Study and Analysis of Minimization Algorithms for VLSI Circuit Synthesis

Thesis

Submitted for the award of

DOCTOR OF PHILOSOPHY

by

MANU BANSAL

Regn No. 951006005

Under the supervision of

Dr. Alpana Agarwal

Associate Professor

Electronics & Communication Engineering Department,
TIET, Patiala- 147004

October 2018

CERTIFICATE

I hereby certify that the work which is being presented in the thesis entitled, **"Study and Analysis of Minimization Algorithms for VLSI Circuit Synthesis,"** for the award of degree of **Doctor of Philosophy** in Electronics and Communication Engineering Department, Thapar Institute of Engineering and Technology, Patiala, is an authentic record of my own work carried out under the supervision and guidance of Dr. Alpana Agarwal, Associate Professor, ECED, Thapar Institute of Engineering and Technology, Patiala.

The results presented in this thesis have not been submitted in part or in full to any other University or Institute for the award of any degree or diploma.

Manu Bansal

(Manu Bansal)

951006005

This is to certify that the above statement made by the candidate is correct and true to the best of my knowledge and belief.

Alpana

(Dr. Alpana Agarwal)

Associate Professor

ACKNOWLEDGMENT

Before I express my heartfelt gratitude towards all my mentors, first and foremost, I would like to thank **God**. He has given me the power to believe in myself.

I would like to express my sincere gratitude to my advisor Dr. **Alpana Agarwal** for the continuous support of my PhD study and related research, for her patience, motivation, and immense knowledge. Her guidance helped me in all the time of research and writing of this thesis. I could not have imagined having a better advisor and mentor for my Ph.D. study.

Besides my advisor, I would like to thank the rest of my thesis committee: **Dr. Kulbir Singh, Dr. Gagandeep Kaur, and Dr. Vinay Kumar,** for their insightful comments and encouragement, but also for the hard question which helped me to widen my research from various perspectives.

I would like to thank **Dr. Rafat Siddique** (Current Dean-Research Sponsored and Planning) **and Dr. O. P. Pandey** (Ex Dean-Research Sponsored and Planning) for guiding me and continuously supporting me like a pillar.

I would like to thank Dr. **Rinkle Rani** (Associate Professor, CSED) for supporting me in my thesis writing. I would like to acknowledge SMDP-VLSI (Phase- II) and SMDP C2SD Project sponsored by DietY (Govt. of India).

At last, I want to thank my husband **Amit** and my sons **Pranav and Vaibhav** who always missed their mother in the hour of need.

Last but not the least; I would like to thank my family for supporting me spiritually throughout writing this thesis and my life in general.

Manu Bansal

(Manu Bansal)

ABSTRACT

Low power consumption has emerged as a key design parameter for digital VLSI systems. Therefore, accurate methods are required to estimate the switching activity at the internal nodes of the logic circuits to determine the average power dissipation. Since, manipulation of Boolean functions is an important element of most logic synthesis algorithms, including logic optimization and logic verification of sequential and combinational circuits, therefore, it is important to have efficient methods to represent and manipulate such functions. A major problem with binary decision diagrams (BDD) based manipulation is the need for application-specific heuristic algorithms to order the input variables before processing. Therefore, finding a good variable order for ordered binary decision diagrams (OBDDs) is an essential part of OBDD-based CAD tools.

The three techniques *i.e.* Genetic Algorithm, Hybridized Genetic Algorithm and Modified Memetic Algorithm (MMA) have been proposed for the variable reordering problem and determining and minimizing signal activity in BDDs. Ordering of variables in BDDs play a major role in reduction of nodes and hence the area. The performance of the genetic algorithms depends, to a great extent, on the performance of the crossover operator used. Three new versions of the crossover operator which overcome these problems are order crossover, cycle crossover and partially mapped crossover (PMX). All the three proposed algorithms based on the crossover techniques are implemented and validated on multi-input multi-output (MIMO) LGSynth93 Benchmark Circuits in order to find an optimal input variable order to reduce node count hence area, simultaneously reducing the signal activity hence reducing power dissipation using BDD-based probabilistic technique.

The proposed Modified Memetic algorithm shows better results as compared to Genetic and Hybridized Genetic algorithms for all MIMO circuits. In terms of node count, area is reduced by 10% to 68% and best results are obtained with partially mapped crossover, however it takes some extra computation time as compared to order crossover and cycle crossover. To reduce the switching activity and hence power, the Probabilistic Approach has been found to be better than MUX based approach and reduces power up to 81%.

CONTENTS

CERTIFICATE **i**
ACKNOWLEDGMENT **ii**
ABSTRACT **iii**
CONTENTS **iv**
LIST OF FIGURES **vi**
LIST OF TABLES **viii**
ACRONYMS **ix**

Chapter 1 Introduction

1.1 Background 1
1.2 Binary Decision Tree (BDT) 5
1.3 Ordered Binary Decision Diagram (OBDD) 6
1.4 Reduced Ordered Binary Decision Diagram(ROBDD) 7
1.5 Applications of BDDs 8
1.5.1 Functional Synthesis: BDD as MUX Circuits 8
1.5.2 Functional Verification of Logic Circuits 9

1.6 Motivation 10
1.7 Organization of Thesis 11

Chapter 2 Literature Survey 13
2.1 State-of-the-Art (SOA) Analysis 22

Chapter 3 Problem Formulation 26

3.1 Research Gaps 26
3.2 Objectives and Methodology 26
3.3 Methodology 26

Chapter 4 Optimization Algorithms 29

4.1 Variable Reordering Problem in BDDs 29
4.2 Static Variable Ordering Techniques 30
4.3 Dynamic Variable Ordering Techniques 31
4.4 Algorithms used for Variable Ordering in BDDs 33
4.5 Comparison among various Variable Ordering Techniques 36
4.6 Evolutionary Algorithms 38
4.6.1 Genetic Algorithm 38
4.6.2 The Simple GA 42
4.6.3 The Steady- State Algorithm 44
4.6.4 Genetic Operators 46
4.6.4.1 Selection 46
4.6.4.2 Crossover 48
4.6.4.3 Mutation 51

4.6.4.4 Fitness Scaling 53
4.6.4.5 Inversion 53
4.7 Simple Genetic Algorithm (SGA) 54
4.8 Hybrid Genetic Algorithm (HGA) 55
4.9 Particle Swarm Optimization Algorithm (PSO) 56
4.10 Memetic Algorithm (MA) 58

Chapter 5 Algorithms Applied For BDD Optimization 61

5.1 Genetic Algorithm (GA) 61
5.2 Hybridized Genetic Algorithm (HGA) 61
5.3 Solution or Meme Representation 63
5.4 Fitness Function Calculation 64
5.5 Local Search Operation for Meme Improvement 64
5.6 Genetic Operators 65
5.6.1 Crossover Technique for Generating New Population 65
5.6.2 Mutation Operation 66
5.7 Parameter Settings and Flow Chart 67
5.8 Overview of BDD Package 68
5.8.1 BDD Algorithm 68
5.8.2 Dynamic Variable Ordering 68
5.8.3 Garbage Collection 70

Chapter 6 Estimation of Power Using Probabilistic Technique 71

6.1 Reduction of Power at Different Abstraction Levels 71
6.2 Types of Power Dissipation 73
6.3 Why Estimate Power Dissipation? 74
6.4 Probabilistic Technique for Switching Power Optimization 75
6.4.1 Switching Activity Estimation 75
6.4.2 Low Power Estimation 76
6.4.3 BDD Mapped Circuits 76

Chapter 7 Results and Discussions 78
7.1 Comparison of Node Count among various Evolutionary Algorithms 78
7.2 Graphical Analysis of Results 80
7.3 Estimation of Node Count and Computation Time by Implementing Genetic Algorithm with Crossover Techniques 81
7.4 Node Count and Computation Time estimation for Hybridized Genetic Algorithm 85
7.5 Power Estimation 88
7.6 Estimation of Node Count, Computation Time and Power for Modified Memetic Algorithm 92

Chapter 8 Conclusion and Future Scope 98
8.1 Conclusion 98
8.2 Future Scope of Work 99

REFERENCES
LIST OF PUBLICATIONS

LIST OF FIGURES

Figure 1.1	Different OBDDs for same Boolean function F with different variable order F = ab + a'c + bc'd	6
Figure 1.2	Ordered BDD and corresponding ROBDD for the Boolean Function, f = ab+c	8
Figure 1.3	A 2x1 Multiplexer and corresponding transistor realization	9
Figure 1.4	ROBDD and corresponding MUX representation for boolean function, f = x1x2'+ x1' (x2'x3'+ x2)	9
Figure 1.5	Equivalence Verification of boolean functions f1 = ab+a'c+bc and f2 = ab+a'c	10
Figure 4.1	The simple genetic algorithm	42
Figure 4.2	Flowchart of simple genetic algorithm	44
Figure 4.3	Flowchart of steady-state genetic algorithm	45
Figure 4.4	Proportionate Selection Schemes	47
Figure 4.5	Modified Alternating Crossover Mechanism	48
Figure 4.6	Order Crossover Mechanism	50
Figure 4.7	Cycle Crossover Mechanism	50
Figure 4.8	PMX Crossover Mechanism	51
Figure 4.9	Mutation operator	52
Figure 4.10	Inversion operator	54
Figure 4.11	Operation Flow of Genetic Algorithm	55
Figure 4.12	Flow Chart for Hybrid Genetic Algorithm	56
Figure 4.13	Flow Chart for Particle Swarm Optimization algorithm	58
Figure 4.14	Local Search LS using pair-wise swapping chromosome after local search	59
Figure 4.15	Flow Chart for Basic Memetic Algorithm	60
Figure 5.1	Flow-chart of the Genetic Algorithm based technique used for BDD Variable Ordering	62
Figure 5.2	Flow-chart of the Hybridized Genetic Algorithm based technique used for BDD Variable Ordering	63
Figure 5.3	Solution or Meme Encoding in Chromosome Form	64
Figure 5.4	Local Search LS mechanism for improving the performance of best s solutions	65
Figure 5.5	Search LS mechanism for improving the performance of worst *s* solutions	65
Figure 5.6	Modified Alternating Crossover Mechanism	66
Figure 5.7	Mutation Operation incorporated in MMA	67
Figure 5.8	Flow Chart for proposed Modified Memetic Algorithm	67
Figure 5.9	Shifting process for Dynamic Variable Reordering	70
Figure 6.1	Different abstraction levels to reduce power dissipation	73
Figure 6.2	Pi chart showing types of power dissipations	74
Figure 7.1	Graph showing runtime differences in HGA and MMA	80
Figure 7.2	Graph showing reduction in BDD size in terms of node count using different algorithms	81
Figure 7.3	Graphical Analysis showing Node Count comparison by applying Proposed Genetic algorithms LGSynth93 Benchmark Circuits	83

Figure 7.4	Graphical Analysis representing Computation Time for Proposed Genetic algorithm on LGSynth93 Benchmark Circuits	85
Figure 7.5	Graphical Analysis showing Node Count comparison for Proposed Hybridized Genetic algorithms LGSynth93 Benchmark Circuits	86
Figure 7.6	Graphical Analysis representing Computation Time for Proposed Hybridized Genetic algorithm on LGSynth93 Benchmark Circuits	88
Figure 7.7	Comparison of the power estimated by Probabilistic Technique and Existing Technique in Proposed Genetic Algorithm for LGSynth93 Benchmark Circuits	91
Figure 7.8	Power Estimation using Probabilistic Technique and Existing Technique in Hybridized Genetic Algorithm on LGSynth93 Benchmark Circuits	92
Figure 7.9	Graphical Analysis showing Node Count comparison using MMA for LGSynth93 Benchmark Circuits	94
Figure 7.10	Graphical Analysis representing Computation Time for Proposed MMA with Crossover algorithm on LGSynth93 Benchmark Circuits	96
Figure 7.11	Graphical Analysis representing Power Estimation Proposed MMA with Crossover algorithm on LGSynth93 Benchmark Circuits	97

LIST OF TABLES

Table 1.2	Truth Table for Boolean Function f= ab+c	7
Table 2.1	Comparison of Various Performance Parameters for different Optimization Algorithms / Techniques / Approaches.	21
Table 4.1	Summary of Variable Ordering Algorithms	36
Table 7.1	Node Count Comparison of MMA results with other Evolutionary Algorithms	78
Table 7.2	Node Count Comparison of MMA with Dynamic Variable Ordering Algorithms	79
Table 7.3	Comparison of Node Count for Dynamic and Proposed Genetic algorithms LGSynth93 Benchmark Circuits	82
Table 7.4	Comparison of Computation Time for proposed Genetic algorithm for LGSynth93 Benchmark Circuits	84
Table 7.5	Comparison of Node Count for Dynamic and Hybridized Genetic algorithms for LGSynth93 Benchmark Circuits	85
Table 7.6	Comparison of Computation Time for Different Crossover Operators in Hybridized Genetic algorithm for LGSynth93 Benchmark Circuits	87
Table 7.7	Report for Power calculation using Synopsys tool	88
Table 7.8	Comparison of the Power estimated using Probabilistic technique and MUX based technique (values calculated using Synopsys tool) in Proposed Genetic Algorithm for LGSynth93 Benchmark Circuits.	90
Table 7.9	Comparison of the Power calculated using Probabilistic technique and the existing technique in Hybridized Genetic Algorithm for LGSynth93 Benchmark Circuits	91
Table 7.10	Comparison of Node Count using MMA for LGSynth93 benchmark circuits with Dynamic Algorithms	93
Table 7.11	Comparison of Computation Time for Different Crossover Operators in Modified Memetic algorithm for LGSynth93 Benchmark Circuits	95
Table 7.12	Comparison of MMA results for LGSynth93 benchmark circuits with existing algorithm	96

ACRNOYMS

DSP	DIGITAL SIGNAL PROCESSING
VLSI	VERY LARGE SCALE INTEGRATED CIRCUIT
BDD	BINARY DESCISION DIAGRAM
BDT	BINARY DESCISION TREE
MUX	MULTIPLEXER
OBDD	ORDERED BINARY DESCISION DIAGRAM
ROBDD	REDUCED ORDER BINARY DESCISION DIAGRAM
PTL	PASS TRANSISTOR LOGIC
GAs	GENETIC ALGORITHMS
TLU	TABLE LOOK UP
SGA	SIMPLE GENETIC ALGORITHM
HGA	HYBRIDIZED GENETIC ALGORITHM
PSO	PARTICLE SWARM OPTIMIZATION
MA	MEMETIC ALGORITHM
MMA	MODIFIED MEMETIC ALGORITHM
QAP	QUADRATIC ASSIGNMENT PROBLEM
FPGA	FIELD PROGRAMMABLE GATE ARRAY
HAS	HARMONY SEARCH ALGORITHM
TLs	TRANSMISSION LINES
ACO	ANT COLONY OPTIMIZATION
QoS	QUALITY of SERVICE
QDV	QUALITY BASED DISTANCE VECTOR
BB	BRANCH AND BOUND
HSPPSO	HIERARCHICAL STRUCTURE POLY-PARTICLE SWARM OPTIMIZATION
DE	DIFFERENTIAL EVOLUTION
SR	STOCHASTIC RANKING
LS	LOCAL SEARCH

CHAPTER 1
INTRODUCTION

1.1 Background

With the expansion of lightweight and available portable devices, energy efficiency is going to play a key role in digital design. Battery only adds weight and volume to the portable devices existing in the market. To minimize power consumption by the circuit, mostly low voltage power supply is used. This in turn results in the performance degradation of the circuit in terms of the reduced voltage swings at the output. Supply voltage scaling still requires the scaling of the threshold voltage that result in increased static power dissipation. With the advent of scaling, digital integrated circuits have been benefitted tremendously but analog integrated circuits could hardly find any advantage of scaling. Different performance advancements in analog circuits have finally resulted to intricate and expensive circuits. Hence, an alternative method to find solutions for these difficulties faced in an analog circuit design is to redesign these analog circuits in a digital manner. This is an irresistible technique as it put the circuit designer in the world of just two logic levels, *i.e.* 0 or 1, thus removing complexities from everything. By employing this method, design efforts and time-to-market can be reduced and porting to new technologies becomes easier.

In CMOS circuits, fundamental power utilization by the circuit is due to charging or discharging of node capacitances. Diminishing the power supply voltage for similar clock period results in a quadratic impact on limiting the energy dispersal. In this manner, the digital based designs can be redesigned for energy-efficient applications in newer CMOS technologies which are scaled. In analog designs supply voltage scaling degrades the performance. Scaling reduces the transistor device size which leads to reduction in the node capacitances and leakage current, since both parameters are in direct proportion to transistor size. Energy-efficient devices are in great demand in this universe of power-hungry and faulty appliances [1].

Arithmetic circuits are the heart of every digital system. Everything else is side-dish. They determine the performance of the system and dictate the parameters like clock rate, speed, and area. If arithmetic circuits are optimized, the performance will definitely improve and there are always opportunities for improvement.

Arithmetic circuits are the standard model for computing polynomials. Colloquially, an arithmetic circuit takes as inputs either variables or numbers, and is allowed to either add or multiply two expressions already computed. Arithmetic circuits give us a conventional way for understanding the complexity of computing polynomials. Arithmetic circuits in real computers are constructed from combinational elements that are interconnected by wires. The optimization of arithmetic circuits lead to the optimization of whole design. The achievement of an arithmetic circuit hugely relies upon its structural portrayal. For instance there are a wide range of different adders, *e.g.*, the Carry Look-ahead Adder, Ripple Carry Adder, Carry Select adder and so forth. A few models are minimized for, area, while some for critical path delay. Since distinct architectures effectively implement the similar Boolean expression, out of numerous identities one identity can be utilized, *e.g.*, $abc = (a \oplus b \oplus c)(a.b \oplus b.c \oplus a.c)$, to change over one form to other. In any case, because of the wide range of relations and the restrictions of algebraic factorization for finding fitting solutions automatically, the current logic synthesis techniques are not able to apply numerous identities. Where a digital circuit is executed in a specific architecture, synthesis tools just investigate a little design space according to input specifications. The need to perform physical synthesis was emphasized to get a better estimation of delays and shorten the time-to-market.

Minimization of VLSI circuits counts largely on proficient implementation of arithmetic operations keeping in mind signal delay, power consumption, and chip area. Methodologies for computing rely on the fact that many applications like multimedia, image processing, can tolerate some errors and erratum in computation and, hence, the solution can tolerate some degree of uncertainty. Commonly used multimedia applications have Digital Signal Processing (DSP) blocks as their backbone. Multiplication is commonly required in digital signal processing and act as one of the fundamental building block. Multiplier is an arithmetic circuit. Parallel multipliers give high speed method for multiplications but, require large area for

VLSI implementations. The increase in use of portable communication, computing devices and advancement in mobile multimedia applications has made power consumption and area critical to optimize in the design of digital signal processing architectures. The speed of arithmetic circuits for instance adders and multiplier mainly determines the performance of most of the very-large-scale-integrated circuits such as digital signal processing chips.

Many widely used applications today, such as signal processing, pattern recognition, and machine learning, do not require perfect computation. Instead, results with small errors are still acceptable. A new design paradigm, known as approximate computing [2], is recently proposed to design circuits for those error-tolerant applications. Exploiting the error tolerance of applications, it deliberately sacrifices a small amount of accuracy to achieve improvement in area, performance, and power consumption. High-performance energy-efficient approximate arithmetic circuits [3] such as adders and multipliers are designed by deliberately introducing errors [4].

Arithmetic circuits are also used in stochastic computing [5]. A necessary part of stochastic computing is the input stochastic bit streams. Generating these random bits usually takes a lot of hardware area. Various algorithms are explored to automatically synthesize stochastic circuits for arbitrary arithmetic functions [6]. Stochastic computing is applied to different applications, such as image processing and numerical integration [7].

The finish of the most exceptional synthesis techniques is hugely reliant on the input information of the circuit in numerous cases. Logic synthesis performs local optimizations quite productively once the input specification is changed to an appropriate design. Despite the remarkable development of logic synthesis [8] in the previous years, however locating the optimal structural design for a given circuit still happens to be a reasonable and to a great extent irresolvable issue, particularly for arithmetic circuits.

The efficient synthesis of circuits is a well-studied problem. However, the heuristics presented by several researchers in the past, which are also adopted by commercially available tools, are able to generate near-optimal design implementation for most arithmetic circuits. These heuristics exploit the rules of Boolean algebra, involving the

logical operations OR and AND, in order to transform the circuit. The approach works well for common logic circuits where OR and AND gates constitute the major portion of the circuitry. On the other hand, on arithmetic circuits including a large proportion of XOR gates in addition to the other two basic types, these heuristics perform poorly. For arithmetic circuits, current logic synthesis tools generate design implementations which are far from optimal.

The challenges in the efficient synthesis of arithmetic circuits are explored and a set of efficient algorithms to overcome these challenges are presented in this work. The presented algorithms vary in their computational complexity, and granularity of the circuit details. A methodology to combine these algorithms is presented so that they can be applied on larger circuits without losing performance. The presented algorithms have been tested on a wide range of arithmetic circuits. Although the algorithms presented here are not specific to arithmetic circuits and can be applied to any circuit, due to higher complexity compared to other logic synthesis heuristics, the use of the proposed methods is recommended for arithmetic circuits.

Boolean function manipulation [9] forms vital part of numerous logic synthesis techniques, including logic optimization and logic verification of combinational and sequential circuits. Synthesis, verification, and testing algorithms of VLSI circuits manipulate a extensive number of switching functions. Therefore, it is important to have effective methods to symbolize and keep in line such functions. A large class of problems in the area of VLSI CAD can be solved by espousing effective underlying data structures. During the last decade, several methods based on Decision Diagram have been suggested and successfully used in many industrial applications. A Boolean function defines a digital circuit that can be pictured as a Binary Decision Tree (BDT), which is a directed non-cyclic graph favouring an arrangement of properties with respect to an order. The size is determined by count of its non- terminal nodes. In multiplexer based logic styles *i.e.* Pass Transistor Logic (PTL), a lesser node count straight forwardly shows little chip area. A BDT can be actualized in canonical structures such as reduced ordered binary decision diagram ROBDD or ordered binary decision diagram OBDD. On the off chance indistinguishable nodes / hubs are shared and surplus nodes /hubs are ruled out, the OBDD is abbreviated and is called ROBDD.

The size of a BDT is firmly subjected to the variable ordering of inputs. Unusually the area of an OBDD increases exponentially by utilizing one variable order, but linearly utilizing another variable order. It is critical to keep the size of the OBDD as little as possible, because the memory requirement and the CPU processing time increase with the increment in the OBDD area. Hence, a good number of heuristics and algorithms have been evolved to determine the optimal variable ordering of inputs.

1.2 Binary Decision Tree (BDT)

BDT symbolizes a arrangement of Boolean functions and is described as a multi-rooted directed acyclic graph (DAG) [9][10]. Each node/hub in the DAG symbolizes a Boolean function F and has an related variable x_i and pointer to two different nodes (functions) in the DAG. The node / hub F is composed as the tuple (x_i, G, H) where x_i is called the best factor of the function F, G is the positive cofactor of F with respect to xi *i.e.* G = F_{xi}, and H is the negative cofactor of F with respect to x_i *i.e.* H = Fxi′. At every node Shannon decomposition is conducted and node F is represented as F = xiG + xi′H. The constant functions 0 and 1 are shown by terminal nodes. BDTs are not only used for hardware verification but also for logic synthesis. These applications are benefitted from minimization of BDTs with respect to various objective Boolean expressions. In this, BDT minimzations materialize at a rooted and abstruse logic level. In a VLSI design flow, this is a beginning time before progression of technology mapping. After BDT minimization, it is usually conceivable to specifically transfer the accomplished optimizations to directed computerized designs. Given below are few couple of cases of BDT optimization and their applications:

i. A simulator assesses a given expression by coordinating use of the showing BDT in BDT-based functional simulation. A crucial factor is the runtime expected in assessing a Boolean expression. Subsequently, BDT minimizations are required to optimize assessment time [11],[12].
ii. To straight forwardly map the BDT to a multiplexer circuit is one approach to synthesize a circuit for a given Boolean expression. It is realized that improvements in BDD size and average path length specifically gets transferred to the determined circuit. Keeping node count and runtime into consideration, applications are focused in the design area of logic synthesis

1.3 Ordered Binary Decision Diagram (OBDD)

Bryant [2] proposed the Ordered binary decision diagram (OBDD). An Ordered Binary Decision Diagram (OBDD) is a set (π,G) where π signifies the variable ordering of the OBDD and G is a limited DAG *i.e.* G = (V,E) where, V indicates the pair of vertices and E denotes the pair of edges of the DAG, with precisely one root node and the following properties:

i. A node in V is either a non-terminal node or one of the two terminal nodes in {**1**, **0**}

ii. Each non-terminal node v is labelled with a variable in X_n, signified var(v), and has precisely two offspring nodes in V which are indicated then(v) and else(v)

iii. On every way from the root node to a terminal node the variables are encountered at most once and in the similar order.

Thus, OBDD [13] is a restrained choice graph in which the variable ordering is uniform on all ways of the diagram. The order selected for decision variables can

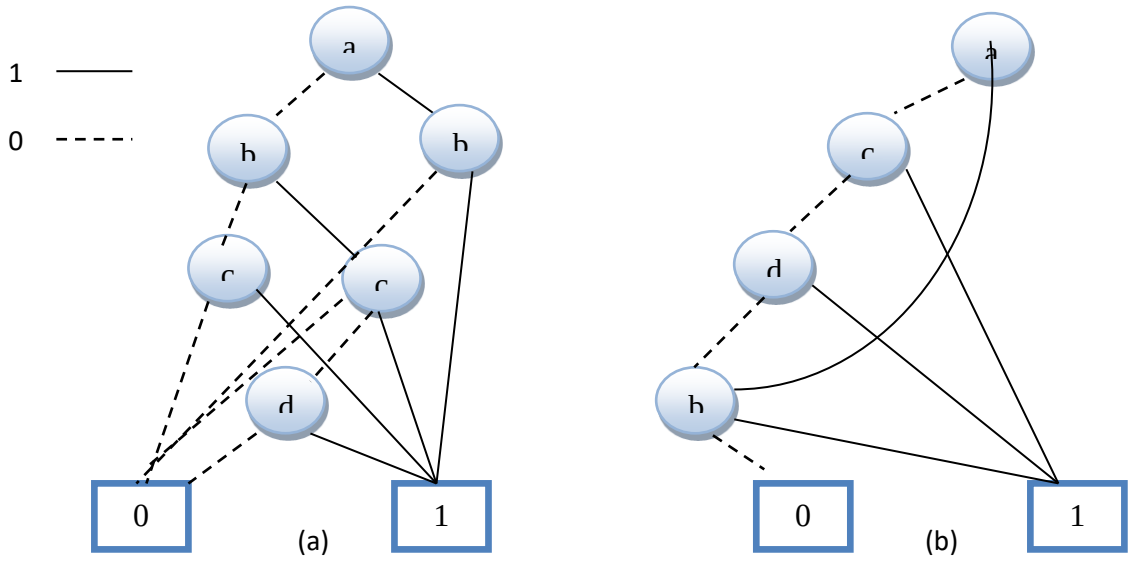


Figure. 1.1 Different OBDDs for same Boolean function F with different variable order
F = ab + a'c + bc'd

fundamentally influence the node count in a binary decision diagram. The BDD in Figure 1.1 (a) utilizes input variable order a < b < c < d, while the BDD in Figure 1.1 (b) depicts the similar expression, with a distinct variable order a < c < d < b. Both figures have the same function but different number of nodes. These sub graphs can be shared and the subsequent diagram stays canonical as a result of this ordering restriction. Thus, for a given logic function and a given variable order, the diagram is special.

1.4 Reduced Ordered Binary Decision Diagram (ROBDD)

The reduced order binary decision diagram representation has to be defined by levying restrictions on the BDD specification of Akers [10] such that the resulting form is canonical. BDD is said to be decreased, on the off chance that it consists no isomorphic sub diagrams and each node has particular kids. Reduced order binary decision diagrams ROBDDs [9] are canonical depictions of Boolean expressions. Two functions are said to be equivalent if and only if their binary decision diagrams are identical for a fixed variable order. The following are the reduction rules for converting an OBDD to an ROBDD:

a. Merging equivalent nodes / hubs
b. Merging isomorphic nodes /hubs
c. Eliminating redundant tests

Redundant nodes can be eliminated in the diagram *i.e.* nodes which are superfluous do not represent a Boolean expression. Reduced order binary decision diagrams permit a reasonable trade-off among productivity of control and minimization. Contrasted with different methods to represent Boolean expressions functions, *e.g.* truth tables or Karnaugh maps, reduced order binary decision diagrams ordinarily need considerably less memory space. Quicker techniques do exist for their handling. Table 1.2 demonstrates the truth table for a Boolean function and the related OBDD and ROBDD.

Table 1.2 Truth Table for Boolean function f = a.b+c

a	b	C	f
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	1

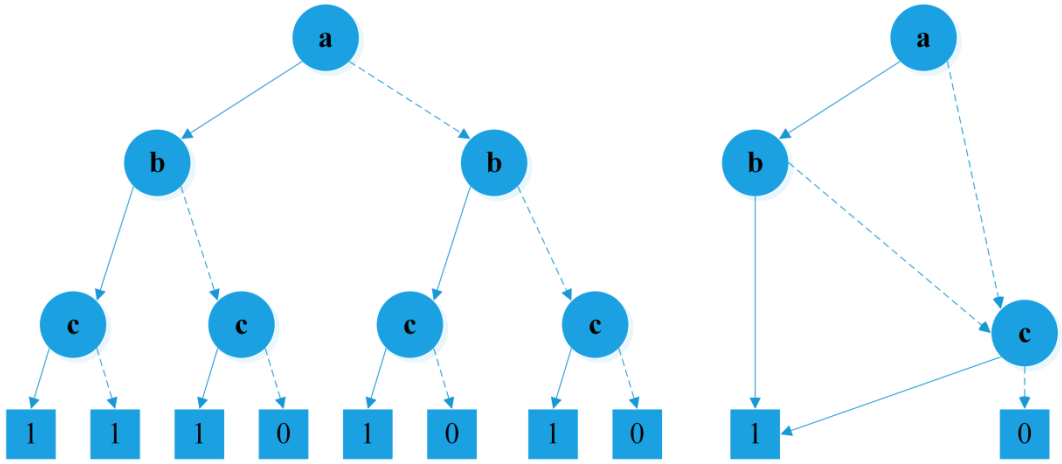


Figure 1.2 Ordered BDD and corresponding ROBDD for the Boolean function, f = ab+c

1.5 Applications of BDDs

Binary Decision Diagrams (BDDs) have accomplished incredible ubiquity as data structures for depicting Boolean expressions in determining majority of the combinational issues that emerge in physical synthesis and logic verification of digital systems [9] [10] [11]. The two important properties of BDDs which form the basis for applications of BDDs in different areas are:

i. BDDs are canonical. This property is valuable when confirming the equality between two Boolean functions [11]. Accordingly for a given variable ordering, two functions are said to be equal if and only if their binary decision diagram are indistinguishable [14].

ii. BDDs are powerful data structures for the depiction of extensive combinatorial arrangements.

1.5.1 Functional Synthesis: BDDs as MUX Circuits

Multiplexer based logic styles like transmission gate or pass transistor gate can be utilized to acknowledge Boolean functions, represented by BDDs. The CMOS transmission gate consists of one nMOS and one pMOS transistor, connected in parallel. Pass gate logic utilizes just two transistors to realize a multiplexer *i.e.* a wired OR of two MOS transistors. The pass transistor logic circuits are utilized as a substitute to static CMOS design because of their advantages like higher energy

efficiency (low power), reduced circuit area and better circuit speed. Each node in the corresponding BDD for a Boolean function represents a 2x1 MUX. In 2x1 multiplexer when select line, *s* is set to zero then output *y* is equal to *a* and when *s* is set to one then output is *b* which is shown in Figure 1.3 and is expressed logically as $y = s'.a + s.b$. The MUX representation is obtained by back propagation from leaf (terminal) nodes to the root nodes of the BDD, as shown in Figure 1.4. Hence, reduction in node count is desirable in order to minimize the net area for realization and fabrication of a digital circuit.

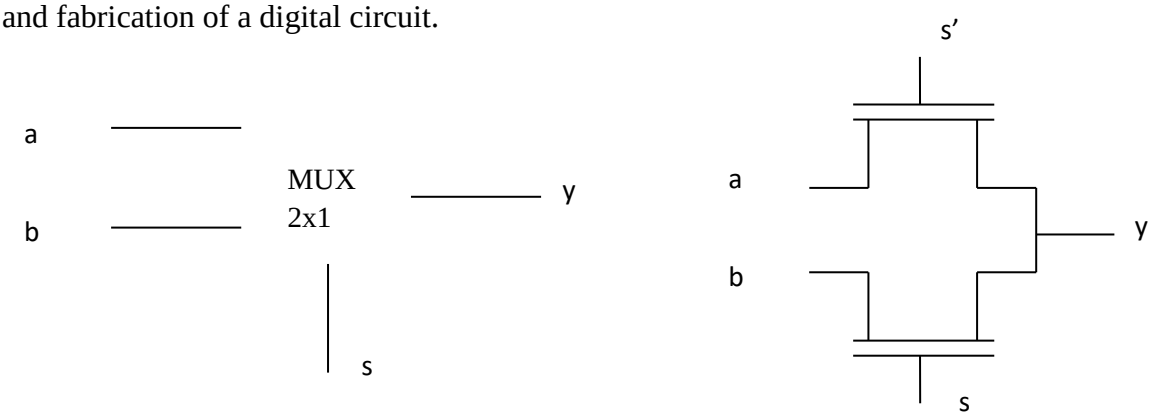


Figure. 1.3 A 2x1 Multiplexer and corresponding transistor realization

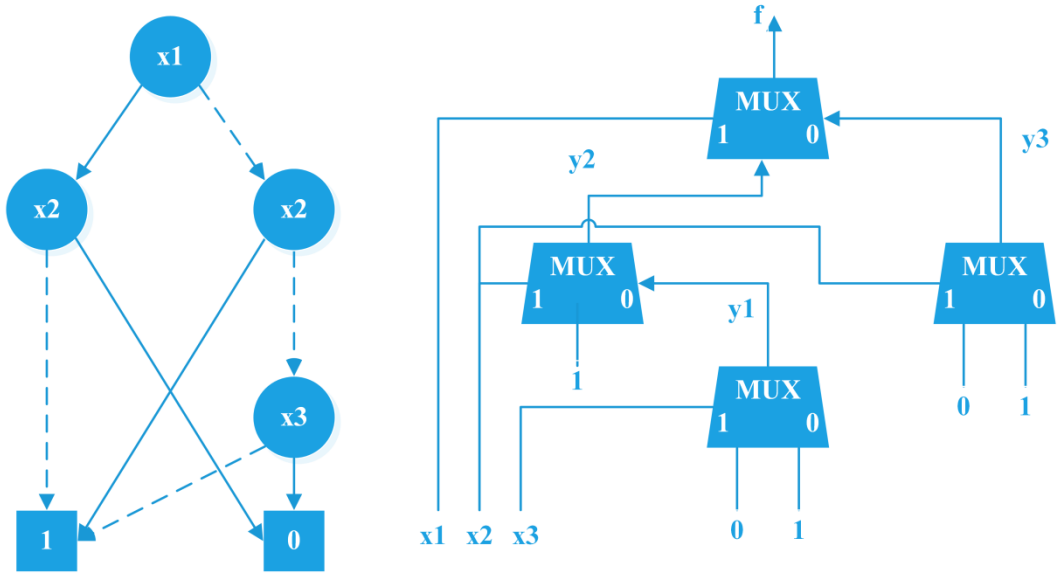


Figure. 1.4 ROBDD and corresponding MUX representation for boolean function, f = x1x2'+ x1' (x2'x3'+ x2)

1.5.2 Functional Verification of Logic Circuits

Binary decision diagrams (BDDs) were basically discovered for hardware verification to proficiently stock a vast number of nodes and these nodes share many things which are common [9]. The canonical property of BDDs makes them an efficient alternate approach for logic circuit comparison. The basic aim for hardware verification is to

compare a new design to a *known good* design [15]. Two logic circuits are equivalent if and only if their compact representations in the form of BDDs are identical provided the same variable ordering is used in both representations as illustrated in Figure 1.5. On applying the reduction rules, *i.e.* eliminating redundant test, ROBDD for function f1 is obtained with variable order a < b < c as shown in Figure 1.5 (b). ROBDD for function f2 = ab + a'c with order a < b < c is shown in Figure 1.5 (c). The two figures being identical show that the two Boolean functions f1 and f2 are identical.

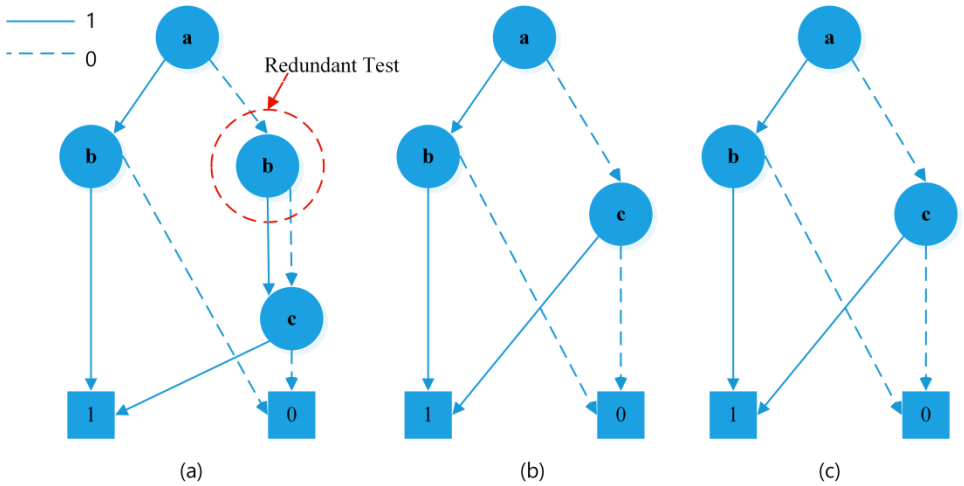


Figure 1.5 Equivalence Verification of Boolean functions f1 = ab+a'c+bc and f2 = ab+a'c

1.6 Motivation

The decomposition of binary decision diagrams and along with that memory requirement and runtime of algorithms utilizing binary decision diagrams regularly fundamentally relies totally on the order chosen for the input variables, varying it from linear to exponential [16]. Since recommending the best input variable ordering is NP-hard, hence various heuristics tend to discover nearly ideal solution for variable ordering of BDDs,. The BDD variable ordering algorithms are widely classified as static, dynamic and evolutionary variable ordering algorithms.

Static ordering [16]-[17] is adopted to fetch only the starting order of the input variables and the node order is ascertained before generating binary decision diagrams. Inspiration driving the evolutionary originates from perception that binary decision diagrams have a tendency to be compact when input variables related to each

other are close in the order. These variables determine sub circuit's outputs and in order should be near each other. Once determined, the order is kept up all through the subsequent processing. Then again, in order to limit the quantity of nodes the variables are reordered periodically in BDD package using Dynamic ordering heuristics [18]–[20]. In dynamic reordering, variable order is altered automatically by the BDD bundle, straightforward to the client. The BDD bundle decides the approximate points where to quit the handling, reorder the variables, and after that restart the optimization. Various available evolutionary algorithms [21], [22], [24], [27] and their hybrid versions [23], [25], [26] have been proposed and implemented to arrive at almost best results. Whereas most of these approaches have been implemented in the standard BDD bundles such as Buddy, CUDD *etc*., the algorithms proposed in the thesis has not been implemented in any such package, to the best of my knowledge.

1.7 Organization of Thesis

The thesis report is organized in the following manner.

- The thesis report begins with a brief introduction to Binary Decision Diagrams (BDDs), Ordered Binary Decision Diagram (OBDD) and Reduced Ordered Binary Decision Diagram (ROBDD).
- Chapter 2 discusses the literature survey.
- Chapter 3 gives insight into the problem, research gaps, objectives, and research methodology.
- Chapter 4 gives an overall view of different optimization algorithms and their limitations and advantages. Static, Dynamic and Evolutionary Algorithms are being seen as a potential solution to variable ordering problems in BDDs.
- Chapter 5 provides a detailed narration of the Proposed Genetic Algorithm, Hybridized Genetic Algorithm and Modified Memetic Algorithm (MMA), along with their flow charts and applications using the standard BDD package.
- Chapter 6 presents an explanation of power estimation techniques and estimation of power using probabilistic techniques for switching power dissipation.

- Chapter 7 demonstrates test outcomes for different LGSynth93 Benchmark circuits showing the efficiency of the proposed methods by comparing them with existing variable ordering methods.
- Chapter 8 presents the conclusions and future scope of work.

CHAPTER 2
LITERATURE SURVEY

Akers [9] thought of a technique for characterizing, breaking down, testing, and actualizing substantially large digital functions by methods for binary decision diagrams.

Bryant [10] recommended and proposed diagram based algorithms for Boolean function manipulation.

Liu *et al.* [12] proposed methods to generate a Binary Decision Diagram (BDD) with least possible expected path length.

Sasao [15] came up with and evolved a simplified algorithm EXMIN 2 for XOR sum-of-product Boolean expressions for multiple-input and two-output Boolean functions.

Fujita *et al.* [16] presented the experimental results regarding ordering of variables using a practical algorithm called depth-first traversal through a circuit from the output to the inputs.

Fujii *et al.* [17] defined and described variable ordering approaches for multi-output benchmark circuits. For some of the benchmark circuits, ordered binary decision diagrams have been strongly produced by utilizing the newer algorithms. Accordingly, the proposed variable ordering approaches seems to be intensive and OBDD-based CAD tools can be applied to more extensive class of VLSI circuits.

Richard Rudell [18] talked and proposed another OBDD minimization algorithm, called the sifting algorithm, to do reordering of the variables of the OBDD.

Felt *et al.* [19] displayed a proficient heuristic algorithm for progressively diminishing the size of vast reduced order binary decision diagrams ROBDDs by optimally reordering little window of consecutive variables.

Meinel *et al.* [20] suggested and proposed another algorithm, known as linear sifting, for the minimization of binary decision diagrams which joins the efficiency of sifting and the energy of linear transformations. This linear sifting technique is relevant to extensive Boolean functions, and prompts significantly more smaller diagrams when contrasted with normal variable reordering algorithm. Linear sifting lessened the combined BDD size with a normal change of 22% in the node count within a wide range of trials.

Hung *et al.* [21] offered an acceptable compromise between BDD node size reduction and runtime using scatter search optimization. On smaller benchmarks it provides optimal reduction in BDD size with less runtime, but for bigger benchmark circuits lesser BDD sizes are obtained at the cost of longer runtime in comparison with genetic algorithm or simulated annealing.

Drechsler *et al.* [22] offered by experimental testing that better BDD sizes can be obtained by utilizing genetic algorithms. It created optimal outcomes for most thought about benchmarks, still it is applicable to functions with in excess of 20 input variables because of its shorter runtimes.

Cheung *et al.* [23] proposed an efficient algorithm for technology mapping targeting table look-up (TLU) blocks.

Chaudhary *et al.* [24] formulated and prescribed two algorithms, specifically, a genetic algorithm along with a branch and bound algorithm to generate an best variable order. The reordering of nodes/hubs was dealt with care by the standard BDD package-2.4. These variable reordering algorithms helps in improving the BDD size *i.e.* area and power dissipation.

Takapoo *et al.* [25] used ID3 as a heuristic method to improve the genetic algorithm that has been used for reordering BDD. The improvement was obtained in terms of decreased node number and was 11.183% and decreased number of iteration was around 25.79%.

Mingquan *et al.* [26] presented another new BDD optimization approach in light of a genetic tabu hybrid strategy that consolidates the worldwide search space of genetic technique with the nearby neighbourhood seek and the worldwide streamlining of tabu search.

Mitra *et al.* [27] presented a particle swarm optimisation (PSO) based technique to deal with minimize BDD size as far as number of nodes and average path length of the binary decision diagram (BDD) is concerned.

Emad Elbeltagi *et al.* [28] looked at the after effects of five recent evolutionary-based algorithms: genetic, memetic, particle swarm, ant-colony systems, and shuffled frog leaping and compared them.

Merz *et al.* [29] presented comparison between genetic local search technique to the quadratic assignment problem (QAP).

Liu *et al.* [30] A hierarchical structure poly- particle swarm optimization (HSPPSO) approach utilizing the hierarchical structure idea of control theory is exhibited.

Rojas *et al.* [31] proposed another memetic algorithm (MA) for taking care of constrained minimization problems over unending search spaces. Memetic Algorithm is composed by a global search mechanism based on differential evolution (DE), a constraint- handling technique called stochastic ranking (SR) and a local search (LS) procedure which adopts a simplex crossover operator. We show that the performance of our algorithm is improved by the influence of its LS mechanism.

Lenders *et al.* [32] described the utilization of genetic algorithms to enhance the input variable ordering of a BDD. A new crossover technique was presented that in combination with sifting algorithm as hybridization technique ended up being extremely valuable.

Drechsler *et al.* [33] presented a new BDD – based design style that thinks about switching activity optimization. Experimental trials on a set of MCNC and ISCAS89 benchmarks demonstrated that reduction in power estimate ranges from 0% to 95%.

Cheng *et al.* [34] targeted field-programmable gate array (FPGA) performance optimization using a novel binary decision diagram (BDD)-based synthesis paradigm. In this paper, the author concentrated on delay reduction and concluded
that there is a large optimization margin through BDD synthesis for FPGA performance optimization. The contributions made by them were:
i) A gain-based clustering and partial collapsing algorithm was proposed to prepare the initial design for BDD synthesis for better delay;
ii) A technique called linear expansion was used for BDD decomposition,
iii) Special decomposition scenarios coupled with linear expansion were considered for further improvement on the quality of results.
Experimental results showed that a 30% performance gain can be achieved with
a 22% area overhead on the average compared to a previous state-of-the-art BDD-based FPGA synthesis tool, namely, BDSpga. Compared to DAOmap, a 33% performance gain with only an 8% area overhead and compared to the ABC mapper, 20% performance gain with only an 8% area overhead can be achieved.

Robert Meolic [35] developed Biddy a BDD package under GPL at University of Maribor. It utilizes ROBDDs with complement edges.

Raseen *et al.* [36] developed a numerical method for the size variation of the ROBDD for various min-terms and various values of don't cares. The size of ROBDD was found by simply assigning zero for the number of don't cares.

Singh *et al.* [37] developed an automated CDISI procedure and formulated a technique such that the proposed method can be easily implemented on a chip.

Dutta *et al.* [38] proposed a robotized floor plan utilizing genetic algorithm. The algorithm is assessed for the uniqueness of solution and for the generation of alternative designs, subsequently offering the client / user some elective alternatives to pick.

Sidhu *et al.* [39] introduced a novel approach in light of the Harmony Search Algorithm (HSA) to ideally discover reasonable wavelet functions and wavelet disintegration levels for exact fault characterization in Transmission Lines (TLs).

Dhurandher *et al.* [40] utilized the thoughts of naturally occurring ants foraging behavior and composed an energy-efficient routing protocol. It comprises of the impact of energy utilization in routing packets and the multi-path transmission properties of ant swarms are accomplished which bulids up the battery life of a vertex / node.

Dhurandher *et al.* [41] formulated a convention, quality-based distance vector routing (QDV), for securing wireless sensor networks (WSN) utilizing ideas that are based on Ant Colony Optimization (ACO). Two essential performance parameters are utilized i.e. quality of service (QoS) and reputation.

Chaudhary *et al.* [42] proposed a Genetic Algorithm based output phase selection of the multi-output function (in BDD form). In Binary Decision Diagram (BDD) based realization of logic circuits, the area and power consumption is determined by the total number of BDD nodes and the expected switching activity of the nodes. A proper polarity selection of the sub-functions can not only reduce the number of BDD nodes, but also the switching. For a multi-output function, more the sharing between the sub-functions, more is the minimization in the number of BDD nodes and also the power. This idea was applied to a number of benchmark circuits, which resulted in reduction in the number of BDD nodes and hence the area and power. A trade-off has also been done for combined area and power minimization, considering the node switching as the major candidate for power consumption. It has been found that the proposed method minimizes the area by about 3% and power by about 14%.

Drechsler *et al.* [43] applied genetic algorithm to find a variable ordering that reduces the size of ordered binary decision diagrams (OBDDs). Optimal results were obtained for most considered benchmarks. Genetic Algorithm is also applicable to functions with more than 20 variables due to short runtimes.

Ebendt *et al.* [44] presented a new branch and bound technique for determining an optimal variable order. In contrast to all previous approaches that only considered one lower bound, they used a method which makes use of a combination of three bounds and, by this, avoided unnecessary computations. The lower bounds were derived by generalization of a lower bound. They allow one to build the BDD either by top down approach or bottom up approach. Experimental results were reported that clearly demonstrated the efficiency of both top down and bottom up approach. A comparison to the best minimization algorithm has shown that runtime can be reduced by up to 49%. The new bottom up approach results in a speed up by two orders of magnitude compared with the best bottom up approach.

Chaudhary *et al.* [45] proposed two algorithms namely, a genetic algorithm based technique and a branch and bound algorithm to find an optimal input variable order. The node reordering is taken care of by the standard BDD package buddy-2.4. The performances of the proposed algorithms have been evaluated by running an exhaustive search program. Experimental results showed a substantial saving in area and power.

Vemuri *et al.* [46] described an FPGA specific logic synthesis approach, which unites multi-level logic transformation, decomposition, and optimization techniques into a single synthesis framework. These techniques were built upon a BDD-based logic decomposition system. With this system, both AND-OR decompositions and AND-XOR decompositions can be identified, which results in large area savings for synthesized XOR-intensive circuits. To induce good decompositions, a maximum fan out free cone (MFFC) based partial clustering and collapsing technique is used. This step is followed by an area-minimizing variable partitioning heuristic which decomposes collapsed nodes into LUT-feasible sub-functions. As a post-processing step, a performance-driven re-synthesis phase is performed to alleviate increased delay caused by excessive logic sharing. Experimental results indicated that the circuits generated by these techniques are not only smaller, but are also significantly faster than those synthesized by conventional FPGA synthesis tools. Furthermore, the computation times required by our techniques are significantly smaller than those of previous techniques.

Muma *et al.* [47] proposed a novel method to improve the utilization efficiency and performance of field-programmable gate arrays (FPGAs). The proposed method, ExorBDD, used a stage of exclusive-sum-of-product (ESOP) minimization, followed by a stage of decomposition using binary decision diagrams (BDDs). For exclusive OR (XOR)–intensive circuits, experiments were conducted on 19 MCNC benchmark parity circuits (ranging from 5 to 25 inputs). The results using the proposed approach show significant improvements over Exorcism4, BDS, and commercial tools. On average, the new approach uses only 30.3% as many look-up tables as are used by Xilinx tools (and only 16.4% in comparison to Altera). On average, the new approach has a maximum combinational path delay of 89.2% compared to the delay with Xilinx tools (80.3% compared to Altera). Experiments were also conducted on non-XOR-intensive circuits. These results show that ExorBDD also performs well for arbitrary circuits.

Yang *et al.* [48] described a novel logic decomposition theory and a practical logic synthesis system, BDS. It is based on a new binary decision diagrams (BDD) decomposition technique which supports all types of decomposition structures, including AND, OR, XOR, and complex MUX, both algebraic and Boolean. The method is very efficient in synthesizing both AND/OR and XOR-intensive functions. It also has a capability to handling very large circuits, as it employs the BDD decomposition in the partitioned Boolean network environment. The experimental results show that BDD-based logic decomposition is a good alternative to the existing logic optimization approaches as, it offers a superior runtime advantage over traditional logic synthesis systems.

Chen *et al.* [49] studied the technology mapping problem for FPGA architectures to minimize chip area, or the total number of lookup tables (LUTs) of the mapped design, under the chip performance constraint. Experimental results show that our mapping algorithm, named DAOmap, produces significant quality and runtime improvements. Compared to the state-of-the-art, depth optimal, area minimization mapping algorithm CutMap [21], DAOmap is 16.02% better on area and runs 24.2X faster on average when both algorithms are mapping to FPGAs using LUTs of five inputs.

Lin et al. [50] developed an algorithm SMAC (simultaneous mapping and clustering) that performs mapping and clustering simultaneously and optimally under a delay model widely used. It is the first delay-optimal algorithm to generate a synthesis solution that considers a combination of both steps. Compared to a synthesis flow using state-of-the-art mapping and clustering algorithms DAOmap [7] + T-VPACK [17] - SMAC achieves a 25% performance gain with a 22% area overhead under the clustering delay model. After placement and routing, SMAC is 12% better in performance.

Ling et al. [51] exploited the efficiency of BDD operations for new areas in CAD including cut generation and clustering by reducing these problems to BDDs and solving them using Boolean operations. The authors showed that applying BDD reduction to cut generation and clustering dramatically improves their scalability. As a consequence, this technique is an order of magnitude faster than previous techniques and, as a result, this work can be applied to new areas in the CAD flow. This new elimination algorithm results in an overall speedup of 6x in FBDD with no impact on circuit area.

Table 2.1 Comparison of Various Performance Parameters for different Optimization Algorithms / Techniques / Approaches.

Ref. No.	**Year**	**Approach/Algorithms/ Techniques applied**	**Area / Number of LUTs**	**Maximum Combinational Path Delay**	**Power**	**Place & Route Delay**	**Perfor-mance Gain**	**Runtime**
[18]	1993	Dynamic variable ordering	45% ↓	-	-	-	-	-
[42]	2006	Output Phase Assignment	15.72 % ↓	-	19.18 % ↓	-	-	-
[43]	2006	Scatter Search Heuristics Algorithm	48.1625 % ↓	-	-	-	-	-
[44]	2011	Branch & Bound Based Optimization	53.5145 % ↓	-	-	-	-	-
[45]	2011	Genetic Algorithm Based Optimization	51.9807 % ↓	-	-	-	-	-
[46]	2002	BDD- Based Logic Synthesis	41.6% ↓	93.9% ↓	-	-	-	-
[47]	2008	ESOP with BDD	73.1% ↓	91.5% ↓	-	-	-	-
[48]	2002	BDS compared to SIS map	25% ↓	41% ↓	-	-	–	Improved
[49]	2004	DAO map	16.02% ↓	-	-	-	-	24.2X ↓
[34]	2008	Delay Driven BDD (DDBDD) compared to BDS-pga	22% area ↑	-	-	-	30% ↑ w.r.t. [21]	-
		Delay Driven BDD (DDBDD) compared to DAO-map	8% area ↑	-	-	-	33% ↑ w.r.t. [22]	-
		Delay Driven BDD (DDBDD) compared to	8% area ↑	-	-	-	20% ↑	-

		ABC-mapper						
[50]	2006	Simultaneous Mapping & Clustering (SMAC)	22% area ↑	-	-	12.3% ↓	25%	100X ↑
[51]	2008	Scalable Synthesis & Clustering	15% area ↑	-	-	4-8% ↓	-	10X ↑

2.1 State – of – the – Art (SOA) Analysis

Characterization and testing of substantially large Boolean functions proposed in [9] was further modified by Bryant [10] in which he suggested a better algorithm called diagram based algorithm for Boolean function manipulation but with high expected path length. Another method to reduce the possible expected path length was proposed by Liu [12]. For multiple and two output Boolean functions, a simplified algorithm EXMIN 2 for XOR sum-of-product was proposed by Sasao [15]. Experimental results regarding ordering of variables using a practical algorithm called depth-first traversal through a circuit from the output to the inputs were proposed by Fujita [16]. For some of the benchmark circuits, ordered binary decision diagrams have been strongly produced by utilizing the newer algorithms. Accordingly, the proposed variable ordering approaches seems to be intensive and OBDD-based CAD tools can be applied to more extensive class of VLSI circuits. For this, Fujii [17] defined and described variable ordering approaches for multi-output benchmark circuits. To do reordering of the variables another efficient algorithm called sifting algorithm was proposed by Rudell [18]. In an another method, Felt *et al.* [19] displayed a proficient heuristic algorithm for progressively diminishing the size of vast reduced order binary decision diagrams ROBDDs by optimally reordering little window of consecutive variables. For minimization of binary decision diagrams which joins the efficiency of sifting and the energy of linear transformations, a linear sifting algorithm was proposed by Meinel [20]. This linear sifting technique is relevant to extensive Boolean functions, and prompts significantly smaller diagrams when contrasted with normal variable reordering algorithm. Linear sifting lessened the combined BDD size with a normal change of 22% in the node count within a wide

range of trials. So far the literature survey talks about only node count reduction but as the complexity increases the runtime also becomes important. Therefore an acceptable compromise between BDD node size reduction and runtime using scatter search optimization was proposed by Hung [21]. On smaller benchmarks it provides optimal reduction in BDD size with less runtime, but for bigger benchmark circuits lesser BDD sizes are obtained at the cost of longer runtime in comparison with genetic algorithm or simulated annealing. Another algorithm for better BDD size based on genetic algorithms was proposed by Drechsler [22]. It created optimal outcomes for most thought about benchmarks; still it is applicable to functions with in excess of 20 input variables because of its shorter runtimes. This work was further carried on by Chaudhary [24] in which he formulated two algorithms, specifically, a genetic algorithm along with a branch and bound algorithm to generate an best variable order. The reordering of nodes/hubs was dealt with care by the standard BDD package-2.4. These variable reordering algorithms helps in improving the BDD size *i.e.* area and power dissipation.

In order to reduce the number of iterations and computational power, a heuristic method was proposed by Takapoo [25] to reorder BDD. The improvement was obtained in terms of decreased node number and was 11.183% and decreased number of iteration was around 25.79%. Another BDD optimization was presented by Mingquan [26] called genetic tabu hybrid strategy that consolidates the worldwide search space of genetic technique with the nearby neighbourhood seek and the worldwide streamlining of tabu search. As far as number of nodes and average path length of the binary decision diagram (BDD) is concerned, Mitra *et al.* [27] presented a new approach called particle swarm optimization (PSO) based technique to minimize the BDD size. Emad Elbeltagi *et al.* [28] looked at the after effects of five recent evolutionary-based algorithms: genetic, memetic, particle swarm, ant-colony systems, and shuffled frog leaping and compared them. These techniques can be applied on BDD to reduce the number of nodes and runtime. Utilizing the hierarchical structure idea of control theory, Liu [30] presented an optimization technique called hierarchical structure poly- particle swarm optimization (HSPPSO). To take care of constrained minimization problems over unending search spaces a new approach called memetic Algorithm is presented by Rojas [31]. It is composed by a global search mechanism based on differential evolution (DE), a constraint- handling technique called stochastic ranking (SR) and a local search (LS) procedure which

adopts a simplex crossover operator. It shows that the performance of the above algorithm is improved by the influence of its LS mechanism. A new BDD – based design style that thinks about switching activity optimization was presented by Drechsler [33]. Experimental trials on a set of MCNC and ISCAS89 benchmarks demonstrated that reduction in power estimate ranges from 0% to 95%. A novel binary decision diagram based synthesis paradigm was proposed by Cheng [34] targeting FPGA performance optimization. Experimental results showed that a 30% performance gain with a 22% area overhead was achieved on the average compared to a previous state-of-the-art BDD-based FPGA synthesis tool, namely, BDSpga. A BDD package under GPL at University of Maribor was developed by Robert Meolic [35]. It utilizes ROBDDs with complement edges. A numerical method for the size variation of the ROBDD for various min-terms and various values of don't cares was developed by Raseen [36]. The size of ROBDD was found by simply assigning zero for the number of don't cares. A robotized floor plan utilizing genetic algorithm was proposed by Dutta [38]. The algorithm is assessed for the uniqueness of solution and for the generation of alternative designs. Another novel approach in light of Harmony Search Algorithm was introduced by Sidhu [39] in Transmission lines which can be applied to BDD for reducing node count. The thoughts of naturally occurring ants foraging behavior were utilized by Durandher [40].The same can be utilized for reducing the BDD sixze and runtime. An FPGA specific logic synthesis approach, was described by Vemuri [46] which unites multi-level logic transformation, decomposition, and optimization techniques into a single synthesis framework. Experimental results indicated that the circuits generated by these techniques are not only smaller, but are also significantly faster than those synthesized by conventional FPGA synthesis tools. Furthermore, the computation times required by our techniques are significantly smaller than those of previous techniques. Another novel method to improve the utilization efficiency and performance of field-programmable gate arrays (FPGAs) was proposed by Muma [47]. For exclusive OR (XOR)–intensive circuits, experiments were conducted on 19 MCNC benchmark parity circuits (ranging from 5 to 25 inputs). On average, the new approach uses only 30.3% as many look-up tables as are used by Xilinx tools (and only 16.4% in comparison to Altera). On average, the new approach has a maximum combinational path delay of 89.2% compared to the delay with Xilinx tools (80.3% compared to Altera). A novel logic decomposition theory and a practical logic synthesis system, BDS based on a new binary decision

diagrams (BDD) decomposition technique was described by Yang [48]. It supports all types of decomposition structures, including AND, OR, XOR, and complex MUX, both algebraic and Boolean. The experimental results show that BDD-based logic decomposition is a good alternative to the existing logic optimization approaches as it offers a superior runtime advantage over traditional logic synthesis systems.

CHAPTER 3

Problem Formulation

3.1 Research Gaps

1. Various optimization algorithms for variable ordering of BDDs are used for performance evaluation for logic circuits focusing primarily on area minimization. Mostly the state-of-art BDD-based logic synthesis algorithms for FPGAs were able to achieve performance in terms of gain and area overhead. But power minimization has not been considered so far.
2. An ExorBDD approach for synthesizing logic circuits into FPGAs shows very encouraging results with Xylinx synthesis tools and Altera Synthesis tools. An ExorBDD approach can be applied to other available synthesis tools like Synopsis and Cadence to find an improvement in average number of LUTs, and average delay.
3. For low power applications, ExorBDD approach can be applied to various synthesis tools.

3.2 Objectives

1. Study of different variable ordering techniques for minimization of logic networks for optimization of performance parameters like area, power, delay *etc*.
2. Study of different approaches/algorithms on the benchmark circuits to analyze the trade-off among various performance parameters using EDA tools.
3. To propose an approach/technique to minimize the network for optimized parameters.

3.3 Methodology

The methodology followed is as under:

1. *Study of available variable ordering techniques for minimization of logic networks for optimization of performance parameters like area, power and delay.*

   The available minimization algorithms were studied and their comparison was done based on study. Evolutionary Algorithms EAs are population-based metaheuristic optimization algorithms. These are stochastic search methods that mimic natural biological evolution and/or the social behaviour of species. The Evolutionary Algorithms have been developed to arrive at near-optimum solutions to complex and large-scale optimization problems which cannot be solved by gradient-based mathematical programming techniques [28]. Every EA has three main components:

   i.) **Population** of individuals representing candidate solutions.

   ii.) **Fitness function** that determines the environment within which the solutions live and the strength of an individual as an optimum solution to the given problem.

   iii.) **Evolutionary operators** leading to the evolution of population of individuals that are better suited to their environment than the individuals from where they were created.

   Thus, Evolutionary Algorithms work on populations of individuals instead of single solutions. In this way the search is performed in a parallel manner resulting in the faster convergence of these algorithms.

2. *Study of different approaches/algorithms on the benchmark circuits to analyze the trade-off among various performance parameters using EDA tools.*

   Various available approaches / algorithms were implemented on LGSynth93 benchmark circuits and multi-input adders to analyze the performance parameters like area and power. Simple genetic algorithm (SGA), hybrid genetic algorithm (HGA), and the latest particle swarm optimization algorithm (PSO) were implemented using C++ codes and the simulation was carried out on the BUDDY 2.4 package on Ubuntu 12.04.

   It is observed that in comparison with other existing algorithms, the proposed Genetic Algorithm and hybridized genetic algorithm gives minimum node count. Also, when the crossover operators are compared, Partially Mapped Crossover gives minimum nodes, while the Cycle Crossover gives minimum computation time for most of the circuits. The order crossover too shows optimum results. In

multi-input adder circuits, as number of inputs are increasing, the proposed genetic algorithm with PMX crossover gives best reduction in node count, where as with cycle crossover it gives best reduction in computation time. For 3-bit adder is about 18.75%, for 4-adder is about 44.44%, for 5-adder is about 65.07%, for 6-adder is about 77.07%, for 7-adder is about 85.82%. Therefore, the proposed Genetic algorithm and Hybridized Genetic algorithm is suitable for multi-input multi-output (MIMO) VLSI circuits.

3. *To propose an approach/technique to minimize the network for optimized parameters.*

   A comparative study of different optimization techniques was conducted to optimize power, area and performance for digital logic circuits. The results of MMA based program were compared with the other evolutionary algorithms i.e. simple genetic algorithm (SGA), hybrid genetic algorithm (HGA) and the latest particle swarm optimization algorithm (PSO) [27].However, results with HGA are different from those in [23] as parameter settings are different.

A comparison with the reordering heuristics have been incorporated in the BDD package *Buddy-2.4*, such as Window Permutation WIN2, WIN2ite, WIN3 and Sifting algorithm [18]-[19]. Percentage improvement was observed by using MMA as variable reordering algorithm with respect to original size and as compared to Sifting algorithm is also indicated. The results indicate that in 55.55% cases, our MMA based approach has resulted in lesser number of node counts for the SDDs. Also, for all circuits, number of iterations to generate optimum variable order is lesser than the corresponding HGA. Memetic Algorithm resulted in an average reduction of 24.03% in SDD sizes. Also the proposed MMA provides a 2.75% average improvement in node count of Benchmark circuits as compared to the best known dynamic algorithm *i.e.* Sift Algorithm.

- .

CHAPTER 4

Optimization Algorithms

4.1 Variable Reordering Problem in BDDs

The decomposition of binary decision diagrams and along with that memory requirement and runtime of algorithms utilizing binary decision diagrams regularly fundamentally relies totally on the order chosen for the input variables, varying it from linear to exponential [16]. Since recommending the best input variable ordering is NP-hard, hence various heuristics tend to discover nearly ideal solution for variable ordering of BDDs,. The BDD variable ordering algorithms are widely classified as static, dynamic and evolutionary variable ordering algorithms.

Static ordering [16]-[17] is adopted to fetch only the starting order of the input variables and the node order is ascertained before generating binary decision diagrams. Inspiration driving the evolutionary originates from perception that binary decision diagrams have a tendency to be compact when input variables related to each other are close in the order. These variables determine sub circuit's outputs and in order should be near each other. Once determined, the order is kept up all through the subsequent processing. Then again, in order to limit the quantity of nodes the variables are reordered periodically in BDD package using Dynamic ordering heuristics [18]–[20]. In dynamic reordering, variable order is altered automatically by the BDD bundle, straightforward to the client. The BDD bundle decides the approximate points where to quit the handling, reorder the variables, and after that restart the optimization. Various available evolutionary algorithms [21], [22], [24], [27] and their hybrid versions [23], [25], [26] have been proposed and implemented to arrive at almost best results. Whereas most of these approaches have been implemented in the standard BDD bundles such as Buddy, CUDD *etc.*, the algorithm proposed in the thesis has not been implemented in any such package, to the best of my knowledge.

4.2 Static Variable Ordering Techniques:

i. *Depth First Traversal Algorithm* [16]*:* It is based on a depth-first traversal from output to inputs through a logic circuit. It includes drawing a circuit diagram from a net list depiction, and after that utilizing the variable order showing up in the diagram. This algorithm depends upon the perception that input variables who are topologically near each other in the diagram seems closer in the variable order. In organized circuits, the obtained variable order is a standout among other requests. In any case, we might get great variable orders in some circuits, since mostly circuits are not tree organized. This algorithm has been produced for circuits with single output. For circuits with multiple outputs, the variable reordering algorithm is enforced by taking a duplicate output to which every main output is associated. The output order is resolved with an end goal that the output having more complex logic has the higher priority. This algorithm has the disadvantage that the variable order generated for the highest priority output node is used for other output nodes, even if it doesn't give the best result for them. Also, this algorithm cannot be used for larger circuits.

ii. *Variable Interleaving* [17]*:* These algorithms were developed by Fujii *et al.* for multiple output circuits. While conventional algorithms use variable appending, the new algorithms use variable interleaving and are based on circuit topology. The principle of the algorithm is to merge a variable order for an output into a variable order for outputs with higher priority maintaining the order for the output as much as possible. In the conventional algorithms, a newly ordered variable is always appended to the end of the ordered variable set, while in the new algorithm, a newly ordered variable is inserted into an appropriate position of the ordered variable set. These variable merging methods are called variable appending and variable interleaving, respectively. The order of outputs is determined in the depth first traversal way. The conventional algorithm is different from the new algorithms only in the variable merging method. These algorithms have been found to be much more effective than conventional algorithms and can be applied to wider class of circuits.

4.3 Dynamic Variable Ordering Techniques:

The idea behind dynamic variable reordering techniques is that the ordered binary decision diagram *i.e.* OBDD bundle decides and finds the variable order of the inputs. As the operations are performed transparent to the user, the order of input variable is altered naturally via OBDD bundle [18]. Since the order of variables inside OBDD is never fixed, this method is known as dynamic variable ordering. When utilizing dynamic variable ordering, a total order is described for all variables before and after every bundle activity. However, the order is systematically adjusted by the OBDD bundle, as an outcome of an activity, to locate a good order. Reasonably, the variable order alters in the middle of the package activities. Hence, advantages like canonicity and efficient recursive algorithms are implemented by the ordered binary decision diagram. Different algorithms executing this technique are:

i. **Window Permutation Technique** [18], [19]: This technique minimizes the size of an OBDD by utilizing side by side variable trade. The window permutation method continues by taking a level *m* in the directed acyclic graph and meticulously searching all *n!* combinations of the n adjoining variables beginning at level *m*. This is finished by utilizing n! - 1 set wise swaps followed by up to n (n - 1)/2 set wise swaps to re-establish the optimal result. These steps are then rehashed, beginning from every level unless any improvement in the DAG size is observed. After the change, a level is fixed which is known optimal. Since the exchange of two adjoining input variables is effective, the window permutation method stays handy for n values varying from 4 to 5**.** The window permutation method is restricted in its space to discover optimal variable orders. A restriction on this method seems that several steps are needed to move a variable to a long distance and these steps are obstructed by an transitional up-hill step. Hence, it works well for less number of variables only. With the increase in the number of inputs, the intricacies increase.

ii. **Sifting Algorithm** [18]**:** This algorithm locates the best or optimum position for a variable, assuming every other single variable stay static. If there are variables in the graph (excluding the constant level), at that point there are *i* potential positions for a variable, along with its present position. Among these *i* positions, the sub-goal hired by the sifting method is to discover the point that will limit the size of the graph *i.e.* DAG. The sift algorithm has an added feature

that a variable can traverse a large distance in the ordering sequence. The graph size can increment altogether after few variables exchanges initially, and afterward in the end decreases. This supports a kind of bottom up move to be considered. The acknowledgement of the whole sequence of pair-wise exchanges totally depends on the optimal position. Little importance to any increase in the intermediate graph size is shown.

iii. **Linear Sifting Algorithm** [20]: This algorithm joins the proficiency of shifting and the capability of linear transformations. Linear transformations change a variable, x_i, with a fixed combination of variables. A linear combination is obtained by having the exclusive OR of the inputs or its inversion. Sifting is a local search algorithm that alternatively enhances the order of input variables by a progression of exchanges of neighbouring inputs. Each input is taken turn by turn and is carried at the top and bottom in the order to hold all places progressively. The variable is then brought back to one place where the optimum BDD size was recommended. The procedure at that point proceeds with another input variable. The adequacy of shifting nodes from its position to move a variable to any other position in the variable order in a brief time span. Its time efficiency depends on the capacity to rapidly exchange adjoining inputs. To perform such kind of exchange by penetrating only the nodes labelled by the two inputs being traded is conceivable. Each variable is taken into consideration, and, same as in sifting, it is shifted at the top and bottom in the variable order. Let x_i, be the selected input variable, and let x_j, be the variable quickly tailing it in the order. An essential stage of linear sifting comprises of the following three segments:

   a. Variables x, and x_j are exchanged and let the size of the BDD after the exchange be *m1*
   b. The linear transformation $x_j \rightarrow x_i, \equiv x_j,$ is performed and let the resulting size of the BDD be *m2*
   c. If m1 <= m2 then the linear transformation is fixed. This is obtained by basically performing the transformation once more, since it is its own complement.

   The resulting effect of the three-segment system is that x_i is moved one position ahead in the order, and then linearly joined with x_j. Conversely, if x_j is the variable being moved, it is first of all exchanged and after that x_i is joined with

x_j. Linear sifting is easier than standard sifting; in light of the fact the cost of a linear transformation is practically identical to the cost of a variable exchange. If we expect that the tree structures controlled by the two algorithms are roughly similar size, at that point linear sifting comes out to be around three times slower than normal sifting.

4.4 Algorithms used for Variable Ordering in BDDs:

i. **Scatter Search Algorithm** [21]: Scatter search presents a sensible trade-off between quality (BDD size reduction) and runtime. A smaller benchmark circuit conveys relatively optimal BDD size with less runtime. A bigger benchmark circuit conveys reduction in BDD sizes at the expense of longer runtime. Scatter search approach was presented by Glover in 1977. Scatter search is extremely hostile and battles hard to discover quicker top notch solutions. The algorithm works on a arrangement of potential solutions and the reference set by joining these to make new solutions. The principle method for joining solutions is by linearly combining two different solutions and from that a new solution is made. Dissimilar to "populace" in genetic approach, in scatter search approach the reference set of candidate solutions has a tendency to be smaller. It picks at least two elements of the reference set efficiently with the motivation behind making new solutions. The genetic approach and simulated annealing additionally results in genuinely smaller BDD sizes. They are populace-based minimization which has some similitude with scatter search approach. The genetic approach can acquire nearly same BDD sizes as in scatter search method. In different cases, Simulated Annealing get BDD sizes similar to that of scatter search. The extensive computation time for scatter search is the exchange off amongst BDD size and time. For CPU runtime, a greater enhancement set really does not increase the search time excessively, but expanding the reference set size makes it tougher to come together, bringing about more extended execution time.

ii. **Genetic Algorithm Based Technique** [22]–[24]: This is a fantabulous multi-objective optimization technique. Genetic Algorithms (GAs) are gin stochastic minimization in light of rule of natural selection and genetics. They start with a first initial set of populace comprising of randomly created

solutions. The mutation and crossover are permitted to develop over various ages in view of a conceptive arrangement. The chromosomes are assessed in light of a fitness criterion after every generation. Depending upon the fitness value and selection policy, the pair of chromosomes for next generation is picked. At last, when there is almost no change in the solution over a fixed number of iterations, the algorithm comes to an end. The best solution is favoured as the solution at that generation framed by genetic algorithm. For any issue, the formulation of Genetic Algorithm includes the keen, thoughtful and dynamic encoding of the candidate solutions to frame chromosomes, mutation, crossover and a cost function estimating power and robustness of the chromosomes in a populace. Genetic Algorithm based technique for BDD optimization was first proposed by Drechsler [22] in 1996 [22]. After that various modified techniques based on this algorithm have been proposed and implemented [23], [25] in order to further reduce node count as well as computation time for BDD optimization.

iii. **Branch and Bound Algorithm** [24]**:** Branch and Bound (BB) based eager approach is a brilliant optimization approach for multi-input multi-output circuits. It has a substantial number of possible solutions. For a given circuit, branch and bound algorithm looks through the total space of solutions for optimum solution or ideal arrangements. This action is performed by an iteration procedure which consists of three principle segments: Solution set is selected for bound estimation and branching. Branching, with its repeated application, characterizes a tree like structure where nodes are attained from the past step by separation i.e. subdivision of the node space into at least two subspaces to be examined in a resulting generation. The following stage is called bounding and is a technique that processes just the lower limits of the solution set by figuring out the limits for every solutions, within the given arrangement of solution.. In view of the fitness, the lower limits are computed by setting the target function of the suggested question. If the lower limit for a tree node X is greater than the lower limit for another node Y in the similar step, at that point X might be securely disposed of from the investigation [24]. This step is known as pruning, and is materialized by holding up a global variable n (shared among all nodes of the tree) that notes the minimum lower limit among all

candidate solutions. Any node whose lower bound is greater than n can be disposed of. The bounding function for the subspace is figured and is compared with the presently accessible optimal solution.

iv. **Genetic Tabu Hybrid Strategy** [26]: This technique combines the global space hunt of genetic approach with the neighbourhood search and the continuous global optimization of Tabu search. This algorithm tries finding an optimum or ideal solution in the neighbourhood of the present solution while maintaining a strategic distance from the issue of sub-optimal solution trap. Its fundamental rule is to seek local search at whatever point it experiences a local optimum or ideal solution. The algorithm is partitioned into two segments. At first stage, a guess set of N attainable candidate solutions each representing a variable order developing a populace is picked arbitrarily in the search space. Each parent is permitted to create more than one child to shape a family. The children having a place within same family turn into the offspring's being chosen in the primary stage where Tabu Search *i.e.* Tabu Search is utilized to pick the optimal or an ideal offspring as parent for the next iteration. The second-stage determination performs competition among different families giving a measure of the fitness of each family and basic genetic tasks are implemented. Average fitness of each family is calculated and the number of children (NoC) distributed to every family for the next iteration is made proportional to their fitness values and is given as

$$NoC = (No\ x\ NoCo\ x\ F)\ /\ S$$

Where, N_o is the initial number of families,

NoC_o is the initial number of children produced in each family,

F is the fitness of a family,

S is the sum of all families' fitness.

The aggregate number of children in next iteration is kept steady. Subsequently fitter members show signs of improvement in survival chance and in the long run just a single ideal family survive. The normal estimation of objective functions have a place within every family and is assigned as

the fitness F of every family. The effectiveness of this algorithm is totally dependent on the neighbourhood structure and initial solution set Also, the convergence speed decreases due to swapping operations on neighbourhood solutions.

v. **Particle Swarm Optimization Technique** [27]: Particle Swarm Optimization (PSO) is a populace-based stochastic procedure propelled by the social conduct of bird flocking or fish tutoring. In particle swarm optimization, the potential solutions, called 'particles' fly through the issue by following the present ideal or best particles. The algorithm requires numerous particles framing a population of potential solutions that can investigate the whole scope of accessible positions, gain experience through their past, gain experience from other particles and lastly converge close to the potential solution, which might be the optimal or reasonably good after fulfilling distinct completion criterion. The particles feel their closeness to a decent solution utilizing a fitness function as the parameter. Every particle has a fitness value that is assessed by the fitness function *i.e.* the cost function is to be minimized and has a velocity that coordinates its flight. Amid every iteration, every particle is refreshed depending upon its initial position, the position of global best solution and its velocity. Huge diminishments in node count as well as runtime have been watched using this algorithm as compared to other evolutionary approaches, such as GA. The only limitation is the timing overhead as the number of variables increase.

4.5 Comparison among various Variable Ordering Techniques:

Table 4.1 Summary of Variable Ordering Algorithms

Algorithm	Static/ Dynamic	Basic Principle	Applicability	Complexity
Depth First Traversal	Static	Variables closer in circuit topology are kept close in order.	Smaller circuits only	Increases with the increase in variables.
Variable Interleaving	Static	For multi-output circuits, variables for different outputs are interleaved instead of appending	For large multi-output circuits. Can be extended for single output circuits	Increases with the increase in variables

Window Permutation	Dynamic	Variables are exchanged with the adjacent variables at all levels	Smaller circuits	Increases with the increase in inputs
Sifting Algorithm	Dynamic	Finding optimum position for a variable, keeping others fixed, and setting it at the position which reduces size of BDD	In small as well as large circuits. Efficient than Window Permutation	Linearly increase with the number of inputs
Linear Shifting	Dynamic	Combines shifting and linear transformation of variable	All circuits, but comparatively slower than Sifting Algorithm	Increases with number of inputs
Scatter Search	Evolutionary, Dynamic	Combines different variable orders from a given reference population to create improved order that reduces BDD size. Iterational process	Highly efficient for smaller circuits. Takes longer time as the circuit size increases	Increases with number of inputs
Branch and Bound	Evolutionary, Dynamic	Dividing solution space into subsets and finding lower bounds i.e. fitness value of each solution in the solution set The set with high bound value is discarded. Iterational process.	Efficient for all circuits. But provide number of solutions for a single problem	Linear increase with inputs
Genetic Algorithm based	Evolutionary, Dynamic	Imitates the Darwin theory of evolution. Selects solutions, find fitness and generate best result after applying genetic operators i.e. crossover, mutation. Iterational process	Works for all circuits	Linear increase with number of inputs
Genetic Tabu search based	Evolutionary, Dynamic	Combines worldwide space search of genetic algorithm with nearby search of Tabu search. Tries to locate optimum solution in the neighbourhood of the current solution Iterational Process	Works for all circuits	Depends on initial solution set and number of inputs
Particle Swarm Optimization	Evolutionary, Dynamic	Stochastic technique motivated by social conduct of bird flocking with solutions trying to follow the global best or optimum solution with	Works for all circuits	Time complexity increase with number of inputs

		every iteration by updating their current values		

4.6 Evolutionary Algorithms

Evolutionary Algorithms (EAs) are populace-based metaheuristic optimization algorithms. They are speculative search techniques that copy natural biological development or the social conduct of group. These algorithms have been produced to reach at near-optimum or ideal solutions for unpredictable and large optimization problems which can't be unravelled by slope-based mathematical programming [28]. Every EA has three main components:

i. **Population** of individuals representing candidate solutions for the basic underlying optimization problems.
ii. **Fitness function** or objective function that determines the environment within which the solutions live and which determines the strength of an individual as an optimum solution to the given problem.
iii. **Evolutionary operators** leading to the development of populaces of individual that are more qualified to their condition than the parents

Thus, Evolutionary Algorithms take a shot at populations of people rather than single solutions. Thusly, the search is performed in a parallel way resulting in faster convergence of these evolutionary algorithms. A description of some widely used evolutionary algorithms is given as under.

4.6.1 Genetic Algorithm

The Genetic Algorithm (GA) was concocted by John Holland [52] at the University of Michigan in 1975, and it has been shaped generally by David Goldberg [53] at the University of Illinois. The aboriginal genetic method and its numerous variations, collectively called genetic algorithms, are computational methods that copy the common procedure of development. The theories of evolution and natural selection were first proposal by Darwin to explain his observations of plants and animals in the natural worlds [54] Darwin observed that, as variations are brought into a populace with each new generation, the less-

fit people tend to cease to exist in the competition for food, and this survival of the fittest rule prompts improvements in the species. The data concept of natural selection was used to explain how species have been able to adapt to changing environments and how the species that are very similar in adapting may have evolved.

All data needed for production of appearance and behavioural highlights of a living species is present in its chromosomes. Reproduction in general includes two guardians. The chromosomes of the offspring are produced from bits of chromosomes received from the guardians. Along these lines, the offspring acquire a blend of qualities taken from the guardians. Genetic algorithms attempt to use a similar method of inheritance to solve various problems, such as those involving adaptive systems [52]. The goal of the genetic algorithm is then to locate an optimal or ideal solution to a problem. Since genetic algorithms are heuristic procedures, they are not ensured to locate the optimum solution, but rather encounter has demonstrated that they can discover great solutions for an extensive variety of problems.

Genetic algorithms work by developing a populace of individuals over various generations. A fitness value is relegated to every individual within the populace, where the fitness calculation relies on the application. For every iteration, people are chosen from the populace for reproduction, they are crossed to create new people or individuals, and the new people are transformed with some low mutation likelihood. The new people might totally replace the old people within the populace, with distinct generations evolved [53]. On the other hand, the new people may join with the old ones within the populace [54]. In this case, reduction in population is the aim in order to maintain a constant size, *e.g.*, by choosing the best individuals from the populace. Both of these approaches have been used for applications that have yielded good results. The choice of approach totally depends upon the application. Since selection is more biased profoundly towards fit people, the average fitness of the populace has a tendency to improve starting from one generation onto the next generation. The fitness of the best people is required to enhance over time, and the best individuals might be picked up as optimal solution after a few generations.

Genetic algorithms utilize two essential methods from evolution: Inheritance or the passing of features from one generation onto the next, and competition, or survival of the fittest that results in removing the bad features from people in the populace. The fundamental point of interest is:

i. They are flexible, versatile and gain from experience

ii. They have inborn analogy

iii. These are effective for complexities

iv. These are easy to correlate without much correspondence overhead

The two basic GA approaches are presented next, first the simple GA, also called the total replacement algorithm, and then the steady-state algorithm, which is characterized by overlapping populations.

Every genetic algorithm works on a populace or an accumulation of a few elective solutions for a given problem. Every individual in the populace is known as a string or chromosome, in normal framework. These individuals are regularly coded as binary strings, and the individual characters or symbols in the strings are referred to as genes. In every cycle or iteration, another age or generation is evolved from the current populace trying to acquire better solutions. The populace size decides the quantity of data saved by the genetic algorithm. The genetic algorithm populace is evolved over various ages or generations.

An assessment or evaluation function is utilized to decide the fitness of each individual. The fitness is the opposite for the most part known as the cost of optimisation problems. It is standardized to depict genetic algorithms in terms of fitness rather than cost. The evaluation function is normally user-defined or client characterized and problem-specific.

People are chosen from the populace for multiplication or reproduction, with the choice biased towards all the more fit people. Selection is one of the key operators or administrators which guarantee survival of the fittest.

Crossover is the primary operator utilized for multiplication. It joins segments of two people to make two new people, called children. These children acquire a blend of highlights of the parents. For every pair of parents, crossover is performed with a high likelihood *Pc*, which is called the crossover likelihood.

With probability *Pc*, crossover is not performed and the offspring pair is the same as the parent pair.

Mutation is an incremental change made to every individual from the population, with a very little probability or likelihood. Mutation empowers new features or highlights to be brought into a population. It is performed probabilistically such that the probability of a change in each gene is defined as the mutation probability.

Inversion is a genetic operator which does not change the solution represented by the chromosome, but rather changes the chromosome itself, or the binary representation of the solution. The inversion probability is denoted by P1.

The generation gap is the fraction of individuals in the population that are replaced from one generation to the next and is equal to 1 for the simple GA.

A Schema is a specific set of values assigned to a subset of the genes in a chromosome. It is a partial solution and represents a set of possible fully specified solutions.

For a given problem, various genes may be linked, and specific values may be required for groups of genes in order to obtain a good solution. These schemata represent the various features of the candidate solutions. GAs implicitly operates upon the various schemata in parallel that is the reason they are so fruitful in taking care of complex optimization problems. The genetic operators make a new generation of individuals by joining the composition of parents chosen from the present generation. Because of the speculative selection process, the fitter parents with some good schemata are probably going to create good number of children. At same time, the terrible or bad parents, containing some bad or awful schemata deliver lesser children. Therefore, in the next iteration, the count of examples with good schemata increments and the count of cases with bad schemata diminish. The fitness of whole populace hence gets progressed.

In a typical, binary-coded GA, where the chromosomes are bit strings, each string in the population is a case of 2^L schemata, where L is length of every individual string. The number of different strings or possible solutions to the problem is also 2^L, and the total number of different schemata contained in all these strings is 3^L,

since each gene in a schema may be 0,1, or don't care, x. Thus, the population represents countless number of schemata, even for relatively little population sizes. By evaluating another child, a rough estimate of the fitness of greater part of its schemata is made. The numbers of these schemata display in the populace is in this way balanced by their relative fitness values. This impact is known as the intrinsic or inherent parallelism. When more individuals are assessed, the different schemata in the populace mirror their fitness values more precisely. When a better schema is produced into the populace through one child, it is acquired by other offspring's in the succeeding iteration and hence its extent in the population increases. It begins forcing out the less fit schemata, and the normal fitness of the populace improves.

4.6.2 The Simple GA

The simple GA (referred as the total replacement algorithm) is illustrated in Figure 4.1

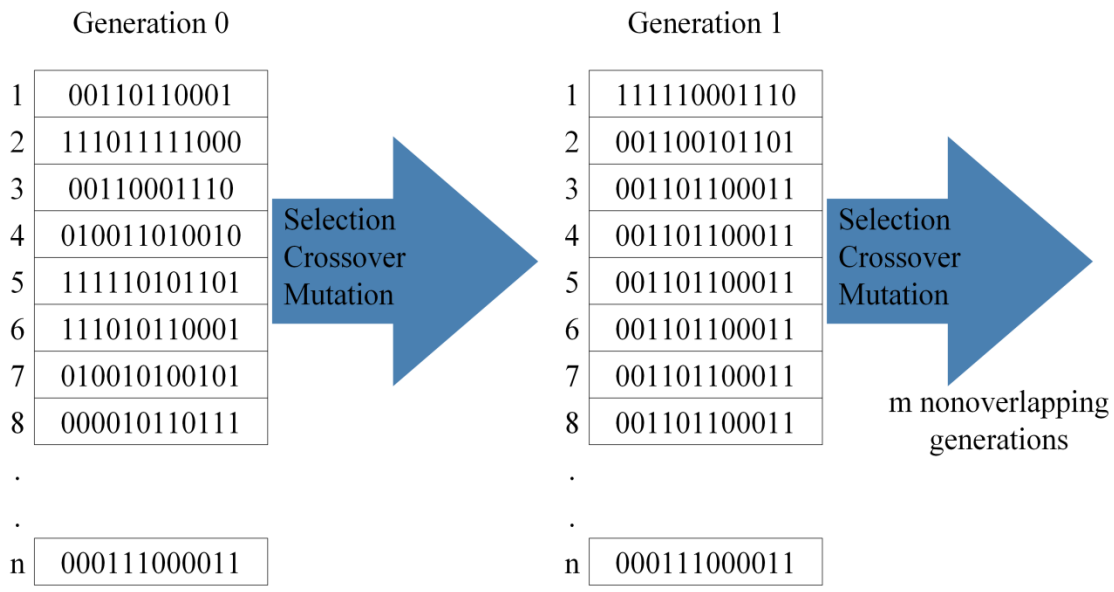


Fig.ure 4.1 The simple genetic algorithm

The flow chart of GA is shown in Figure 4.2

The simple GA is made out of populaces of strings or chromosomes and three evolutionary operators: selection, crossover, and mutation [53]. The chromosomes might be binary-coded, or they may contain characters from a bigger letters in order [55], [56]. Every chromosome is an encoding of a solution for current

problem and every individual has a related fitness which relies upon the application. The initial populace is generated haphazardly, yet it might be provided by the client or user. A very fit populace is developed through a few iterations by choosing two people, crossing the two people to create two new people and mutating traits in the new people with given mutation likelihood. Choice is done probabilistically however is one-sided toward all the more profoundly fit people, and the populace is basically kept up as an unordered set Recognizable generations are developed, and procedures of selection, crossover, and mutation are rehashed till the point all sections in another generation are fitted. At that point, the older generation might be disposed of. Presently generations are developed until the point when some stopping criterion is met The genetic algorithm might be constrained to a fixed number of generations, or it might be ended when all people in the populace merge to a similar string or no enhancements in fitness values are found after a given number of generations. Since choice is one-sided towards all the more very fit people, the fitness of the general populace is relied upon to increment in progressive generations. Thus, the best individual may show up in any generation.

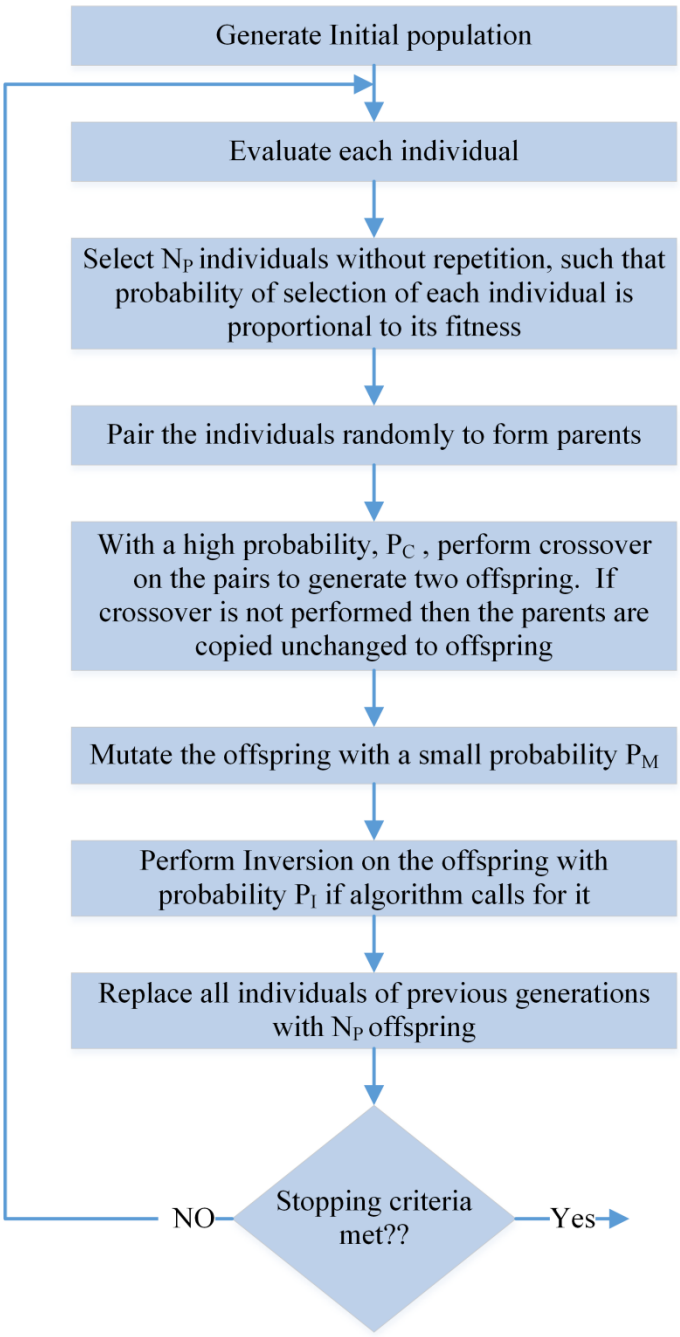


Figure 4.2 Flowchart of simple genetic algorithm

4.6.3 The steady-State algorithm.

In a genetic algorithm overlapping generations, just a small amount of the people are supplanted in each generation [57], [58]. The flow chart for steady-state algorithm is outlined in Figure 4.3.

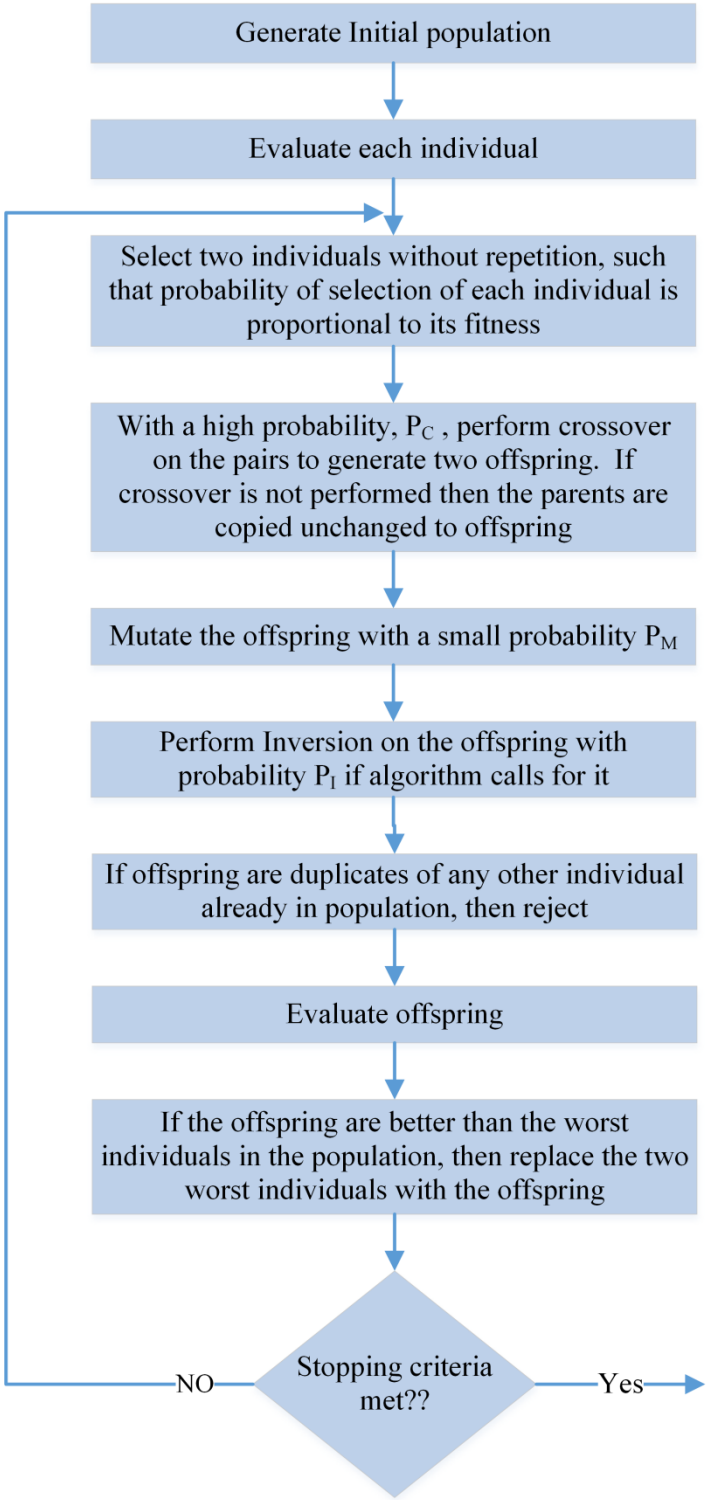


Figure 4.3 Flowchart of steady-state genetic algorithm

In each generation, two unique people are chosen as guardians in light of their fitness. Crossover is performed with a high likelihood, Pc, to form offspring. The offspring are mutated with a low likelihood, P_M and inverted with probability P_1, if necessary. A duplicate check may follow, in which the offspring are rejected

without any evaluation if they are duplicates, of some chromosomes already in the population. The offspring that survive the duplicate check are evaluated and are introduced into the populace just if they are superior to the present worst member of the population. In that case, the offspring replaces the worst member. This completes the generation. In the steady-state GA, the generation gap is minimal, since only two offspring are produced in each generation.

Duplicate checking may be beneficial because a finite population can hold more schemata if the population members are not duplicated. Since the offspring of two identical parents are identical to the parents, once a duplicate individual enters the population, it tends to produce more duplicates and individuals varying by only slight mutations. Premature convergence may then result. Duplicate checking is fruitful under the following conditions:

i. The population size is small.

ii. The chromosomes are short.

iii. The evaluation time is large.

Each of the above conditions reduces the duplicate checking time in comparison to the evaluation time. If the duplicate checking time is negligible compared to the evaluation time then duplicate checking improves the efficiency of the GA.

The steady-state GA is susceptible to stagnation. Since a large majority of offspring are inferior, the steady-state algorithm rejects them, and it keeps making more trials on the existing population for very long periods of time without any gain. Because the population size is small compared to the search space, this is equivalent to long period of localized search.

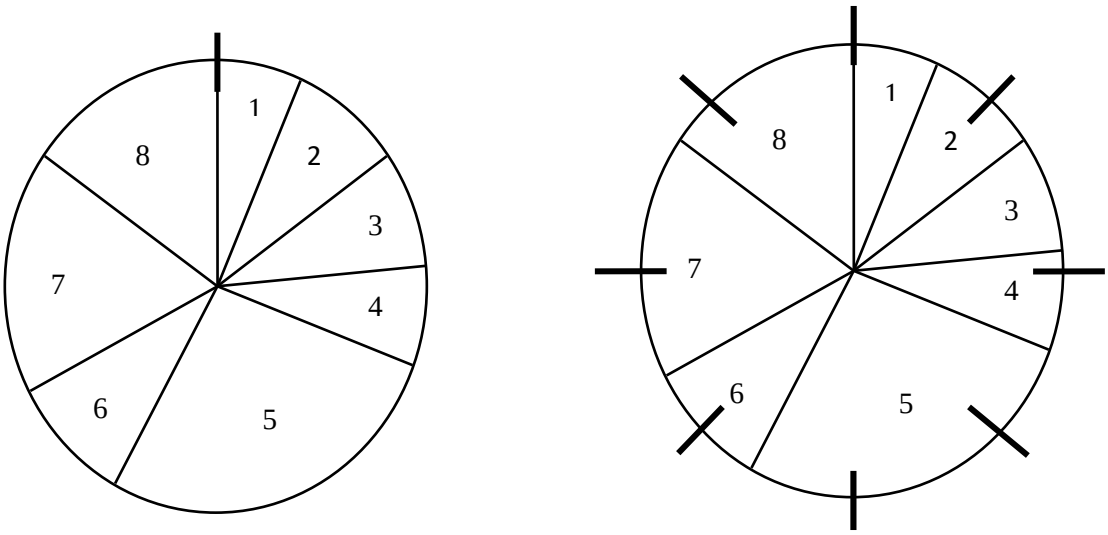
4.6.4 Genetic Operators

The following are the genetic operators: selection, crossover, mutation, fitness scaling and inversion.

4.6.4.1 Selection

Different choice plans have been utilized, yet focus is on roulette wheel selection, stochastic universal selection, and binary tournament selection with and without

substitution [59], [60]. Figure 4.4(a) illustrates roulette wheel selection a proportionate choice scheme in which the slots of a roulette wheel are measured according to the fitness of each people or individual in the populace. An individual is chosen by turning the roulette wheel and noting the position of the marker.



(a) Roulette Wheel (b) Stochastic Universal

Figure 4.4 Proportionate Selection Schemes

The probability or likelihood of choosing an individual is hence proportional to its fitness. As illustrated in Figure 4.4(b), stochastic universal selection is a less noisy form of roulette wheel selection in which *M* equidistant markers are put around the roulette wheel, where *M* is the number of people in the populace. M people or are chosen in a spin of the roulette wheel, and the number of duplicates of every individual or people chosen is equal to the number of markers within the corresponding slot. In binary tournament selection, two people are selected indiscriminately and the better one is selected from those two individuals. If binary tournament selection is conducted without substitution, then at that point the two people are put aside for the next selection procedure and they might not be substituted into the populace. Since two individuals are pulled back or withdrawn from the populace, then for every individual chosen the populace size stays consistent starting from one iteration onto the next iteration. The original populace is re-established after the new populace is half-filled. Subsequently, the best member will be selected twice and the most exceedingly bad member won’t be

picked by any means. The number or quantity of duplicates picked out of some other individual can't be anticipated aside from that it is 0, 1, or 2. In binary tournament selection, the two people are promptly substituted into the populace for the following selection procedure. The goal of the genetic algorithm is to cover an optimal or ideal solution and selection process is the main thrust which finds the rate of convergence. A selection pressure is high then the populace will converge rapidly, possibly at the expense of an imperfect outcomes. Roulette wheel selection in a typical fashion which gives the highest selection pressure in the initial generations, particularly when a couple of people or individuals have fundamentally higher fitness values than various other people. Competition gives more pressure or weight in later iterations when fitness values of people or are not fundamentally unique. Thus, roulette wheel selection is more likely to converge to a suboptimal result if individuals have large variations in fitness values.

4.6.4.2 Crossover

A modified alternating crossover mechanism is adopted to generate the next generation as shown in Figure 4.5

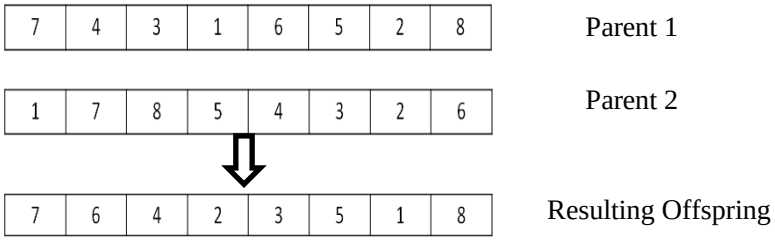


Figure 4.5 Modified Alternating Crossover Mechanism

This crossover mechanism enhances the performance of the algorithm as compared to conventional crossover schemes. A new solution is generated from two randomly selected parents in such a way that memes are copied alternately from these parents. This technique is a hybrid version of the alternating crossover mechanism as presented in [32], the difference being that memes are copied from opposite ends of both the parents without allowing the constraints violation.

Theoretical Review of Cross-over Operators

Two fundamental strategies that a strong crossover approach should use to find the ideal or optimum are investigation and exploitation. Investigation ability of a

crossover method supports the scan or search for a quick and exhaustive revelation of the design space. A powerful investigation keeps the hunt from finding a nearby pinnacle. However, its insignificant utilization ends up wasteful in making utilization of the already obtained points to achieve better points. Then again, a strong misuse utilizes the beforehand discovered points to achieve the ideal or optimum. Hence, for this situation a moderate convergence is watched joined by an increasing danger of finding a nearby pinnacle. In fact, these two strategies are conflicting, and an adjusted utilization of them is crucial for accomplishing a proficient search through crossover [61].

Types of Cross-over Operators

Crossover gives a trade-off of design characteristics and attributes between combined people or paired individuals. A few crossover operators have been contrived to achieve this assignment and amongst these are order cross-over, cycle cross-over and partially mapped cross-over operators.

Memetic Algorithm can quickly recognize discrete zones inside an expanded search space area to concentrate the scan or search for an ideal arrangement or optimum solution. This procedure changes commonly or mutually defined parts of two individual members selected and acquires diverse and distinct individual members that give new points in the search space [62]. The crossover operators utilized in the present study are summarized below.

The crossover point is randomly selected (represented by dashed line) in Figure 4.6 between 1 and (L−1) where L is the length of the chromosome. Two new individuals were acquired by moving parts, after this the cut-off point is coordinated in two individuals.

1. Order Crossover Operator

Order crossover operator finds a sub-arrangement from the principal parent with no progressions and afterward to resolve problems, compresses and packs the second parent by wiping out the strings passed on by the primary parent and after that moving the rest of the strings to the one side without altering their order. It then duplicates the compacted second part into the rest of the offspring cluster. Firstly the crossover point is randomly selected and then the offspring is obtained by relocating parts of the parents as shown in the Figure 4.6

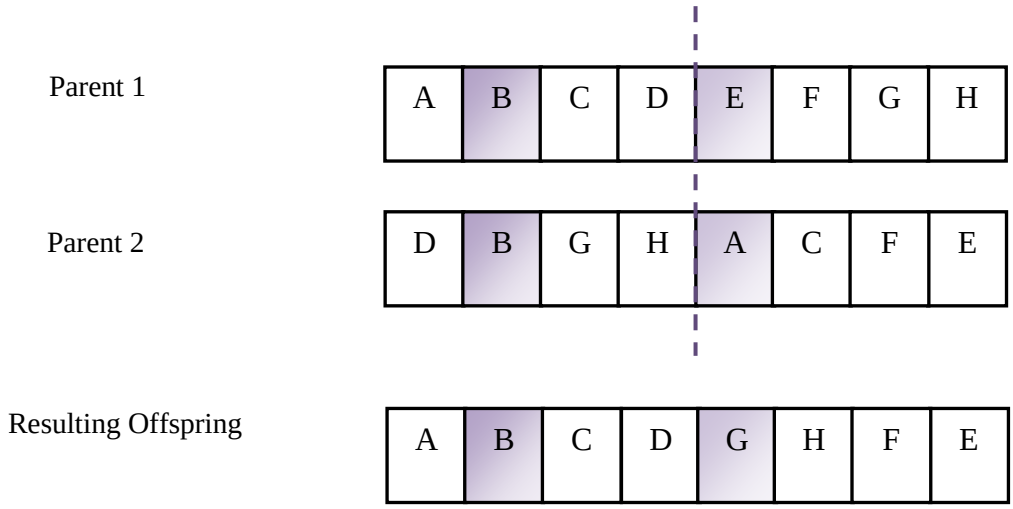


Figure 4.6 Order Crossover Mechanism

2. Cycle Crossover Operator

The offspring produced by the cycle crossover, each string is in the similar location as in one parent or the other parent. The basic thought behind the cycle crossover is to stay away from the string conflicts by discovering non-overlapping sets of strings passed on from the parents. The offspring is generated by relocating the elements from parent1 in the offspring taking care of the fact that the element present at the same position in parent2 will be prioritize to get relocate in the offspring. As shown in the Figure 4.7, 'A' gets relocated in the offspring at the first position then instead of relocating 'B', 'D' is prioritized and is relocated at the fourth position in the offspring and similarly other elements are relocated in a cycle.

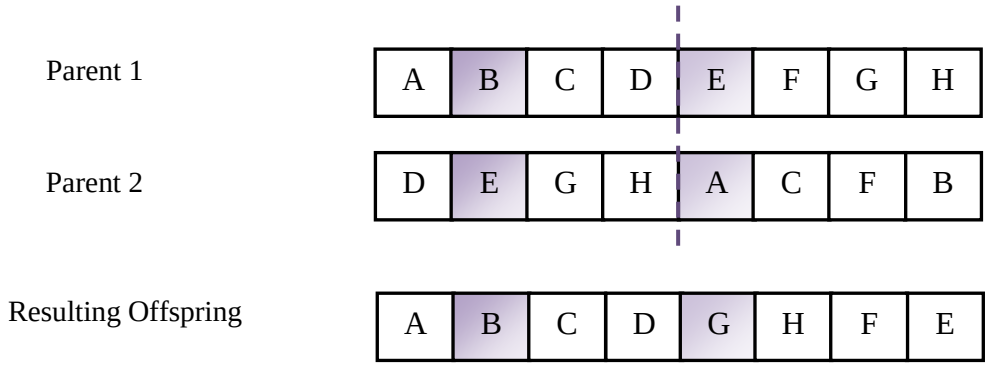


Figure 4.7 Cycle Crossover Mechanism

3. Partially Mapped Crossover Operator

In Partially mapped cross-over mechanism (also known as PMX crossover mechanism), with the randomly selected crossover point, the offspring is obtained by partially mapping the elements of the parents as shown in the Figure 4.8. It proceeds with position wise exchange. A string in the segment of the main parent

and a string in the similar location of the second parent characterize which strings in the principal parent must be traded to produce the child.

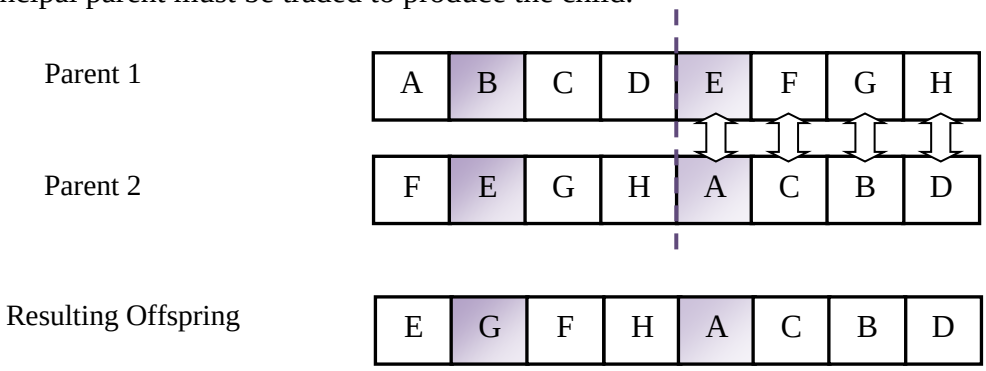


Figure 4.8 PMX Crossover Mechanism

Crossover consolidates the schemata or building blocks from two unique solutions in different combinations. Smaller good schemata or building blocks are changed over into continuously bigger good building blocks after some time until the point an entire decent solution is achieved. Crossover is a irregular procedure, and a similar process brings about blend of terrible or bad building blocks to result in poor offspring's. Yet, these offspring's are dismissed by the selection operator in the next generation of people.

The quantity of crossover is monitored by the crossover likelihood or probability that is characterized as the ratio of the number of offspring's produced in every iteration to the populace size. Higher crossover probability permits investigation of a greater amount of the solution space and diminishes the chances of agreeing to a false optimum. Lower crossover likelihood or probability enables abuse of existing people or individuals in the populace that have moderately higher fitness.

4.6.4.3 Mutation

As new people or individuals are produced, each character is changed or mutated with a given probability. In a binary-coded genetic algorithm, mutation might be performed by flipping a bit, while in a non binary-coded genetic algorithm; mutation includes arbitrarily creating another character in a pre-defined position. Mutation produces incremental arbitrary changes in the offspring generated through crossover, as shown in Figure 4.9. When used by itself, without any crossover, mutation is proportionate to irregular search, comprising of incremental arbitrary adjustments in the current solution or an acknowledgement

if there is change. However, when used in the GA, its behaviour changes radically. In the genetic algorithm, mutation serves the critical part of replacing the gene qualities lost from the populace amid the selection procedure with the goal that they can be attempted in another specific situation, or of providing the gene qualities that were absent in the initial populace.

Before Mutation:	110100010011
After Mutation:	110000010011

Figure 4.9 Mutation operator

For example, say a particular bit position, bit-10, has the same value, say 0, for all people or individuals in the populace. In such a situation, crossover alone will not help, since it is just an inheritance mechanism for existing gene values. That is, crossover cannot create an individual with a value of 1 for bit 10, since it is 0 in all parents. If a value of 0 for bit 10 turns out to be suboptimal, then, without the mutation operator, the algorithm will have no chance of finding the best solution. The mutation operator, by producing random changes, provides a small probability that a 1 will be reintroduced in bit 10 of some chromosome. If this results in an improvement in fitness, then the selection algorithm will multiply this chromosome, and the crossover operator will distribute the 1 to other offspring. Thus, mutation influences the whole reachable search space, despite of a limited population size. Although the crossover operator is the most efficient search mechanism, by itself, it does not ensure the reachability of the whole hunt space with a limited population size. Mutation operator covers this gap.

The mutation probability P_M is defined as the probability of mutating each gene. It controls the rate at which new gene qualities are brought into the population. If by chance the mutation probability is too low, numerous gene qualities that would have been helpful are never tried and tested. If it is too high, an excess of irregular disturbance will occur, and the offspring will lose their likeliness to the guardians or parents. The capacity of the algorithm to gain from the historical backdrop of the search will therefore be lost.

4.6.4.4 Fitness Scaling

If a proportionate selection scheme is used, the genetic algorithm chose people or individual with probability in proportion to their fitness. If the raw fitness values are used for this probabilistic selection, without any scaling or normalization, then one of two things can happen. If the fitness values are too far apart, then it will select several copies of the good individuals, and many other worse or bad people or individuals will not be selected in any manner. This has the tendency to fill the whole populace with similar type of chromosomes and restricts the ability of the genetic algorithm to investigate bigger and larger search space. Then again, if the fitness values are excessively near each other, then the genetic algorithm will tend to select one copy of each individual, with only random variations in selection. Consequently, it will not be guided by small fitness variations and will be reduced to random search. Fitness scaling is utilized to scale the crude fitness values with the goal that the genetic algorithm finds a sensible amount of difference in the scaled fitness values of the best versus the worst people or individuals. Accordingly, fitness scaling controls the selection pressure of discriminating power of the genetic algorithm.

4.6.4.5 Inversion

The inversion or reversal operator randomly picks up a random in a string of solutions and complements it from one end to other end (Refer Figure 4.10). This task is performed in a way with the end goal that the solutions which are represented by the string are not modified or altered.. Rather, adjustments in the representation of the solution are made. In this manner, the patterns comprising the solution string have an interpretation independent of their positions. This must be accomplished by adding an identification number with every pattern within the string and deciphering solution string with these identification numbers instead of the cluster indices. When a pattern moves in the cluster, its identification number moves along with it and subsequently, the understanding of the pattern stays unaltered. For example Figure 4.10 shows a chromosome. Let us assume a very simple evaluation function such that the fitness is the binary number consisting of all bits of the chromosome, with bit 0 being the least significant, and bit-9 being the most significant. Since the bit identification numbers are moved with the bit

values during the inversion operator, bit 0, bit 1, *etc.*, still have the same values, although their sequence in the chromosome is different. Hence, the fitness value remains the same. The inversion probability is the probability of performing inversion or reversal on every person individual during every generation. It controls the measure of group formation.

Before	B_9	B_8	B_7	B_6	B_5	B_4	B_3	B_2	B_1	B_0
Inversion	1	1	0	0	0	0	1	1	0	0
After	B_9	B_8	B_7	B_6	B_5	B_4	B_3	B_2	B_1	B_0
Inversion	1	1	0	1	1	0	0	0	0	0

Figure 4.10 Inversion operator

Inversion changes the sequence of genes randomly, in the hope of discovering a sequence of linked genes placed close to each other.

4.7 Simple Genetic Algorithm (SGA)

An evolutionary process is inspired a family of computational models named as Genetic Algorithms, GAs [29]. These algorithms encode a potential solution to a particular problem on a simple and straightforward chromosome-like data structure and apply recombination operators to these structures keeping in mind the end goal to protect and preserve crucial stats. Starting with an arbitrary populace of solutions, these algorithms find the fitness of every chromosome against a problem specific objective function to determine the best and worst solutions. Based on natural selection and populace genetics qualities, genetic algorithm is a searching method that has wide practicability.

To emulate the Darwin theory of survival of the fittest, best chromosomes of the parent generation exchange information through genetic operators such as crossover, inversion or mutation to create their offspring chromosomes. These offspring chromosomes are evaluated and permitted to reproduce if they are better than the previous generation. Hence, worst results are eliminated and best results are allowed to evolve with iterations. This process is preceded for an extensive number of iterations until a near-optimal or best-fit solution is observed.

Principle parameters affecting the performance of genetic algorithms are populace size, number of generations (iterations), crossover rate and mutation rate. Bigger the populace size (number of chromosomes) and substantial number of iterations

increase the probability of acquiring a global best solution, but this in turn results in increased runtime. Crossover between two parents to produce an offspring is a natural evolution process; hence its rate normally ranges between 0.6 and 1.0. However, mutation being a uncommon procedure that resembles a sudden change in the offspring, its rate is kept low. The operation flow of genetic algorithm is shown in Figure 4.11

Genetic Algorithm

Plan and formulate initial populace

Initialize the populace in a random manner

Repeat

Assess the target function

Locate the fitness function

Apply genetic operators

Reproduction

Crossover

Mutation

Until criteria cease to exist

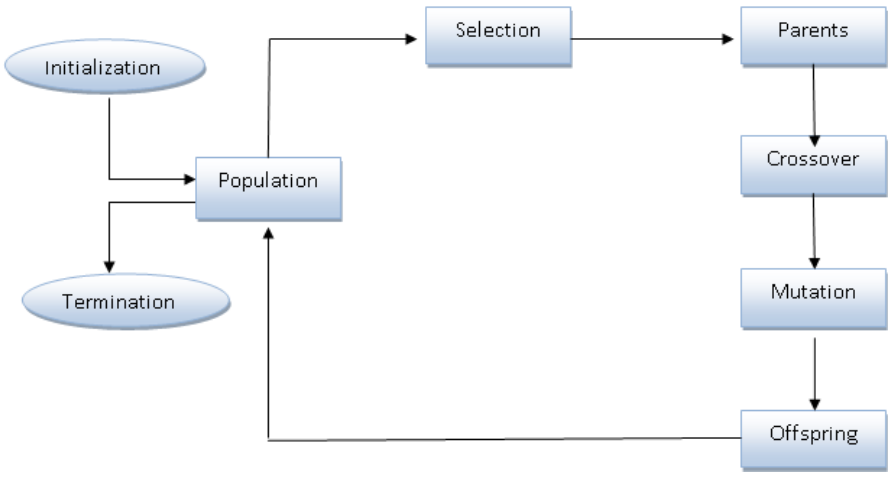


Figure 4.11 Operation Flow of Genetic Algorithm

4.8 Hybrid Genetic Algorithm (HGA)

Hybrid Genetic algorithm is an improved version of the basic Genetic Algorithm. In a basic genetic algorithm, the important parameters such as remaining populace

size, crossover rate and mutation rate are set, while in this hybrid genetic algorithm the parameters influencing are allowed to change dynamically depending upon the fitness of the next generation of population [23]. Also, the basic genetic crossover operation is modified so that invalid solutions are avoided [23]. In the talked algorithm, mutation rate, remaining populace size and remaining number of iterations are assigned values depending upon the improvement found in the individuals. A complete flow chart of HGA is illustrated in Figure 4.12.

The performance of genetic algorithms is being affected by the following parameters such as populace size, number of generations (iterations), crossover rate and mutation rate along with constants to decrease or increase remaining generations, populace size and mutation rate depending upon the quality of next generation.

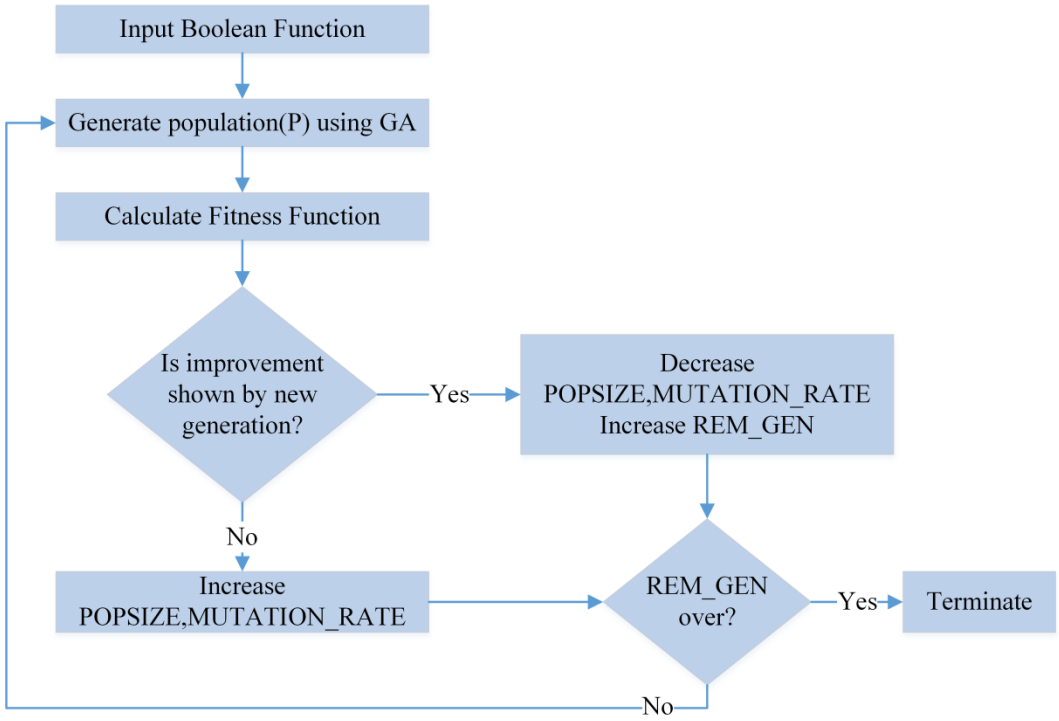


Figure 4.12 Flow Chart for Hybrid Genetic Algorithm

4.9 Particle Swarm Optimization Algorithm (PSO)

Particle swarm optimization algorithm (PSO) was first proposed by Dr. Kennedy and Eberhart in 1995 [30]. It is another intelligent optimization algorithm created lately, which mimics the relocation and collection of bird flock when they look for

food for sustenance [30]. PSO is a computational technique that enhances a problem by iteratively attempting to improve a candidate solution for a given problem in regard to a given measure of quality or fitness. PSO improves a problem by having a populace of candidate solutions, known as particles, and moving these particles around in the search-space as indicated by simpler scientific formulae over the particle's present position and speed without utilizing any complicated evolution operation. Each particle's development is affected by its best known position in neighbourhood and is likewise guided towards the best known position in the search-space globally, which is refreshed every time better positions are found by different particles. Thus, throughout the process, three important values are monitored for each particle: particle's current position *cBest*, its best known position in previous iterations *pBest* and its current velocity *V*.

With iterations, position of the global best particle *gBest* is computed and is taken as the best fitness. Accordingly, remaining particles update their velocities to find the best particle. Based on the current velocity, previous best position and global best position, each particle's position is updated according to the following equations:

$$\text{New } V = \omega \text{ X current } V + \text{c1 X } rand(\,) \text{ X } (pBest - cBest) + \text{c2 X } Rand(\,) \text{ X } (gBest - cBest)$$

$$\text{New position } cBest = \text{current position } cBest + \text{New } V;$$

$$V_{max} \geq V \geq -V_{min,}$$

where, ω is an inertia weight employed for improvement to control the impact of previous velocities on current velocity, c_1 and c_2 are two positive constants called learning constants, *rand( )* and *Rand( )* are two random functions in the range [0,1], V_{max} is the upper velocity limit and V_{min} is the lower velocity limit. [28] This eventually moves the swarm towards the best solution. Figure 4.13 explains the working of this algorithm through a flow chart.

The main parameters affecting the performance of PSO algorithm are population size *i.e.* number of particles, number of iteration cycles (generations), the maximum change in velocity of a particle and inertia constant ω [28].

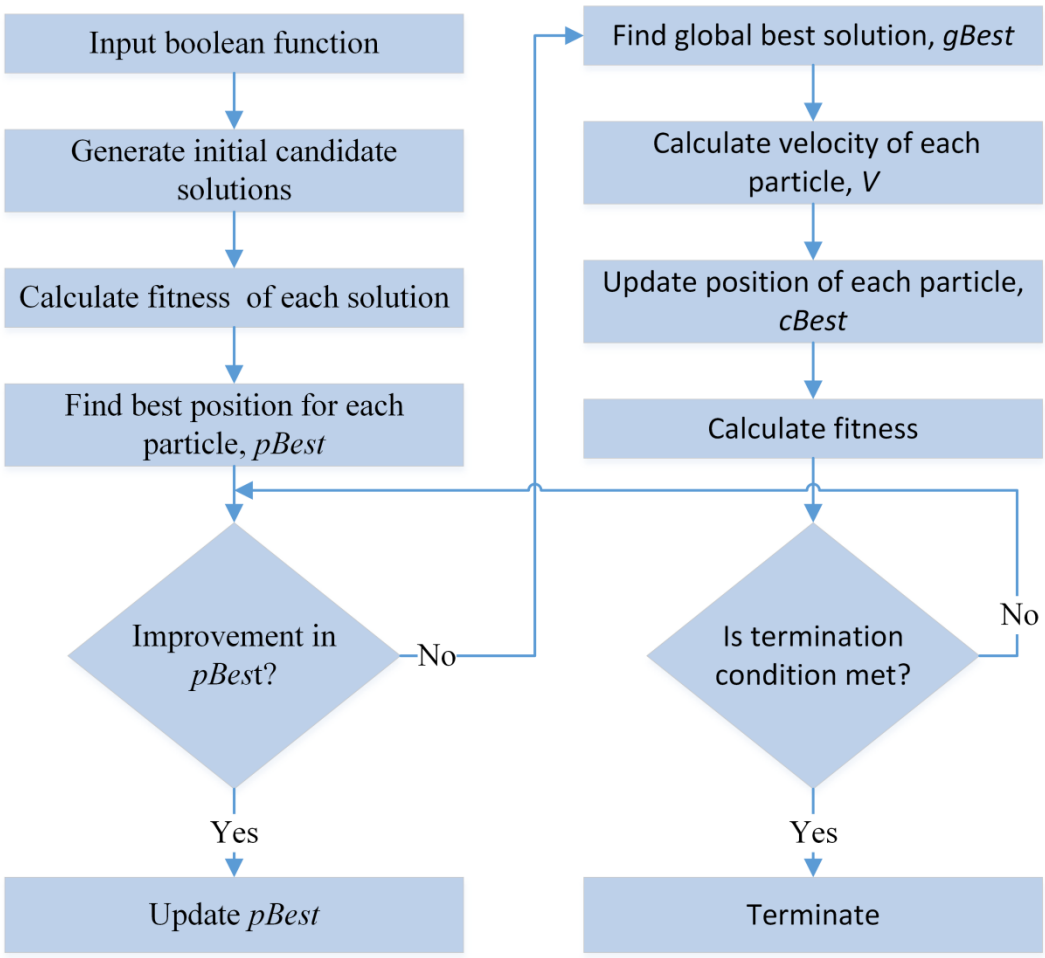


Figure 4.13 Flow Chart for Particle Swarm Optimization algorithm

4.10 Memetic Algorithm (MA)

Memetic Algorithms are inspired by Dawkins' notion of a meme [61]. These are based on "Universal Darwinism", a theory coined by Richard Dawkins in 1983 [63]. These are population-based optimization algorithms in which the candidate solutions are subjected to procedures of competition among them and mutual cooperation in a way that matches the behavioural patterns of living beings or creatures from similar species. Memetic Algorithms are very much similar to Genetic approach, however, the components shaping into a chromosome are called memes, but not genes. Meme is the basic unit of cultural transmission or imitation passed on by non-genetic operators.

The remarkable part of Memetic Algorithm is that all chromosomes in a generation are permitted to increase their understanding and get some experience, through a nearby search (LS) [31], before being associated within the evolutionary process. On a randomly created initial population, each and every chromosome is

searched locally with a specific end goal to enhance its experience and hence acquire a populace of optimum solutions locally. The population at that point undergoes basic genetic operations to produce new generation, which is again subjected to local search. Various approaches have been proposed [29] to perform search locally in neighbourhood that include set-wise swap operations among the memes of a chromosome as shown in Figure 4.14 or adding a constant number to each meme. The performance of each chromosome is evaluated again after performing LS. The change is held if the performance enhances else it is reversed. Principle parameters impacting the performance of Memetic Algorithm are populace size, number of iterations, crossover rate and mutation rate along with local-search mechanism. Figure 4.15 provides a description of this algorithm in the form of flow chart.

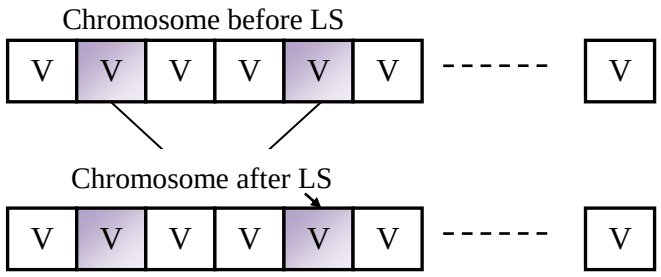


Figure 4.14 Local Search LS using pair-wise swapping chromosome after local search

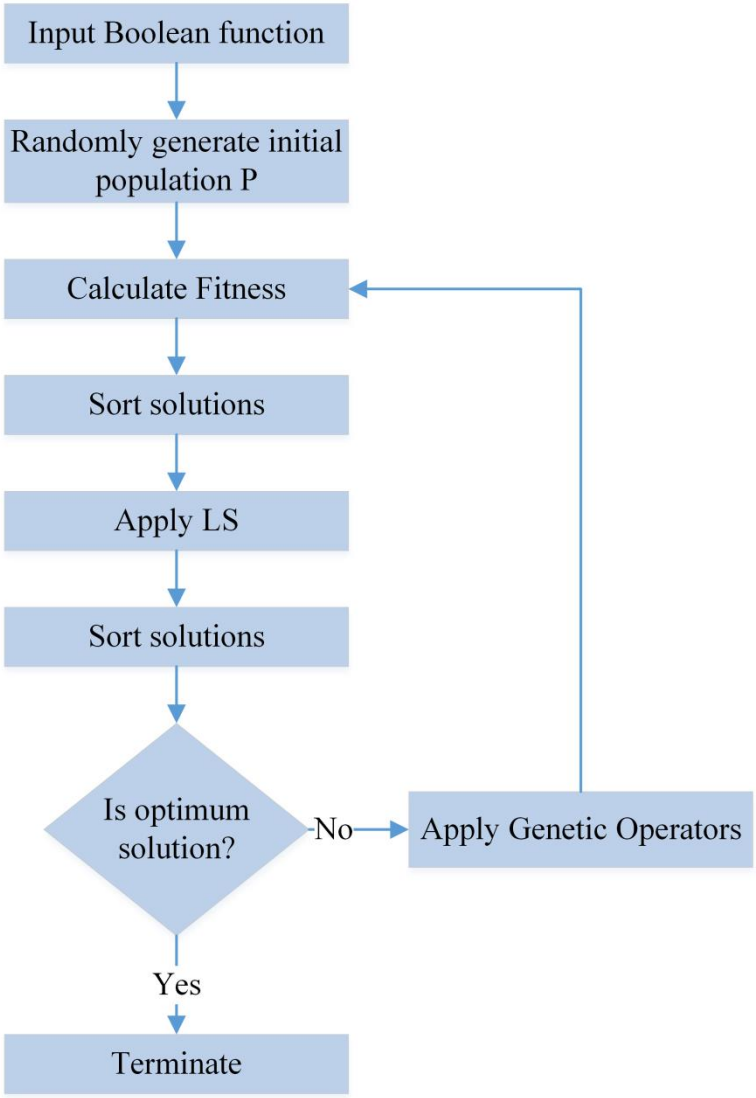


Figure 4.15 Flow Chart for Basic Memetic Algorithm

CHAPTER 5

ALGORITHMS APPLIED FOR BDD OPTIMIZATION

In this chapter Genetic Algorithm, Hybridized Genetic Algorithm and Modified Memetic Algorithm based approach that has been used to identify a good variable ordering in a shared BDD (SDD) is presented. This chapter explains the functioning of basic mechanisms that are used to implement these algorithms. The algorithm incorporates global search exploration feature of GA and local search exploitation of MA with various hybrid operations in a way that provides impulse to the search operation and at the same time prevents the solution to land in local optimum situation.

5.1 Genetic Algorithm (GA) Figure 5.1 shows the flowchart for BDD minimization using genetic algorithm.

5.2 Hybridized Genetic Algorithm (HGA)

An Evolutionary algorithm can be hybridized with other evolutionary or dynamic algorithms to find optimized results. In Hybridized Genetic Algorithm [64], genetic Algorithm is embedded with Branch and Bound Algorithm to intelligently seek for the best candidate solution of an optimization problem. The objective of Hybridized Genetic Algorithm is to locate an optimal or best variable ordering of BDDs with reduced node count and computation time. It works by finding a best individual *i.e.* BDD ordering from a population of individuals evolved over many generations. A fitness value *i.e.* in terms of node count is assigned to each individual *i.e.* BDD order. A population of random individuals is initialized. Parents are selected based on their fitness values. Branch and Bound Algorithm discards fruitless solutions by using upper and lower bounds on fitness value. Crossover and mutation operations are executed on them to produce offspring with a better fitness value i.e. node count. The process is continued till an optimum solution *i.e.* BDD with least node count is found. Figure 5.2 shows the flowchart for BDD minimization using hybridized genetic algorithm.

Flowchart for BDD Minimization using Genetic Algorithm

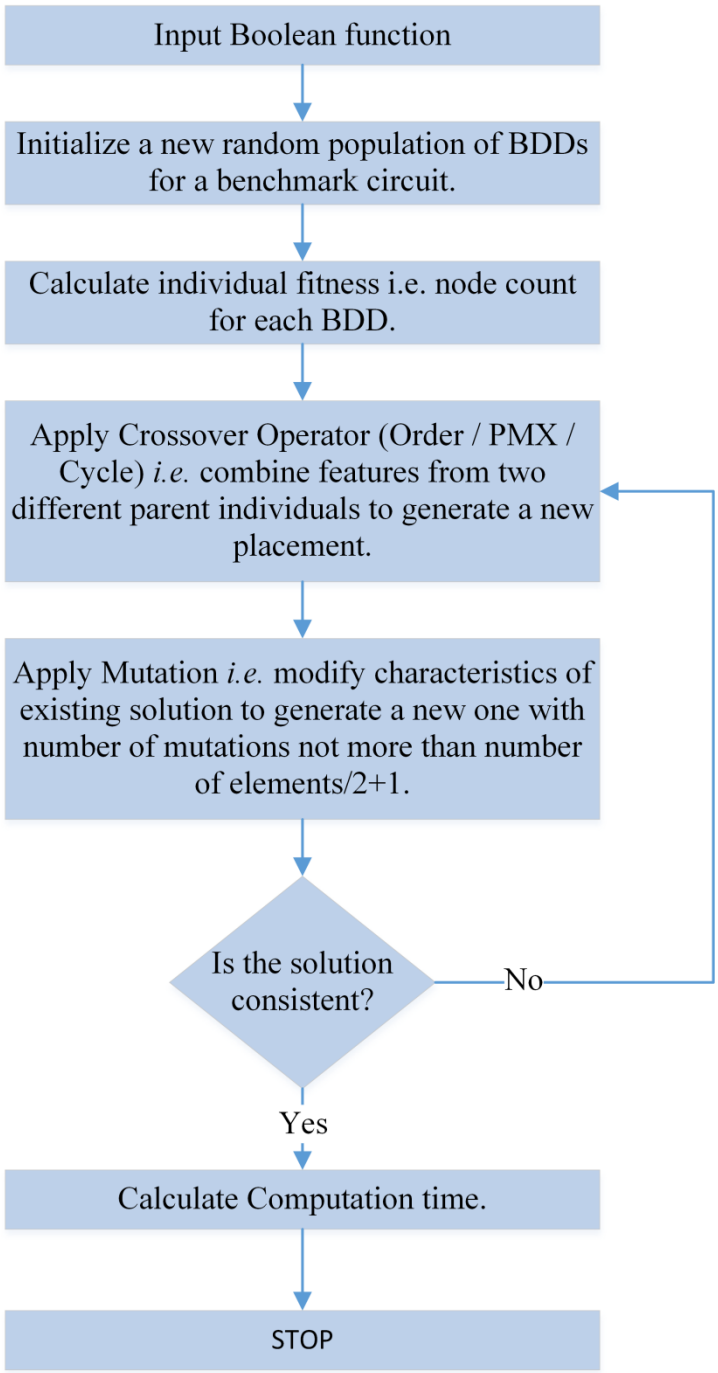


Figure 5.1 Flow-chart of the Genetic Algorithm based technique used for BDD Variable Ordering [38]

Flowchart for BDD Minimization Using Hybridized Genetic Algorithm

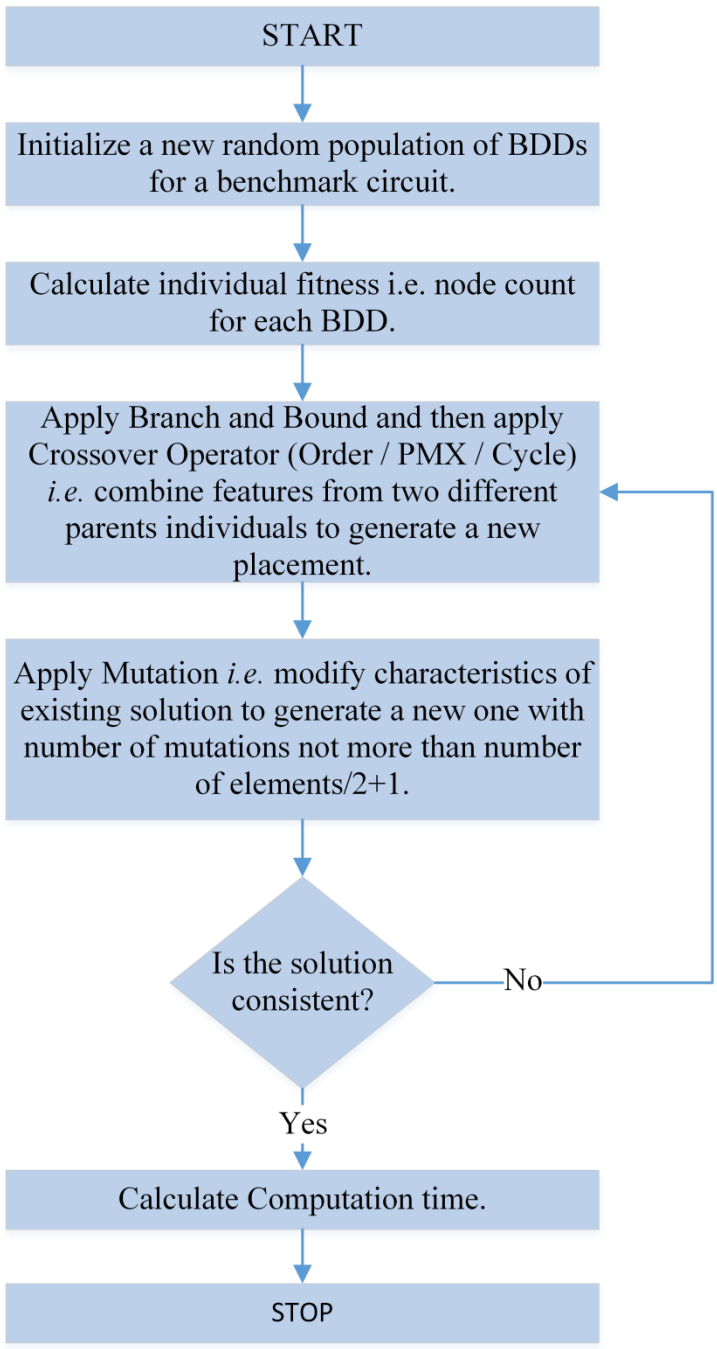


Figure 5.2 Flow-chart of the Hybridized Genetic Algorithm based technique used for BDD Variable Ordering [57]

5.3 Solution or Meme Representation

The operation of Modified Memetic Algorithm begins with a populace (P) of randomly generated chromosomes representing potential candidate solutions to the variable ordering problem in BDDs. A chromosome, representing a Boolean function of *n* variables, itself is a set of *n* memes (*i*1, *i*2, *i*3, *i*4, *i*5,…… *i*n), each

represented by an integer number between 1 to *n* without repetition. For example, let us consider a digital circuit combinational in nature consisting of 7 input variables, accordingly, the chromosome can be represented by any of these forms:

3	4	6	1	7	5	2

011	100	110	001	111	101	010

Figure 5.3 Solution or Meme Encoding in Chromosome Form

This chromosome finds a unique ordering of the input variables to be utilized in building up the shared BDD for the multi-output function. The input variables are ordered according to the numbers appearing in the chromosome and the corresponding SDD is generated. Figure 5.3 shows Solution or Meme encoding in chromosome form

5.4 Fitness Function Calculation

Fitness function is calculated to find the quality of every potential solution. Fitness of each individual has to be evaluated in terms of the node count in the corresponding SDD that represents fitness function for the given problem, as determined by the standard BDD package *Buddy-2.4.*

$$Fitness\ Function, F\ =\ Number\ of\ nodes\ in\ SDD$$

5.5 Local Search Operation for Meme Improvement

All solutions are sorted in ascending order and the best and the worst *S* solutions among them undergo local-search (LS) mechanism [65]. The LS mechanism used for optimizing the best *S* solutions performs pair-wise swap of memes in a chromosome, as depicted in Figure 5.4 , until either a better local optimum result is obtained or for maximum

$$M = n\,X\,(n\text{-}1)\,/\,2$$

where, n is the number of input variables.

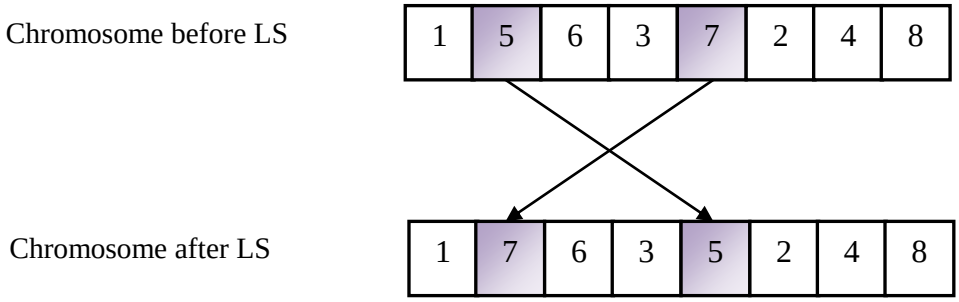


Figure 5.4 Local Search mechanism for improving the performance of best s solutions

The LS mechanism used for optimizing the worst *S* solutions is different. These solutions are made to mimic the global best solution. Any worst solution *w* is obtained by randomly selecting a subsequence from the best solution to replace the corresponding position in worst *wth* solution, while keeping the other positions unchanged. However, if the constraint violation occurs, then the memes are randomly relocated to form a new feasible solution as illustrated in Figure 5.5 The new solution is retained if the fitness improves, else it is reversed.

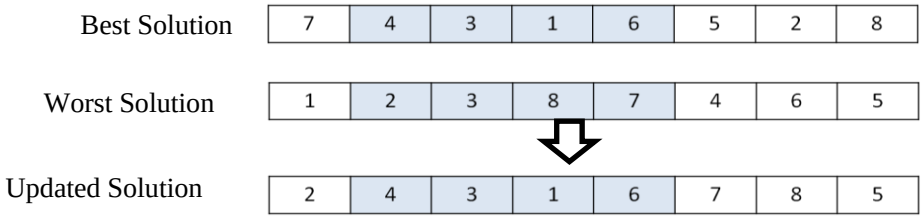


Figure 5.5 Local Search LS mechanism for improving the performance of worst *s* solutions

5.6 Genetic Operators

The new local optimum solutions obtained after LS operation are again sorted and allowed to evolve by crossover and mutation operations. This process continues until an optimum variable order is obtained. The evolutionary technique incorporated in this algorithm is hybrid genetic algorithm as proposed in [66], [67] in which important parameters such as size of populace, crossover rate, number of generations, mutation rate are not fixed, rather, these parameters are made dynamic and change depending upon the improvement found in every generation.

5.6.1 Crossover Technique for generating new population

A modified alternating crossover mechanism is adopted to generate next generation as shown in Figure 5.6.

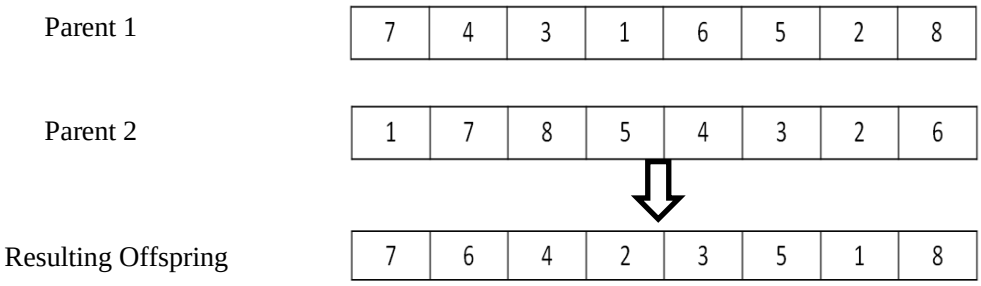


Figure 5.6 Modified Alternating Crossover Mechanism

This crossover mechanism enhances the performance of the algorithm as compared to conventional crossover schemes. A new solution is generated from two randomly selected parents in such a way that memes are copied alternately from these parents. This technique is a hybrid version of the alternating crossover mechanism as presented in [32], the difference being that memes are copied from opposite ends of both the parents without allowing the constraints violation.

5.6.2 Mutation Operation

This operation is used to introduce variety in solution. In other words, mutation is defined as a process of randomly disturbing genetic information. This is done to stop all potential solutions in a populace to fall into a local optimum of solved problems. Mutation operation depends upon the type of encoding scheme used to represent the chromosomes. For example, for binary encoding several randomly selected bits are switched from 1 to 0 or from 0 to 1. Since integers are used to represent memes in this algorithm, the mutation operation incorporated is similar in a way that memes are complimented and replaced with their corresponding integer compliments as shown in Figure 5.7 Other memes are modified so as not to violate constraints.

$$\text{Compliment of integer } n \;=\; \text{ELEMENTS} - n \;+ 1$$

where, ELEMENTS represent the total number of input variables.

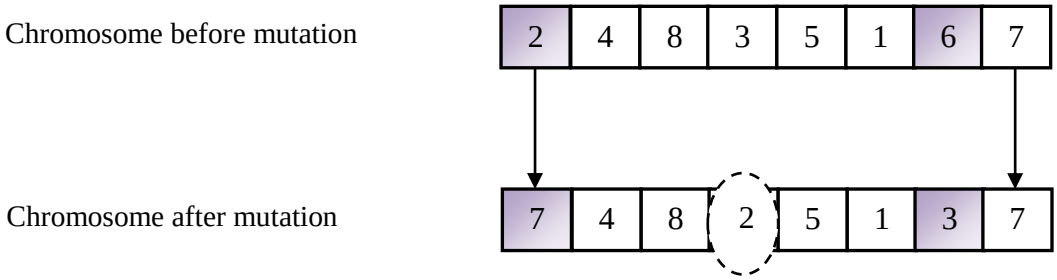


Figure 5.7 Mutation Operation incorporated in MMA

5.7 Parameter Settings and Flow Chart

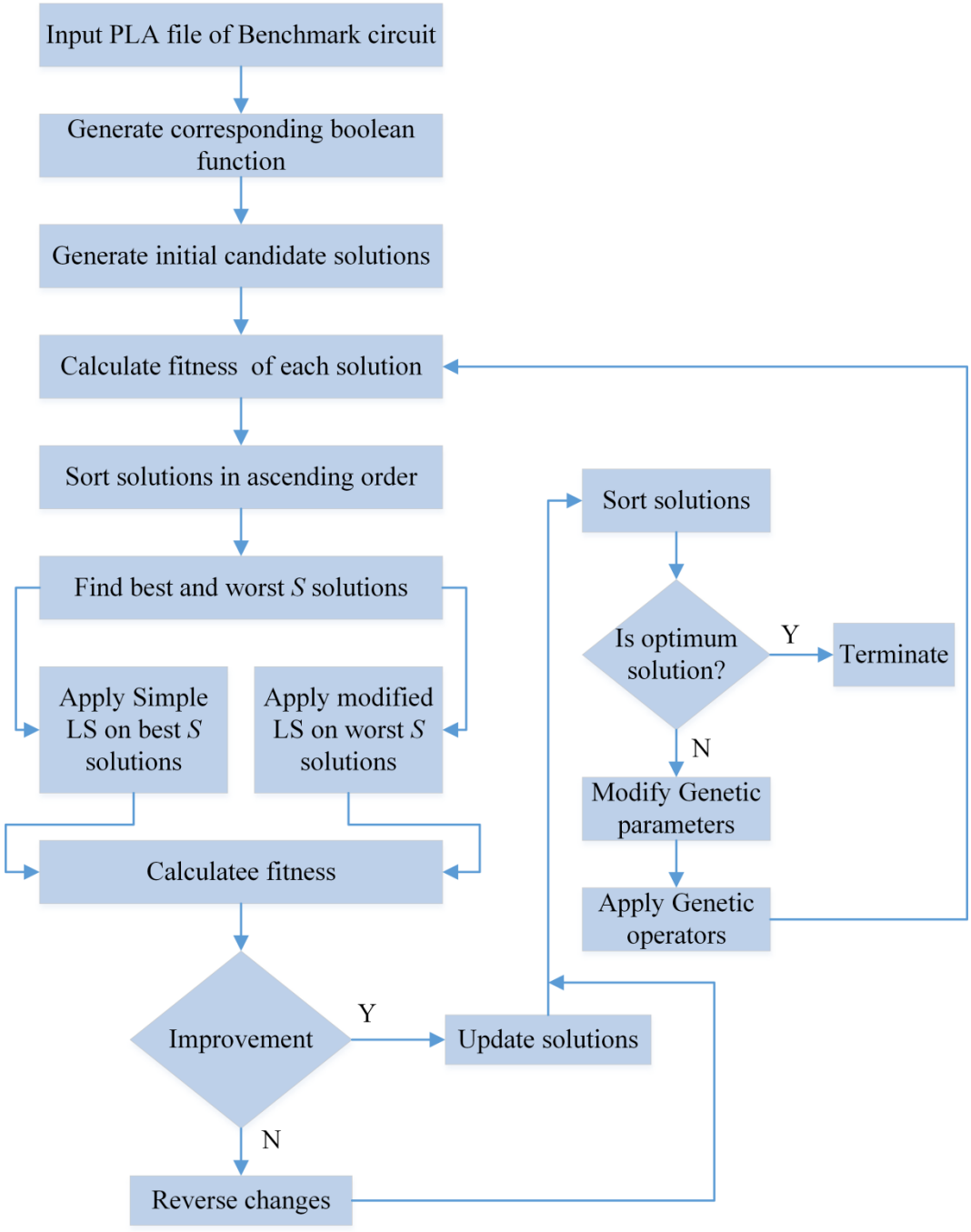


Figure 5.8 Flow Chart for proposed Modified Memetic Algorithm

This algorithm takes a population size of 20 chromosomes per generation along with crossover rate of 80% and mutation rate of 4%. The generation size begins with 20 generations and constants δ_1, $\delta_{2,}$ $\gamma_{1,}$ and γ_2 as given in [66] have been fixed at 10%. The constant μ for change in mutation rate is set as 0.05. The constants MAX_POP and MAX_GEN are kept to be 50 and the constant MIN_POP is fixed at 5. For Simple Genetic Algorithm SGA, population size and number of iterations is kept fixed at 50, other parameters being same.

A complete flowchart of the proposed MMA is presented in Figure 5.8

5.8 Overview of BDD Package

A BDD package has three main components: the BDD algorithm component, the dynamic variable ordering component and the garbage collection component. The common features of these components are described as under:

5.8.1 BDD Algorithm

This component computes the resultant BDDs for different Boolean functions. The implementation of these algorithms is typically based on depth-first traversal. The unique tables are known as hash tables with the hash collisions resolved by tying them together *i.e.* chaining. A separate unique table is related with every variable to facilitate and encourage the dynamic-variable-reordering process. The computed reserve or cache is a hash-based direct mapped (1-way associative) cache. BDD nodes support complement edges where for every edge, an additional bit is utilized to show regardless of whether the objective function should be complemented. The advantage of this encoding is that a function and its complement can be represented by the same BDD and use this extra bit in the reference edge to interpret the BDD either in the positive or the negated form. Implementation-wise, this extra bit is typically encoded in the least significant bit of the address pointer (the reference edge) to avoid incurring extra memory cost. This encoding exploits the property that address pointers in modern machines are always at least 4-byte aligned, which means the least significant bit is always 0. Thus it can be used to encode the complement information.

5.8.2 Dynamic Variable Ordering

Since in a graph, size of a BDD is sensitive to the variable input order selected, dynamic variable reordering is an important portion of every modern BDD

package. This segment aims to progressively built up a decent variable input order as the calculation progresses. When a variable reordering algorithm is invoked in a regular manner, all top-level operations that are at present being processed are prematurely ended. At that point when the variable reordering algorithm ends, these prematurely ended tasks are restarted from the starting point. The dynamic variable reordering algorithms are based on the *sifting* algorithm *i.e.* the input variable orders are changed by exchanging nodes in one level with nodes in the next adjoining level. This process is illustrated in Figure 5.9. This figure shows the sifting process for exchanging the orders of the variable *a* and the variable *b*. Each node is tagged with a number so that these can be referred easily. Let *f* be the function representing a BDD in terms of its cofactors.

$$f = v'.f|v<-0 + v.f|v<-1$$

where, *v* is the variable in b's root node and the 0-cofactor *f |v<-0* is recursively defined by the reachable sub graph of b's 0-branch child. Similarly, the 1-cofactor *f |v<-1* is recursively defined by the reachable sub graph of b's 1-branch child. Using this definition, the BDD in Figure 5.9(a) can be represented as

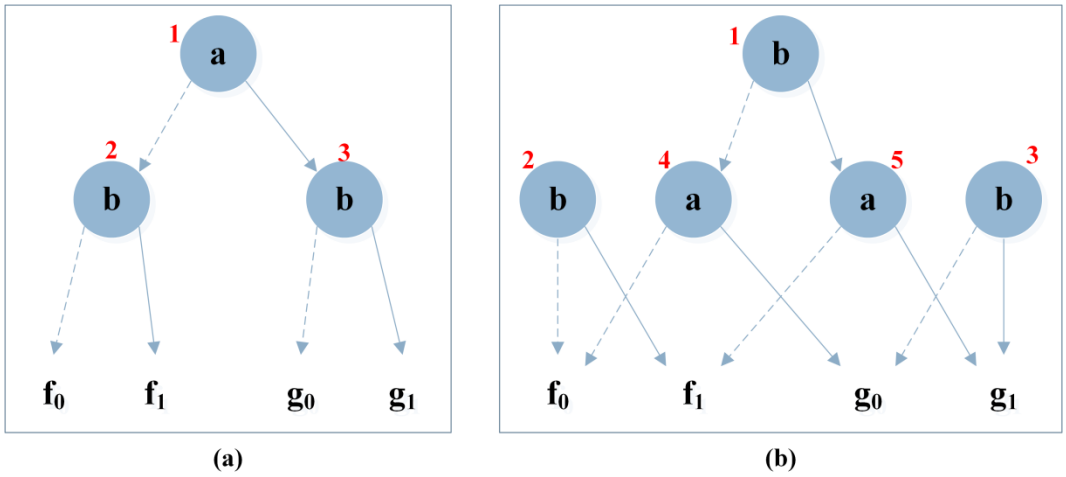
$$a'.(b'.f0 + b.f1) + a.(b'.g0 + b.g1)$$

By rearranging this formula, we get

$$b'.(a'.f0 + a.g0) + b.(a'.f1 + a.g1)$$

New BDD after the performing sifting operation is shown in Figure 5.9(b). Implementation-wise, node 4 and node 5 are created to represent the new children of node 1. Node 1 is updated to reference these new children and is relabelled with variable b. As for node 2 and node 3, they remain unchanged and if there are no references to them, they will be garbage collected later. Note that because node 1 might be referenced by others, it is important that node 1 is reused with its reachable graph representing the same function. Without this reuse, a new node will have to be created in place of node 1 and all the references to node 1 will be needed to be relocated and updated in order to reference this new node, which is

very inefficient. The node-reuse technique allows the sifting algorithm to be a local operation involving only the node and its children.



g1

Figure 5.9 Shifting process for Dynamic Variable Reordering

5.8.3 Garbage Collection

BDD computations are implicitly memory intensive because, it is all about sweeping and constructing graphs. Furthermore, in verification, many intermediate BDD results are produced to arrive at a final answer—true or false. Thus, it is important to have a good garbage collector to automatically remove BDD nodes that are no longer useful. A BDD node is referred to as approachable or reachable if it is existing in some BDD that outside user has a reference to. As outside users free the references to BDDs, some BDD nodes may no longer be approachable (deaths). These nodes are referred to as unreachable BDD nodes. Consequently all the inaccessible nodes are garbage collected by the package leading to the fast manipulation and construction of BDDs by reducing the memory requirement.

CHAPTER 6
ESTIMATION OF POWER USING PROBABLISTIC TECHNIQUE

In VLSI Design, the silicon area and the expected performance are accurately estimated before the circuit goes into fabrication. However, with the requirements of low power, the designers are faced with the added burden of qualifying their designs with respect to power dissipation. Hence, work has started in earnest to accurately estimate power dissipation at different levels. Earlier, the major factors of the VLSI designer were area, cost, performance and reliability, power considerations were only of secondary importance. But now, this has been changed and power is being given equal importance if not more in comparison to area and performance. Lots of factors have contributed to this trend. This is mainly due to the remarkable and huge success of personal and portable computing devices which demand very high speed computations and complex functionality with low power consumption.

6.1 Reduction of power at different Abstraction Levels

Limits discussed so far have been fundamental since they do not depend on the technology or the choice of power supply voltage. However, there a number of obstacles or technological limitations are in the way to approaching these limits in practical circuits and ways to reduce the effect of these various limitations could be found at all levels of digital design ranging from device to system. Battery powered systems such as laptop, computers, electronics devices etc. the need of these systems arise from the need to extend battery life. Low power design is not only needed for portable applications but also to reduce the power of high performance systems with large integration density and improved speed of operation.

Emerging of portable computing and communication equipment such as laptop, palmtops, cell-phones, *etc.* growth rate of this portable equipment are very high. Following are different levels of abstraction:

- **System level**- The system is described in terms of a set of hardware,

software and memory components and interacting algorithms they perform to provide certain functionality.

- **Behavioural or architectural level**- The individual components such as ASICs, data paths, software processes are described in the terms of their algorithmic behaviour, typically in a hardware description languages, such as behavioural VHDL, or a high-level programming language such as C. Typically, there is no timing, but only causal information is available on this level and the interface protocols controlling the communication between the interacting processes has been already fixed.
- **Register-transfer level**- In this, hardware is described in terms of arithmetic modules, registers and multiplexers and interconnects to steer data flow. RT level representations describes the basic building blocks of a system and their interconnections on clock-cycle level; it specifies the "micro-architecture" of the system under consideration.
  - **Gate-level**- A circuit is described in terms of a net list of gates or a set of Boolean equations reflecting its functionality.
  - **Transistor level**- A circuit is described in the terms of its transistor network structure.
  - **Physical level**- A circuit is described in terms of its mask layout as it will be used for fabrication. The basic building blocks on this level are polygons which reflect different materials used for fabrication such as metal, polysilicon, oxide etc.

At system level, the first decision to be made on system level is that of algorithm selection, *i.e.* selecting, from a set of given algorithms, one that provides the required functionality of the system under design while best meeting design constraints. The power consumption induced by an algorithm depends on its characteristics in terms of overall complexity, complexity of the basic operations, communication requirements etc. Although the selected algorithm itself can later be optimized, subsequent optimizations are typically of a local scope and will not be able to compensate the differences in power consumption across different algorithms, which can be up to several orders of magnitude. One of the most important factors to be taken into consideration during algorithm selection is that of memory access and management. Different abstraction levels to reduce power dissipation are shown in Figure 6.1

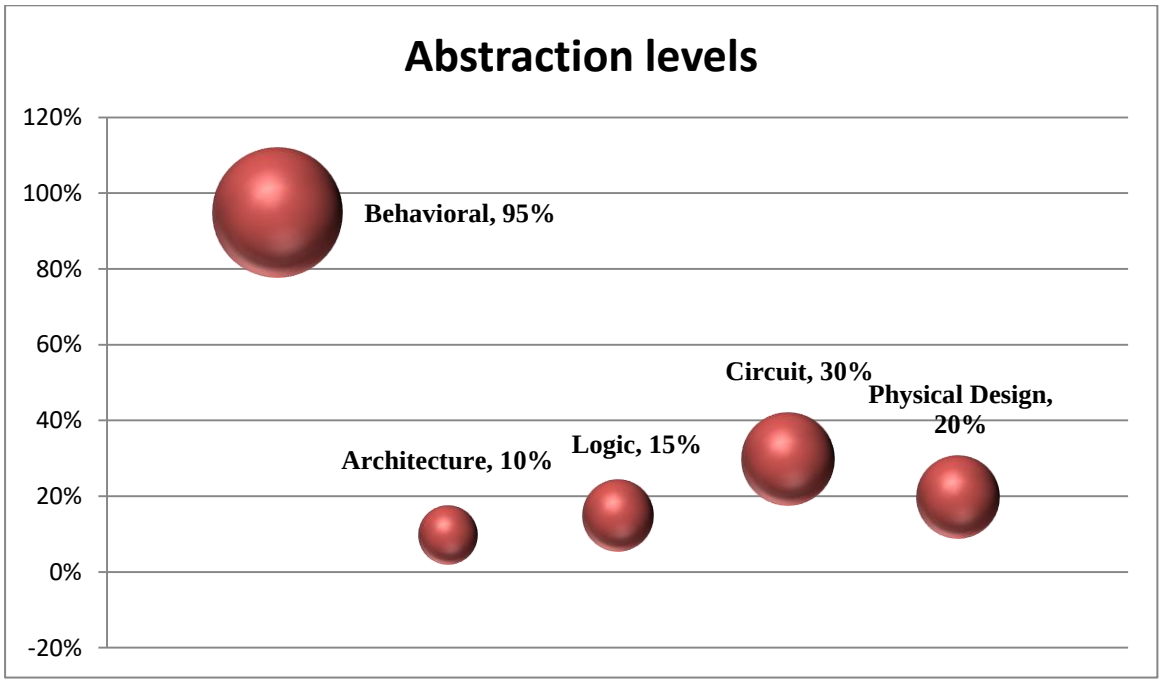


Figure 6.1 Different abstraction levels to reduce power dissipation

Behavioral level power optimization is performed during behavioural synthesis, i.e. when mapping an algorithm onto a hardware register-transfer level description. The optimization techniques can distinguish between optimizations which operate on the behavioural description alone and those which optimize the mapping of operations onto hardware units. Optimizations in register-transfer level operate by structurally transforming RT level net lists. There is a general consensus that design decisions on higher levels of abstraction have the most significant impact on the overall structure and quality of the resulting design. For instance, choosing different partitioning of functionality on design components on the system level can result in vastly different requirements on the characteristics of the individual components in terms of area, performance or power. Accordingly, optimizing transformations on higher levels of abstraction have the largest impact on the power consumption of the design as a whole. It is therefore desirable to take power into account as a cost function as early as possible in the design flow, *i.e.* on system and behavioral level, in the design flow, since in the later stages optimizations will only be of a very local scope, and hence of limited effectiveness.

6.2 Types of Power Dissipation

It has been noted that the power dissipation in CMOS circuits is mainly due to

dynamic (switching and short circuit) current and the sub threshold leakage current. With the scaling down of device sizes and transistor threshold, the sub-threshold current will increase and can become a sizable component of total power dissipation. However, in the current-day technology, the dominant component power is still the switching component (about 80% of the total power dissipation) as shown in the figure 6.2. Therefore, we will devote a considerable effort in estimation of switching power in VLSI circuits. Pi chart showing types of power dissipations are shown in Figure 6.2

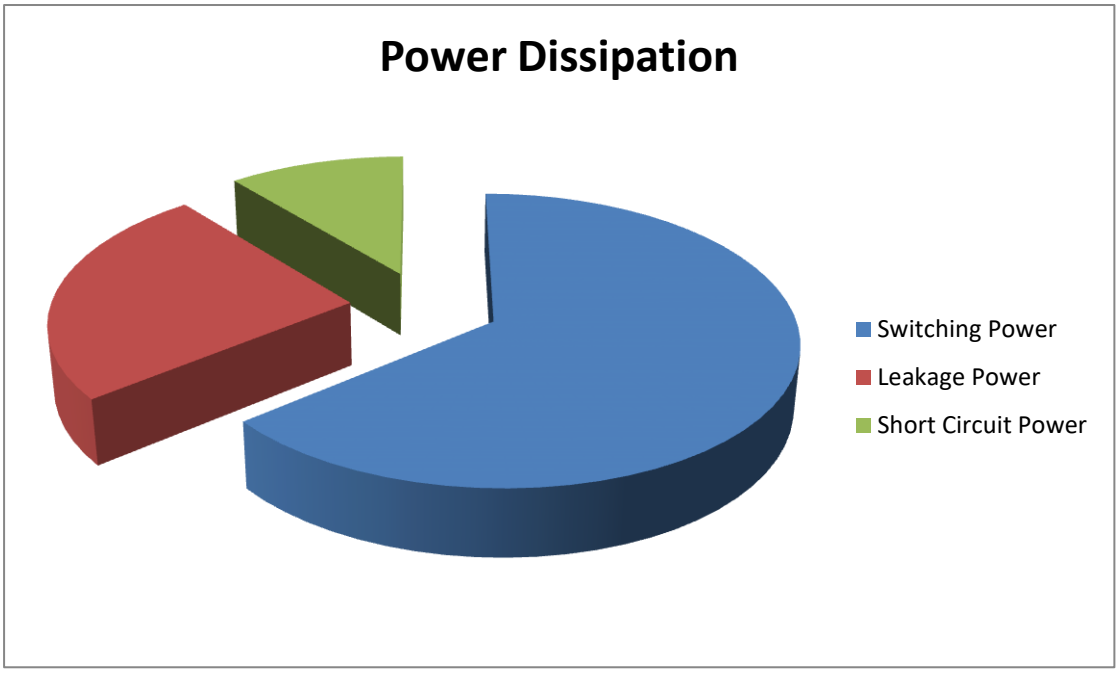


Figure 6.2 Pi chart showing types of power dissipations

6.3 WHY ESTIMATE POWER DISSIPATION?

Average power estimation involves accurately estimating the average switched capacitances at the internal nodes of the circuit. Accurate estimation of switched capacitance at the architecture or behavioural level is quite difficult, while they can be more accurately estimated at the transistor circuit level. An accurate estimate of power dissipation during various phases of VLSI design can verify the power dissipation requirements and thereby avoid complicated and expensive design changes that might be required due to power constraint violations. Power estimates can also be used during the circuit, logic and high-level synthesis to

accurately trade off power versus other design parameters such as area, performance and noise margin.

6.4 Probabilistic technique for Switching Power Optimization

The importance of low-power optimization is growing due to the increased use of battery-powered embedded systems. In order to optimize for low power dissipation, statistical information about the behavior of the system can be exploited. The switching activity of a circuit node in a CMOS digital circuit directly contributes to the overall dynamic power dissipation [68]. Temporal correlation of the occurring input signals can have a significant effect on the switching activity and hence the power consumption. The power dissipation estimate for a mapped BDD node is based on its switching activity and its fan out (corresponding to the capacitive load).

6.4.1 Switching Activity Estimation

To calculate the switching activity, assumption is made that the input signals are mutually independent (spatially uncorrelated) and that the signals can be modeled as strict-sense stationary (SSS) and mean-ergodic with zero delay which means that all switching is carried out simultaneously, and signal probability values and switching activity do not vary over time [69].

P (f) denotes the probability that f is 1 (the output probability of *f*), and a (f) denotes the activity for *f* (the probability that f will change in value from one cycle to the next).

In order to devise an improved low-power synthesis method for BDD mapped circuits [69], an accurate and computationally efficient switching activity estimation method is needed that is able to utilize temporal correlation. To avoid the high computational complexity of an exact method, it is assumed that there is no spatial correlation between the Shannon cofactors of the function of interest. The approximation technique provides the exact result for the case where the cofactors are spatially uncorrelated. In the case where cofactors are positively correlated an overestimate is obtained, since the switching of a top variable is less prone to cause a true switching of the node's output. The opposite holds for negatively correlated cofactors. This observation allows for the application of

Theorem 3.1 from [70]. The formula for the estimation of switching activity is derived using the multiplexer-based circuit model [69]:

$$a(f) = \left(\frac{P(f_0) + P(f_1) - 2\,P(f_0)\,P(f_1)}{1-(a(f_0)+a(f1)-a(f_0)a(f_1))} - \frac{\frac{1}{2}(a(f_0)+a(f_1)-a(f_0)a(f_1))}{1-(a(f_0)+a(f_1)-a(f_0)a(f_1))}\right) * (a(v)(1-a(f0))(1-a(f1)))$$

$$+ \left(\frac{1-P(v)-\frac{a(v)}{2}}{1-a(v)}\right) * a(f0)(1-a(v))(1-a(f1))$$

$$+ \left(\frac{P(v)-\frac{a(v)}{2}}{1-a(v)}\right) * a(f1)(1-a(v))(1-a(f0))$$

$$+ \frac{1}{2}a(f0)a(v)(1-a(f1)) + \frac{1}{2}a(f1)a(v)(1-a(f0))$$

$$+ a(f0)a(f1)(1-a(v)) + \frac{1}{2}a(f0)a(f1)\,a(v)$$

.....(1)

In (1), v is the input variable, f0 is the low cofactor, and f1 is the high cofactor. This formula is used recursively in a bottom-up approach to calculate the activity for each node in the BDD.

6.4.2 Low-power Estimation

The power dissipation of each node of the binary decision diagram is computed by the estimated switching activity and the fan out of the node.

6.4.3 BDD mapped circuits

A BDD is directly mapped to a multiplexer-based circuit and the resulting circuit is obtained by replacing BDD vertices with small sub circuits and BDD edges with wires. The diagram size (and therefore the circuit complexity) is sensitive to the ordering of the function variables. The complexity varies from linear to exponential under different orderings for some functions. The power dissipation of each node is computed by the estimated switching activity and the fan out of the node. The variable order of the underlying BDD influences not only the area (number of nodes) but also the internal switching activity. The technique is also used with BDDs using complemented edges. The use of complemented edges is shown to both reduce BDD complexity and improve performance of operations.

1. The output probability, P ($\bar{f}$), of $\bar{f}$ is equal to 1-P (f).

2. The switching activity, a (f^{-}), of f^{-} is equal to a (f).

These properties are used to compute local switching probabilities during variable exchange operations on BDDs with complemented edges. A cost model based on the total circuit switching activity under a given set of dependent-variable output probabilities is defined. The dependent variables are denoted as support variables. The sum of all internal switching activities at each BDD vertex is minimized. Each BDD node is then mapped into a multiplexer-based circuit. The number of stages of active buffers is determined by the fan out of each BDD node, which is equivalent to the number of BDD edges pointing to the node. The power dissipation for the mapped node PD_n is estimated using the relationship in (2) [61, 16]:

$$\mathbf{PDn = a\ (n)\ x\ driver\ (fan\ out\ (n))\ +\ leakage\ (n)} \qquad \mathbf{(2)}$$

In (2), the power dissipation due to leakage is ignored as its contribution to total power dissipation is very less; therefore, leakage (n) is ignored.

CHAPTER 7

RESULTS AND DISCUSSIONS

7.1 Comparison of Node Count among various Evolutionary Algorithms

In this chapter, computational results are shown by applying the proposed Modified Memetic Algorithm with a number of Benchmark circuits taken from the LGSynth93 benchmark circuit suites. The MMA based technique, is implemented with C++ codes and experimented by running on a Ubuntu 12.04. The node count comparison between various Evolutionary algorithms on LGSynth93 Benchmark circuits has been shown in Table 7.1

Table 7.1 Node Count Comparison of MMA results with Evolutionary Algorithms

Circuit Name	**#i**	**#o**	**Original Node Count**	**SGA(within 10 runs)**	**HGA (within 5 runs) It**		**PSO [17]**	**MMA (within 2 runs) It**	
					Nodes	it		Nodes	it
5xp1	7	10	88	68	68	15	68	68	8
9sym	9	1	33	33	33	15	----	33	8
b12	15	9	91	68	67	16	62	59	9
bw	5	28	118	106	106	15	100	106	8
clip	9	5	254	96	95	19	112	93	8
con1	7	2	18	15	15	11	16	15	6
inc	7	9	73	62	61	17	79	61	9
misex1	8	7	47	37	37	14	37	36	8
misex2	25	18	140	99	95	18	89	86	14
misex3c	14	14	844	527	456	16	462	445	8
rd53	5	3	23	23	23	16	----	23	6
rd73	7	3	43	43	43	16	----	43	6
rd84	8	4	59	59	59	16	----	59	6
sao2	10	4	154	97	95	18	91	85	7
sqrt8	8	4	42	35	35	14	33	33	7
squar5	5	8	38	37	37	17	37	37	6
vg2	25	8	1087	222	255	18	182	151	9
Z5xp1	7	10	69	68	68	14	----	68	8

where, *It* represents the number of iterations.

The Table 7.1 compares the results of MMA based program with the other evolutionary algorithms *i.e.* simple genetic algorithm (SGA), hybrid genetic algorithm (HGA) and the latest particle swarm optimization algorithm (PSO) [30].

Table 7.2 Node Count Comparison of MMA with Dynamic Variable Ordering Algorithms

Circuit name	**Initial Node Count**	**Win 2**	**Win 2ite**	**Win 3**	**Sift**	**MMA (Proposed)**	**% improvement w.r.t. Original size**	**% improvement w.r.t. Sift Algo**
5xp1	88	87	82	82	82	**68**	22.72	17.07
b12	91	86	86	65	65	**59**	35.16	9.23
Bw	118	113	112	106	106	**106**	10.17	0.00
Clip	254	178	108	105	105	**93**	63.37	11.42
con1	18	18	18	18	18	**15**	16.67	16.67
Inc	73	75	69	68	68	**61**	16.43	10.29
misex1	47	43	42	41	41	**36**	23.40	12.19
misex3c	844	750	719	490	454	**445**	49.66	1.98
sao2	154	149	109	91	92	**85**	44.81	7.60
sqrt8	42	40	39	37	37	**33**	21.43	10.81
squar5	38	38	38	38	38	**37**	2.63	2.60
vg2	1087	864	113	104	103	**151**	86.11	-46.70
Z5xp1	69	68	68	68	68	**68**	1.45	0.0
Average							***24.03***	***2.75***

Table 7.2 performs comparison with the reordering heuristics that have been incorporated in the BDD package *Buddy-2.4*, such as Window Permutation WIN2, WIN2ite, WIN3 and Sifting algorithm [16], [17]. Percentage improvement observed by using MMA as variable reordering algorithm with respect to original size and as compared to Sifting algorithm is also indicated. The results indicate

that in 55.55% cases, our MMA based approach has resulted in lesser number of node counts for the SDDs. Also, for all circuits, number of iterations to generate optimum variable order is lesser than the corresponding HGA.

From the data in Table 7.2, it is observed that MMA resulted in an average reduction of 24.03% in SDD sizes. Also the proposed MMA provides a 2.75% average improvement in node count of Benchmark circuits as compared to the best known dynamic algorithm *i.e.* Sift Algorithm.

7.2 Graphical Analysis of Results

Figure 7.1 shows the differences in runtime of HGA and MMA in finding the optimum variable ordering for different BDD circuits. It provides a graphical view of the reduction in number of iterations done by the proposed MMA in achieving better results as compared to those achieved by using HGA. The Figure 7.1 shows that as the number of input and output variables in the circuit increases, the runtime of MMA remains half that of HGA even when all the performance influencing constants are kept same in these algorithms.

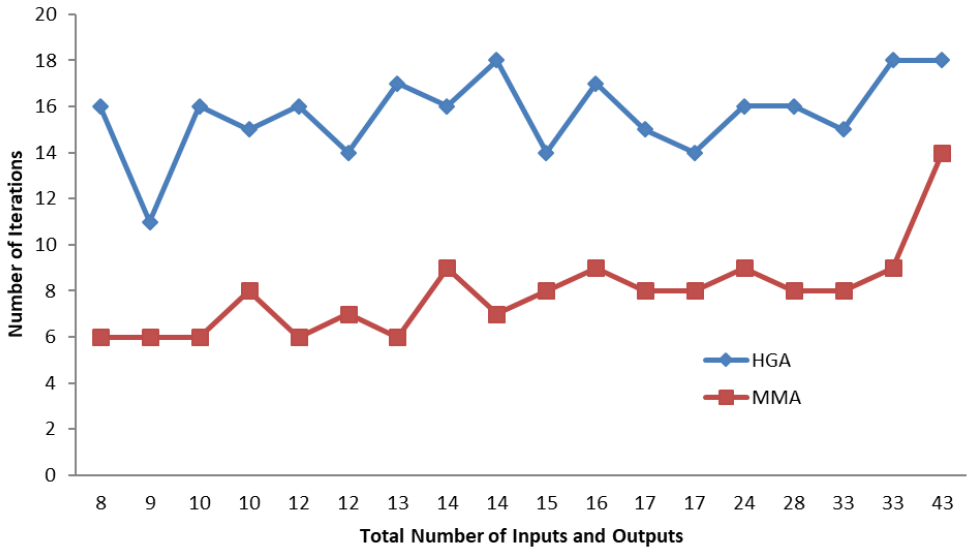


Figure 7.1 Graph showing runtime differences in HGA and MMA

A graphical view of reduction in BDD sizes in terms of node count by using various optimization algorithms has been shown in Figure 7.2. Accordingly, it can be observed that in most of the circuits, the results provided by the proposed

MMA are better as compared to other *state-of-art* algorithms.

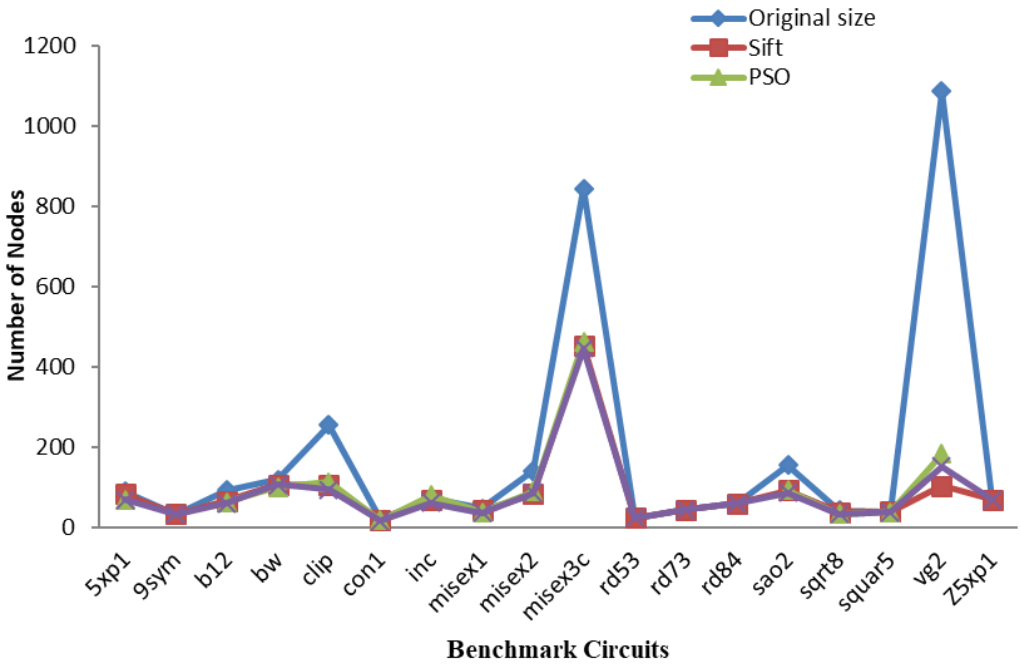


Figure 7.2 Graph showing reduction in BDD size in terms of node count

7.3 Estimation of Node Count and Computation Time for Genetic Algorithm along with crossover techniques

The Proposed Genetic Algorithm and Hybridized Genetic Algorithm, using three types of crossover operators, are implemented with C++ codes and simulated using BUDDY 2.4 package on Ubuntu 12.04. Simulation results *i.e.* node count and computation time for LGSynth93 Benchmark Circuits have been represented in respective tables. These proposed algorithms and approaches provide the optimal variable order with minimum node count in minimum iterations for every function.

In comparison with other existing algorithms, it is observed that the proposed Genetic Algorithm gives minimum number of nodes. When the crossover operators are compared, Partially Mapped Crossover gives minimum number of nodes, while the Cycle Crossover gives minimum computation time for most of the LGSynth93 Benchmark Circuits. The order crossover also results in near - optimum variable order. In the LGSynth93 Benchmark circuits, as number of input variables are increasing, the proposed genetic algorithm with PMX crossover gives best reduction in number of nodes, whereas with cycle crossover it gives best diminution in computation time.

Table 7.3 Comparison of Node Count for Dynamic and Proposed Genetic algorithms LGSynth93 Benchmark Circuits

Benchmark Circuits	I/O	Initial Node Count	WIN2	WIN2ite	WIN 3	SIFT	Random	Proposed GA with Crossover		
								Order	Cycle	PMX
b12	15/9	91	86	86	65	65	76	66	68	60
sao2	10/4	154	149	109	91	92	132	90	92	85
5xp1	7/10	88	87	82	82	82	88	68	68	68
Bw	5/28	118	113	112	106	106	118	106	106	106
con1	7/2	18	20	18	16	18	18	15	16	15
Inc	7/9	81	83	80	74	73	86	72	72	72
sqrt8	8/4	42	40	39	37	37	48	33	34	33
squar5	5/8	38	38	38	38	38	38	37	37	37
Clip	9/5	105	254	178	108	105	105	107	157	94
xor5	5/1	9	9	9	9	9	9	9	9	9
Pdc	16/40	696	666	658	640	640	709	665	696	630
duke2	22/29	976	825	777	360	358	414	452	537	364
misex1	8/7	47	43	42	41	41	47	37	37	36
misex2	25/18	99	140	126	111	86	83	105	122	89
misex3c	14/14	750	719	490	454	527	844	549	680	469
table5	17/15	873	776	738	717	712	974	826	926	740
t481	16/1	32	32	32	32	32	74	32	89	32

From Table 7.3 it is inferred that there is no reduction in node count for multi-input single output benchmark circuits, but for multi-input multi-output benchmark circuits there is percentage reduction in node count in the following order i.e. for squar5 it is about 3%, for pdc 9%, for bw, clip and misex2 10%, for inc 11%, for table5 15%, for con1 17%, for sqrt8 21%, for 5xp1 and misex1 23%,

for b12 34%, for misex3c 37%, for sao2 45% and for duke 63%. Therefore, the proposed Genetic algorithm is well suited for multi-input multi-output (MIMO) VLSI circuits.

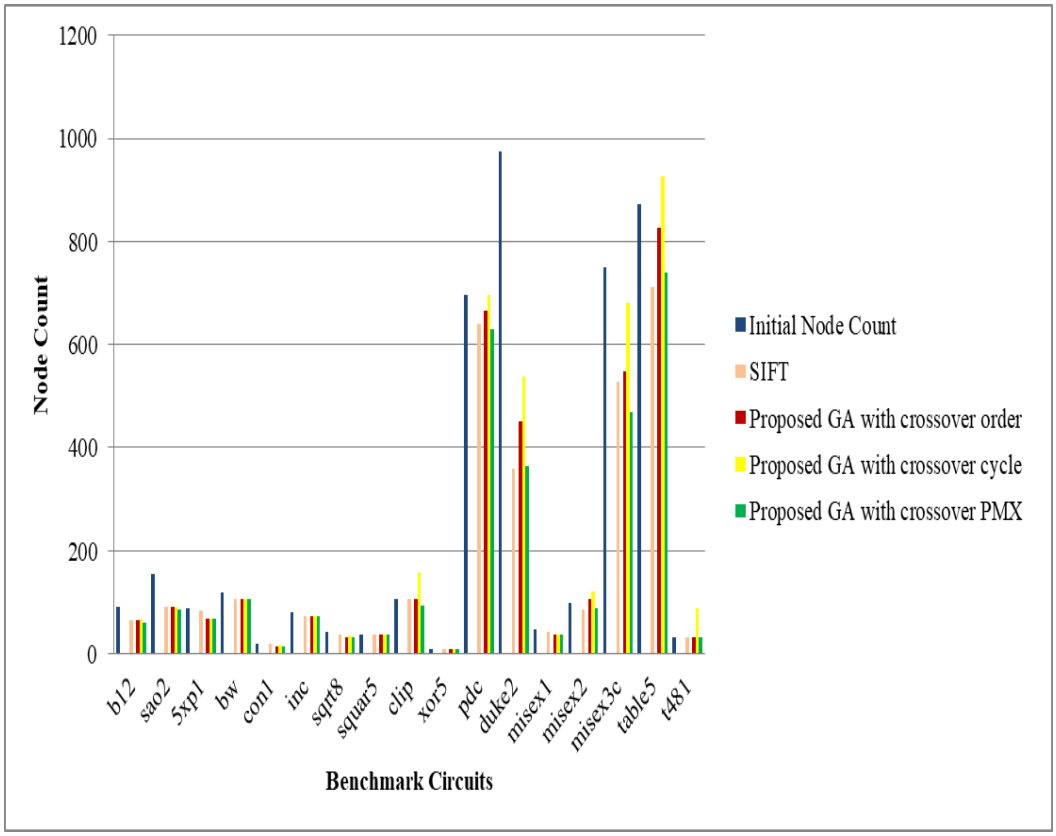


Figure 7.3 Graphical Analysis showing Node Count Comparison by applying Proposed Genetic algorithms for LGSynth93 Benchmark Circuits

Figure 7.3 shows graphical representation of node count by applying proposed genetic algorithm on LGSynth93 benchmark circuits. The initial node count is quite high for every benchmark circuit shown in Figure 7.3 and proposed genetic algorithm gives optimum result with PMX crossover operator shown.

Table 7.4 Comparison of Computation Time for proposed Genetic algorithm for LGSynth93 Benchmark Circuits

Benchmark Circuits		Inputs/ Outputs	Proposed GA with crossover		
			Order(sec)	Cycle (sec)	PMX (sec)
1	**b12**	15/9	0.13	0.12	0.3
2	**sao2**	10/4	0.22	0.22	0.47
3	**5xp1**	7/10	0.15	0.19	0.31
4	**bw**	5/28	0.37	0.58	0.66
5	**con1**	7/2	0.07	0.06	0.09
6	**inc**	7/9	0.14	0.16	0.2
7	**sqrt8**	8/4	0.1	0.1	0.19
8	**squar5**	5/8	0.08	0.13	0.15
9	**clip**	9/5	0.26	0.18	1.51
10	**xor5**	5/1	0.08	0.08	0.1
11	**pdc**	16/40	0.55	0.5	3.42

From the Table 7.4 it is evident that for order crossover the computation time comes out to be less as compared to the two other crossover techniques *i.e.* cycle and PMX when implemented on LGSynth93 Benchmark Circuits. In **clip at SNo. 9**, the PMX crossover is yielding computation time of 1.51 seconds whereas cycle crossover is yielding computation time of 0.18 seconds where as in **pdc at SNo. 11.** PMX yields 3.42 seconds and order gives 0.5 as the computation time. As while moving down from b12 to pdc in the Table 7.4, cycle crossover or order crossover are yielding minimum computation time in comparison to PMX crossover.

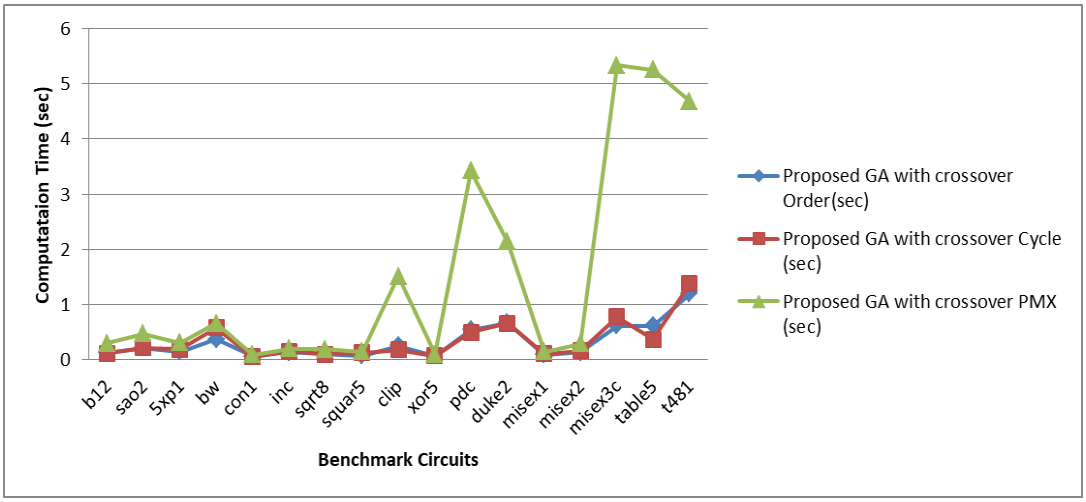


Figure.7.4 Graphical Analysis representing Computation Time for Proposed Genetic algorithm on LGSynth93 Benchmark Circuits

Figure 7.4 shows the graphical representation of computation time for genetic algorithm on LGSynth93 Benchmark Circuits.

7.4 Node Count and Computation Time estimation for Hybridized Genetic Algorithm

Table 7.5 Comparison of Node Count for Dynamic and Hybridized Genetic algorithms for LGSynth93 Benchmark Circuits

Benchmark Circuits	#I / #O	Initial Node	Win2	Win2ite	Win 3	Sift	Random	Proposed HGA with Crossover		
								Order	Cycle	PMX
b12	15/9	91	86	86	65	65	76	40	39	30
sao2	10/4	154	149	109	91	92	132	96	88	70
5xp1	7/10	88	87	82	82	82	88	33	59	49
Bw	5/28	118	113	112	106	106	118	106	106	106
con1	7/2	18	20	18	16	18	18	15	15	15
Inc	7/9	81	83	80	74	73	86	72	72	72
sqrt8	8/4	42	40	39	37	37	48	33	33	33
squar5	5/8	38	38	38	38	38	38	21	21	19
Clip	9/5	105	254	178	108	105	105	94	94	94
xor5	5/1	9	9	9	9	9	9	9	9	9
duke2	22/29	976	825	777	360	358	414	508	473	446
misex1	8/7	47	43	42	41	41	47	36	36	36

misex2	25/18	99	140	126	111	86	83	92	96	90
misex3c	14/14	750	719	490	454	527	844	528	548	506
t481	16/1	32	32	32	32	32	74	59	58	74

It is observed that the proposed Hybridized Genetic Algorithm gives better results in terms of number of nodes as compared to existing Dynamic algorithms and proposed genetic algorithm. When the proposed algorithm is implemented on LGSynth93 Benchmark circuits, Partially Mapped Crossover yields reduced number of nodes and the Cycle Crossover gives diminished computation time. The order crossover yields near-optimal results.

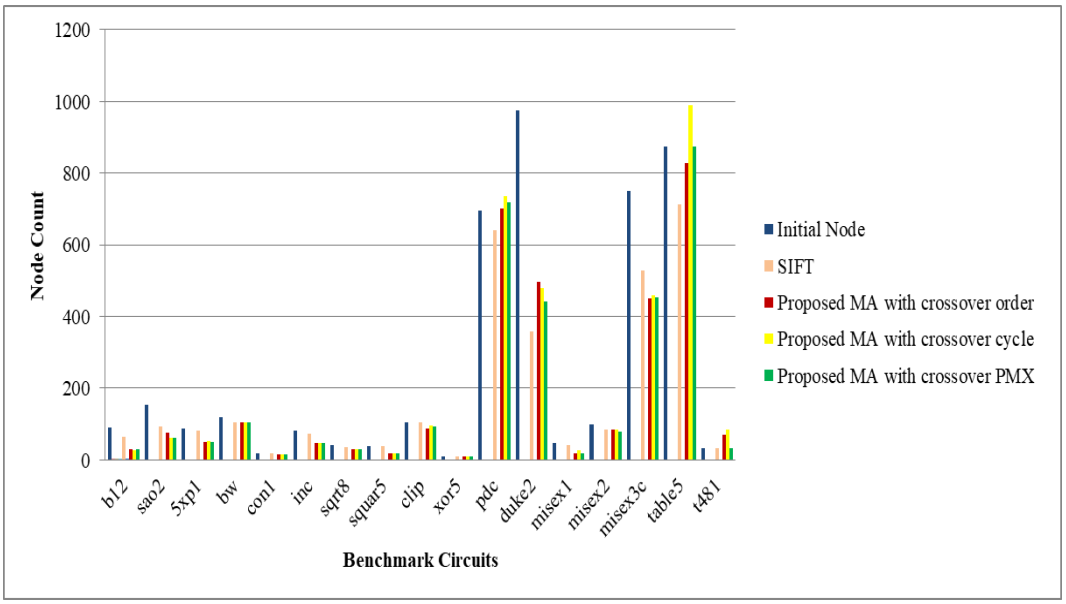


Figure.7.5 Graphical Analysis showing Node Count Comparison for Proposed Hybridized Genetic algorithms LGSynth93 Benchmark Circuits

Table 7.5 represents that for multi-input single output there is no reduction in node count, but for multi-input multi-output circuits i.e. squar5 it is about 50%, for bw, 10%, for clip 12.38%, for misex2 19.2%, for inc 42%, for misex1 61.7%, for con1 16.6%, for sqrt8 28.6%, for 5xp1 42%, for b12 68%, for misex3c 39.5%, for sao2 60.4% and for duke2 54.7%. Therefore, the proposed Hybridized Genetic algorithm is better suited for multi-input multi-output (MIMO) VLSI circuits as compared to proposed genetic algorithm because for the same set of benchmark circuits proposed Hybridized Genetic algorithm is giving better node count. Figure 7.5 represents the graphical analysis of node count for propose hybridized genetic algorithm with dynamic variable ordering algorithms.

Table 7.6 Computation Time comparison for Different Crossover Operators in Hybridized Genetic algorithm for LGSynth93 Benchmark Circuits

Benchmark Circuits	**Inputs**	**Outputs**	**Proposed HGA with crossover**		
			Order(sec)	Cycle (sec)	PMX (sec)
b12	15	9	0.13	0.03	0.18
sao2	10	4	0.22	0.2	0.47
5xp1	7	10	0.12	0.08	0.14
bw	5	28	0.25	0.08	0.14
con1	7	2	0.01	0.01	0.01
inc	7	9	0.07	0.04	0.11
sqrt8	8	4	0.04	0.01	0.07
squar5	5	8	0.02	0.03	0.06
clip	9	5	0.34	0.29	0.91
xor5	5	1	0.01	0.02	0.03
pdc	16	40	0.29	0.2	0.32
duke2	22	29	0.73	0.35	1.76
misex1	8	7	0.04	0.11	0.07
misex2	25	18	0.07	0.06	0.22
misex3c	14	14	1.18	1.05	2.86
table5	17	15	1.26	0.27	3.14
t481	16	1	1.48	1.23	3.08

From the Table 7.6 it is apparent that for cycle crossover the computation time comes out to be minimum as compared to the two other crossover techniques when implemented on LGSynth93 Benchmark Circuits. In **clip**, the PMX crossover is yielding computation time of 0.91 seconds whereas cycle crossover is yielding computation time of 0.29 seconds whereas in **pdc** PMX yields 0.32 seconds and cycle gives 0.06 as the computation time. As we go down from b12

to t481 in the Table 7.6, cycle crossover or order crossover are yielding minimum computation time in comparison to PMX crossover.

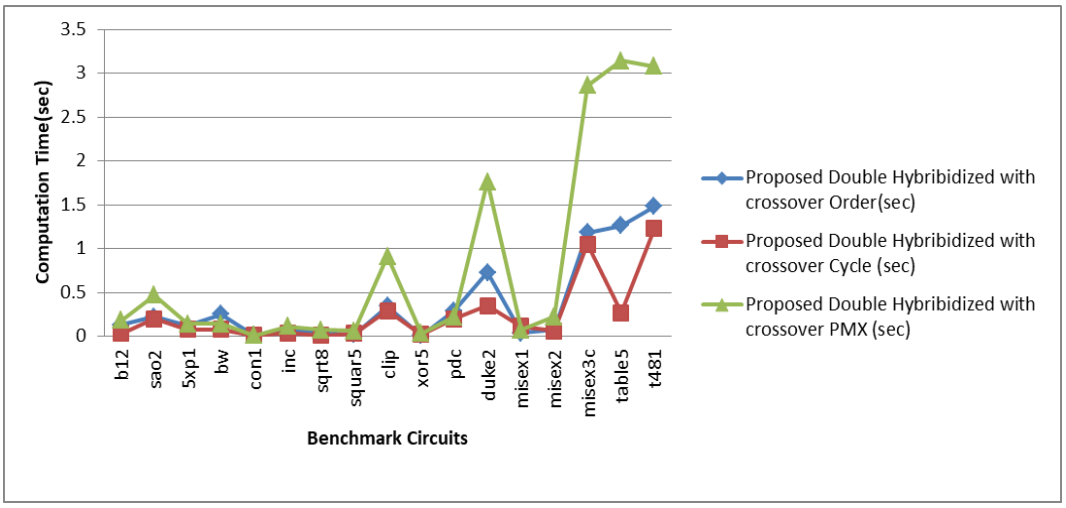


Figure 7.6 Graphical Analysis representing Computation Time for Proposed Hybridized Genetic algorithm on LGSynth93 Benchmark Circuits.

7.5 Power Estimation

In BDD, each node is represented as a single 2x1 multiplexer. After Synthesis of a 2x1 multiplexer in Synopsys tool, the power calculated is 762.3125 nW as shown in Table 7.7.

Table 7.7 Report for Power calculation using Synopsys tool

****

Report : power -
analysis_effort
low Design :
mux_2 Version:
Y-2006.06-SP4

Date : Thu May 29 12:18:01 2014

****

Library(s) Used:

fsa0a_c_generic_core_tt1p8v25c(File:/cad/DigitalFDKs/faraday180nm/core180nm/fsa0a _c/
2009Q2v2.0/GENERIC_CORE/FrontEnd/synopsys/fsa0a_c_generic_core_tt1p8v25c.db) Operating Conditions: TCCOM Library:
fsa0a_c_generic_core_tt1p8v25c

Wire Load Model Mode:
enclosed Design Wire
Load Model Library
---
mux_2 G5K
fsa0a_c_generic_core_tt1p8v25c
Global Operating Voltage = 1.8
Power-specific unit information:

Voltage Units = 1V
Capacitance Units =
1.000000pf Time Units
= 1ns
Dynamic Power Units = 1mW (derived from
V,C,T units) Leakage Power Units = Unit less

Cell Internal Power = 765.1176 nW (87%)
Net Switching Power = 97.1948 nW (13%)

Total Dynamic Power = 762.3125 nW (100%)

***** End Of Report **
Total power dissipation of the circuit can be calculated by multiplying the power calculated for a BDD (represented as 2x1 Multiplexer) with the node count. It can be represented as below

Total power = node count * 762.3125nw (1)

Results have been calculated for LGSynth93 Benchmark Circuits using various

crossover operators in Genetic and Hybridized Genetic Algorithms. Table 7.8 performs comparison of the Power estimated using Probabilistic techniques [15] with the already existing algorithms (values calculated using Synopsys tool). The results indicate that in more than 90% cases, proposed Genetic and Hybridized Genetic algorithm based approach using probabilistic analysis has shown lesser power dissipation as compared to existing MUX based technique.

Table 7.8 Comparison of the Power estimated using Probabilistic technique and MUX Based technique (values calculated using Synopsys tool) in Proposed Genetic Algorithm for LGSynth93 Benchmark Circuits

Bench-mark Circuits	Power for Order Crossover Operator(in mW)		Power for Cycle Crossover Operator(in mW)		Power for PMX Crossover Operator(in mW)	
	Proposed Probabilistic Technique	Existing MUX based Technique	Proposed Probabilistic Technique	Existing MUX based Technique	Proposed Probabilistic Technique	Existing MUX based Technique
b12	0.000121	0.00335	0.0000257	0.050312	0.051837	0.045738
5xp1	0.000062	0.001882	0.000080	0.051837	0.051837	0.051837
Bw	0.073300	0.000009	0.000010	0.080805	0.080805	0.080805
con1	0.006226	0.011720	0.011720	0.011434	0.012197	0.011434
Inc	0.000152	0.000697	0.000697	0.054886	0.054886	0.054886
sqrt8	0.000052	0.003183	0.0000545	0.025156	0.025919	0.025156
xor5	0.866600	0.866600	0.866600	0.006860	0.068600	0.068600
Pdc	0.000008	0.000931	0.000004	0.506938	0.530569	0.480256
duke2	0.000349	0.000636	0.000100	0.000349	0.000636	0.0001
misex1	0.000010	0.000433	0.000819	0.028205	0.028205	0.028205
misex3c	0.003610	0.091900	0.006960	0.41851	0.518372	0.357524
table5	0.000101	0.000914	0.000037	0.000101	0.000914	0.000037
t481	0.0000086	0.000054	0.000003	0.024394	0.067845	0.024394
squar5	0.000314	0.000314	0.000314	0.028205	0.028205	0.028205
Clip	0.000065	0.0000045	0.000002	0.081567	0.119683	0.071657
sao2	0.000004	0.000008	0.000001	0.068608	0.070132	0.064796

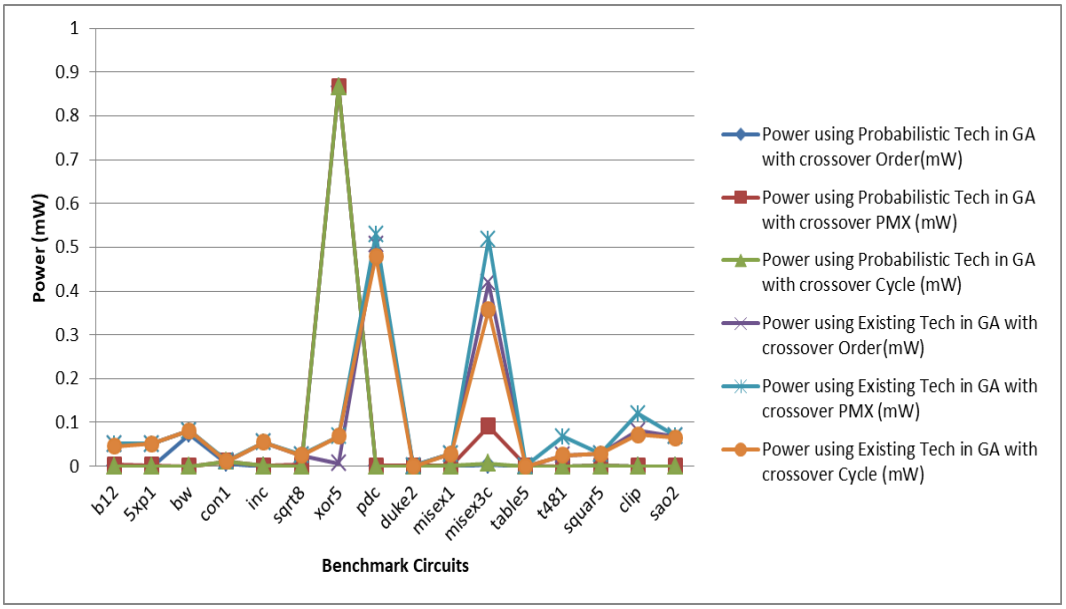


Fig. 7.7 Comparison of the power estimated by Probabilistic Technique and Existing Technique in Proposed Genetic Algorithm for LGSynth93 Benchmark Circuits.

Table 7.9 Comparison of the Power calculated using Probabilistic technique and the existing technique in Hybridized Genetic Algorithm for LGSynth93 Benchmark Circuits

Benchmark Circuits	Power for Order Crossover Operator(in mW)		Power for Cycle Crossover Operator(in mW)		Power for PMX Crossover Operator(in mW)	
	Proposed Probabilistic Technique	Existing MUX Based Technique	Proposed Probabilistic Technique	Existing MUX Based Technique	Proposed Probabilistic Technique	Existing MUX Based Technique
b12	0.00041	0.001708	0.000129	0.050313	0.052599	0.046501
sao2	0.000009	0.000064	0.000004	0.068608	0.074707	0.064796
5xp1	0.001531	0.019833	0.003654	0.051837	0.051837	0.051837
Bw	0.012910	0.093140	0.021436	0.080805	0.035066	0.035066
con1	0.011721	0.021983	0.010114	0.011435	0.011435	0.011435
Inc	0.0025651	0.002565	0.002565	0.054886	0.057936	0.054886
sqrt8	0.0002155	0.001640	0.000361	0.025156	0.025919	0.025156
squar5	0.0004335	0.000434	0.000433	0.028205	0.028205	0.028205
Clip	0.144639	0.000038	0.000010	0.070895	0.089953	0.070895
xor5	0.866615	0.866615	0.866615	0.006860	0.006860	0.006861
Pdc	0.019800	0.083800	0.010101	0.154749	0.174569	0.153225
duke2	0.036190	0.091365	0.010139	0.341516	0.477207	0.322458
misex1	0.000819	0.000844	0.000636	0.027443	0.028205	0.027443
misex2	0.000018	0.000249	0.000063	0.067845	0.083854	0.069370
misex3c	0.001936	0.061943	0.006611	0.358287	0.451289	0.343803

table5	0.009654	0.012391	0.003416	0.551151	0.754689	0.610612
t481	0.0032611	0.003261	0.003261	0.024394	0.068608	0.024394

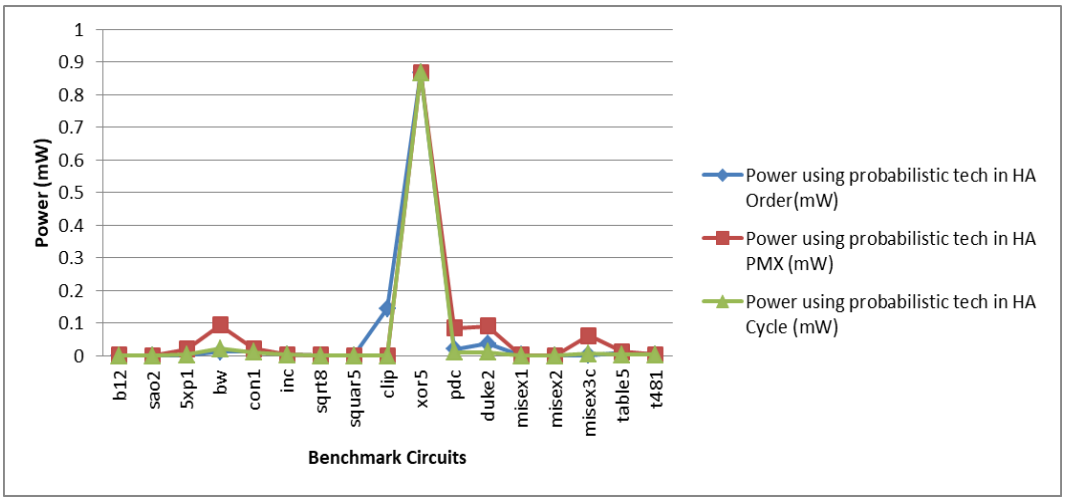


Figure. 7.8 Power Estimation using Probabilistic Technique and Existing Technique in Hybridized Genetic Algorithm on LGSynth93 Benchmark Circuits

In case of the estimation of power dissipation, it is observed that that in more than 90% cases, Genetic and Hybridized algorithm based approach using probabilistic analysis has shown lesser power dissipation than the Multiplexer based existing technique. From Tables 7.8 and 7.9, it can be concluded that the power dissipation in case of PMX and Order Crossover Operators is lesser than Cycle Crossover Operator in case of both Simple Genetic Algorithm and Hybridized Genetic Algorithm for 80% benchmark circuits.

Therefore, it is concluded that the Evolutionary algorithms namely Simple Genetic Algorithm and Hybridized Genetic Algorithm are suited for the optimization of area, speed and power in MIMO VLSI circuits.

7.6 Estimation of Node Count, Computation Time and Power for Modified Memetic Algorithm

Table 7.10 shows comparison with the reordering heuristics that have been incorporated in the BDD package *Buddy-2.4*, such as Window Permutation WIN3 and Sifting algorithm [15], [16]. The results indicate that in 75.55% cases, our MMA based approach has resulted in lesser number of node counts for the SDDs. Also, for ll circuits, number of iterations to generate optimum variable order is

lesser than the corresponding MMA. It is observed that MMA resulted in an average reduction of 24.03% in SDD sizes. Also the proposed MMA provides a 2.75% average improvement in node count of LGSynth93 Benchmark circuits as compared to the best known dynamic algorithm *i.e.* Sift Algorithm

Table 7.10 Comparison of Node Count using MMA for LGSynth93 benchmark circuits with Dynamic Algorithms

Benchmark Circuits	**#I / #O**	**Initial Node Count**	**Win2**	**Win2ite**	**Win 3**	**Sift**	**Random**	**Proposed MMA with Crossover**		
								Order	Cycle	PMX
b12	15/9	91	86	86	65	65	76	29	28	29
sao2	10/4	154	149	109	91	92	132	76	63	61
5xp1	7/10	88	87	82	82	82	88	50	53	51
Bw	5/28	118	113	112	106	106	118	106	105	106
con1	7/2	18	20	18	16	18	18	16	16	15
Inc	7/9	81	83	80	74	73	86	47	47	47
sqrt8	8/4	42	40	39	37	37	48	30	30	30
squar5	5/8	38	38	38	38	38	38	19	19	19
Clip	9/5	105	254	178	108	105	105	87	96	92
xor5	5/1	9	9	9	9	9	9	9	9	9
duke2	22/29	976	825	777	360	358	414	497	478	442
misex1	8/7	47	43	42	41	41	47	19	26	18
misex2	25/18	99	140	126	111	86	83	86	84	80
misex3c	14/14	750	719	490	454	527	844	451	460	454
t481	16/1	32	32	32	32	32	74	70	85	32

It is observed that the proposed Modified Memetic Algorithm gives best results in terms of number of nodes as compared to existing Dynamic algorithms and proposed genetic and Hybridized Genetic algorithms when implemented on LGSynth93 Benchmark circuits, Partially Mapped Crossover yields reduced number of nodes and the Cycle Crossover gives diminished computation time. The order crossover yields near-optimal results.

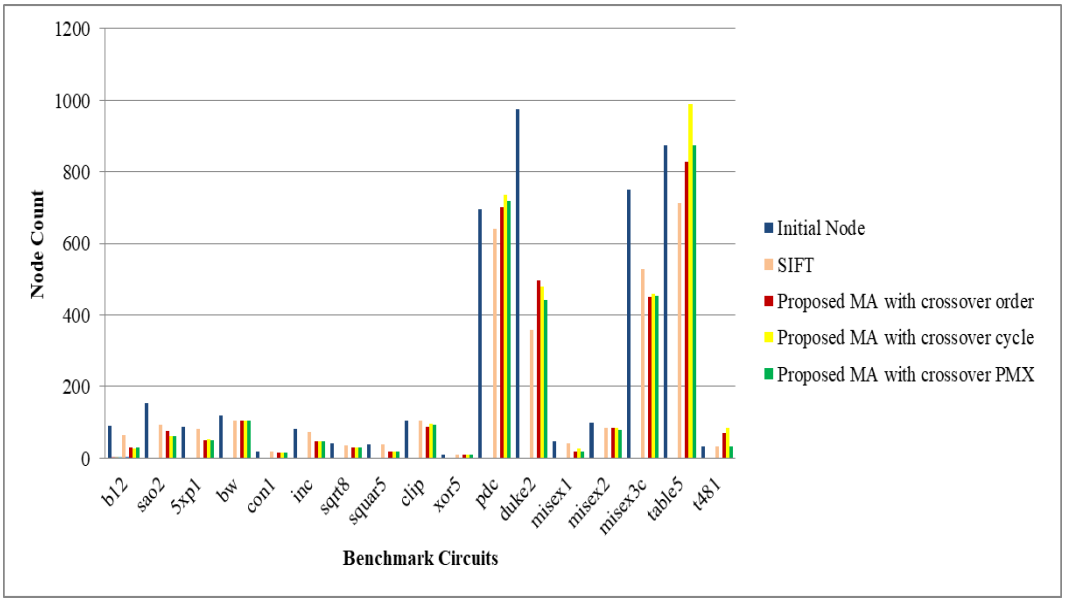


Figure 7.9 Graphical Analysis showing Node Count comparison using MMA for LGSynth93 Benchmark Circuits

Table 7.10 represents that for multi-input single output there is approximately zero reduction in the node count, but for multi-input multi-output circuits there is subsequent reduction in terms of percentage i.e. for **bw** it is about 10%, for **clip** 12.38%, for **con1** 16.6%, for **misex2** 19.2%, for **sqrt8** 28.6%, for **misex3c** 39.5%,for **inc** 42%, for **5xp1** 42%, for **squar5** 50%, for **duke2** 54.7%, for **sao2** 60.4%, for **misex1** 61.7%, and for **b12** 68%. Therefore, the modified Memetic algorithm is better suited for multi-input multi-output (MIMO) VLSI circuits as compared to proposed GA and HGA because for the same set of benchmark circuits MMA is giving better node count.

From Table 7.11 it is inferred that PMX gives more computation time as compared to other two crossover operators *i.e.* cycle and order crossover. There is a trade-off between node count and computation time. As the node count reduces the computation time increases and vice-versa. For PMX crossover, the node count reduces but as trade off computation time increases. For multi-input and single output the computation time is almost similar for all the three crossover operators whereas for MIMO LGSynth93 Benchmark circuits it is comparable showing more computation time for PMX crossover operators and less for the cycle and order crossover operators.

Table 7.11 Comparison of Computation Time for Different Crossover Operators in Modified Memetic algorithm for LGSynth93 Benchmark Circuits

Benchmark Circuits	**Inputs**	**Outputs**	**Proposed MMA with crossover**		
			Order(sec)	Cycle (sec)	PMX (sec)
b12	15	9	0.41	1.03	1.84
sao2	10	4	5.9	5.489	4.185
5xp1	7	10	1.38	1.21	1.1
bw	5	28	0.26	0.3	0.26
con1	7	2	0.24	0.15	0.14
inc	7	9	0.24	0.21	0.2
sqrt8	8	4	2.72	3.56	3.95
squar5	5	8	0.14	0.12	0.12
clip	9	5	73.7	126.7	129.7
xor5	5	1	0.07	0.05	0.06
pdc	16	40	4.35	5.06	10.39
duke2	22	29	8.17	5.31	10.34
misex1	8	7	0.8	0.27	0.15
misex2	25	18	0.17	0.1	0.06
misex3c	14	14	5.19	9.23	4.33
table5	17	15	4.68	1.18	5.32
t481	16	1	10.58	10.04	15.51

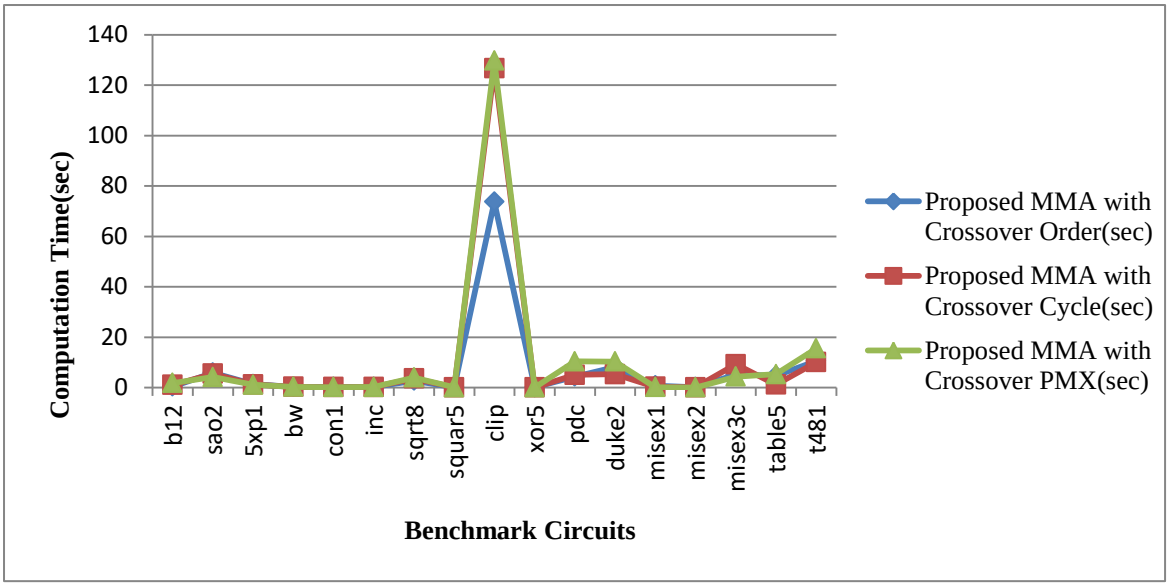


Figure 7.10 Graphical Analysis representing Computation Time for Proposed MMA with Crossover algorithm on LGSynth93 Benchmark Circuits

The next Table 7.12 shows power estimation of Modified Memetic Algorithm for existing MUX based algorithm and Probabilistic algorithm applied on LGSynth93 Benchmark circuits and multi-input adders circuits. There is a drastic reduction in power by applying Probabilistic approach as compared to MUX based technique where power is calculated by multiplying the node count with the power estimated for 2:1 MUX using Synopsys tool. Power is measured in microwatt.

Table 7.12 Comparison of MMA Power results for LGSynth93 benchmark circuits with existing algorithm

Bench-mark Circuits	Order Crossover (mW)		PMX Crossover (mW)		Cycle Crossover Operator(mW)	
	Proposed Probabilistic Technique	Existing MUX Based Technique	Proposed Probabilistic Technique	Existing MUX Based Technique	Proposed Probabilistic Technique	Existing MUX Based Technique
b12	$330.11x10^{-6}$	0.022107	$4.76x10^{-6}$	0.022107	$0.55x10^{-6}$	0.021345
5xp1	$45.78x10^{-6}$	0.038116	45.78×10^{-6}	0.038878	$45.78x10^{-6}$	0.040403
B	$0.015x10^{-6}$	0.080805	0.015×10^{-6}	0.080805	$0.015x10^{-6}$	0.080043
Inc	$41.00x10^{-6}$	0.035829	41.00×10^{-6}	0.035829	$41.00x10^{-6}$	0.035829

xor5	$866.62x10^{-6}$	0.006861	$866.62x10^{-6}$	0.006861	$866.62x10^{-6}$	0.006861
misex2	$0.015x10^{-6}$	0.065559	0.03×10^{-6}	0.060985	$0.015x10^{-6}$	0.064034
t481	$0.13x10^{-12}$	0.053362	2.27×10^{-12}	0.024394	$0.25x10^{-12}$	0.064797
sao2	$0.032x10^{-6}$	0.057936	$0.124x10^{-6}$	0.046501	$0.003x10^{-6}$	0.048026
Clip	$96.88x10^{-12}$	0.065559	$0.066x\ 10^{-12}$	0.070133	$0.0014x10^{-12}$	0.073945

A graphical view of reduction of power dissipation using probabilistic technique [22] in comparison to values calculated using Synopsys tool has been shown in Figure 7.10 and Figure 7.11 Accordingly, it can be observed that in most of the circuits, the results provided by the proposed MMA are better as compared to other evolutionary variable ordering algorithms implemented.

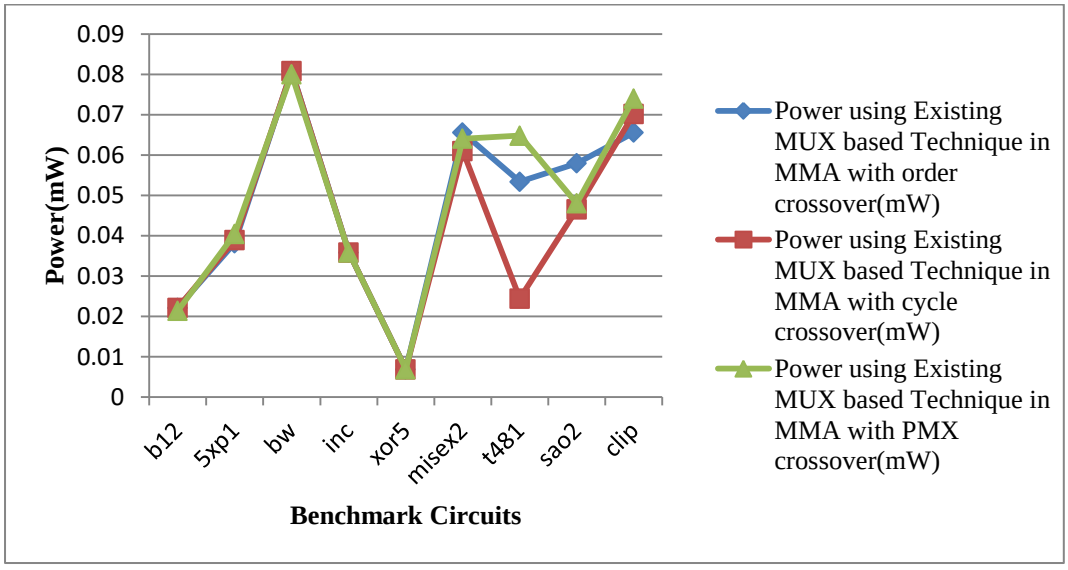


Figure. 7.11 Graphical Analysis representing Power Estimation Proposed MMA with Crossover algorithm on LGSynth93 Benchmark Circuits

CHAPTER 8

Conclusion and Future Scope

8.1 Conclusion

- The proposed algorithms and approaches provide the optimum variable order with minimum number of nodes in minimum iterations and number of runs for LGSynth93 Benchmark Circuits.
- It is ascertained that in comparison with other existing dynamic variable ordering algorithms, the proposed Genetic Algorithm gives minimum node count. When the crossover operators are compared, Partially Mapped Crossover gives minimum nodes, while the Cycle Crossover gives minimum computation time for most of the LGSynth93 Benchmark circuits. The order crossover also gives optimum results. Therefore, the proposed Genetic algorithm is suitable for multi-input multi-output (MIMO) VLSI circuits.
- In the second proposed algorithm *i.e.* Hybridized Genetic Algorithm, the BDD size gets more reduced in terms of node count in comparison with proposed genetic algorithm. When the crossover operators are compared, Partially Mapped Crossover gives minimum nodes, while the Cycle Crossover gives minimum computation time. Therefore, the proposed Hybridized Genetic algorithm is better suited for multi-input multi-output (MIMO) VLSI circuits as compared to proposed genetic algorithms
- In the third proposed algorithm i.e. modified memetic algorithm, the BDD size gets best reduced in terms of node count in comparison with proposed genetic algorithm and hybridized genetic algorithm. When the crossover operators are compared, Partially Mapped Crossover gives minimum nodes, while the Cycle Crossover gives minimum computation time. Therefore, the proposed modified memetic algorithm is best suited for multi-input multi-output (MIMO) VLSI circuits.
- It is perceived that the computation time for Order and Cycle Crossover Operators is less in comparison with that of PMX Crossover operator in case

of Genetic, Hybridized Genetic and Modified Memetic Algorithms for more than 70% of LGSynth93 benchmark circuits.

- In terms of the power estimation, it is observed that that in more than 90% cases, Genetic and Hybridized genetic algorithm based approach using probabilistic analysis has shown lesser power dissipation than the Multiplexer based existing technique. It can be concluded that the power dissipation in case of PMX and Order Crossover Operators is lesser than Cycle Crossover Operator in case of Genetic, Hybridized Genetic and Modified Memetic Algorithms for 80% LGSynth93 benchmark circuits.
- Therefore, it is concluded that the Evolutionary algorithms namely Genetic Algorithm, Hybridized Genetic Algorithm and Modified Memetic Algorithm are well suited for the optimization of area, speed and power in MIMO VLSI circuits.

8.2 Future Scope of Work

- Various other existing Evolutionary Algorithms can be implemented on the LGSynth93 benchmark circuits for the optimization of area, power and speed.
- Performance improvement in terms of memory requirement can be another parameter to be explored in future, apart from achieving low area, low power and high speed.
- BDD-based probabilistic techniques can be implemented for other evolutionary algorithms.

REFERENCES

[1] A.G.Ruzzelli, C. Nicolas, A. Schoofs, and G. M. P. O'Hare, "Real – Time Recognition and Profiling of Appliances through a Single Electircity Sensor," in *7th Annual IEEE Communications Society Conference on Sensor Mesh and Ad Hoc Communications and Networks (SECON)*, 2010.

[2] J. Han and M. Orshansky, "Approximate computing: An emerging paradigm foe energy efficient design," in *18th IEEE European Test Symposium (ETS)*, 2013.

[3] V. Gupta, D. Mohapatra, S. P. Park, A. Raghunathan, and K. Roy, "IMPACT: imprecise adders for low power approximate computing," in *Proceedings of the 17th IEEE/ACM International Symposium on Low Power Electronics and Design*, 2011, pp. 409–414.

[4] A. Sampson, W. Dietl, E. Fortuna, D. Gnanapragasam, L. Ceze, and D. Grossman, "Ener J: Approximate Data Types for Safe and General Low Power Computation," in *Proceedings of the 32nd ACM SIGPLAN Conference on Programming Language Design and Implementation*, 2011, pp. 164–174.

[5] P. Jeavons, D. A. Cohen, and J. Shawe-Taylor, "Generating Binary Sequences for Stochastic Computing," *IEEE Trans. Inf. Theory*, vol. 40, no. 3, 1994.

[6] B. D. Brown and H. C. Card, "Stochastic neural computation. I. Computational Elements," *IEEE Trans. Comput.*, vol. 50, no. 9, pp. 891–905, 2001.

[7] Y.-N. Chang and K. K. Parhi, "Architectures for Digital Filters using Stochastic Computing," in *IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, 2013.

[8] S. Venkataramani, A. Sabne, V. Kozhikkottu, K. Roy, and A. Raghunathan, "No Title," in *Proceedings of the 49th Annual Design Automation Conference (DAC)*, 2012, pp. 796–801.

[9] S. B. Akers, "Binary Decision Diagrams," *IEEE Trans. Comput.*, vol. C-27, no. 6, pp. 509–516, 1978.

[10] R. E. Bryant, "Graph based Algorithms for Boolean Function Manipulations," *IEEE Trans. Comput.*, vol. 35, no. 1, pp. 667–691, 1986.

[11] K. Priyank, "VLSI Logic Test, Validation and Verification, Properties & Applications of Binary Decision Diagrams," *Department of Electrical and Computer Engineering University of Utah, Salt Lake City, UT 84112*, 1997.

[12] Y. Y. Liu, K. H. Wang, T. T. Hwang, and C. L. Liu, "Binary Decision Diagram with Minimum Expected Path Length," *Des. Autom. Test*, pp. 708–712, 2001.

[13] S. J. Friedman and K. J. Supowit, “Finding the optimal Variable Ordering for Binary Decision Diagrams,” *IEEE Trans. Comput.*, vol. 39, no. 3, 1990.

[14] N. Ishiura, H. Sawada, and S. Yajima, “Minimization of Binary Decision Diagrams Based on Exchanges of Variables,” *IEEE*, 1991.

[15] Y. Iguchi, T. Sasao, and M. Matsuura, “Evaluation of Multiple-Output Logic Functions using Decision Diagrams,” in *Asian South Pacific Design Automation Conference*, 2003, pp. 312–315.

[16] M. Fujita, H. Fujisawal, and Y. Matsunaga, “Variable Ordering Algorithms for Ordered Binary Decision Diagrams and Their Evaluation,” *IEEE Trans. Comput. Des.*, vol. 12, no. 1, pp. 6–12, 1993.

[17] H. Fujii, G. Ootomo, and C. Hori, “Interleaving Based Variable Ordering Methods for Ordered Binary Decision Diagrams,” in *Proceedings of IEEE/ACM International Conference on Computer-Aided Design*, 1993, pp. 38–41.

[18] R. Rudell, “Dynamic Variable Ordering for Ordered Binary Decision Diagrams,” in *Proceedings of IEEE/ACM International Conference on Computer-Aided Design*, 1993, pp. 42–47.

[19] E. Felt, G. York, R. Brayton, and A. Sangiovami-Vincentelli, “Dynamic Variable Reordering for BDD Minimization,” in *Proceedings of EURO-Design Automation Conference*, 1993, pp. 130–135.

[20] C. Meinel and F. Somenzi, “Linear Sifting of Decision Diagrams,” *IEEE Trans. Comput. Des. Integr. Circuits Syst.*, vol. 19, no. 5, pp. 521–533, 2000.

[21] W. N. N. Hung, X. Song, E. M. Aboulhamid, and M. A. Driscoll, “BDD Minimization by Scatter Search,” *IEEE Trans. Comput. Des. Integr. Circuits Syst.*, vol. 21, no. 8, pp. 974–979, 2006.

[22] R. Drechsler, B. Becher, and N. Gockel, “A Genetic Algorithm for Variable Ordering of OBDDs,” in *IEEE Proceedings on Computer and Digital Techniques*, 1996, pp. 364–368.

[23] N. Zhuang, M. S. T. Benten, and P. Y. K. Cheung, “Improved Variable Ordering of BDDs with Novel Genetic Algorithm,” in *IEEE International Symposium on Circuits and Systems*, 1996, pp. 414–417.

[24] S. Chaudhary and A. Dutta, “Algorithmic Optimization of BDDs and Performance Evaluation for Multi-level Logic Circuits with Area and Power Trade-offs,” in *Proceedings of Seventeenth International Conference on Electronics, Circuits and Systems*, 2010, pp. 627–630.

[25] M. Takapoo and M. . Ghaznavi-Ghoushchi, “IDGBDD: The Novel use of ID3 to Improve Genetic algorithm in BDD Reordering,” in *International Conference on Electrical Engineering/Electronics Computer Telecomm. and Information Technology*, 2010, pp. 117–121.

[26] W. Mingquan and Y. Haibin, “BDD Minimization Based on Genetic Tabu

Hybrid Strategy," in *Proceedings of the Sixth International Conference on ASIC*, 2005, pp. 948–952.

[27] A. Mitra and S. Chattopadhyay, "Variable Ordering for Shared Binary Decision Diagrams Targeting Node Count and Path Length Optimization using Particle Swarm Technique," *IET Comput. Digit. Tech.*, vol. 6, no. 6, pp. 353–361, 2011.

[28] E. Elbeltagi, T. Hegazy, and D. Grierson, "Comparison Among Five Evolutionary-based Optimization Algorithms," *Int. J. Adv. Eng. Informatics*, pp. 43–53, 2005.

[29] M. P and F. B, "A Genetic local search approach to the quadratic assignment problem," in *Proceedings of Seventh International Conference on Genetic Algorithms*, 1997, pp. 465–472.

[30] L. Lu, Q. Luo, J. Liu, and C. Long, "An Improved Particle Swarm Optimization Algorithm," in *EEE International Conference on Granular Computing*, 2008, pp. 486–490.

[31] M. P. Rojas and C. A. Coello, "A Miriam Pescador Rojas and Carlos A. Coello Coello," in *World Automation Congress 2012*, 2012, pp. 1–6.

[32] W. Lenders and C. Baier, "Genetic Algorithms for the Variable Ordering Problem of Binar Decision Diagrams," pp. 1–20, 2004.

[33] R. Drechsler, M. Kerttu, P. Lindgren, and M. Thornton, "Low-Power Optimization Techniques for BDD Mapped Circuits using Temporal Correlation," *Can. J. Elect. Comput. Eng*, vol. 27, no. 4, 2002.

[34] L. Cheng, D. Chen, and M. D. F. Wong, "DDBDD: Delay-Driven BDD Synthesis for FPGAs," *IEEE Trans. Comput. Aided Des. Integr. Circuits Syst.*, vol. 27, no. 7, 2008.

[35] R. Meolic, "Biddy --- a Multi-Platform Academic BDD Package," *J. Softw.*, vol. 7, no. 6, pp. 1358–1366, 2012.

[36] M. Raseen, S. Deivanai, and K. Thanushkodi, "Effect of Don't Cares on the Size of ROBDD," *Eur. J. Sci. Res.*, vol. 63, no. 2, pp. 192–203, 2011.

[37] A. M. Dixit and H. Singh, "Soft Computing Approach to Crack Detection and Impact Source Identification with Field-Programmable Gate Array Implementation," *Adv. Fuzzy Syst. by Hindawi Publ. Corp.*, 2013.

[38] A. Kumar, K. Dutta, A. Gupta, S. Badyal, and D. Rohan, "Assisting an Architect with Alternative Automated Space Layout Designs using Order Crossover Genetic Algorithm in AutoCAD," in *EEE International Conference on Advances in Mechanical, Industrial, Automation and Management Systems (AMIAMS)*, 2017.

[39] T. S. Abdelgayed, W. G. Morsi, and T. S. Sidhu, "A New Harmony Search Approach for Optimal Wavelets Applied to Fault Classification," *IEEE Trans. Smart Grid*, 2016.

[40] S. Misra, S. K. Dhurandher, M. S. P. G. Obaidat, K. Verma, and P. Narula, “An ant swarm-inspired energy-aware routing protocol for wireless Ad-hoc networks,” *J. Syst. Softw.*, vol. 83, pp. 2188–2199, 2010.

[41] S. K. Dhurandher, S. Misra, M. S. Obaidat, and N. Gupta, “An Ant Colony Optimization Approach for Reputation and Quality-of-Service-based Security in Wireless Sensor Networks,” *J. Secur. Commun. Networks*, vol. 2, pp. 215–224, 2009.

[42] S. Chaudhury and S. Chattopadhyay, “Output Phase Assignment for Multilevel Multi-output Logic Synthesis with Area and Power Trade-offs,” in *Annual IEEE India Conference*, 2006, pp. 1–4.

[43] R. Drechsler, B. Becker, and N. Gockel, “A Genetic Algorithm for Variable Ordering of OBDDs,” *IEEE Proc. Comput. Digit. Tech.*, vol. 143, no. 6, pp. 363–368, 1996.

[44] R. Ebendt, W. Gnther, and R. Drechsler, “An Improved Branch and Bound Algorithm for Exact BDD Minimization,” *IEEE Trans. Comput. Des. Integr. Circuits Syst.*, vol. 22, no. 12, pp. 1657–1663, 2003.

[45] S. Chaudhary and A. Dutta, “Algorithmic Optimization of BDDs and Performance Evaluation for Multi-level Logic Circuits with Area and Power Trade-offs,” *Circuits Syst.*, vol. 2, no. 3, pp. 217–224, 2011.

[46] N. Vemuri, P. Kalla, and R. Tessier, “BDD-Based Logic Synthesis for LUT-Based FPGAs,” *ACM Trans. Des. Autom. Electron Syst.*, vol. 7, no. 4, pp. 501–525, 2003.

[47] K. Muma, D. Chen, Y. Choi, D. Dodds, M. H. Lee, and S.-B. Ko, “Combining ESOP Minimization with BDD-Based Decomposition for Improved FPGA Synthesis,” *Can. J. Electr. Comput. Eng.*, vol. 33, no. 3–4, pp. 177–182, 2008.

[48] C. Yang and M. Ciesielski, “BDS: A BDD-based Logic Optimization System,” *IEEE Trans. Comput. Des. Integr. Circuits Syst.*, vol. 21, no. 7, pp. 866–876, 2002.

[49] D. Chen and J. Cong, “DAOmap: A Depth-Optimal Area Optimization Mapping Algorithm,” in *Proceedings of International Conference on Computer Aided Design*, 2004, pp. 752–759.

[50] J. Y. Lin, D. Chen, and J. Cong, “Optimal Simultaneous Mapping and Clustering for FPGA Delay Optimization,” in *Proceedings of the annual 43rd Design Automation Conference*, 2006, pp. 472–477.

[51] A. C. Ling, J. Zhu, and S. D. Brown, “Scalable Synthesis and Clustering Techniques Using Decision Diagrams,” *IEEE Trans. Comput. Des. Integr. Circuits Syst.*, vol. 27, no. 3, 2008.

[52] J. H. Holland, *Adaption in Natural and Artificial Systems, Ann Arbor,*. MIJ: University of Michigan Press, 1975.

[53] D. E. Goldberg, *Genetic Algorithms in Search, Optimization and Machine*

Learning, Reading. MA : Addison –Wesley, 1989.

[54] C. R. Darwin, *On the Origin of Species by Means of Natural Selection, London: John Murray*. 1859.

[55] L. J. Eshelman and J. D. Schaffer, "Real-coded Genetic Algorithms and Interval Schemata," in *Foundations of Genetic Algorithms 2, San Mateo, CA: Morgan Kaufmann*, 1993, pp. 187–202.

[56] D.E.Goldberg, "Real-coded Genetic Algorithms, Virtual Alphabets, and Blocking," *Complex Syst.*, vol. 5, pp. 139–167, 1991.

[57] K. A. De Jong and J. Sarma, "Generation Gaps Revisited," *Foundations of Genetic Algorithms 2, San Mateo, CA: Morgan Kaufmann*, vol. 2, pp. 19–28, 1993.

[58] P. Winter, "Steiner Problem in Networks: A survey," *Networks*, vol. 17, pp. 129–167, 1987.

[59] J. E. Baker, "Reducing bias and Inefficiency in the Selection Algorithm," in *Proc. Second Int. Conf. Genetic Algorithms Second Int. Conf. Genetic Algorithms*, 1967, pp. 14–21.

[60] D. E. Goldberg and K. Deb, "A Comparative Analysis of Selection Schemes used in Genetic Algorithms," *Found. Genet. Algorithms 2, San Mateo, CA Morgan Kaufmann*, vol. 1, pp. 69–93, 1991.

[61] W. Contributors, "Memetic algorithm," 2018. [Online]. Available: https://en.wikipedia.org/w/index.php?title=Memetic_algorithm&oldid=844503612.

[62] O. Hasancebi and F. Erbatur, "Evaluation of Crossover Techniques in Genetic Algorithm based Optimum Structural Design," *Comput. Struct.*, pp. 435–448, 1999.

[63] R. Dawkins, "The necessity of Darwinism," *New Sci.*, vol. 94, pp. 130–132, 1982.

[64] O. Brudaru, R. Ebendt, and I. Furdu, "Optimizing Variable Ordering of BDDs with Double Hybridized Embryonic Genetic Algorithm," in *12th International Symposium on Symbolic and Numeric Algorithms for Scientific Computing*, 2010.

[65] F. Towhidi, A. H. Lashkari, and R. S. Hosseini, "Binary Decision Diagram (BDD)," in *International Conference on Future Computer and Communication*, 2009, pp. 496–499.

[66] P. Porwik, K. Wrobel, and P. Zaczkowski, "Some Practical Remarks about Binary Decision Diagram Size Reduction," *IEICE Electron. Express*, vol. 3, no. 3, pp. 51–57, 2006.

[67] B. Bontoux, C. Artigues, and D. Feillet, "A Memetic Algorithm with a Large Neighborhood Crossover Operator for the Generalized Traveling Salesman Problem," *Comput. Oper. Res.*, pp. 1844–1852, 2009.

[68] K. Roy and S. Prasad, *Low-Power CMOS VLSI Circuit Design*. Hoboken, N.J.: Wiley Interscience, 2000.

[69] R. Drechsler, M. Kerttu, P. Lindgren, and M. Thornton, “Low Power Optimization Techniques for BDD Mapped Circuits Using Temporal Correlation,” *Can. J. ECE*, vol. 27, no. 4, pp. 159–164, 2002.

[70] M. Fujita, Y. Matsunaga, and T. Kakuda, “On Variable Ordering of Binary Decision Diagrams for the Application of Multi-Level Logic Synthesis,” in *Proceedings of the European Conference on Design automation*, 1991, pp. 50–54.

LIST OF PUBLICATIONS

Papers published in Peer Reviewed Journals

1. Manu Bansal andAlpanaAgarwal "A Genetic Algorithm for ordering and reduction of BDDs using Crossover Operators for MIMO VLSI Circuits."*Research Cell: An International Journal Of Engineering Sciences*, Vol.24, July 2017.

2. Manu Bansal andAlpanaAgarwal "Low Power Optimization Technique and a genetic minimization algorithm for variable ordering of BDD mapped VLSI Circuits." *International Journal of Electrical & Computer Sciences, IJECS-IJENS,* Vol. 17, No. 5, October 2017.

Papers published in International Conference(s)

1. Manu Bansal andAlpanaAgarwal, "Genetic algorithm for ordering and reduction of BDDs for MIMO circuits", *Proceedings of IEEE Third International Conference on Innovative Computing Technology*, organized at London, UK, page no.411-414, from Aug 29-31, 2013.

Papers published in National Conference(s)

1. Manu Bansal andAlpanaAgarwal, "Ordering and Reduction of BDDs for multi-input adders Using Evolutionary Algorithm" *International Conference on Advanced Electronic Systems* (ICAES-2013) organized at CSIR-CEERI, Pilani from 21-23 September, 2013.