a positive test result in subjects that do not possess the attribute for which the test is conducted. Electronic systems test engineers define false positives as a Type I tester error [29]. In testing, this means that a test will identify good product as bad at some non-negligible rate. In fact, data at the board and system level has shown that as many as 46% of all failures are not actually failures, but false positives, [30].

all the process steps with multiple branches and TDR operations requires the development of a new search algorithm that is introduced in Chapter 3.

2.1 Development of the Test/Diagnosis/Rework Model

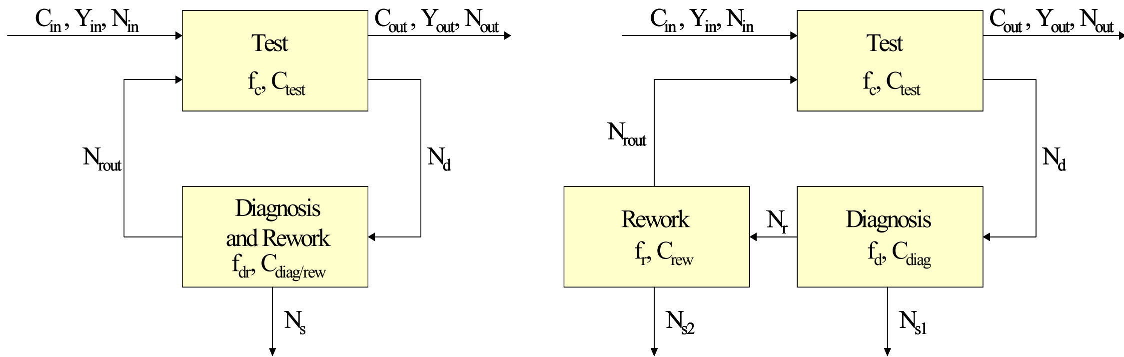


Figure 2.1: Example test/diagnosis/rework models currently in use for technical cost modeling.

There are several existing TDR models that are applicable to technical cost modeling.[5] The basic architectures of TDR models are shown in Figure 2.1.

In the example test/diagnosis/rework models shown in Figure 2.1 all parts coming from production are tested; the diagnosis and repair are applied to all the parts that are identified as defective during the test; and all reworkable parts are retested. Many versions of these models supporting some subset of the variables shown have been developed including single rework attempt models and multiple rework attempt models [32-38]. Athough the detail involved in these models varies, in general they do not account for new defects introduced during the test, diagnosis or rework steps; false

[5] Technical cost modeling is defined as a process-based, “bottoms-up” approach to cost estimation, with total cost broken into a set of individual cost elements. Each of these elements is estimated separately, then summed to provide an estimate of the total cost [31].

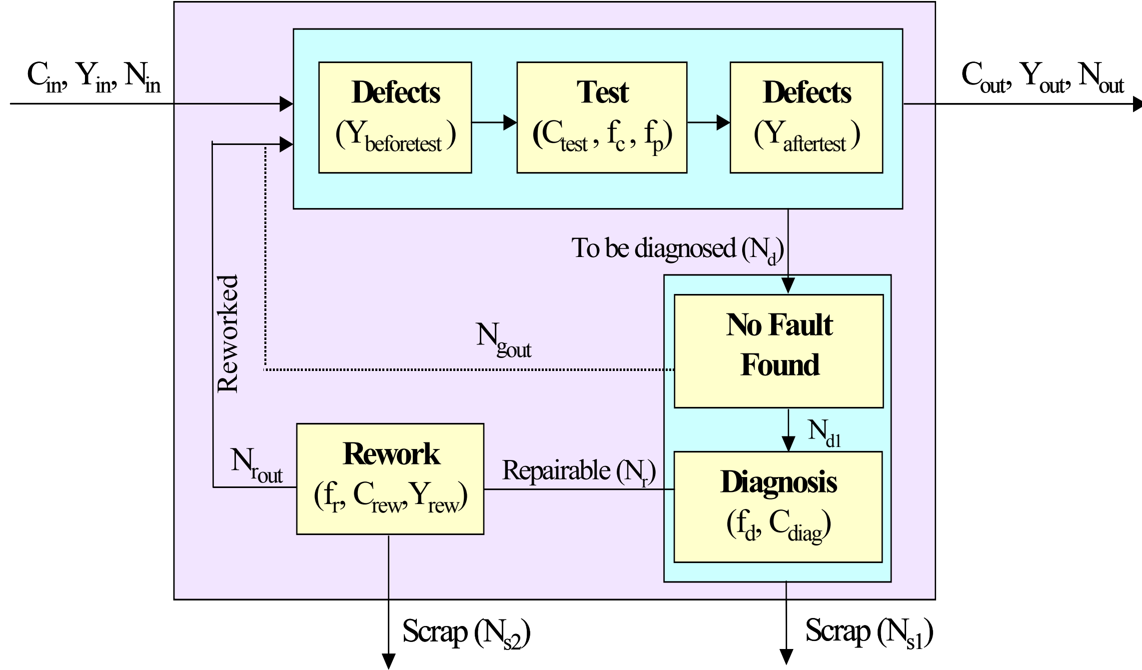


Figure 2.2: Organization of the test/diagnosis/rework model (Table 2.1 describes the notation appearing in this figure).

positives in testing, or uncertainties that may be present the input data. In order to accommodate the additional effects and obtain a model that is readily useable in a technical cost modeling optimization environment, Trichy *et al.* [9] developed a model that accommodates the test/diagnosis/rework effects relevant to printed circuit board fabrication and electronic system assembly processes.

In this work, a modified (corrected, extended and generalized) form of the Trichy *et al.* [9] model was used to compute the feature parameters of test/rework operations. Figure 2.2 shows the content of the test/diagnosis/rework model. In the following description the term "module" was used to refer to the item being tested (e.g., a printed circuit board with chips assembled on it). Inputs to this model are the accumulated cost and yield of upstream processes (C_{in} and Y_{in}), the number of modules (N_{in}) is not a

required input and is only included for convenience in the formulation of the model.[6] Yield is the ratio of non-defective (good) modules to all modules, non-defective and defective (bad). The test portion of the model is the top most group of three steps in Figure 2.2. This model can be used to account for defects introduced by the test operation both prior to the actual test (e.g., loading the module into the tester or stationing the probes on the module) and after the test result is recorded (e.g., unloading the module from the tester). The modules that are determined to be faulty go on to the diagnosis step. Three outcomes are possible from diagnosis: 1) no fault is found in which case the module goes back for retesting, 2) the module is determined to be reworkable and sent on to rework, or 3) the module is determined to be non-diagnosiable or non-reworkable and sent to scrap. The rework process operates on the reworkable modules and scraps modules that can not be successfully reworked. The reworked modules are re-tested and if the reworked modules are found to be faulty again, the modules are again sent for diagnosis. This rework process can be performed a specified number of times (attempts).

There are several key assumptions made in the formulation of this model: defects introduced by the diagnosis step are not explicitly treated; and false positive (f_p) and fault coverage (f_c) act simultaneously and they are independent of each other, i.e., the fault coverage acts only on bad modules and the false positive acts either only on good modules or on all modules.

[6] In general, yield and cost results from this model are independent of N_{in}, however, if equipment, tooling, or other non-recurring costs are included, the results become dependent on N_{in} and can be computed from accumulations of time that specific equipment is occupied or the quantity of tooling used to produce a specific quantity of modules, e.g., see Trichy *et al.* [9].

Table 2.1: Nomenclature used in Figure 2.2 and throughout the discussion in this section.

C_{in}	Cost of a part entering the test/diagnosis/rework process	N_{in}	Number of parts entering the test/diagnosis/rework process
C_{test}	Cost of test/part	N_d	Total number of parts to be diagnosed
C_{diag}	Cost of diagnosis/part	N_{gout}	Number of no fault found parts
C_{rew}	Cost of rework/part	N_{d1}	$N_d - N_{gout}$
C_{out}	Effective cost of a part exiting the test/diagnosis/rework process	N_r	Number of parts to be reworked
f_c	Fault coverage	N_{rout}	Number of parts actually reworked
f_p	False positives fraction, or the probability of testing a good part as bad	N_{s1}	Number of parts scrapped by diagnosis process
f_d	Fraction of parts determined to be reworkable	N_{s2}	Number of parts scrapped during rework
f_r	Fraction of parts actually reworked	N_{out}	Number of a parts exiting the test/diagnosis/rework process
Y_{in}	Yield of a part entering the test/diagnosis/rework process		
$Y_{beforetest}$	Yield of processes that occur entering the test		
$Y_{aftertest}$	Yield of processes that occur exiting the test		
Y_{rew}	Yield of the rework process		
Y_{out}	Effective yield of a part exiting the test/diagnosis/rework process		

Versions of C_{in}, Y_{in} and N_{in} appear both with and without subscripts in the proceeding discussion. When the variables appear with out subscripts they refer to the values entering the process. When they have subscripts, they represent specific iterations in the rework process.

2.2 Trichy *et al.* [9] Model Formulation

The cost incurred by all the parts that eventually pass the test step is given by,

$$C_1 = \sum_{i=0}^{n} \left(C_{in_i} + C_{test}\right) N_{out_i} \tag{2.1}$$

where n is the number of rework iterations allowed, i.e., the maximum number of attempts to rework an individual part and N_{outi} is number of parts passed by the test in the ith rework attempt. When i=0, C_1 is the total cost of the parts that pass the test without ever going through diagnosis or rework. The cost incurred by all the parts scrapped by the diagnosis step is given by,[7]

$$C_2 = \sum_{i=1}^{n-1} \left(C_{in_i} + C_{test} + C_{diag}\right) N_{s1_i} \tag{2.2}$$

and the cost incurred by all the parts scrapped by the rework step is given by,[8]

$$C_3 = \sum_{i=1}^{n-1} \left(C_{in_i} + C_{test} + C_{diag} + C_{rew}\right) N_{s2_i} \tag{2.3}$$

where N_{s1i} and N_{s2i} are defined in (2.8) and (2.9). After the final rework (n^{th} rework attempt), the parts that do not pass the test are scrapped. The first term in (2.4) accounts

[7] In [9], (2), the maximum of the summation should be n-1 instead of n.
[8] In [9], (3), the maximum of the summation should be n-1 instead of n.

for the parts scrapped by the test, and the second term accounts for any false positives that are encountered during the final test,[9]

$$C_4 = \left(C_{in_n} + C_{test}\right)N_{d_n} + \left(C_{in_n} + C_{test}\right)N_{in_n} Y_{in_n} Y_{beforetest} f_p \tag{2.4}$$

where N_{ini} is defined in (2.11). The total cost of all these parts (including scrapped parts) is the sum of C_1 through C_4. The total cost per output part associated with this model is the total cost divided by the total number of output parts (parts that are eventually passed by the test),

$$C_{out} = \frac{C_1 + C_2 + C_3 + C_4}{N_{out}} \tag{2.5}$$

The number of parts moving through the process are shown in (2.6)-(2.11),[10]

$$N_{out_i} = \begin{cases} N_{in_i}\left(1 - f_p Y_{in_i} Y_{beforetest}\right)\left(\dfrac{\left(1 - f_p\right)Y_{in_i} Y_{beforetest}}{1 - f_p Y_{in_i} Y_{beforetest}}\right)^{f_c}, \text{ when } f_p \text{ applies to only good parts} \\ N_{in_i}\left(1 - f_p\right)\left(Y_{in_i} Y_{beforetest}\right)^{f_c}, \text{ when } f_p \text{ applies to all parts} \end{cases} \tag{2.6}$$

$$N_{d1_i} = \begin{cases} N_{in_i}\left(1 - f_p Y_{in_i} Y_{beforetest}\right) - N_{out_i}, \text{ when } f_p \text{ applies to only good parts} \\ N_{in_i}\left(1 - f_p\right) - N_{out_i} + f_p N_{in_i}\left(1 - Y_{in_i} Y_{beforetest}\right), \text{ when } f_p \text{ applies to all parts} \end{cases} \tag{2.7}$$

$$N_{s1_i} = \left(1 - f_d\right)N_{d1_i} \tag{2.8}$$

[9] In [9], (4a) can be used for either definition of f_p with $N_{d_{n+1}}$ changed to N_{d1_n}.

[10] In [9], (13), the subscript of N_r should be i-1 instead of i when i>0.

$$N_{s2_i} = (1 - f_r) N_{r_i} \tag{2.9}$$

$$N_{r_i} = f_d N_{d1_i} \tag{2.10}$$

$$N_{in_i} = \begin{cases} N_{in} \text{ when } i = 0 \\ f_r N_{r_{i-1}} + f_p N_{in_{i-1}} Y_{in_{i-1}} Y_{beforetest} \text{ when } i > 0 \end{cases} \tag{2.11}$$

where parameters without subscripts (N_{in}, C_{in}, and Y_{in}) indicate values entering the process (Figure 2.2) and the form of (2.6) follows from [13]. The total number of parts that successfully pass the test process is given by,

$$N_{out} = \sum_{i=0}^{n} N_{out_i} \tag{2.12}$$

The part counting in (2.6)-(2.11) assumes that all false positives on good parts go through diagnosis and back into test without scrapping of parts in diagnosis or rework. The formulation is also only valid when $f_p < 1$, $Y_{in} > 0$ and $Y_{beforetest} > 0$. The input cost (C_{ini}) that appears in (2.1)-(2.4) is given by C_{in} when i = 0 and by (2.13) when i > 0.

$$C_{in_i} = \frac{\left(C_{in_{i-1}} + C_{test} + C_{diag}\right) f_p Y_{in_{i-1}} Y_{beforetest} N_{in_{i-1}} + \left(C_{in_{i-1}} + C_{test} + C_{diag} + C_{rew}\right) f_r N_{r_{i-1}}}{N_{in_i}} \tag{2.13}$$

The input yield (Y_{ini}) that appears in (2.4) and (2.6)-(2.13) is given by Y_{in} when i = 0 and by (2.14) when i > 0.

$$Y_{in_i} = \frac{f_p Y_{in_{i-1}} Y_{beforetest} N_{in_{i-1}} + Y_r f_r N_{r_{i-1}}}{N_{in_i}} \tag{2.14}$$

The final yield of parts that successfully pass the process is given by,

$$Y_{out} = \begin{cases} \dfrac{\sum_{i=0}^{n} Y_{aftertest} N_{out_i} \left(\dfrac{(1 - f_p) Y_{in_i} Y_{beforetest}}{1 - f_p Y_{in_i} Y_{beforetest}} \right)^{1-f_c}}{N_{out}}, \text{ when } f_p \text{ applies to only good parts} \\ \dfrac{\sum_{i=0}^{n} Y_{aftertest} N_{out_i} \left(Y_{in_i} Y_{beforetest} \right)^{1-f_c}}{N_{out}}, \text{ when } f_p \text{ applies to all parts.} \end{cases} \tag{2.15}$$

Note, N_{in} cancels out of (2.5) and (2.15) making the total cost per part and final yield independent of the number of parts that start the process, which is intuitively correct since no volume sensitive effects (such as material or equipment costs) are included in this recurring cost model.

2.3 Variable Test and Rework Cost Model

The Trichy *et al.* model provides the accurate computation of the feature parameters: C_{out} and Y_{out}, etc. for a single TDR operation in a process flow in which several variables where C_{test} and C_{rew} are defined as the constants. For real processes C_{test} is not a constant and instead is related to the fault coverage of the test. The rework yield (rework success rate) is also not a constant. It depends on the specific rework actions taken. In practice the rework operation may cause additional defects to be inserted in the product [14]. For use in this work, the model in [9] is extended by defining general forms of the relationships

among the feature parameters, e.g., the costs of test and rework in terms of fault coverage and rework yield respectively.

Functional Relationship between Fault Coverage and Test Cost

Empirical data shows that the test process can be divided in two phases. A relatively small subset of T_I (number of tests in Phase I see Figure 2.3) of the total set of tests provides a fault coverage ranging from 65% to 85% for most combinational logic circuits [39]. For Phase II of the test generation, the number of additional tests required is approximately a linear function of the number of untested faults remaining at the end of Phase I. Unlike Phase I, in Phase II each generated test tends to detect fewer faults than the one before it and the average cost per detected fault increases.

For the purpose of simplifying the relationship, an exponential function can be used to approximately simulate Phase I and a linear function in Phase Π on the assumption that Phase II could not reach a full coverage (100% of fault coverage) in practical testing.

A relationship between the cost of test (proportional to test time or number of tests) and fault coverage has been suggested by Goel [39]. The quantitive expression of untested faults with tests for combinational logic circuits is given in (2.16)

$$T_0 = T_I + Be^{-a}\left(\frac{G}{k}\right) \tag{2.16}$$

where:

T_0 = the number of tests required to achieve a maximum possible test coverage;

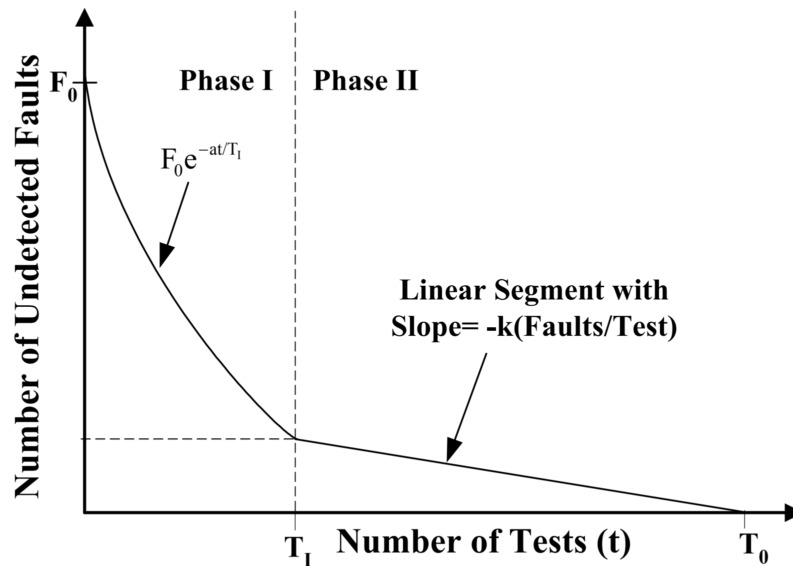


Figure 2.3: Typical curve of untested faults versus the number of tests.
"a" is a constant whose value is in the range 1-2 for a given logic structure.

T_I = the number of tests at the end of Phase I (see Figure 2.3) and is relatively small when compared with T_0;

G = the number of gates in the combinational circuit;

B = the average number of non-equivalent single stuck faults per gate in the circuit;

-k = the slope of Phase II of the curve in Figure 2.3 where k is the number of faults detected per test in Phase II;

F_0e^{-a} defines the number of untested faults at the end of Phase I, and a ranges from 1 to 2 for most circuits.

Assuming that the test cost is proportional to the number of tests, a relationship between test cost and fault coverage can be derived,

$$C_{test} = p_t[b_t \ln(1 - f_c) - r_t] + C_{ft}, f_c \in (0,1) \quad (2.17)$$

where p_t is the cost coefficient; b_t is the coefficient of test characteristic, r_t is the fault ratio, f_c is the fault coverage of the test, C_{ft} is the fixed cost of test.[11] Figure 2.4 shows a plot of (2.17) using the values in Table 2.2.

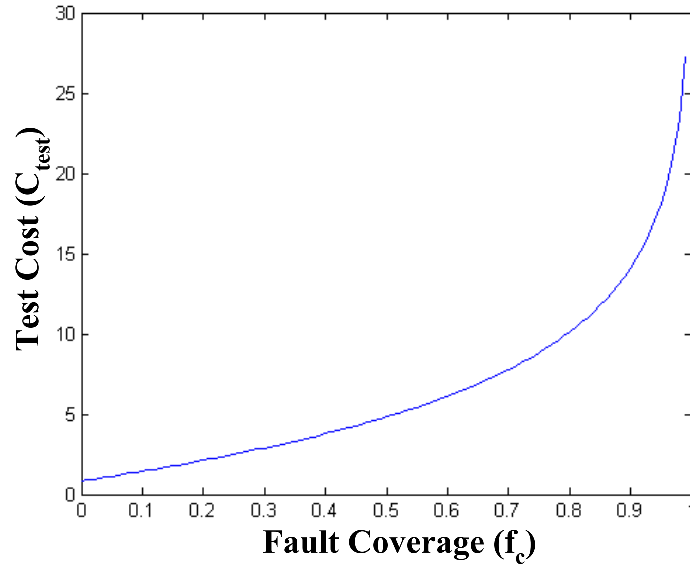


Figure 2.4: Example relationship between the test cost and fault coverage.

Table 2.2: Example values of factors in (2.17) and (2.18).

p_t	b_t	r_t	C_{ft}	p_r	b_r	r_r	C_{fr}
0.02	-288	8.2	1	0.02	-300	10	1

There is similar relationship between rework yield (y_r) and rework cost (C_{rew}),

$$C_{rew} = p_r[b_r \ln(1 - y_r) - r_r] + C_{fr}, y_r \in (0,1) \quad (2.18)$$

[11] C_{ft} accounts for fixed costs associated with testing, i.e., there is a minimum fixed cost for having even a very small fault coverage.

Fault Coverage Threshold

In (2.17) a small cost of test will be predicted for a small fault coverage. Actually, depending on the type of the test operation there is a minimum test cost that is independent of fault coverage at very small fault coverage, i.e., the capital expense of the test equipment. A threshold is defined as the minimum nonzero fault coverage that we can practically purchase to remove the non-effective test. For any fault coverages below this threshold, there is no test considered (zero fault coverage and zero test cost). An

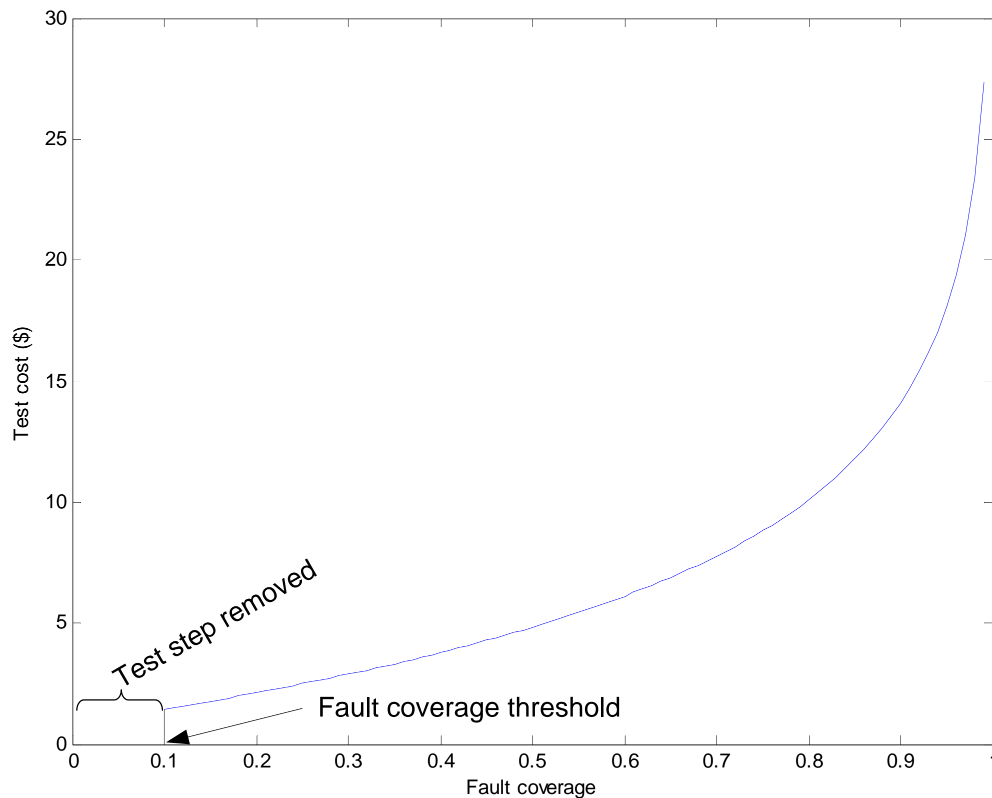


Figure 2.5: Example relationship between the test cost and fault coverage with 10% threshold.

example functional relationship between test cost and fault coverage with 10% threshold is shown in Figure 2.5.

The concept of having a choice of the fault coverage to purchase is straightforward – fault coverage is a measure of the ability of a set of tests (a collection of test vectors) to detect a given class of faults that may occur in a device under test, the fault coverage attained with a test is dependent on the number of test vectors exercised, which determines the test time and thereby the test cost. In the case of rework, the rework yield (really the rework success rate) depends on types of faults that are selected for repair and potentially the thoroughness of that repair.

Functional relationships between fault coverage and test cost, and rework yield and rework cost obviously depend on the type of system being considered. The relationships in (2.17) and (2.18) were used for the remaining work in this dissertation as examples only. The methodology that is the subject of this dissertation, will work successfully with alternative models.

2.4 Application of the TDR Model When Multiple Faults are Present

It is common to have multiple independent faults in an electronic systems assembly, which require effectively independent TDRs.[12] The TDRs for the multiple faults can be modeled as parallel test/rework operations that may occur at each process steps. Figure 2.6 shows an example of a pair of parallel test operations that treat different fault types.

[12] Note, a single type of test could have different fault coverages for different fault types.

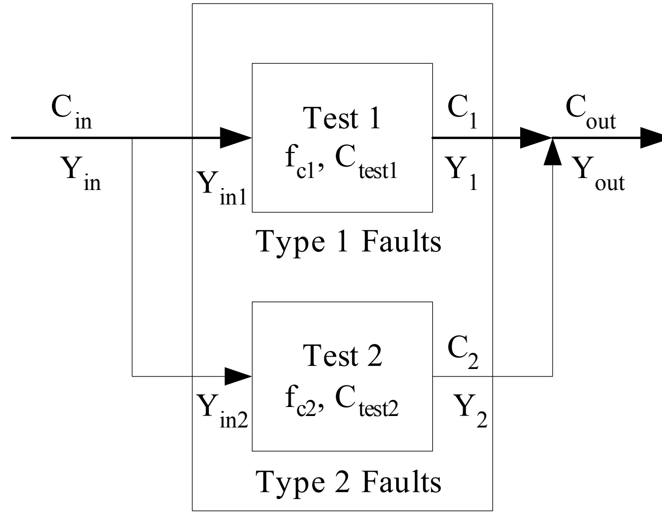


Figure 2.6: Parallel test operations.

In Figure 2.6, there are two types of test operations responsible for addressing two kinds of faults. The TDR can be divided into two parallel test operations, no rework is considered here for simplicity. According to the functional relationship in Figure 2.6, we have,

$$Y_{in} = Y_{in1} Y_{in2} \tag{2.19}$$

where Y_{in1} and Y_{in2} could represent the product yield with respect to different independent defect mechanisms. If this is the case then,

$$Y_{out} = Y_1 Y_2 = \frac{Y_{in}}{Y_{in1}^{f_{c1}} Y_{in2}^{f_{c2}}} \tag{2.20}$$

$$C_{out} = C_1 + C_2 = \frac{C_{in} + C_{test1}}{Y_{in1}^{f_{c1}}} + \frac{C_{in} + C_{test2}}{Y_{in2}^{f_{c2}}} \tag{2.21}$$

if we denote the probability that faults of type 1 will happen as P_1 and the probability of faults of type 2 is P_2 ($P_1+P_2=1$), then the following relationships can be derived,

$$Y_{in1} = 1-(1-Y_{in})P_1 = 1-P_1+Y_{in}P_1 \tag{2.22}$$

$$Y_{in2} = 1-(1-Y_{in})P_2 = 1-P_2+Y_{in}P_2 \tag{2.23}$$

So given the probability of defects and the characteristics of the TDR, the yielded cost for multiple faults can be derived from the computation of single TDR operation.

2.5 Relationship of the TDR Model to Loop Number

Loop number is as common metric used to describing systems subject to rework. It is useful to relate the loop number to the TDR model formulation in this chapter. A tester can theoretically detect all faults with one test operation. In practice, for example, if there two kinds of possible faults—Fault A and Fault B, it is normal to set up the system so that it stops testing at the end of the Fault A test portion of the test program, if any Faults A have been detected. The reason for this is that the presence of Fault A will make it difficult or even impossible to measure or test accurately some of Faults B. This can result in inaccurate diagnosis messages and possibly cause the system to indicate the presence of faults where none exist (false positives). So any parts in which both Faults A and Faults B are present will pass through the entire diagnosis/rework loop a minimum of two times. Therefore, the average number of diagnosis/rework loops or attempts, even if all diagnosis and rework is 100 percent correct, will be something greater than 1.0.

If the diagnosis and rework operations are 100 percent correct and additional faults are added during the rework process, then the loop number[13] — average number of times each faulty part passes around the diagnosis/rework loop [14] can be defined by,

$$\text{Loop Number} = \frac{\text{PFB} + \text{PFB}_b}{\text{PFB}} \tag{2.24}$$

where PFB is the probability of a part being faulty—or the proportion of boards that contain one or more faults and PFB_b is the probability of a failing part having both Faults A and B. For the single type of fault case, PFB_b is zero and then the loop number will be 1, which means that faulty parts with just one type of fault can be repaired completely after the first visit to the rework operation.

In practice, the rework operation is never performed correctly and completely for all defects on each visit to rework operation for the following reasons:

1. The diagnosis from the tester may be incorrect or ambiguous.
2. The diagnostic message may be interpreted incorrectly.
3. The rework operation may cause another defect (characterized using rework yield).
4. The tester may have 'detected' a non-existent defect (false positive).

Whatever the reason, there will usually be more rework operations than the number of faults originally present in the parts. To allow for this in modeling rework operations, a correction factor to the calculations involving diagnosis or rework is often applied. This

[13] The loop number corresponds to the rework attempts — the number of rework iterations allowed in the Trich *et al.* model.

correction factor is usually referred to as the loop-number multiplier. There are several ways that this factor can be determined, but the simplest thing to do is to estimate a percentage of incorrect or unnecessary rework operations. This can be based upon past experience, if the data exists, or a best guess at what the figure might be. The loop-number multiplier has the following relationship relative to the rework yield in Trichy *et al.* TDR model,

$$\text{Loop Number Multiplier} = 1 + (1 - Y_{rew}) = 2 - Y_{rew} \qquad (2.25)$$

where Y_{rew} is the rework yield. For example, if rework yield of the rework operation is 0.8, the loop-number multiplier will be set at 1.2, which implies that 20 percent more rework operations than are theoretically required will take place. In this dissertation, the rework operation is characterized by rework attempts and rework yield instead of using loop number and loop number multiplier.

2.6 An Alternative Model of TDR Operation Cost

An alternative model for calculating the rework cost has been proposed in [40], which has been used to predict the cost of test, diagnosis, and rework activities in the manufacture of printed wiring assemblies (PWAs).

The model in [40] calculate the cost of rework as,

$$C_{rew} = t_{rew}\left(W_{rew} + \frac{M_{rew} + C_{R.Prog}}{7{,}200{,}000NS} \right) \qquad (2.26)$$

where

M_{rew} = total cost of rework equipment (reflow device, controller, fittings, inspection equipment, etc.),

W_{rew} = wage paid to rework technician(s),

t_{rew} = average time to rework a faulty PWA,

$C_{R.Prog}$ = cost of initial reflow controller programming,

S = number of shifts worked,

N = number of years in the payback period (Note that 50 weeks per year times 40 hours per week times 3600 sec/hour yields 7,200,000 seconds per shift-year).

(2.26) relates the rework cost to the cost of rework equipment, and the wages and cost of software. C_{rew} can be referred as the fixed cost of rework in (2.18) — C_{fr}. But there is no relationship between the cost of rework and rework yield in (2.26) as in (2.18).

The model in [40] also considers the rework attempts in the calculating the cost of TDR operations through the following formulations. Since each reworked PWA must be tested again, the diagnostic and rework cost per PWA for ith rework loop is written by,

$$CDR_i = (C_{diag} + C_{rew} + C_{test})(1 - FPY)x_c^{i-1} \tag{2.27}$$

where CDR is the cost of diagnosis and rework operations, C_{test}, C_{rew}, and C_{diag} are the cost of test, rework and diagnosis operations respectively, x_c is defined to be the proportion of reworked PWA's, which contain a fault, FPY is the first pass yield (i.e., the fraction of PWA's which pass the first circuit test). If the number of rework loops is n,

the cost of the TDR can be calculated by accumulating all the diagnosis and rework costs of the PWA for each rework loop and the cost of test. The final cost of the PWA is given by,

$$\begin{aligned} \mathrm{CTDR} &= C_{test} + \sum_{i=1}^{n} (C_{diag} + C_{test} + C_{rew})(1 - \mathrm{FPY})(x_C)^{i-1} \\ &= C_{test} + \frac{(C_{diag} + C_{test} + C_{rew})(1 - \mathrm{FPY})(x_C{}^{n} - 1)}{x_C - 1} \end{aligned} \quad (2.28)$$

(2.28) gives the calculation of cost of TDR operations for multiple rework attempts on the assumption that virtually all reworked PWA's pass the second circuit test However, (2.28) does not consider the impacts of false positive and feature parameters of TDR to the cost of the PWA compared with Trichy *et al.* model. The model also does not discuss the functional relationship between the cost of test and fault coverage.

2.7 Single TDR Process Flow Calculation

For the purposes to demonstrate, Figure 2.7 shows a single TDR process model with the following assumptions imposed:

- Whatever rework claims is repaired is in fact repaired, i.e., single pass rework
- Rework, diagnosis and test do not introduce any new defects
- The test step does not have any false positives.

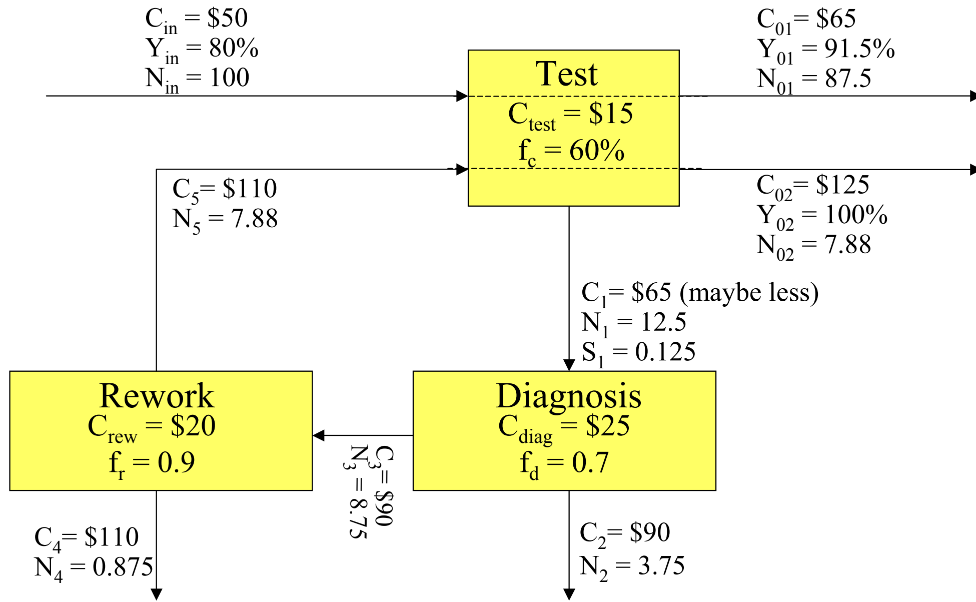


Figure 2.7: Single pass rework numerical example.

Given the inputs C_{in}, Y_{in}, and N_{in}, and the characteristics of each step in the process (shown inside the boxes), the number of parts, their cost, and yield can be computed on each branch (arrow) subject to the three assumptions above. Using the relations developed in Section 2.2, the values of the costs, yields and quantities traced through the process are given by,

$$\left.\begin{aligned} C_{01} &= C_{in} + C_{test} = 50 + 15 = 65 \\ Y_{01} &= Y_{in}^{(1-f_c)} = 0.8^{(1-0.6)} = 0.915 \\ N_{01} &= PN_{in} = Y_{in}^{f_c} N = 0.8^{0.6} 100 = 87.5 \end{aligned}\right\} \text{Product passed by the test ignoring rework}$$

$$\left.\begin{aligned} C_1 &= C_{in} + C_{test} = 50 + 15 = 65 \\ N_1 &= N_{in} - N_{01} = 100 - 87.5 = 12.5 \\ S_1 &= 1 - P = 1 - Y_{in}^{f_c} = 1 - 0.8^{0.6} = 0.125 \end{aligned}\right\} \text{Product scrapped by the test}$$

$$\left.\begin{aligned} C_2 &= C_1 + C_{diag} = 65 + 25 = 90 \\ N_2 &= (1 - f_d) N_1 = (1 - 0.7) 12.5 = 3.75 \end{aligned}\right\} \text{Product scrapped by the diagnosis}$$

$$\left.\begin{aligned} C_3 &= C_1 + C_{diag} = 65 + 25 = 90 \\ N_3 &= f_d N_1 = (0.7) 12.5 = 8.75 \end{aligned}\right\} \text{Product passed by the diagnosis}$$

$$\left.\begin{aligned} C_4 &= C_4 + C_{rew} = 90 + 20 = 110 \\ N_4 &= (1 - f_r)N_3 = (1 - 0.9)8.75 = 0.875 \end{aligned}\right\} \text{Product scrapped by the rework}$$

$$\left.\begin{aligned} C_5 &= C_4 + C_{rew} = 90 + 20 = 110 \\ N_5 &= f_r N_3 = (0.9)8.75 = 7.88 \end{aligned}\right\} \text{Product successfully repaired by the rework}$$

$$\left.\begin{aligned} C_{02} &= C_5 + C_{test} = 110 + 15 = 125 \\ Y_{02} &= 1.0 \\ N_{02} &= N_5 = 7.88 \end{aligned}\right\} \text{Repaired product passed by the test}$$

So, the total quantity of product continuing through the process (ultimately passed by the test) is given by (2.29),

$$N_{out} = N_{01} + N_{02} = 87.5 + 7.88 = 95.38 \tag{2.29}$$

The yield of the product passed by the test step is derived from (2.30),

$$Y_{out} = \frac{\text{good product passed by the test}}{\text{all product passed by the test}} = \frac{Y_{01}N_{01} + N_5}{N_{out}} = \frac{87.88}{95.38} = 0.9214 \tag{2.30}$$

The total money spent on all the product in this process is calculated in (2.31),

$$C_{01}N_{01} + C_1N_1 + C_2N_2 + C_4N_4 + C_{02}N_{02} = \$7106 \tag{2.31}$$

So, the effective cost per passed part and the effective cost per good passed part (yielded cost) are given by (2.32) and (2.33),

$$C_{out} = \frac{7106}{87.5 + 7.88} = \$74.50 \tag{2.32}$$

$$C_Y = \frac{74.50}{0.9214} = \$80.86 \tag{2.33}$$

and the total fraction of the original product that is scrapped by the process is given by (2.34),

$$S_{total} = \frac{N_2 + N_4}{N_{in}} = 0.046 \tag{2.34}$$

If we considered the process shown in Figure 2.7 without any rework (just scrapping the product that the test step considers bad on the first pass), the output could be calculated by (2.35) to (2.39),

$$N_{out} = N_{01} = 87.5 \tag{2.35}$$

$$Y_{out} = Y_{01} = 0.915 \tag{2.36}$$

$$C_{out} = \frac{C_{01}N_{01} + C_1N_1}{N_{out}} = \$74.31 \tag{2.37}$$

$$C_Y = \frac{74.31}{0.915} = \$81.22 \tag{2.38}$$

$$S_{total} = \frac{N_1}{N_{in}} = 0.125 \tag{2.39}$$

Comparing these results to the results with the diagnosis and rework process, we see that although the cost per passed product increased when rework was used (this should be intuitive), the yielded cost per passed product decreased. In fact, if the yielded cost per passed product does not decrease when rework is used, then very possibly products should be scrapped rather than reworked.

2.8 Calculation of the Yielded Cost

After the calculation of accumulated cost and final yield for process flows has been performed, it is necessary to formulate the yielded cost. Yielded cost will be used as the objective function in the optimization process (see Section 3.5).

Yielded cost (C_Y) has been defined as the effective cost per good product instance passed by the process flow, which is a ratio of the final cumulative cost and final yield, [15]:

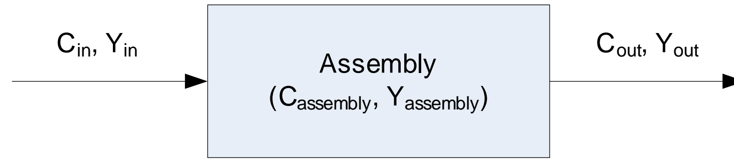


Figure 2.8: A simple non-test process flow.

$$C_Y = \frac{C_{out}}{Y_{out}} \tag{2.40}$$

The calculations of output yield and output cost have been categorized as the following:

Case 1: Single-step process. There is just one process step and no TDR operation involved in the process. For example, consider the single assembly process in Figure 2.8, the assembly step has its own cost ($C_{assembly}$) and yield ($Y_{assembly}$). C_{in} and Y_{in} represent the incoming cost and yield respectively (determined by whatever activities take place prior to this process step). The calculation of the outgoing cost and yield of the process is given by (2.41) and (2.42) respectively.

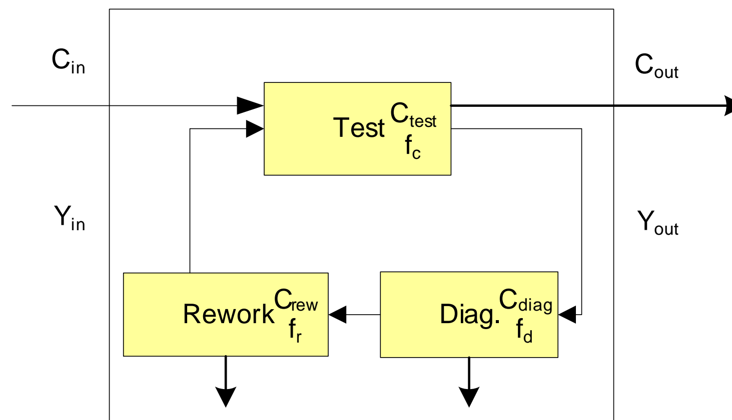


Figure 2.9: A single-TDR process flow.

$$C_{out} = C_{in} + C_{assembly} \tag{2.41}$$

$$Y_{out} = Y_{in} Y_{assembly} \tag{2.42}$$

Case 2: Single-TDR process. If one TDR package is associated with the product (see Figure 2.9), the calculation of output cost and output yield are given by the Trichy *et al.* model in Section 2.2. The outgoing cost is:

$$C_{out} = \frac{C_1 + C_2 + C_3 + C_4}{N_{out}} \tag{2.43}$$

C_1, C_2, C_3, C_4 and N_{out} can be calculated using Trichy *et al.* model based on the input cost and yield and given values of characteristics of the TDR operation, e.g., fault coverage (f_c), rework fraction (f_r), and diagnosis fraction (f_d) etc. The outgoing yield after the TDR package is:

$$Y_{out} = \begin{cases} \dfrac{\sum_{i=0}^{n} Y_{aftertest} N_{out_i} \left(\dfrac{(1 - f_p) Y_{in_i} Y_{beforetest}}{1 - f_p Y_{in_i} Y_{beforetest}} \right)^{1-f_c}}{N_{out}}, \text{when } f_p \text{ applies to only good parts} \\ \dfrac{\sum_{i=0}^{n} Y_{aftertest} N_{out_i} \left(Y_{in_i} Y_{beforetest} \right)^{1-f_c}}{N_{out}}, \text{when } f_p \text{ applies to all parts.} \end{cases} \tag{2.44}$$

Case 3: Complex process flows that include multiple assembly process steps and TDR packages and multiple branches. The final cost and yield of the process must be accumulatively calculated using a search algorithm to traverse each process step (including all possible TDR packages) according to the sequence defined by the process.

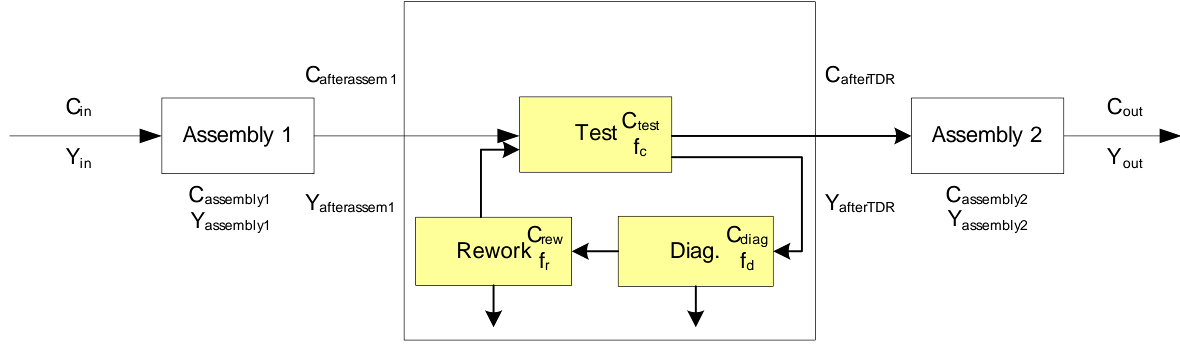


Figure 2.10: An example process flow with two assembly steps and a TDR package.

During the search process, the algorithm will check the property of the next step (general assembly or TDR operation); if the step is a general assembly step then (2.41) and (2.42) in Case 1 are used to calculate the outgoing cost and yield after the step. If the step is a TDR package then (2.43) and (2.44) in Case 2 are applied to derive the output cost and yield of product after the package.

For example, in the process flow shown in Figure 2.10, there are two assembly steps and one TDR inserted between them. In order to derive the final cumulative cost and final yield of the process, the search algorithm will start from step of "Assembly 1" and calculate the cost and yield after the "Assembly 1" step using equations from Case 1 because it is a general assembly step. The $C_{afterassem1}$ and $Y_{afterassem1}$ can be given by (2.45) and (2.46):

$$C_{afterassem1} = C_{in} + C_{assembly1} \tag{2.45}$$

$$Y_{afterassem1} = Y_{in} Y_{assembly1} \tag{2.46}$$

Next, $C_{afterassem1}$ and $Y_{afterassem1}$ become the input cost and input yield to the TDR operation respectively. The outgoing cost ($C_{afterTDR}$) and yield ($Y_{afterTDR}$) can be calculated

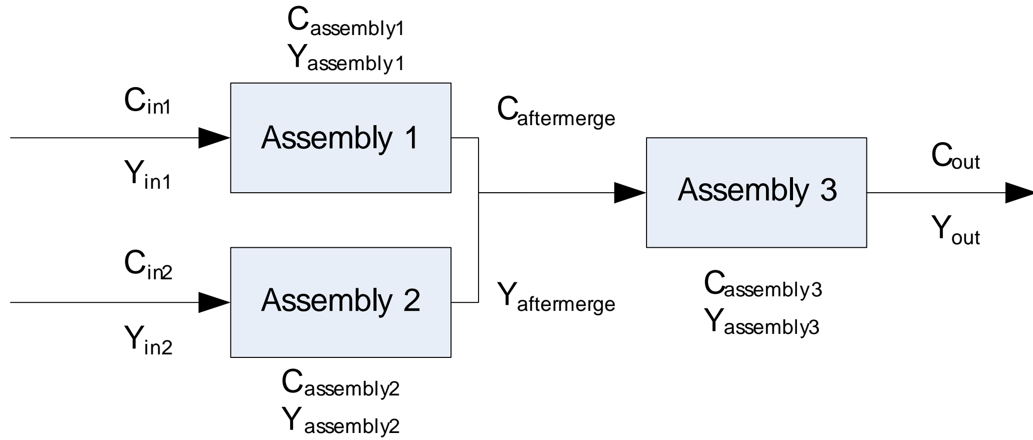


Figure 2.11: An example process flow with two incoming branches.

using the equations from Case 2. The last process step "Assembly 2" in the process flow in Figure 2.10 is another assembly step so the final cumulative cost and final yield of the product after the entire process can be derived by (2.47) and (2.48) respectively:

$$C_{out} = C_{afterTDR} + C_{assembly2} \tag{2.47}$$

$$Y_{out} = Y_{afterTDR} Y_{assembly2} \tag{2.48}$$

Another example process flow with two incoming branches is shown in Figure 2.11. To calculate output yield and cost after the branched process, first the outgoing cost and yield for each branch is calculated then the branches are merged together by adding the costs and multiplying the yields. The cost ($C_{aftermerge}$) and yield ($C_{aftermerge}$) after two branches are merged in the process flow in Figure 2.11 are given by (2.49) and (2.50) respectively:

$$C_{aftermerge} = (C_{in_1} + C_{assembly_1}) + (C_{in_2} + C_{assembly_2}) \tag{2.49}$$

$$Y_{aftermerge} = (Y_{in_1} Y_{assembly_1})(Y_{in_2} Y_{assembly_2}) \tag{2.50}$$

The final output cost and yield after "Assembly 3" step are calculated by (2.51) and (2.52):

$$C_{out} = C_{aftermerge} + C_{assembly_3} \tag{2.51}$$

$$Y_{out} = Y_{aftermerge} Y_{assembly_3} \tag{2.52}$$

The cumulative calculation of output cost and yield for simple processes can be easily performed using hand calculations or spreadsheets. For general analysis of complex process flows with multiple branches, a search algorithm is required to traverse each process step in the process flow and control the accumulation of final cost and yield of product. The details of searching algorithm for the complex process flow will be addressed in Chapter 3.

2.9 Summary

This chapter reviewed the development of a comprehensive TDR model which integrates the computations of cost and yield of systems into the process flow with the considerations of TDR characteristics — fault coverage, false positives, etc., in which several corrections of the formulations are made to Trichy *et al.* model. The variable test cost model was developed to define the functional relationship between fault coverage and test cost instead of using a fixed constant. The extension of the model to include multiple fault types and rework loops and an alternative TDR cost model were also discussed in this chapter. To demonstrate the application of Trichy *et al.* model, a simplified scenario is given to illustrate the calculation of process flow.

After the single TDR operation is resolved, the next issue is to cumulatively compute the objective function of feature parameters of a general process flow that includes multiple TDR operations. For the purpose of derivation of the objective function, search algorithms are needed to traverse the process flow with the computation performed according to the sequence of process steps. The Chapter 3 presents a graph-based search algorithm to fit to the typical structure of a general process flow.

Chapter 3

Search Algorithm for Use in the Process Flow

Using Test/Diagnosis/Rework model in Chapter 2, the feature parameters of a single test step can be computed. A general process flow may, however, have many different TDR activities located within it. The next issue to be addressed is how to obtain the objective function (yielded cost, i.e., cost divided by yield) of an entire general process flow. For the general analysis, a portion of a complex process flow is given in Figure 3.1. The corresponding simplified parameters describing the process steps are also defined within the boxes representing each of the process steps.

In order to analyze general process flows, which can be viewed as a directed graph — digraph[14] [41, 42], Graph Theory is introduced to transform the process flow into a directed graph, and Waiting Sequential Search (WSS) algorithms are applied to traverse all the process steps of the process flow to obtain the objective function for the purpose of optimization.

[14] A directed graph is one in which the edges have direction. Directed edges are called arcs.

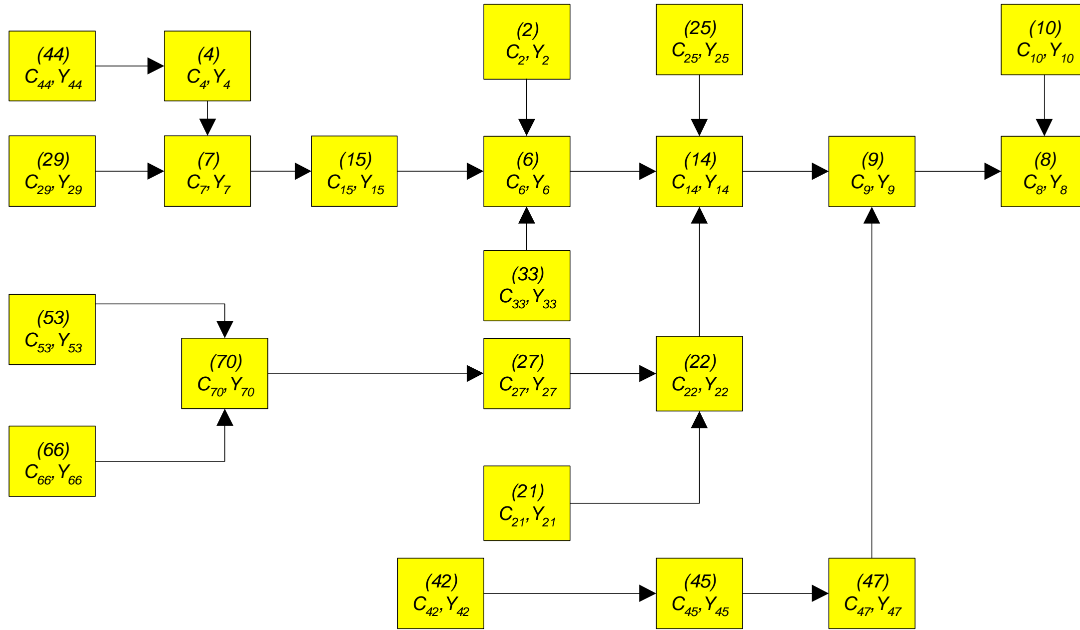


Figure 3.1: A portion of a complex process flow.

3.1 Graphical Representation of a Process Flow

First, we review basic graph notations that are used in this dissertation. A graph G consists of a set of nodes V and a set of edges E, $G = < V, E >$. Here G denotes the whole process flow, where V denotes the set of process steps and an edge $< X, Y > \in E$ denotes the directed flow between the two process steps. The degree of a node is the number of the neighbors adjacent to it. We write $X \rightarrow Y$ when $< X, Y > \in E$ is in a directed graph [43]. We define $\text{PRED}(X)$ as the set of all predecessors of node X, and SUCC (X) as the set of successors of X. if $X \rightarrow Y$, then $X \in \text{PRED}(Y)$ and $Y \in \text{SUCC}(X)$. The *indegree* $\text{id}(X)$ of a node X is the cardinality of PRED (X) and the *outdegree* $\text{od}(X)$ of a node X is the cardinality of SUCC (X). The $\text{indegree}(X)$ is the number of edges of the from $W \rightarrow X$, which means that there are branches merge to X at the amount of

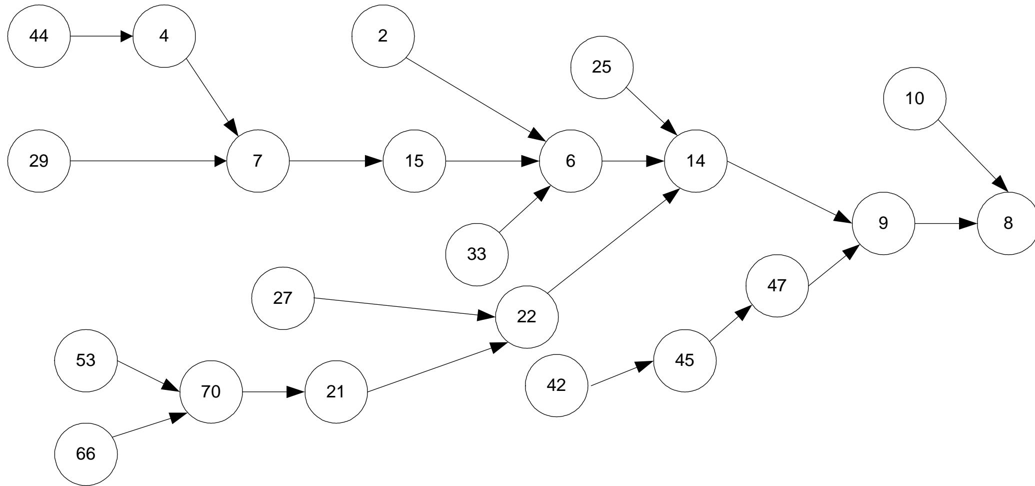


Figure 3.2: Graphical representation of process flow given in Figure 3.1.

indegree(X), and the outdegree(X) is the number of edges $X \rightarrow Y$. If $W \rightarrow X$, then $W \in id(X)$ and $X \in od(W)$. So the od(W) is a subset of SUCC (X) and the id(X) is a sub set of PRED (X). The graph representation of the process flow in Figure 3.1 is shown in Figure 3.2.

3.2 Definitions of Basic Types of Vertices (Process Steps)

From the analysis of a generic process flow like the one in Figure 3.2, four types of basic steps have been identified. The process steps can be defined in the following ways (v denotes the process step):

- Start Step, $od(v) = 1$ and $id(v) = 0$

There are no inputs to the Start Step and just one output from it. There may be multiple Start Steps in a complex process flow.

- Sequence Step, $od(v) = 1$ and $id(v) = 1$

There is one input and one output for a sequence step. Sequence Steps are the most common type of step in process flows for electronic systems.

- Cross Step, $od(v) = 1$ and $id(v) \geq 1$

There are multiple inputs and just one output associated with this kind of step. The complexity of the problem is significantly increased by each Cross Step in the process flow.

- End Step, $od(v) = 0$ and $id(v) \geq 1$

An End Step represents the end of the process flow, which merges all the branches to one. The objective function of the process flow is derived from an End Step. There may only be one End Step in a process flow.

The graph representation of process flow is called an in-branching tree[15] in which there is always the path in the graph from every Start, Sequence and Cross step to the End step.

3.3 Representations of Graphs ─Adjacency Matrix

There are various matrix representations of graphs, each of which take a different view of the graph that is used for a different purpose [42]. An adjacency matrix [41,42] is the

[15] In-branching tree is defined as a rooted spanning tree in a directed graph, such that there is a path from each vertex to the root, as opposed to an out-branching tree (a rooted spanning tree in a directed graph, such that there is a path from the root to each vertex).

most suitable to represent the problem discussed in this dissertation because the edges from all the vertices that belong to od(X) to process step X need to be checked to determine whether they have been visited before continuing the computation of the characteristics of X. It is very convenient to obtain the adjacency information among the vertices using adjacency matrix.

An adjacency matrix representation of a graph describes the graph in terms of the adjacency of every pair of vertices in G. A matrix S is called the vertex adjacency matrix of a graph. The matrix S is always a square matrix. Elements on the diagonal of S represent loops. The number of nonzero elements in S is less than or equal to the total number of arcs and loops in a directed graph. The Table 3.1 gives the adjacency matrix representation of Figure 3.2.

Table 3.1: Adjacency matrix of complex process flow given in Figure 3.1.

	V_{2}	V_{4}	V_{6}	V_{7}	V_{8}	V_{9}	V_{10}	V_{14}	V_{15}	V_{21}	V_{22}	V_{25}	V_{27}	V_{29}	V_{33}	V_{42}	V_{44}	V_{45}	V_{47}	V_{53}	V_{66}	V_{70}
v_{2}	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
V_{4}	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
V_{6}	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
V_{7}	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0
V_{8}	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
V_{9}	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
V_{10}	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
V_{14}	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
V_{15}	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
V_{21}	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0
V_{22}	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0
V_{25}	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0
V_{27}	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0
V_{29}	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
V_{33}	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
V_{42}	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0
V_{44}	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
V_{45}	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0
V_{47}	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
V_{53}	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1
V_{66}	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1
V_{70}	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0

3.4 Waiting Sequential Search (WSS)

Various search algorithms for graphs have been described in the literature, [42-46]. These search algorithms are applicable to problems that need to find shortest path from one vertex to another, to check whether the graph is connected or to find a spanning tree. The problem in this dissertation requires traversing the whole graph of the process flow according to the sequence among the process steps from Start to End. Existing algorithms that have been applied to resolve similar search problems include [47-54]. Most of these algorithms consider the stochastic process in the graph of the fault diagnosis and repair but there is no projection on the practical process flow. For the process flow problem addressed in this dissertation, there are several specific features different from the classical ones, which make it attractive to propose a new search algorithm to perform an efficient search.

From the Figure 3.2, the following features of the general complex process flow can be observed:

- The $\mathrm{od}(v) \le 1$ is always true for all the vertices (process steps) in the process flow;
- There are sequential search requests for the graph, i.e., only when all the predecessors $\mathrm{PRED}(Y)$ of $v(Y)$ have been visited, then $v(Y)$ could be visited.

According to this structure of the process flow, a waiting sequential search algorithm has been developed to search all the steps of the process flow. WSS begins from the lowest-numbered vertex that belongs to a Start Step then proceeds to search the next step. By checking the type of the successor, the algorithm decides to continue to search to the

next Sequence Step or wait at the Cross or End steps. After moving to the next step, the corresponding computation of feature parameters of the previous step is performed and the outcome is stored in a data table in which all the property information associated with process steps are recorded. If Cross or End types of steps are encountered, the visitation status of all predecessors will be checked. If all the predecessors have been checked, the searching continues, if not, the search begins from another Start Step type of vertex until the last Start Step vertex is visited. The algorithm requires that the searching of the next step continue only after all the branches of the present step have been visited. There are waiting actions for the sequential search based on the characteristics of the process flow. A detailed flowchart of the algorithm is shown in Figure 3.3.

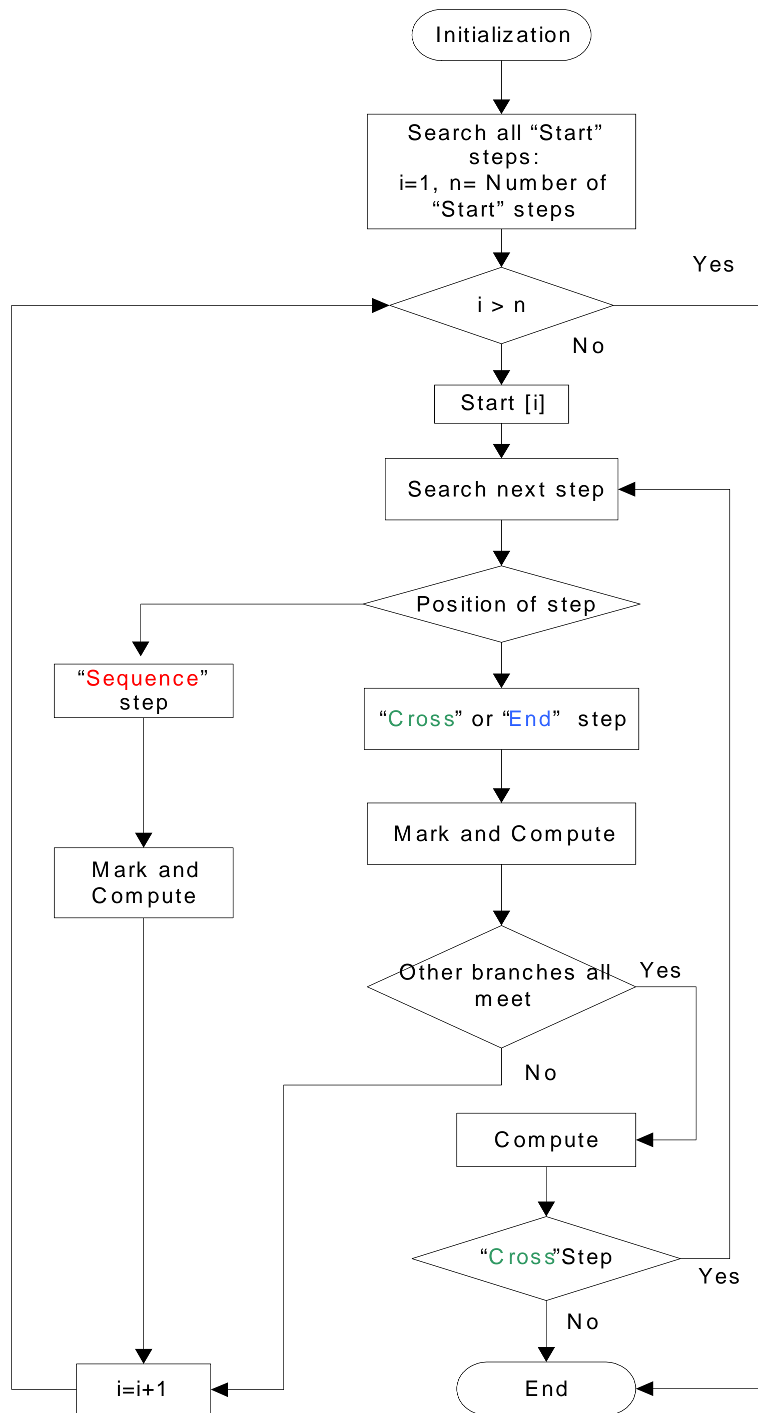
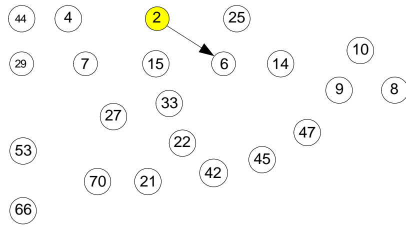


Figure 3.3: Waiting Sequential Search (WSS) algorithm flowchart.

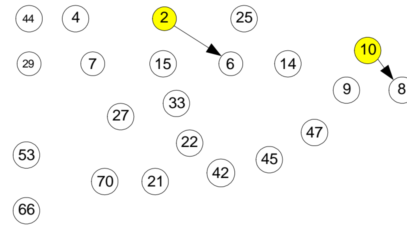
The WSS algorithms have been applied to analyze the process flow in Figure 3.1. From the Figure 3.4, it can be clearly seen that the WSS is efficient for solving the problem of searching in large-scale process flow — an especially complicated low-outdegree digraph. WSS is efficient because every step is visited only once in the entire search and there is no recurring traverse repeated in WSS, which means that all step nodes in the graph can be traversed using WSS with time complexity of O (l) (l is the number of process steps in the process flow).

The searching process of the process flow in Figure 3.1 is described below:

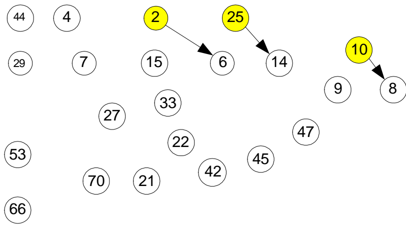
1. First, begin from the lowest number of Start steps (2), (2) → (6), check whether all the other branches have been searched, compute then wait;
2. (10) → (8), check the other branches and wait;
3. (25) → (14), wait;
4. (27) → (22), wait;
5. (29) → (7), wait;
6. (33) → (6), wait;
7. (42) → (45)→ (47) → (9), wait;
8. (44) → (4) → (7) → (15) → (6) → (14), wait
9. (53) → (70), wait;
10. (66) → (70) → (21) → (22) → (14) → (9) → (8), all the branches have met and the process ends.



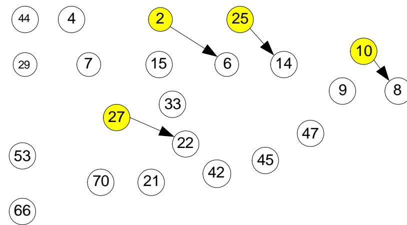
(a)



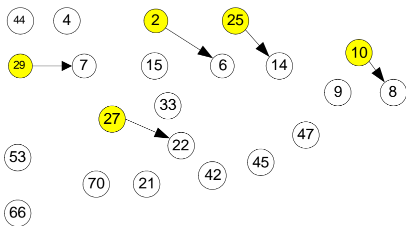
(b)



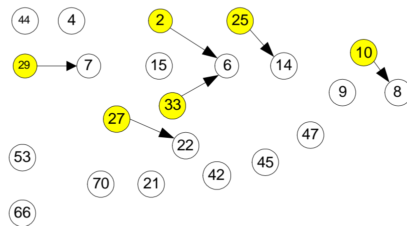
(c)



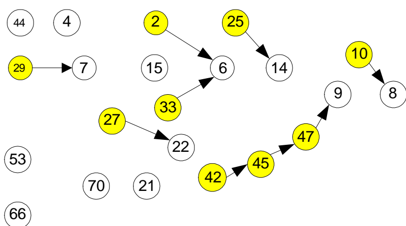
(d)



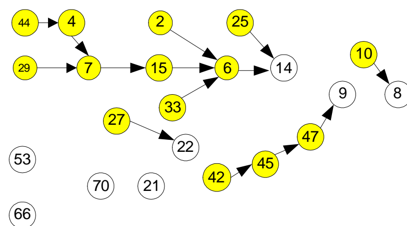
(e)



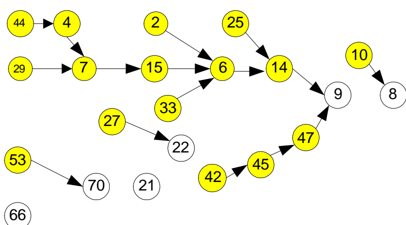
(f)



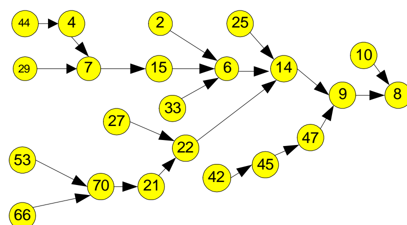
(g)



(h)



(i)



(j)

Figure 3.4: Applying a Waiting Sequential Search (WSS) to the process flow in Figure 3.1.

Table 3.2: Search index (SI) of the vertices in the process of WSS to the process flow.

v	v_2	v_4	v_6	v_7	v_8	v_9	v_{10}	v_{14}	v_{15}	v_{21}	v_{22}
SI(v)	1	11	14	12	22	21	2	20	13	18	19
V	v_{25}	v_{27}	v_{29}	v_{33}	v_{42}	v_{44}	v_{45}	v_{47}	v_{53}	v_{66}	v_{70}
SI(v)	3	4	5	6	7	10	8	9	15	16	17

The search index (SI) that represents the searching sequences is shown in Table 3.2. During the search of the process flow, the extended adjacency metrics (EAM) would be used to decide which is the successor of the vertex being visited. To form the EAM, the adjacency matrix is extended using information describing the flow between the two steps. For instance, the property of Met[16] of the flow will be recorded in the matrix after the successors have been reached. The adjacency matrix also needs to exchange data with the properties array of the process steps, in which the type, name, description, etc. are recorded for the program use. It becomes complicated to represent all the information of a more than ten-vertex graph making the use of data table[17] necessary. Data tables are also an effective way to transfer the data to the extended adjacency metrics when graphs become large. The process of data exchange is described in the Figure 3.5.

[16] The property Met is a Boolean bit used to record whether all the branches that merge into process step X have been traversed. If so, Met = True; if not Met = False.
[17] Data table describes two-dimension relationship among the data objects [55].

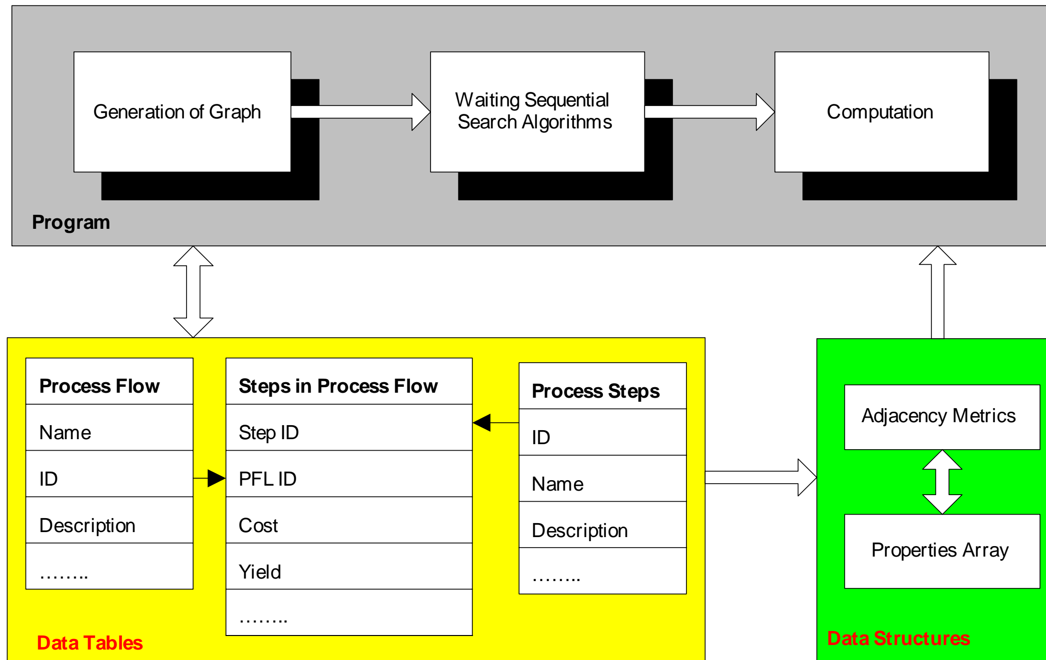


Figure 3.5: Data exchange in the implementation of WSS to the process flow.

3.5 The Multi-Variable Optimization Function

The objective of optimizing TDR location(s) and characteristics is to minimize the yielded cost [15] of the entire process flow. The yielded cost we are interested in is the final cost per product instance (after the final processing step and/or TDR operation) divided by the final product yield. This yielded cost gives a measure of the effective cost per good product instance after all the manufacturing and TDR operations are completed. The final cost per product instance and yield are determined by accumulating (sum or product) the individual process step costs and yields and the TDR operation costs and yields ((2.5) and (2.15)) in the appropriate sequence through the process.

The objective function in which feature parameters of all possible TDR operations are considered can be derived from sequential cumulative computation from the Start Steps

to the End Step. When the WSS algorithm (discussed in Chapter 3) traverses the entire process flow, a cumulative function is computed for use as the objective function that will be minimized in the optimization. The optimization problem becomes:

$$\min_{x \in X} C_Y(x_1, x_2, \cdots x_m) \tag{3.1}$$

Equation (3.1) also can be written as (3.2)

$$\min_{x \in X} \frac{C_{out}(x_1, x_2, \cdots x_m)}{Y_{out}(x_1, x_2, \cdots x_m)} \tag{3.2}$$

where,

m = number of feature parameters to be optimized;

C_Y = yielded cost of the process flow, cumulative cost (C_{out}) divided by final yield (Y_{out}).

In the optimization, first the TDR operations would be placed in all possible locations or be chosen according to expert suggestions. The fault coverage (f_{c_i}), rework yield (Y_{r_i}) and rework attempts for the ith TDR operation need to be optimized in order to minimize the total yielded cost of the process flow. For example, for the complex process flow in Figure 3.1, there are 22 (including the location after the End Step-8) possible TDR operation locations following each of the process steps if there were no specific location constraints as to where a TDR operation could or could not be placed provided by the user. If just fault coverage and rework attempts (r_a) of the TDR operations are to be optimized, the objective function can be written as (3.3),

$$\min_{f_c \in X_1, r_a \in X_2} C_Y(f_c, r_a), (X_1 \in [0,1], X_2 \in [0,10]) \tag{3.3}$$

Equation (3.1) is one possible objective function. The objective function could be changed to satisfy the specific manufacturing process requirements. For example, (3.1) can be divided into two separate objective functions if the manufacturer focuses more on the cost or the yield of products individually. In this case (3.4) is performed to minimize the cumulative cost of process and/or (3.5) is used to maximize the final yield of products.

$$\min_{x \in X} C_{out}(x_1, x_2, \cdots x_m) \tag{3.4}$$

$$\max_{x \in X} Y_{out}(x_1, x_2, \cdots x_m) \tag{3.5}$$

Other objective functions that represent some weighted combination of cost and/or yield are also possible. One could also formulate objective functions that include minimization of N_{in} or maximization of N_{out} (N_{in} and N_{out} are given by (2.11), (2.6) and (2.12)).

In the Chapter 4, Real-Coded Genetic Algorithms (RCGAs) are applied to minimize the objective function with feature parameters constrained to practical ranges. Because of the objective-independent property of Genetic Algorithms, the objective function can be modified without changing the optimization algorithm necessarily. Section 3.6 provides the data structures and pseudo code of WSS algorithms for reference.

3.6. Data Structures and Pseudo-Code for WSS Implementation

The following Pseudo Code defines the class 'process step' representing the type of vertex and 'flow between two steps' as the edge. The method 'SeachInNextStep' implements the searching all through the process flow according to Figure 3.3.

Digraph Vertex – process step

```
{
  Double Array Properties [ν]
  Integer SearchIndex [ν]
  Integer Array NumberOfSteps [ν]
}
```

Digraph Edge – flow between two steps

```
{
  Integer Array ExtendedAdjaencyMetrics [E]
}
```

Function SearchInNextStep ()

```
{
   Initialization

  {
     Inputs:
     ExtendedAdjacencyMetrics [n, n] By Reference
     Properties [n] By Reference
     Integer n NumbersofStartSteps
     Integer Array Start [n]
  }
  Integer i =0
  For i =0 to n

  {
    Integer j=0
    For j=0 to n
     {

      If  AdjacencyMetrics [Start[i], j] =1 then
       {
         Select case properties [j]. Position
                Case "Sequence"
                {
```

```
Compute

{
 Select Case Properties [j].Type
 Case "General Fab."

{
```

$$C_{out} = C_{in} + C_{step}$$

....

$$Y_{out} = Y_{in} \cdot Y_{step}$$

```
 }
 Case "Test"

 {
```

$$C_{out} = \frac{(C_{in} + C_{test})}{{Y_{in}}^{f_c}}$$

....

$$Y_{out} = {Y_{in}}^{(1-f_c)} \cdot Y_{test}$$

```
 }
 }
 Case "Cross" or "End"

  {
  Integer k =0
  For k=0 to n
  {
   Boolean AllMet = True
   If ExtendedAdjacencyMetrics [k, j] .met =0 then
   AllMet=False
   End if
  }
  Next k
  If AllMet then

  {
   Compute {Merge the branches}
   If Properties [j]. Position ="End" then
   Exit
   End if
  }
  End if
  }
  Next j
    }
    Next i

Outputs:
 ExtendedAdjacencyMetrics[n,n]
}
```

3.7 Summary

This chapter starts from a single TDR calculation model discussed in Chapter 2 and extends it to the computation of the yielded cost for the multiple TDRs process flow, in which the directed graph is used to represent it with the definitions of all types of vertices and a waiting sequential search algorithm is developed to traverse each step of the process flow with all possible TDR operations included. A multi-variable objective function to minimize the cumulated yielded cost of the process flow is proposed to be optimized using the optimization algorithm discussed in the next chapter.

Chapter 4

TDR Optimization with Real-Coded Genetic Algorithms (RCGAs)

In Chapter 3, the multi-variable objective function that needs to be optimized was obtained by applying Waiting Sequential Search (WSS) algorithms to search a process flow. The next issue is optimizing the feature parameters of the candidate TDR operations to reach the global minimum of the yielded cost of the process flow. Complex process flows with hundreds of process steps are not uncommon, [56]. A general optimization of such a process flow requires an equally large stream of TDR operations resulting in several hundred variables to be optimized for the tradeoff analysis. To optimize feature parameters of possible TDR operations in order to find the global optimum of yielded cost in the process flow, RCGAs are used. This chapter discusses the use of Real-Coded Genetic Algorithms (RCGAs) to perform the concurrent optimization of TDR locations and feature parameters.

A comprehensive comparison of optimization methods has been presented in [57]. There are many traditional deterministic algorithms available to solve optimization

problems for a local minimum. Among optimization algorithms, Gradient-based Methods (GMs) are well-known optimization methods, which probe the optimum by calculating the local gradient information [58]. There are many other algorithms available in addition to these. The above methods all require the evaluation of gradient information in order to solve problems and can only optimized a few continuous parameters [57]. Gradient evaluations can become difficult and time-consuming when complex objective functions are present. Probabilistic Global Optimization (GO) methods like simulated annealing [59] and Genetic Algorithms [60], do not require gradient information and taking cost function derivatives and have a capability of finding a global optimum, thus avoid many of the drawbacks of traditional optimization methods.

Recent work compares GO methods in terms of effectiveness (accuracy), efficiency (number of needed function evaluations) and reliability [61]. Genetic algorithms (GAs) are stochastic, directed search algorithms that have proved useful in finding global optima in both static and dynamic environments [62, 63]. GAs deal efficiently with a large number of parameters and simultaneously searches from a wide sampling of the solution range [63-65], which fits well to the characteristics of optimization problem introduced in the dissertation.[18]

4.1 Introduction to Real-Coded Genetic Algorithms

Finding a global optimum in a continuous domain is challenging for Genetic Algorithms (GAs). Traditional GAs use binary representation that evenly discretizes a real variable

[18] It is outside of the scope of the present work to determine the “best” optimization approach to use to solve the TDR location problem. Rather, demonstrating that the optimization can be done and produce a useful result is sufficient for the present effort.

space. Although Binary-Coded GAs (BCGAs) have been successfully applied to a wide range of optimization problems, they suffer from disadvantages when applied to the real-world problems involving a large number of real variables [65]. Since binary substrings representing each parameter with the desired precision are concatenated to represent an individual, the resulting string encoding a large number of design variables could have an impractical string length. For example, the string length associated with a process flow of 100 process steps is approximately 1000 (assuming a precision of three digits), see (4.1):

$$\text{Stringlength} = 100 \times \frac{3}{\log 2} = 1000 \tag{4.1}$$

GAs would perform poorly for such problems. Previous applications have avoided this problem by sacrificing precision or narrowing the search regions prior to the optimization. However, for the problem addressed in this dissertation, such approaches might exclude the region that actually has the global optimum.

Another drawback of the BCGAs applied to parameter optimization problems in continuous domains comes from discrepancy between the binary representation space and the actual problem space. For example, two points close to each other in the representation space might be far apart in the binary represented problem space. It is still an open question to construct an efficient crossover operator that is suited to such a modified problem space. A simple solution to these problems is the use of the floating-point representation of parameters [66-69]. In these real-coded GAs, an individual is coded as a vector of real numbers corresponding to the design variables. The real-coded GAs are robust, accurate, and efficient because the floating point representation is conceptually closest to the real variable space, and moreover, the string length reduces to

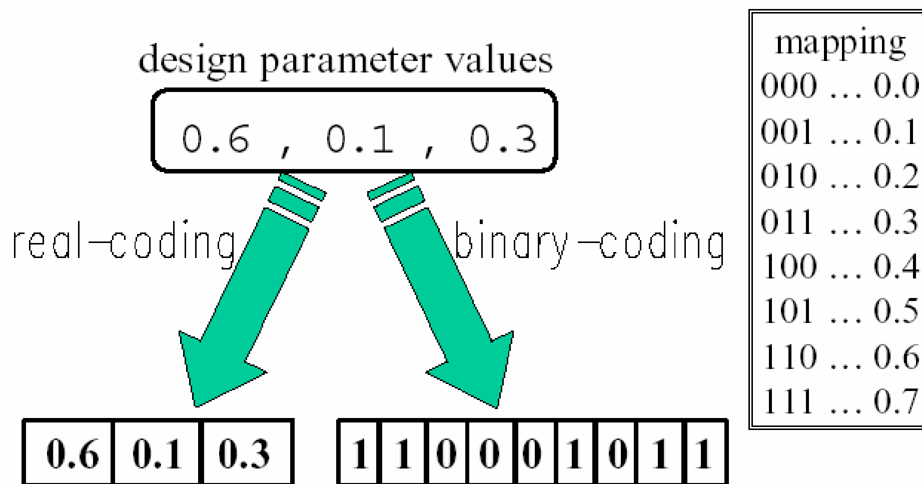


Figure 4.1: An example of binary and floating-point representations [70].

the number of variables, Figure 4.1 gives an example of representation in BCGAs and RCGAs.

In Figure 4.1, there are 3 design parameters that range from 0.0 to 0.7 in the objective function. If using binary encoding method, 9 bits (three bit for each one) in memory need to implement it. However, switching to the floating-point encoding, just 3 bits are needed. A real-coded representation can save more memory space and reduce the computing time dramatically when the objective function becomes more complex.

The objective of this chapter is to identify and apply robust and efficient GAs applicable to optimization of location(s) and characteristics of TDR operations in the process flow of electronic systems assembly. To achieve this goal, the idea of the dynamic coding is incorporated with the use of the floating-point representation. The real-coded GAs are expected to possess both advantages of the binary-coded GAs and the

floating-point representation to overcome the problems of having a large search space that requires continuous sampling.

4.2 The Application of RCGAs to the Process Flow

The following subsections focuses on the application of RCGAs to the optimization of TDR operations (see Figure 4.2). The comparisons of the binary and float-point representations of GAs are also provided to make their similarities and differences clear.

GAs have three operators: selection, crossover, and mutation. Selection is devised to inherit good-working individuals from generation to generation. If an individual has a relatively high fitness value, the chances are greater that its offspring will be present in the next generation based on a fitness-based selection rule. This operator of selection is

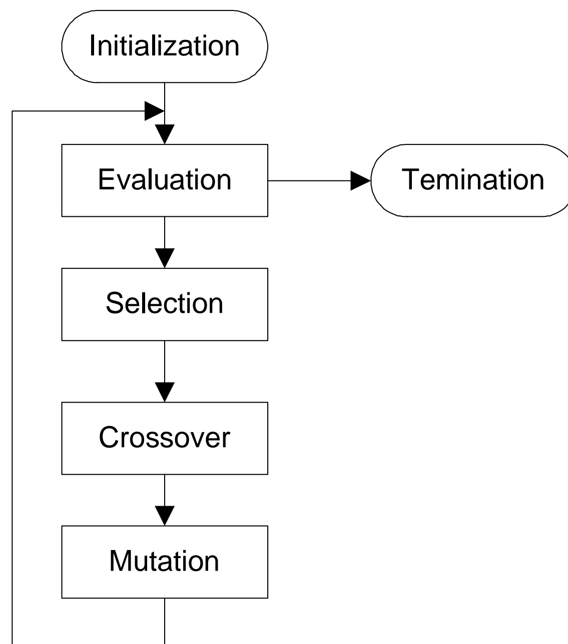


Figure 4.2: GA optimization process.

an artificial version of natural selection. In the crossover phase, two individuals in a population are randomly selected. And they exchange their bits from the crossing site determined by another random number. During the mutating process, every bit has an equal probability of being replaced by its complement number. This is an imitation of a natural crossover. Each item in Figure 4.2 will be discussed in the following sections.

4.2.1 Chromosomal Representation

The use of the floating-point representation originates in Evolutionary Programming (EP) and Evolutionary Strategies (ESs). In the floating-point representation, an individual is characterized by a vector of real numbers. It is more natural to use the floating-point representation for real parameter optimization problems because it is conceptually closest to the real space, and moreover, the string length is reduced to the number of design variables. It has been reported that real-coded Evolutionary Algorithms (EAs) outperformed binary-coded EAs in many design problems [66, 67]. Therefore, the GAs used in this dissertation adapt the floating-point representation.

An example of binary and floating-point representations is illustrated in (4.2). BCGA has chromosomes made of binary bits 0 and 1 (s_b), and RCGA has chromosomes made up of real numbers (s_r),

$$
\begin{aligned}
s_b &= (\underbrace{\overbrace{11\cdots01}^{l}}_{fc_1}\,\underbrace{\overbrace{11\cdots10}^{l}}_{fc_2}\cdots\underbrace{\overbrace{11\cdots11}^{l}}_{fc_{22}}\,\underbrace{\overbrace{01\cdots01}^{l}}_{Yr_1}\cdots\underbrace{\overbrace{01\cdots01}^{l}}_{Yr_{22}}) \\
s_r &= (\underbrace{0.923}_{fc_1}\,\underbrace{0.897}_{fc_2}\cdots\underbrace{0.988}_{Yr}\cdots\underbrace{0.954}_{Yr_{22}})
\end{aligned}
\tag{4.2}
$$

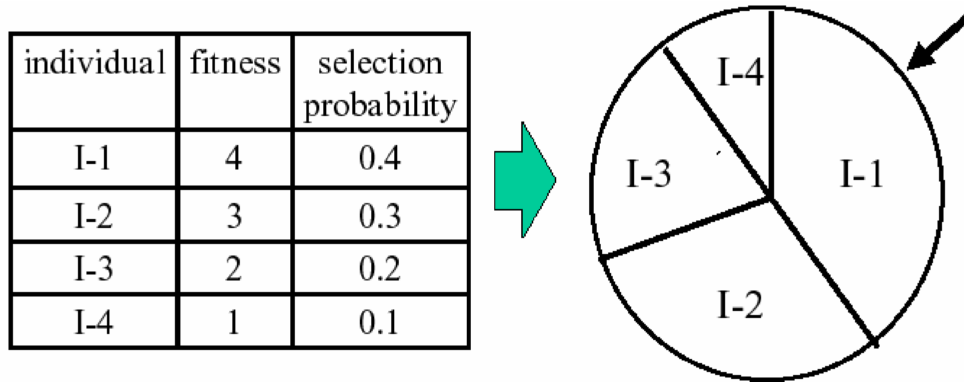
where s_b and s_r represent concatenated strings of BCGA and RCGA, respectively. For the process flow in Figure 3.1, the total number of parameters is 44.

4.2.2 Initial Population

Once a suitable representation has been decided upon for the chromosomes, it is necessary to create an initial population to serve as the starting point for the genetic algorithm. This initial population is usually created randomly. From empirical studies performed over a wide range of function optimization problems, a population size of between 30 and 100 is usually recommended, [65].

4.2.3 Selection of Mating Pairs

This phase selects pairs of individuals from the mating pool that will produce offspring for the next generation. The commonly used approach is to assign each individual a probability of selection on the basis of its fitness. Goldberg and Richardson also suggest selection according to fitness modified by sharing function for the maintenance of diversity in the population [71]. A widely used method is the fitness-proportional selection [64]. In this method, the selection probability of each individual is calculated by dividing its fitness by the sum of the fitness of all individuals. Then, the parents are selected by either roulette-wheel selection [64] or Stochastic Universal Sampling (SUS) [72]. Figure 4.3 shows the operation of the roulette-wheel selection that assigns a portion of the wheel proportional to the selection probability and starts spinning the roulette wheel: each time, a single individual is selected.



individual	fitness	selection probability
I-1	4	0.4
I-2	3	0.3
I-3	2	0.2
I-4	1	0.1

Figure 4.3: Roulette-wheel selection.

4.2.4 Crossover

Crossover is a process in which new individuals are generated by exchanging features of the selected parents with the intent of improving the fitness of the next generation. These new individuals are then subjected to mutation. There are a number of different ways in which the recombination operation can be implemented. The conventional crossover operators include one-point, two-point, multi-point, and uniform crossovers. In the one-point crossover for the bit-string representation, the bits are swapped between the two parents in segments at a random point of a chromosome in between the bits, [64]. In the floating-point representation, the vector components of two parents are swapped in groups at a random space of a vector between the vector components. For example, given a five dimensional space, *Parent1* has a vector (x1, x2, x3, x4, x5) and *Parent2* has a vector (y1, y2, y3, y4, y5) and the crossover point was selected to be 3, then the offspring will carry vectors (x1, x2, x3 y4, y5,) and (y1, y2, y3, x4, x5). An example of one-point crossover is illustrated in Figure 4.4. Two-point, multi-point, and uniform crossovers for the floating-point implementation can be defined in the same manner.

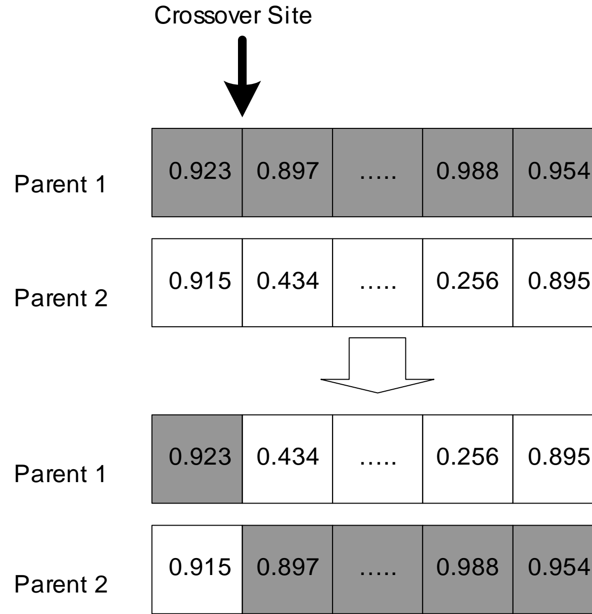


Figure 4.4: One-point crossover for floating-point representation.

4.2.5 Mutation

The function of mutation is to keep diversity of a population and promote the searching in the solution space that cannot be represented by the strings of the present population. One of the most common mutation methods in real-coded GAs is uniform mutation by which a selected real-valued gene is transformed randomly between its upper and lower bounds. The principle of uniform mutation is shown in (4.3),

$$\begin{aligned}&(fc_1 \cdots fc_i \cdots fc_{22}) \Rightarrow (fc_1 \cdots \overline{f}c_i \cdots fc_{22}),\\&\overline{f}c_i = \lambda(fc_i{}^U - fc_i{}^L) + fc_i{}^L\end{aligned} \tag{4.3}$$

where $\overline{f}_{c_i} \in [f_{c_i}^U, f_{c_i}^L]$ is a mutated ith gene and $\lambda \in [0,1]$ is a random number.

4.2.6 Elitist Strategy

Since evolution in GAs depends on stochastic operators, GAs do not guarantee a monotonic improvement in objective function value of the design unless deterministic overlapping systems are used. To ensure a monotonic improvement, De Jong [73] proposed so-called elitist strategy, in which some of the best individuals are copied into the next generation without applying any evolutionary operators. GAs incorporating non-overlapping system usually adopt this strategy.

With RCGAs integrated, the framework shown in Figure 4.5 has been developed aimed at providing intelligent solutions to the placement of TDR operations in process

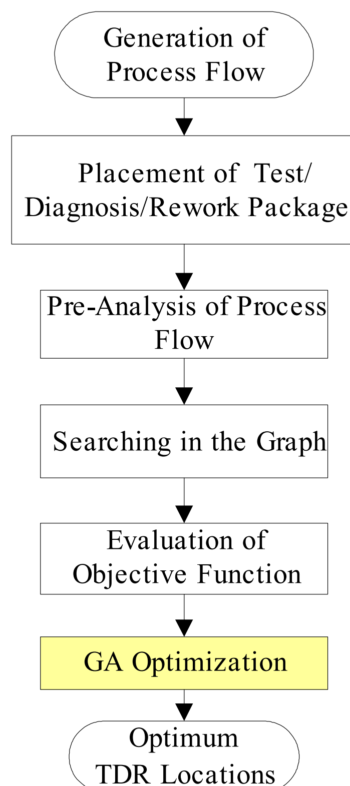


Figure 4.5: The framework of process flow cost optimization modeling system.

flow. The system gives suggestions on where the TDR operations should be placed or removed and the optimal values of feature parameters of TDR operations based on the optimization of a yielded cost objective function. The following sections provide a verification of the optimization methodology by comparison with functional computation in simple cases and general linear and nonlinear (multi-branch) process flow demonstrations.

4.3 Verification of the Optimization Model

To provide testing and demonstration of the approach, the following test cases are used:

- Test Case 1: Single test case, fault coverage varies and no rework is included;
- Test Case 2: One-TDR operation case in which fault coverage and rework yield are allowed to vary (a maximum of one rework attempt assumed);
- Test Case 3: A general linear process flow case (no branches), fault coverage and rework yield are allowed to vary (a maximum of one rework attempt assumed);
- Test Case 4: A general process flow case with multiple branches and variable fault coverages and rework yields (a maximum of one rework attempt assumed).

Test Cases 1 and 2 are trivial and used to validate the optimization model by comparison with closed-form calculations of the yielded cost associated with the process flow. The cases are categorized into high-yield and low-yield inputs in order to reflect the impacts of input yield on the optimum solution. Test Case 3 provides demonstrations of

the optimization of TDR characteristics in general linear process flow. Test Case 4 is a general process flow with multiple branches and demonstrates the value of the optimization methodology on a case where the optimum placement and characteristics of the TDR operations is not obvious.

4.3.1 Validation from the Simple Cases

For the simple process flow in Figure 4.6, the objective function can be formed by step computation. The corresponding optimization is presented for each case:

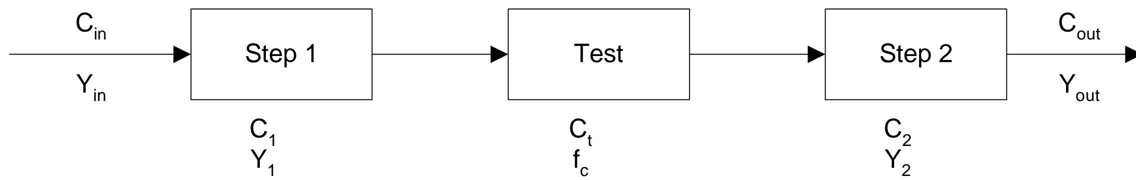


Figure 4.6: Simple case with just one test step.

Table 4.1: Values of input parameters

	C_{in}	Y_{in}	C_1	Y_1	C_2	Y_2
High-Yield	21	0.97	10	0.98	7	0.93
Low-Yield	21	0.48	10	0.32	7	0.93

If there is no test operation in the process flow shown in Figure 4.6, the C_{out} and Y_{out} are computed separately using (4.4) and (4.5) respectively,

$$C_{out} = C_{in} + C_1 + C_2 \tag{4.4}$$

$$Y_{out} = Y_{in} Y_1 Y_2 \tag{4.5}$$

the yielded cost is derived in (4.6)

$$C_Y = \frac{C_{out}}{Y_{out}} = \frac{\left(C_{in} + C_1 + C_2\right)}{Y_{in} Y_1 Y_2} \tag{4.6}$$

To evaluate the expression, the input parameters in Table 4.1 are used. Substituting the inputs of Table 4.1 into (4.6), the yielded cost is given as (4.7),[19]

$$C_Y = \begin{cases} 266.02 & \text{(low - yield)} \\ 41.64 & \text{(high - yield)} \end{cases} \tag{4.7}$$

4.3.1.1 One-Test Case (Test Case 1):

If one test step is inserted between Step 1 and Step 2 in Figure 4.5, the cost and yield of the process flow gives by (4.8) and (4.9):

$$C_{out} = \frac{(C_{in} + C_1) + C_{test}}{(Y_{in} Y_1)^{f_c}} + C_2 \tag{4.8}$$

$$Y_{out} = (Y_{in} Y_1)^{(1-f_c)} Y_2 \tag{4.9}$$

which results in a yielded cost of ,

$$C_Y = \frac{C_{out}}{Y_{out}} = \frac{\left(C_{in} + C_1 + C_{test}\right) + C_2 (Y_{in} Y_1)^{f_c}}{Y_{in} Y_1 Y_2} \tag{4.10}$$

substituting (2.17) into (4.10), the yielded cost of the process flow becomes,

---

[19] Equation (4.7) is used to compare with the optimum obtained by the RCGAs applied in Test Case 1 and Test Case 2.

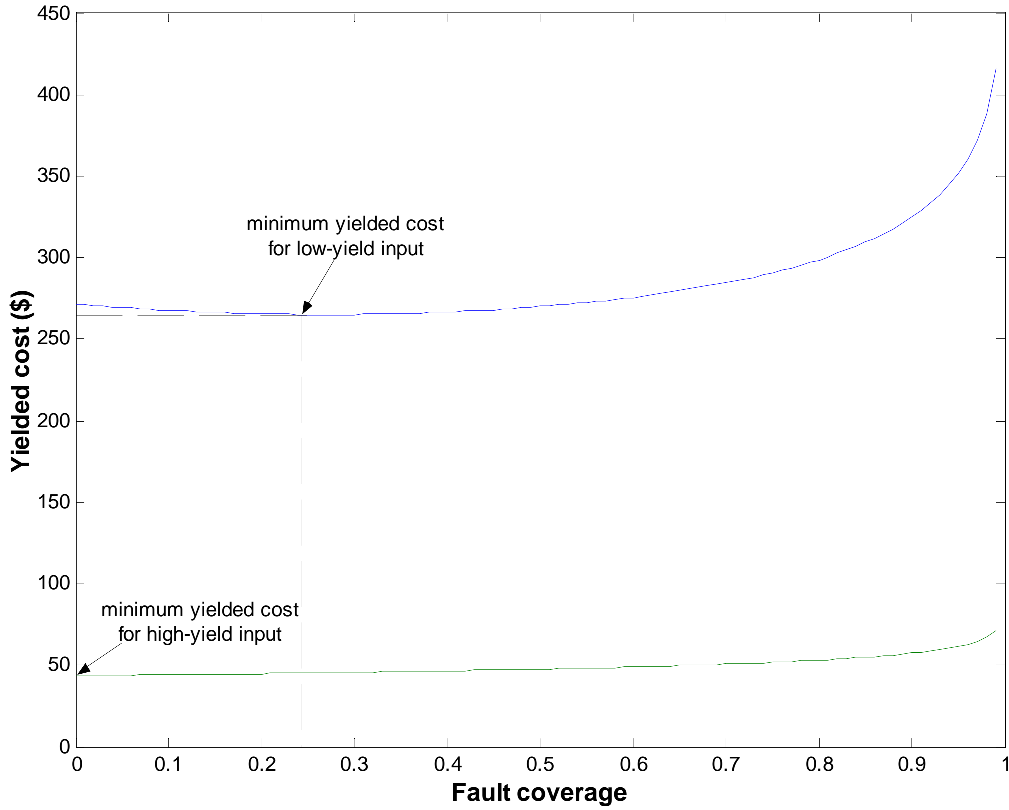


Figure 4.7: Yielded cost of process flow in Figure 4.6 versus fault coverage of test.

$$C_Y = \frac{(C_{in} + C_1 + C_{ft} - r_t p_t) + b_t p_t \ln(1 - f_c) + C_2 (Y_{in} Y_1)^{f_c}}{Y_{in} Y_1 Y_2} \quad (4.11)$$

Equation (4.11) can be written as:

$$C_Y = A + B \ln(1 - f_c) + C D^{f_c} \quad (4.12)$$

where $A = \frac{C_{in} + C_1 + C_{ft} - r_t p_t}{Y_{in} Y_1 Y_2}$, $B = \frac{b_t p_t}{Y_{in} Y_1 Y_2}$, $C = \frac{C_2}{Y_{in} Y_1 Y_2}$, $D = Y_{in} Y_1$.

Figure 4.7 plots (4.12) for the high-yield and low-yield cases define in Table 4.1. The following conclusions can be drawn from Figure 4.7:

1. For high-yield inputs, the yielded cost of the one-test case primarily depends on the change of $\ln(1-f_c)$ instead of the power of the fault coverage (i.e., D in (4.12) is very small). The minimum of yielded cost should be at the point when the fault coverage is zero. This is to say that the minimum yielded cost is for no test inserted into the process flow, which makes intuitive sense because the input to the process flow is high yield.

2. For low-yield inputs, we can see that (4.12) clearly differs from the characteristic of curve of yielded cost at high-yield inputs due to increasing importance of the power of the fault coverage in determining the yielded cost of the product resulting from the process flow. The minimum of yielded cost should be between $260 and $280 while the optimal fault coverage falls into the range from 0.2 to 0.3.

Verification for High-Yield Inputs

The results of the simple calculation above have been used to check the optimization methodology developed in this proposed research. RCGAs are applied to cases of high-yield and low-yield inputs respectively. The specification of RCGAs is summarized in the following:

- Maximum generation= 20,
- Population size= 50,
- Mutation probability= 0.1,
- Crossover probability= 0.95,
- f_T = 10% (fault coverage threshold),
- Termination criteria = maximum generations.

The threshold of fault coverage (f_T) has been defined in Chapter 2 as the minimum nonzero fault coverage that we can practically purchase. For any fault coverage below this threshold, there is considered to be no test present (zero fault coverage and zero test cost).[20] In the optimization model, the algorithm decides whether the value of fault coverage associated with the best fitness for a particular TDR is below the threshold in each generation. If it is, the test step is discarded, which means there is no need for a test at the particular location and also the corresponding diagnosis/rework operation will obviously not be present. The threshold of fault coverage is set to 10% in the analyses performed in this dissertation.

The results from the optimization of yielded cost in the single-test case at high-yield inputs are shown in Figure 4.8. In the first generation, the RCGAs have found the optimized solution and show the best fitness (the best value of objective function in one population) of 41.64 when the corresponding fault coverage is set at a very small value. After the evolution of 20 generations (termination criteria), the algorithm converges to the same value of yielded cost, which agrees with (4.7). In the last generation, the fault coverage optimization terminates at a small amount of fault coverage (0.015), lower than the fault coverage threshold for testing to be present. The test is thus removed from the process flow based on the optimization analysis from the RCGAs, which coincides with the characteristics of Figure 4.7 (high-yield relation) and (4.7) for high-yield input, where no test step is considered. In addition to validating the optimal yielded cost of the process flow, RCGAs are continuously convergent in searching for the optimum, see Figure 4.8,

[20] Effectively, the model for test cost versus fault coverage has a "step" in it at the threshold fault coverage, i.e., there is a minimum investment in the test operation to obtain any fault coverage.

and the mean (mean value of objective function in one population) of yielded cost levels out as well.

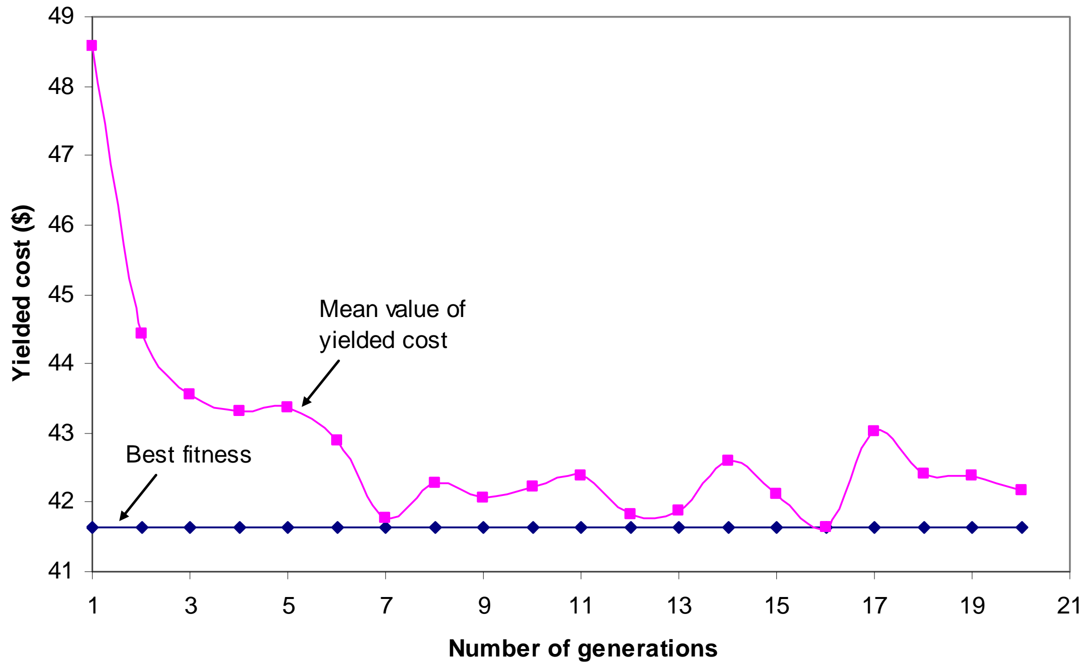


Figure 4.8: Optimization of yielded cost in the single-test case (Test Case 1) at the high-yield level.

Verification for Low-Yield Inputs

The specification of RCGAs used in the low-yield input scenario is identical to that applied in the high-yield case except the maximum number of generations has been increased to 50. The optimal solution of $265.1 and a fault coverage of 0.27 that agrees with the observation from Figure 4.7 (low-yield relation) and it is smaller than $266.02 — the result computed in (4.7) for low-yield input with no testing, which shows the yielded cost in no-test case is not the minimum.

4.3.1.2 One-TDR case (Test Case 2)

In this test case, a complete TDR operation (including rework) is placed between the two process steps in Figure 4.9 in order to trace the effect of rework yield and fault coverage on yielded cost. Optimizations of one-TDR case with feature parameters at high-yield and low-yield inputs in Table 4.1 are given in Figure 4.10 and 4.11. The optimal values of fault coverage and rework yield validate the same conclusion as Test Case 1 that there is no need for a TDR operation for high-yield inputs. In Table 4.2, the optimum of yielded cost reaches $41.64 (same with the optimal value in single-test case and no-test case) in 50 generations when the fault coverage converges to a very small amount (smaller than f_T), Figure 4.10, and the rework yield levels out without convergence in Figure 4.11, i.e., the rework is useless and should be also removed with the test operation for the high-yield inputs.

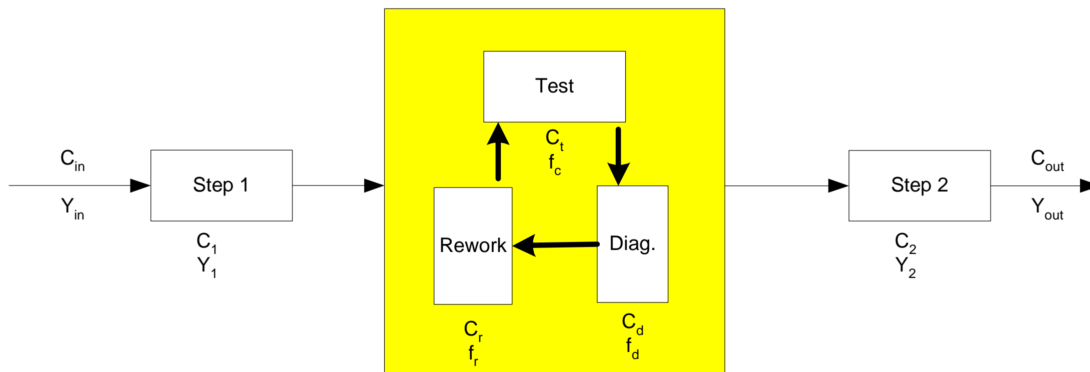


Figure 4.9: Process flow with one Test/Diagnosis/Rework operation (Test Case 2).

For the low-yield inputs, the rework operation plays an important role on improving the yield of process flow. Table 4.3 shows an optimal yielded cost of $99.29, which is much lower than the value we have computed in Section 4.3.1 when the no rework operation was considered.

Simple cases (Test Cases 1 and 2) validate the optimization model addressed in this dissertation with results that agree with the close-form computation. The remaining sections in this chapter exercise the optimization algorithms for more general sequential and branched process flows.

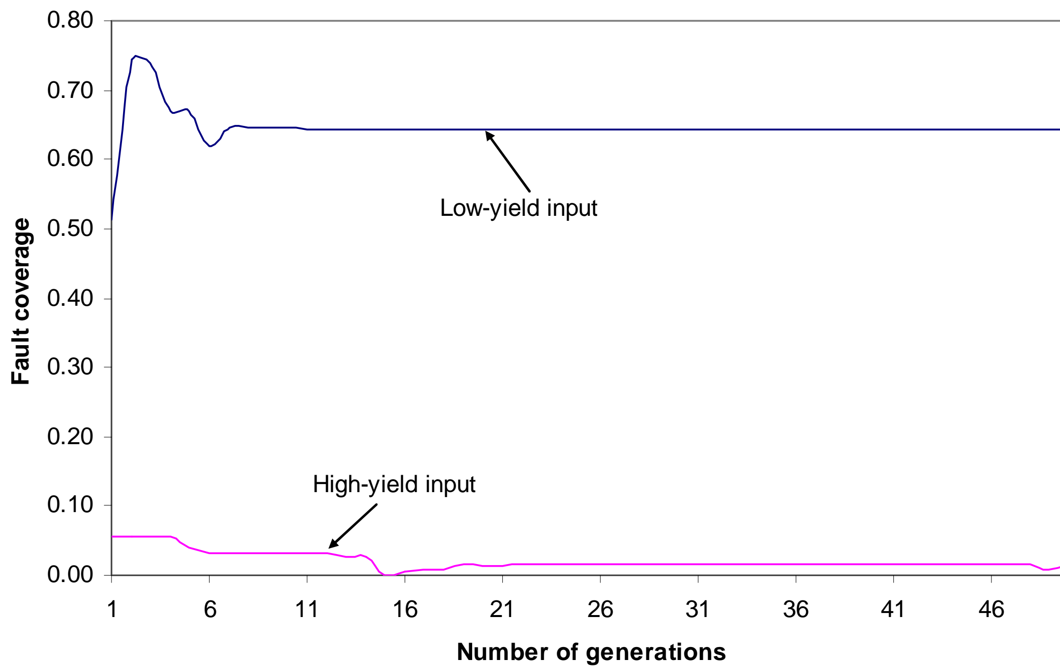


Figure 4.10: Optimization histories of fault coverage in one-TDR case (Test Case 2).

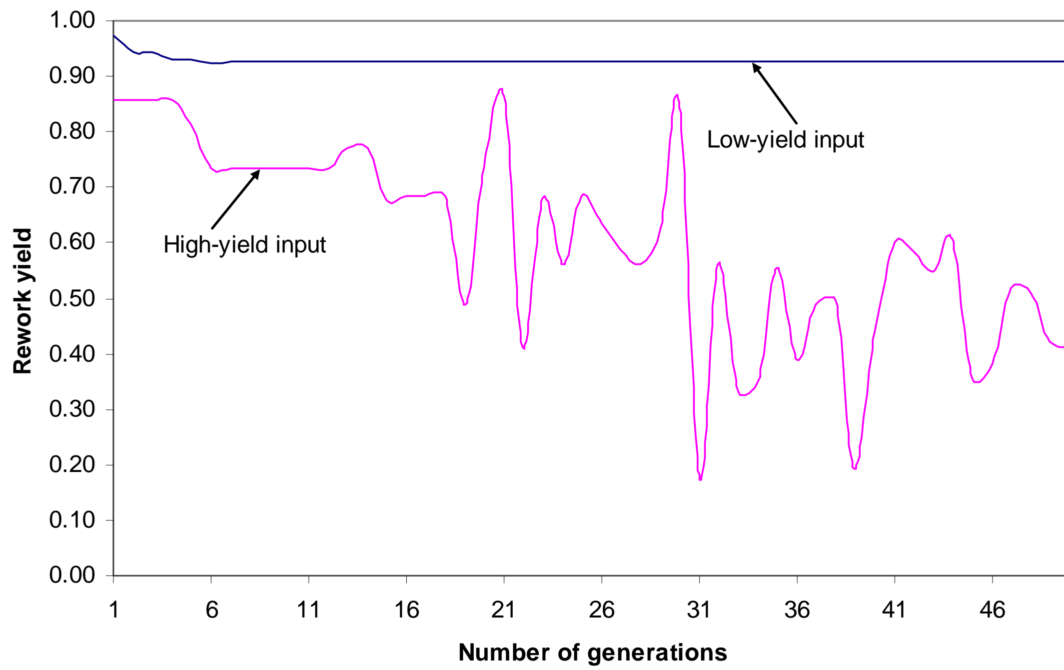


Figure 4.11: Optimization histories of rework yield in one-TDR case (Test Case 2).

Table 4.2: Results of RCGAs applied in the one-TDR case at high-yield inputs (Test Case 2).

Gen.	Best Fitness (Yielded Cost $)	Mean (Yielded Cost $)	Fault Coverage	Rework Yield
1	44.26	51.64	0.06	0.86
2	44.26	46.59	0.06	0.86
3	44.26	45.59	0.06	0.86
4	44.26	45.23	0.06	0.86
5	44.16	44.46	0.04	0.82
6	44.12	45.33	0.03	0.73
7	44.12	44.57	0.03	0.73
8	44.12	44.91	0.03	0.73
9	44.12	45.08	0.03	0.73
10	44.12	45.40	0.03	0.73
11	44.12	45.03	0.03	0.73
12	44.12	44.98	0.03	0.73
13	44.09	44.79	0.03	0.77
14	44.09	44.80	0.03	0.77
15	42.98	44.33	0.00	0.68
16	42.98	44.37	0.00	0.68
17	42.98	44.59	0.01	0.69
18	42.98	43.84	0.01	0.69
19	42.98	44.51	0.02	0.49
20	42.98	43.53	0.01	0.74
21	42.98	43.19	0.01	0.86
22	42.98	43.37	0.02	0.41
23	42.98	43.98	0.02	0.68
24	42.98	43.67	0.02	0.56
25	42.98	43.25	0.02	0.69
26	42.98	44.04	0.02	0.64
⋮	⋮	⋮	⋮	⋮
45	42.98	43.58	0.02	0.35
46	42.98	43.51	0.02	0.38
47	42.98	43.77	0.02	0.52
48	42.98	43.80	0.02	0.51
49	42.98	43.78	0.01	0.42
50	42.98	43.87	0.02	0.41

Table 4.3: Results of RCGAs applied in the one-TDR case at low-yield inputs (Test Case 2).

Gen.	Best Fitness (Yielded Cost $)	Mean (Yielded Cost $)	Fault Coverage	Rework Yield
1	103.04	184.14	0.51	0.97
2	100.66	129.32	0.75	0.94
3	100.51	111.23	0.74	0.94
4	99.39	108.15	0.67	0.93
5	99.39	112.03	0.67	0.93
6	99.37	101.50	0.62	0.92
7	99.29	107.25	0.65	0.93
8	99.29	101.83	0.65	0.93
9	99.29	103.28	0.65	0.93
10	99.29	103.32	0.65	0.93
11	99.29	103.71	0.64	0.93
12	99.29	111.08	0.64	0.93
13	99.29	106.81	0.64	0.93
14	99.29	109.36	0.64	0.93
15	99.29	101.26	0.64	0.93
16	99.29	102.08	0.64	0.93
17	99.29	105.05	0.64	0.93
18	99.29	110.84	0.64	0.93
19	99.29	104.28	0.64	0.93
20	99.29	114.62	0.64	0.93
21	99.29	104.04	0.64	0.93
22	99.29	105.79	0.64	0.93
23	99.29	105.68	0.64	0.93
24	99.29	103.67	0.64	0.93
25	99.29	112.96	0.64	0.93
26	99.29	105.75	0.64	0.93
⋮	⋮	⋮	⋮	⋮
45	99.29	104.58	0.64	0.93
46	99.29	102.31	0.64	0.93
47	99.29	110.46	0.64	0.93
48	99.29	110.29	0.64	0.93
49	99.29	110.94	0.64	0.93
50	99.29	111.38	0.64	0.93

4.3.2 Optimization in a General Process Flow

Section 4.3.1 presented validation of the optimization analysis for trivial cases. Next we will apply the optimization analysis to complex process flows with multiple TDR operations to assess the feasibility of the algorithms.

4.3.2.1 Optimization in a Sequential Process Flow (Test Case 3)

An example of linear process flow (no branches) in which the input yield determines the final yield and yielded cost of process flow is shown in Figure 4.12 and Table 4.4 gives the corresponding process step properties (cost and yield); the characteristics of the TDR operations 1-5 are solved for in the optimization process.

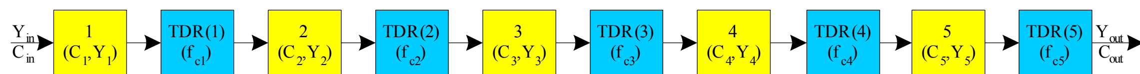


Figure 4.12: linear process flow (all possible TDR locations are shown).

Table 4.4: Characteristics of the process steps in Figure 4.12.

Process Step	Cost ($)	Yield (fraction)
Input to 1	42	0.95
1	12	0.95
2	31	0.92
3	25	0.945
4	47	0.932
5	28	0.93

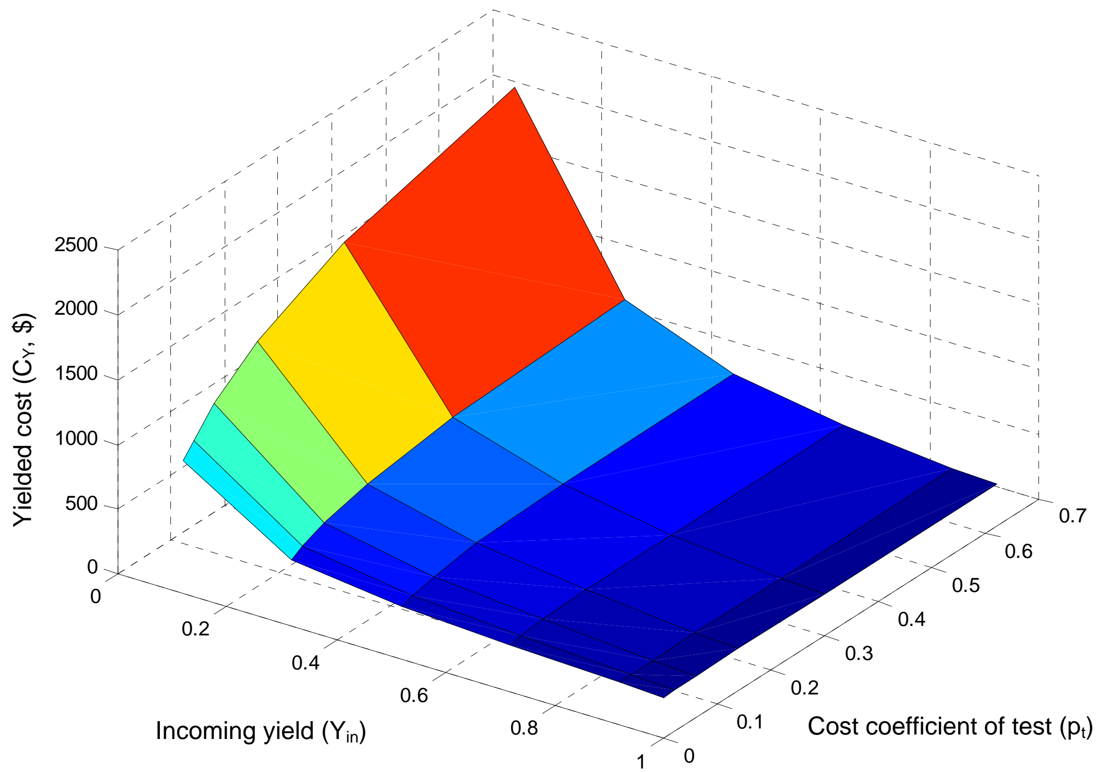


Figure 4.13: Computed optimum yielded cost of process flow in Figure 4.12.

Figure 4.13 gives the computed optimum yielded cost (with the corresponding individual cost in Figure 4.14 and yield of the process flow in Figure 4.15) for the linear process flow in Figure 4.12 generated using the optimization algorithm. Figure 4.16-4.21 (series f_{c1} to f_{c5} in these figures represent the optimum fault coverages of five possible TDR operations in Figure 4.12) present the computed optimum fault coverages for the test cost coefficient set to various values. When the cost of test is very inexpensive (at a small value of p_t in (2.17)) in Figure 4.16 and 4.17, the optimum prefers higher fault coverages, otherwise all the tests are assigned the lower fault coverages and would be removed from the process flow if they are below the threshold. In Figure 4.20 and 4.21 most of the TDR operations stay at the very low fault coverages for the expensive test

cost but the tests are still needed to ensure that the optimum yielded cost of the process flow when the incoming yield is very low.

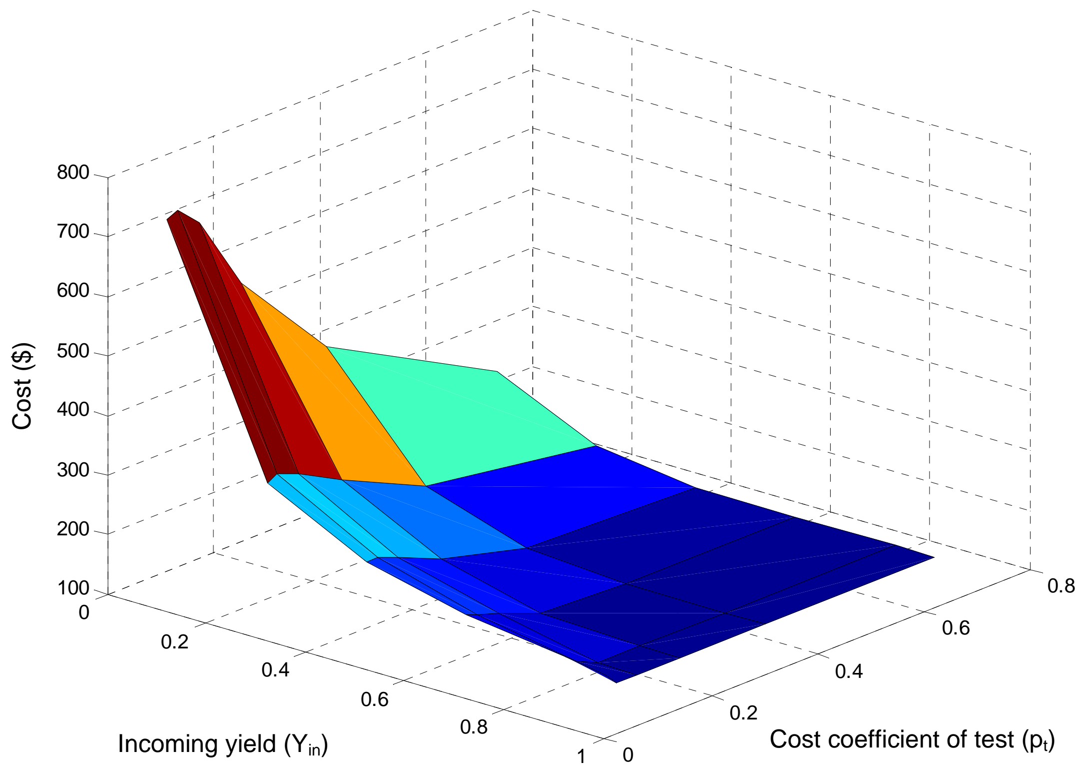


Figure 4.14: Computed cost of process flow in Figure 4.12.

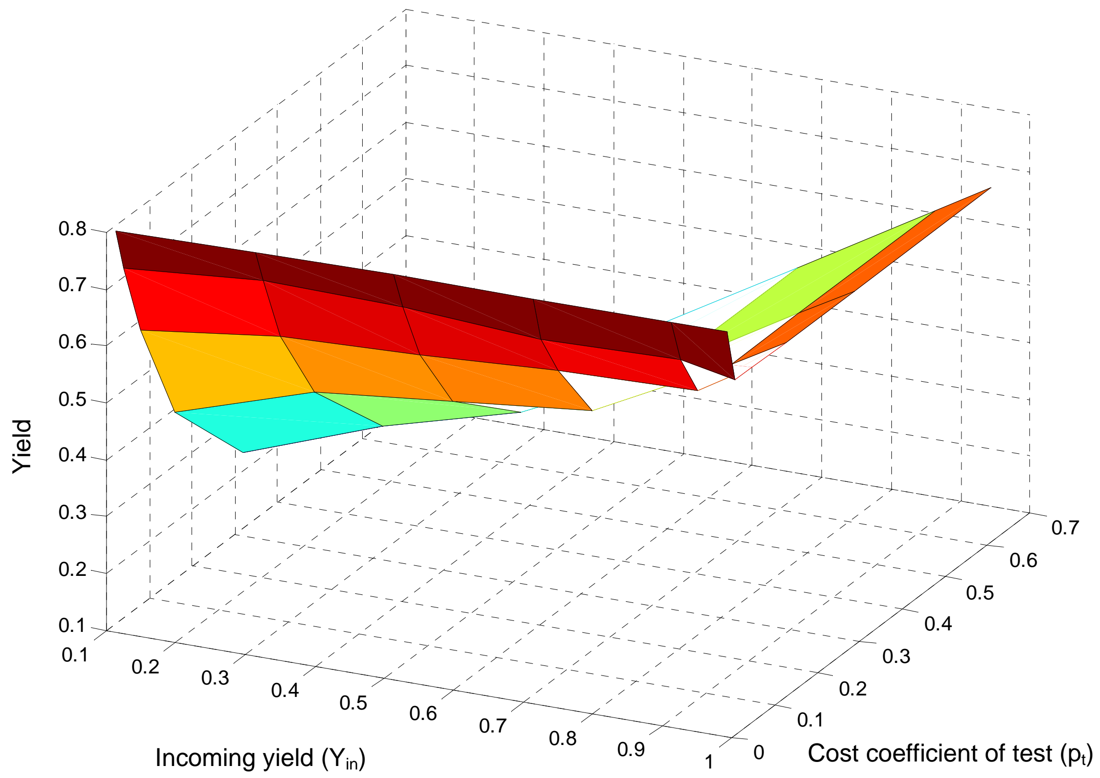


Figure 4.15: Computed yield of process flow in Figure 4.12.

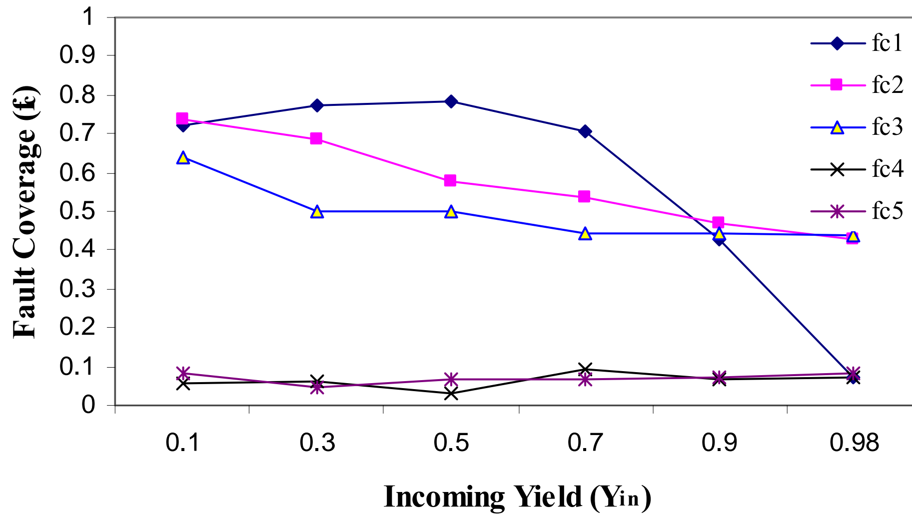


Figure 4.16: Computed optimum fault coverage for TDR Operations when test cost coefficient (p_t) is 0.02 in the process flow shown in Figure 4.12.

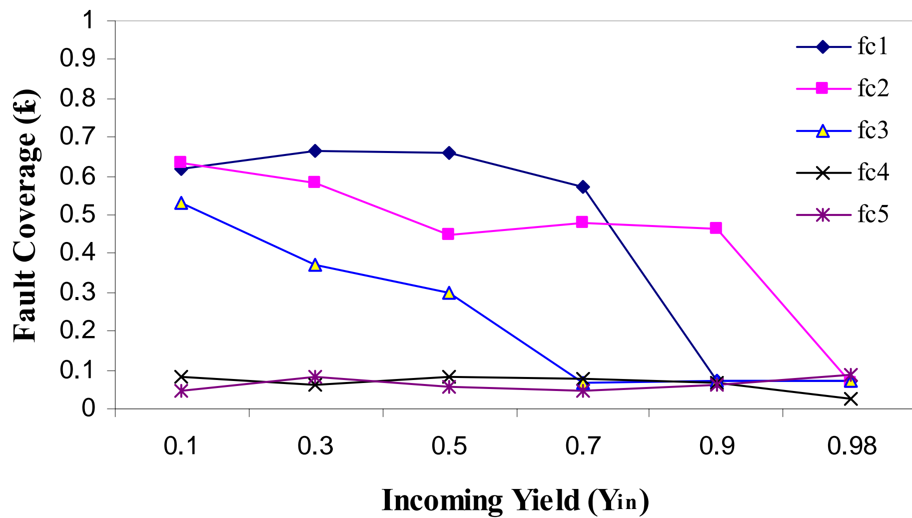


Figure 4.17: Computed optimum fault coverage for TDR Operations when test cost coefficient (pt) is 0.04 in the process flow shown in Figure 4.12.

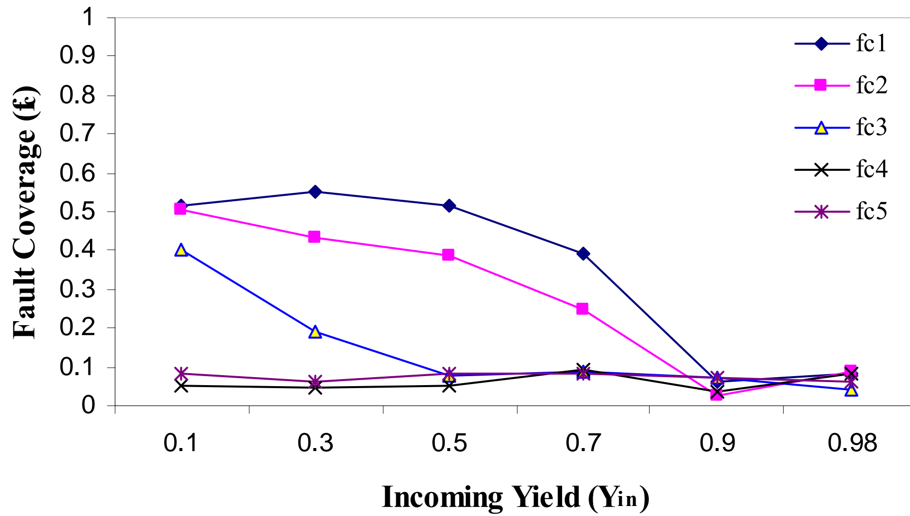


Figure 4.18: Computed optimum fault coverage for TDR Operations when test cost coefficient (p_t) is 0.08 in the process flow shown in Figure 4.12.

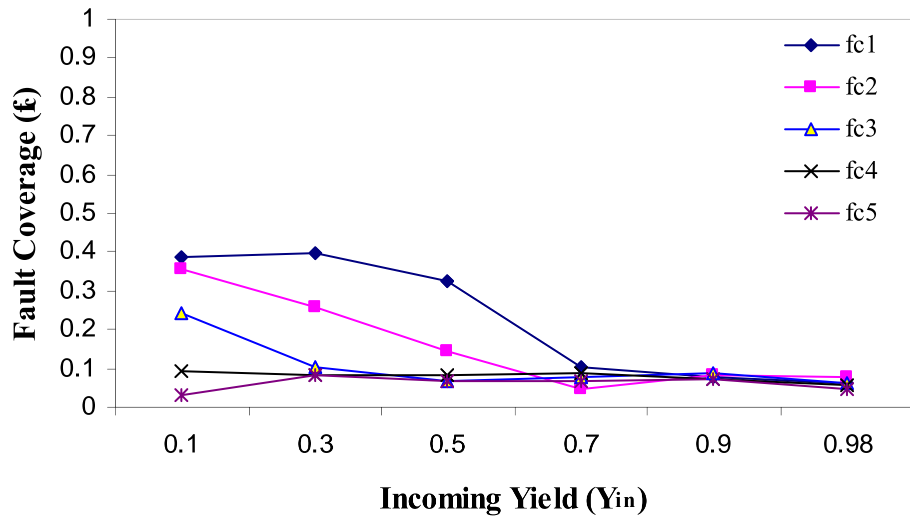


Figure 4.19: Computed optimum fault coverage for TDR Operations when test cost coefficient (p_t) is 0.16 in the process flow shown in Figure 4.12.

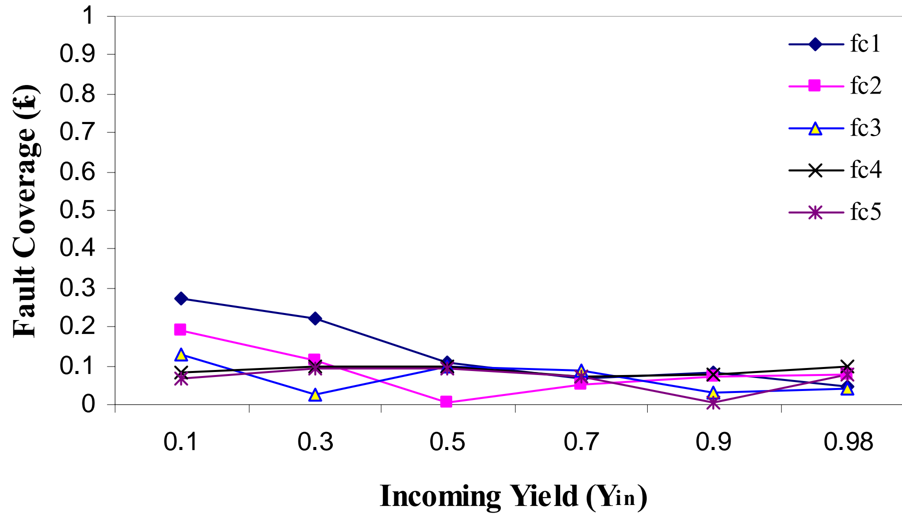


Figure 4.20: Computed optimum fault coverage for TDR Operations when test cost coefficient (p_t) is 0.32 in the process flow shown in Figure 4.12.

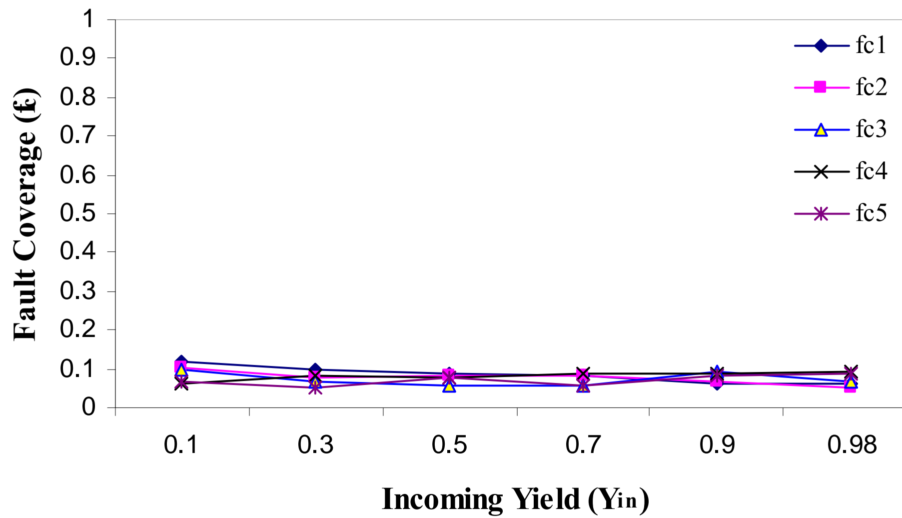


Figure 4.21: Computed optimum fault coverage for TDR Operations when test cost coefficient (p_t) is 0.64 in the process flow shown in Figure 4.12.

4.3.2.2 Optimization in a Branched Process Flow (Test Case 4)

Figure 4.22 shows a complex process flow with all possible TDR locations marked in it. Table 4.5 shows the corresponding process step properties (cost and yield); the characteristics of the TDR operations 14-25 are solved for in the optimization process. The cost and yield associated with the individual process steps is example data that is representative of high-end electronic system assembly. As an example comparison analysis, the optimum fault coverage of all possible TDR operations for various fixed cost values of test and rework[21] are shown from Figure 4.23 to 4.26. The test and rework step properties are given in Table 4.5 except for the values C_{ft} and C_{fr}, which are varied as indicated in corresponding Figures.

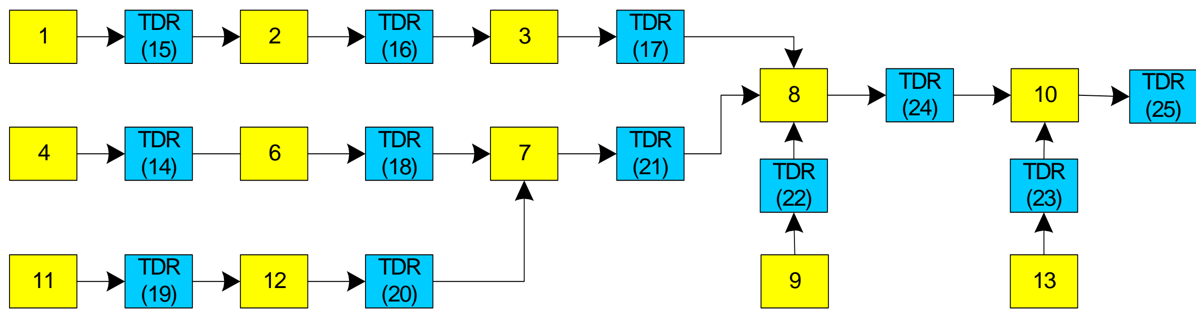


Figure 4.22: A complex process flow with all possible TDR operations shown.

[21] There is no functional relationship between rework cost and rework yield applied in the optimization of process flow in Figure 4.22. But the rework operation is still assumed a fixed cost that will vary for each case.

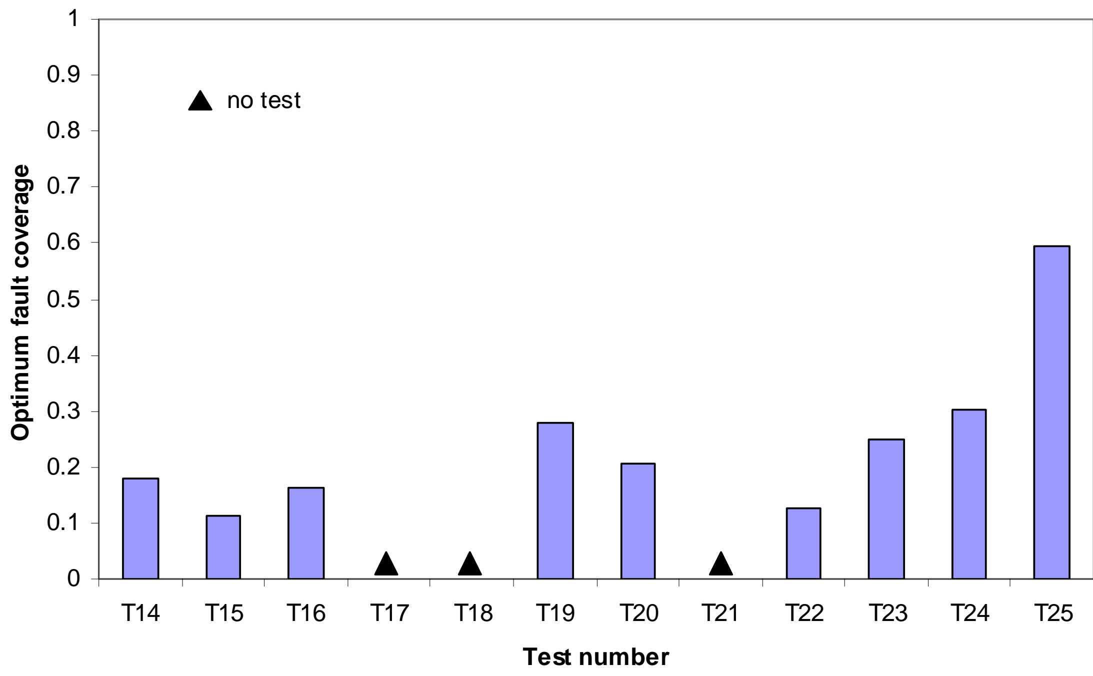
Table 4.5: Characteristics of the process steps in Figure 4.22 (note, there is no step 5)

Process Step	Cost ($)	Yield (fraction)
Input to 1	41	0.91
Input to 4	60	0.36
Input to 9	37	0.42
Input to 11	9	0.95
Input to 13	75	0.96
1	21.1	0.95
2	12.3	0.88
3	14	0.89
4	23.8	0.94
6	45	0.96
7	11.3	0.86
8	33.8	0.92
9	13	0.9
10	15	0.92
11	60	0.95
12	78	0.91
13	54	0.94

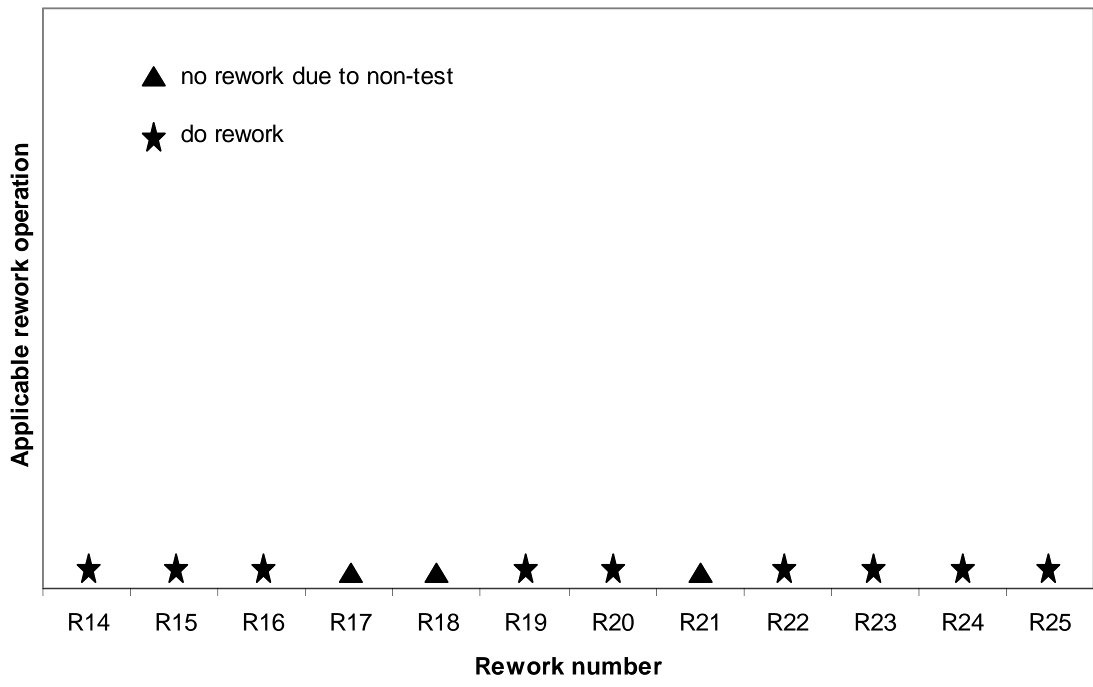
The algorithm begins by automatically placing TDR operations in all possible locations: 14-25 in Figure 4.22. The algorithm then determines the fault coverage and rework characteristics of each TDR that minimize the final yielded cost. Figure 4.23 — 4.26 shows that results from the optimization for different assumptions about the fixed costs associated with the test and rework.[22] Figure 4.23 has low (inexpensive) test and rework — C_{ft} and C_{fr} are both small; as a result, 9 of the 12 possible locations for tests are present (i.e., have fault coverages above the threshold for testing, which is a fault coverage of 0.1). Because rework is also inexpensive in this case, rework is being done at all the actual test locations. The Figure 4.23 result is intuitive, if test and rework are inexpensive, then test and rework will be done after nearly every process step. Figure

[22] Various fixed test cost used in (2.17) assumed, other parameters given in Table 2.2.

4.24 has inexpensive testing (same as the case in Figure 4.23), but the rework is expensive. As a result, significantly fewer rework opportunities are actually exercised. Notice also that the optimum test locations (and fault coverages) change due to the inclusion or exclusion of rework possibilities, even though the characteristics of the testing are the same. Figure 4.25 shows the same test costs as 4.23 and 4.24, but no rework is allowed – the optimum test locations and fault coverages again differ from cases 4.23 and 4.24 and the average value of fault coverage increases to compensate for the loss of rework capabilities (the optimization algorithm is attempting to maximize the final yield in order to minimize cost divided by yield). Figure 4.26 has expensive test and expensive rework. As a result, only 4 of 12 possible test locations are used (Test 17, 23, 24 and 25 only), however, rework is included with all three of these tests. In this case, the majority of the testing is focused on test operations near the end of the process indicating that if test is expensive and defects introduced in the process are spread over the entire process (as opposed to being focused on a single process step), the optimum test location is near the end of the process, which is intuitive.

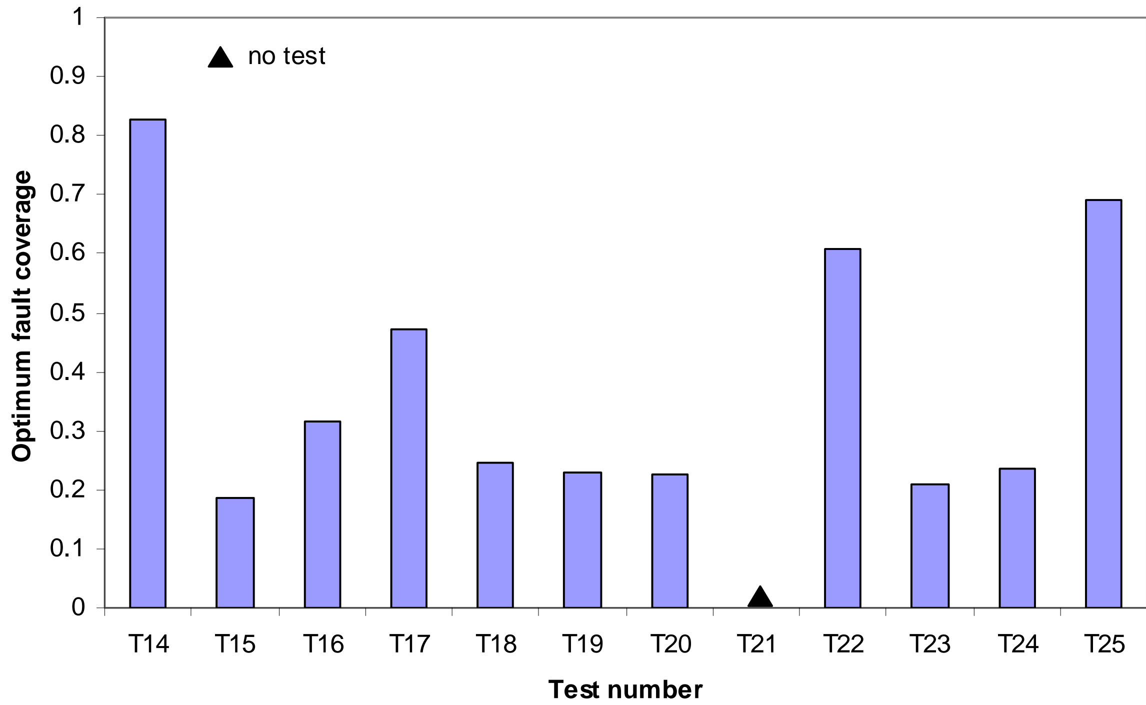


(a) Optimum fault coverage

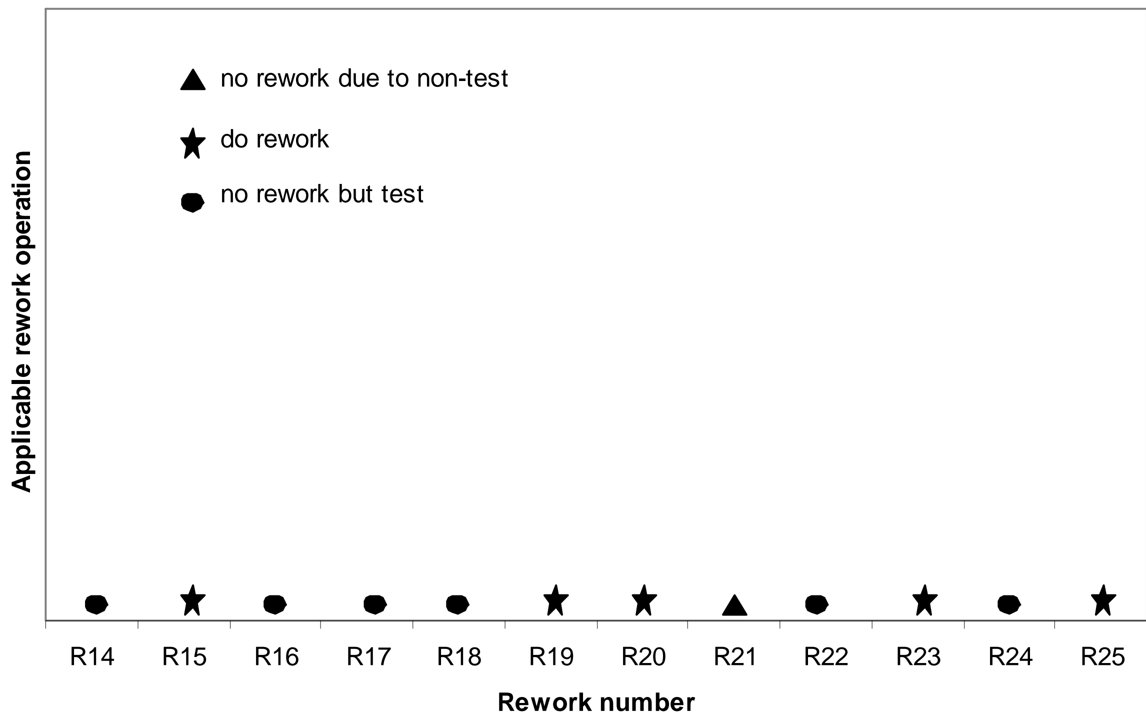


(b) Applicable rework operations

Figure 4.23: Computed optimum fault coverage and applicable rework TDR operations in the process flow shown in Figure 4.22 (C_{ft}= \$1 and C_{fr}= \$1).



(a) Optimum fault coverage



(b) Applicable rework operations

Figure 4.24: Computed optimum fault coverage and applicable rework operations in the process flow shown in Figure 4.22 (C_{ft}= \$1 and C_{fr}= \$100).

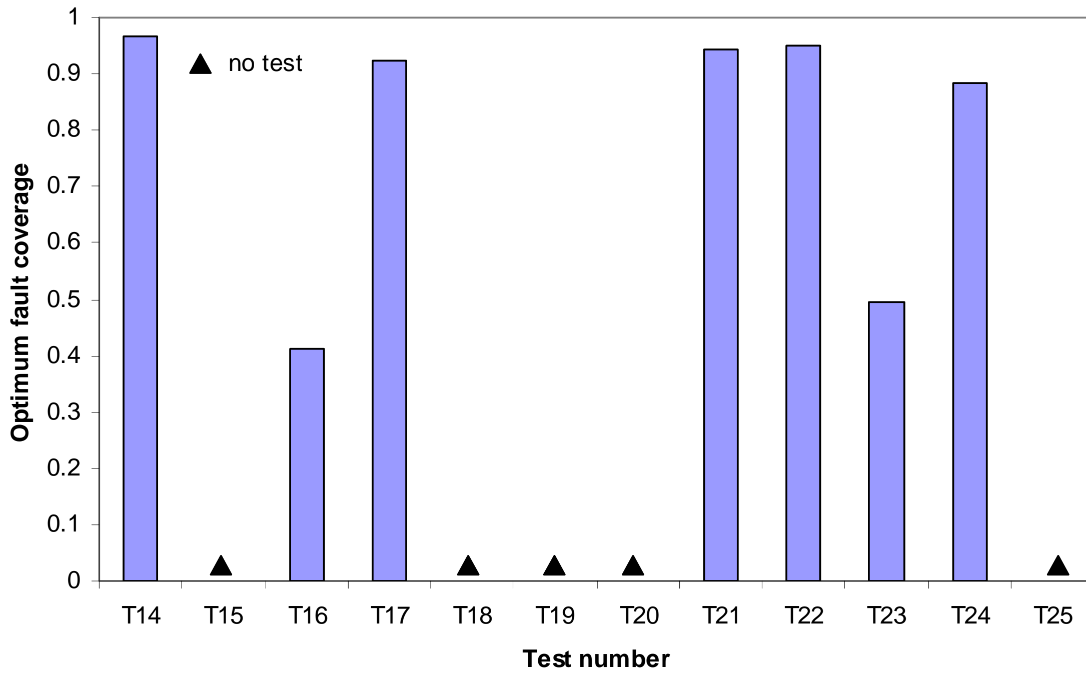
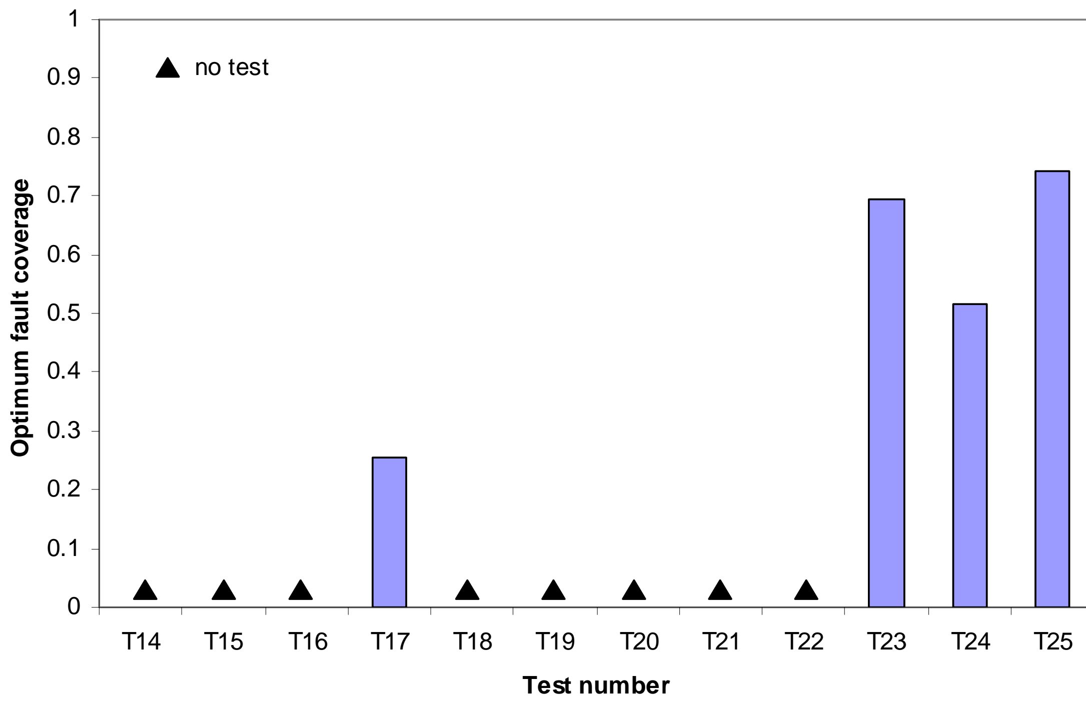
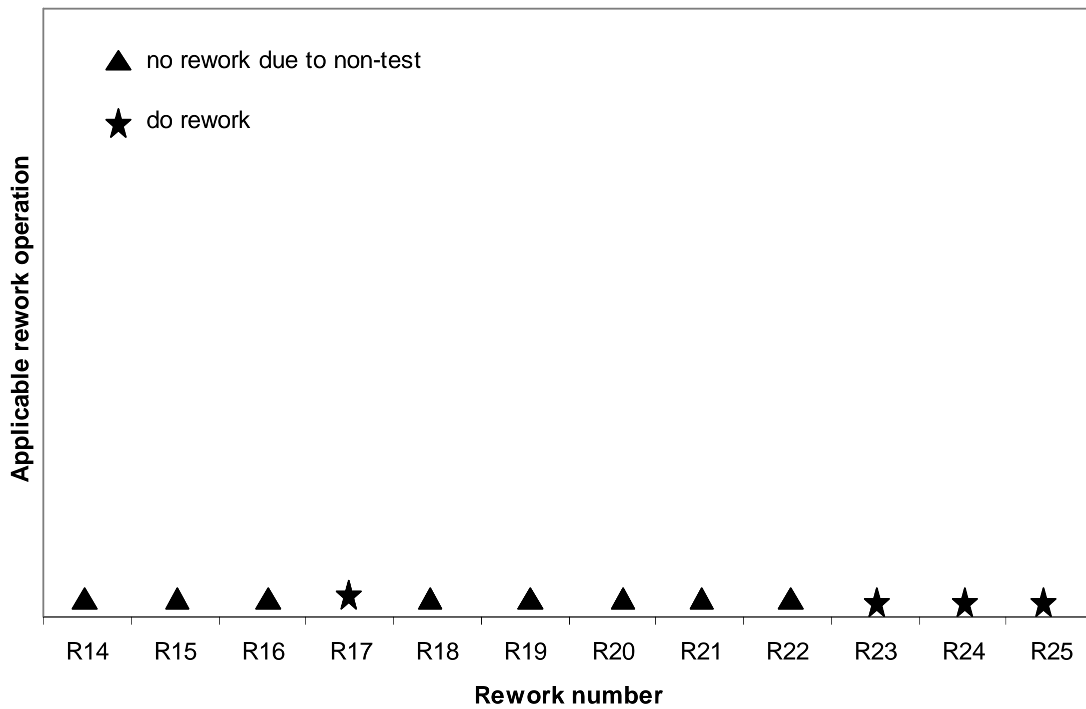


Figure 4.25: Computed fault coverage of test operations in the process flow shown in Figure 4.22 (C_{ft}= $1 and no rework).



(a) Optimum fault coverage



(b) Applicable rework operations

Figure 4.26: Computed optimum fault coverage and applicable rework operations in the process flow shown in Figure 4.22 (C_{ft}= \$50 and C_{fr}= \$100).

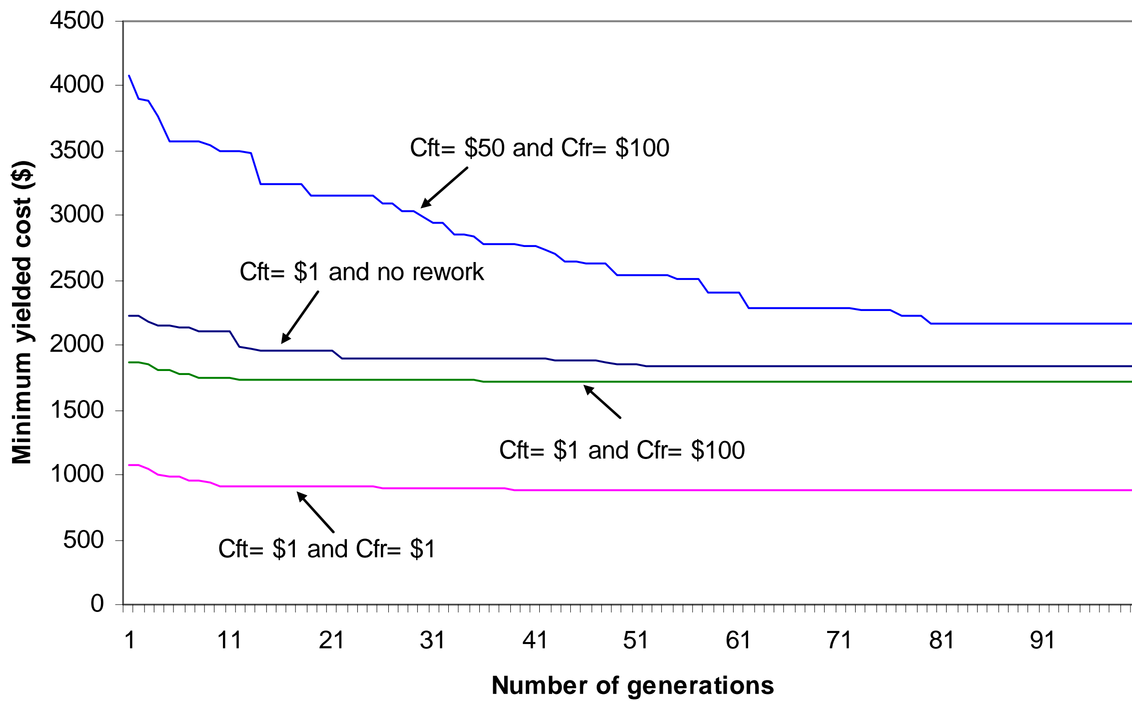


Figure 4.27: Optimization of yielded cost for various levels of fixed cost of test and rework operations.

Figure 4.27 shows example convergence characteristics of the minimum yielded cost of the complex process flow with the number of generations of the RCGAs. Table 4.6 compares the optimized TDR locations for various test and rework costs in the process flow in Figure 4.22.[23]

4.4 Optimization of Rework Attempts

The number of rework attempts is a characteristic of the rework operation that determines the tradeoff between cost and yield of products. With the increasing of rework attempts, the yield of products will increase correspondingly, but the cost increases too. According to the multi-variable objective function defined in Chapter 3, there could be a sequence of

[23] The 'Y' in Table 4.6 indicates that the test or rework operation is kept after the optimization and the blank means there is no test or rework operation placed in the corresponding location.

variables to be optimized simultaneously during the GAs optimization process, i.e., the optimization algorithms can be easily extended to accommodate another variable that needs to be optimized in order to minimize the yielded cost. The difference between placing the rework attempts into the optimization loop and optimizing the fault coverage of test is the range of variables that has been predefined. For the fault coverage, the GAs generate the gene within the range from 0 to 1. Otherwise, the gene to represent the rework attempts should be the integer number that is greater than 0.[24] That means the GAs sample the values of genes for each variable from a different solution space. As one of advantages using real-coded genetic algorithms, it is convenient for RCGAs to represent real values within different ranges.

Table 4.6: Optimized TDR locations in the process flow shown in Figure 4.22 for various fixed test cost and rework cost.

	C_{ft}= \$1 and C_{fr}= \$1		C_{ft}= \$1 and C_{fr}= \$100		C_{ft}= \$1 and no rework		C_{ft}= \$50 and C_{fr}= 100	
	Test	Rework	Test	Rework	Test	Rework	Test	Rework
TDR 14	Y	Y	Y		Y			
TDR 15	Y	Y	Y	Y				
TDR 16	Y	Y	Y		Y			
TDR 17			Y		Y		Y	Y
TDR 18			Y					
TDR 19	Y	Y	Y	Y				
TDR 20	Y	Y	Y	Y				
TDR 21					Y			
TDR 22	Y	Y	Y		Y			
TDR 23	Y	Y	Y	Y	Y		Y	Y
TDR 24	Y	Y	Y		Y		Y	Y
TDR 25	Y	Y	Y	Y			Y	Y

[24] For practical rework operations, the number of rework attempts usually is limited to below 10. The range of gene to represent the variable of rework attempts is defined between 1 and 10 in this dissertation. Zero rework attempts means there is no rework operation included.

To optimize the number of rework attempts, the rework operation should be included with testing, i.e., the rework is not optional. Rework attempts for each TDR location is considered as the variable to be optimized along with fault coverage. The details of the application to optimize the rework attempts for a real manufacturing process will be addressed in next chapter.

4.5 Summary

An optimization modeling methodology with Real-Coded Genetic Algorithms (RCGAs) integrated has been developed to optimize critical parameters and possible TDR locations for general process flows. The methodology developed in this chapter guides the placement of TDR operations in practical manufacturing processes and has been applied to several manufacturing processes for the purposes of simple quantitative and qualitative verification.

The next chapter will apply the optimization algorithms to a real case — a multichip module assembly process flow, in which the minimized yielded cost with optimum feature parameters will be compared to the actual test location and characterization chosen for the module.

Chapter 5

Optimization of TDR in Multichip Module (MCM) Assembly

This chapter will explore a real case of multichip module (MCM) manufacturing. The test process flow that minimizes the yielded cost with optimum fault coverage and rework attempts using the optimization algorithms developed in this dissertation will be performed. The optimum results will be compared with the calculation from the test strategies proposed in [74, 75], in which specific fault coverage tests are applied without any optimization techniques.

5.1 Introduction to Multichip Module

The multichip module is a packaging approach in which multiple bare die are mounted and interconnected on a substrate [76, 77]. Since the substrate, or the chip carrier, usually has substantially finer conductor lines, smaller dielectric thickness, and a denser via grid than conventional printed circuit boards, MCMs are not subject to conventional printed circuit board design rules and assembly restrictions. Figure 5.1 show a picture of a typical multichip module.

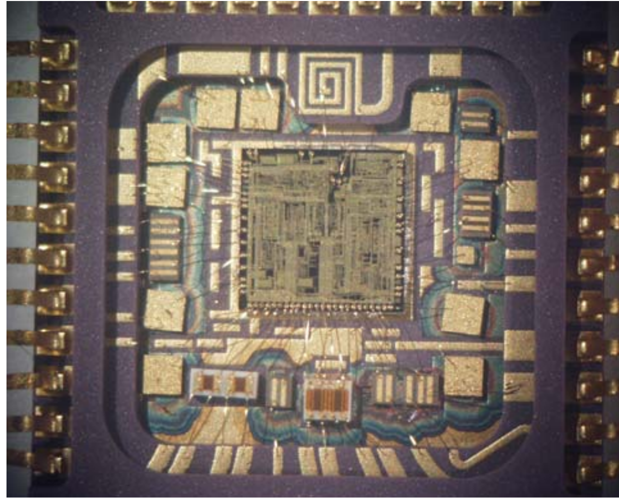


Figure 5.1: A typical multichip module. (Courtesy Maxtek Co.)

The substrate is used for supporting the die as well as the media for the interconnections among the die. The bare die are attached to the substrate by using a method called die bonding. The interconnections among the die are made using metal traces plated on layers in the substrate. Typically, different layers are used for making horizontal and vertical connections. Additional layers are provided for power and ground connections. The electrical connections between the die and the substrate are called chip I/Os and connections between the MCM and the outside world such as a PCB are module I/Os.

MCMs are usually categorized by substrate type and have been defined by the Institute for Interconnecting and Packaging Electronic Circuits (IPC) [78, 79]:

- MCM-L: Substrates based on laminated, multilayer PCB technology [80]
- MCM-C: Substrates based on co-fired ceramic or glass-ceramic technology
- MCM-D: Interconnection pattern formed by the deposition of dielectics and conductors, on a base substrate, typically by a thin film process.

MCMs completely eliminate chip packages, which allows closer chip-to-chip spacing and improved reliability due to a dramatic reduction in the number of fatigue prone solder joints and hermetic seals [81-83]. The requirement for testing is even higher in MCMs than in PCB or VLSI IC, since they are usually more expensive and used in high performance applications [84-87]. The next section will discuss the testing process of MCMs.

5.2 Testing of an MCM

Testing MCMs is a difficult task because they contain multiple high I/O count chips connected together into one circuit with high density interconnections. MCMs differ from conventional electronic assemblies in that the ICs are “bare” or un-packaged. Using bare die instead of die packaged into chips allows smaller systems to be built and eliminates electrical parasitics associated with the chip packaging.

Die yield plays an important role in MCMs because the die cannot always be pre-tested prior to assembly the way they would be if they were packaged. Die packaging provides protection for the die, a format that is easy to store and handle, and most importantly a format that enables die testing. For example, for a 50-chip MCM, if the yield of the incoming die is 95%, then the yield of the assembled MCM before testing will be 7.7% (without considering any other source of yield loss, see Figure 1.7). This means that more than 90% of the assembled MCMs have to be diagnosed and repaired, a costly and time consuming task.

Abadir *et al.* [74] have done extensive of work on the test strategies for MCMs. They analyzed the test/diagnosis/rework (TDR) process for an MCM using a simple MCM

manufacturing and test model, and calculated the cost of an MCM after TDR with various test fault coverages, which are applied to die of different incoming yield. The paper [74] compares the results from the various test cases (different fault coverage and test cost), but does not determine the optimum fault coverage of the test process to minimize the yielded cost of the MCM after TDR. The following will explore the optimization of a TDR process for the MCM described in [74] and show that the optimized test coverage leads to a lower yielded cost than the calculation from the conventional scenarios of specified test coverages considered in [74]. Before applying the optimization algorithms developed in Chapter 4 to the MCM assembly process flow, the model of the MCM manufacturing process and the tradeoff analysis for various test strategies from [74] will be introduced.

5.3 The Test/Diagnosis/Rework (TDR) of an MCM

A generic assembly, test and rework simulation model has been developed to analyze the relationship among all the parameters of the test process and how they impact the final cost and quality of the MCM. Figure 5.2 illustrates how the three basic process models are used to create a simple MCM manufacturing flow.

Each one of the boxes in Figure 5.2 represents a certain type of process (assembly, test and rework). The assembly box takes one or more inputs that correspond to the various types of die to be assembled and the substrate. Each of inputs is characterized by a cost yield and a count indication of how many times it is used (e.g., ten SRAM chips). The assembly process itself has a cost and a yield factor, as does the pre-tested MCM substrate. The assembled module is tested with a process that has a cost and a certain

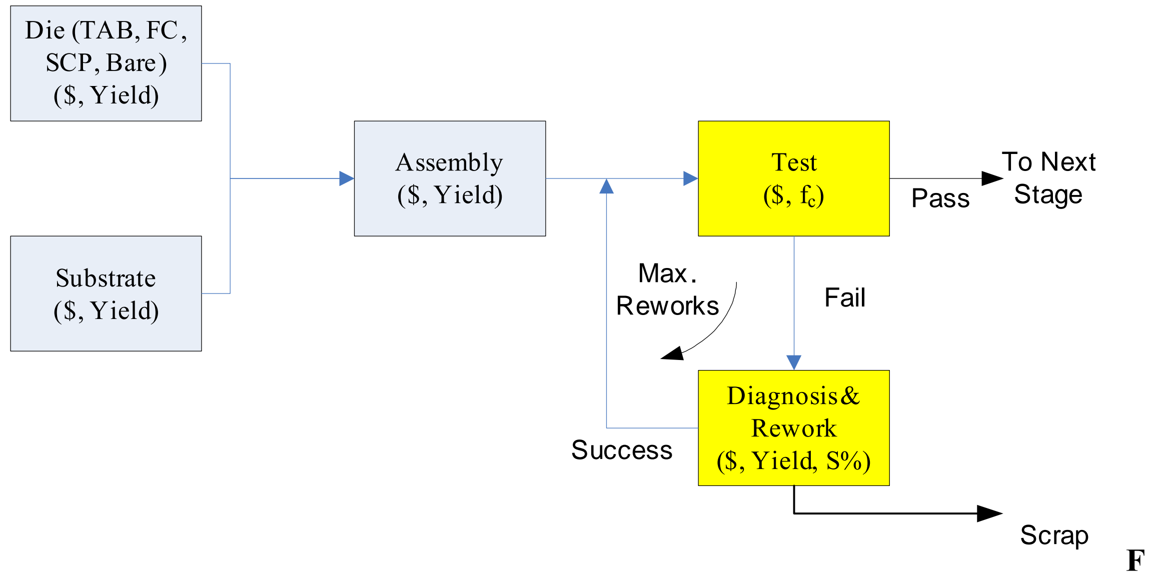


Figure 5.2: A simple MCM manufacturing and test model [74]

degree of fault coverage. The modules that pass the test are assumed good and are passed to the next level of assembly. However, since the testing process cannot detect all possible defects, some defective modules may escape and reduce the overall quality of the output modules.

The modules that fail the test are diagnosed for possible rework — some modules may not be repairable and are scrapped. The reworkable modules are reworked and are subjected again to the test process. The diagnosis and rework process has an average cost and also has a yield (i.e., not all repaired modules are good due to rework assembly defects, new components defects, or because of misdiagnosis). A module is scrapped if it fails to pass the test after being repaired a maximum allowable number of times (maximum number of rework attempts).

The simple model of Figure 5.2 does not have an explicit MCM burn-in processing step.[25] Such a step can be added at the end and be followed by another test and rework cycle, or it can be inserted after assembly and before test.

In this chapter we use the example described in [74] as the tested MCM (to compare it with the optimization results generated by algorithms in this dissertation). The problem to be treated is the following: An MCM is to be built containing 50 identical die. These die can be procured (purchased or made internally) at the different cost and quality levels indicated in Table 5.1. For example, the table indicates that a die that costs $2 has an 80% yield (i.e., 20% of those die are defective), while a version of the same die can be bought for $9 with 99.9% yield. Note that this is the yield of die before assembly onto the MCM. This is not an unrealistic situation. Bare die may be purchasable as wafers from the semiconductor manufacturer, or in a tested (i.e., Known Good Die) form from an aftermarket supplier who has performed bare die test (and burn-in) to pre-sort the good (no-defective) die from the bad (defective) die.

[25] Burn-in is used to eliminate early failures in components or systems, which is the application of elevated temperature to cause latent defects to fail in the manufacturing process rather than to have the unit fail during subsequent assembly or use process [84, 88].

Table 5.1: Chip yield versus MCM yield, cost and yielded cost.[26]

Case 1: f_c = 95%, Test Cost= $100							
Die Cost ($)	Die Yield	Yield of MCM (before test)	Defect Level (after test, PPM)	Yield of MCM (after test)	Yield of MCM (after rework)	Cost of MCM (after rework, $)	Yielded Cost ($)
2	0.8	1.43E-05	427652	0.572	0.989	1034	1045.61
3	0.85	2.96E-04	333987	0.666	0.992	1038	1046.69
4	0.9	5.15E-03	231681	0.768	0.993	1039	1046.32
5	0.94	0.05	143446	0.857	0.989	1037	1048.85
6	0.97	0.22	73460	0.927	0.980	1024	1045.43
7	0.99	0.61	24959	0.975	0.983	959	975.49
8	0.995	0.78	12601	0.987	0.990	961	971.2
9	0.999	0.95	2647	0.997	0.998	963	965.41
Case 1: f_c= 99%, Test Cost= $150							
Die Cost ($)	Die Yield	Yield of MCM (before test)	Defect Level (after test, PPM)	Yield of MCM (after test)	Yield of MCM (after rework)	Cost of MCM (after rework, $)	Yielded Cost ($)
2	0.8	1.43E-05	105599	0.894	0.998	1171	1173.59
3	0.85	2.96E-04	78073	0.922	0.998	1165	1166.99
4	0.9	5.15E-03	51345	0.949	0.999	1157	1158.62
5	0.94	0.05	30493	0.970	0.998	1148	1150.53
6	0.97	0.22	15143	0.985	0.996	1124	1128.51
7	0.99	0.61	5042	0.995	0.997	1034	1037.53
8	0.995	0.78	2533	0.997	0.998	1025	1027.16
9	0.999	0.95	530	0.999	1.000	1016	1016.5
Case 1: f_c= 99.9%, Test Cost= $200							
Die Cost ($)	Die Yield	Yield of MCM (before test)	Defect Level (after test, PPM)	Yield of MCM (after test)	Yield of MCM (after rework)	Cost of MCM (after rework, $)	Yielded Cost ($)
2	0.8	1.43E-05	11098	0.989	1.000	1302	1302.26
3	0.85	2.96E-04	8095	0.992	1.000	1288	1288.26
4	0.9	5.15E-03	5257	0.995	1.000	1272	1272.13
5	0.94	0.05	3091	0.997	1.000	1255	1255.25
6	0.97	0.22	1524	0.998	1.000	1220	1220.49
7	0.99	0.61	505	0.999	1.000	1107	1107.33
8	0.995	0.78	253	1.000	1.000	1088	1088.22
9	0.999	0.95	53	1.000	1.000	1069	1069.11

[26] The data shaded in GRAY in Table 5.1 is from [74], the other values were calculated in this dissertation.

The assumptions in Table 5.2 are used for the various parameters of the MCMs manufacturing and test flow in Figure 5.1:

Table 5.2: Values of parameters used in the MCM manufacturing and test flow in Figure 5.1.

Assembly cost of 50 die on substrate	$200
Substrate cost	$200
Testing cost of an 50-chip MCM	$150
Fault coverage on testing MCMs	95%
Repair cost of one MCM failed in test	$100
Rework scrap yield	5%
Maximum rework attempts	1
Rework yield of repairing a single MCM	Minimum (chip yield, 95%)

Both the substrate and the assembly process to mount the 50 die onto the substrate were assumed to have a perfect yield (100%) in this exercise. Hence, the die were the only source of defects for this module. The rework scrap yield is the fraction of MCMs that cannot be reworked and are scraped.

In [74], three different test strategies were proposed to compare the yielded cost of a MCM:

- Case 1: Test coverage on MCMs testing =95%, Test Cost = $100;
- Case 2: Test coverage on MCMs testing =99%, Test Cost = $150;
- Case 3: Test coverage on MCMs testing 99.9%, Test Cost = $200.

Table 5.1 lists all the original data (from [74]) and calculation results according to each case for various incoming yield of die. For the purpose of explaining how the yielded cost of MCMs after the TDR operation is derived, the calculation of the MCM assembly process cost and yield is reviewed below.

For example, in Case 1, if the input of die yield is 0.8, the yield of MCM module before test should be:

$$\text{die yield} = (\text{die yield})^{\text{number of die}} = (0.8)^{50} = 1.43 \times 10^{-5} \tag{5.1}$$

The final cost of a MCM module is the accumulation of die cost, substrate cost and assembly cost:

$$\text{Cost of MCM} = \text{Die cost} + \text{Substrate cost} + \text{Assembly cost} \tag{5.2}$$

$$= 50(\$2) + \$200 + \$200$$

$$= \$500$$

The final yield of MCM module can be determined by multiplying the die yield, substrate yield and assembly yield:

$$\text{Yield of MCM} = (\text{die yield})\ (\text{substrate yield})\ (\text{assembly yield}) \tag{5.3}$$

$$= (0.8)^{50}\ (1)\ (1)$$

$$= 1.43 \times 10^{-5}$$

which is a very small number.

After the test, the yield of MCM module can be calculated using (5.4):

$$\text{Yield of MCM} = (Y_{in})^{(1\text{- fault coverage})} \tag{5.4}$$

$$= (1.43\times10^{-5})^{(1\text{-}0.95)}$$

$$= 0.5723$$

The defect level of MCM module in parts per million (ppm) is derived from (5.5),

$$\text{Defect level} = (1\text{-yield of MCM})10^{6} = 4.3\times10^{5} \text{ ppm} \tag{5.5}$$

The yield, cost and yielded cost of MCM module after the rework for this case (one TDR) can be calculated following the example in Section 2.7.

All the data (except values shaded in GRAY) in Table 5.1 are calculated using above calculation procedure. From Table 5.1, we can conclude:

1) Using the die of highest yield for the MCMs assembly is not always the best choice for the MCM manufacturer, which means the very high quality of die may lead to more expensive systems, at least from a yielded cost standpoint. An alternative is to select the middle-level quality die and test MCMs with high fault coverage to lower the final yielded cost.

2) The fault coverage of testing is related to the input yields. For the lower input die yield, the high fault coverage does make sense because the bad parts can be identified from test and be repaired during rework operation. The total yielded cost of MCMs still can be saved by test and rework operations instead of scraping the whole module.

However for the high-yield die, high fault coverage tests only increase the cost of systems without improving the final yield distinctly.

3) For the given range of desired system yield, there are several combinations of incoming die yield and fault coverage that produce the same yielded cost. For example, we can use $6 die combined with 95% fault coverage, which results in the roughly the same yielded cost of MCMs with choosing $7 die under the testing of 99% fault coverage.

The test case scenarios considered in [74] are representative of the conventional approach to determining test locations within a system in which a tradeoff analysis of test strategies is based on enumerating all the different cases. There is no optimization method included in [74], which makes it impossible to perform the analysis of test strategies for the manufacturing or assembly process of complicated electronic products or systems. Furthermore, a manufacturing manager doesn't usually care about how many alternatives can be chosen, rather, the manager wants to know what the best assembly process flow is for achieving the goal of minimized yielded cost.

The conventional wisdom on test placement does not always lead to an optimum solution as we will demonstrate in the next section.

5.4 Optimization of Fault Coverage in the MCM Assembly Process Flow

This section will apply the optimization method proposed in Chapter 4 to the MCM manufacturing and test process flow discussed in [74]. The optimum fault coverages to

minimize the yielded cost of the MCM for specific input die yield are generated by RCGAs without enumerating all the combinations of possible test strategies manually. Abadir, *et. al.* [74] only gives the testing cost of a 50-chip MCM for 3 different fault coverages assumptions, the functional relationship between test cost and fault coverage used in optimization algorithms should be determined first.

In this real case of the MCM assembly process in [74], a functional relationship between test cost and the resulting fault coverage has been determined based on the cost data of Table 5.3 used by [74] and (2.17).

Table 5.3: Fault coverage with its corresponding testing cost used in [74].

Fault coverage of test	Testing cost of an 50-chip MCM
95%	$100
99%	$150
99.9%	$200

Using a nonlinear regression analysis technique, Equation (5.6) gives the natural relationship between test cost and fault coverage that is used in the RCGAs optimization of the MCM assembly process, [27]

$$C_{test} = \begin{cases} -21.68\ln(1-f_c)+25 & (0.1 < f_c < 1) \\ 0 & (0 \le f_c \le 0.1) \end{cases} \qquad (5.6)$$

[27] Equation (5.6) fits the relationship between fault coverage and test cost using the data of Table (5.3) with a function of the form given by (2.17).

Figure 5.3 plots (5.6) and shows the 10% threshold of fault coverage. The threshold is defined as the lowest nonzero fault coverage that test equipment can provide or we can purchase practically. For any fault coverage below this threshold, there is no test present (zero fault coverage and zero test cost).

Abadir *et al.* [74] only consider three combinations of fault coverage and test cost to show which test strategy should be selected for the better yield and lower cost of the MCM after test/diagnosis/rework operations. There are no optimum solutions provided in [74] that minimize the yielded cost of the assembled module with the optimized fault coverage. In the remainder of this section, the optimization methodology developed in Chapters 3 and 4 will be applied to the real MCM case presented in [74] and it will be demonstrated that an optimal solution can be obtained that results in a lower yielded cost system than the scenarios considered in [74].

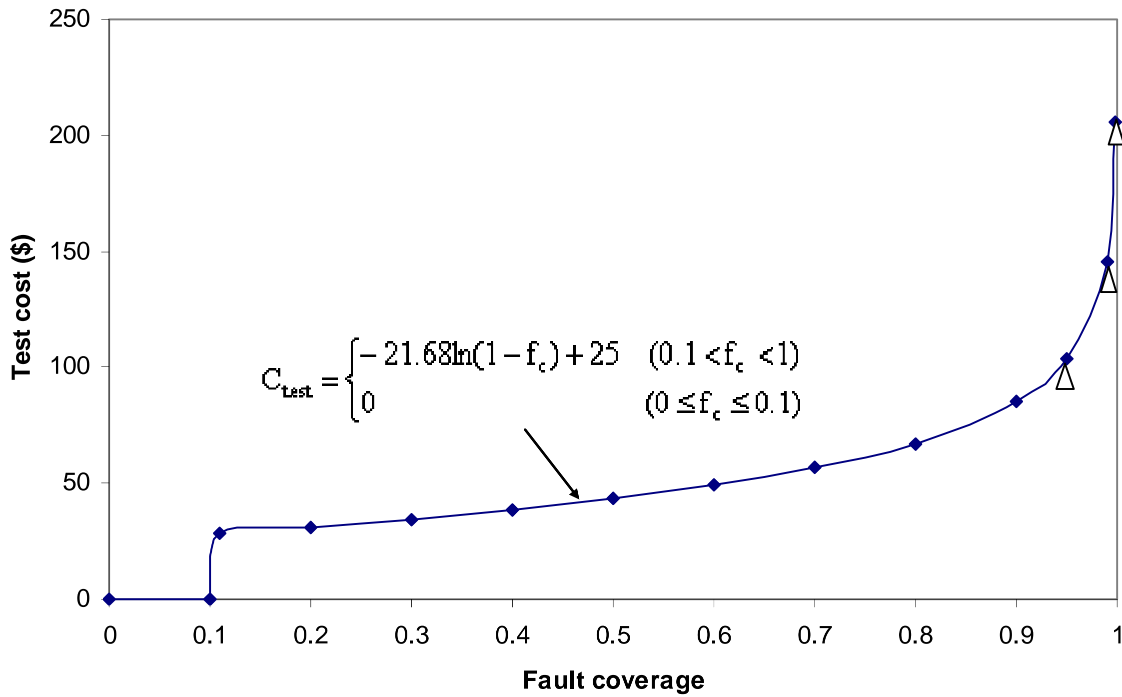


Figure 5.3: Functional relationship between fault coverage and test cost applied to the MCM example (The data points with triangle are from [74]).

Tables 5.4 — 5.11[28] show the optimization results for yielded cost generated by the RCGAs compared with data of three specific cases discussed in [74] for different inputs of die yield. The GAs can find the optimum fault coverage to minimize the yielded cost of the MCM which is always lower any calculation resulting from the various test strategies in [74]. From these tables, it is clear that the highest fault coverage may not lead to the lowest yielded cost due to the corresponding higher test cost.

[28] The data shaded in GRAY is from [74] and the GA optimum data is generated by the RCGAs.

Table 5.4: Comparison of GA optimum yielded cost with results from test strategies in [74] (die cost = $2).

Die Cost = $2					
	Fault Coverage	Test Cost($)	MCM Yield	MCM Cost ($)	MCM Yielded Cost ($)
Case 1	0.95	100	0.9889	1034	1045.61
Case 2	0.99	150	0.9978	1171	1173.58
Case 3	0.999	200	0.9998	1302	1302.26
GA Optimum	0.385	37.75	0.858	769.1	896.03

Table 5.5: Comparison of GA optimum yielded cost with results from test strategies in [74] (die cost = $3).

Die Cost = $3					
	Fault Coverage	Test Cost($)	MCM Yield	MCM Cost ($)	MCM Yielded Cost ($)
Case 1	0.95	100	0.9917	1038	1046.69
Case 2	0.99	150	0.9983	1165	1166.98
Case 3	0.999	200	0.9998	1288	1288.26
GA Optimum	0.48	42.23	0.9	828.26	921.46

Table 5.6: Comparison of GA optimum yielded cost with results from test strategies in [74] (die cost = $4).

Chip Cost = $4					
	Fault Coverage	Test Cost($)	MCM Yield	MCM Cost ($)	MCM Yielded Cost ($)
Case 1	0.95	100	0.993	1039	1046.32
Case 2	0.99	150	0.9986	1157	1158.62
Case 3	0.999	200	0.9999	1272	1272.13
GA Optimum	0.62	50.53	0.926	887.17	957.68

Table 5.7: Comparison of GA optimum yielded cost with results from test strategies in [74] (die cost = $5).

Chip Cost = $5					
	Fault Coverage	Test Cost($)	MCM Yield	MCM Cost ($)	MCM Yielded Cost ($)
Case 1	0.95	100	0.9887	1037	1048.85
Case 2	0.99	150	0.9978	1148	1150.53
Case 3	0.999	200	0.9998	1255	1255.25
GA Optimum	0.77	63.36	0.94	938.91	1003.63

Table 5.8: Comparison of GA optimum yielded cost with results from test strategies in [74] (die cost = $6).

Chip Cost = $6					
	Fault Coverage	Test Cost($)	MCM Yield	MCM Cost ($)	MCM Yielded Cost ($)
Case 1	0.95	100	0.9795	1024	1045.43
Case 2	0.99	150	0.996	1124	1128.51
Case 3	0.999	200	0.9996	1220	1220.49
GA Optimum	0.77	63.35	0.94	932.05	993.62

Table 5.9: Comparison of GA optimum yielded cost with results from test strategies in [74] (die cost = $7).

Chip Cost = $7					
	Fault Coverage	Test Cost($)	MCM Yield	MCM Cost ($)	MCM Yielded Cost ($)
Case 1	0.95	100	0.9831	959	975.49
Case 2	0.99	150	0.9966	1034	1037.53
Case 3	0.999	200	0.9997	1107	1107.33
GA Optimum	0.85	74.3	0.95	912.09	962.28

Table 5.10: Comparison of GA optimum yielded cost with results from test strategies in [74] (die cost = $8).

Chip Cost = $8					
	Fault Coverage	Test Cost($)	MCM Yield	MCM Cost ($)	MCM Yielded Cost ($)
Case 1	0.95	100	0.9895	961	971.2
Case 2	0.99	150	0.9979	1025	1027.16
Case 3	0.999	200	0.9998	1088	1088.22
GA Optimum	0.79	65.43	0.95	908.67	951.75

Table 5.11: Comparison of GA optimum yielded cost with results from test strategies in [74] (die cost = $9).

Chip Cost = $9					
	Fault Coverage	Test Cost($)	MCM Yield	MCM Cost ($)	MCM Yielded Cost ($)
Case 1	0.95	100	0.9975	963	965.41
Case 2	0.99	150	0.9995	1016	1016.51
Case 3	0.999	200	0.9999	1069	1069.11
GA Optimum	0	0	0.95	850	893.6

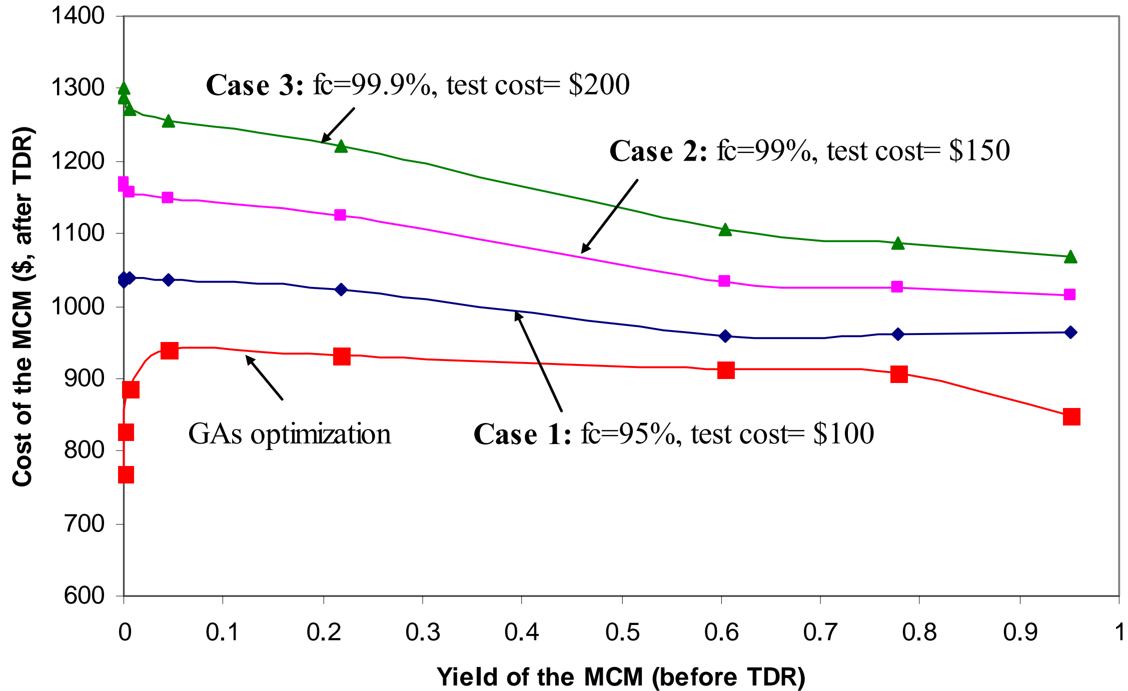


Figure 5.4: Cost of MCM after TDR versus incoming yield of MCM for the three test scenarios in [74] plus the optimum solution. [29]

Figure 5.4 shows how the cost of MCM after the test/diagnosis/rework operation varies with the yield of the MCM before testing. It is obvious that higher fault coverage tests always lead to higher cost. Comparing with the calculation results from the three test strategies discussed in [74], the cost of the MCM after TDR for each yield level of MCM before test is lower, which is derived from the optimum test coverage generated by GAs optimization. This means choosing or buying a specific fault coverage for the entire range of incoming die yield is not appropriate and a fixed fault coverage cannot ensure the lowest cost of product for different input yield. The optimum fault coverage generated by the GAs varies with the different die yield to the MCM assembly process flow. The next

[29] Yield of the MCM (before TDR) refers to the yield of the MCM after all die have been assembled onto the substrate, i.e., the incoming yield to the test operation shown in Figure 5.2.

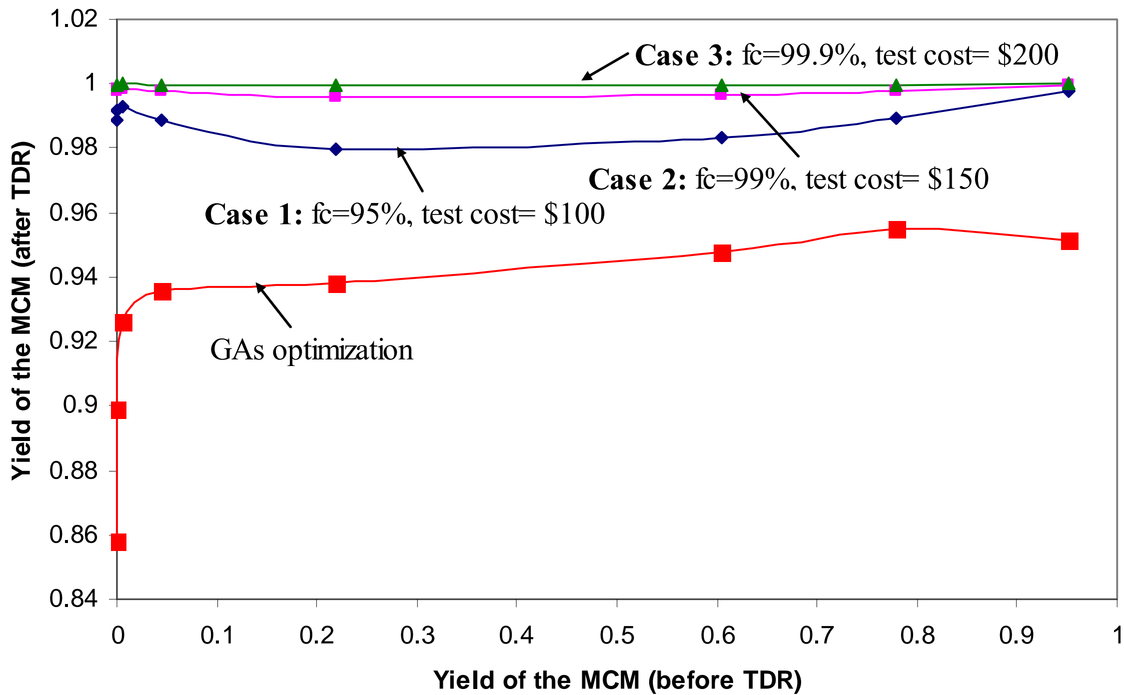


Figure 5.5: Output yield of MCM after TDR versus incoming yield of MCM for the three test scenarios in [74] plus the optimum solution.

chapter will improve the optimization algorithms developed in Chapter 4 in order to find the optimum fault coverages that are applicable to a range of input yields (e.g., a normal distribution) instead of being restricted to a single value.

The final yield of the MCM after going through all the assembly and test process flow is shown in Figure 5.5 in terms of the yield of assembled the MCM before testing. The GAs optimization results in the lowest outgoing yield of the MCM by applying the optimum fault coverage to the test operation. It is always true that higher incoming yield leads to higher yield level of the output. A manufacturing manager may be willing to pay some additional cost to reduce the defect level of die. Not only might this reduce test and repair costs at higher levels of integration, but more importantly it reduces field returns and the potential loss of customer satisfaction and future sales.

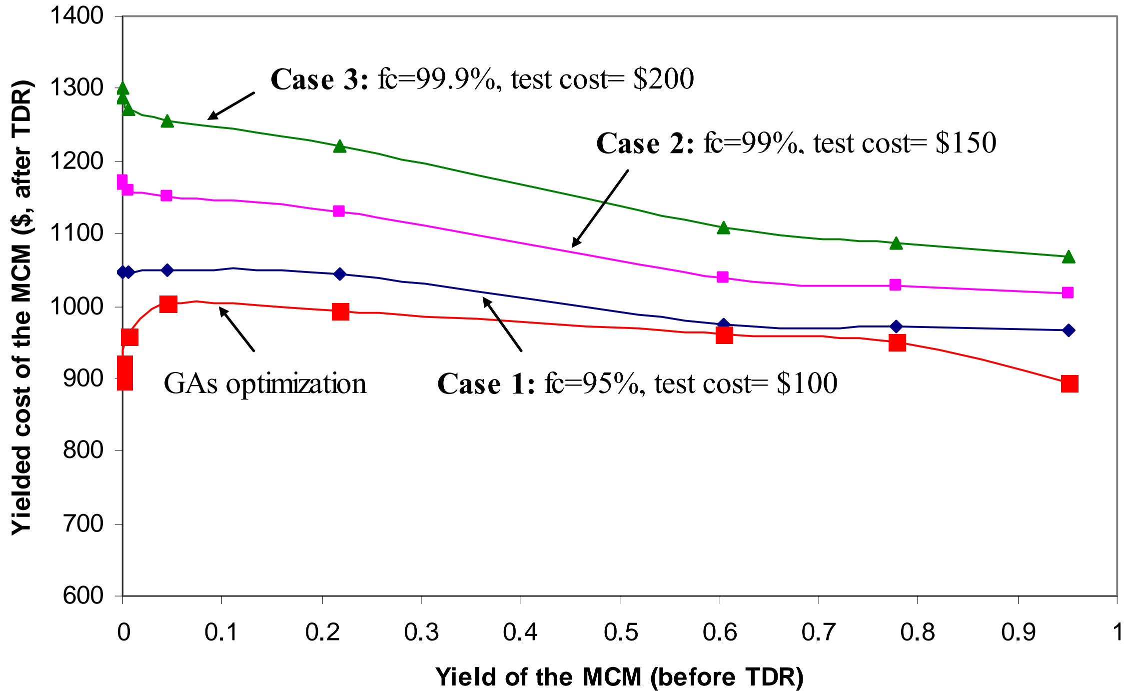


Figure 5.6: Yielded cost of MCM after TDR versus incoming of MCM for the three test scenarios in [74] plus the optimum solution.

Figure 5.6 implies that the optimization algorithm proposed in Chapter 4 does generate the optimum solution of yielded cost of MCM after final assembly and test. The optimization results stay at the lower yielded cost of the whole range of MCM module yields. Figure 5.7 shows the optimum fault coverage (determined by the GA optimization) for the various inputs of incoming yield of MCM module. For high incoming yield, the optimization algorithm chooses the lower fault coverage to decrease the cost of test. The optimization results prefer less than 100% fault coverage due to high cost associated with the test process as shown in Figure 5.7.

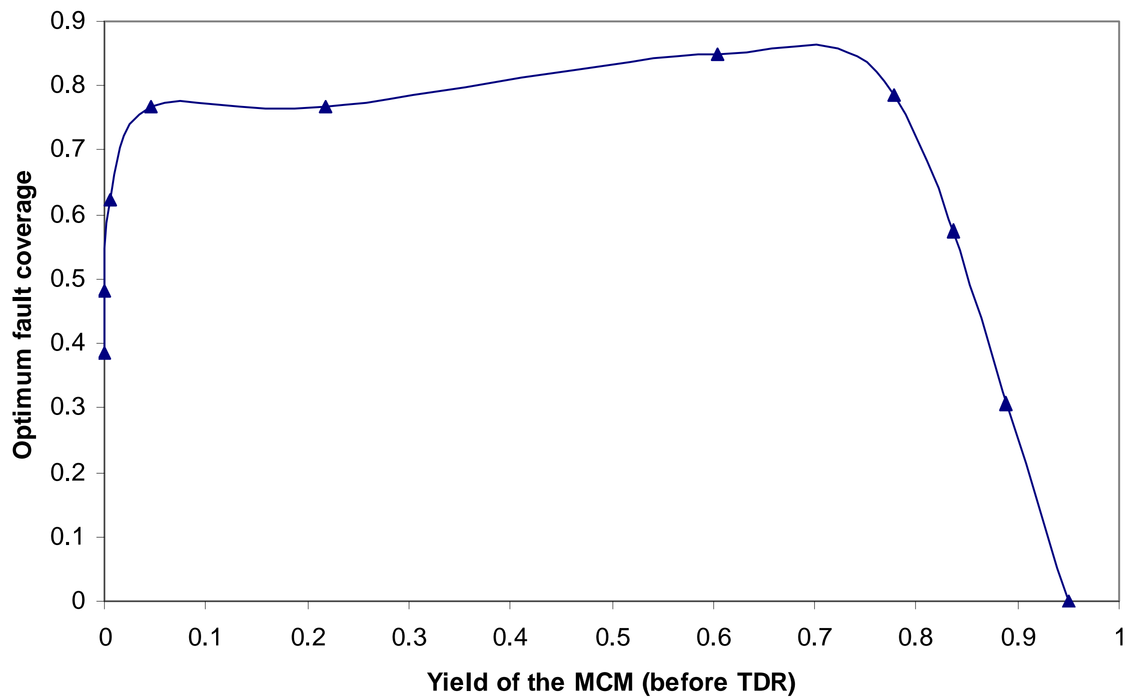


Figure 5.7: GA-based optimum fault coverage versus incoming yield of MCM.

An issue that needs to be addressed in the Figure 5.6 curve is why there is a drop in the optimum yielded cost when the incoming yield of assembled MCM is very low. Intuitively, the yielded cost should increase as input yield decreases. It will cost more to repair an MCM with a poor input yield level and the final yielded cost should be the highest compared with the MCM assembled by the high-yield die.

But, Figure 5.6 shows that there is decreasing of yielded cost on the low input yield of MCM before TDR, which is incorrect. This problem occurs because an assumption has been made in the optimization process of the TDR that the rework yield of repairing an MCM is fixed and the TDR model proposed in Chapter 2 only treats the repair operation as a whole without consideration of rework details of specific products. As defined in Table 5.2, the rework yield is the minimum value of 95% or die yield, which means the bad die identified in the test are replaced by the die with the yield level that is same as the yield of ones assembled onto MCM if the yield is lower than 95%. The Trichy *et. al.* model calculates the cost of rework process assuming a fixed rework yield of die no matter how many die are assembled onto it and does not consider the possibility that more than one die failed in the test.

Therefore, the optimization model, as used here, is not valid for very small incoming yields where the rework operation dominates the outgoing yield of products. The model should be modified to have flexible rework cost calculation based on the specific manufacturing process to fix the modeling error for the very low input yield.

After all the optimum fault coverages are generated by the GAs optimization, the results can be verified using the Trichy *et al.* model discussed in Chapter 2 with a full

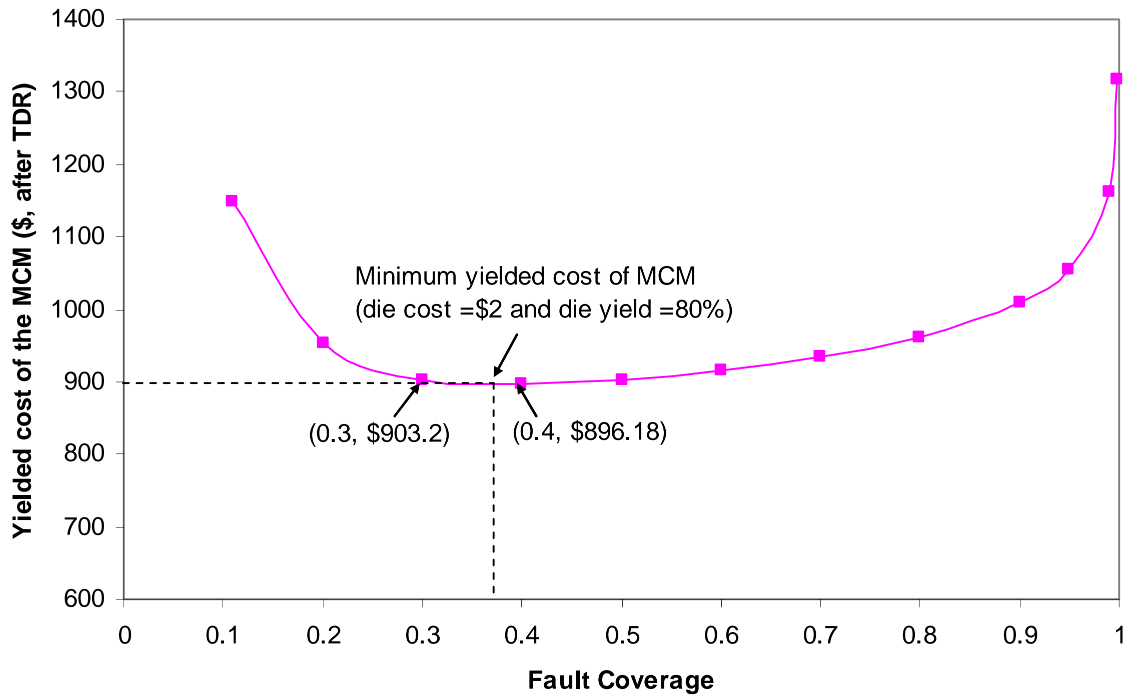


Figure 5.8: Yielded cost of the MCM (after TDR) versus fault coverage (die cost = $2 and die yield = 80%).

range of fault coverages for a specific incoming die yield. Then we compare the GAs optimum fault coverage value with these calculated values.

Figure 5.8 describes the relationship between all possible fault coverages of testing MCM and the yielded cost of the MCM after the TDR is done for the cost of incoming die at $2 with its yield level of 80%. On Figure 5.8, the expected minimum yielded cost should be lower than $896.18 with the possible range of fault coverage between 0.3 and 0.4, which validates the optimum result from Table 5.4 where the minimized yielded cost is $896.03 and the optimum of fault coverage is set at 0.39. Table 5.12 gives the optimization history for this case.

Table 5.12: Optimization of fault coverage in the MCM assembly process flow with the cost of incoming die is $2.

Number of generations	Optimum yielded cost ($)	Mean of yielded cost ($)	Optimum fault coverage
1	896.04	2102836.25	0.39
2	896.04	905.27	0.38
3	896.03	900.57	0.39
4	896.03	905.07	0.39
5	896.03	896.22	0.39
6	896.03	898.98	0.39
7	896.03	1402157.25	0.39
8	896.03	901.51	0.39
9	896.03	904.50	0.39
10	896.03	701532.00	0.39
11	896.03	905.99	0.39
12	896.03	900.61	0.39
13	896.03	701527.63	0.39
14	896.03	896.24	0.39
15	896.03	900.92	0.39
16	896.03	904.34	0.39
17	896.03	898.73	0.39
18	896.03	900.37	0.39
19	896.03	905.07	0.39
20	896.03	701531.63	0.39
21	896.03	900.65	0.39
22	896.03	897.69	0.39

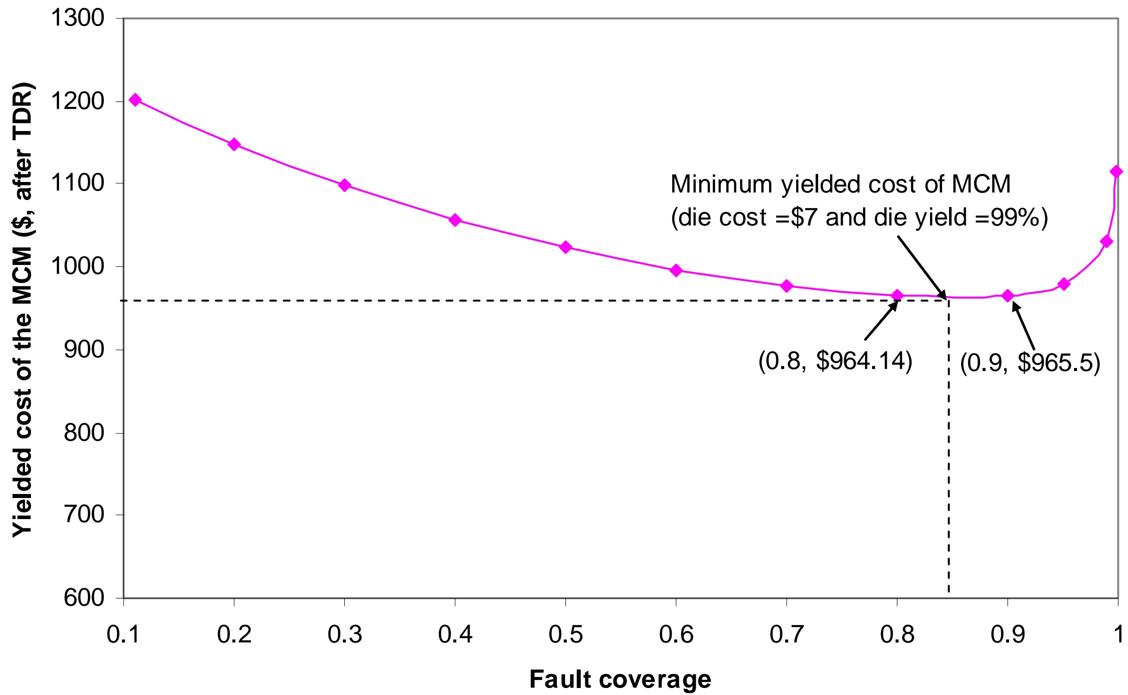


Figure 5.9: Yielded cost of MCM (after TDR) vs. fault coverage of test (die cost =$7 and die yield = 99%)

Similarly, Figures 5.9 and 5.10 verify the minimized yielded cost for incoming die costs of $7 and $8 with the corresponding optimization history given in Table 5.13 and 5.14 respectively. The GAs optimization effectively finds the minimum yielded cost of MCM in less than 10 generations.

Table 5.13: Optimization of fault coverage in the MCM assembly process flow with the cost of incoming die is $7.

Number of generations	Optimum yielded cost ($)	Mean of yielded cost ($)	Optimum fault coverage
1	962.40	1060.02	0.86
2	962.40	988.55	0.86
3	962.28	971.38	0.85
4	962.28	963.76	0.85
5	962.28	970.31	0.85
6	962.28	969.32	0.85
7	962.28	979.37	0.85
8	962.28	962.32	0.85
9	962.28	968.79	0.85
10	962.28	962.88	0.85
11	962.28	964.37	0.85
12	962.28	974.23	0.85
13	962.28	981.52	0.85
14	962.28	965.27	0.85
15	962.28	976.89	0.85
16	962.28	970.93	0.85
17	962.28	967.38	0.85
18	962.28	967.01	0.85
19	962.28	971.55	0.85
20	962.28	975.65	0.85
21	962.28	970.02	0.85
22	962.28	969.02	0.85

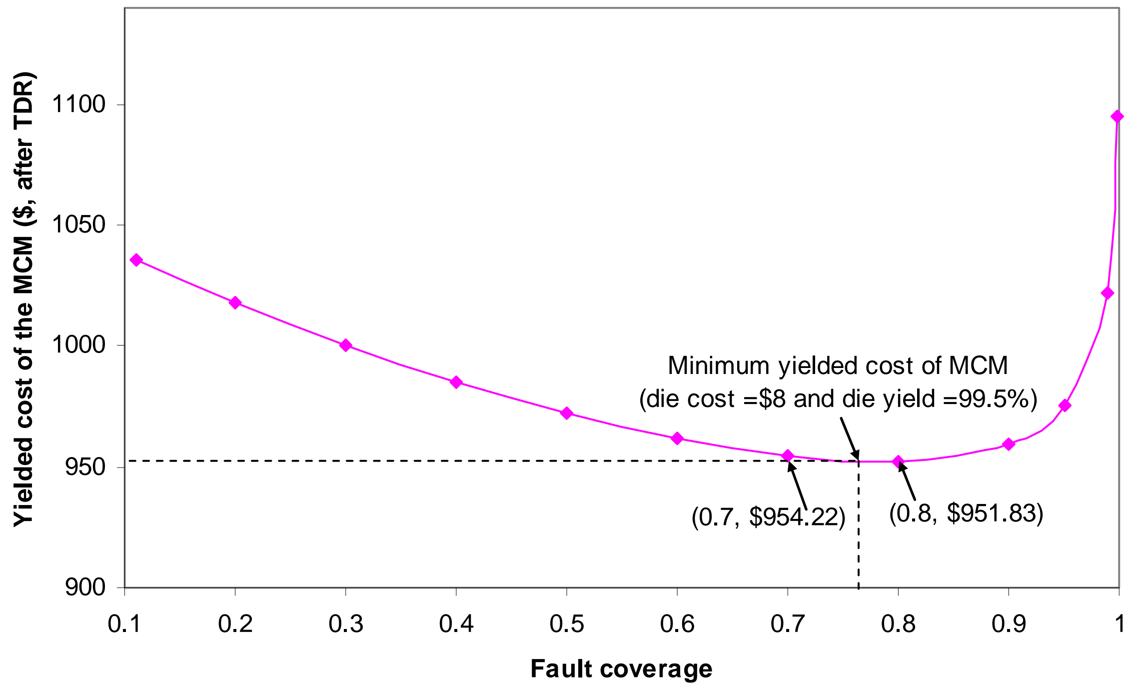


Figure 5.10: Yielded cost of MCM (after TDR) vs. fault coverage of test (die cost =$8 and die yield = 99.5%)

Table 5.14: Optimization of fault coverage in the MCM assembly process flow with the cost of incoming die is $8.

Number of generations	Optimum yielded cost ($)	Mean of yielded cost ($)	Optimum fault coverage
1	951.89	987.27	0.77
2	951.79	963.23	0.78
3	951.75	957.50	0.79
4	951.75	956.00	0.79
5	951.75	954.59	0.79
6	951.75	953.65	0.79
7	951.75	957.03	0.79
8	951.75	952.08	0.79
9	951.75	955.93	0.79
10	951.75	953.71	0.79
11	951.75	953.29	0.79
12	951.75	954.33	0.79
13	951.75	953.66	0.79
14	951.75	952.88	0.79
15	951.75	954.92	0.79
16	951.75	953.97	0.79
17	951.75	954.43	0.79
18	951.75	958.27	0.79
19	951.75	954.41	0.79
20	951.75	953.17	0.79
21	951.75	954.54	0.79
22	951.75	955.65	0.79

This section applied the GAs optimization to minimize the yielded cost of an MCM during its assembly process with fault coverage optimized. The optimization results have been verified as the minimum for various combinations of die cost and yield. Compared with the test strategies discussed in [74], in which there is no optimization performed, the GAs based optimization successfully generates the lowest value of objective function defined in Chapter 3 by selecting the optimum characteristics of the test. From Table 5.15, the GAs-based optimum yielded cost is an average of 6.85% lower than that obtained when using test strategies based on selecting specific fault coverages as in [74]. Furthermore, in a complicated process flow, it would be impossible to enumerate all fault coverages manually without employing optimization techniques. The optimization algorithms developed in this dissertation will effectively find the optimized solution in less time than traditional approaches.

Table 5.15: Cost reduction efficiency of minimum yielded cost with various input of die yield.

Die cost ($)		\$2	\$3	\$4	\$5	\$6	\$7	\$8	\$9	Average
Minimum yielded cost ($)	Test strategies in [74]	1046	1047	1046	1049	1045	975	971	965	1018.12
	GAs-based Optimization	896	921	958	1004	994	962	952	894	947.507
	Cost reduction efficiency[30]	0.14	0.12	0.08	0.04	0.05	0.01	0.02	0.1	0.0685

[30] The cost reduction efficiency is defined as the ratio of cost reduction between the minimum yielded cost generated by GAs-based optimization and calculated using the test strategies in [74] divided by the later value.

5.5 Optimization of Rework Attempts for MCM Assembly Process Flow

Rework attempts is fixed at one for the optimization of the MCM assembly case in all of the discussion so far in this chapter. This section will include rework attempts as a variable to be optimized for the lowest yielded cost of systems.

The MCM assembly process flow (the die yield is 85% and die cost is $3) is also used for the example demonstration of application of the GAs optimization with the rework attempts to be optimized. The rework cost is assumed as a constant of $100 used for all attempts of the rework operation and the rework yield is also fixed at the minimum of

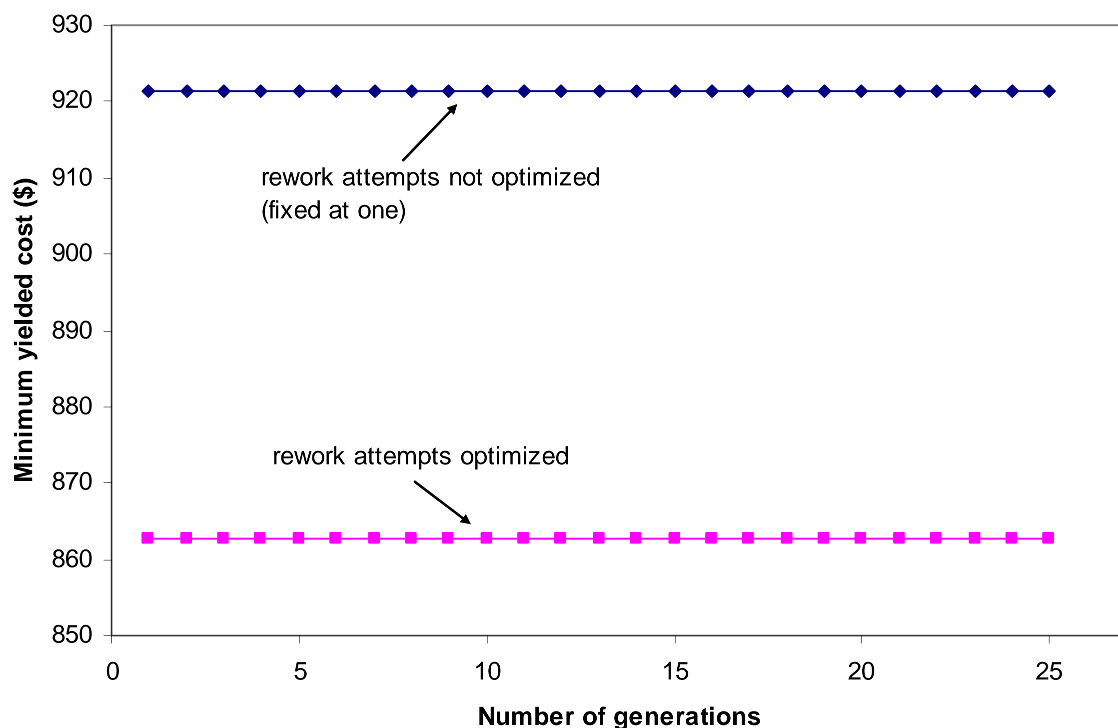


Figure 5.11: Comparison of minimum yielded cost generated by optimization of various rework attempts and fixed rework attempts.

95% or the die yield.[31] Figure 5.11 compares the minimum yielded cost with a fixed number of rework attempts (one). The yielded cost has been decreased with the rework attempts optimized. The number of rework attempts at the minimum yielded cost is shown in Figure 5.12. Assuming a fixed rework cost, the GAs optimization prefers more rework operations to improve the outgoing yield of the MCM. Figure 5.13 shows the change of fault coverage at the minimum yielded cost when the number of rework attempts is optimized. In Figure 5.13, the fault coverage at the minimum yielded cost after the number of rework attempts has been optimized increases compared to the optimum value with rework attempts fixed at one. Figure 5.13 indicates that the GAs optimization is willing to pay for higher fault coverage in order to have more defective MCM failed in the test when more rework attempts are available. Tables 5.16 and 5.17 list the optimization history for both cases discussed above.

[31] Trichy *et al.* model [9] treats the rework operation as a whole, and the variation of rework yield and rework cost depending on the structure of assembled modules or systems is not addressed.

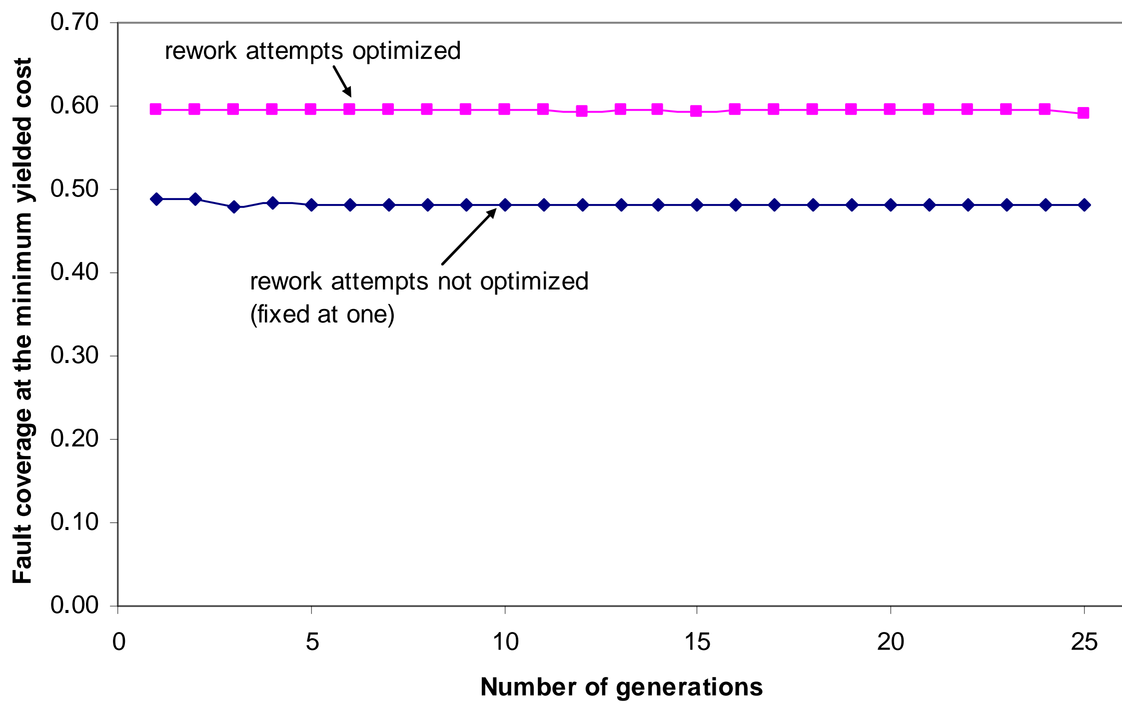


Figure 5.12: Comparison of fault coverage at the minimum yielded cost at various rework attempts and fixed rework attempts.

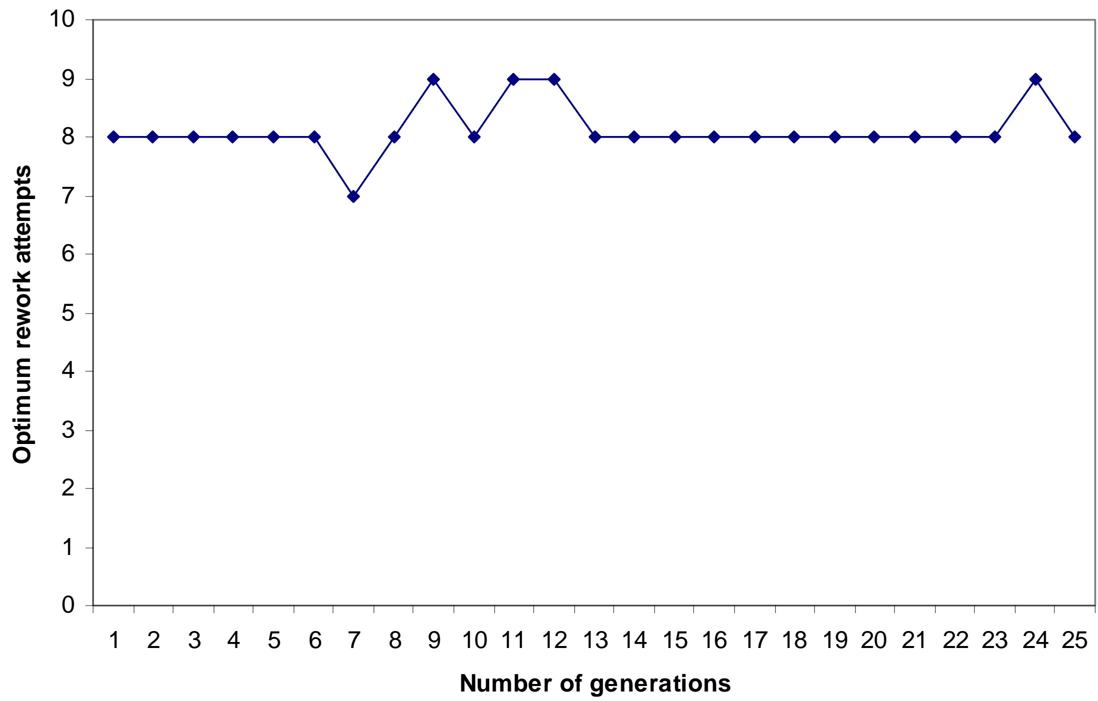


Figure 5.13: Rework attempts at the minimum yielded cost for the single TDR MCM process flow.

Table 5.16: Optimization of the MCM assembly process flow with the rework attempts fixed at one (die cost is $3).

Number of generations	Optimum yielded cost ($)	Mean of yielded cost ($)	Optimum fault coverage	Rework attempts
1	921.49	224024.52	0.49	1
2	921.49	38103.94	0.49	1
3	921.47	927.25	0.48	1
4	921.46	38103.07	0.48	1
5	921.46	923.88	0.48	1
6	921.46	923.66	0.48	1
7	921.46	926.45	0.48	1
8	921.46	926.41	0.48	1
9	921.46	38100.50	0.48	1
10	921.46	930.72	0.48	1
11	921.46	928.46	0.48	1
12	921.46	926.65	0.48	1
13	921.46	38101.57	0.48	1
14	921.46	925.24	0.48	1
15	921.46	926.52	0.48	1
16	921.46	925.79	0.48	1
17	921.46	932.58	0.48	1
18	921.46	925.07	0.48	1
19	921.46	38096.52	0.48	1
20	921.46	927.86	0.48	1
21	921.46	925.46	0.48	1
22	921.46	928.85	0.48	1
23	921.46	38104.64	0.48	1
24	921.46	926.22	0.48	1
25	921.46	38099.30	0.48	1

Table 5.17: Optimization of the MCM assembly process flow with the rework attempts optimized (die cost is $3).

Number of generations	Optimum yielded cost ($)	Mean of yielded cost ($)	Optimum fault coverage	Rework attempts
1	862.68	149628.16	0.60	8
2	862.68	882.78	0.60	8
3	862.68	865.18	0.60	8
4	862.68	867.06	0.59	8
5	862.68	38041.73	0.59	8
6	862.68	38038.13	0.59	8
7	862.68	876.17	0.59	7
8	862.68	38040.37	0.59	8
9	862.68	862.75	0.59	9
10	862.68	866.72	0.59	8
11	862.68	75217.18	0.59	9
12	862.68	878.75	0.59	9
13	862.68	865.49	0.59	8
14	862.68	38052.18	0.59	8
15	862.68	868.01	0.59	8
16	862.68	880.77	0.59	8
17	862.68	880.66	0.59	8
18	862.68	873.01	0.59	8
19	862.68	864.95	0.59	8
20	862.68	870.22	0.59	8
21	862.68	38051.58	0.59	8
22	862.68	869.76	0.59	8
23	862.68	864.60	0.59	8
24	862.68	75216.52	0.59	9
25	862.68	801.07	0.59	8

5.6 Summary

The objective of this chapter was to apply the RCGAs developed in Chapter 4 to a real case — the MCM assembly process discussed in [74]. Abadir *et al.* [74] only enumerated three different test strategies with a combination of fault coverages for different input yields of die. The GAs optimization used in the MCM assembly process successfully generates a minimum yielded cost optimum that is an average of 6.85% lower than the calculated values using the test strategies by selecting specific fault coverages as in [74] for various incoming die yield. It would be almost impossible to predict the yielded cost of systems after more than hundreds of manufacturing process steps only by manually selecting test coverages without optimization techniques included. Furthermore, the optimization of rework attempts is applied to the MCM example.

Chapter 6

Optimization of TDR Under Uncertain Inputs

Chapter 4 demonstrates the optimization algorithms to determine the location(s) and characteristics of TDR operations for specific values of the inputs describing the process. Chapter 5 applies the GA-based optimization to a real MCM assembly process under certain inputs of die yield and cost. In this chapter, the optimization algorithm to process uncertainties of inputs is discussed and Monte Carlo methods will be integrated inside the optimization loop to sample values from the input distributions.

6.1 Uncertain Inputs

A consideration for the evaluation of electronic systems cost is that many of the necessary inputs have large uncertainties, e.g., the yields of individual process steps or the costs of assembly. As an example, the simple process flow in Figure 5.1 has uncertain inputs of incoming cost and yield represented by corresponding probability distributions. After the parts go through the process flow, the outgoing cost and yield will also be

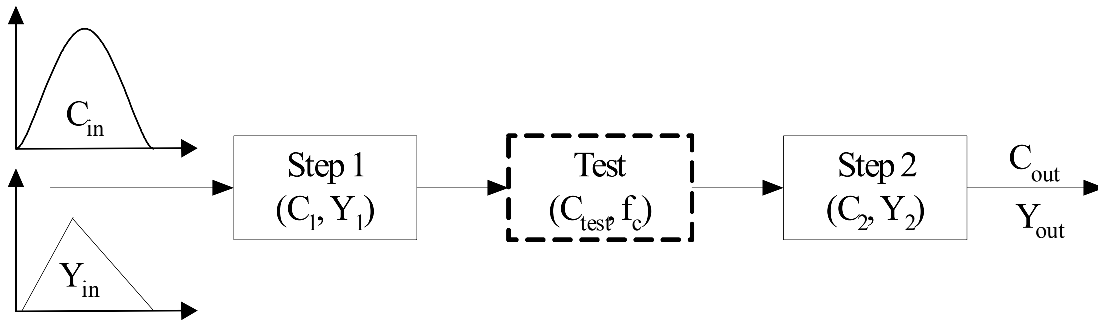


Figure 6.1: A simple process flow example with uncertain inputs.

distributions, i.e., the final yielded cost of the process flow will not be a single value but a probability distribution.

One way to optimize the yielded cost objective function under uncertain inputs is to use a statistical method to sample values from the input probability distributions. The yielded costs of the process flow are then calculated for each sampled input value. After the distribution of yielded costs is derived some statistical value (e.g., the mean of the distribution) can be chosen as the fitness function used in the GAs optimization. The Next section will introduce Monte Carlo methods into the optimization model.

6.2 Monte Carlo Methods

Numerical methods that are known as Monte Carlo methods can be loosely described as statistical simulation methods, where statistical simulation is defined in general terms to be any method that utilizes sequences of random numbers to perform the simulation. The essence of the methods is the game of the chance whose behavior and outcome can be used to study interesting phenomena [89, 90]. The Monte Carlo method consists of creating a statistical situation (probability space) in which a required answer is the expected value of some random variable; the answer is then estimated by sufficient

sampling of this random variable to ensure adequate accuracy. This is typically done using a computer model with so-called "pseudo-random" or "quasi-random" number generators.

The inputs to the process flow discussed in this chapter will have probability distributions from which the Monte Carlo methods are applied to sample the values. The yielded cost is calculated from each sampling value with the test characteristics generated by the GAs. The next section will explore the details of integration of Monte Carlo methods to the GAs optimization algorithms.

6.3 Application of Monte Carlo Method in Optimization of TDR Operations for Uncertain Inputs

A Monte Carlo method will be integrated into the optimization model proposed in Chapter 4 to obtain robust optimum solutions to the objective function. Figure 6.2 presents the proposed process for applying Monte Carlo in the optimization of TDR operations while the inputs are uncertain.

In Figure 6.2, the population of RCGAs that represent groups of initialized variables, i.e., sets of fault coverages or rework attempts, is created in the initialization process. According to the procedure of GAs described in Figure 4.2, the evaluation of the objective function should be done to determine the fitness of the variables — the value of the objective function.

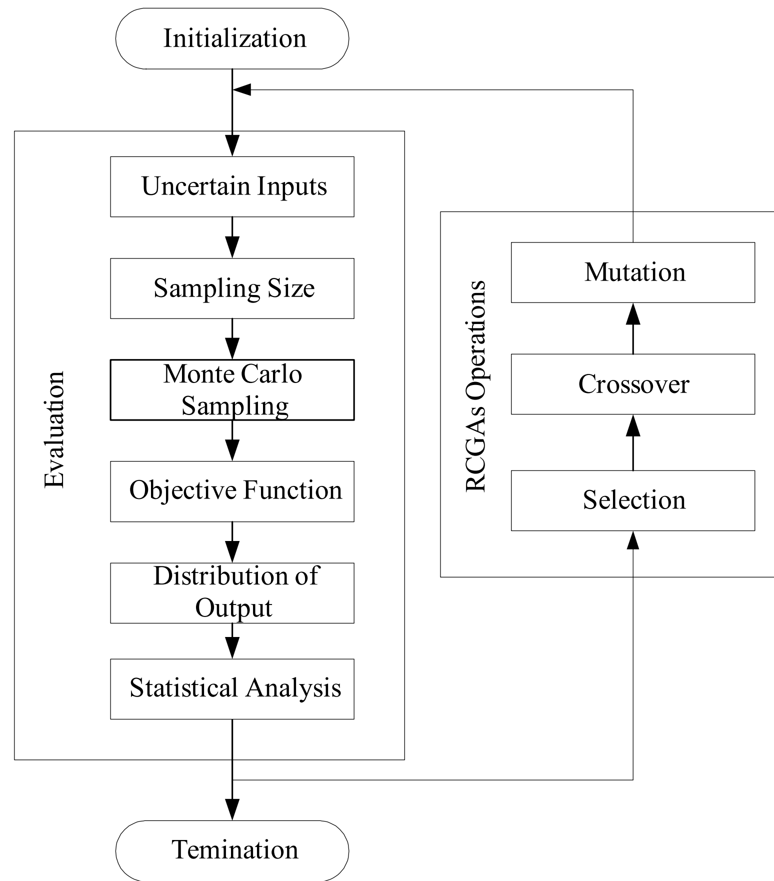


Figure 6.2: Application of Monte Carlo methods in the optimization of TDR operations for uncertain inputs.

In the optimization model of Chapter 4 the objective function is calculated with certain inputs. However, if there are uncertain inputs, a Monte Carlo method is applied to sample from the input distributions. The sampling size depends on the precision of the estimation of the mean or the other statistical characteristics [91, 92]. After the calculation of the objective function for the specified number of samples, the distribution of yielded cost of the process flow is determined for the specific set of fault coverages or the corresponding variables to be optimized. A value can be chosen from the distribution of yielded costs to represent the fitness of the objective function for the uncertain

inputs.[32] Then the RCGAs continue to evolve and calculate the value of the objective function by the GAs' operations until the algorithm terminates. After the optimization is performed, the optimum solutions minimize the yielded cost given the uncertain inputs are obtained.

Figure 6.2 presents the optimization model in Chapter 4 with improvements to accommodate the Monte Carlo analysis in the calculation of objective function. The improved GAs optimization model integrates the Monte Carlo method inside its evolution process. This makes sense because the fitness of the objective function is the mean of distribution of yielded cost for a specified group of variables to be optimized. The optimum of yielded cost obtained from the GAs algorithms is the lowest value under the group of optimum characteristics given the uncertain inputs.

The inclusion of the Monte Carlo analysis inside the optimization algorithm can be described by Figure 6.3. During the RCGAs optimization process, the population made up of chromosomes (e.g., array of fault coverages to be optimized) is generated by the algorithm then compare with each other by determining the value of the fitness function. The fitness function is derived from the multi-variable objective function to minimize the yielded cost of process flow. If there is no uncertain input to the process flow, the value of fitness function can be calculated by traversing the whole process flow using the searching algorithms developed in Chapter 3 under certain inputs. While the uncertain inputs are introduced into the process flow, the calculation process can't be performed without handling the probability distributions of inputs correctly. The yielded cost of the

[32] The particular value chosen from the distribution to serve as the fitness is not clear. Candidates include the mean, the most likely value and the median. In this dissertation, the mean of distribution of yielded cost is chosen to represent the fitness of objective function.

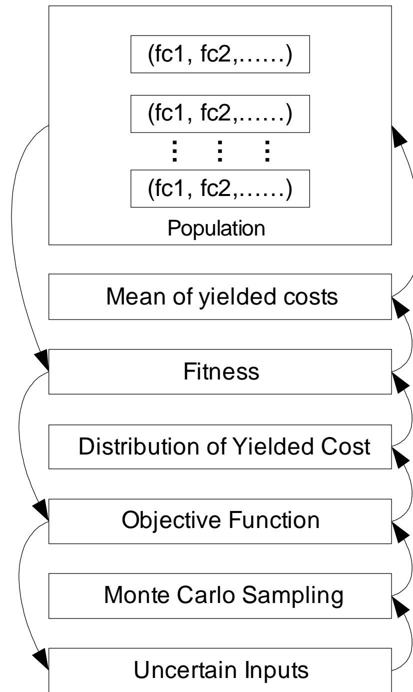


Figure 6.3: Monte Carlo sampling inside the GAs optimization of TDR operations.

process flow is no longer a certain value, but a probability distribution. Here the Monte Carlo analysis is applied inside the optimization process to account for the uncertain inputs. To derive the distribution of yielded cost for a chromosome or an array of fault coverages that need to be optimized, we pick the values from the distribution of uncertain inputs (yield of chip, yield of substrate, etc.) using Monte Carlo sampling and calculate the yielded cost of process flow for each combination of uncertain values.[33] After running a sufficient number of samples, the values represent the distribution of objective function. The mean of the random values of objective function is then calculated as the fitness function for each chromosome in a population. By determining the value of the mean of yielded cost, the RCGAs choose the chromosomes with the best value of the fitness

[33] The calculation for each set of sampling values from inputs distributions should use the same searching process for certain inputs.

function and do the predefined genetic operations — crossover, mutation to implement the optimization process as described in Chapter 4.

Because the objective function used in this dissertation is the minimization of the yielded cost of the products, which is the effective cost of good parts or assemblies after passing through a process, a variance analysis doesn't make sense for the cost evaluation even though there are uncertain inputs involved in the process. Actually, the total yielded cost or the average yielded cost (equal to the total yielded cost divided by the sample size) for a statistically significant number of instances of a product is used to determine the set of fault coverages that lead to the lowest cost for the product. Hence, although the variances of the yielded cost distributions are different for the various set of fault coverages but have the same average yielded cost, we still treat them as equivalent, which means whatever the set of fault coverages, the total yielded cost for the whole production run after the entire manufacturing process is the same (irrespective of the variance). Therefore, the mean of the yielded cost distribution is sufficient to determine the set of fault coverages that should be chosen to minimize yielded cost.

Compared with the RCGAs for certain inputs in Chapter 4 (in Table 6.1), the major change in the optimization process for uncertain inputs lies in the calculation of fitness function. For each chromosome representing the variables to be optimized in the population, the GAs for certain inputs use the value calculated from the multi-variable objective function — a single value as the fitness function, while the GAs for uncertain inputs calculate the mean of the objective function distribution derived by the Monte Carlo sampling and choose it as the value to select the best chromosome.

Table 6.1: Comparison between RCGAs for certain and uncertain inputs.

	RCGAs for certain inputs	RCGAs for uncertain inputs
Inputs	Single value	Distribution
Fitness function	Single value of yielded cost — objective function	Mean of yielded costs — the distribution of objective function
Monte Carlo sampling	N/A	Integrated
Computing Time	Short	Longer

When the Monte Carlo analysis is incorporated inside the optimization process, the sample size should first be estimated to make sure that a statistically valid solution can be obtained. The next section will introduce the method used to determine the sampling size.

6.3.1 Estimation of the Number of Samples

Many different approaches to estimate the sample size exist and the method used depends on what you want from the analysis:

- Precision of the estimate of the mean.
- Precision of the estimate of the cumulative distribution.

Approach 1: Estimate of the mean

Suppose we have r Monte Carlo runs producing solutions z_1, z_2, z_3, ……, z_r, the mean of the solutions can be derived from,

$$\mu = \frac{1}{r}\sum_{i=1}^{r} z_i \qquad (6.1)$$

The standard deviation is given by,

$$\sigma^2 = \frac{1}{r-1}\sum_{i=1}^{r}(z_i - \mu)^2 \tag{6.2}$$

The sampling variance of the mean is calculated using Equation (6.3):

$$\mathrm{Var}|\mu| = \frac{\sigma^2}{r} \tag{6.3}$$

From the central limit theory [92], the standard error on the mean is,

$$\sqrt{\mathrm{Var}|\mu|} = \frac{\sigma}{\sqrt{r}} \tag{6.4}$$

If the condition of $\frac{\sigma}{\sqrt{r}} \leq 0.01\mu$ can be satisfied (the standard error on the mean is equal to or less than the 0.01% of the mean), the Monte Carlo sampling is stopped.

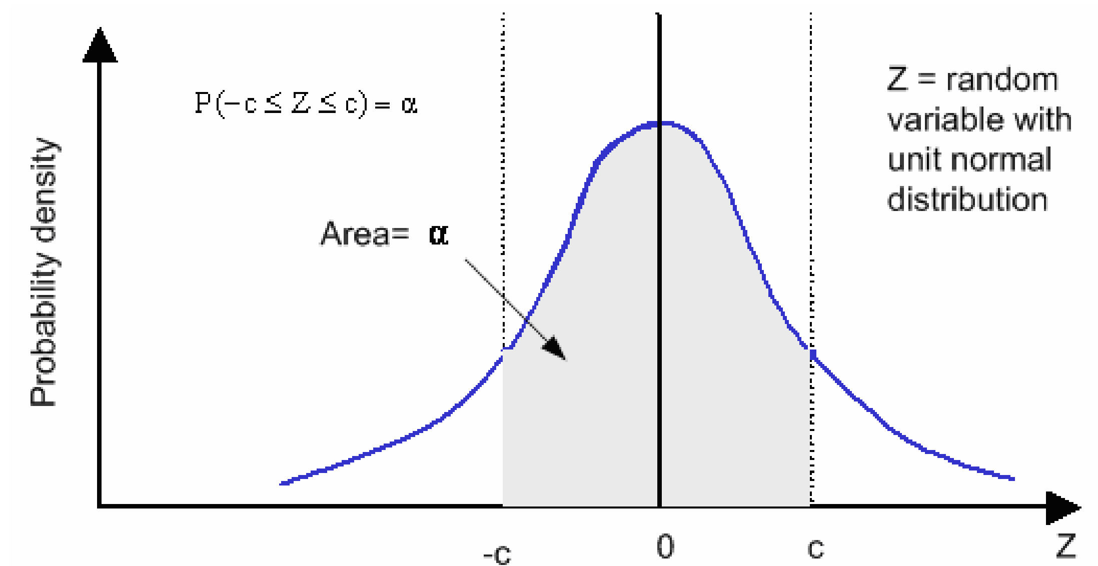


Figure 6.4: Confidence level for Monte Carlo size estimation

Approach 2: Estimate of sample size based on a given confidence level

If given a confidence level of α is chosen for the mean of the generated solutions in Approach 1, the range $(\mu - c\frac{\sigma}{\sqrt{r}}, \mu + c\frac{\sigma}{\sqrt{r}})$ has an α probability of containing μ which is described in Figure 6.4.

If the desired confidence level is smaller than ω, then we have,

$$2c\frac{\sigma}{\sqrt{r}} < \omega \tag{6.5}$$

The sample size can be estimated by,

$$r > (\frac{2c\sigma}{\omega})^2 \tag{6.6}$$

For the given confidence level, the estimation process in Figure 6.5 can be followed.

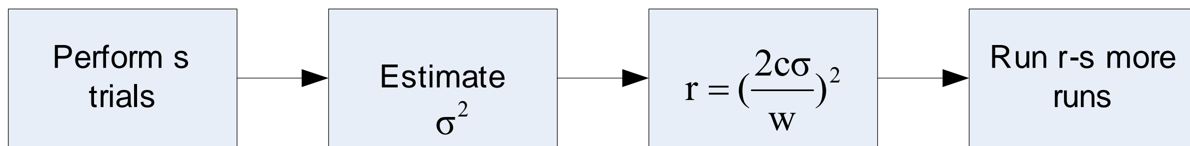


Figure 6.5: Estimation of sample size for a given confidence level.

6.3.2 Monte Carlo Implementation

After the sample size has been determined, the following procedure was used to implement the Monte Carlo analysis [89]:

1. Inputs are obtained in the form of distributions
2. A random number (r1) is generated between 0 and 1
3. This random number (r1) is used to choose a value from the distribution describing the first input.

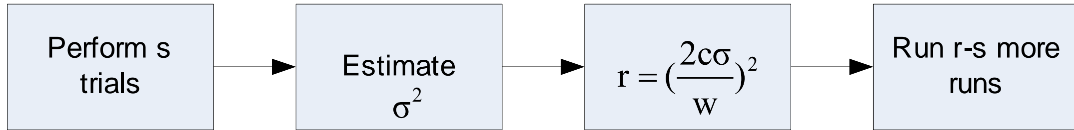


Figure 6.6: MCM assembly process flow with uncertain incoming yield of single die.

4. Values are chosen from the distributions representing all the inputs by repeating steps 2 and 3.
5. These random numbers (inputs) are then accumulated according to the equations specified in the algorithms to get the final result (one sample).
6. Steps (2)-(5) are repeated u times (u is the number of samples to be used for the Monte Carlo analysis)
7. The mean and variance of the result can then be calculated.

The simulation of Monte Carlo sampling of the yielded cost of MCM assembly process flow (in Figure 6.6) is shown in Figure 6.7 given a normal distribution (mean=0.9, standard deviation=0.3, and sample size=600) of die yield.

From Figure 6.7 and Table 6.2, the stopping criteria in Approach 1 can be satisfied for most cases. So (6.7) is added to determine the number of samples.

$$\frac{\mu_m - \mu_{m-1}}{Y_{C_m}} \leq 0.001 \tag{6.7}$$

which means that the difference between the two means from Monte Carlo sampling is less than 0.1 % of the yielded cost (usually less than \$0.1). For most test cases considered in this dissertation, Monte Carlo sampling stops within of 400 samples by satisfying the criteria of (6.7).

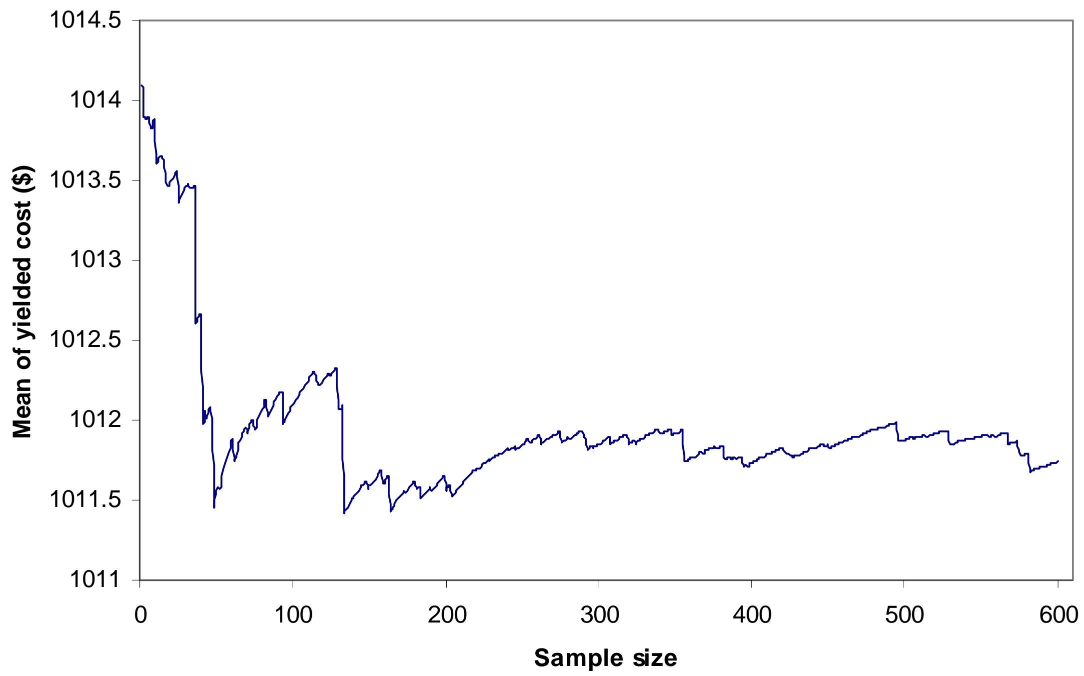


Figure 6.7: Monte Carlo sampling size estimation for the normal distribution of die yield.

Table 6.2: Monte Carlo sampling for a normal distribution of die yield.

Sample size	Yielded cost ($)	Mean (μ, $)	Standard deviation	Standard error of the mean	0.01 μ
1	1014.099	1014.099	2.97E-13	1.71E-08	10.14099
2	1014.007	1014.053	0.00417	0.002028	10.14053
3	1013.572	1013.893	0.153892	0.01232	10.13893
4	1013.906	1013.896	0.000131	0.00036	10.13896
5	1013.841	1013.885	0.002419	0.001545	10.13885
6	1013.952	1013.896	0.003731	0.001918	10.13896
7	1013.422	1013.828	0.192776	0.013789	10.13828
8	1013.846	1013.831	0.000274	0.00052	10.13831
9	1014.131	1013.864	0.080129	0.00889	10.13864
10	1014.109	1013.889	0.054133	0.007307	10.13889
⋮	⋮	⋮	⋮	⋮	⋮
591	1011.308	1011.706	0.158693	0.012524	10.11706
592	1013.444	1011.709	3.013262	0.054575	10.11709
593	1013.917	1011.713	4.863829	0.069336	10.11713
594	1013.624	1011.716	3.646496	0.060036	10.11716
595	1014.138	1011.72	5.854553	0.076071	10.1172
596	1013.93	1011.724	4.874214	0.06941	10.11724
597	1014.072	1011.728	5.506309	0.073773	10.11728
598	1014.059	1011.732	5.422624	0.07321	10.11732
599	1013.806	1011.735	4.295354	0.065158	10.11735
600	1014.102	1011.739	5.59118	0.074339	10.11739

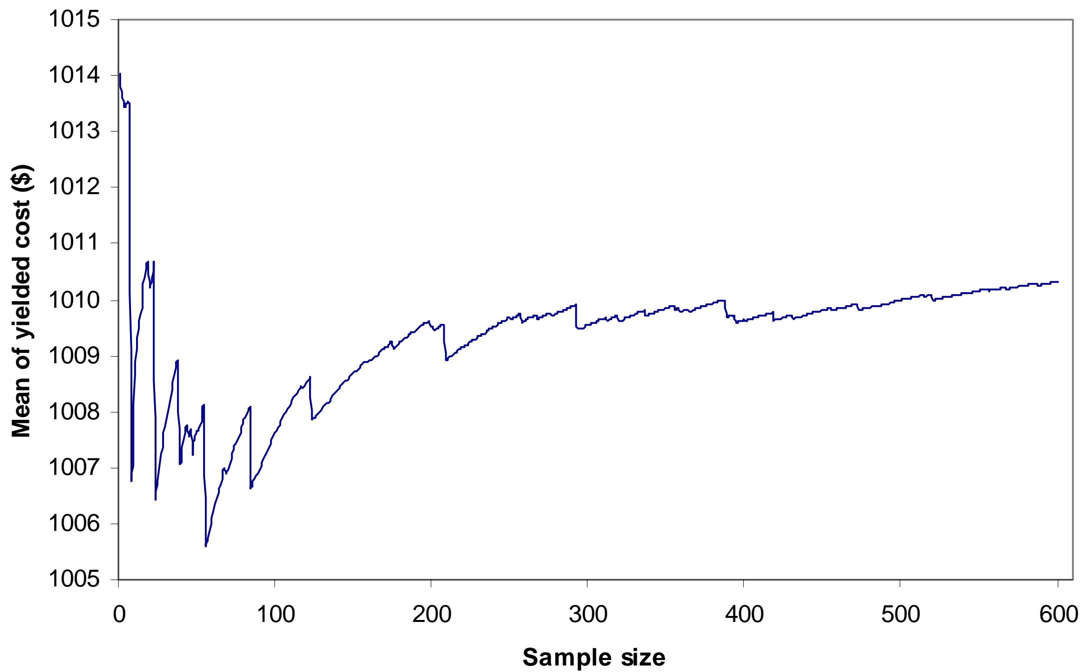


Figure 6.8: Monte Carlo sampling size estimation
for the triangular distribution of die yield**.**

Figure 6.8 shows another Monte Carlo sampling process for a triangular distribution of single chip yield (mean=0.9, low value=0.7 and high value=0.99). The stopping criterion of (6.7) is also applicable to the input of a triangular distribution.

Table 6.3: Monte Carlo sampling for a triangular distribution of die yield.

Sample size	Yielded cost	Mean (μ, $)	Standard deviation	Standard error of the mean	0.01 μ	Step yield
1	1014.021	1014.021	4.19E-11	2.03E-07	10.14021	0.003665
2	1013.22	1013.621	0.320665	0.017786	10.13621	2.88E-06
3	1013.243	1013.495	0.094905	0.009677	10.13495	2.46E-05
4	1013.412	1013.474	0.005211	0.002268	10.13474	0.000324
5	1013.217	1013.423	0.053064	0.007236	10.13423	4.96E-07
6	1014.047	1013.527	0.324465	0.017892	10.13527	0.00401
7	1013.267	1013.49	0.057999	0.007565	10.1349	5.32E-05
8	960.1955	1006.828	2485.223	1.571105	10.06828	0.212711
9	1013.507	1007.57	39.65842	0.198395	10.0757	0.000578
10	1013.22	1008.135	28.73512	0.168829	10.08135	2.88E-06
⋮	⋮	⋮	⋮	⋮	⋮	⋮
590	1013.219	1010.261	8.765162	0.093146	10.10261	1.78E-06
591	1013.391	1010.266	9.781729	0.098399	10.10266	0.000277
592	1013.516	1010.271	10.54748	0.102178	10.10271	0.000606
593	1013.411	1010.277	9.840645	0.098694	10.10277	0.000323
594	1013.691	1010.283	11.63528	0.107317	10.10283	0.001255
595	1013.958	1010.289	13.48348	0.115526	10.10289	0.012142
596	1013.24	1010.294	8.69578	0.092775	10.10294	2.1E-05
597	1013.594	1010.299	10.87452	0.103748	10.10299	0.000864
598	1013.494	1010.305	10.18944	0.100427	10.10305	0.017548
599	1013.813	1010.31	12.28674	0.110278	10.1031	0.001891
600	1013.265	1010.315	8.715516	0.092879	10.10315	5.12E-05

6.4 Real Case Analysis

The improved GAs as described in Figure 6.2 have been applied to the real cases of the MCM assembly process discussed in Chapter 5 to validate whether the algorithm can find the optimal solutions to minimize the yielded cost given uncertain inputs. The validation process is divided into two steps: the first one is to verify the single TDR of the MCM assembly case in Figure 6.6 and the second one is extended the optimization algorithm to a multiple-TDR case.

6.4.1 MCM Assembly Process Flow with Single TDR and Uncertain Inputs

In this section, the GAs with Monte Carlo integrated will be applied to the MCM assembly process flow in Figure 6.6 to optimize the objective function defined in Chapter 3 with uncertain inputs. For the comparison purposes, the optimization results of process flow in Figure 6.6 without uncertain inputs are also given to demonstrate how the improved GAs handle the optimization process with the consideration of probability distribution of inputs. Figure 6.9 shows that the optimum yielded cost doesn't vary with the evolution of GAs. For the simple process flow with only one TDR, the RCGAs find the optimum solution in the very first generation (see Table 6.4).

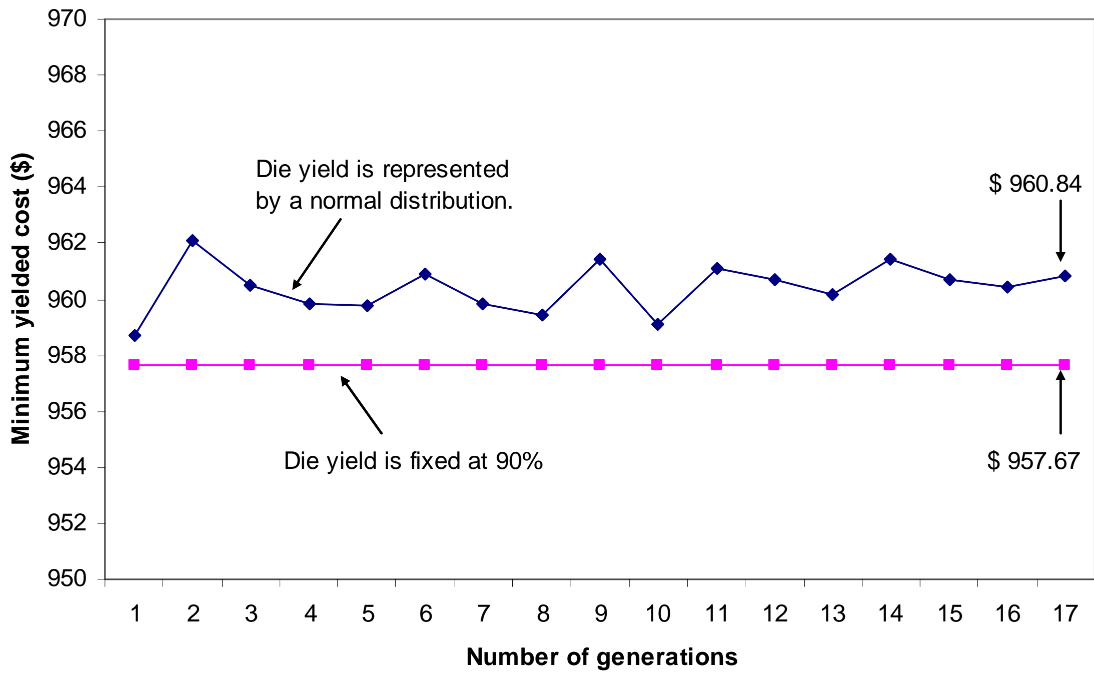


Figure 6.9: Comparison of the minimum yielded cost
for single value and a normal distribution of incoming die yield.

Table 6.4: Optimization of fault coverage in single TDR MCM assembly process flow in Figure 6.6 with the cost of incoming die yield is 90%.

Number of generations	Optimum yielded cost ($)	Mean of yielded cost ($)	Optimum fault coverage
1	957.67	3331.18	0.62
2	957.67	969.22	0.62
3	957.67	968.02	0.62
4	957.67	983.50	0.62
5	957.67	3269.59	0.62
6	957.67	963.73	0.62
7	957.67	5578.54	0.62
8	957.67	3267.41	0.62
9	957.67	3269.03	0.62
10	957.67	3270.63	0.62
11	957.67	3268.79	0.62
12	957.67	966.12	0.62
13	957.67	981.33	0.62
14	957.67	3269.25	0.62
15	957.67	5585.89	0.62
16	957.67	971.17	0.62
17	957.67	969.95	0.62
18	957.67	3273.55	0.62
19	957.67	962.82	0.62
20	957.67	5580.44	0.62
21	957.67	3275.61	0.62
22	957.67	3267.28	0.62

Figure 6.9 also describes the optimization process of yielded cost of process flow in Figure 6.6 given the normal distribution of incoming die yield (shown in Figure 6.10). The improved GA's search for the optimum value is not as convergent as it is under certain input shown in Figure 6.9. In the optimization process with Monte Carlo analysis included, each calculation of the fitness function depends on the sampling results from the input distribution(s). Because of random sampling from the probability distribution, the calculated mean of the objective function in the previous generation is varied from the one computed in the next generation although the test was at the same fault coverage. The optimum yielded cost for a set of fault coverages in one generation will change (lower or higher) when passed into the next generation. More samples can improve the accuracy of the optimums and lead to fast convergence [93-95] but also take more execution time.

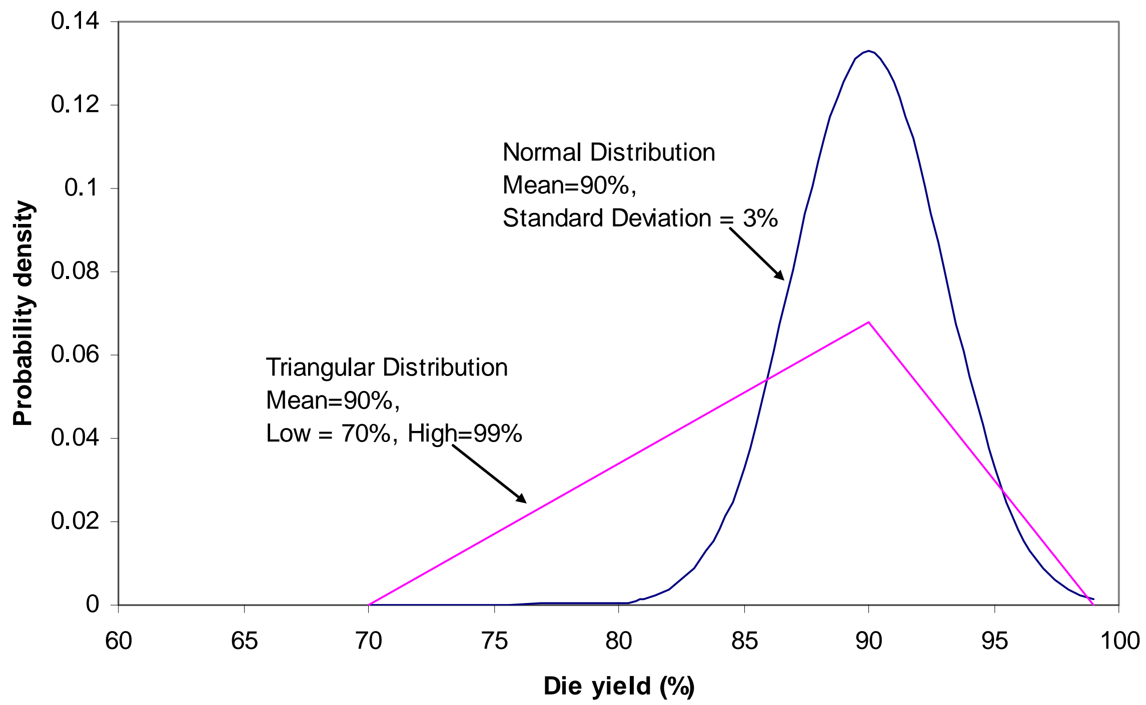


Figure 6.10: Distributions of die yield.

Figure 6.11 compares the optimum fault coverage for the single value input to the normal distribution input of incoming die yield to process flow in Figure 6.6. The optimum fault coverage for the minimum yielded cost is not convergent to a straight line when the Monte Carlo method is included, which also results from the inaccuracy of sampling from input distribution(s).

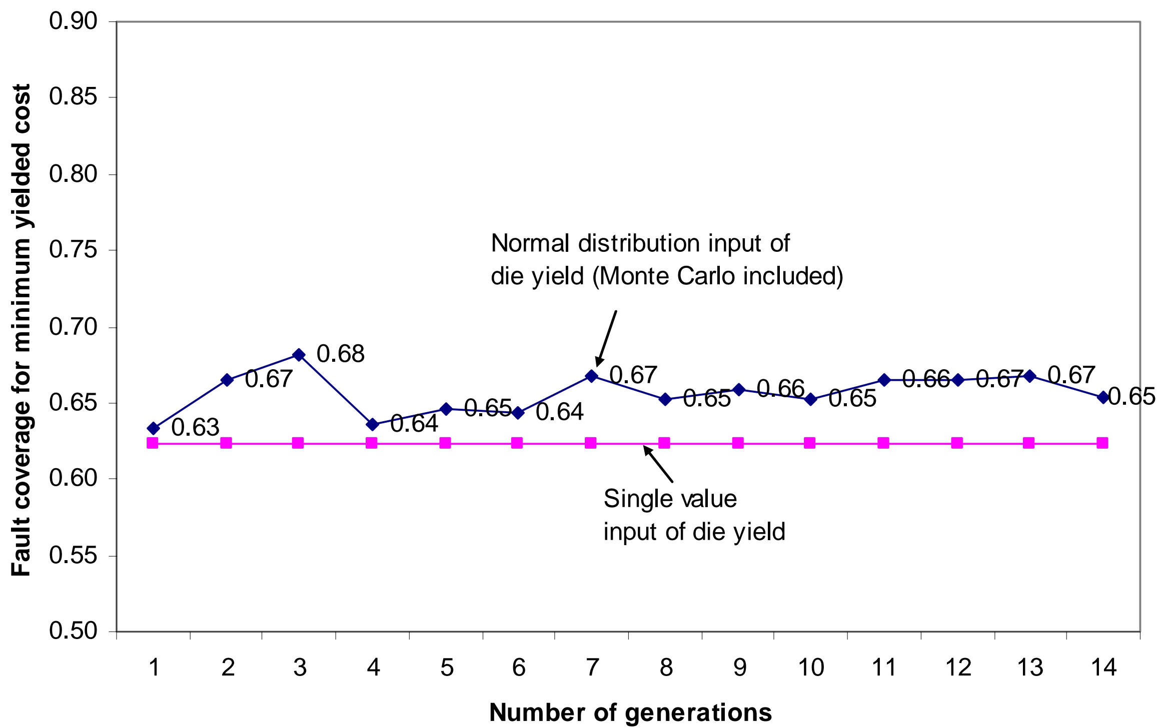


Figure 6.11: Comparison of the optimum fault coverage for single value input and a normal distribution of incoming die yield.

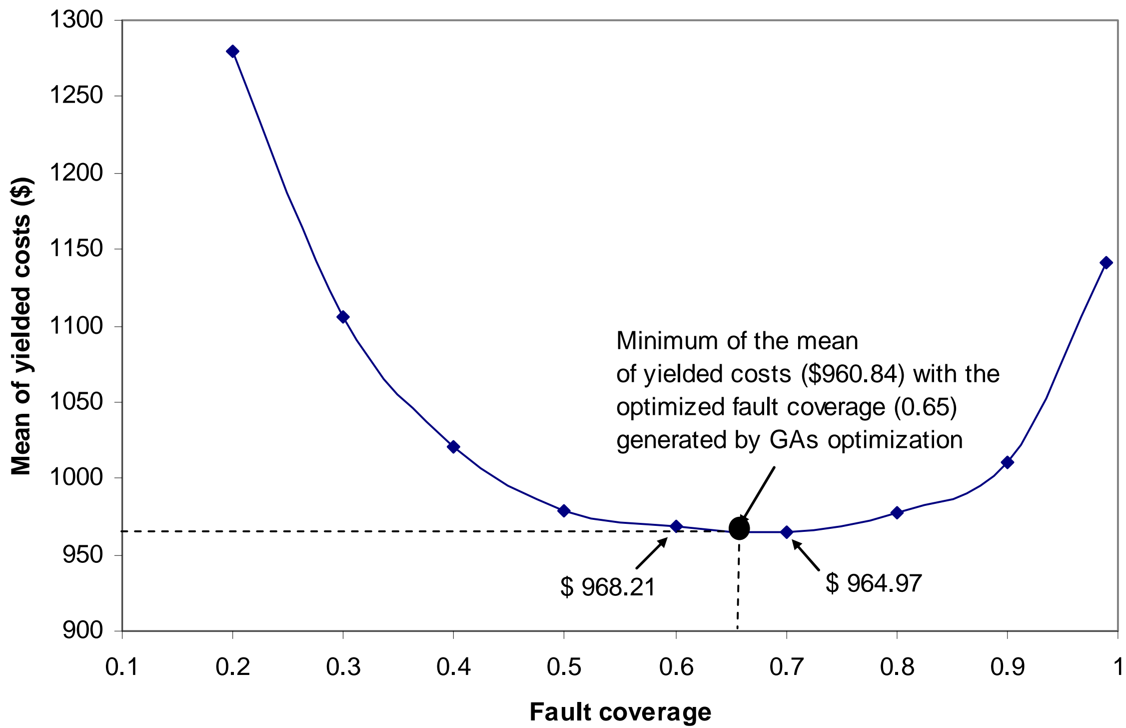


Figure 6.12: Mean of yielded costs varies with the various fault coverage.

After the optimum fault coverage (0.65, in Figure 6.11) that leads to the minimum mean of yielded cost ($960.84, in Figure 6.9) is generated by the GAs optimization under a normal distribution of incoming die yield, the results can be qualitatively verified using the Trichy *et al.* model discussed in Chapter 2. The procedure of verification can be described as following:

1. Choose a specific value from the full range of fault coverages (above 10% threshold, 0.2, 0.3, etc.).
2. Calculate the mean of yielded costs for each designated fault coverage by running the Trichy *et al.* model with a Monte Carlo method included to sample the value from input yield distribution (See Table 6.5).

3. Plot the mean of yielded costs with its corresponding fault coverage (see Figure 6.12).
4. Check whether the minimum mean of yielded costs with the optimized fault coverage generated by GAs optimization is the minimum value on the plot.

Figure 6.12 demonstrates that the optimum yielded cost ($960.84) with its corresponding fault coverage (0.65) generated by the RCGAs is in fact the minimum given by the input distribution. From Figure 6.12, we can see the minimum of the mean should be lower than $964.97 with the specified fault coverage in range of 0.6 to 0.7, which demonstrates the minimized yielded cost of $960.84 in Figure 6.10 and the optimum fault coverage of 0.65 are the exact values that we need to find.

Alternatively, when the triangular distribution shown in Figure 6.10 is used as the die yield input to the process flow in Figure 6.6. The mean of the triangular distribution is lower than the one corresponding to the normal distribution.

Table 6.5: Mean of yielded costs with the corresponding fault coverage calculated using Trichy *et al.* model with a Monte Carlo method included.

Fault coverage	Mean of yielded costs ($)
0.2	1280.15
0.3	1105.35
0.4	1020.66
0.5	978.72
0.6	968.21
0.7	964.97
0.8	977.96
0.9	1011.06
0.99	1140.70

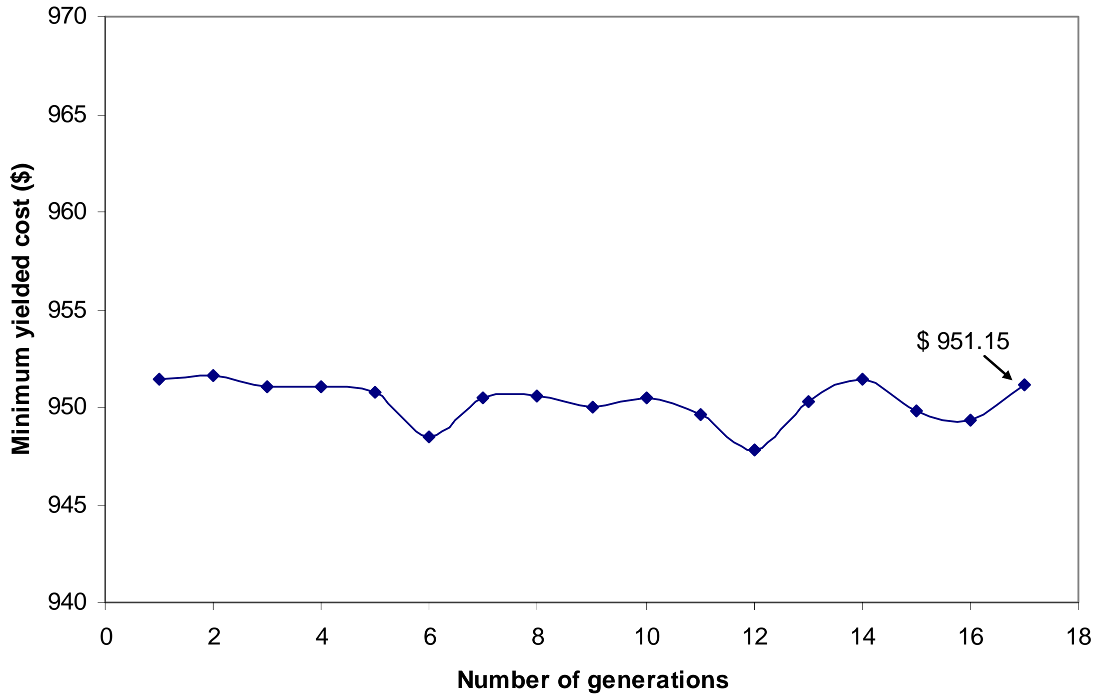


Figure 6.13: Minimum yielded cost when the single die yield is represented by a triangular distribution.

The optimization of yielded cost for the triangular distribution input is shown in Figure 6.13. Figure 6.14 gives the comparison of the optimum fault coverage for the single value input and uncertain triangular distribution incoming die yield. The optimum fault coverage of 0.59 generated by RCGAs can be verified to produce the minimum yielded cost of $951.15 by Figure 6.15.

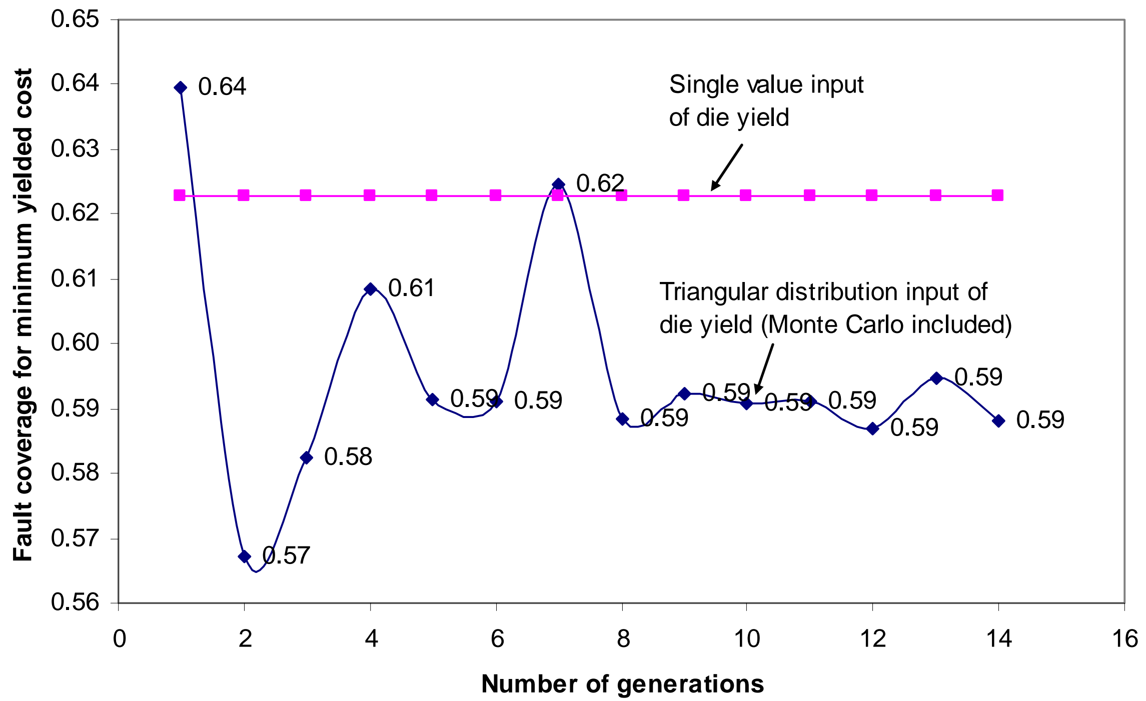


Figure 6.14: Comparison of the optimum fault coverage for single value input and a triangular distribution of incoming die yield.

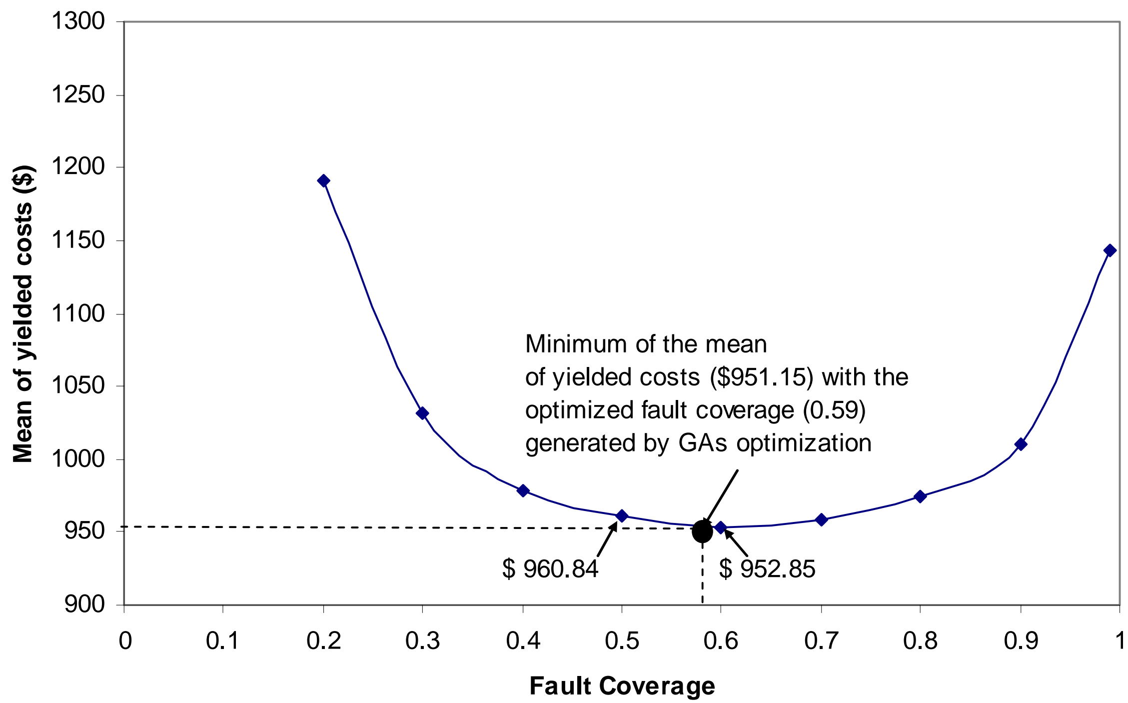


Figure 6.15: Mean of yielded costs varies with the various fault coverage under the triangular distribution input of die yield.

The improved GAs are applicable to handling the uncertain inputs characterized by various probability distributions and can find the minimum value of fitness function in a limited number of generations of evolution. The optimum yielded cost depends on the distribution shape. In the example cases performed in this section, the mean of the triangular distribution is lower than the mean of the normal distribution, which projects that the final minimums of the yielded cost from various input distributions should vary with each other as Figure 6.16 describes.

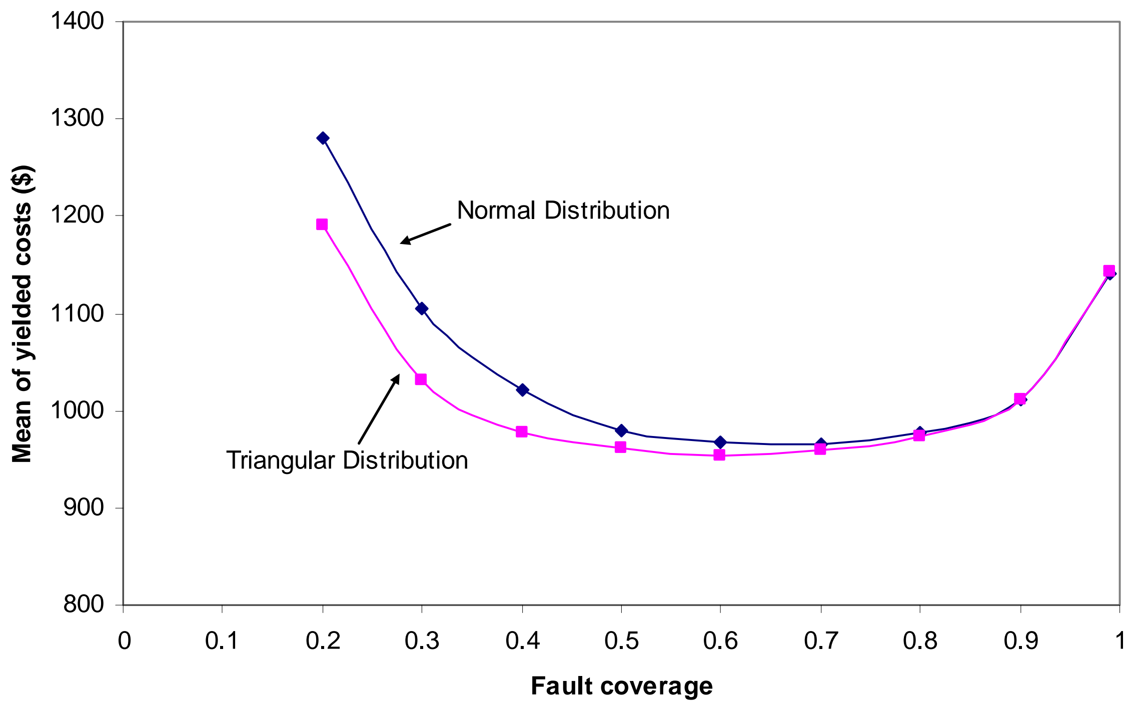


Figure 6.16: Comparison of the mean of yielded cost for different input distributions.

6.4.2 MCM Assembly Process Flow with Multiple TDRs and Uncertain Inputs

This section discusses the application of the optimization algorithm including the Monte Carlo method to the MCM assembly process flow extended to include multiple TDR operations. The single TDR MCM process flow in Figure 6.6 has been extended by adding another TDR after the final assembly of the MCM (shown in Figure 6.17). All of the incoming parameters for the assembly steps are the same as ones used in the single-TDR MCM assembly except the cost and yield of the new step, 'Final Assembly'. The cost of Final Assembly has been set to \$200 and the step yield is 99%.

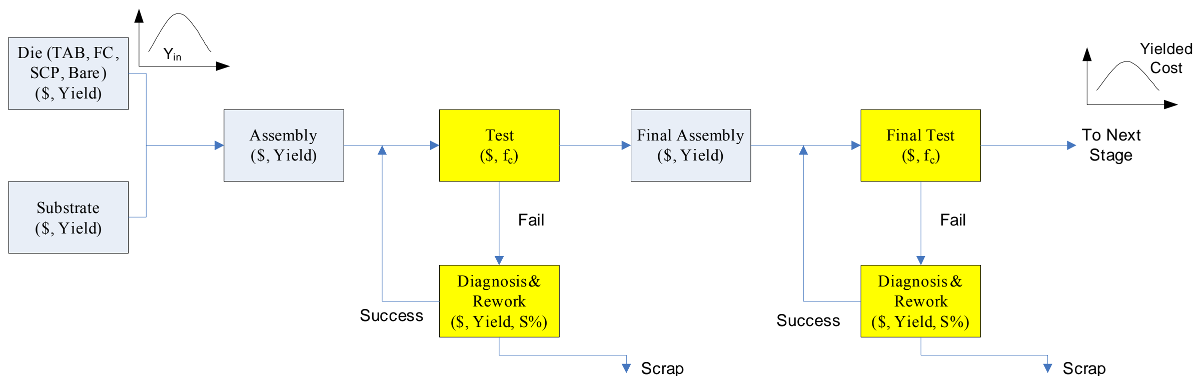


Figure 6.17: MCM assembly process flow with multiple TDRs

Applying the GAs with Monte Carlo methods included to the process flow in Figure 6.18, the minimum yielded cost is found in 15 generations (less than twice the running time of the single-TDR case) in Table 6.6. The complexity of optimization algorithm relies more on the sample size for the Monte Carlo analysis than the structure of the process flow. The optimization process of multi-TDR MCM assembly for normally distributed incoming die yield is shown in Figure 6.18 and Figure 6.19 gives the optimum fault coverages at the minimum yielded cost for each test operation. The mean of the

optimized fault coverages is almost the same value as the one of single-TDR MCM assembly process (see Figure 6.12). Because the step yield of Final Assembly is very high, the two test operations function as a single test, i.e., the two test operations can be replaced by one.[34]

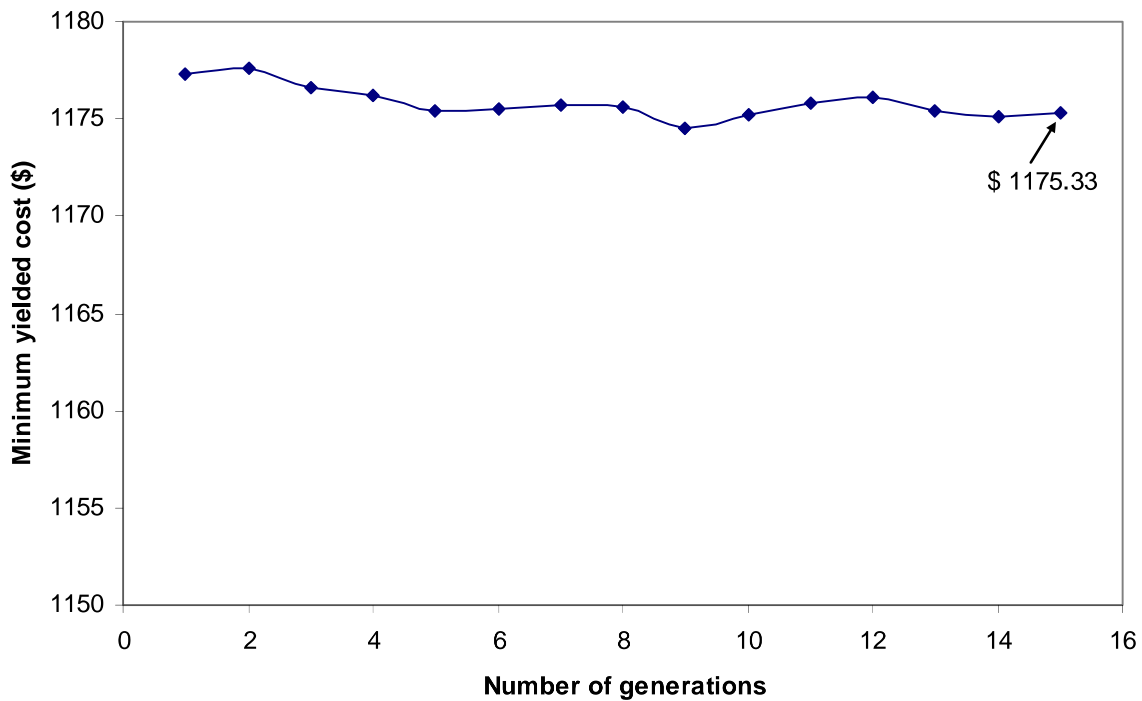


Figure 6.18: Minimum yielded cost of MCM assembly process with multi-TDR and a normal distribution of input die yield.

[34] There is no expert suggestions included in the optimization algorithms discussed in this dissertation, e.g., if the yield of one step is very high (>= 95%), the two tests placed before it and after it can be combined to one. With more heuristics added into the algorithms, the optimization searching will be more intelligent and practical [96-98].

Table 6.6: Optimization of fault coverage
in multi-TDR MCM assembly process flow in Figure 6.17

Number of generations	Optimum yielded cost ($)	Mean of yielded cost ($)	Optimum fault coverage of Test	Optimum fault coverage of Final Test
1	1177.28	1294.73	0.42	0.83
2	1177.56	1208.86	0.46	0.79
3	1176.57	1187.16	0.48	0.76
4	1176.25	1185.41	0.48	0.76
5	1175.38	1180.75	0.49	0.76
6	1175.47	1185.89	0.49	0.73
7	1175.76	1180.99	0.44	0.80
8	1175.58	1181.21	0.50	0.76
9	1174.53	1181.90	0.47	0.75
10	1175.22	1182.36	0.48	0.76
11	1175.85	1183.14	0.46	0.75
12	1176.13	1179.47	0.46	0.76
13	1175.40	1182.41	0.47	0.75
14	1175.14	1182.19	0.46	0.76
15	1175.33	1182.66	0.48	0.76

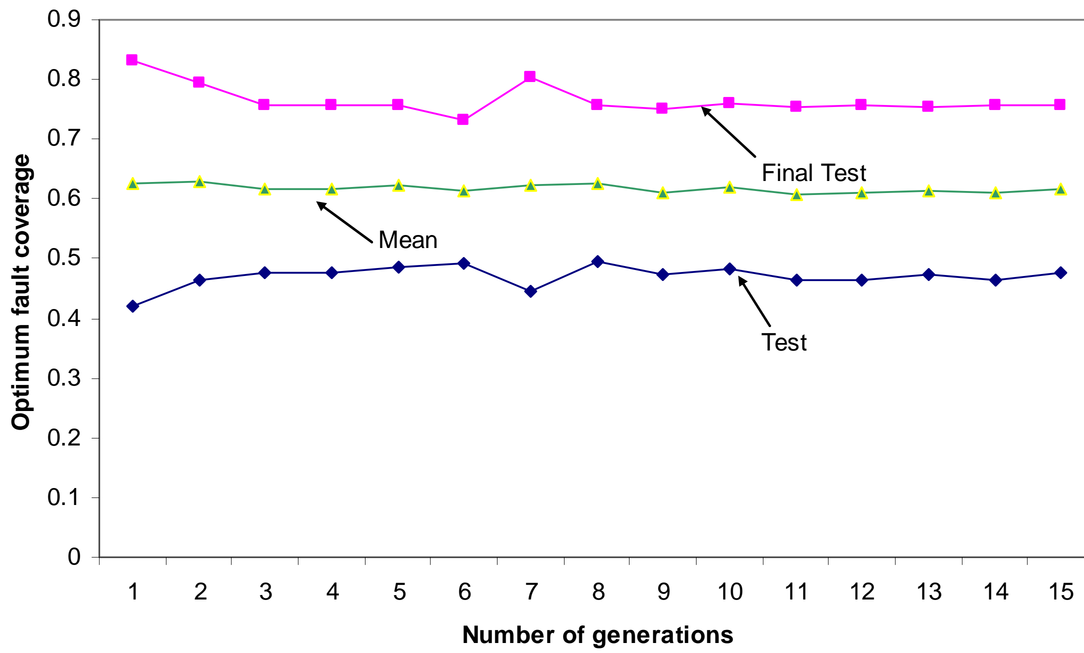


Figure 6.19: Optimum fault coverages of various TDRs in the process flow in Figure 6.17.

6.5 Summary

This chapter extends the optimization algorithm to process flows with uncertain inputs. Monte Carlo methods are integrated inside the RCGAs developed in Chapter 4 to sample the values from the input probability distribution(s). For each set of sampled values of input(s), the yielded cost was calculated using the searching algorithm developed in Chapter 3 with the fault coverage(s) generated by the GAs. The mean of the distribution of yielded costs serves as the fitness of the objective function. The GAs perform the same genetic operations as described in Chapter 4 to find the minimum mean of yielded costs. The GAs with Monte Carlo methods included are applied to single test and multi-TDR MCM assembly cases, which demonstrates that the optimization algorithms discussed in this dissertation are applicable to find the minimized yielded cost under uncertain input(s).

Chapter 7

Conclusions and Contributions

The goal of this dissertation is to develop a methodology for optimizing the location(s) and characteristics of test/diagnosis/rework (TDR) operations for complex process flows in order to minimize the yielded cost of electronic systems.

In this dissertation, a practical electronic products manufacturing and assembly process is analyzed and represented using a process flow model made up of a series of process steps. There are two kinds of information that should be included in the process flow model. One are step details, e.g., the step's impact on the product's cost and quality level. The other is the relationship among the process steps, i.e., which one is the predecessor and which one is the successor. The relationship among process steps determines the calculation sequence of yields and costs of products.

There are no manufacturing processes that assure the 100% yield of product, tests to find the faults that exist within defective products and rework to correct the defects can be placed in the manufacturing and assembly process of electronic systems. When a TDR operation is introduced into a process flow model, the calculation of feature parameters of products must take into account the cost and yield impact of the TDR operation. A

comprehensive TDR model that includes a detailed calculation of the feature variables for the single TDR operation has been developed by Trichy *et al.* This model accurately formulates the calculation of the yield and cost of product when it passes through a TDR operation with given characteristics, e.g., fault coverage and number of rework attempts. The Trichy *et al.* model does not, however, include a functional relationship between the test cost and fault coverage. The Trichy *et al.* TDR model was extended by relating cost to the fault coverage of test. Multiple fault types are also considered in the model extensions.

The cost of test and rework steps are associated with the corresponding characteristics – fault coverage and rework yield of the TDR. The quantitative relationships among these variables dictate that the tradeoff between the quality and cost can be balanced by optimizing the location(s) of test and characteristics of TDR operations to minimize the yielded cost of the process flow. A general process flow may, however, have many different TDR activities located within it. Some type of search algorithm is needed to address each process step in the process flow with all possible TDR operations included. For the computation of the final yielded cost of the process flow, a graph-based search algorithm, named "Waiting Sequential Search" or WSS was developed to search all the process steps and perform the accumulated computations of yields and cost of products to determine the value of a yielded cost objective function. The search process performs the sequential calculations from the start to the end of process flow and the single TDR model is used when it encounters the step nodes of TDR operations. The search algorithm differs from existing graph-based search algorithms in that specific features of the

process flow are considered and an adjacency matrix is used to represent the information associated with it.

After all the process steps have been visited, a multi-variable objective function to minimized the yielded cost of products can be derived from the search algorithm, in which the characteristics of all possible TDR operations are variables instead of constants. The objective function means that we can choose optimum locations of TDR operations and characteristics associated with them to minimize the final yielded cost of products. It is impossible to enumerate all the combinations of characteristics and locations of TDR operations, calculate the yielded cost and perform comparisons for complex process flows without any optimization techniques included. Optimization algorithms must be applied to effectively find the minimum value of the objective function.

Genetic algorithm (GA) – a natural optimization method is used as the optimization technique to solve the minimize problem posed in this dissertation. To represent the variables needs to be optimized with the floating-point numbers, the real-coded genetic algorithms (RCGAs) are developed to diminish the drawbacks of binary-coded genetics algorithms (BCGAs) and overcome the shortcomings in chromosome representation when the process flow becomes complex and a great number of TDR operations are involved within it. The first step in the application of RCGAs is to set the ranges of genes (the variable representation in GAs), which should be the same one as defined for each variable itself because of the real number encoding, e.g., the range of genes of fault coverage is from 0 to 1 and the range of genes of rework attempts could vary from 1 to

10. After the range setting, the fitness function should be defined to evaluate which chromosome that represents a combination of alternative solutions to variables in GAs leads to the minimized value of it. The definition of fitness function can be divided into two scenarios. The first scenario is that there are no uncertain input(s) to the process flow that needs to be optimized, in which case the fitness function is the value of the multi-variable objective function that has been derived by the proposed search algorithms. In the second case, when the inputs to the process flow are not constants but rather probability distributions, the mean of values of objective function for a given combination of variables serves as the fitness function. In this case a Monte Carlo method is applied inside the GAs optimization to sampling the values from the input distribution(s) and the distribution of objective function is produced by sampling the input values.

The GAs-based optimization method applied in this dissertation provides an intelligent solution to the placement of TDR operations in process flow. With RCGAs integrated, an optimization methodology has been developed in this dissertation that provides suggestions as to where the TDR operations should be placed or removed and the optimal values of feature parameters of TDR operations based on the minimization of a yielded cost objective function. The optimization results can guide the placement of TDR operations in practical manufacturing processes and be used as the feedback to DFT of electronic systems.

To validate the correctness and feasibility of the optimization model developed in this dissertation, a range of test cases from simple to complex branched process flow and a

real multichip module (MCM) assembly process are used. To verify that the algorithms can find the minimum values of the objective function successfully, a comparison to manually calculated results generated for simple cases by choosing the test strategies and characteristics without any optimization techniques involved was performed. The GAs with Monte Carlo methods included were applied to single test and multi-TDR MCM assembly cases, which demonstrates that the optimization algorithms discussed in this dissertation are applicable to find the minimized yielded cost under uncertain inputs(s).

7.1 Contributions

The main contributions of this dissertation are summarized below:

1. **First known attempt to determine optimum test location(s), fault coverages, and number of rework attempts in a complex process flow.** The methodology developed in this dissertation will help manufacturers choose the best assembly process flow for achieving the goal of minimized yielded cost by enabling the identification of optimum test locations and their characteristics. The optimization results can also be used as the feedback to Design for Test (DFT) of electronic systems.
2. **Demonstrated that the optimum yielded cost resulting from the methodology developed in this dissertation is an average of 6.85% lower than that obtained when using test strategies based on selecting specific fault coverages as in [74].** The optimization results always lead to equal or lower yielded cost than that found by manually choosing test strategies and characteristics without any optimization techniques involved. Furthermore, in a

complicated process flow, it would be impossible to enumerate all the combinations of fault coverages with various inputs. The optimization methodology developed in this dissertation will effectively find the optimized solution in less time than traditional approaches.

3. **Extended and enhanced TDR modeling to include: the functional relationships among the feature parameters, e.g., the costs of test and rework in terms of fault coverage and rework yield respectively and representation for multiple faults, and multiple fault types.** A quantitative relationship has been derived to relate the fault coverage to the testing cost and used to replace the constant representation used in Trichy *et al.* model. A method to represent multiple fault types is also addressed in this dissertation and can be integrated into the optimization algorithm for the single fault type.
4. **Developed a graph-based search algorithm for efficiently traversing a general process flow for the purpose of deriving a multi-variable objective function to minimize yielded cost of process flow.** Waiting Sequential Search (WSS) was proposed to traverse every process step in the graph representation of process flow and control the accumulative calculation of final cost and yield of products to derive the yielded cost objective function. WSS is efficient because every step is visited only once in the entire search and there is no recurring traverse repeated in WSS. The complexity of WSS depends on the number of process steps in the process flow.

7.2 Recommendations for Future Work

In order to build a practical cost estimation and optimization system for real electronic systems assembly, several extensions will be needed to this research. Below are the some of suggestions for future extensions:

1. **Multiple fault coverages (corresponding to different fault types) associated with a test.** It is not uncommon that there are multiple types of faults occurring to electronic systems assembly process. The optimization methodology developed in this dissertation should be enhanced to optimize multiple fault coverages associated with one test operation.
2. **Expert suggestions and heuristics.** There are no expert suggestions or heuristics applied in this dissertation. TDR operations are placed in all possible locations in the process flow before the optimization. Actually, in the practical manufacturing of electronic systems, some locations cannot be set test or rework, i.e., the number of variables to represent the characteristics of TDR that need to be optimized can be reduced depending on the specific manufacturing and assembly process of products. The time and cost to finish the optimization of a complex process flow can be reduced by introducing the experts' suggestions.
3. **Robust optimization.** The robust optimization incorporates uncertainty in the problem design itself, and thereby generates solutions, which are less sensitive to realizations of the model data [99-102]. The robust optimization results in the design, which performs optimally under the variable (or uncertain) operating conditions over the entire lifetime of the design. The optimization algorithm will

produce a less sensitive optimum solution to various inputs when the robust techniques are included into it.

4. **Application-specific modeling.** For the real electronic products manufacturing, the test process and rework operation are different from each other. The TDR model discussed in this dissertation doesn't consider the details for each kind of products and provides no ideas about how similar manufacturing and assembly process can be grouped. For example, the rework operation of the MCM assembly is different from repairing a printed circuit board, in which the bad parts of the MCM will be replaced by good ones therefore the rework yield and rework cost depend on how many die have been replaced and the single TDR model should be modified to it. The future extension to this research can categorize the test and rework operation based on the specific manufacturing and assembly process and provides application-specific cost model.
5. **Finding the "best" optimization method.** There are several categories of optimization methods that are applicable to the minimization problem discussed in this dissertation. Different optimization methods applied to minimizing the yielded cost of process flows should be compared according to their efficiency and robustness, some of which can be combined together to get rid of shortcomings of a single optimization method.

It is expected that the recommended future extension will enhance the usefulness of this research, and will result in development of a fully robust and practical cost estimation and optimization system for electronic systems assembly.

References

[1] J. Turino, *Design to Test – A Definitive Guide for Electronic Design, Manufacture, and Service*, Van Nostrand Rienhold, New York, NY, 1990.

[2] W. Rhines, *Keynote address at the Semico Summit*, Phoenix, AZ, March 2002.

[3] "Test and Test Equipment," *The International Technology Roadmap for Semiconductors*, Semiconductor Industries Association, 2001.

[4] M. Abramovici, M. A. Breuer, and A. D. Friedman, *Digital Systems Testing and Testable Design*, Computer Science Press, 1990.

[5] J. Bateson, *In-Circuit Testing*, Van Nostrand Rienhold, New York, NY, 1985.

[6] D. A. Beaujean and S. B. Morgan, "An Educational Perspective on In-Circuit Testing", *IEE Proc. of Science, Measurement and Technology*, Vol. 135, Pt. A, No. 4, April 1988, pp. 233-240.

[7] F. P. M. Beenker, R. G. Bennetts and A. P. Thijssen, *Testability Concepts for Digital ICs*, Kluwer Academic Publishers, 1995.

[8] T. Trichy, *Full Monte Carlo Technical cost Modeling with Detailed Test/Rework/Diagnosis Analysis for Inclusion in a Multi-attribute Optimization Environment*, MS. Thesis, Department of Mechanical Engineering, University of Maryland, College Park, Maryland, August, 2000.

[9] T. Trichy, P. Sandborn, R. Raghavan, and S. Sahasrabudhe, "A new test/diagnosis/rework model for use in technical cost modeling of electronic

systems assembly," *International Test Conference*, October 2001, pp. 1108-1117.

[10] J. Cudmore, "Rework Management and Optimization," *SMT Magazine*, October 1998, pp. 11-13.

[11] J. Petek and H. K. Charles, "Known Good Die, Die Replacement (Rework), and Their Influences on Multichip Module Costs", *IEEE Electronic Components and Technology Conference,* May 1998, pp. 909 –915.

[12] P. Sandborn, *Life Cycle Cost Analysis*, CALCE, University of Maryland, 2001.

[13] T. W. Williams, and N. C. Brown, "Defect Level as a Function of Fault Coverage", *IEEE Transactions on Computers*, December 1981, pp. 987-988.

[14] B. Davis, *The Economics of Automatic Testing*, McGraw-Hill Book Company, 1994.

[15] D. Becker and P. Sandborn, "One the Use of Yielded cost in Modeling Electronic Assembly Processes," *IEEE Trans. on Electronics Packaging Manufacturing*, Vol. 24, July 2001, pp. 195-202.

[16] R. Raghavan, *Analysis of Test/Diagnosis/Rework Operation Placement in the Technical Cost Modeling of Advanced Electrical Power System Modules*, MS. Thesis, Univ. of Maryland, College Park, Maryland, Aug., 2001.

[17] P. K. Nag, A.Gattiker, S. Wei, R.D. Blanton and W. Maly, "Modeling the economics of testing: a DFT perspective", *IEEE Design & Test of Computers,* Vol. 19, Jan/Feb 2002, pp 29-41 .

[18] J. Turino, "Test economics in the 2lst Century", *IEEE Design & Test of Computers,* Vol. 14, Jul-Sep 1997, pp. 41-44.

[19] M. L. Bushnell and V. D. Agrawal, *Essentials of Electronic Testing for Digital, Memory and Mixed-Signal VLSI Circuits*, Boston: Kluwer Academic Publishers, 2000.

[20] A. P. Ambler, M. Abadir, and S. Sastry, *Economics of Design and Test for Electronic Circuits and Systems*, Ellis Horwood Lit., Chichester, UK, 1992.

[21] J. Miranda, "A BIST & Boundary Scan Economics Framework", *IEEE Design and Test of Computers*, Vol. 14, No. 3, 1997, pp. 17-23.

[22] D. Wheater, P. Nigh, J. T. Mechler, and L. Lacroix, "ASIC Test Cost/Strategy Trade-offs", *Proc. of International Test Conference*, 1994, pp. 93-102.

[23] C. Dislis, I. D. Dear, J. R. Miles, S. C. Lau and A. P. Ambler, "Cost Analysis of Test Method Environments", *Proc. of International Test Conference*, 1989, pp. 875-883.

[24] P. Varma, "An Analysis of the Economics of Scan-Path Design for Testability", *Journal of Electronic Testing: Theory and Application*, Vol. 5, May/Aug. 1994, pp. 179-193.

[25] W. Maly, "Cost of Silicon Viewed from VLSI Design Perspective," *Proc. of the 31st Design Automation Conference*, June 1994, pp. 135-142.

[26] W. Maly, "Design for Manufacturability in Submicron Domain," *Proc. of ICCAC, Nov. 1996*, pp. 690-697.

[27] W. M. Needham, *Designer's Guide to Testable ASIC Devices*, Van Nostrand Reeinhold, New York, 1991.

[28] S. Wei, P. K. Nag, R. D. Blanton, A. Gattiker and W. Maly, "To DFT or Not To DFT?" *IEEE International Test Conference*, 1997, pp. 557-566.

[29] R. H. Williams, R. G. Wagner, and C. F. Hawkins, “Testing Errors: Data and Calculations in an IC Manufacturing Process,” *Proceedings of the International Test Conference*, 1992, pp. 352-361.

[30] C. L. Henderson, R. H. Williams, and C. F. Hawkins, “Economic Impact of Type I Test Errors at System and Board Levels,” *Proceedings of the International Test Conference*, 1992, pp. 444-452.

[31] J. Busch, “Cost Modeling as a Technical Management Tool,” *Research Technology Management*, November 1994.

[32] C. Dislis, J. H. Dick, I.D. Dear, I. N. Azu, and A. P. Ambler, "Economics Modeling for the Determination of Test Strategies for Complex VLSI Boards", *Proceedings of the International Test Conference*, 1993, pp. 210-217.

[33] M. Abadir, A. Parikh, L. Bal, P. Sandborn, and C. Murphy, "High Level Test Economics Advisor", *Journal of Electronic Testing: Theory and Applications*, Vol. 5, May 1994, pp. 195-206.

[34] P. A. Sandborn, and H. Moreno, *Conceptual Design of Multi-chip Modules and Systems*, Kluwer Academic Publishers, pp 152-169, Boston, 1994.

[35] M. V. Tegethoff, *Manufacturing Test SIMulator, A Concurrent Engineering Tool for Boards and Multi-Chip Modules*, Ph.D. Dissertation, Colorado State University, 1994.

[36] M. Scheffler, D. Ammann, A. Thiel, C. Habiger, and G. Troster, “Modeling and Optimizing the Costs of Electronic Systems,” *IEEE Design & Test of Computers*, Vol. 15, no. 3, July-September 1998, pp. 20-26.

[37] C. Dislis, J. H. Dick, I. D. Dear, and A. P. Ambler, *Test Economics and Design for Testability, Ellis Horwood*, 1995.

[38] V. Garg, D. J. Stogner, C. Ulmer, D. Schimmel, C. Dislis, S. Yalamanchili, and D. S. Wills, "Early Analysis of Cost/Performance Trade-Offs in MCM Systems", *IEEE Transactions on Component, Packaging and Manufacturing Technology*, Part B, Vol. 20, No. 3, Aug. 1997, pp. 308-319.

[39] P. Goel, "Test Generation costs analysis and projections," *Proc. of Design Automation Conference,* 1980*,* pp. 77-84.

[40] M. Driels and J. S. Klegka, "Analysis of Alternative Rework Strategies for Printed Writing Assembly Manufacturing Systems," *IEEE Transactions on Components, Hybrids, and Manufacturing Systems*, Vol. 14, No. 3, September 1991, pp. 637-644.

[41] V. Chachra, P. M. Ghare and J. M. Moore, *Applications of Graph Theory Algrorithms*, Elsevier North Holland, Inc., 1979.

[42] G. Chartrand and O. R. Oellermann, *Applied and Algorithmic Graph Theory*, McGraw-Hill, Inc., 1993.

[43] J. A. Bondy and U. S. R. Murty, *Graph Theory with Applications*, North-Holland, New York, 1976.

[44] N. Hartsfield and G. Ringel, *Pearls in Graph Theory*, Academic Press, San Diego, 1990.

[45] O. Ore, *Gaphs and Their Uses*, (Revised by R. J. Wilson) Math. Assoc. of Amer., Washington, D.C., 1990.

[46] R. J. Wilson and J. J. Watkins, *Graphs: An Introductory Approach*, Wiley New York, 1990.

[47] N. S. V. Rao, "On parallel algorithms for single-fault diagnosis in fault propagation graph systems," *IEEE Transactions on Parallel and Distributed Systems,* Vol. 7, December 1996, pp. 1217 –1223

[48] D. M. Blough and A. Pelc, "Complexity of Fault Diagnosis in Comparison Models," *IEEE Trans, Computers*, Vol. 41, No. 3, 1992, pp. 318-324.

[49] N. S. V. Rao and N. Viswanadham, "Computational Complexity Issues in Operative Diagnosis of Graph-based Systems," *IEEE Trans. Computers*, Vol. 42, No. 47, 1993, pp.447-45.

[50] N. S. V. Rao and N. Viswanadham, "Fault Diagnosis in Dynamical Systems: A Graph Theroretic Approach," Int' J. Systems Science, Vol. 18, No. 4, 1987, pp.687-695.

[51] R. E. Tarjan, "Depth-First Search and Linear Graph Algorithms," *SIAM J. Computing*, Vol. 1, No.2, pp.146-160, 1972.

[52] V. Raghavan, K. R. Pattipati, and M. Shakeri, "Optimal and Near-Optimal Test Sequencing Algorithms with Realistic Test Models," *IEEE Trans. Syst., Man, Cybern.,* Vol. 29, Jan. 1999, pp. 13-25.

[53] V. Raghavan, M. Shakeri, and K.R. Pattipati, "Test Sequencing Algorithms with Unreliable Tests", *IEEE Trans. Syst., Man, Cybern.,* Vol. 29, no. 4, 1999, pp347-357.

[54] K. W. Choi and A Chatterjee, "Efficient instruction-level optimization methodology for low-power embedded Systems", *Proc. of 14th Int'l Symposium. on System. Synthesis*, 2001, pp.147 –152.

[55] J. D. Ullman and J. Widom, *A First Course in Database Systems*, Prentice Hall, Inc., 1998.

[56] "Test and Test Equipment," *The International Technology Roadmap for Semiconductors*, Semiconductor Industries Association, 2002.

[57] R. Haupt, "Comparison between Genetic and Gradient-based Optimization Algorithms for Solving Electromagnetics Problems," *IEEE Transactions on Magnetics,* Vol. 31, May 1995, pp. 1932-1935.

[58] P. Wolfe, "Methods of Nonlinear Programming," *Recent Advances in Mathematical Programming,*, McGaw-Hill New York,1963, pp. 67-86.

[59] S., Kirkpatrick, C. C. Gelatt, and M. P. Vecchi, "Optimization by Simulated Annealing," *Science,* Vol. 220, 1983, pp. 671-680.

[60] J. H. Holland, *Adaptation in Natural and Artificial Systems*, University of Michigan Press, Ann Arbor, Michigan, 1975.

[61] D. P. Solomatine, "Two strategies of adaptive cluster covering with descent and their comparison to other algorithms," *Journal of Global Optimization*, Vol. 14, No. 1, 1999, pp. 55-78.

[62] Z. Michalewicz, *Genetic Algorithm + Data Structure = Evolution Program,* Springer-Verlag, Inc., Heidelberg, Berlin, 1996.

[63] J. H. Holland, "Genetic Algorithms," *Scientific American,* July 1992, pp. 66-72.

[64] D. E. Goldberg, *Genetic Algorithms in Search, Optimization and Machine Learning*, Addison-Wesley Publishing Company, Inc., Reading, MA, 1989.

[65] R. L. Haupt and S. E. Haupt, *Practical Genetic Algorithms*, Wiley, Inc., 1998.

[66] C. Z. Janikow and Z. Michalewicz, "An Experimental Comparison of Binary and Floating Point Representations in Genetic Algorithms," *Proceedings of the Fourth International Conference on Genetic Algorithms*, Morgan Kaufmann Publishers, Inc., San Mateo, CA, 1991, pp.31-36.

[67] A. H. Wright, "Genetic Algorithms for Real Parameter Optimization," *Foundations of Genetic Algorithms*, Morgan Kaufmann Publishers, San Mateo, CA, 1991, pp.205-218.

[68] L. J. Eshelman and J. D. Schaffer, "Real-coded genetic algorithms and interval schemata," *Foundations of Genetic Algorithms.2*," Morgan Kaufmann Publishers, Inc., San Mateo, CA, 1993, pp.187-202.

[69] J. D. Schaffer, "Multiple Objective Optimization with Vector Evaluated Genetic Algorithms," *Proceedings of the First International Conference on Genetic Algorithms and Their Applications*, Lawrence Erlbaum Associates, Publishers, Hillsdate, NJ, 1985, pp.93-100.

[70] A. Oyama, S., Obayashi, and K. Nakahashi, "Wing Design Using Real-Coded Adaptive Range Genetic Algorithm," *Proceedings of 1999 IEEE International Conference on Systems, Man, and Cybernetics*, Vol. 4, October 1999, pp. 475 -480.

[71] D. E. Goldberg, and J. Richardson, "Genetic Algorithms with Sharing for Multimodal Function Optimization," *Proceedings of the Second International*

Conference on Genetic Algorithms, Morgan Kaufmann Publishers, Inc., San Mateo, CA, 1987, pp.41-49.

[72] J. E. Baker, "Reducing Bias and Inefficiency in the Selection Algorithm," *Proceedings of the Second International Conference on Genetic Algorithms*, Morgan Kaufmann Publishers, Inc., San Mateo, CA, 1987, pp.14-21.

[73] K. A. De Jong, *An Analysis of the Behavior of a Class of Genetic Adaptive Systems*, Doctoral Dissertation, University of Michigan, Ann Arbor, 1975.

[74] M. Abadir, "Economics of Multichip Modules Testing Strategies," *IEEE Transactions on Components, Packaging, and Manufacturing Technology,* Vol. 21, No. 4, November 1998, pp. 360-370.

[75] M. Abadir, A. Parikh, L. Bal, P. Sandborn, R. Ghosh, and C. Murphy, "High Level Testability Economics Advisor (Hi-TEA)," *Journal of Electronic Testing (JETTA),* Vol. 5, No. 2/3, 1994, pp. 195-206.

[76] J. Licari, *Multichip Module Design, Fabrication, and Testing*, McGraw-Hill Company, 1995.

[77] N. Sherwani, Q, Yu, and S. Badida, *Introduction to Multichip Modules*, John Wiley and Sons, Inc., 1995.

[78] C. J. Bartlett, "Advanced Packaging for VLSI," Solid-State Technology, June 1986, pp. 119-123.

[79] L. A. Bixby and D. T. Dimatteo, "Multichip Modules: An Alternative Packaging Technology," Hybrid Circuit Technology December 1990 pp. 9-13.

[80] W. Blood and A. Dixon, "MCM-L: Cost Effective Multichip Module Substrates," *Inside International Society for Hybrid Microelectronics*, March/April 1993 pp. 7-12.

[81] D. A. Doane and P. D. Franzon, "Multichip Module Technologies and Alternatives: the Basics," Van Nostrand Reinhold, 1993.

[82] T. Mazzullo, "Introduction to MCM Design," Proceedings of the Third Annual PCB Design Conference, 1994, pp. 85-92.

[83] J. Reche, "High Density Multichip Module Fabrication," International Journal of Hybrid Microelectronics, December 1990 pp. 490-498.

[84] P. Sandborn, H. Hashemi, L. Bal, and M. Abadir, "Technology Application Tradeoff Studies in Multichip Systems," IEEE Transaction on Computer and Packaging, and Manufacturing Technology, Vol. 17 May 1994, pp. 161-169.

[85] C. Dislis, A. P. Ambler, and J. Dick, "Economic Effects in Design and Test," *IEEE Design and Test Computer*, Vol. 6, December 1991, pp. 64-67.

[86] C. Dislis, J. Dick, and A. P. Ambler, "An Economics based Test Strategy Planner for VLSI," Proceedings of 2nd European Test Conference, 1991.

[87] G. W. Zobrist, VLSI Fault Modeling and Testing Technologies, Ablex Publishing Corp. 1992.

[88] D.R. Conti, and J Van Horn, "Wafer Level Burn-in," Proceedings of the 50th Electronic Components and Technology Conference, May 2000, pp. 815 -821.

[89] M.H. Kalos, and P.A. Whitlock, *Monte Carlo methods, Vol. 1: Basics*, John Wiley &Sons, New York, 1986.

[90] K. Binder, *Monte Carlo simulation in statistical physics: an introduction*, New York, Springer, 2002.

[91] A. W. Drake, *Fundamentals of Applied probability theory*, McGraw-Hill Company, 1967.

[92] S. Ross, *A First Course in Probability, 5th edition*, Prentice-Hall International Inc., 1998.

[93] D. Greenhalgh and S, Marshall, “Convergence Criteria for Genetic Algorithms,” Society for Industrial and Applied Mathematics Journal, Vol. 30, No. 1, May 2000 pp. 269-282.

[94] H. Aytug and G. J. Koehler, “Stopping criteria for Finite Length Genetic Algorithms,” ORSA Journal Computer, Vol. 8, 1996 pp. 183–191.

[95] L. Davis, Handbook of Genetic Algorithms, Van Nostrand Reinhold, New York, 1991.

[96] Y. Deng, X. Ren, C. Zhao, and D. Zhao, “A Heuristic and Algorithmic Combined Approach for Reactive Power Optimization with Time-varying Load Demand in Distribution Systems, IEEE Transactions on Power Systems, Vol. 17, November 2002, pp. 1068 – 1072.

[97] M. Fan, Z. Zhang, and C.E Lin, “Optimization of Tap Settings and Capacitor Units with Expert System on Distribution System,” IEEE Proceedings of Computer, Communication, Control and Power Engineering, Vol. 5, October 1993, pp. 292 – 295.

[98] L. Shi, and G. Xu, “A General Global or Near Global Optimization Method – Self-adaptive Heuristic Evolutionary Programming,” Proceedings of the 3rd

World Congress on Intelligent Control and Automation, Vol. 5, June 2000, pp. 3481 – 3485.

[99] H. Yue and W. Jiang, "A New Probabilistic Robust Optimization Method," IEEE International Conference on Systems, Man, and Cybernetics, Vol. 6 October 1996, pp. 2205 - 2209.

[100] L. Ghaoui and F. Seigneuret, "Robust Optimization Methodologies for the Free Route Concept," Proceedings of the American Control Conference, Vol.3, June 1998, pp. 1797 – 1799.

[101] T. B. Trafalis, and S.A. Alwazzi, "Robust Optimization in Support Vector Machine Training with Bounded Errors, Proceedings of the International Joint Conference on Neural Networks Vol.3, July 2003, pp. 2039 – 2042.

[102] A. Ben-Tal, and A. Nemirovski, "Robust Solutions to Uncertain Linear Programs via Convex Programming," Operations Research Letters, Vol. 25, 1996, pp. 1-17.