FSM STATE ASSIGNMENT FOR AREA, POWER AND TESTABILITY USING ITERATIVE ALGORITHMS

by

FAISAL NAWAZ KHAN

A Thesis Presented to the
DEANSHIP OF GRADUATE STUDIES

In Partial Fulfillment of the Requirements
for the Degree

MASTER OF SCIENCE

IN

Computer Engineering

KING FAHD UNIVERSITY
OF PETROLEUM AND MINERALS

Dhahran, Saudi Arabia

May 2005

KING FAHD UNIVERSITY OF PETROLEUM & MINERALS
DHAHRAN 31261, SAUDI ARABIA

DEANSHIP OF GRADUATE STUDIES

This thesis, written by FAISAL NAWAZ KHAN under the direction of his thesis advisor and approved by his thesis committee, has been presented to and accepted by the Dean of Graduate Studies, in partial fulfillment of the requirements for the degree of **MASTER OF SCIENCE IN COMPUTER ENGINEERING**.

Thesis Committee

_____
Dr. El – Maleh, Aiman (Chairman)

_____
Dr. Sait, Sadiq Mohammed (Member)

_____
Department Chairman

_____
Dr. Naseer, A.R. (Member)

_____
Dean of Graduate Studies

_____
Date

Contents

List of Tables

List of Figures

THESIS ABSTRACT

Name: FAISAL NAWAZ KHAN

Title: FSM STATE ASSIGNMENT FOR AREA, POWER AND TESTABILITY USING ITERATIVE ALGORITHMS

Major Field: COMPUTER ENGINEERING

Date of Degree: MAY 2005

The rapid increase in complexity of VLSI circuits along with their proliferation in new domains have posed new challenges to the VLSI CAD industry. Mobile devices have given rise to new requirements such as low power in addition to conventional area (size) and Performance (timing) goals. The increase in chip complexity (size) has further increased the complexity of VLSI devices. With ever shrinking design to market windows, there has always been a strong need to develop synthesis tools to address such multi- objectives efficiently. One central problem in the synthesis of digital systems is controller synthesis which can be accomplished via FSMs, microprograms, etc. The complexity of FSM implementation depends on its state assignment. State assignment of FSMs for efficient area implementation alone is an NP-hard problem. The problem is further involved if we consider additional objectives such as low-power and ease of testability. Non-deterministic evolutionary heuristics such as Genetic Algorithms and Tabu Search have shown to yield good results in solving such multiobjective hard combinatorial optimization problems in other areas of design automation.

In this work new ideas for FSM state assignment to address area, power and testability as a single and multiobjective optimization are evaluated. New methods in estimating area, power and testability are suggested and their efficiencies compared with previously proposed measures. The work uses non-deterministic heuristics, Genetic and Tabu Search. Another important focus of the work is efficient design of the non-deterministic heuristics for ease of exploration of the search space.

MASTER OF SCIENCE DEGREE

King Fahd University of Petroleum and Minerals, Dhahran.
May 2005

Chapter 1

Introduction and Motivation

1.1 Introduction

Technological advancements in Very Large Scale Integration (VLSI) have empowered the industry to integrate millions of transistors on a single chip. The contemporary approach adopted to address design of devices with high complexity is by following concepts of structured designing and design abstraction [1]. Abstraction is used to utilise the efforts of designers at higher levels. This allows fast initial prototyping with refinements left to be added at lower stages using detailed circuit information. Typical levels of abstraction, together with their corresponding functionalities, are illustrated in Figure 1.1. Computer Aided Design (CAD) tools automate the VLSI design process at all levels of design abstraction.

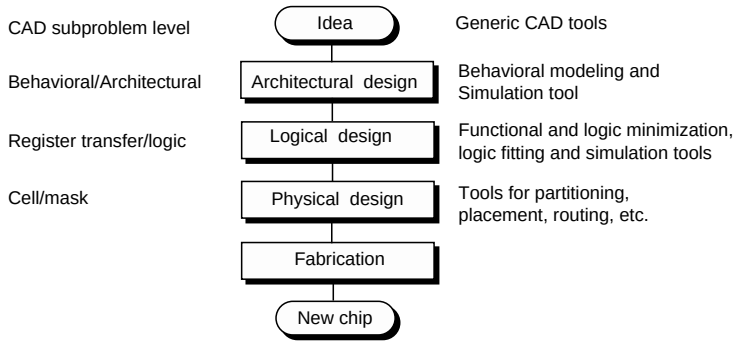


Figure 1.1: Levels of abstraction and corresponding design steps.

With the rapid increase in system functionality, new paradigms such as mobile computing have emerged. Mobile computing has added a new facet

of power efficiency in the complexity of VLSI design [2]. At the same time, the increasingly complex VLSI devices are proving more and more difficult to test. Efficient testing is no longer the sole responsibility of test engineers and the focus is now on better design strategies to make the device more easily testable [3]. Thus, testability of a VLSI chip adds another objective to the increasingly complex task of designing VLSI circuits.

1.2 Motivation

The complexity of today's digital systems is tackled by partitioning and automating the design process using CAD tools. Digital Systems are broadly composed of two subcomponents namely a controller and a datapath. The datapath performs all the arithmetic and logical operations required on the data. The controller is responsible for controlling the sequence of operations on the datapath and is generally implemented as a finite state machine (FSM).

The increasing complexity in VLSI systems along with increasing number of issues is posing ever increasing challenge for CAD industry to automate the tasks for a VLSI designer. Literature reports a wealth of research to address the issues (see chapter-2). Most of the heuristics reported address the problem of FSM state assignment for either a single or dual objectives. These heuristics are discussed in detail in chapter-2. However there is a lack of research that targets all the three objectives simultaneously. As will be shown later, there exists a strong interaction among area, power and testability issues. This correlation makes it more necessary to address the three objectives together.

Traditionally, synthesis of FSMs is targeted for area minimization which itself is an NP-hard problem [4]. The degree of diversity in reported results suggests that there is a strong need to further investigate the problem to come up with measures that have better correlation with the objectives. The three objectives focus of this work, area, power and testability, may also not be in total harmony with one another. This makes the problem of FSM state-assignment for area, power and testability, even more challenging.

Non-deterministic evolutionary heuristics/meta-heuristics like Genetic Algorithms and Tabu Search have shown good results in solving various combinatorial optimization problems [5]. These heuristics try to optimize user defined goals of the problem encapsulated within a cost function. The quality of the solution thus depends on how closely the problem is modeled using the cost measures.

There have been several attempts in using genetic algorithm (GA) for FSM

state assignment problem. However there has been no study that compares the design alternatives to optimize the performance of GA for the problem. Similarly, Tabu Search (TS) has been more promising than GA in solving other combinatorial optimization problems. However, the use of Tabu Search in solving the problem of FSM state assignment is yet to be addressed.

1.3 FSM State Assignment Problem

An FSM M can be formally defined as a 5-tuple $M = (S, I, O, T, \delta)$ where S represents the finite state space, I represents the finite input space and O the finite output space, $\delta : I \text{x} S \rightarrow S$ is the next state function and T is the transition relation defined as I x $S \rightarrow O$ (for a Mealy machine) or $T : S \rightarrow O$ (for a Moore machine).

State assignment involves an injective mapping f: S $\rightarrow B^n$ where n is the code length ($n \geq \lceil \log_2 |S| \rceil$) and B^n is an n-dimensional boolean space, a boolean hypercube.

1.4 Thesis Objective

The focus of this work is to address the NP-hard problem of FSM state assignment such that area and power are reduced while increasing the testability of the circuit synthesized by the assignment. Minimum code-length is considered as constraint in this work.

Individual objectives are initially explored using existing heuristics reported in literature and optimized using new measures. The individual objectives are then combined using various integrating mechanisms (discussed later) to solve multi-objective combinatorial optimization problem. Non-deterministic evolutionary heuristics like Genetic Algorithm and Tabu Search are utilized for search space exploration. Thus another objective of this work is the design of the exploration-heuristics for efficiently exploring FSM state assignment search space.

1.5 Contributions

The work presents the results of investigations related to the objectives discussed in the previous section. In particular, the main contributions can be summarized as follows:

- The work proposes use of Expand-function as an efficient measure for FSM state assignment for multilevel area minimization.
- A new power reduction strategy is explored that combines the traditional minimum weighted hamming distance approach with area estimate obtained using Expand cover. The objective of the technique is to minimize the fanout of frequently switched states.
- A new loop detection algorithm is proposed.
- The work demonstrates initializability detection of sequential elements as an important factor affecting testability of a circuit, along with traditional depth and number of loops.
- A new method of initializability detection using Expand cover is proposed.

1.6 Organization of the Thesis

The rest of the thesis is organized as follows. In chapter-2, literature survey is presented. This chapter builds up the basics in FSM state assignment for area, power and testability and covers various methodologies reported in literature concerning the objectives of this work.

Chapter-3 formally defines the problem and discusses some new measures that are developed in this work. This is followed with a detailed discussion on the design of iterative heuristics for the current problem in chapter-4.

In chapter-5, experimental setup and results are presented using the measures discussed in chapter-3 and compared with those discussed in chapter-2. Implementation details for Genetic Algorithm and Tabu search are also discussed and their relative performance is compared in solving single as well as multiobjective optimization problem. The results thus obtained are also compared and contrasted with those reported in the literature. This thesis ends with conclusion and some future directions in chapter-6.

Chapter 2

Literature Review

In this chapter, a detailed survey of heuristics for FSM synthesis for area, power and testability is reported. FSM synthesis for area is usually targeted independently for two-level and multi-level realizations. FSM synthesis for power involves calculating transition probabilities in between states and to reduce switching involved. FSM synthesis for higher testability is usually targeted by reducing the sequential depth and the number of loops in the synthesized circuit. Description of various cost models to model the optimization objectives at a higher level are also detailed.

Review for FSM synthesis strategies is followed by a survey of iterative heuristics, genetic algorithm and tabu search. These heuristics have shown good results in optimizing hard combinatorial problems.

2.1 Finite State Machine State Assignment

State assignment in FSMs [4] is one of the main problems in the synthesis of sequential machines. The complexity of FSMs lie in their combinational circuit that heavily depends on chosen state assignment or encoding for its sequential elements. Similarly power dissipation and testability of the FSM are also functions of the state assignment. Thus, depending on the requirements, the assignment of states can be subject to different constraints. Gaining insight into the problem of assigning state codes is thus useful in coming up with solutions which will lead to structures and complexity that will satisfy the required objectives and constraints. This section thus proceeds with a discussion on relevant mathematics concerning state assignment problem.

2.1.1 Encoding and Partitioning

The state assignment problem of an FSM can be viewed as a coding problem or as a partitioning problem [6, 7, 4]. The coding problem requires each state to be assigned a unique binary pattern. From the partitioning point of view, each state variable, y_i (one of the bits of the memory part of FSM), partitions the assigned states into two sets. All states in one set are those for which y_i is 1, and those in the other set for which y_i is 0.

Therefore a partition on a set S of states is a collection of disjoint subsets whose set union is S. The disjoint subsets are called the *blocks* of the partition. A partition is called an m-block partition if the number of blocks in it are m.

The partition induced by a state variable y_i is represented with the Greek symbol Tau, $\tau(y_i)$. As an example, consider a machine M with four states (A,B,C,D) and a single input (x) as given in Table 2.1.1. The above state

PS	NS	
	x=0	x=1
A	A	D
B	A	C
C	C	B
D	C	A

Table 2.1: State Machine - 1

machine with a state assignment is shown in Table 2.2.

y_1y_2	Y_1Y_2	
	x=0	x=1
A $\rightarrow$ 00	00	10
B $\rightarrow$ 01	00	11
C $\rightarrow$ 11	11	01
D $\rightarrow$ 10	11	00

Table 2.2: A sample encoding for State Machine - 1

In the assignment used in Table 2.2, $y_1 = 0$ for states A and B, and $y_1 = 1$ for states C and D. Therefore, y_1 induces a 2-block partition $\tau(y_1) = (\overline{AB}; \overline{CD})$. Similarly, y_2 induces another 2-block partition $\tau(y_2) = (\overline{AD}; \overline{BC})$, on the states of machine M.

If every state of the state machine is assigned a unique code then the product of all the partitions is a partition that has as many blocks as the number of states. We call such a partition as a *zero partition* represented by $\pi(0)$. Mathematically

$$\prod_{i=1}^{k} \tau(y_i) = \pi(0) \tag{2.1}$$

where k is number of partitions (which is also the number of state variables).

For example, the product of the partitions induced by coding of Table 2.2, $\tau(y_1)$ and $\tau(y_2)$ is given as

$$\tau(y_1) \cdot \tau(y_2) = (\overline{A}; \overline{B}; \overline{C}; \overline{D}) = \pi(0) \tag{2.2}$$

where the dot operator $(\cdot)$ refers to the intersection operation on the states of blocks in the individual partitions.

The problem of state assignment is to find a set of partitions such that (2.1) is true.

Closed Partition

A partition is said to be closed if for any two states S_i and S_j which are in the same block, and for any input I_k, the next states denoted by $\mathrm{I}_k.S_i$ and $\mathrm{I}_k.S_j$ are in a common block of the partition. This condition must be true for all pairs of states in every block. Such a partition is said to be closed and is represented by π. For example, partition $\tau(y_1)$ in Table-2.2 is a closed partition. A closed partition is a special form of a partition in which the next block can be uniquely determined from the knowledge of present block and inputs. For example, suppose that r state variables are assigned to a closed partition, where $r = \lceil \log_2 |\pi| \rceil$ ($|\pi|$ is the number of blocks in a closed partition) among k state variables of the sequential machine, where $k = \lceil \log_2(n) \rceil$, n being the number of states. Then, according to the definition of closed partition, the r state variables are independent of the remaining $k - r$ state variables. Closed partition is thus referred to as zero-dependency condition. In the above example, y_1 is independent of y_2. The equation for Y_1 is

$$Y_1 = \overline{X} Y_1 + X \overline{Y_1} \tag{2.3}$$

Note that the partition $\tau(y_2)$ is not a closed partition

Parallel and Serial Decompositions

The presence of closed partition indicates that some of the state variables can be independently determined irrespective of the other state variables. Thus, if we can find a set of closed partitions such that Condition (2.1) above is satisfied, then the machine can be decomposed into parallel sub-machines, equal to the number of closed partitions in the set, operating independently. Such a decomposition is referred to as parallel decomposition. Mathematically

$$\pi(1).\pi(2)\cdots\pi(k) = \pi(0) \tag{2.4}$$

However, if such a set of closed partitions could not be found, we need to find a partition denoted by T such that Condition (2.1) can be satisfied, i.e.,

$$\pi(1)\cdot\pi(2)\cdots\pi(v)\cdot T = \pi(0); \qquad (v < k) \tag{2.5}$$

In such a case, the partitions $\pi(1)$ to $\pi(v)$ are still closed and so self-dependent. However, the partition T is not closed and so is dependent on state variables other than those assigned to itself. This yields a serial decomposition of a state machine in which independent subsets of the state machine feed the dependencies required for dependent subset of the machine.

Partition Pairs

The structure of sequential machines is much more complicated than a bunch of parallel or serially connected sub-machines. There are sub-machines that are cross dependent. The concept of partition-pairs helps analyze such dependencies.

A partition pair *(T, T')* on the states of a sequential machine **M** is an ordered pair of partitions such that, if S_i and S_j are in the same block of T, then for every input I_k in I, $I_k.S_i$ and $I_k.S_j$ are in the same block of T'. The partition T is called the predecessor partition and T' the successor partition.

Consider a state machine with a state assignment shown in Table-2.3. Partitions induced by state variables, y_2 and y_3 are given as

$$\tau(y_2) = \tau_1 = (\ \overline{A, E};\ \overline{B, C, D, F}) \tag{2.6}$$

$$\tau(y_3) = \tau_2 = (\ \overline{A, C, D, E};\ \overline{B, F}) \tag{2.7}$$

Clearly, (τ_1, τ_2) form a partition pair since the next state at any input for a pair of states in a block in τ_1 lie in some block of τ_2. τ_1 is said to be

PS	$y_1y_2y_3$	NS x1x2			
		00	01	10	11
A	000	A	C	D	F
B	011	C	B	F	E
C	010	A	B	F	D
D	110	E	F	B	C
E	100	E	D	C	B
F	111	D	F	B	A

Table 2.3: State Machine - 2

predecessor partition and τ_2 the successor. Thus, to uniquely determine the next block in the successor partition, one needs to know the present block in the predecessor partition along with inputs. That is to say that the successor partition is dependent on the information of the state variables that induce the predecessor partition. Thus, a partition pair can be thought of as one (or single) dependency condition.

P-Dependency Condition

A P-dependency condition, where P is greater than one, can be derived in a similar manner. This requires the computation of what is known as *Mm-pairs* [4].

A partition *M(T')* is the summation (union) of all partitions T_i such that *(T_i, T')* is a partition pair. Thus, *M(T')* is the largest partition, i.e., a partition containing the biggest blocks whose successor blocks are contained in *T'*. Similarly, a partition *m(T)* is the product (intersection) of all partitions $T_{i'}$ such that *(T, $T_{i'}$)* is a partition pair; where *m(T)* is the smallest partition, i.e., a partition containing the smallest blocks that can be the successors of the blocks of *T*.

Consider again the state machine of Table-2.3. The smallest partition that can be implied by states A and C may be called as τ_{AC}. Such a partition includes states A and C together in a block (AC) and leaves all other states in separate blocks, i.e.,

$$\tau_{AC}(\overline{A,B};\ \overline{B};\ \overline{D};\ \overline{E};\ \overline{F}) \tag{2.8}$$

then a set of partition pairs (τ_{AC}, τ_x) and (τ_{AC}, τ_y) can be given as in

equations-2.9 and 2.10 respectively.

$$(\tau_{AC}, \tau_x) = ((\overline{A,C};\ \overline{B};\ \overline{D};\ \overline{E};\ \overline{F}), (\overline{A};\ \overline{B,C};\ \overline{D,F};\ \overline{E})) \tag{2.9}$$

$$(\tau_{AC}, \tau_y) = ((\overline{A,C};\ \overline{B};\ \overline{D};\ \overline{E};\ \overline{F}), (\ \overline{A,B,C};\ \overline{D,F};\ \overline{E})) \tag{2.10}$$

Though both τ_x and τ_y contain the successor blocks for partition τ_{AC} but partition τ_x is said to contain more information. This is because there are lesser number of possible states in successor blocks, i.e. more information about possible next state(s) from a given predecessor block. Thus there can exist a number of successor partitions containing different degrees of information.

As described perviously, the smallest successor partition for a partition T, i.e. a partition having highest number of blocks or smallest block sizes, is given by *m(T)*. Hence, *m(T)* describes the largest amount of information that can be obtained from T regarding the next state of machine-M. In the example partition pairs above, τ_x is actually $m(\tau_{AC})$. A formal procedure for evaluating *m(T)* will be described shortly.

The P-dependency condition states that if the next-state variable Y_i can be computed from the external inputs and a subset P_i of the state variables, then the product of partitions induced by the subset P_i should be contained within M of the partition induced by y_i. Mathematically

$$\prod_{y_j \in P_i} \tau(y_j) \subseteq M[\tau(y_i)] \tag{2.11}$$

where $\tau(y_k)$ represents the partition induced by variable y_k. The product is taken over all $\tau(y_j)$, such that y_j is contained in the subset P_i. The subset is started with minimum number of variables and gradually expanded until the condition is met. The satisfaction of condition means that Y_i can be derived using the variables used in the subset. Of course, the condition will remain satisfied if the subset is further expanded but we are only interested in minimum number of support variables.

The P-dependency condition is also referred to as information flow inequality [4]. The condition can be efficiently used to find the number of dependencies of state variables. Such a use of the inequality will next be explained using an example.

An Example

Consider the state machine whose state table is given in Table-2.4. We begin by finding smallest partitions implied by pairs of states. Let τ_{AB} be partition that includes a block (AB) and leaves all other states in separate blocks (τ_{AB} $=(\overline{AB};\overline{C};\overline{D};\overline{E})$. Then, by definition, the smallest such partition containing the block implied by τ_{AB} is $m(\tau_{AB})$, which can be determined by looking at the successive or next states of states in τ_{AB}. In the present case,

PS	NS				z
	I_0	I_1	I_3	I_2	
A	C	A	D	B	0
B	E	C	B	D	0
C	C	D	C	E	0
D	E	A	D	B	0
E	E	D	C	E	1

Table 2.4: State Machine - 3

$$m(\tau_{AB}) = \{\ \overline{A,C,E};\overline{B,D}\ \} = \tau_1'$$

Clearly, $(\tau_{AB}, m(\tau_{AB}))$ is a partition pair.

Similarly, the rest of the smallest partitions implied by other pairs of states can be found as follows.

$$\begin{gathered} m(\tau_{AC}) = m(\tau_{DE}) = (\overline{A,C,D};\overline{B,E}) = \tau_2' \\ m(\tau_{AD}) = m(\tau_{CE}) = (\overline{A};\overline{B};\overline{C,E};\overline{D}) = \tau_3' \\ m(\tau_{AE}) = m(\tau_{CD}) = (\overline{A,B,C,D,E}) = \pi(I) \\ m(\tau_{BC}) = m(\tau_{BE}) = (\overline{A};\overline{B,C,D,E}) = \tau_4' \\ m(\tau_{BD}) = (\overline{A,C};\overline{B,D};\overline{E}) = \tau_4' \end{gathered}$$

Let the three state variables needed to encode the 8-states be $y1$, $y2$ and $y3$ and their partitions represented as τ_{y1}, τ_{y2}, τ_{y3} respectively. Then the problem of state assignment is to encode $y1$, $y2$ and $y3$ such that

$$\tau_{y1}.\tau_{y2}.\tau_{y3} = \pi(0)$$

One such state assignment can be

$$\tau_{y1} = (\overline{A, C, E}; \overline{B, D})$$
$$\tau_{y2} = (\overline{A, B, D}; \overline{C, E})$$
$$\tau_{y3} = (\overline{A, C, D}; \overline{B, E})$$

The corresponding M of the above partitions are found out as follows

$$M(\tau_{y1}) = \tau_{AB} + \tau_{AD} + \tau_{CE} + \tau_{BD} = (\overline{A, B, D}; \overline{C, E})$$
$$M(\tau_{y2}) = \tau_{AD} + \tau_{CE} = (\overline{A, D}; \overline{B}; \overline{C, E})$$
$$M(\tau_{y3}) = \tau_{AC} + \tau_{DE} = (\overline{A, C}; \overline{B}; \overline{D, E})$$

where the operator + is the union of two partitions defined as union of every two blocks in the two partitions provided that the intersection of the two blocks is not empty.

We can now use information flow inequality 2.11 to find out dependencies of state variables for the given state assignment. The inequality states that dependency of a state variable inducing partition τ_{yi} is equal to the smallest subset of the product of partitions $\tau_{y1}\tau_{y2}\tau_{y3}$ that is lesser or equal to $M(\tau_{yi})$. Thus, we see that

$$\tau_{y2} = M(\tau_{y1})$$
$$\tau_{y2}.\tau_{y3} \subset M(\tau_{y2})$$
$$\tau_{y1}.\tau_{y3} \subset M(\tau_{y3})$$

Consequently, Y_1 is dependent on y_2 while Y_2 depends on the information supplied by y_2 and y_3. Similarly, Y_3 receives its inputs from y_1 and y_3. This can be translated as

$$Y_1 = f_1(Inputs, y_2)$$
$$Y_2 = f_2(Inputs, y_2, y_3)$$
$$Y_3 = f_3(Inputs, y_1, y_3)$$

2.2 FSM Encoding for Area

The encoding for a finite state machine (FSM) determines its combinational component. The number of storage bits n_b used to store the state assignment also affects the encoding and so the FSM's complexity. The area of an FSM is also a function of the type of flip-flop being employed. Encoding for finite state machines has traditionally been targeted for reducing the complexity of its combinational part. A good survey of FSM encoding for area can be found in [8].

The use of D-type flip-flops is most prevalent in VLSI circuits today. This work will also be implicitly using D-type flip-flops for storing the finite state machine's state assignment.

The minimum number of state variables needed for state assignment is given as

$$r_0 = \lceil \log_2(s) \rceil \tag{2.12}$$

where s is equal to the number of states of the FSM.

An assignment using the minimum number of state variables has the benefit of using the minimum number of storage devices. However, with such an assignment, there is a potential of reduced flexibility in satisfying the number of encoding constraints (discussed later). The problem is further investigated in [9, 10, 11, 12, 13, 14]

Even if we consider minimal state assignments with D type flip-flop, the number of possible combinations is exhaustively large [15].

$$N = \frac{2^{n_b}!}{(2^{n_b} - s)!} \tag{2.13}$$

Thus, exhaustive evaluation is not feasible and heuristics are generally employed to tackle the problem of FSM encoding.

Logic minimization aims to optimize the combinational logic of an FSM. This in turn depends on the degree of freedom provided by an efficient FSM encoding. A good encoding can help the logic minimizer to achieve a better realization in terms of logic cost. Logic minimizers employ different heuristics for two-level and multilevel circuits as their cost measures differ.

A two level implementation realizes a logic function as a sum of product terms. The circuit complexity of such a representation is related to the number of inputs, outputs, number of product terms and number of variables utilized in a product term, i.e. the number of literals.

The simplest way to encode an FSM is by assigning 1-hot state codes. In 1-hot encoding for a state, the corresponding code bit for a state is set to 1 and all others to 0. Thus, 1-hot encoding is a case of non-minimal state assignment such that the number of variables required is equal to the number of states. It is further noticed [16, 17] that such an encoding is poor to minimizing the size in sum of products representation.

An objective of state encoding could be to reduce dependencies among states [18, 19]. The rationale is that by having dependencies reduced, literal count will decrease and so will interconnect. However, reduced dependencies

correlate weakly with the minimality of sum of products representation.

The complexity of a two level realization can be reduced by using mechanisms such as implicant merging, code covering and disjunctive coding [8]. The idea behind ***Implicant Merging*** (See Table-2.5) is to assign adjacent codes to states that produce either same next-state or output or both at similar input conditions. This yields bigger cubes while doing Karnaugh minimization, and results in a simpler final expression. Implicant merging requires adjacency constraints to be met by the state assignment algorithm. ***Code Covering*** involves a code word of a state covering a code word of some other state(s), i.e. all the bit positions for which the second code word is 1, correspond to 1 in the first code word. An example utilizing code covering is illustrated in Table-2.6. Assume that S_1 is encoded with 110 and S_2 with 100. In this case, the input condition (s, 001) can be treated as don't care condition for the next state S_2, reducing the cover cardinality from three to two. Covering constraints produce covering codewords. Reducing cover cardinality using ***Disjunctive Coding*** is illustrated in Table-2.7. Disjunctive constraints require that the disjunction of state codes is equal to some other state code. In the example shown, the states are encoded such that the code for S_2 is the disjunction of the state codes for S_1 and S_3. As such, the second implicant with the input field 101 gets contained in other input conditions and thus is completely saved.

PS	I	NS	z
S_1	i	S	o
S_2	i	S	o
S_3	i	S	o
0--	i	S	o

Table 2.5: Implicant Merging

The major difficulty for 2-level realization of an FSM is the simultaneous consideration of all the types of constraints [14]. In general, it is not possible to satisfy all the coding conditions with a code using the minimum number of bits r_0. By increasing the number of code bits to $r > r_0$, more coding constraints can be satisfied. The increase in the number of storage elements and state signals to be generated has to be justified against the potential of reducing combinational logic by satisfying additional coding constraints.

PS	I	NS	z
S	001	S_1	o
S	000	S_2	o
S	01-	S_2	o
S	001	110	o
S	0--	100	o

Table 2.6: Code Covering

PS	I	NS	z
S	001	S_1	o
S	101	S_2	o
S	111	S_3	o
S	-01	100	o
S	1-1	010	o

Table 2.7: Disjunctive Coding

The problem with many approaches to two-level assignment is that no exact predictions are possible, as to how the satisfaction of coding conditions affects the complexity of the resulting combinational logic, since the different coding conditions interact with each other in a complex way. The application of coding constraints and finding out their effect would be excessively costly as it would require a huge number of logic minimizations. To mitigate this problem, [11] proposed an elegant solution of ***symbolic minimization***. By using symbolic minimization techniques, it is possible to optimize the function independently of the encoding and determine the codes at a later time. This requires performing the minimization at the symbolic level, before the encoding.

In contrast to two level circuits, multiple level circuits provide much more degree of freedom in optimizing combinational network and satisfying coding constraints. This is because of the flexibility provided due to operations such as common subexpression extraction and factorization. Unfortunately, it comes with an increase in the difficulty of modeling and optimizing the multilevel network themselves.

The complexity measure for multilevel circuits is the encoding length and the number of literals in the optimized logic network. Since encoding length is mostly taken constant, literal saving by extracting common subexpressions has been the focus of most of the work done for multilevel FSM optimization. This involves finding the state pairs which when encoded carefully can result in extracting common subexpressions. In contrast to two level circuits, state pairs in multilevel implementations do not necessarily have to be given adjacent codes. If two states have n state bits in common, the combination of the two states result in a common subexpression with n literals. To identify the states to assign close codes, two heuristics proposed by [20, 21] standout. The first called fanout oriented, tries to assign closer codes to the states that have same next state transition. The rationale is to maximize the size of common cube by assigning closer codes (lesser hamming distance) to such states. The second approach is referred as fanin oriented in which state pairs with incoming transitions from the same states are given high weights for closer code assignment. Here the motivation is to maximize the frequency of common cubes in the encoded next state function. The schemes are improved upon in [22]. Rules for detecting potential common cubes and formulae for more precise evaluation of literal savings have been proposed in [23]

There have been a few attempts of utilizing genetic algorithm for solving state assignment problem [24, 25]. Almaini et al [24] utilize ESPRESSO tool in SIS for their cost calculation which though being accurate is computationally

infeasible. Amaral et al [25] used a cost function proposed by Armstrong [26]. The cost model tries to combine the properties of fanin and fanout oriented algorithms. The contribution in the above works is in the design of genetic algorithm for state assignment problem. However, the authors did not try to take advantage of having the state codes in cost function computation.

2.2.1 Jedi

In Jedi [22], the encoding affinity cost is modeled as a function of how many times a pair of states are represented in next state and output functions. The cost function of Jedi is given in equation-2.14.

$$J^P_{k,l} = \sum_{i=1}^{m_0}(P^o_{k,i} + P^o_{l,i}) + \frac{n_E}{2}\sum_{i=1}^{n_s}(P^s_{k,i} + P^s_{l,i}) \tag{2.14}$$

where,
$P^o_{k,i}$ is number of times state k is represented in output i,
$P^s_{k,i}$ is number of times state k is represented in state i,
m_o is the number of outputs,
n_s is the number of states,
n_E is the number of encoding bits.

For example, consider the state machine in Table-2.4. The next state equations for states A-E are given as.

$$\begin{aligned}
A &= A.I_1 + D.I_1 \\
B &= A.I_2 + B.I_3 + D.I_2 \\
C &= A.I_0 + B.I_1 + C.I_0 + C.I_3 + E.I_3 \\
D &= A.I_3 + B.I_2 + C.I_1 + D.I_3 + E.I_1 \\
E &= B.I_0 + C.I_2 + D.I_0 + E.I_0 + E.I2
\end{aligned}$$

and the output equation is given as

$$O_0 = E$$

Here, $P^s_{C,D}$ is the number of times state C is present in next-state equation of state D which is equal to one. Similarly, $P^s_{E,E}$ is two. $P^o_{C,0_0} = 0$ and $P^o_{E,0_0} = 1$

2.2.2 Mustang

In Mustang [21], the authors observed that if $P^{s/o}_{k,i} = 50$ and $P^{s/o}_{l,i} = 2$, states k and l will have less common cubes due to the state assignment than if they

were $P^{s/o}_{k,i} = 26$ and $P^{s/o}_{l,i} = 26$, even though the sums are the same. They thus proposed the use of multiplication in place of addition to represent encoding affinity. The cost function of Mustang is given in equation-2.15

$$M^P_{k,l} = \sum_{i=1}^{m_0} (P^o_{k,i} * P^o_{l,i}) + \frac{n_E}{2} \sum_{i=1}^{n_s} (P^s_{k,i} * P^s_{l,i}) \tag{2.15}$$

Jedi and Mustang both try to reduce the Hamming distance between highly recurring states in the next state functions. The flip-flop equations employ some of these next-state equations depending on state assignment. Thus, Jedi and Mustang encodings rely on increased probability of states occurring together if they are frequently occurring in next-state functions and try to minimize hamming distance between them.

It is also possible that two states, though highly recurrent in next state functions, do not appear together in the flip-flop equations. This is to say that either $P_{k,i}$ or $P_{l,i}$ in output or next-state terms is zero. Jedi, because of summation, can give affinity-weight to such a pair of states. However, this situation is more efficiently handled in Mustang by the use of multiplication operation. By multiplying recurrences of pair of states in a next state equation, Mustang guarantees to give weight to only those pair of states that occur together in a next state and consequently in flip-flop functions.

2.2.3 Armstrong

Armstrong defined adjacency cost by combining fanout and fanin based approaches. The cost function describing the desired adjacency is given by equation-2.16

$$\begin{aligned}
A^P_{i,j} = R_1 \sum_{l=1}^{s-1} \alpha_{li}\alpha_{lj}\delta_{ij} & \qquad StateFanin \\
+R_2 \sum_{a=0}^{c-1} \sum_{l=0}^{s-1} \beta_{lia}\beta_{lja}\delta_{ij} & \qquad StateFanout \\
+R_3 M \sum_{b=0}^{v-1} \gamma_{ijb}\delta_{ij} & \qquad OutputFanout(Moore) \\
+R_3(1-M) \sum_{a=0}^{c-1} \sum_{b=0}^{v-1} \Phi_{ijab}\delta_{ij} & \qquad OutputFanout(Mealy) \\
+R_4(\alpha_{ij} + \alpha_{ji})\delta_{ij} & \qquad TieBreaker
\end{aligned} \tag{2.16}$$

where c, v and s denote number of input conditions, output variables and states respectively, and

$$\alpha_{lm} = \begin{cases} 1 & \text{if } S_m \subseteq \ Successor(S_l) \\ 0 & \text{otherwise} \end{cases}$$

$$\beta_{lma} = \begin{cases} 1 & \text{if } S_m \subseteq \ Predecessor(S_l, I_a) \\ 0 & \text{otherwise} \end{cases}$$

$$\gamma_{ijb} = \begin{cases} 1 & \text{if } Z_b(S_i) = Z_b(S_j) \\ 0 & \text{otherwise} \end{cases}$$

$$\delta_{ij} = \begin{cases} 1 & \text{if } i \neq j \\ 0 & \text{if } i = j \end{cases}$$

$$\phi_{ijab} = \begin{cases} 1 & \text{if } Z_b(S_i, I_a) = Z_b(S_j, I_a) \\ 0 & \text{otherwise} \end{cases}$$

$$M = \begin{cases} 1 & \text{for Moore machines} \\ 0 & \text{for Mealy machines} \end{cases}$$

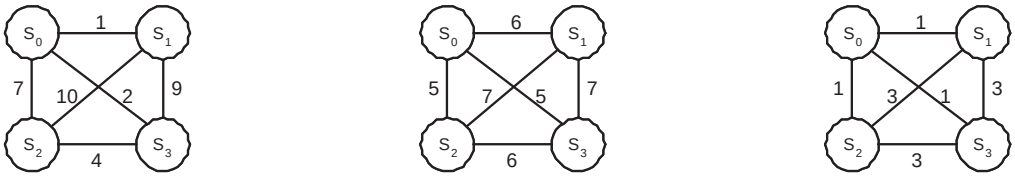
where S_i denotes state-i, I_a being input condition-a, and $Z_b(S_i)$ and $Z_b(S_i, I_a)$ denotes outputs in state-i for moore machine and in state-i along with input-a for mealy machine respectively. The terms R_1, R_2, R_3 to R_4 are scaling factors whose values used in this work are 4, 3, 2 and 1 respectively.

The first term in the Armstrong equation gives a weight of R_1 to pair of states that have a common predecessor or a common fanin state. Similarly the second term gives a weight of R_2 to pair of states that fanout to a common next-state. The third and fourth terms add a weight of R_3 to pair of states having similar output at similar input conditions. The two terms for R_3 distinct between Mealy and Moore types of machines. Finally, R_4 is used as a tie breaker if two states have transitions in between them.

An Example

The above three cost models are next explained by constructing their adjacency graphs using state-machine-4 as given in Table-2.8. The adjacency graphs for the three cost models are shown in Figure-2.1.

Consider the weighted arc between states $S1$ and $S2$ in Armstrong's graph (Figure-2.1(a)). The weight on the edge is calculated as follows: states S_1 and S_2 have a common predecessor state S_0 so R_1 is added. States S_1 and S_2 are



(a) Armstrong (b) Jedi (c) Mustang

Figure 2.1: Adjacency Graphs

both predecessors of states S_3 so R_2 is added. The state machine is of Mealy type and outputs for both the states at $I = 0$ are same (which is logic-1) so we add R_3. Finally there are transitions in between the two states and so R_4 is also added. Thus the total arc weight is $4 + 3 + 2 + 1 = 10$.

The arc between S_0 and S_1 in Figure-2.1(b) is derived as follows: State S_0 does not appear in output equation while S_1 appears twice. Thus the summation of the first term in equation-2.14 evaluates to 2. States S_0 and S_1 are present in next state equations of states S_1, S_2, and S_3 while number of encoding bits used is 2. Thus the second term evaluates to 4 and the total arc weight to 6. Similar calculations using equation-2.15 lead to the Mustang graph in Figure-2.1(c).

Affinity cost as modeled in adjacency graphs is next used to minimize equation-2.17.

$$\sum_{i=1}^{n_s} \sum_{j=1}^{n_s} A_{i,j}^{P} . \Delta(i,j) \tag{2.17}$$

where $\Delta(i, j)$ is the Hamming distance between codes of state i and j
$A_{i,j}^{P}$ being the affinity as given by Jedi, Mustang or Armstrong by their respective equations.

Jedi employed Simulated Annealing to solve equation-2.17. Mustang provided a constructive algorithm while Amaral [25] used Genetic algorithm in solving Armstrong's cost model.

2.2.4 Expand

Expand-function as used in ESPRESSO tool [27] is also utilized in this work in pursuit of a cost measure for area estimation. The Expand-function is discussed in detail in the next chapter.

PS	NS		z	
	$I=0$	$I=1$	$I=0$	$I=1$
S_0	S_1	S_2	0	0
S_1	S_2	S_3	1	1
S_2	S_1	S_3	1	0
S_3	S_3	S_1	0	1

Table 2.8: State Machine - 4

2.3 FSM Encoding for Low Power

Power dissipation has always been one of the major concerns in logic circuits design. Excessive power dissipation often causes chip run-time failure, reduction in chip life-time, and costs more expensive packaging. In recent times, portable electronics applications have given power-aware computing a whole new importance. This is due to the fact that limitations in battery capacities and progress trail far behind the ever increasing computing requirements. Power consumption is thus constrained and optimized at all levels of design hierarchy including technology selection, architectural transformation, logic synthesis and physical design [2]. VLSI designers have thus been faced with another optimization parameter of low power. Recently, a lot of work is reported in the literature to automate the exploration of low power solutions at different levels of VLSI hierarchy [2, 28]

Power Estimation for FSMs

The exact power consumption of a VLSI device is a complex function of many parameters and thus can only be accurately found out by running numerous power simulations on the final device. However, a simpler measure for power dissipation by a CMOS logic gate can be found out by the following equation.

$$P_{ave} = \frac{C_L V dd^2 E_{SW}}{2.T_{cyc}} \tag{2.18}$$

where T_{cyc} is the cycle time, C_L the load capacitance of a CMOS gate and E_{SW} being the expected switching activity at the gate's outputs.

The above equation shows that by reducing switching, supply voltage or capacitance seen by the gate, the power consumption of a CMOS device can be reduced.

There is a rich amount of work reported in the literature for power estimation of sequential circuits [29, 30, 31, 32]. The power estimation techniques can be broadly classified into statistical [33] or probabilistic [34]. Both the approaches are implemented in SIS [35] version 1.2. The statistical approaches work by simulating the state machine using the user provided input vectors and determining the state probabilities based on it. Probabilistic approaches on the other hand try to correlate the various probabilities in order to calculate state probabilities if the FSM is simulated for infinite amount of time. Statistical techniques can be fast and accurate if a short representative sequence for an FSM can be determined. However, determining such a sequence is an open research problem. Najm in [36] reports a statistical power estimation technique using randomly generated input sequences until a desired accuracy is achieved. Najm et al in [37] propose a technique to estimate power within a desirable accuracy of an FSM by simulating fraction of a large input set. The technique tries to simulate FSM repeatedly by blocks of consecutive vectors at random until a desired accuracy is achieved. A Monte-Carlo approach for power estimation for sequential circuits is also proposed [38]. The technique generates mutually independent power samples using multiple copies of the circuit that are simulated in parallel with mutually independent input vector streams. Samples are collectively analyzed to check for the terminating condition.

The power estimation problem is addressed even at a more higher level using entropy as power estimating function [39, 40]. The rationale is that since entropy is a measure of information-carrying capacity, a higher entropy on a state line means higher number of transitions on it. The maximum transition can be attributed when the probability on a line is exactly half and corresponds to its maximum entropy value.

A state transition graph (STG) is denoted by *G(V, E)* where a vertex $S_i \in V$ represents a state of the FSM and an edge $e_{i,j} \in E$ represents a transition from state S_i to S_j. Let P_{Si} denote the state probability, that is, the probability of finding the state machine in S_i at any given time, and p_{ij} denotes the conditional (state) transition probability, which is the probability of the machine making a transition from state S_i to state S_j, that is

$$p_{ij} = Probability(Next = S_j | Present = S_i) \tag{2.19}$$

A STG can be interpreted as a Markov chain. A Markov chain is a representation of a finite state Markov process [41]. A Markovian process is termed as memoryless since the probability distribution at any time depends only on

the present time and not on how the process arrived till that period. For a large class of Markovian processes for which our STG is also a member, the probability of a state is the limiting value approached as it is run for infinite amount of time. This is termed as *limiting state probability theorem* [42]. Mathematically

$$P_{Sj} = limit_{t->\infty} p_{ij} P_{Sj}(t) \tag{2.20}$$

The above can be iteratively found out by solving Chapman-Kolmogorov equations [43] as follows

$$\begin{aligned} P_{Si}(n+1) &= \sum_{j \in In\_State(i)} p_{ji} P_{Sj}(n) \\ & i = 1, 2, \cdots, M-1 \\ 1 &= \sum_{j} P_{Sj}(n+1) \end{aligned} \tag{2.21}$$

where n is iteration number and $In\_State(i)$ is the set of fanin states of i in the STG.

The process is terminated once state probabilities converge so that the difference between successive iterations is within a user defined tolerance value. To tackle the complexity of solving the above system of equations, approximate methods have been proposed in [44, 45]

The *Total State Transition Probability* for a transition from a state S_i to state S_j is the probability that the machine transits to state S_j given that it is in state S_i. The total state transition probability can thus be calculated as follows [46]

$$P_{ij} = p_{ij}.P_{Si} \tag{2.22}$$

where P_{ij} is the total state transition probability from state S_i to state S_j.

The sum of total state transition probabilities in between two states indicates the amount of switching in between them. This sum can be treated as a weight between the two states attributed on a single edge connecting them.

$$W_{ij} = P_{ij} + P_{ji} \tag{2.23}$$

A STG in which all the transitions between two states are replaced with a weighted edge is called a *weighted graph*. The weight on an edge indicates the relative proximity in the state assignment of the two connected states on that edge. By assigning shorter distance codes to states connected with

higher weights, i.e higher transition probability, the overall switching on the state lines of the FSM can be minimized. Thus a cost model for minimizing power consumption can be to have *Minimum Weighted Hamming Distance* (MWHD). Mathematically

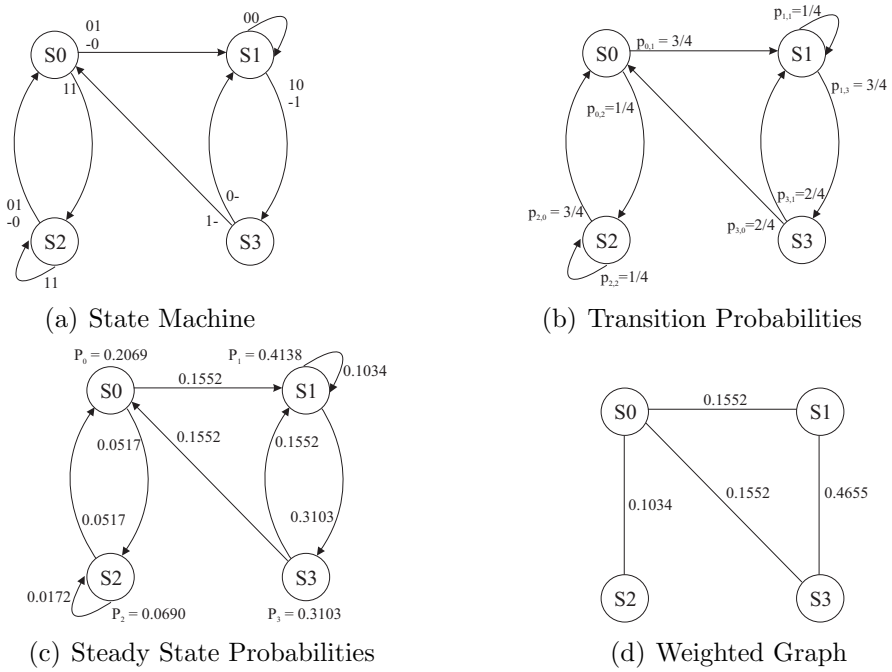
$$\sum_{S_i S_j \in S} W_{ij} H(S_i, Sj) \tag{2.24}$$

Consider a state machine as shown in state transition graph (STG) of Figure-2.2(a). The edge labels correspond to input configurations that cause a transition from a state at the tail of the edge to the state at its head. For example, in state S0, an input of either 00, 01 or 10 will cause the machine to transit to state S1, whereas an input of 11 will cause a transition to state S2. The information contained in STG is next used to compute a static probabilistic model of the FSM based on transition probabilities of the FSM. This is done by interpreting FSM as a Markov chain. The Markov chain model of the FSM can be described by a directed graph with a structure isomorphic to the STG and with weighted edges. The weight on the edges for a transition from state S_i to state S_j represents the conditional probability of the transition, p_{ij}. This model is shown in Figure-2.2(b). Calculation of weights on the edge is straightforward. For example, $p_{0,1} = P(00) + P(01) + P(10) = 3/4$, where $P(X)$ is the probability of input taking the value X. The next step is the computation of steady state and total transition probabilities from the Markovian model. Steady state probabilities are calculated by repeatedly iterating equation-2.21 until difference in successive iterations gets within user defined limits, i.e. convergence in values is achieved. The procedure can be started with any initial value. A detailed description of calculating steady state probabilities is given in Appendix-A. Figure-2.2(b) shows steady state probabilities achieved for the given example. The figure also shows total transition probability of the edges obtained using equation-2.22. The total transition probabilities between two states are finally added to construct affinity model for power minimization referred to as weighed graph. This weighted graph for the example is shown in Figure-2.2(d) and is next used to optimize MWHD problem.

Previous Work

Most of the work reported in the literature [47, 48, 49] tries to achieve minimum weighted hamming distance by optimizing the above equation for low power realization of FSMs.

However, as (2.18) shows, power consumption depends on how much ca-



(a) State Machine

(b) Transition Probabilities

(c) Steady State Probabilities

(d) Weighted Graph

Figure 2.2: MWHD formulation

pacitance is switched. A reduced amount of switching on greatly increased load capacitance may well offset any savings achieved. Thus, by reducing the switching activity, the problem is only half solved. However, the knowledge of the gate loading can only be accurately found out once the design is synthesized and mapped on a specific library.

Kang et al in [50] try to take into account area into their cost equation for low power FSM realization. The cost function used is a linear combination of minimum weighted hamming distance for power and the literal savings by Jedi cost model for area. However, since there is no correlation between the two terms, the technique does not aim at minimizing switched capacitance but merely tries to achieve a low power and area FSM solution. The rationale being that a low area solution will anyhow contribute towards a low power solution. The problem is solved using genetic local search algorithm.

Suresh et al [51] describe a modification of MWHD scheme. The algorithm tries to identify code swaps between states such that the final cost in terms of weighted switching can be reduced. The authors define base switching as the minimum amount of switching that is possible if the all the states are assigned a unidistance code. Relative switching is defined as a measure of goodness of how close the average switching is to the minimal possible base switching value. The algorithm then identifies 'slack' values which is the amount by

which the cost can be decreased if two state codes are exchanged. Edges with high slack values are first identified, then sorted and finally those that yield better costs are exchanged. The algorithm terminates when there is no more good exchanges remaining. The algorithm suffers from complexity of $O(n^3)$ as for every exchange, it has to take care of its effect on other edges connected to the two nodes in focus. Moreover, the greedy algorithm proposed is vulnerable to get stucked in a local minima.

Roy et al addressed the problem of minimizing power in sequential circuits in Syclop [52]. The authors use conditional transition probabilities in place of steady state probabilities while solving a MWHD solution. The hard nature of the problem is addressed by using simulated annealing algorithm. Once the state codes with reduced MWHD cost are found out, constrained multilevel logic synthesis is performed. A set of kernels are computed for each logic expression and a non-trivial intersection of kernels is selected so that fanout for nodes having high transition density can be reduced. The rationale is that reduced fanout on highly switched state lines will result in low switched capacitance.

A MWHD scheme is employed for non-minimal state encoding by Koegst et al in [53]. The authors advocate the use of a user specified input sequence for measuring total state transition probabilities, and thus weights, instead of equation-2.21. Koegst in [54] used a multi-criteria non-minimal state assignment for low power where assignment helps deactivating idle parts of FSM along with reducing MWHD.

A novel technique for low power state assignment is proposed by Majid et al [55]. The authors note that an optimal solution for MWHD problem can be otbained using Integer Linear Programming (ILP). However, any such technique suffers from exponential complexity of ILP itself. This can be mitigated if ILP has to be applied on small finite sets. They thus proposed a semi-gray encoding technique in which the states are partitioned into small groups in decreasing order of their weights. The states within a group are then assigned gray codes using ILP.

A low power FSM realization is proposed using Huffman style to provide non-uniform state codes in [56]. The technique proposes shorter codes for states with higher switching activities and more for lesser switched states. The rationale being that lesser state lines will yield lesser weighted switching as well as switched capacitance. However, the overhead of the scheme barred the authors to implement it as it is. Instead, the state set is encoded using only two different code lengths. Moreover, a logic is proposed to shut off clock for the inactive set.

Another interesting variation in MWHD approach is proposed by Silvano et al [57]. The authors note that the state assignment procedure can be broken down into state ordering and state encoding sub-steps. For state ordering, various techniques have been proposed so that a chain of highly probable states is formed. The rationale in doing so is that consecutive states from a highly probable state are more likely to be visited than stand alone nodes with high probability. Once states are ordered, they are encoded using encoding techniques described in [58]. The sate encoding techniques try to reduce hamming distance between consecutive states in the state ordering list as well as the states that are connected to those states.

Benini [59] proposed state assignment technique for low power based on total state probabilistic MWHD algorithm. The authors propose the use of a greedy variation of column-based encoding [17, 60]. The cost function also tries to minimize area using cost metrics of multilevel minimizers used in Jedi and Mustang. The two costs are minimized independent of one another and thus the technique essentially aims for low power and low area solutions simultaneously.

Pedram et al in [61] describe a novel technique for low power state assignment by introducing the concept of literal power savings. A power value to every literal is assigned based on its switching estimate. The literal weight is then used to find minimum weighted (switched) literal solution similar to MWHD. Power cost models for both two and multi-level logic implementation are described. Simulated Annealing algorithm is utilized for search space exploration.

Another interesting work for power and area minimization is presented in [62] by Chao et al. The authors use entropy measure to calculate the probability distribution of an FSM. They then distribute the number of possible codes into groups such that the codes within a group have equal number of ones. Each state is then assigned to a group so as to minimize the overall switching. A state is finally assigned a unique code within a group using literal saving estimates.

In some recent work, Almaini et al [63] have employed Genetic Algorithm [5] for independent power and area optimization. The area estimate utilized is based on exact number of cubes in a synthesized machine while MWHD metric is used for estimating power. The two estimates are combined using linear summation and also by having their product. Pomeranz et al [64] have also used Genetic Algorithm to partition the FSM such that inactive partitions be turned off to reduce power. They propose to do state encoding such tshat it can also determine the partition as well as state assignment.

2.4 FSM Encoding for Testability

Testability of a VLSI circuit is attributed to how efficiently the various faults in the circuit can be excited and observed. This involves generation and application of test sets at primary inputs of a circuit to excite its various faults and observe them at the outputs. The test sets can either be manually generated or using automatic CAD tools. Automatic test generation tools are efficient in terms of cost and effectiveness and so are generally employed to find the test patters. This work will also consider the use of Computer Aided Automatic Test Pattern Generation (ATPG) tools.

ATPG tools use both random and deterministic techniques to build the test set. Deterministic test set takes into account the behavior and structure of the circuit under test to build its test set. They thus yield higher fault coverage though being more computationally expensive. The complexity and type of a circuit, whether combinational or sequential, determines how efficiently automatic test pattern generator performs.

Test generation for combinational circuits is known to be NP-hard problem [65]. The worst case size of the search space is bounded by 2^i, where i is equal to the number of inputs. However, techniques have been developed to reduce this large search space by an intelligent search of the primary input combinations. These techniques include D-algorithm [66, 67], PODEM [68], and FAN [69]. ATPG tools based on these algorithms are quite efficient in finding test patterns to detect all the testable faults in an integrated circuit.

Automatic test pattern generation for sequential circuit is much more involved than combinational circuits. Unlike combinational ATPG, existing sequential ATPG tools may not produce satisfactory results for some class of circuits due to their complexity. For this reason, design for test techniques like partial scan [70] have been used to improve the testability of the circuit. The increased complexity of sequential circuits arises from memory feature in their behavior. To excite a fault, memory elements have to be first initialized to a proper fault exciting value. This requires a justification sequence to traverse a circuit from its current state to the initialized state. Fault excitation is followed by fault propagation to the primary outputs. Thus, sequential testing involves a time domain component and is usually performed in multiple clock periods. To cope with this difficulty, *Iterative Array Model* was proposed [3]. The model transforms the time domain aspect of sequential circuit into space domain by unrolling the sequential behavior into multiple iterations of its combinational circuit, effectively making it as a large combinational circuit. The iterative model thus permits the automatic test pattern generation algorithms

for combinational circuit to be extended to sequential logic.

A sequential circuit can be classified as cyclic or acyclic. If a node can be revisited after starting from that node in the forward direction without visiting any other node again, then a cycle is said to be present in the sequential circuit. The length of the cycle (cycle length) is said to be the number of sequential elements encountered during the traversal. Sequential depth refers to the number of sequential elements from primary input to the primary output.

The complexity of an ATPG can be attributed to the time it takes to attain the required level of test completeness. This in turn is a strong function of the complexity of the circuit. The upper bound on the number of vectors needed to test all testable faults in an acyclic sequential circuit with i inputs and sequential depth d is $d * 2^i$ [71], which is comparable to the complexity of a combinational circuit. However, a cyclic sequential circuit may require an initialization sequence to test the combinational logic in a given state. This initialization sequence can be as long as the $M-1$, M being the number of possible states for the state machine. Thus the upper bound for a cyclic sequential circuit having i primary inputs and M states, using s number of sequential elements is $M * 2^{s+I} = 2^{2s+I}$ [71]. It clearly shows that the complexity for ATPG of sequential circuit increases exponentially with the number and length of the cycles. The complexity of sequential ATPG is investigated by Lioy et al in [72]. The authors contend that that the complexity of sequential ATPG depends on the number of flip-flop per loop (FF/L) and the number of loops per flip-flop (L/FF). The former estimates the cyclic structure of the circuit and the latter predicts how much the design is 'winded up' on itself or how much interdependence exists between the loops. The authors note that the test generation complexity increases with FF/L and L/FF, while increasing the number of state-controlling inputs reduces its complexity. The authors further propose a formal algorithm to identify the loops within a sequential circuit. Marchok et al in [73] also note that the complexity of sequential ATPG varies with retiming. Furthermore, the authors cite a new factor, *density of encoding*, which gives the measure of degree of valid states compared to the number of possible states in the state machine, to be key indicator in the complexity of structural sequential ATPG. The complexity of ATPG is carefully investigated in [74].

As described earlier, the nature of encoding strongly determines the structure of the sequential circuit, its various dependencies, cycles and interconnections. The information flow inequality of state machines (section-2.1) enables us to quickly realize its structure prior to its synthesis. This can help provide an accurate measure of complexity of a sequential circuit at a higher level of

abstraction.

Pomeranz et al [75] explored the possibility of controlling more state lines in order to increase the testability of a sequential circuit. They have proposed a synthesis technique that evaluates some state variable functions using primary inputs or primary output functions. The motivation is that since primary outputs are directly observable and primary inputs directly controllable, an increase in testability can be achieved.

Cheng et al [76] have proposed a novel method of encoding that reduces the feedbacks in a sequential circuit. The motivation is to reduce the cyclic nature of the sequential circuit. The authors propose state encoding by following states merging according to some rules. The first rule tries to maximize the number of blocks in a partition while the second tries to merge two states having the same next state. The rationale for the former rule is that by having a large number of blocks in a partition, more information can be derived from primary inputs alone for the next state function, that in turn reduces the number of feedbacks. The latter rule aims at area minimization by incorporating the commonly used cost metrics used in multilevel area minimization for a sequential circuit.

Mohat et al in [77] try to take into account the testability for PLA-based FSMs. The authors propose K-hot encoding scheme to deal with various types of PLA faults. In K-hot code, exactly K-bits are set equal to 1. The rationale is that many types of PLA faults can be easily detected if exactly K lines are high.

Prinetto et al in [78] discuss testability measure for inputs and outputs of an FSM. The authors note that optimal testability using pseudo-random patters is achieved when outputs are high for half of the possible inputs and low for the other. They further note that such a condition runs counter to power minimization condition where the aim is to have reduced switching.

2.4.1 Testability Parameters

Testability of a circuit is generally measured in terms of three important parameters namely *Fault Coverage*, *Fault Efficiency* and *CPU-Time*. Fault Coverage denotes the ratio of faults that can be detected with the total number of faults in a circuit. Mathematically,

$$FaultCoverage = \frac{Total\ Number\ of\ Detected\ Faults}{Total\ Number\ of\ Faults} \tag{2.25}$$

Fault coverage thus denotes the degree of coverage obtained by the testa-

bility tool in detecting or covering the faults of the circuit.

Fault Efficiency is another ratio describing the number of faults that are detected or proven to be undetectable with the total number of faults in the circuit. Fault efficiency signifies the efficiency of the testability tool in exploring the total number of faults. A difference between fault-efficiency and fault-coverage means that some of the faults in the circuit, though attempted or explored by the ATPG tool, could not be excited or detected. This is generally due to the circuit structure as will be shown in the next chapter.

CPU-Time is CPU or system-time consumed by the ATPG software for its processing. CPU-time denotes the degree of difficulty in doing the ATPG. For example in a sequential circuit, a higher CPU time maybe because the circuit had to be unrolled for a bigger number of sequential iterations. In other words, the circuit may have series of interdependent flip-flops or loops due to which there exists a difficulty in justification and propagation of faults. There can be other factors effecting CPU-time like fault excitation through unused states in a sequential machine.

2.5 Multiobjective Optimization

Many real-world optimization problems involve two types of difficulties: a) multiple, conflicting objectives, and b) a highly complex search space. On the one hand, instead of a single optimal solution, competing goals give rise to a set of compromise solutions, generally denoted as Pareto-Optimal. In the absence of preference information, none of the corresponding trade-offs can be said to be better than the others. On the other hand, the search space can be too large and too complex to be solved by exact methods. Thus, efficient optimization strategies are required that are able to deal with both difficulties. FSM state assignment problem is not far from these real-world problems as it also involves *multiple, possibly conflicting objectives and a highly complex search space.*

A general *multiobjective optimization problem* (MOP) includes a set of n parameters (decision variables), a set of k objectives, and a set of m constraints. Objective functions and constraints are functions of the decision variables. The optimization goal is defined as,

$$minimize \quad y = f(x) = (f_1(x), f_2(x), ..., f_k(x)) \tag{2.26}$$

$$subject\ to \quad e(x) = (e_1(x), e_2(x), ..., e_m(x)) \leq 0 \tag{2.27}$$

where

$$\begin{aligned} x &= (x_1, x_2, ..., x_n) \in X \\ y &= (y_1, y_2, ..., y_k) \in Y \end{aligned}$$

and x is the decision vector, y is the objective vector, X is denoted as the decision space, and Y is called the objective space. The constraints $e(x) \leq 0$ determine the set of feasible solutions. The feasible set X_f is defined as the set of decision vectors x that satisfy the constraints $e(x)$[79].

In the state assignment problem (SAP) addressed in this work, two of the three objectives area and power, are to be minimized while testability is to be increased under the constraint that minimal length encoding is used. The encoding length is fixed at minimal during the assignment so the constraint is always being met. Then an optimal solution might be an assignment which achieves minimal area, minimal power dissipation, with total testability. If such a solution exists, we actually only have to solve a single objective optimization (SOP). The optimal solution for any objective is also the optimum for other objectives [79]. However, what makes MOP's difficult is the common situation when the individual optima corresponding to the distinct objective functions are sufficiently different. Then the problem has usually no unique, perfect solution, but a set of equally efficient, or non-inferior, alternative solutions, known as the Pareto-optimal set [80]. SAP is not far from this difficulty, because it is possible that for a particular change in assignment solution there is a decrease in one cost but it may result in the increase in other cost. For example it is possible that certain change results in decrease in overall area, but the number of nets having high switching probability may increase resulting in high power dissipation, or the power dissipation may decrease but it may also increase testability difficulty as certain logic values may be difficult to control/observe due to lesser logic switching in the resulting circuit. Therefore, it is needed to solve SAP as an MOP.

2.5.1 Search and Decision Making

Depending on how optimization and the decision process are combined, multi-objective optimization methods can be broadly classified into three categories [79].

(a) Decision making before search

The objectives of the MOP are aggregated into a single objective which implicitly includes preference information given by a human decision maker (DM).

(b) Search before decision making

Optimization is performed for individual objectives, without any preference information given. The result of the search process is a set of (ideally Pareto-optimal) candidate solutions from which the final choice is made by the DM.

(c) Decision making during search

The DM can articulate preferences during the interactive optimization process. After each optimization step, a number of alternative trade-offs are presented on the basis of which the DM specifies further preference information, which guide the search.

The aggregation of multiple objectives into one optimization criterion has the advantage that the classical single-objective optimization strategies can be applied without further modification. In this thesis the same approach is used. In order to combine all the objectives into a single objective function, a fuzzy goal based aggregation is used. In this aggregation, fuzzy logic is combined with a modified goal programming approach. In the next two sections goal programming and fuzzy logic concepts are presented, followed by the formulation of fuzzy goal based aggregating function for SAP problem.

2.5.2 Goal Programming

In this aggregation method, the decision maker has to assign targets or goals that he/she wishes to achieve for each objective. These values are incorporated into the problem as additional constraints. The objective function will then try to minimize the absolute deviation from the targets to the objectives. The simplest form of this method may be formulated as,

$$minimize \quad f(x) = \sum_{i=1}^{k} w_i \left| f_i(x) - T_i \right|, \qquad subject\ to\ x \in X_f \tag{2.28}$$

where T_i denotes the target or goal set by the decision maker for the ith objective function $f_i(x)$, w_i is the weight of $f_i(x)$, and X_f is the set of feasible solutions as mentioned before. A more general formulation of goal programming objective function is weighted sum of the pth power of the deviation

$|f_i(x) - T_i|$. Such a formulation has been called *generalized goal programming* [81]

The main strength of this technique is its computational efficiency in case we know the desired goals that we wish to achieve, and if they are in feasible region. However, its main weakness is that, it needs appropriate weights or priorities for the objectives, which in most cases is difficult unless there is prior knowledge about the shape of the search space. Also, if the feasible region is difficult to approach, this method becomes very inefficient. This technique is useful if a linear or piecewise-linear approximation of the objective functions can be made.

2.5.3 Fuzzy Logic

Fuzzy Logic is a mathematical tool invented to express human reasoning. In classical (crisp) reasoning a proposition is either true or false whereas in fuzzy system a proposition can be true or false with some degree.

Fuzzy Sets

A classical (crisp) set is normally defined as collection of elements or objects $x \in X$. Each single element x either belongs to the set X (true statement), or does not belong to the set (false statement). Whereas a fuzzy set can be defined as,

$$A = \{(x, \mu_A(x)) | x \in X\}$$

$\mu_A(x)$ is called the membership function or grade of membership (or degree of truth) of x in A that maps X to the membership space M. The range of the membership function is a subset of the non-negative real numbers whose supremum is finite [82]. Elements with zero degree of membership are normally not listed.

Like crisp sets, operations such as union, intersection, and complementation etc., are also defined on fuzzy sets. There are many operators for fuzzy union and intersection. For fuzzy union, the operators are known as **s-norm** operators (denoted as $\oplus$). While fuzzy intersection operators are known as **t-norm** (denoted as *).

Fuzzy Reasoning

Fuzzy reasoning is a mathematical discipline to express human reasoning in vigorous mathematical notation. Unlike classical reasoning in which propositions are either true or false, fuzzy logic establishes approximate truth value of

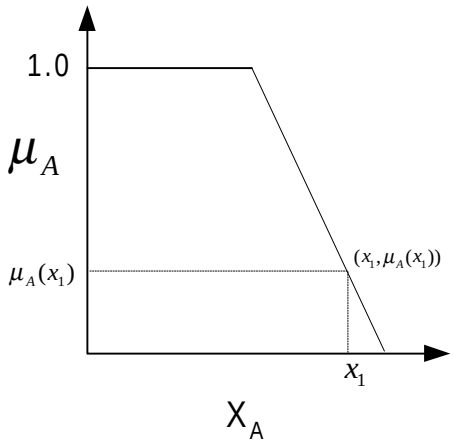


Figure 2.3: Membership function of a fuzzy set A.

propositions based on linguistic variables and inference rules [83]. By linguistic variable we mean a variable whose values are words or sentences in natural or artificial language [84]. The linguistic variables can be composed to form propositions using connectors like AND, OR and NOT. Formally, a linguistic variable comprises five elements [85].

1. The variable name.
2. The primary term set.
3. The Universe of discourse U.
4. A set of syntactical rules that allows composition of the primary terms and hedges to generate the term set.
5. A set of semantic rules that assigns each element in the term set a linguistic meaning.

For example area can be used as linguistic variable for FSM state-assignment problem. According to the syntactical rule, the set of linguistic values of area may be defined as very big, big, medium, small, and very small. The universe of discourse for linguistic variable is positive range of area of a design, eg., [150 $literals$, 30 $literals$]. The set of semantic rules define fuzzy sets for each linguistic value. A linguistic value is characterized by its corresponding fuzzy set. The membership in fuzzy set is controlled by membership functions like Fig. 2.3. It shows the designer knowledge of problem [83].

Fuzzy Operators

There are two basic types of fuzzy operators: operators for the intersection, interpreted as the logical "and," and union, interpreted as the logical "or," of fuzzy sets. The intersection operators are known as triangular norms (t-norms), and union operator as triangular conorms (t-conorms or s-norms) [82]. Normally "or" logic is implemented using maximum operator defined as,

$$\mu(x) = max\{\mu_A(x), \mu_B(x)\} \tag{2.29}$$

Whereas, "and" logic is normally implemented using minimum operator defined as,

$$\mu(x) = min\{\mu_A(x), \mu_B(x)\} \tag{2.30}$$

Also the fuzzy complementation operator is defined as,

$$\bar{\mu}_B(x) = 1 - \mu_B(x) \tag{2.31}$$

Ordered Weighted Averaging Operator

Generally, formulation of multi criteria decision functions do not desire pure "anding" of **t-norm** nor the pure "oring" of **s-norm**. The reason for this is the complete lack of compensation of **t-norm** for any partial fulfillment and complete submission of **s-norm** to fulfillment of any criteria. Also the indifference to the individual criteria of each of these two forms of operators led to the development of Ordered Weighted Averaging (OWA) operators [86, 87]. This operator allows easy adjustment of the degree of "anding" and "oring" embedded in the aggregation. According to [86, 87], "orlike" and "andlike" OWA for two fuzzy sets A and B are implemented as given in Eqn. 2.32 and Eqn. 2.33 respectively,

$$\mu_{A\cup B}(x) = \beta \times \max(\mu_A, \mu_B) + (1-\beta) \times \frac{1}{2}(\mu_A + \mu_B) \tag{2.32}$$

$$\mu_{A\cap B}(x) = \beta \times \min(\mu_A, \mu_B) + (1-\beta) \times \frac{1}{2}(\mu_A + \mu_B) \tag{2.33}$$

β is a constant parameter in the range [0,1]. It represents the degree to which OWA operator resembles a pure "or" or pure "and" respectively.

To solve MOP using fuzzy logic, first all the objectives are defined in terms of linguistic variable. A linguistic rule is made using ("and" and "or" logic) in order to combine these linguistic variable. Each linguistic variable is also mapped to a fuzzy membership value in the fuzzy set of good in terms of that

objective. This membership value is the functions of some base value based on the numerical value of the actual cost. All the membership values are combined into one membership value, using t-norm or s-norm operators. The selection of t-norm or s-norm operator depends upon the predefined linguistic rule. The combined membership value is now used as aggregating function. The best solution is that, which results in the highest combined membership value.

2.6 Iterative Algorithms

A number of iterative algorithms are proposed in the literature. The motivation for using iterative algorithms becomes clear when recalling the hard nature of the FSM encoding problem as mentioned above. These algorithms are capable of efficiently searching for a near optimal solution in a large solution space and have been very successful in solving a number of combinatorial optimization problems in various disciplines of science and engineering. In the following, a brief description of genetic algorithm (GA) and tabu search (TS) algorithms is presented.

2.6.1 Genetic Algorithm (GA)

GA is an elegant search technique that emulates the process of natural evolution as a means of progressing towards the optimal solution. A high level algorithmic description of GA is given in Figure 2.4 [5]. GA uses an encoded representation of a solution in the form of a string made up of symbols called *genes*. The string of genes is called *chromosome*. The algorithm starts with a set of initial solutions called *population* that may be generated randomly or taken from the results of a constructive algorithm. Then, in each iteration (*known as generation in GA terminology*), all the individual chromosomes in the population are evaluated using a *fitness function*. Then, in the *selection* step, two of the above chromosomes at a time are selected from the population. The individuals having higher fitness values are more likely to be selected. After the selection step, different operators namely *crossover, mutation,* and *inversion* act on the selected individuals for evolving new individuals called *offsprings*. These genetic operators are described below.

Crossover is an important genetic operator. It is applied on two individuals that are selected in the selection step to generate an offspring. The generated offspring inherits some characteristics from both parents in a way similar to

```
Algorithm (Genetic_Algorithm)
  (N_p = Population Size)
  (N_g = Number of Generations)
  (N_o = Number of Offsprings)
  (P_i = Inversion Probability)
  (P_μ = Mutation Probabilty)
  Begin
    (Construct initial population)
    Construct_Population(N_p);
    For j = 1 to N_p
      Evaluate_Fitness (Population[j])
    EndFor;
    For i = 1 to N_g
      For j = 1 to N_o
(Choose parents with probability proportional to fitness value)
        (x,y) ← Choose_parents;
        (Perform crossover to generate offsprings)
        offspring[j] ← Crossover(x,y)
        For k = 1 to N_p
          With probability P_μ apply Mutation (Population[k])
          With probability P_i apply Inversion (Population[k])
        EndFor;
        Evaluate Fitness(offspring[j])
    EndFor;
    Population ← Select(Population, offspring, N_p)
  EndFor;
  Return highest scoring configuration in population
End. (Genetic Algorithm)
```

Figure 2.4: Outline of simple Genetic Algorithm [5].

natural evolution. There are different crossover operators namely *simple, order, partially mapped, and cycle*. The simple crossover operation for instance, works by choosing a random cut point in both parent chromosomes (the cut point should be the same in both parents) and generating the offspring by combining the segment of one parent to the left of the cut point with the segment of the other parent to the right of the cut [5]. For description of other crossover operators, see [88, 5, 89].

The *mutation* operator is used to introduce new random information in the population. It helps to prevent the search process from trapping in local minima. An example of mutation operation is the swapping of two randomly selected genes of a chromosome. The importance of this operation is that it can introduce a desired characteristic in the solution that could not be introduced by the application of the crossover operator alone. However, mutation is applied with a low rate so that GA does not turn into a memory-less search process [88].

There is an addition of offsprings in the population size after crossover operation. In order to keep the number of members in a population fixed, a constant number of individuals are selected from this set which consists of both the individuals of the initial population, and the generated offsprings. If M is the size of the initial population and N_o is the number of offsprings created in each generation, then, before the beginning of next generation, we select M new parents from $M + N_o$ individuals. There can be various selection policies to select the next set of parents within the pool. One such policy could be to greedily select the best individuals from the set. Another policy could be to be random in selection.

The quality of the solution obtained from GA is dependent on the choice of certain parameters such as population size, number of generations, crossover and mutation rates and also the type of crossover used. The selection of values for these parameters is problem specific and so there are no hard and fast rules for this purpose. The choice of these parameters is left to the conception and intuition of the person applying GA to a specific problem.

2.6.2 Tabu Search (TS)

Tabu search is an iterative heuristic that has been applied for solving a range of combinatorial optimization problems in different fields [5]. Tabu search starts from an initial feasible solution and carries out its search by making a sequence of random moves or perturbations. A *tabu list* is maintained that

stores the attributes of a number of previous moves. This list prevents bringing the search process back to already visited states. In each iteration, a subset of *neighbor* solutions is generated by making a certain number of moves and the best move (the move that resulted in the best solution) is accepted, provided it is not in the tabu list. Otherwise, if the said move is in the tabu list, the best solution is checked against an *aspiration criterion* and if satisfied, the move is accepted. Thus, the aspiration criterion can override the tabu list restrictions. It is desirable in certain conditions to accept a move even if it is in the tabu list, because it may take the search into a new region due to the effect of intermediate moves. The behavior of tabu search heavily depends on the size of tabu list as well as on the chosen aspiration criterion. Different sizes of tabu list result in short-term, intermediate term, and long-term memory components that can be used for intensifying or diversifying the search. The aspiration criterion determines the extent to which the tabu list can restrict the possible moves. If a tabu move satisfies aspiration criterion, then the move is accepted and tabu restriction is overridden. The structure of TS is given in Figure 2.5. The detailed description of tabu search can be found in [5].

Algorithm *Tabu_Search*

Ω : Set of feasible solutions
S : Current solution
S* : Best solution
Cost: Objective function
N(S): Neighborhood of S ∈ Ω
V* : Sample of neighborhood solutions
T : Tabu list
AL : Aspirartion level

Begin

Start with an initial feasible solution S ∈ Ω
Initialize tabu list and aspiration level
For fixed number of iterations **Do**
    Generate neighbor solutions V* ⊂ N(S)
    Find best S* ∈ V*
    If move S to S* is not in T **Then**
        Accept move and update best solution
        Update T and AL
    Else
        If Cost(S*) < AL **Then**
            Accept move and update best solution
            Update T and AL
        End If
    End If
End For

End.

Figure 2.5: Outline of Tabu Search algorithm [5].

Chapter 3

Problem Formulation and Solution Methodology

3.1 Introduction

A typical VLSI design process is divided into several levels of design abstraction as depicted in Figure 1.1. More knowledge of design is added as we move down the abstraction levels. Thus a more accurate estimation of circuit attributes is possible down the hierarchy. However, the increase in accuracy is mostly gained at a cost of increase in complexity of cost estimation as there are additional details down the hierarchy that were abstracted at the higher level.

In this work, we have utilized a number of cost/fitness functions to model the problem of finite state machine state assignment and discussed their effectiveness in solving our objectives. This chapter formulates the state assignment problem and details the cost models we have used in our work.

3.2 Problem Statement

An FSM M can be formally defined as a 5-tuple $M = (S, I, O, T, \delta)$ where S represents the finite state space, I represents the finite input space and O the finite output space, $\delta : I \text{x} S \rightarrow S$ is the next state function and T is the transition relation defined as I x $S \rightarrow O$ (for a Mealy machine) or $T : S \rightarrow O$ (for a Moore machine).

State assignment involves an injective mapping f: S $\rightarrow B^n$ where n is the code length ($n \geq \lceil \log_2 |S| \rceil$) and B^n is an n-dimensional boolean space, a boolean hypercube. Objectives addressed in this thesis are the minimization of area, power and testability of the synthesized state machine circuit. Minimum

code-length is considered as constraint.

3.3 Cost Functions for Multilevel Area Minimization

As discussed earlier in chapter-2, multilevel area is estimated in terms of number of literals synthesized from a state assignment. In this work, a number of cost measures for multilevel area are experimented with. These include Jedi, Mustang and Armstrong cost measures as described in section-2.2. We also have used Expand-function that is utilized in ESPRESSO tool. Moreover, this work also investigates the use of support function as discussed earlier in section-2.1 and four new literal saving estimates that are the focus of this section.

3.3.1 Literal Savings - 1

The first literal saving measure, given in equation-3.1, is an exact number of literals that can be saved from a pair of states.

$$LS1_{S_iS_j} = \sum_{k=1}^{m_0} 2(n_E - \Delta_{ij} - 1)T_{ij}^{O_k} + \sum_{k=1}^{n_s} 2\lambda_k(n_E - \Delta_{ij} - 1)T_{ij}^{NS_k} - (n_E - \Delta_{ij}) \tag{3.1}$$

where,
Δ_{ij} represents the hamming distance between states i and j,
$T_{ij}^{O_k}$ and $T_{ij}^{NS_k}$ are boolean values representing if both states i and j are present in an output function O_k, or next-state functions NS_k, respectively,
m_o is the number of outputs,
n_s is the number of states,
n_E is the number of encoding bits,
λ_k is the number of ones in the statecode of state k.

The first two terms give the number of literals that can be saved by taking out a pair of common-literal set from output and next-state functions respectively, while the last term accounts for a single instantiation of the saved term.

3.3.2 Literal Savings - 2

The second literal saving measure improves upon the previous literal saving estimate by noting that a common-literal set can be extracted from several terms in an output or next-state equation. This exact number is obtained by summing the recurrences of a pair of states in output or next state equations. Equation-3.2 gives the cost model for second literal savings function.

$$LS2_{S_iS_j} = \sum_{k=1}^{m_0}(n_E - \Delta_{ij} - 1)(P^o_{jk} + P^o_{ik}) + \sum_{k=1}^{n_s}\lambda_k(n_E - \Delta_{ij} - 1)(P^{NS}_{jk} + P^{NS}_{ik}) - (n_E - \Delta_{ij}) \tag{3.2}$$

where,
$P^o_{k,i}$ is the number of times state k is represented in output i,
$P^s_{k,i}$ is the number of times state k is represented in state i.

The above model is very similar to Jedi (equation-2.14) with the terms abstracted before are accounted by exact values. This is because of availability of state-coding information in the algorithms used in this work.

3.3.3 Literal Savings - 3

The third literal saving model is based on Mustang (equation-2.15) by multiplying the recurrences instead of addition. The third literal saving model is given in equation-3.3

$$LS3_{S_iS_j} = \sum_{k=1}^{m_0}(n_E - \Delta_{ij} - 1)(P^o_{jk} * P^o_{ik}) + \sum_{k=1}^{n_s}\lambda_k(n_E - \Delta_{ij} - 1)(P^{NS}_{jk} * P^{NS}_{ik}) - (n_E - \Delta_{ij}) \tag{3.3}$$

3.3.4 Literal Savings - 4

The final literal saving model used in this work combines two-level savings with multilevel literal savings models. Model-3 is used for estimating multilevel savings. The cost model is depicted in equation-3.4. The two-level savings estimation is based on the principle of implicant merging (2.2) in which a pair of terms unidistance apart can be merged together. Such a sharing results in a literal savings of $(n_E + \#inputs + 1)$ per merged pair. The literal savings are abstracted to n_E in the model.

$$LS4_{S_iS_j} = \sum_{k=1}^{m_0} n_E U_{S_iS_j} T_{ij}^{O_k} + \sum_{k=1}^{n_s} n_E U_{S_iS_j} T_{ij}^{NS_k} + LS_3 \tag{3.4}$$

where,
$U_{S_iS_j}$ is boolean high if states S_i and S_j are unidistance apart.

3.3.5 Expand

Expand operation is used by many heuristic two-level minimizers, notably ESPRESSO [27] and Mini [90]. The goal of Expand-function is to increase the size of each implicant of a given cover F, so that implicants of smaller size can be covered and deleted. Maximally expanded implicants are primes of the function. As a result, the Expand operator makes a cover prime and minimal with respect to single-implicant containment.

Expand operation uses positional-cube notation (POS) for binary encoding. The binary codes are encoded in positional-cube notation by 2-bit fields as follows:

Binary	POS
0	10
1	01
-	11

Table 3.1: POS Notation/Encoding

The POS notation doubles the number of columns in an implicant table but simplifies various implicant manipulation operations as will be seen with Expand operation.

The expansion of an implicant is done by raising one (or more) of its 0s to 1. This corresponds to increasing its size (by a factor of 2 per raise), and therefore to covering more minterms. The fundamental question in the expansion process is whether the expanded cube is still valid, i.e., it is still an implicant of the function f. This is accomplished by checking for an intersection of the expanded implicant with the off-set, F^{OFF}.

The computational efficiency and the quality of expanded cover depends on the order in which the implicants are being selected for expansion. Heuristics are used for ordering the implicants. The rationale behind the ordering heuristic is to expand those cubes first that are unlikely to be covered by other

cubes, i.e, those having fewer 1s in the densely populated columns. The technique works by computing a vector whose entries are the column sums of the matrix representing F. Each cube is next assigned a weight that is the inner product of the cube itself and the previously computed vector.

The Expand operation is next considered on the state machine given in Figure-3.1(a). A sample state-assignment for the state machine is given in the Figure-3.1(b). The state machine is next represented in the form of state-table with the symbolic state assignments replaced with actual assignments as shown in Figure-3.1(c). We will use the state machine to show Expand operation on F_1 (F_1^{ON}) input cover. Input cover for F_1 in POS notation is given in Figure-3.1(d). The off-set F_1^{OFF} for the given example is the set of all implicants that are not in F_1. The information regarding the off-set is also available in the state-table. Input combinations that remain unspecified in the state-table is the set of don't care values. The given example has completely specified inputs and thus empty don't-care set.

The Expand operation begins by constructing column count vector representing the number of 1s in individual columns of F_1 cover in POS notation. Let the column count vector be ordered from left to right in the same ordering as the cover. Then the vector is $[14144123]^T$. Weights of the implicants are next determined by having the product of the cover with the column count vector. For e.g, the weight of the first implicant W_1 is found as

$$W_1 = 0\text{x}1 + 1\text{x}4 + 0\text{x}1 + 1\text{x}4 + 1\text{x}4 + 0\text{x}1 + 1\text{x}2 + 0\text{x}3 = 14$$

The weights of the implicants thus calculated in F_1 are $(14, 12, 12, 15, 11)$ Thus the fifth implicant is processed first, i.e., 01 01 01 10.

The Expand operator first tries to raise the 0 in column 1 to 1. This yields a temporary implicant 11 01 01 10 that intersects with the off-set of F_1 (10 01 01 10 lies in F_1^{OFF}), and thus is rejected. Similarly, raising of columns 3 and 8 also interest with the off-set and are rejected. However, column 5 can be raised to an implicant 01 01 11 10 that covers the first implicant of F_1. Thus as the result of expansion, the fifth and the first implicants of F_1 get covered in a bigger cube. The two implicants covered are deleted from the list of implicants to be covered.

Among the remaining set of implicants to be covered, there are two least weighted implicants with weights of 12. The second implicant (10 01 10 01) is randomly selected for Expand operation. Column-2 in the implicant is next raised due to which the fourth implicant also gets covered. The last remaining implicant, the third implicant (01 10 10 01), is finally selected for expansion which can only be expanded in the fourth column to re-cover the already

covered fourth implicant. The final cover thus obtained for F_1 is shown in Figure-3.1(e). The final cover for both the sequential elements in PLA format is given in Figure-3.1(f).

The Expand cover for F_0 and F_1 is composed of 8 and 9 literals, respectively. This is done by counting the number of 1s and 0s in the respective covers. There are five implicants where the literal for f_0 is not a don't care, i.e., there are five implicants depending on f_0 or in other words there are five branches stemming out of flip-flop-0. Similarly, flip-flop-1 is being fed to five implicants. The flip-flop fanout calculation is utilized in fanout based power cost measure as will be discussed in the next section.

The current work utilizes the off-set of a function available in the state-table and thus bypasses an expensive operation of complement to find out the off-set from the on-set and don't-care set. Expand operation on an implicant is allowed by only checking the non-intersection with the off-set. Such an expansion thus implicitly uses the undefined values in don't care set as valid positions for expansion.

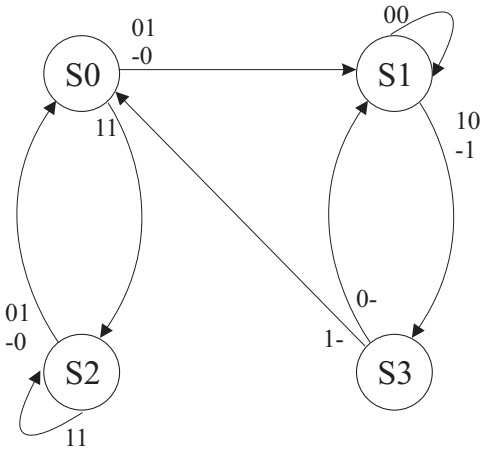
3.4 Cost Functions for Power

Two formulations for power minimization problem in finite state machine are explored in this work. Both the formulations utilize total steady state transition probability between states as discussed in section-2.3. This work utilizes Matlab software for calculation of steady state probabilities. A detailed description of using Matlab software for calculating steady state probabilities is given in Appendix-A.

Power can be reduced if the logic being switched, or in other words the switched capacitance in a circuit can be reduced. This can be achieved by either reducing the total switching in the logic or by reducing the logic itself or both of them. In case of sequential circuit, switching activity in the combinational logic is due to logic transitions on the flip-flops as well as primary inputs. The transitions on sequential elements propagate to the combinational logic cone that is dependant on those flip-flops causing switchings in parts of that logic. Thus, to reduce switched capacitance in an FSM, one can:

1. Minimize flip-flops transition frequencies. A lesser number of transitions will ensure lesser switching in the combinational logic cone and thus reduced power.

2. Minimize fanout branches (fanout) from flip-flops. By reducing fanout of



(a) State Machine

S0 = 00
S1 = 01
S2 = 10
S3 = 11

(b) State Assignment

Inputs $I_1 I_0$	Present State $f_1 f_0$	Next State $F_1 F_0$
00	00	01
01	00	01
10	00	01
11	00	10
00	01	01
01	01	11
10	01	11
11	01	11
00	10	00
01	10	00
10	10	00
11	10	10
00	11	01
01	11	01
10	11	00
11	11	00

(c) State Table

I_1	I_0	f_1	f_0
01	01	10	10
10	01	10	01
01	10	10	01
01	01	10	01
01	01	01	10

(d) F_1 in POS notation

I_1	I_0	f_1	f_0
01	01	11	10
11	01	10	01
01	11	10	01

(e) F_1 Expand Cover

$I_1 I_0 f_1 f_0$

F_1
1 1 - 0
- 1 0 1
1 - 0 1

F_0
- 0 0 -
0 - - 1
0 - 0 -
- - 0 1

(f) Final Expand Cover

Figure 3.1: State

flip-flops, the load capacitance they encounter is reduced thus reducing switched capacitance.

3. Minimize the logic being switched. Fanout from a flip-flop can also be used as an estimate of size of combinational logic cone that is dependant on that flip-flop. A high fanout from a flip-flop means a big cone of logic being fed from that flip-flop which may eventually translate into a bigger combinational circuit. Thus by reducing the fanout size, one may expect to reduce the logic being switched.

The first formulation based on *minimum weighted hamming distance* (MWHD) was discussed in detail in previous chapter. MWHD approach tries to minimize the total transition probability of the state machine in the hope that the total number of logic transitions in the synthesized circuit will also get reduced, i.e. it tries to maximize power reductions only due to point-1 above.

The second formulation used in this work tries to combine all the factors discussed above. It can be observed that points 2 and 3 are interrelated as one factor can be traded for the other. For example, a big combinational logic having lesser switching frequency may be equivalent in power consumption to smaller logic with higher switching frequency. Thus to maximize reduction in switched capacitance, fanout branches can be weighted with respect to flip-flop transition frequency in order to give higher reduction priority to fanouts branches stemming out of frequently switched flip-flops. This can be stated mathematically as minimizing equation-3.5

$$Fanout = \sum_{i=1}^{n_E} T_i B_i \tag{3.5}$$

where B_i and T_i are the number of fanout branches and the transition frequencies of flip-flop-i, respectively. In this work, Expand cover is used to calculate the fanout. A procedure of fanout calculation was discussed in the previous section. Calculation of flip-flop transition frequencies is discussed next.

The steady state transition probabilities of states can be combined with state assignment to determine steady state transition probability of flip-flops. Consider for example the four-state FSM in Figure-2.2. The weighted graph along with a sample state assignment is reproduced in Figure-3.2. The weight on edges connecting two states represent transition probability between the two states.

Two memory elements, F_0 and F_1, are used in encoding the state machine. A flip-flop will switch its logic value if the state machine traverses to a state

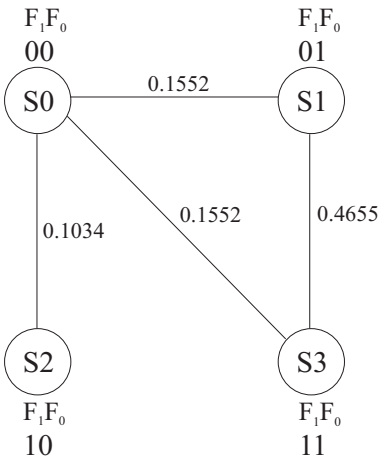


Figure 3.2: Weighted Graph & State Assignment for Fanout based approach

having value opposite to the current value of the flip-flop. Thus, transition for F_0 takes place when the state machine switches between states S0 and S1 or between $S0$ and $S3$. Similarly, F_1 switches during state machine transition in between states $S1$ and $S3$, $S0$ and $S2$ and $S0$ and $S3$, respectively. Flip-flop transition probability can thus be obtained by summing those edge weights where the flip-flop logic transition occurs. Flip-flop transition probabilities can thus be calculated as

$$T_0 = 0.1552 + 0.1552 = 0.3104$$
$$T_1 = 0.1552 + 0.1034 + 0.4655 = 0.7211$$

These values can now be used in equation-3.5 to yield the second formulation used in this work, *minimum weighted fanout*. The approach is also referred as *fanout* in the thesis.

3.5 Cost Functions for Testability

This work addresses testability objective by adapting previously used measures and further building them by utilizing the information available from the Expand cover. The Expand cover provides a good estimate for calculating dependencies of sequential elements resulting from a state assignment. The dependencies are then further processed to provide an estimate of number and depth of loops in the synthesized circuit.

As discussed earlier (see section-2.4), one of the testability measures used previously sums depths for all the loops in a synthesized circuit. This measure

was implemented in the current work and is referred as **TDepth**. Mathematically,

$$TDepth = \sum_{k=1}^{\#loops} Depth_k \tag{3.6}$$

TDepth measure tries to ease difficulty in justification of a desired value by reducing the number and depths of loops. This has the effect of reducing the number of time frames required for processing a value.

ATPG tools like HITEC [91] are based on three-valued logic; the allowed value set being $(0, 1, X)$. Boolean algebra using the three basic gates on three valued logic is tabulated in Tables-3.2 and 3.3.

Inputs	AND	OR
(0, 0)	0	0
(0, 1)	0	1
(1, 0)	0	1
(1, 1)	1	1
(1, X)	X	1
(0, X)	0	X
(X, X)	X	X

Table 3.2: Boolean Algebra using 3-valued logic on two-input AND/OR gates

Inputs	NOT
0	1
1	0
X	X

Table 3.3: Boolean Algebra using 3-valued logic on NOT gate

ATPG tools initialize the sequential elements with don't care value (X). To test the faults present in a circuit, sequential elements are usually required to be initialized to some known state of either logic-1 or logic-0. One way to achieve this is to provide an explicit reset for all the flip-flops. However, if the design does not provision a reset line, the sequential elements are to be initialized using the available control lines, i.e. by using primary inputs of the circuit.

There can be a situation where a flip-flop never gets initialized. This may happen due to three valued logic as is shown in Figure-3.3. Such a circuit converts the D-type flip-flop into a T-type and is a usual occurrence in counter-circuits. The flip-flop is initially initialized to a don't care. This don't care value no matter which input value is XORed with remains a don't care at flip-flop's inputs, inhibiting the flip-flop initialization. Due to this uninitializability, any fault on line A or B will remain untested. The circuit, though being optimal according to equation-3.6, is in fact untestable and should be avoided for testability concerns.

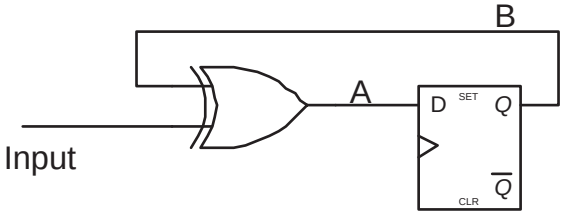


Figure 3.3: Uninitializable Flip-Flop

The second testability measure used in this work estimates uninitializability of a circuit resulting from an assignment and tries to favor initializable implementations. The estimation is based on information contained in the Expand cover.

A flip-flop can be initialized to logic-zero if all the product terms in its cover can be set to logic-zero. In SOP realization, this can be realized as having all inputs of the OR-gate set at logic low. Similarly, to set a flip-flop at logic-high, at-least one of the product terms have to be raised to logic-high.

This zero initializability of a flip-flop can be evaluated by computing complement of its on-set. Presence of a complement term then denotes availability of a condition that can simultaneously turn off all the implicants in the on-set. For example, in the Expand cover of Figure-3.1(e), F_1 can be set to low by setting the inputs, I_0 and I_1, to logic-low. F_1 can also be set to low by setting both the sequential elements at logic-high. However as initially flip-flops are initialized to unknown values according to three-valued logic, only inputs can be used for initializing flip-flops to some binary value in the beginning. The initialized flip-flops can later on be used for initializing the rest of the sequential elements.

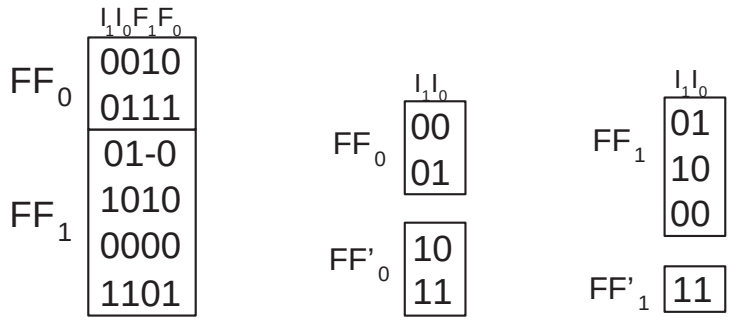
Thus, a logic-low or a logic-high initialization for the first flip-flop after startup can be summarized as follows.

Logic-High Initialization: A flip-flop can be initialized to logic-high if there

exists an implicant in its cover that only depends on inputs.

Logic-Low Initialization: A flip-flop can be initialized to logic-low if their exists an implicant in its complement cover that only depends on inputs.

Initialization of subsequent flip-flops may then proceed by utilizing the flip-flops initialized. Initialization sequence can be best explained with an example. Consider a cover of a circuit having two inputs, I_0 and I_1, and two flip-flops (FF), F_0 and F_1 as shown in Figure-3.4(a).



(a) Initial Cover (b) FF_0 (c) FF_1

Figure 3.4: Zero Initialization Sequence

The initialization sequence can start with any flip-flop. In the sequel, FF_0 is considered first. There are two product terms in the cover of FF_0. As the flip-flops are initialized to don't-cares, initialization may only be possible through controlling input set of the two product terms. There is no product term that depends only on inputs and so logic-one initialization is currently not possible. Input set of the two product terms, reproduced in the upper part of Figure-3.4(b), is next considered for zero initialization through complement. The complemented cover is given in the lower part of Figure-3.4(b). The existence of complement cover for FF_0 predicts its zero-initializability.

FF_1 is next considered using the initialization information available from FF_0. The zero initialization of FF_0 ensures that the last product term in the cover of FF_1, 1101, will automatically be turned off. Thus, the remaining three terms are complemented to get the complemented-cover, FF_1', given in Figure-3.4(c). This indicates that FF_1 is zero-initializable. Moreover, with the zero initialization of FF_0, all the literals in the first product term for FF_1 can be set to logic-high. Thus, FF_1 can also be initialized to logic one.

The initialization may then proceed back to see if this new information can be utilized for logic-one initializability of FF_0. In the given example, FF_0 can now be initialized to logic-one using initialization of FF_0 and FF_1 to logic-zero and logic-one respectively.

An accurate initializability measure would keep on considering all the flip-flops iteratively as long as there is some new initializability information available. However, such an iterative measure becomes too costly to compute. To keep the calculations simple, an iterative initialization is considered where only those flip-flops that were not previously initialized to any logic value are considered. The rationale is that the availability of a number of initialized flip-flops may increase the probability that the previously initialized flip-flops can now be initialized to the other logic values.

Initialization of flip-flops is essential for full testability of a sequential circuit. Ease of testing using the first measure can now be combined with a measure of initializability of the machine to yield an estimate of a fully and easily testable sequential circuit. The second cost model for testability thus developed is given in equation-3.7.

$$Integrated\ Testability\ Measure = \frac{TDepth + 1}{aI_0 + bI_1} \tag{3.7}$$

where I_0 and I_1 are boolean values indicating zero and one-initializability of flip-flops and a and b are weights given to them.

Total loops depth is incremented by one to account for a situation where there are no loops in the circuit. The denominator in the equation estimates initializability of the circuit. A higher number of initialized values on flip-flops thus translates into a reduced final cost. This work uses weights of 2 and 1 for a and b respectively. The use of such weights is explained next.

The cover produced by Expand-function may contain redundant terms that subsequently get removed by other heuristics in the synthesis process. There is thus a potential inaccuracy in the initialization estimate that needs to be further analyzed.

A logic-one initializable cover may not remain logic-one initializable if the product terms responsible for logic-one initializability get eliminated during synthesis process. This can happen if the implicants responsible for logic-one initializability were detected redundant in the cover. Heuristic *Irredundant*, which is the next operation performed after literal expansion in ESPRESSO, removes redundant terms in the cover. However, invalidation of logic-zero initializability may happen due to the alteration in the cover by addition of new product terms. This can happen by the iterative application of *Reduce* and *Expand* functions in ESPRESSO. As gradual steps in the synthesis procedure try to further simplify expanded cover, there is lesser probability of invalidation of logic-zero initializability in the cover than invalidation of logic-one initializability.

The inaccuracy in estimation of logic-one initialization can be best understood using an example. Consider an Expand-cover as

$$F_1 = I_1 I_2 + I_1 F_1' + I_2 F_1 + F_1 F_2$$

There exists a term in the cover that is only dependant on the inputs and thus the flip-flop is predicted as logic-one initializable. The sequential element cannot be initialized to logic-zero. This cover when synthesized yields the following minimized form

$$F_1 = I_1 F_1' + I_2 F_1 + F_1 F_2$$

The term essential for logic-one initialization gets reduced in the final cover, as a consequence of which the flip-flop is no longer logic-one initializable. Such an inaccuracy in estimation of logic-one initialization is encapsulated in the cost model by giving it half the weight to logic-zero initializability.

3.6 Complementation

Complement of a function can be formally defined as the set of all possible values of a function that do not intersect with either the on-set or don't-care-set of a function.

Complement of a function is an important requirement for Expand operation and in evaluation of zero-initializability for the current work. As discussed earlier, the current work bypasses the use of complement during the course of Expand operation. However, complement is an important requirement for evaluating zero-initializability of a flip-flop.

This work employs two methods of computing complement. The first method, a formal way of computing complement, is similar to the one employed in program ESPRESSO which is based on unate recursive paradigm. However, the formal complement is an expensive recursive operation that becomes too costly for bigger sized circuits. Moreover, the complete complement set is also not a requirement for checking logic-zero initializability. The presence of a single complement is enough to predict zero initializability. Thus, the formal complement method is an expensive operation which is also beyond the requirements. To overcome these shortcomings in formal complement, a new heuristic is proposed that quickly checks the presence of complement of a function. These two methods of complementation are discussed next.

3.6.1 Formal Complement

The complement of a function implemented in program ESPRESSO is based on unate recursive paradigm [92]. The idea behind the unate recursive paradigm is to carry out an operation on a variable-by-variable basis, breaking the problem into subproblems of smaller variable sizes. Each variable is individually considered in both its positive (logic-value high) and negative (logic-value low) unate forms or cofactors and their partial results combined. The recursive complement of a function f can thus be expressed as

$$f' = x.f'_x + x'.f'_{x'} \tag{3.8}$$

where f'_x and $f'_{x'}$ denote complement of the cofactors of the function w.r.t. variable x and x' respectively.

The recursive procedure terminates if the (sub)function is found to be tautology or the problem breaks down into a single implicant where the complement can be computed using De Morgan's law. Detailed rules concerning the stopping criteria of complement are described in [92]. Formal complementation is next explained using an example.

Consider a function $f = ab + ac + a'$ whose cover in POS notation can be described as

01 01 11
01 11 01
10 11 11

The efficiency of complement computation depends on the choice of variable used for splitting at each step of recursion while using equation-3.8. Variables that are binate (having both logic-zero and logic-one forms) are preferred choice for splitting.

The only binate variable is a, which is chosen for splitting. The cofactor f_a is given as

11 01 11
11 11 01

and $f_{a'}$ as

11 11 11

The cover $f_{a'}$ is a row of all 1s. Being independent of any other variable, the cover is thus a tautology and will no longer be used in computing complement of f.

We next consider variable b in f_a. The cofactor f_{ab} is [11 11 − 11] which is again a tautology. The cofactor $f_{ab'}$ however is [11 11 − 01] which being a single implicant can be complemented using De Morgan's law. Complement of cofactor $f_{ab'}$ thus is [11 11 − 10]. This being the only complement results in the complement of the function that can be also be expressed in the form $f' = ab'c'$.

3.6.2 Quick Complement Check

The formal complement method is an expensive recursive operation that becomes too costly as the size and number of implicants increase. The complement cover provided by formal complement is also beyond the requirements for zero-initialization. Quick complement check (QCC) heuristic is thus proposed in this work to quickly check the presence of complement in a cover. The heuristic is detailed in this section.

The heuristic tries to find a variable assignment that can turn-off all the implicants in the on-set of a cover. The search for such an assignment is carried out variable by variable basis in steps called *decision steps*. At every decision step of the algorithm, an assignment is made to a variable from the set of unassigned variables; such assignments are referred to as *decision assignments* and unassigned variables as *decision variables*.

Quick complement check falls in the class of deterministic heuristics that constructs a solution by choosing the best local decision assignment at every decision step. The choice is local as the order of variable selection is predetermined. For example, the variable selection order taken by the heuristic in the previous example will be from left to right, i.e. variable a, b and finally variable-c. The order is fixed so to limit the possible number of choices to decide from at every decision step.

The heuristic is based on selecting decision assignment on a chosen decision variable in order to turn-off the implicant(s) that have least flexibility of being turned-off in successive decision steps. To aid the decision assignment based on the above criteria, certain rules have been proposed for QCC to work. These rules can be formally stated as follows:

1. Highest priority in decision assignment is to be given to implicant(s) having least flexibility of being turned-off in successive decision assignments. That is, in other words, implicant(s) having least dependencies

on input controlling variables. The condition can be easily evaluated by counting don't care set in the implicant(s). A high number of don't cares translate into lesser dependency on input controlling variables.

2. If there is a tie in rule-i then preference is to be given to a decision assignment that turns-off maximum number of implicants.

Once a variable gets assigned, it is eliminated from the decision set. If by successive applications of the above rules, an implicant is reduced to only one controlling variable in the decision set then an assignment to that variable becomes mandatory. Such an assignment is called mandatory assignment.

QCC is terminated if

1. Decision variable set becomes empty, i.e. all the variables are assigned in the decision set. After assigning all the variables, if there still exist implicants that could not be turned-off then QCC indicates absence of complement for the given cover.

2. There exist a situation where opposite mandatory assignments are required on a decision variable. Complement for the cover is again reported void in such a case.

3. The given on-set is covered (turned-off) by the decision set. QCC signals presence of complement of the cover.

The heuristic is next explained using examples.

Consider an input cover as shown below.

a	b	c	d
01	11	10	11
10	11	01	01
01	11	01	11

The variable set from left to right are marked as a, b, c and d, respectively that will also be the order in decision assignments.

To assist in the decision assignment according to the above stated rules, the implicants are first given priority weights. The weight on an implicant is equivalent to the number of minterms it covers. The weighted implicants in the current example can thus given as

$$01\ 11\ 10\ 11\ ;\text{Wt} = 4$$
$$10\ 11\ 01\ 01\ ;\text{Wt} = 2$$
$$01\ 11\ 01\ 11\ ;\text{Wt} = 4$$

$$DecisionVariables = (a, b, c, d)$$
$$DecisionSet = (0)$$

According to the rules above, implicant having highest priority weight is to be selected for assignment on a decision variable. In the current sequel, the first and third implicants have highest priority weights and variable-a is the first decision variable to be considered for an assignment. Implicant-1 is randomly selected among the two, which has a logic-one assignment to chosen decision variable. Thus to turn-off implicant-1, an opposite logic-zero assignment is taken for variable-a. As a by-product of the variable assignment, implicant-3 also gets turned off. Thus after the first decision step, the problem status is as follows:

$$11\ 11\ 01\ 01\ ;\text{Wt} = 2$$
$$DecisionVariables = (b, c, d)$$
$$DecisionSet = (a = 0)$$

Variable-b is next considered in the second decision step. However the remaining implicant is independent of variable-b. Thus the example snapshot after second step is as follows

$$11\ 11\ 01\ 01\ ;\text{Wt} = 1$$
$$DecisionVariables = (c, d)$$
$$DecisionSet = (a\ =\ 0,\ b\ =\ -)$$

In the third decision step, variable-c is considered. Variable-c has a logic-one assignment to the remaining implicant in the on-set. Thus to turn the implicant off, variable-c is assigned logic-low. All the implicants are turned off without using variable-d and the decision set obtained is as follows

$$DecisionSet = (a\ =\ 0,\ b\ =\ -,\ c\ =\ 0,\ d\ =\ -)$$

As an another example, consider the cover used in the example for formal complement previously. The weighted implicants with snapshot of internal variables is given below

$$
\begin{gathered}
a \;\; b \;\; c \\
01\ 01\ 11\ ;\mathrm{Wt} = 2 \\
01\ 11\ 01\ ;\mathrm{Wt} = 2 \\
10\ 11\ 11\ ;\mathrm{Wt} = 4 \\
DecisionVariables = (a, b, c) \\
DecisionSet = (0)
\end{gathered}
$$

Decision variables will again be considered in a fixed left to right order for simplicity. Variable-a is thus selected as the first variable to be considered. The highest implicant weight is of implicant-3 for which variable-a has a logic-low assignment. To turn-off this implicant, variable-a is thus assigned logic-high in the first decision step. The updated snapshot is as follows

$$
\begin{gathered}
11\ 01\ 11\ ;\mathrm{Wt} = 2 \\
11\ 11\ 01\ ;\mathrm{Wt} = 2 \\
DecisionVariables = (b, c) \\
DecisionSet = (a \;=\; 1)
\end{gathered}
$$

In the second decision step, both the remaining implicants have the same priority weight; however, implicant-1 requires a mandatory assignment for variable-b. Variable-b is therefore assigned logic-low to turn off the first implicant to yield the following snapshot

$$
\begin{gathered}
11\ 11\ 01\ ;\mathrm{Wt} = 2 \\
DecisionVariables = (c) \\
DecisionSet = (a \;=\; 1,\ b \;=\; 0)
\end{gathered}
$$

Finally, variable-c is used to turn off the remaining implicant with a logic-low assignment. The decision set thus obtained using QCC is $(a = 1,\ b = 0,\ c = 0)$ which is the same as obtained using formal complement.

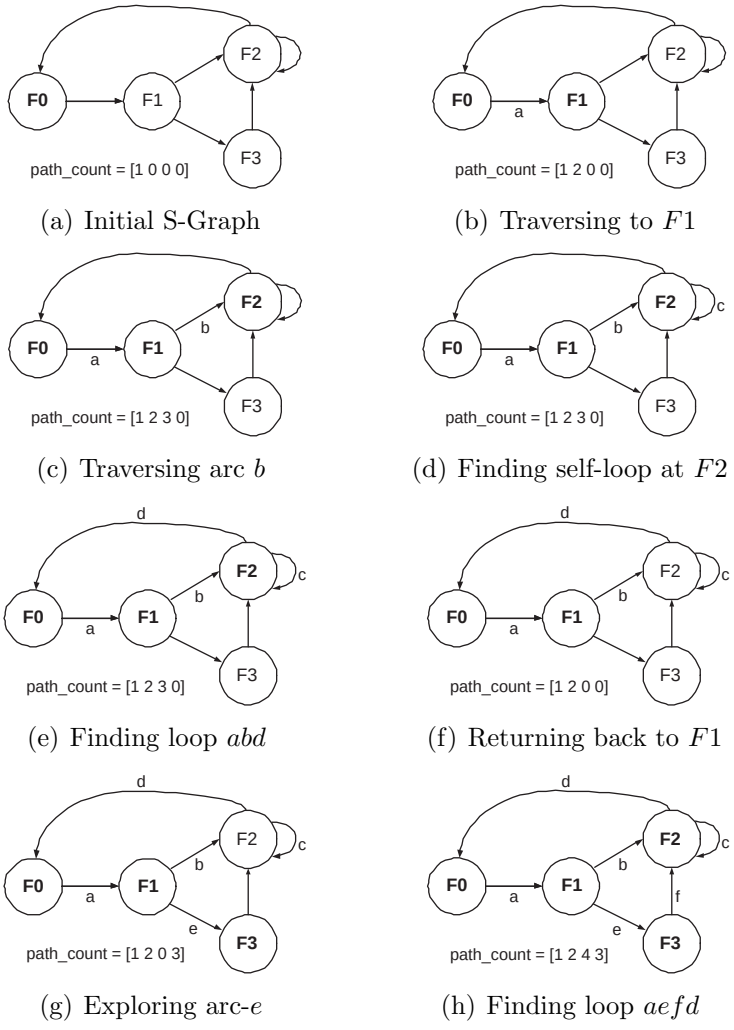
QCC was tested by using formal complement on a large number of cases and was seen to have a high accuracy. Further discussion on accuracy of QCC and its counter-examples will be discussed in the next chapter.

3.7 Loops Calculation

This work proposes a new method of evaluating loops of sequential-elements-dependencies or simply loops. Besides providing number of loops, the method also gives the depth of each loop in the cover.

The method proposed for loops calculation in this work is based on S-Graph based representation of sequential elements dependencies [93][94]. An S-Graph is a directed graph where the vertices represent sequential elements and information dependency between sequential elements is represented with a directed edge. The head of the edge pointing to the node which is dependant on the information of the node at the edge's tail. For example, in the directed graph given in Figure-3.5(a), sequential element $F1$ is dependant on $F0$.

The algorithm used for calculating the number and depths of loops in this work is presented in pseudo-code form in Figure-3.6. The algorithm is next explained using the S-Graph of Figure-3.5(a).



(a) Initial S-Graph (b) Traversing to $F1$

(c) Traversing arc b (d) Finding self-loop at $F2$

(e) Finding loop abd (f) Returning back to $F1$

(g) Exploring arc-e (h) Finding loop $aefd$

Figure 3.5: Loops Evaluation

Initially all the vertices and arcs in the S-Graph are marked as unvisited.

```
function find_cycles()
 for (each node N){                          // Initialize all nodes and arcs in the graph as unvisited
  N.visited = FALSE
   for (all arcs A from N){
    A.visited = FALSE
   }
 }

 for (each node N)  {
  path_count = empty                         // path_count stores the sequence number of visiting a node
  path_count(N) = 1                          // Node N is the first node in the itinerary
   if(N.visited == FALSE)    {
    N.visited = TRUE
    cycle_search(N, 1, path_count, false)
   }
 }
endfunction

function cycle_search(CurrentNode, Seq_Num, path_count, IsNewLoop)
 NewSelfLoop = FALSE
 for (all arcs A out of CurrentNode)    {
  NextNode = Destination(A)                  // NextNode is the node being pointed by arc A
   if(A.visited == FALSE)    {               // A new loop can exist if visiting arc A for the first time
    A.visited = TRUE
     if(NextNode == CurrentNode)
      NewSelfLoop = TRUE
     else
      IsNewLoop = TRUE
  }

   if(path_count(NextNode) != NULL)      {
    if(NewSelfLoop or IsNewLoop)      {
     // NextNode was previously visited and itinerary contains at least one unvisited arc
     // => a new loop has been discovered
      Loop = Loop + 1
      Depth(Loop) = | path_count(CurrentNode) - path_count(NextNode) | + 1          // 1 is added for self-loops
    }
  }
   else {
    if(NextNode.visited == FALSE)      {
    // Add NextNode in the itinerary with proper visit number and search for loops arising from it
     Seq_Num = Seq_Num + 1
     path_count(NextNode) = Seq_Num
     NextNode.visited = TRUE
     cycle_search(NextNode, Seq_Num, path_count, IsNewLoop)
      // Pop back NextNode from itinerary after loops exploration from it
     path_count(NextNode) = 0
     IsNewLoop = FALSE                       // All the arcs are now visited
    }
  }
 }
endfunction
```

Figure 3.6: Loops evaluation algorithm

An array *path_count* is also maintained that marks the sequence in which the nodes are visited. The procedure starts from node, $F0$, marking it as the first node traversed in the *path_count* variable as shown in Figure-3.5(a). Recursive procedure *cycle_search* is then called that evaluates all the cycles containing *CurrentNode*, or $F0$ in the present case.

The procedure cycle_search explores the graph in a depth-first-search manner. A new cycle is discovered if the traversal leads back to a previously visited node using an un-traveled path or in other words at least one non-traversed arc. The array path_count stores the previously visited nodes and their order of traversal. Any new path traversed is remembered in a boolean variable *IsNewLoop*. When a new loop is found, the associated sequence number of the last two nodes traversed are subtracted to find its depth. Self-loops are handled in a similar manner with the exception that any new arc visited in a self-loop is not taken as a new path. Self-loops are considered to be at a sequential-depth of 1 in this work. Thus 1 is added to all the loops evaluated. Various iterations of the recursive procedure cycle_search in detection of cycles present in the S-Graph of Figure-3.5(a) are detailed next.

1. There is only one arc stemming out of $F0$. The arc is traversed to reach to node $F1$. The previously un-traveled arc is marked as visited arc-a or in short a in Figure-3.5(b). A new arc traveled also sets possibility of finding a new loop by setting *IsNewLoop* to true. Node $F1$ is added to path_count list as the second node to be reached in the itinerary. Visited nodes in the current itinerary are marked as bold. Recursive procedure is recalled to search for cycles stemming from node $F1$.

2. $F1$ is new CurrentNode. There are two paths branching out of $F1$. Path leading to $F2$ is randomly selected and arc thus traversed is marked as arc-b in Figure-3.5(c). $F2$ is added as third node visited in the cycle searching sequence.

3. A self-loop is found out at $F2$ (Figure-3.5(d)). Self-loop arc is marked as arc-c.

4. The other arc at $F2$, arc-d is traversed reaching back to a previously visited node $F0$ in Figure-3.5(e), marking discovery of a new loop *abd*. Visit numbers for $F2$ and $F0$ are subtracted and added with self-loop bias to calculate the depth of the loop *abd* to be 3.

5. As all the arcs at the current node ($F2$) are covered and there exists no unvisited next node, the recursive depth-first search procedure rolls back

to $F1$ in Figure-3.5(f). The boolean *IsNewLoop* is set to false marking covering of all previously traveled arcs in loop(s).

6. Arc-e is next traveled to reach to an unvisited node $F3$ in Figure-3.5(g). A new path traversed sets again the possibility of finding a new loop by setting boolean *IsNewLoop* to true.

7. From $F3$, arcs-f and d are next traveled in a similar manner to deduce the final loop $aefd$ as shown in Figure-3.5(h). There lies no new node or path in the graph and so the procedure cycle_search terminates.

3.8 Fuzzy Goal Based Aggregation for SAP

In this method, it is assumed that there are Γ Pareto-optimal solutions. Also a p-valued cost vector $C(x) = (C_1(x), C_1(x), ..., C_p(x))$, where $x \in \Gamma$ is given. There are vectors $O = (O_1, O_2, ..., O_p)$ and $U = (U_1, U_2, ..., U_p)$ that give the lower bounds and upper bounds on the cost for each objective respectively such that $O_j \leq C_j(x) \leq U_j \ \forall j$, and $\forall x \in \Gamma$. These lower bounds and upper bounds are dynamically calculated and updated periodically.

In order to solve multiobjective placement problem, linguistic variables are defined as: area, power dissipation, and testability. The following fuzzy rule is used to combine the conflicting objectives.

Rule R1:

IF a solution is within
acceptable area
AND/OR
acceptable power dissipation
AND/OR
acceptable testability cost
THEN it is an acceptable solution.

The above mentioned linguistic variables are mapped to the membership values in fuzzy sets *within acceptable area, within acceptable power dissipation and within acceptable testability cost*. These membership values are computed using the fuzzy membership functions shown in Fig. 3.7.

For each objective the goal is to have membership value equal to one. Using Eqn. 2.33 and minimum operator, rule **R1** is interpreted as follows,

$$\mu_(^c x) \quad = \quad \beta^c \times O(\mu_a^c(x), \mu_p^c(x), \mu_t^c(x)) + (1 - \beta^c) \times \frac{1}{3} \sum_{j=a,p,t} \mu_j^c(x) \quad (3.9)$$

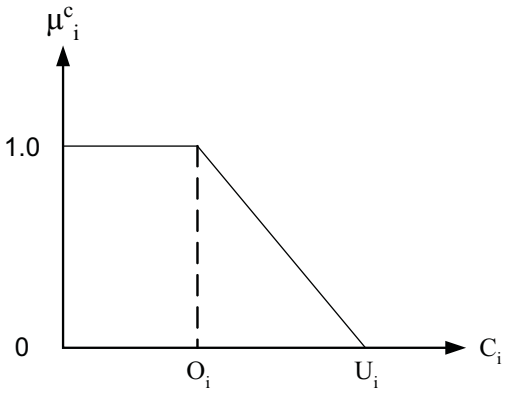


Figure 3.7: Membership functions

where O is min or max operator and $\mu^c(x)$ is the membership of solution x in fuzzy set of acceptable solutions, i.e. having "acceptable area AND/OR acceptable power AND/OR acceptable testability". $\mu^c_j(x)$ for $j = a, d, t$, are the membership values in the fuzzy sets *within acceptable area, within acceptable power, within acceptable testability* respectively. β^c is the constant in the range $[0, 1]$, the superscript c represents the "cost". This work gives equal weightage to averaging and degree of anding/oring in OWAO formulation above by having value of β^c to be 0.5. The solution that results in maximum value of $\mu^c(x)$ is reported as the best solution found by the search heuristic.

Chapter 4

Non-Deterministic Evolutionary Heuristics for FSM State Assignment

4.1 Introduction

This chapter discusses the non-deterministic evolutionary heuristics, Genetic Algorithm and Tabu Search that are employed in solving the objectives of this work. In particular, the chapter focusses on the design of various operators and parameters used by the heuristics in exploring the search space of FSM state assignment problem (SAP). It also discusses the operators as designed by previous researchers along with an analytical discussion on their efficacy when targeting FSM state assignment problem. This is done by carefully analyzing the nature of the problem with a detailed example. The design of evolutionary heurisitcs as discussed in this chapter will be empirically evaluated and analyzed in forthcoming chapter.

To better comprehend the intuition behind operators designed for SAP, the chapter begins with a brief description on the nature of the problem itself. This is followed by discussion on evolutionary heuristic, Genetic Algorithm, and finally Tabu Search.

4.2 SA Inheritance

Genetic Algorithms work on the principles of genetic evolution where characteristics of parents are passed on to offsprings. Thus an important consideration in the design of GA for SAP is to quantify inheritance for SAP.

We shall investigate inheritance by preserving next state functions and state assignments one at a time. These two have an interacting nature which can be best understood using an example. Consider the state machine in Table-4.1 with three different state assignments, alpha, beta, and gamma.

PS	NS				z	Assignt-α	Assignt-β	Assignt-γ
	I_a	I_b	I_d	I_c				
	$\overline{I_0 I_1}$	$\overline{I_0} I_1$	$I_0 I_1$	$I_0 \overline{I_1}$		$F_2F_1F_0$	$F_2F_1F_0$	$F_2F_1F_0$
A	C	A	D	B	0	100	110	010
B	E	C	B	D	0	111	101	111
C	C	D	C	E	0	000	000	000
D	E	A	D	B	0	110	100	110
E	E	D	C	E	1	010	010	100

Table 4.1: State Machine - 1

The unminimized next state equations, $F_2F_1F_0$, for assignment-α are given as follows:

$$F_0 = AI_c + DI_c + BI_d \tag{4.1}$$

$$F_1 = BI_a + EI_a + DI_a + CI_b + EI_b + AI_c + BI_c + CI_c + DI_c + EI_c + AI_d + BI_d + DI_d \tag{4.2}$$

$$F_2 = AI_b + CI_b + DI_b + EI_b + AI_c + BI_c + DI_c + AI_d + BI_d + DI_d \tag{4.3}$$

Assignment-α is next used to synthesize the FSM using ESPRESSO tool with multi-output minimization heuristic. The above functions are reduced to a literal cost of 24 in the following forms:

$$F_0 = I_0 * \overline{I_1} * F_2 * \overline{F_0} + I_0 * I_1 * F_0; \tag{4.4}$$

$$F_1 = \overline{I_0} * I_1 * \overline{F_2} + \overline{I_1} * F_1 + I_0 * F_2 + I_0 * \overline{I_1}; \tag{4.5}$$

$$F_2 = I_1 * F_2 * \overline{F_0} + \overline{I_0} * I_1 * \overline{F_2} + I_0 * F_2; \tag{4.6}$$

The next state function for F_0 in the above sequence is reduced from three implicants (terms) to two after minimization. This is because the codes for states A and D are unidistance apart, resulting in implicant-merging of the terms AI_c and DI_c before into $I_0*!I_1 * F_2!F_0$ after synthesis (see section-2.2). Similar implicant-mergings reduce the size of state functions for F_1 and F_2 as well.

The next state equation for F_1 in equation-4.2 was originally expensive having the highest number of implicants and thus remains so in the minimized form in equation-4.5. A slight modification from Assignment-α to Assignment-β can reduce the implicant size of $F1$ without effecting the functions for F_0 and F_3. The next state equation for of F_1 using Assignment-β thus reduces its size from 13 terms to 7 as follows:

$$F_1 = BI_a + DI_a + EI_a + AI_b + DI_b + CI_c + EI_c \tag{4.7}$$

The minimized expressions using Assignment-β after synthesis are found to be:

$$F_0 = \overline{I_0} * I_1 * \overline{F_2} + I_0 * F_0; \tag{4.8}$$

$$F_1 = \overline{I_0} * I_1 * F_0 * \overline{F_2} + \overline{I_0} * \overline{I_1} * F_0 * \overline{F_1} + \overline{I_1} * \overline{F_0} * F_1 + I_0 * \overline{I_1} * \overline{F_0}; \tag{4.9}$$

$$F_2 = I_0 * \overline{I_1} * F_0 * \overline{F_2} + I_0 * I_1 * F_2; \tag{4.10}$$

The minimized literal count using assignment-β increases to 26 literals. It can also be observed that in spite of reducing the size of unminimized form of F_1, its actual cost after synthesis increases from 9 literals to 14 by using assignment-β. An analysis is thus next carried out to see the cause of discrepancy in the minimized forms of F_1 using the two assignments.

In assignment-α, the set of states requiring adjacent codes for their merger in single cubes are (B, E, D), (E, C), (A, B, C, D, E) and (A, B, D). It can be seen that the given sets are unidistant with each other and with don't-cares resulting in a reduced expression of equation-4.5. The similar sets for assignment-β are (B,D,E), (A,D) and (E,C). However in the first set, assignment of state-E is not unidistant with either assignment of other states in the set or with don't-cares. Therefore minimized expression of F_1 is composed of four terms in equation-4.9.

The above example demonstrates the importance of state-codes in preserv-

ing inheritance for SAP. We next modify assignment-α into assignment-γ such that the original hamming distance between states is preserved.

Assignment-α is next modified so as the original hamming distance between states is preserved. One such possible assignment is assignment-γ in which assignment for only state-D is changed by swapping it with a previously unused assignment having similar number of ones. Thus, all the codes in assignment-γ remain either the same or are equivalent hamming distance apart as in assignment-α.

$$F_0 = I_0 * \overline{I_1} * F_1 * \overline{F_0} + I_0 * I_1 * F_0$$
$$F_1 = I_1 * F_1 * \overline{F_2} + \overline{I_0} * I_1 * \overline{F_0} + I_0 * F_1$$
$$F_2 = \overline{I_0} * I_1 * \overline{F_1} + I_0 * F_1 + \overline{I_1} * F_2 + I_0 * \overline{I_1}$$

Assignment-γ when synthesized yields a literal count of 24 literals which is again an inferior solution as compared to the original assignment-α. This is because although state codes with original hamming distances are retained in assignment-γ, the next state functions have changed. The new functions require different adjacency constraints that remain un-satisfied with assignment-γ.

The example can be summarized as follows:

- Initially, next state equations from assignment-α were tried to be optimized to yield assignment-β. However this also perturbed the hamming distances between states. The changed codes resulted in adjacency constraints remaining unsatisfied between states and yielding a solution with inferior cost.

- Next we tried preserving the original hamming distances in assignment-α by perturbing the solution to assignment-γ. But by doing so, the original next-state equations for whom the adjacency constraints were being satisfied got disturbed, resulting again in a solution with inferior cost .

It can thus be observed from the above example that inheritance in SAP is a complex function of both next state equations and state assignment. These two interact in an intricate way that makes it difficult to predict and preserve inheritance in SAP.

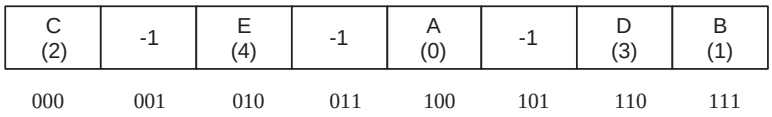


Figure 4.1: Chromosome Representation-1 for Code-α in State Machine-1

4.3 Genetic Algorithm

Genetic Algorithm (GA) (see section-2.6.1) has been applied to state assignment problem (SAP) for area minimization [25, 24]. The problem of state assignment has a search space of $S!$, where S is number of states, having many local minima [50].

As discussed earlier, Genetic algorithms need to preserve inheritance from previous generation while exploring the search space. Furthermore to efficiently explore and find an optimal solution, GA need to come out of any local minima.

This section discusses design of GA for state assignment problem. In particular, the various GA parameters that are utilized in this work are detailed.

4.3.1 Chromosome Representation

The first and foremost task in GA is how the problem is encapsulated in its chromosome. Chromosome representation is important as it decides efficiency of crossover and mutation operators, the two main tools for search space exploration in GA.

Two types of representations are utilized in this work that are suited for two types of crossover operators (discussed next) employed. The first representation considers a solution to be an array of states. A gene is a state placed at its code. An unassigned code or a don't care (DC) is represented with -1. For example, Code-α for state machine -1 is shown in Figure-4.1. Alphabetical state names are replaced with integer values, shown inside parenthesis in the figure.

In the second representation, each state code is described as an array of bits equal to the number of storage elements required. Code-α for state machine-1 is shown with second chromosome representation in Figure-4.2

	F_2	F_1	F_0
State-A	1	0	0
State-B	1	1	1
State-C	0	0	0
State-D	1	1	0
State-E	0	1	0
DC	0	0	1
DC	0	1	1
DC	1	0	1

Figure 4.2: Chromosome representation-2 for code-α in state machine-1

4.3.2 Crossover

Crossover operation is responsible for preserving inheritance from parents to the offsprings. Thus, a good crossover operator is essential for efficient exploration of solutions by GA. As discussed earlier, inheritance is difficult to preserve in SA problem. Therefore, a number of crossover operators are tried in this work. The operators utilized are next discussed.

PMX crossover

PMX crossover is a popular crossover operator that has been utilized previously in many problems [5]. PMX works by using a random cut-point on two parents such that all the genes before the cut-point in an offspring are taken from corresponding positions in parent-1 while parent-2 is utilized for the genes after the cut-point. If a gene in parent-2 has already been taken in the offspring, that gene is searched in parent-1 and its corresponding gene in parent-2 is selected for the offspring. The process is continued until the offspring is created with all unique genes.

Representation-1 of the chromosome is straightforward for PMX crossover and is thus utilized in this work. An example utilizing representation-1 of the chromosome for PMX is illustrated in Figure-4.3. Code-α and Code-γ for State Machine-1 are utilized for parents-1 and 2 respectively. A random cut-point is taken at position 5. To preserve uniqueness among don't care codes, a unique negative identifier is given to each don't care instead of homogeneously assigning -1 to all of them. The creation of the offspring is discussed next.

Initially part of the parent-1 before the cut-point is copied as it is into offspring. Next genes in parent-2 after the cut-point are scanned and new unique

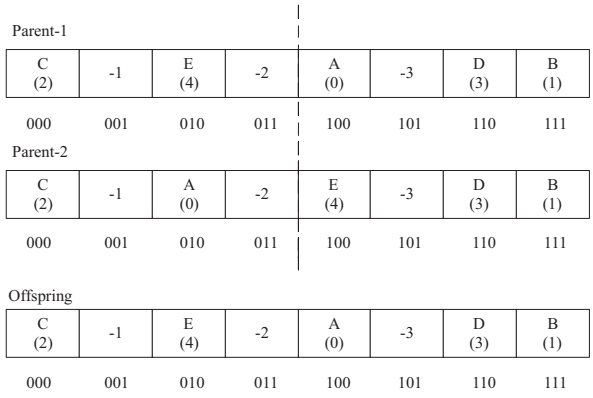


Figure 4.3: Chromosome Representation-2 for Code-α in State Machine-1

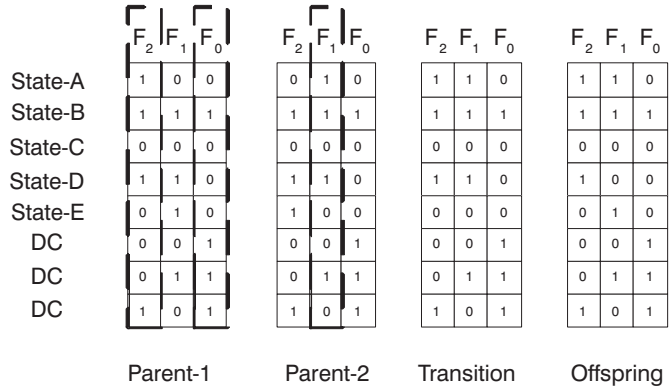


Figure 4.4: Amaral Crossover for Code-α in State Machine-1

gene are also copied in the offspring. For e.g. gene-B is copied at the last position. So far PMX has ensured that as many genes from selected parents retain their positions into the created offspring as possible. The remaining genes in parent-2 after the cut-point represent those that are already present in the offspring, for e.g. gene-E at position 5 is present in the offspring at position-3. This duplication of genes is avoided by selecting a gene from parent-2 at the same location where gene-E is present in parent-1. Thus gene-A is selected which being unique is selected into the offspring at position-5. The remaining genes are filled in a similar fashion.

Amaral crossover

Amaral et al in [25] described another crossover operator based on preserving next-state functions. Individual next-state functions are randomly selected

from any of the two parents to create the offspring. It then involves a post processing step to resolve any duplicates. Duplicates are resolved such that next-state functions derived from parent of higher fitness are not disturbed. Amaral's crossover is illustrated on Code-α and Code-γ for State Machine-1 in Figure-4.4. Next-State functions F_0 and F_2 are selected from parent-1 and F_1 from parent-2 to create transition chromosome. Transition chromosome has two duplicate pairs between states A and D and states C and E. Since parent-1 has got higher fitness (lesser cost), next-state functions for F_0 and F_2, which are taken from parent-1, are not disturbed. F_1 is randomly perturbed to yield the unique offspring.

Amaral's crossover selects random number of next-state equations from either parent. As shown previously, preserving inheritance for SA problem is very delicate job. Any large perturbation may totally disturb inheritance from either parent. To counter such a problem, minimum amount of perturbation is required. This can be achieved if only a single next-state equation is switched at a time. Such a minimal crossover is a subset of Amaral's crossover. The crossover example shown in Figure-4.4 is also an example for minimum crossover.

4.3.3 Mutation

The crossover operator explores the search space by keeping the parents characteristics. This may result in the search being stuck in a local minima when the diversity among parents reduces to such a level that no new offsprings can be created. To ensure such a situation arises only near to the optimal solution, new characteristics must be induced in the population. Mutation operator performs this crucial task by randomly selecting parents and changing characteristics of some of their genes. The amount of randomness is what is called mutation probability and is a GA parameter.

The mutation operator used in this work selects a parent based on mutation probability and swaps two codes in it. An assigned code represents a code assigned to a state. Unused state codes or don't cares like are unassigned codes. Mutation swap can take place between any combination of assigned or unassigned space of state codes. Thus mutation brings about new characteristics in a generation by assigning unused state codes.

Unlike crossover, the mutation operator discussed gaurantees uniqueness of created solution.

4.3.4 Parents Selection

The choice of parents for crossover from the set of individuals that comprise the population is probabilistic. In keeping with the ideas of natural selection, stronger individuals, that is those with lower cost values, are more likely to mate than the weaker ones. One way to simulate this is to select parents with a probability that is proportional to their cost values. That is, the smaller the cost of a certain chromosome, the grater is its chance of being selected as one of the parents for crossover. To accomplish this type of selection, roulette wheel method [5] is used in this work. In this method, a wheel is constructed on which each member of the population is given a sector whose size is proportional to the relative cost of that individual. To select a parent the wheel is spun, and whichever individual comes up becomes the selected parent. Therefore, in this method, individuals with higher cost values also have a finite but lower probability of being selected for crossover.

4.3.5 Next Generation Selection

To keep the number of members in a population fixed, a constant number of individuals are to be selected after every generation from the pool of previous generation members and newly created offsprings. The selection can be totally greedy in nature, selecting the best in the whole set, or even random. While Greedy selection has the benefit of preserving qualities of best individuals in the next generation, a random selection has the advantage of improved variety in the available set while also being quicker to perform. In this work, a combined selection criteria is employed that integrates merits of both the selection policies by being greedy for upper half-set and random in selection of lower half-set of next generation members.

4.3.6 Uniqueness of Offsprings

Increasing the diversity among the generation improves the probability of getting a good solution in a smaller number of generations. The crossover or mutation operations can result in duplicate offsprings. These duplicate offsprings are discarded and only a unique offspring is entered in a new generation.

4.4 Tabu Search

Tabu Search (TS) algorithm (see section-2.6.2) is a non-deterministic iterative heuristic that has previously been successfully applied in solving combinatorial optimization problems in different fields [5]. Tabu Search has previously shown to outperform the solution quality when compared to GA and as such there exists a good potential in utilizing TS for SAP. However, there is no work as yet available in the literature that utilizes TS for FSM state assignment problem.

In this section, the various parameters utilized in TS are discussed in solving the objectives of this work.

4.4.1 Tabu Move and Solution Representation

A tabu move consists of a small perturbation in the current solution to explore solutions that are adjacent to it, i.e. in the neighborhood of the solution. An efficient tabu move is one that possess the diversity as well as ease of evaluation to quickly explore neighborhood of a solution.

There can be several types of tabu-moves for FSM state assignment problem. One strategy could be based on having minimum perturbation of a single bit in a state code. Such a move will require binary representation of a solution (Chromosome in GA terminology) similar to one shown in Figure-4.2. A move will then consist of swapping of states that are unidistance apart.

A second type of tabu-move may involve swapping of two columns of the current solution. However both the strategies would quickly saturate as there are much lesser number of combinations to swap than the possible neighborhood size. They thus lack the diversity needed in a tabu-move to efficiently search the neighborhood.

Another type of tabu-move could involve swapping codes of two randomly picked states. The first form of chromosomal representation, as shown in Figure-4.1, is ideally suited for evaluating such a move. The move will then consist of swapping genes at two randomly picked codes if at least one of the genes being swapped is a state. The move is thus quick to evaluate as it by default takes care of uniqueness of state-codes in a resulting solution. Moreover, such a move on average will perturb half the number of columns for any state, thus having more diversity as compared to the previous two approaches. This work thus utilizes the third type of tabu-move for exploring the neighborhood of a solution.

4.4.2 Neighborhood Size

Neighborhood or candidate list size decides the amount of search space to be explored in the neighborhood of a solution. It thus holds a key place in the search space exploration by TS algorithm. A small neighborhood size (NS) of less than 10 solutions is usually considered as a good figure. A small neighborhood also ensures quicker iterations where the saved time can be used to increase the number of generations, thus trading space saved from smaller NS with iterations/time.

A bigger NS provides bigger exploration space at a cost of increased computation overhead. A bigger NS may ensure arrival of the same solution in lesser number of iterations. It is theoretically possible that the same solution is reached in equivalent amount of time by either using smaller NS and higher number of iterations or bigger NS and smaller number of iterations. An optimal NS may thus have to be empirically evaluated.

4.4.3 Tabu List Size and Tabu Specification

Tabu specification is an identification signature/mark of a previously explored solution. This mark is stored in tabu-list and tabus any new solution that has the identical mark. The idea is to avoid repetition of similar moves, as they might cycle the current solution back to a previously reached solution, as well as to encourage new moves in a sequence of solutions. The length of this sequence is equal to the tabu list size which is implemented as a queue (LIFO). The idea behind favoring new moves is to have more diversity by perturbing unperturbed characteristics (genes) in a solution (chromosome).

This work utilizes one of the states swapped during the move as tabu-mark. A Tabu-list size of 7 happens to be a favored figure using TS algorithm [5] and thus is also used in this work. However, there can be a potential problem using a fixed tabu-list size for SAP. Consider an FSM having lesser than 7 states. In such a case, there can occur a situation where all the states get tabued. This will cause the algorithm to be totally greedy in selection of the next solution, i.e. solely relying on aspiration criteria. This deadlock situation is handled in the current work by slightly modifying the algorithm. The modified algorithm accepts any random solution in the neighborhood if faced with three consecutive tabus. After the selection, the algorithm resets its memory to repeat the modification if again faced with three consecutive tabus. The rest of the algorithm remains the same.

4.4.4 Aspiration Criteria

A standard aspiration criteria is used in the work. The criteria allows a tabued solution to be selected if it has a better cost than the current solution.

Chapter 5

Experimental Results and Analysis

5.1 Introduction

In this chapter, results obtained from different optimization techniques as described in chapter-3 are reported and compared with those given in the literature (see chapter-2). The algorithms and cost functions are evaluated on industry-standard MCNC/LGSynth'89 FSM benchmark suite [95].

The chapter begins by discussion of the simulation environment and benchmark circuits used for experimentation. This is followed by evaluation of genetic and tabu-search algorithms for state assignment problem as discussed in chapter-4. Thereafter, evaluation of cost functions for area, power and testability for single and later multi-objective optimization is presented along with comparison between the search algorithms employed in this work. The chapter also compares the results of single and multi-objective optimizations achieved using the proposed techniques with those reported in literature.

5.2 Simulation Environment

The algorithm and cost functions are coded using C/C++ language simulated on multi-user SUN system using 648 MHz UltraSPARC-IIe processor, 1GB RAM and 512 KB cache. The evaluation is done by using SIS [35], ESPRESSO [27] and HITEC [91] tools to find the actual cost for the best solution of the cost/fitness function utilized. Simulations are terminated either after prescribed number of generations or on achieving a termination condition.

The termination condition used terminates the simulations either if the

number of non-unique discarded offsprings exceeds 500 in a single generation or the average population fitness does not increase for 40 consecutive generations. In the case of tabu-search, simulations are stopped if no new better solution is found for 40 consecutive iterations after reaching ceiling for adaptive neighborhood-size.

5.2.1 Benchmarks

The work utilizes 20 benchmark circuits of varying sizes and complexity from MCNC/LGSynth '89 benchmark suite [95] for evaluation and comparison of cost functions and algorithms employed in this work. The selection provides flavor of circuits of different complexities that are also utilized in previous works. Details of the circuits used in this work are shown in Table-5.1.

Benchmark	States	Inputs	Outputs
bbara	10	4	2
bbsse	16	7	7
cse	16	7	7
dk14	7	3	5
donfile	24	2	1
ex2	19	2	2
ex3	10	2	2
keyb	19	7	2
lion9	9	2	1
planet	48	7	19
pma	24	8	8
s1	20	8	6
s1494	48	8	19
s832	25	18	19
sand	32	11	19
shiftreg	8	1	1
styr	30	9	10
tbk	32	6	3
traian11	11	2	1

Table 5.1: Benchmarks Statistics [95]

5.3 Genetic Algorithm

This section evaluates the performance of various genetic operators utilized in Genetic Algorithm (GA) as discussed in section-4.3. To abstract any anomaly based on inaccurate cost function, an exact cost measure using ESPRESSO single output minimization followed by fast extraction (fx) is utilized. Initially, optimum mutation rate is evaluated by fixing population and generation sizes. This is followed by experimentation to select a good population size by fixing generation size. Once population size is known, it is then used to evaluate a good generation size. These parameters are thereafter used in experimentation employing GA in the rest of the thesis.

5.3.1 Mutation Rate

The mutation rate (M_r) as suggested in the literature [5] is to be within the range of 1% to 5% such that the total number of mutations in a generation are approximately around $M_r.M.n$, where M and n being the population size and number of genes affected by mutation. However, as n is small in the present case, i.e. 2, a suitable mutation rate is evaluated through experimentation. The experiments were carried on four benchmark circuits and their average and best costs were analyzed using a generation size of 250 and a population size of 64 to experiment with 6 different mutation rates. Table-5.2 tabulates the results.

M_r	1	5	10	15	20	25
bbara	80.22/87	90.14/89	87.01/85	87.50/85	87.37/85	88.17/85
bbsse	167.47/165	161.91/157	164.01/161	164.85/160	163.74/160	164.57/160
ex2	141.83/130	170.71/151	169.89/163	171.53/157	174.73/156	177.60/166
train11	26.084/24	25.412/23	23.644/22	23.516/21	25.32/21	27.324/26
planet	994.79/970	1013/970	981.28/953	967.93/920	962.82/902	1038.12/1015

Table 5.2: Mutation Rates (Average/Best)

It can be observed from the above table that a mutation rate of 20 assures the best solution in most of the cases. Moreover, the average cost is also the best in two of the cases, bbara and bbsse, with a slight deterioration in ex2 and train11 circuits. However, any further increase in mutation rate adversely affects average cost and solution quality. This is because too many solutions get disturbed due to which inheritance of genetic gets negatively affected. Thus, a mutation rate of 20 is a good choice and will be used in the rest of the experiments using GA.

5.3.2 Effect of Population Size

Four different population sizes as shown in Table-5.3 are selected to observe their effect on quality of solution. Generation size is fixed to 250. Numbers in brackets in the table represent simulation time that denotes the earliest time in reaching the best solution.

Benchmark	4	8	16	32	64	128
bbara	75(12.73)	59(20.21)	57(34.59)	54(44.52)	51(12.03)	49(133.28)
bbsse	138(2.23)	118(19.73)	105(18.62)	105(36.4)	97(32.88)	97(171.49)
donfile	191(2.87)	148(19.47)	138(44.04)	92(69.26)	68(87.37)	53(147.63)
ex2	159(11.16)	117(15.94)	109(51.52)	80(40.23)	66(136.15)	73(216.63)
keyb	264(27.83)	232(847.81)	167(82.33)	143(110.85)	156(207.92)	151(561.13)
lion9	25(14.16)	10(7.58)	12(11.04)	10(19.7)	10(4.98)	10(65.03)
planet	600(23.69)	541(83.73)	516(110.06)	493(165.95)	472(411.38)	439(753.52)
sand	631(48.96)	578(134.88)	513(153.89)	487(204.99)	473(496.3)	460(928.37)
shiftreg	13(1.02)	2(5.37)	2(0.12)	2(1.13)	2(1.2)	2(1.47)
styr	652(43.24)	525(129.49)	469(146.26)	430(222.55)	423(349.34)	418(782.71)
traian11	34(17.94)	24(17.71)	21(17.3)	20(27.73)	18(12.48)	18(89.65)

Table 5.3: Effect of varying population size

It is observed from the table that literal count decreases by less than 10 literals in 8 out of 11 circuits when population size is increased from 64 to 128. The three circuits where improvement of more than 10 literals is seen, donfile, planet and sand, comes at a cost of increase in simulation time overhead by 68%, 83% and 87%, respectively. However, there is notable literal count decrease in 4 of the circuits with also low simulation time overhead when going from 32 to 64 generation size. Thus, population size of 64 gives a reasonable tradeoff between quality of the solution versus cost paid in terms of simulation time and will therefore be used in rest of the experimentations.

The search space in FSM state assignment is characterized with many local minimas; some of them being very difficult to get out from. A higher number of generations or a bigger population size are two tools available in genetic to come out of such minimas. Thus on average, a higher population size yields better solution quality. However, there is no such guarantee as can be seen from the circuits, ex2, keyb and lion9, where the cost increased with increasing population size.

5.3.3 Effect of Generation Size

In this subsection, the population size of 64 is used for exploring proper generation size for further experiments. The simulation results for six different

generation sizes is shown in Table-5.4. The simulation time reported in brackets is the time after the corresponding number of generations.

Benchmark	50	100	200	350	500	1000
bbara	51(16.84)	51(30.27)	51(56.91)	51(97.08)	51(143.4)	51(295.1)
bbsse	100(16.45)	99(32.18)	97(67.83)	93 (121.83)	93(194.82)	93(421.97)
donfile	121(25.65)	97(45.04)	68(98.55)	61(175.24)	52(242.71)	49(424.99)
ex2	103(30.07)	87(53.24)	72(106.76)	66(194.67)	66(286.53)	66(582.59)
keyb	185(91.53)	162(131.28)	161(188.32)	143(273.95)	134(382.16)	134(768.17)
lion9	10(6.61)	10(17.31)	10(34.91)	10(61.34)	10(83.66)	10(167.84)
planet	567(150.1)	503(233.72)	476(410.85)	468 (621.14)	458(891.04)	458(1768.32)
sand	537(107.46)	505(207.06)	473(511.57)	473(993.3)	473(1458.88)	473(3010.25)
shiftreg	2(1.08)	2(2.33)	2(4.66)	2(6.7)	2(11.65)	2(23.3)
styr	515(155.94)	477(260.34)	423(398.83)	423(664.63)	420(895.5)	379(1732.25)
traian11	18(11.14)	18(20.67)	18(35.3)	18(62.21)	18(88.85)	18(177.77)

Table 5.4: Effect of varying generation size

By following the same reasonings used for selection in proper population size, a proper generation size can be evaluated by comparing the literal count decrease against the processing time cost it incurred. It is observed that from generation size 100 to 200 that 160 literals were saved at an average processing cost of 5.34 seconds per saved literal. Similar costs between generation size 200 and 350 and between 350 to 500 increased to 32.27 seconds and 45.4 seconds respectively. Besides increase in processing times, the number of circuits showing improvements also gradually decreased from 7 between 100 and 200 generation sizes to 5 and 4 circuits between 200 and 350 and 350 to 500 respectively with maximum number of literals saved not being more than 10 in the latter. Generation size of 350 is seen to be a reasonable tradeoff between number of literals being saved, number of circuits offering improvement and the processing cost overhead. Generation size of 350 is thus selected for the rest of the experimentations involving GA.

5.3.4 Crossover Operators

The crossover operators, PMX and Amaral, discussed in section 4.3.2 are next evaluated in this subsection using the generation and population sizes selected above. The results obtained are tabulated in Table-5.5. The values represent the quality of solution obtained after 250 generations using the corresponding crossover. Values in the brackets represent simulation time.

It is seen from the above table that Amaral crossover reaches to better solutions in equal number of generations whereas PMX operator saturates quickly. The observation be observed in more detail from the plot of generation runs using the two operators as shown in Figure-5.1 to Figure-5.4.

Benchmark	Amaral	PMX
bbara	51(34)	57(35.79)
bbsse	103(33.8)	102(50.69)
donfile	68(46.51)	99(52.16)
ex2	66(52.17)	99(63.84)
keyb	156 (142.36)	173(517.34)
lion9	10(25.1)	15(25.24)
planet	472(513.56)	532(153.29)
sand	473(673.14)	513(142.94)
shiftreg	2(5.825)	2(5.73)
styr	423(472.84)	489(237.04)
train11	18(47.01)	21(22.66)

Table 5.5: Amaral Vs PMX comparison for 250 generation size

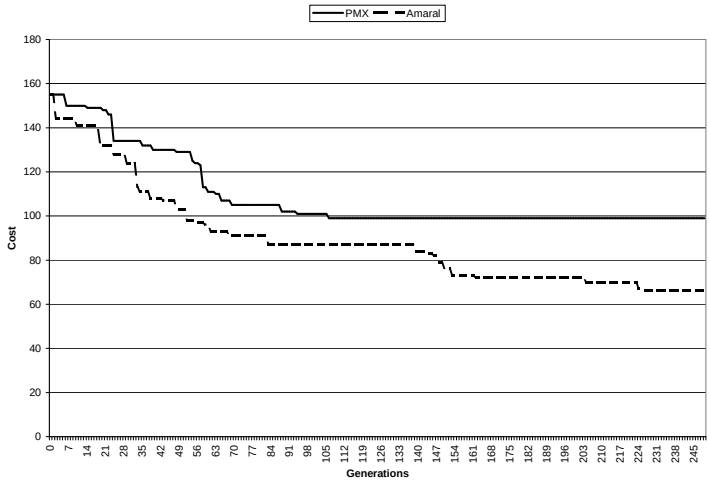


Figure 5.1: Amaral vs PMX on ex2 circuit

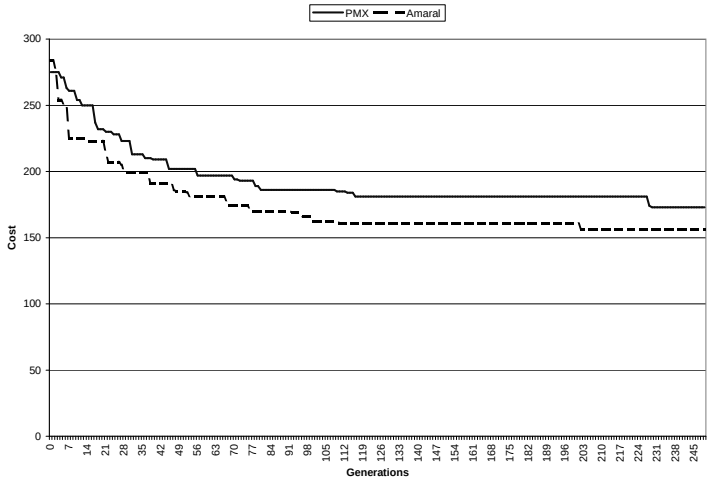


Figure 5.2: Amaral vs PMX on keyb circuit

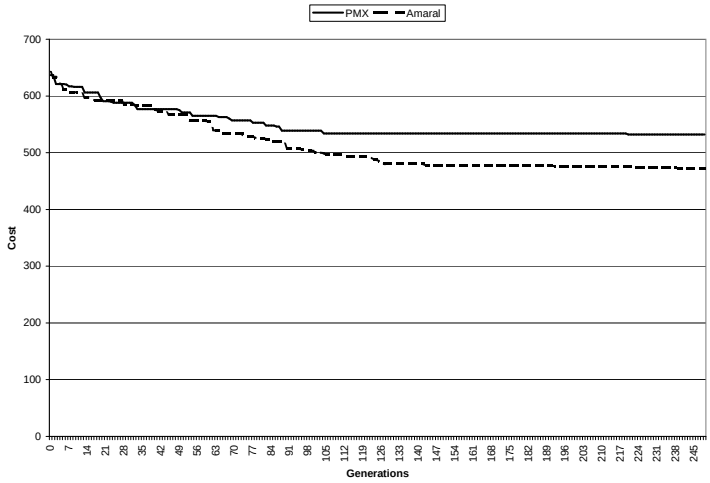


Figure 5.3: Amaral vs PMX on planet circuit

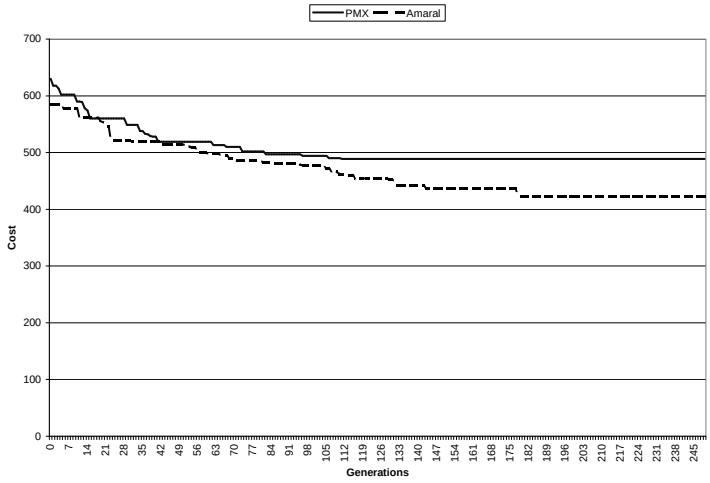


Figure 5.4: Amaral vs PMX on styr circuit

The number of thrown away offsprings are next compared between Amaral and PMX crossovers. Graphs plotting the number of thrown away offsprings between the two crossover operators with increasing generations are given from Figure-5.5 to Figure-5.7.

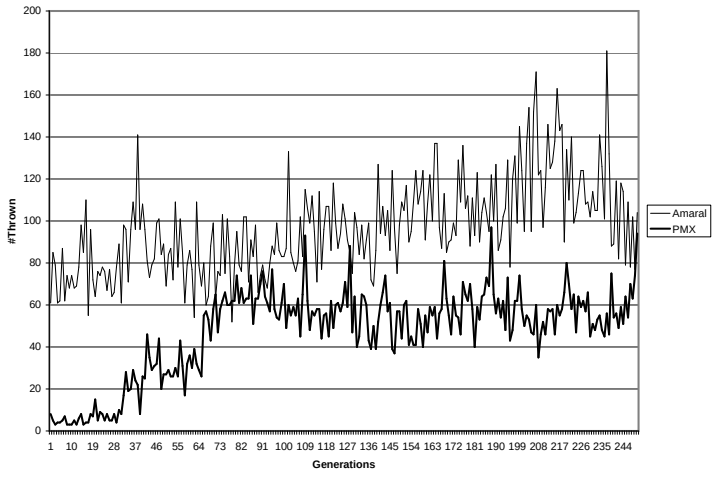


Figure 5.5: Amaral and PMX #thrown offsprings comparison on ex2 circuit

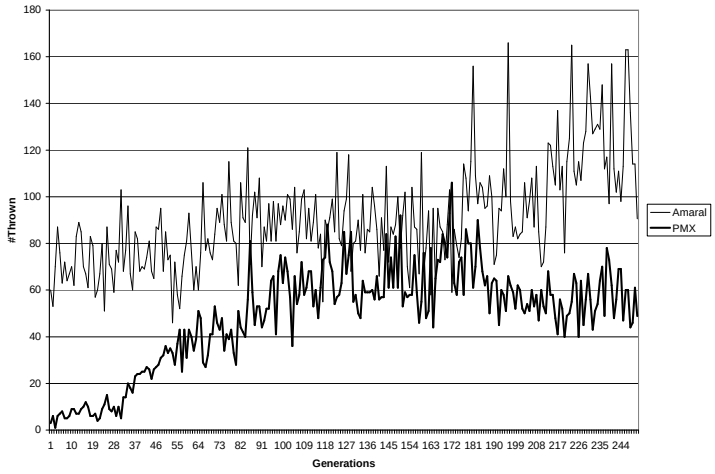


Figure 5.6: Amaral and PMX #thrown offsprings comparison on keyb circuit

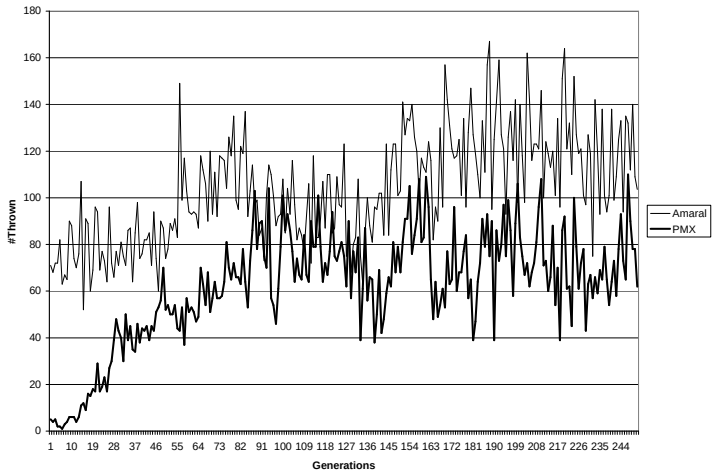


Figure 5.7: Amaral and PMX #thrown offsprings comparison on styr circuit

The thrown away offsprings comparison show that Amaral crossover has more duplicated offsprings than PMX crossover. The reason being the degree of diversity being offered in the two crossovers. While PMX crossover can take the same two parents and generate a number of different offsprings depending on the random cut point, there is lesser such choices available in

Amaral's crossover. The number of choices available for random cut point in PMX being $2^n - 1$ where n is the number of sequential elements or available choices for Amaral's crossover. Thus Amaral's crossover has approximately $log_2(States)$ lesser options than PMX that results in frequent repetition of generated offsprings. An increase in thrown away offsprings translates into increased processing overheads that as can also be seen in Amaral crossover's results in Table-5.5.

To gain a more useful insight in selection of a crossover, simulation time is next fixed to compare the quality obtained from the two crossovers at similar time instant. The time chosen is the lesser of the two times in Table-5.5. For e.g, time 34 seconds is the quicker of the two times in reaching a solution for bbara circuit by either Amaral or PMX. Thus for a fair comparison, the solution obtained at time instant 34 seconds by Amaral is to be compared with a solution obtained by PMX at the similar time. Such a comparison is presented in Table-5.6

	Amaral	PMX
bbara	51	57
bbsse	101	102
donfile	68	99
ex2	66	99
keyb	156	193
lion9	10	15
planet	529	532
sand	524	513
shiftreg	2	2
styr	477	489
train11	18	21
Average	182	192.9091

Table 5.6: Amaral Vs PMX quality comparison at similar time instants

It can be observed that Amaral crossover at similar time instants is saving nearly 10 literals on every circuit. Therefore, it can now be concluded that Amaral crossover is more efficient in search space exploration than PMX crossover. Amaral crossover is thus selected for use as GA crossover in the rest of the experimentations.

5.4 Tabu Search

This section presents results and analysis of experimentation done on Tabu Search (TS) algorithm as discussed in section-4.4. An optimal neighborhood size is empirically evaluated which is later used in the thesis for experimentation on the objectives of this work. The section concludes with a brief comparison of TS with GA with detailed comparisons left for later sections of this chapter.

5.4.1 Neighborhood Size

Neighborhood sizes of 8, 32, 64 and 128 are experimented with in this section. The work also experiments with an adaptive neighborhood size that adapts NS according to the difficulty in overcoming a local minima. Graphs showing the results of varying NS on two circuits, keyb and planet, are plotted next. For clarity purposes, simulation plot for keyb circuit is duplicated with a zoomed-in plot of early stages of the algorithm shown in Figure-5.8 whereas the complete graph is given in Figure-5.9. Simulation plot for planet circuit is given in Figure-5.10. Expand-SO is used as the cost function. The graphs are normalized along time scale. Thus the different curves representing different neighborhood do not actually scale to similar number of iteration sizes.

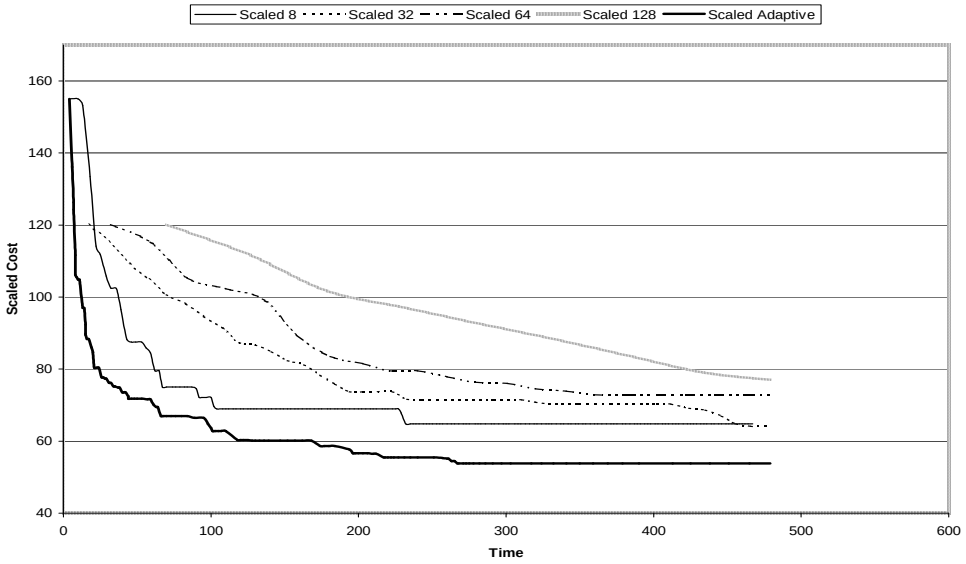


Figure 5.8: Effect of Neighborhood Size on Keyb circuit - 1

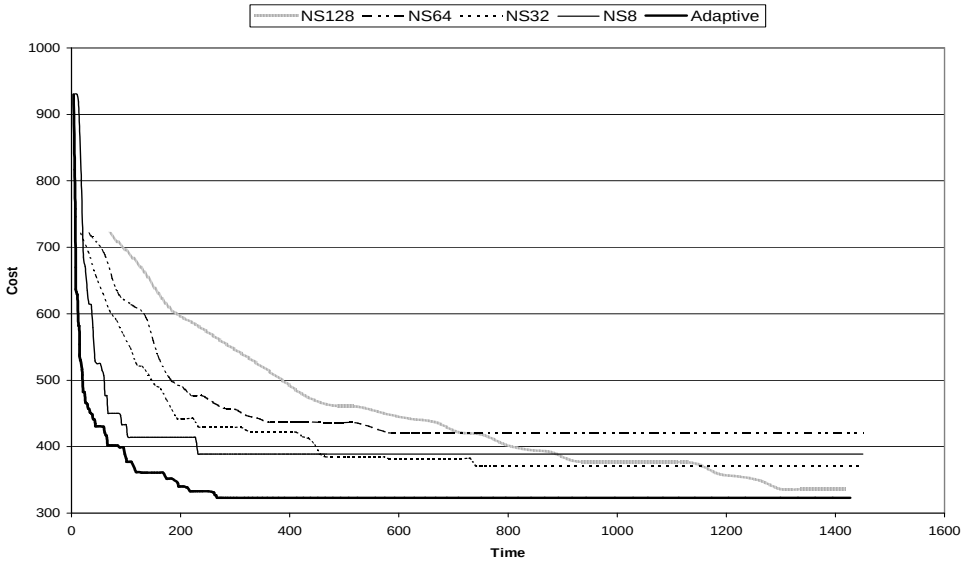


Figure 5.9: Effect of Neighborhood Size on Keyb circuit - 2

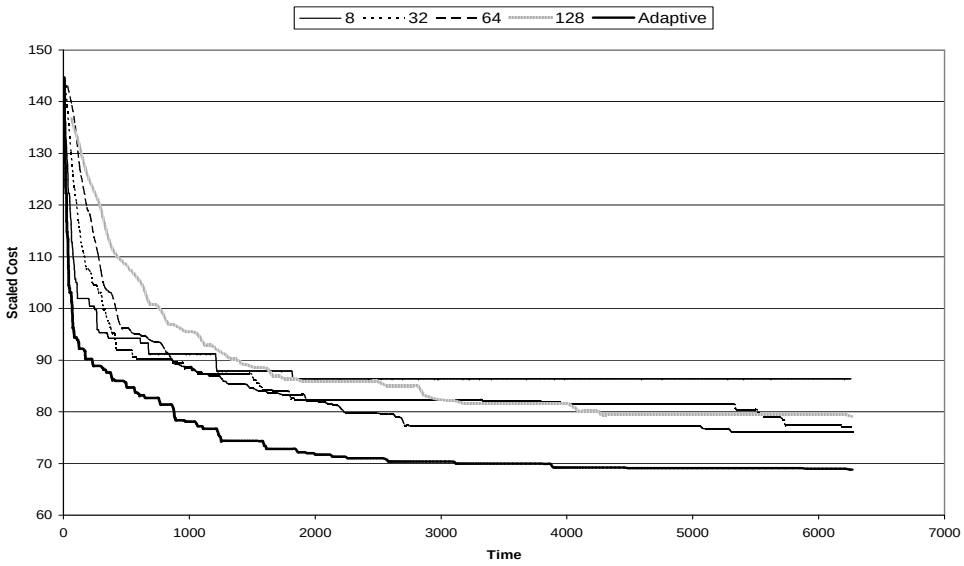


Figure 5.10: Effect of Neighborhood Size on Planet circuit

It can be observed from the above graphs that different neighborhood sizes vary in the quality of results when simulated for equivalent amount of time.

A conclusive argument can also not be built for any particular NS. A small NS is quicker in iterations and so quickly explores better new solutions at early stages of the algorithm. However, it has lesser capability of escaping a local minima and thus saturates quickly as well. On the other hand, a bigger NS, though having an increased processing overhead, is also more capable of search space exploration and outperforms smaller NS in the longer run.

The above observations are combined into an adaptive NS where the NS increases or decreases according to the difficulty of overcoming the local minima. The adaptive NS shown in the above graphs is based on this principle. An upper limit of NS of 512 and a lower limit of 8 is used in adaptive approach. The adaption increases NS in steps of 8 if a consecutive iteration does not lead to a better solution. However, on achieving a new better solution, NS is immediately halved its current size. The adaption mechanism is based on the idea that there are few local minimas that require excessive effort. Once these minimas are overcome, there is a higher probability that better solutions are easier to reach. This principle can be observed in more detail by analyzing plots of NS with solution quality. Figure-5.11 and Figure-5.12 represent such plots for the keyb and planet circuits.

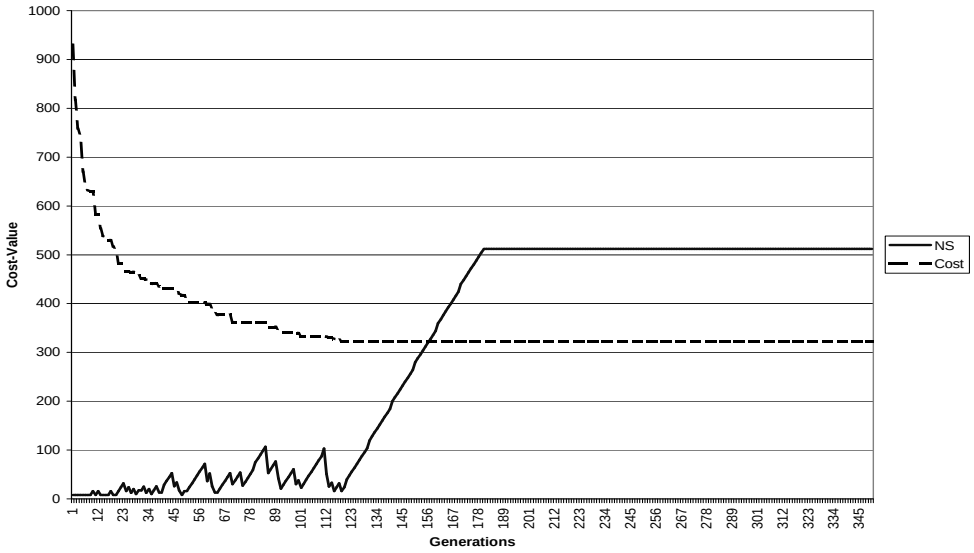


Figure 5.11: Performance of adaptive NS on keyb circuit

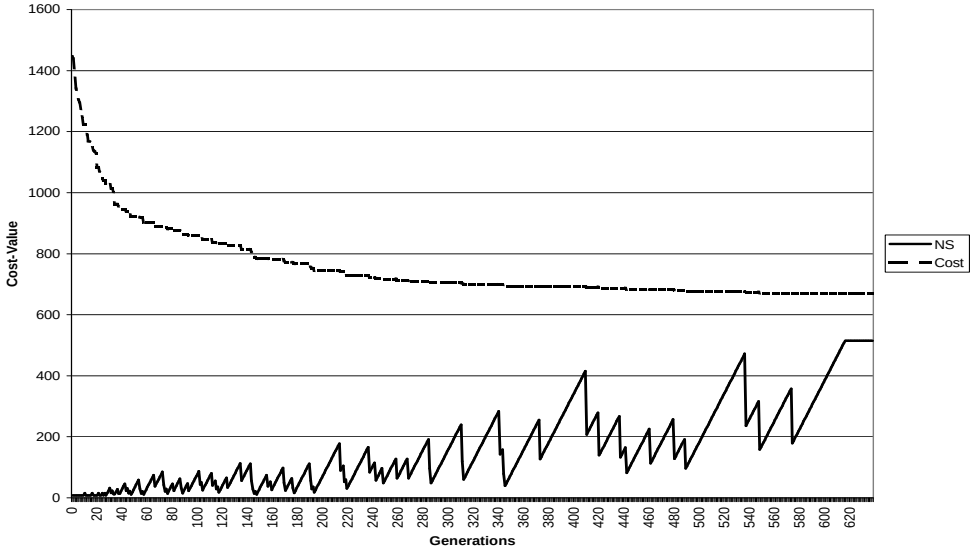


Figure 5.12: Performance of adaptive NS on planet circuit

An adaptive NS is selected to be used in the rest of the experiments employing Tabu Search in this work. It can be seen from the above graphs that there is little improvement in solution quality by having large NS. As larger NS is computationally expensive to evaluate, a moderate size of 128 is seen to be appropriate to be used as upper limit for NS.

5.4.2 Performance Comparison of TS and GA

A comparison of TS with GA is next presented using Expand-SO as cost measure. The comparison is presented in Figure-5.13 and Figure-5.14

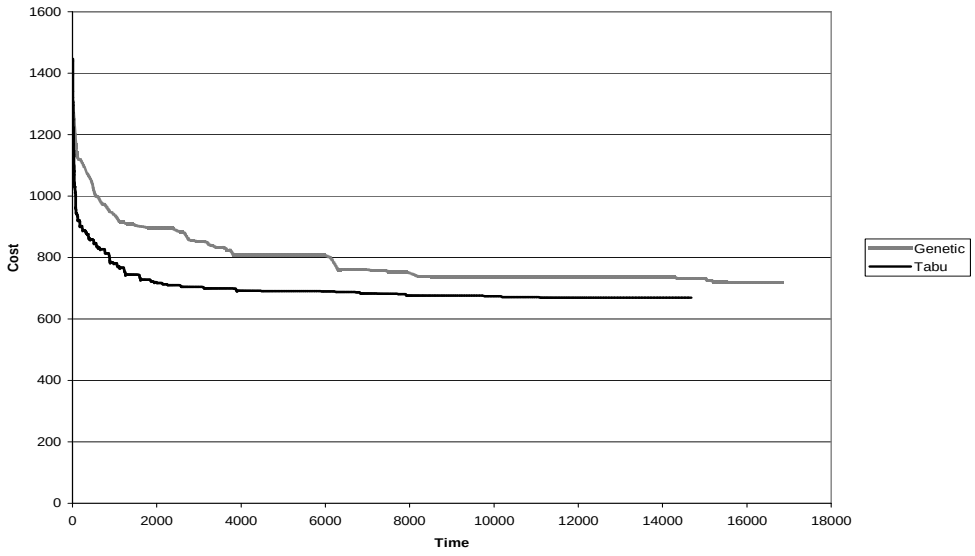


Figure 5.13: TS vs GA on planet circuit

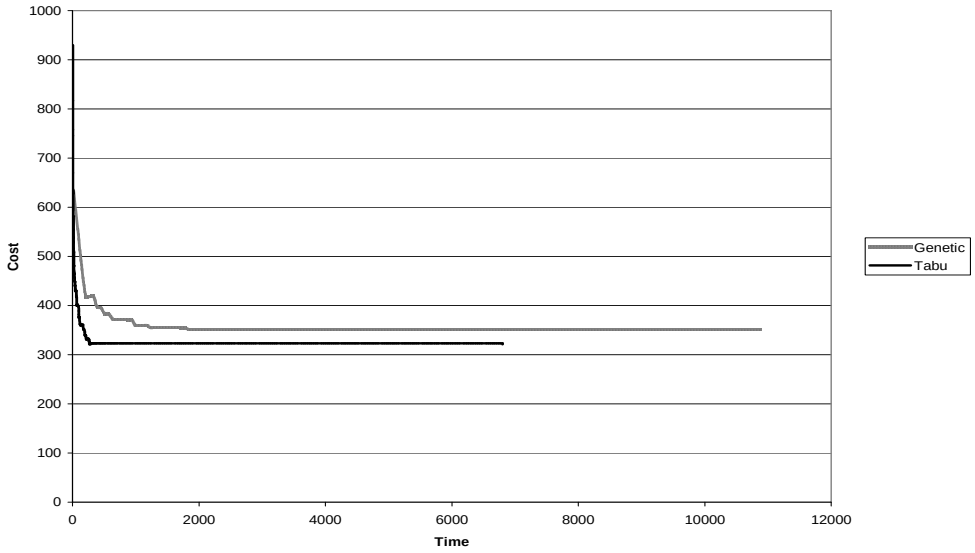


Figure 5.14: TS vs GA on keyb circuit

It is seen that TS quickly achieves solutions of better quality than GA whereas GA sometimes is not able to meet the same solution quality even if simulated for extended amount of time.

5.5 Area

This section discusses the performance of various area cost measures that were described in chapters 2 and 3. The analysis is done by synthesizing and evaluating actual literal count of the best solutions obtained using the cost measures. The values so obtained are then compared with measures reported in the literature. The section also details on the degree of accuracy of the different cost measures by plotting their correlations with actual cost.

A comparative table between different cost functions is given in Table-5.7 and Table-5.8. Values in the table represent number of literals after synthesizing the best solutions obtained using GA. The synthesis is carried out by ESPRESSO single output (SO) minimization followed by fast extraction (Fx). Accurate cost measure tabulated in the last column of the table uses Espresso-SO followed by Fx as cost measure.

Benchmark	Jedi	Mustang	Armstrong	LS1	LS2	LS3	LS4
bbara	83	64	59	65	81	73	92
bbsse	167	140	127	146	165	149	185
dk14	137	128	124	145	126	130	142
donfile	235	136	171	121	289	219	280
ex2	184	155	138	131	190	188	182
keyb	327	179	334	358	350	184	379
lion9	57	43	27	40	46	34	35
planet	631	669	607	669	710	670	740
sand	655	602	619	565	645	605	751
shiftreg	21	13	2	14	21	2	8
styr	618	498	546	580	705	573	734
train11	52	54	32	39	80	46	57
Average	263.9167	223.4167	232.1667	239.4167	284	239.4167	298.75

Table 5.7: Comparison of Cost Functions-1

Benchmark	Expand-MO	Expand-SO	Accurate
bbara	57	56	51
bbsse	120	109	103
dk14	115	104	98
donfile	106	87	49
ex2	130	78	66
keyb	161	199	134
lion9	25	11	10
planet	557	439	439
sand	514	492	473
shiftreg	4	2	2
styr	466	405	379
train11	29	20	18
Average	190.333	166.833	151.833

Table 5.8: Comparison of Cost Functions-2

A cost function can be considered a good estimate of the actual cost if it possesses correlation with the accurate cost. In other words, the actual cost shows proportional deviations with the corresponding fluctuations in estimated

cost. Cost-function correlation is an important aspect in the design of a cost-function, particularly in evolutionary heuristics like genetic algorithms that proceed by retaining good characteristics from previous generations.

Correlation of the cost measures used are therefore next evaluated to estimate their accuracy with actual cost. This is achieved by taking a number of random solutions for a cost function and sorting them based on their actual cost. The actual cost is again obtained by synthesizing the circuit obtained using the solutions' state-codes using Espresso-SO followed by quick factorization. Figure-5.15 to 5.38 plot the correlation graphs of the used cost measures.

Support (dependencies) of a flip-flop as discussed in section-2.1 is also utilized in pursuit of development of a cost measure for multilevel area estimation. The correlation graphs plotted for analyzing support function dependency employ summation of input and flip-flop dependencies on all the flip-flops and outputs of a circuit. The graphs are shown in Figure-5.39 to Figure-5.41.

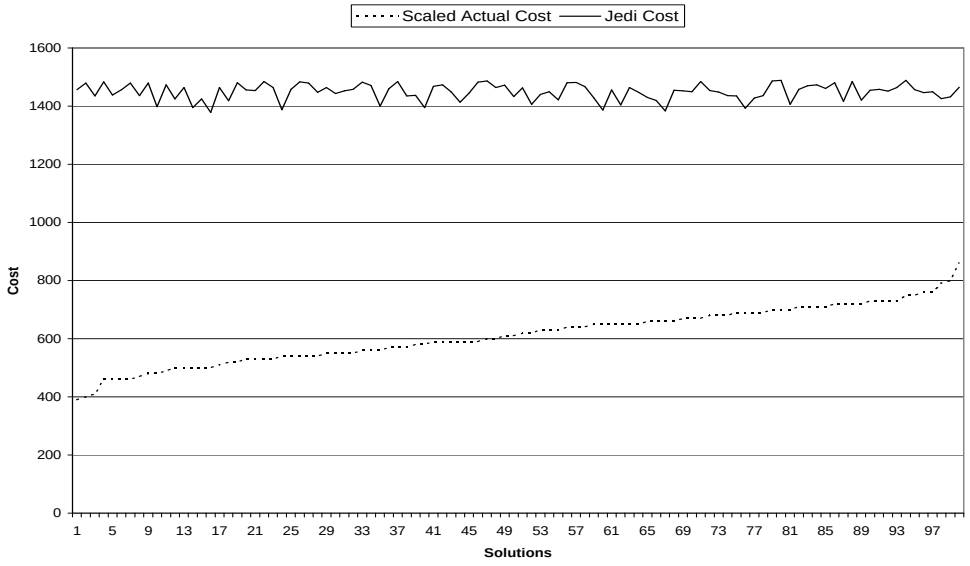


Figure 5.15: Jedi Correlation on train11 circuit

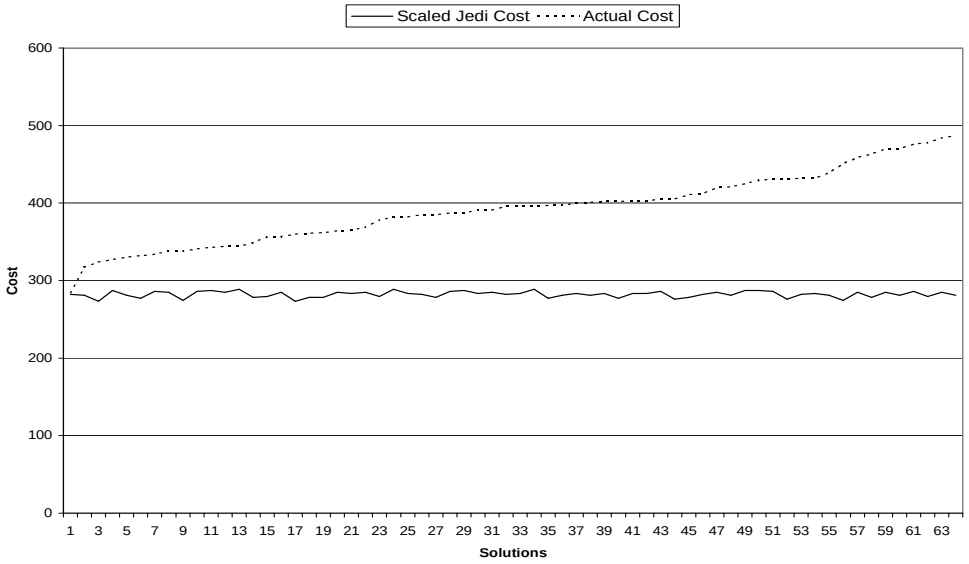


Figure 5.16: Jedi Correlation on keyb circuit

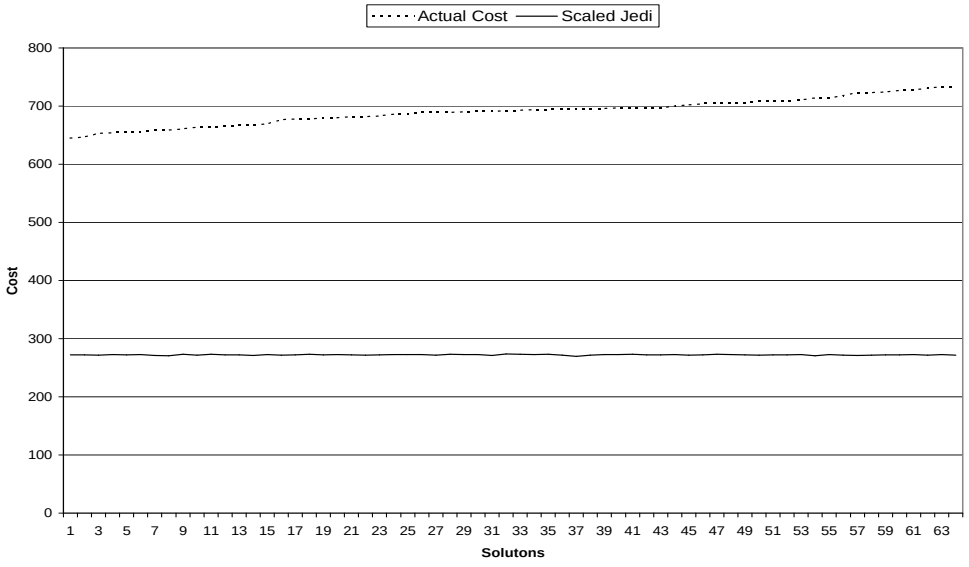


Figure 5.17: Jedi Correlation on planet circuit

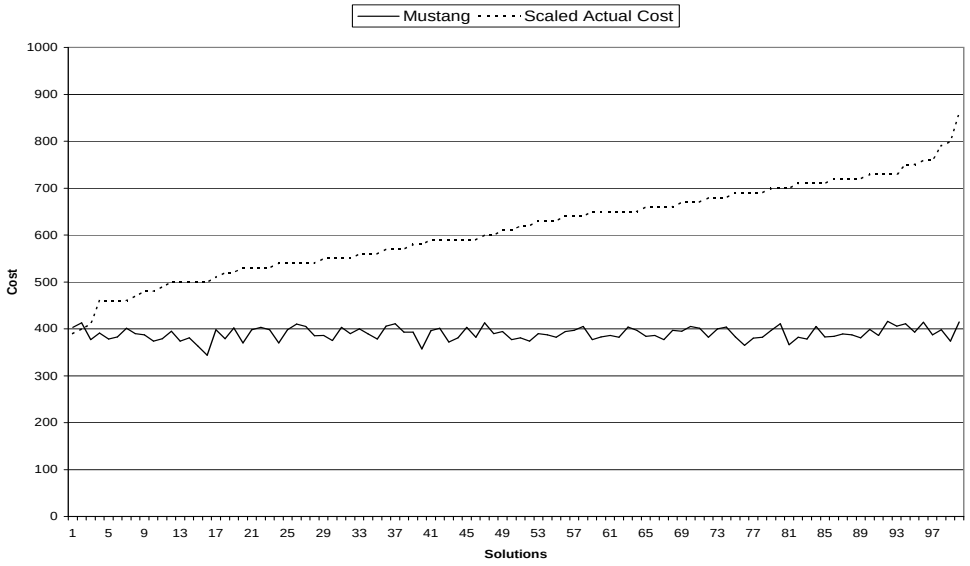


Figure 5.18: Mustang Correlation on train11 circuit

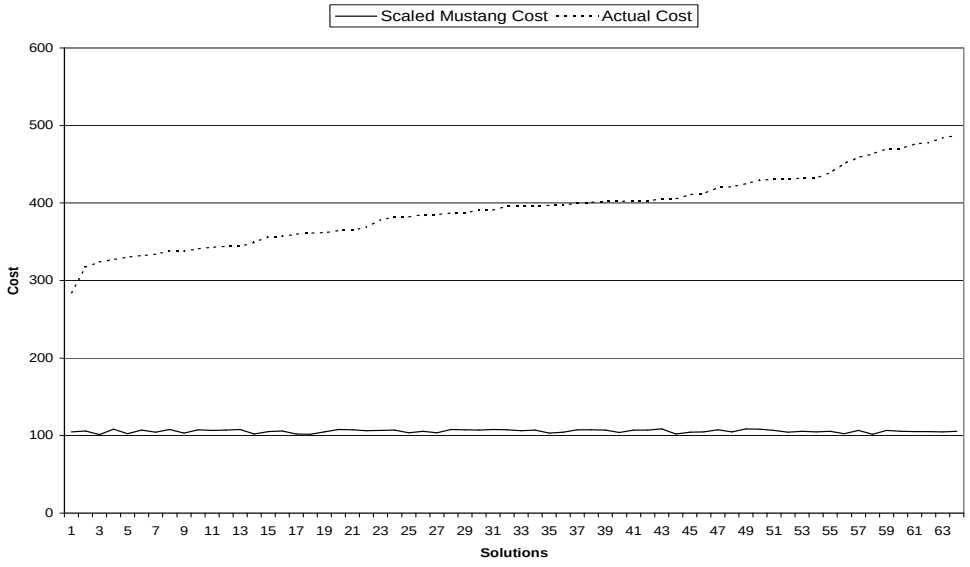


Figure 5.19: Mustang Correlation on keyb circuit

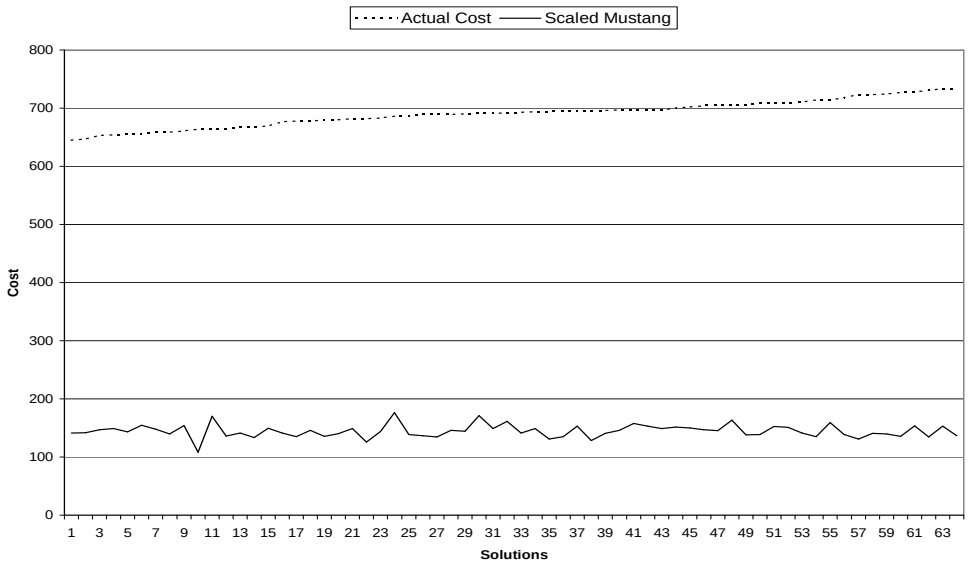


Figure 5.20: Mustang Correlation on planet circuit

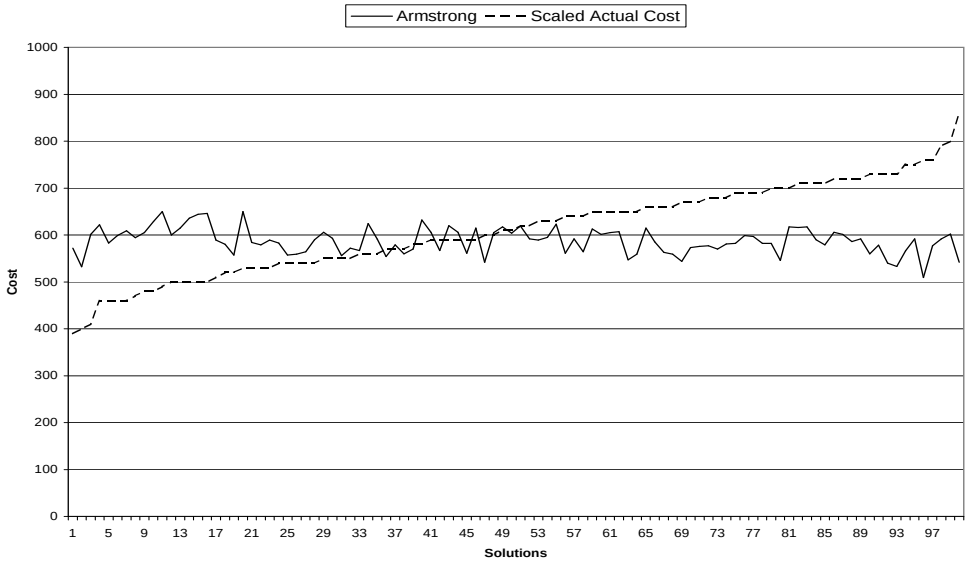


Figure 5.21: Armstrong Correlation on train11 circuit

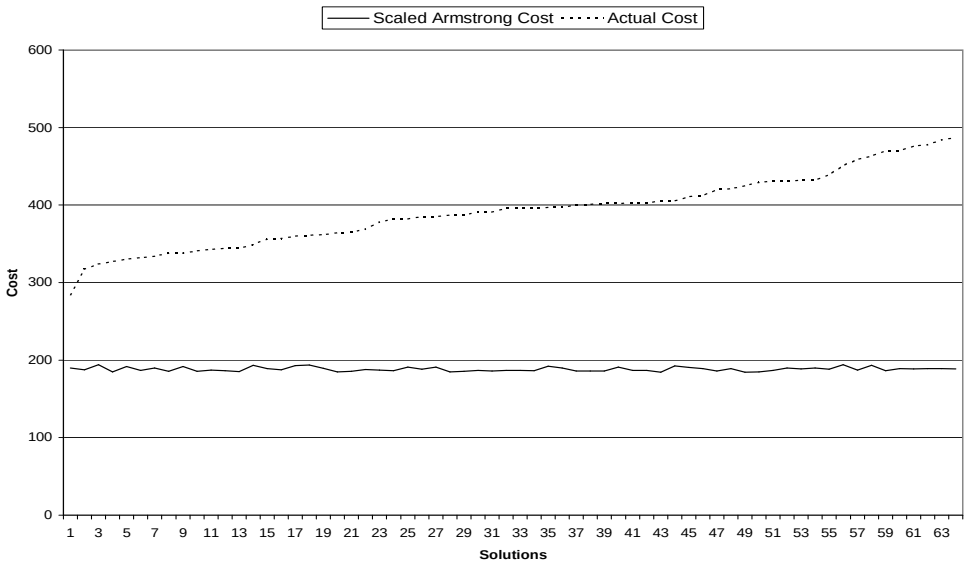


Figure 5.22: Armstrong Correlation on keyb circuit

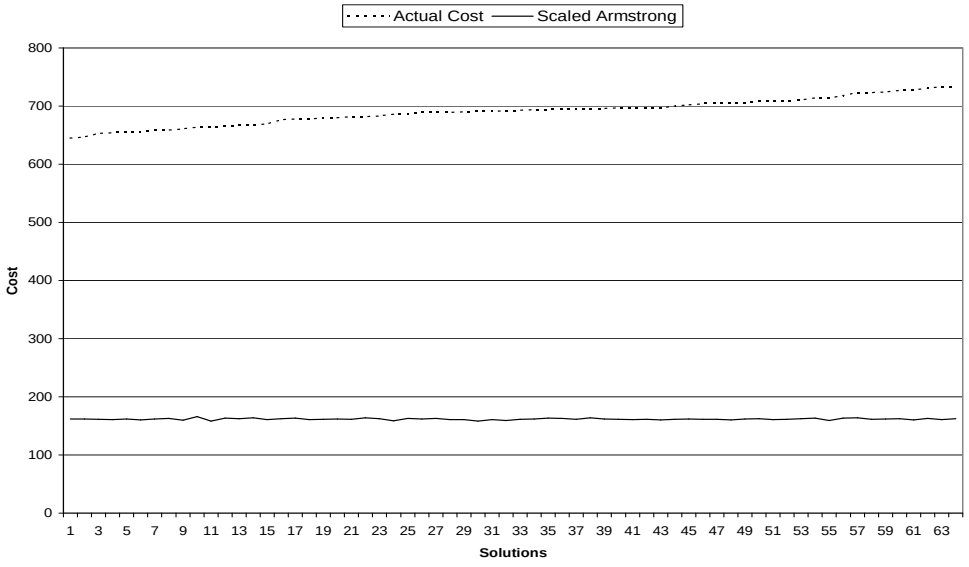


Figure 5.23: Armstrong Correlation on planet circuit

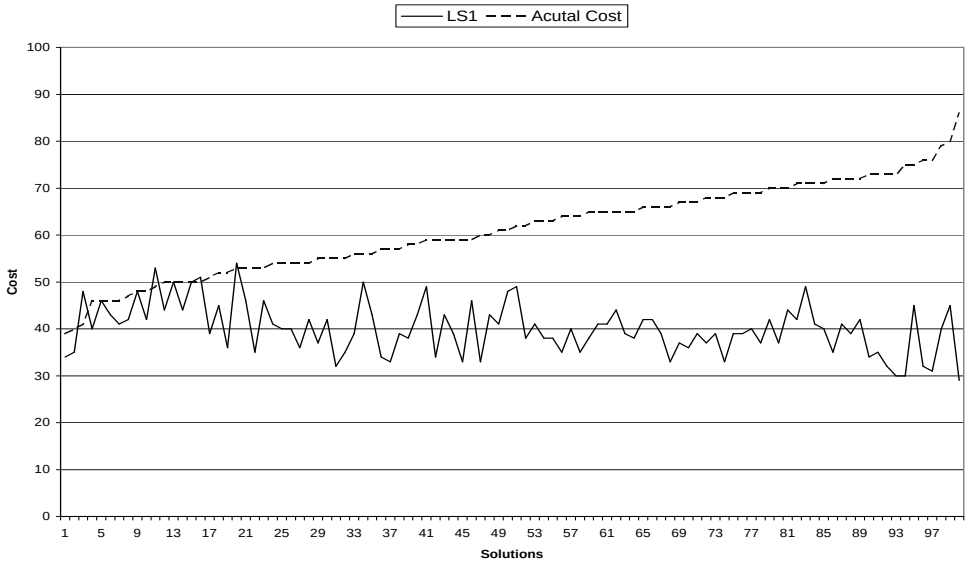


Figure 5.24: LS1 Correlation on train11 circuit

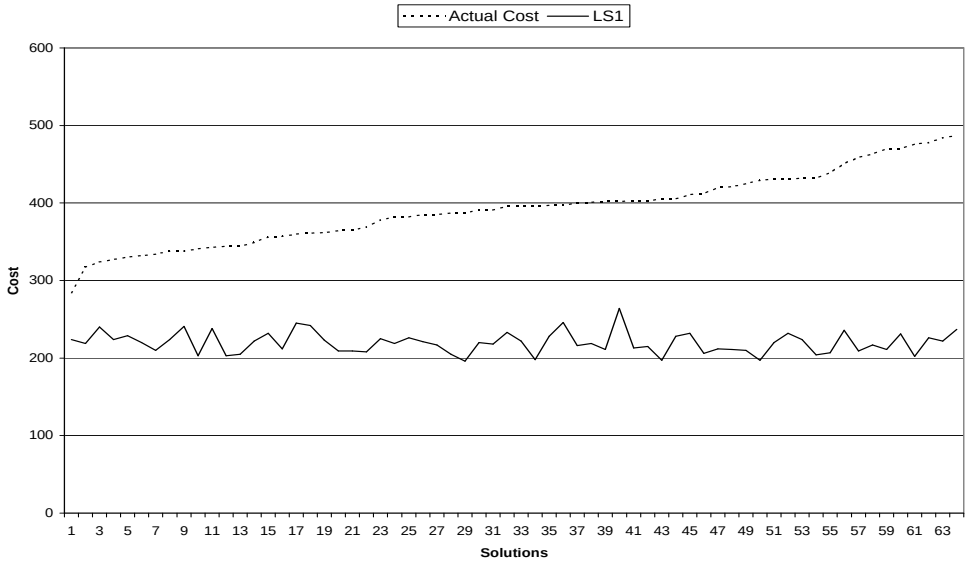


Figure 5.25: LS1 Correlation on keyb circuit

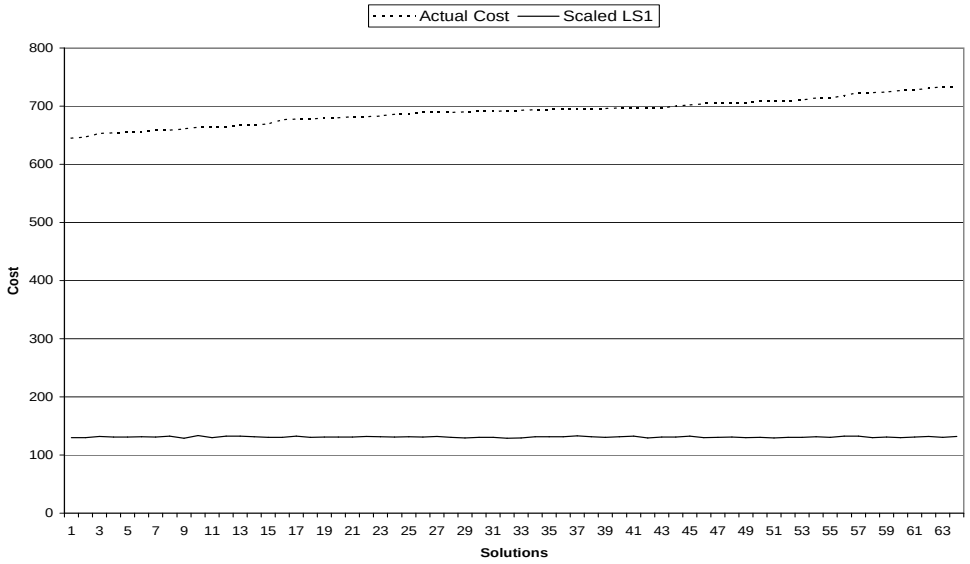


Figure 5.26: LS1 Correlation on planet circuit

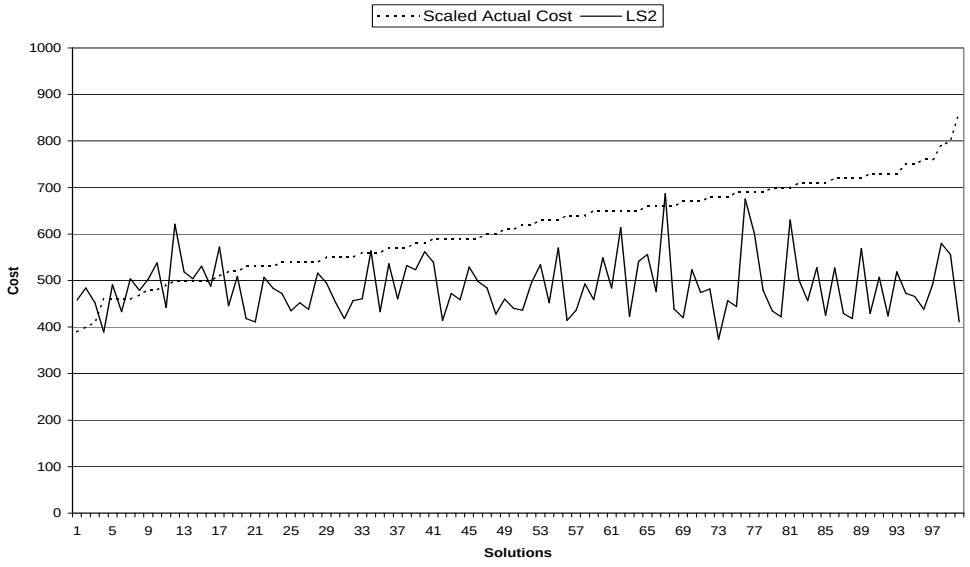


Figure 5.27: LS2 Correlation on train11 circuit

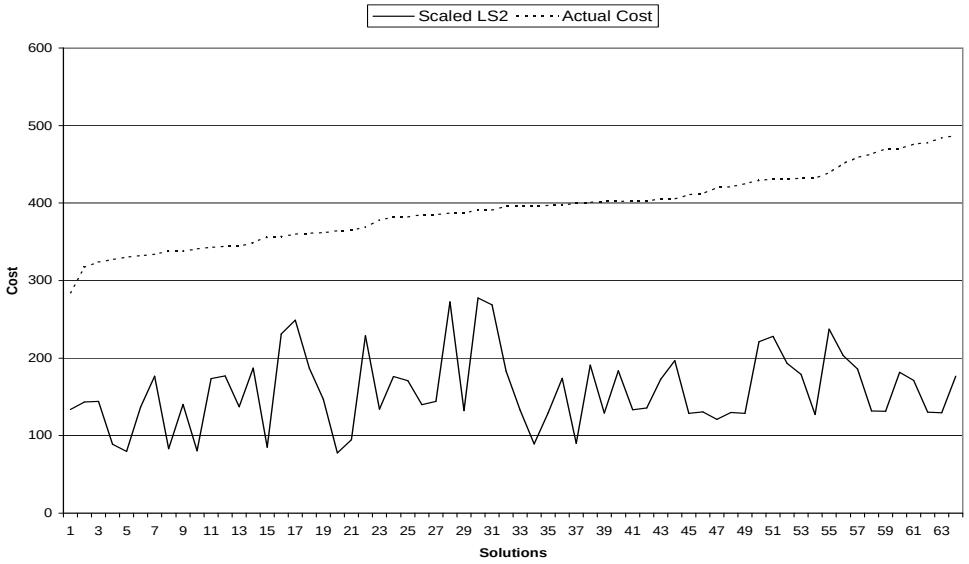


Figure 5.28: LS2 Correlation on keyb circuit

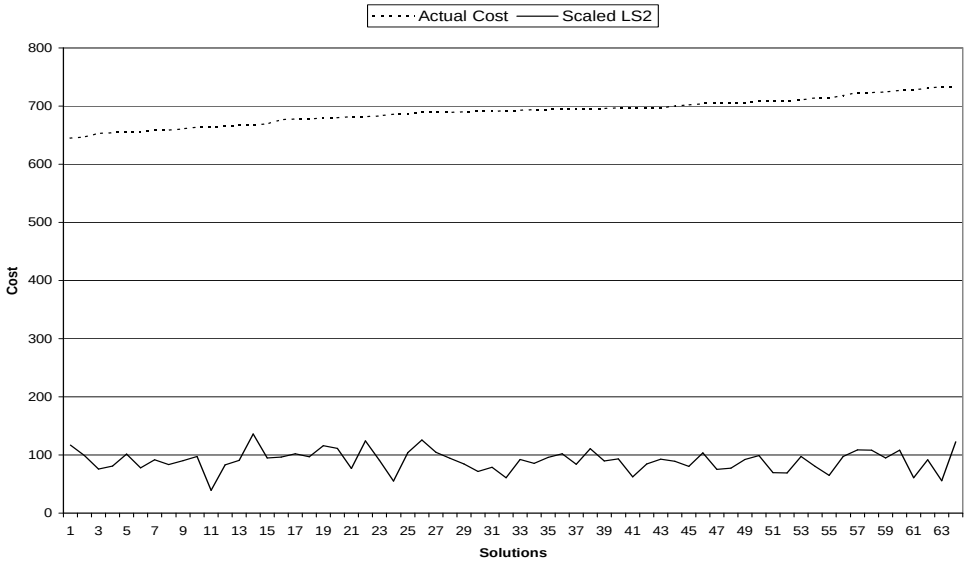


Figure 5.29: LS2 Correlation on planet circuit

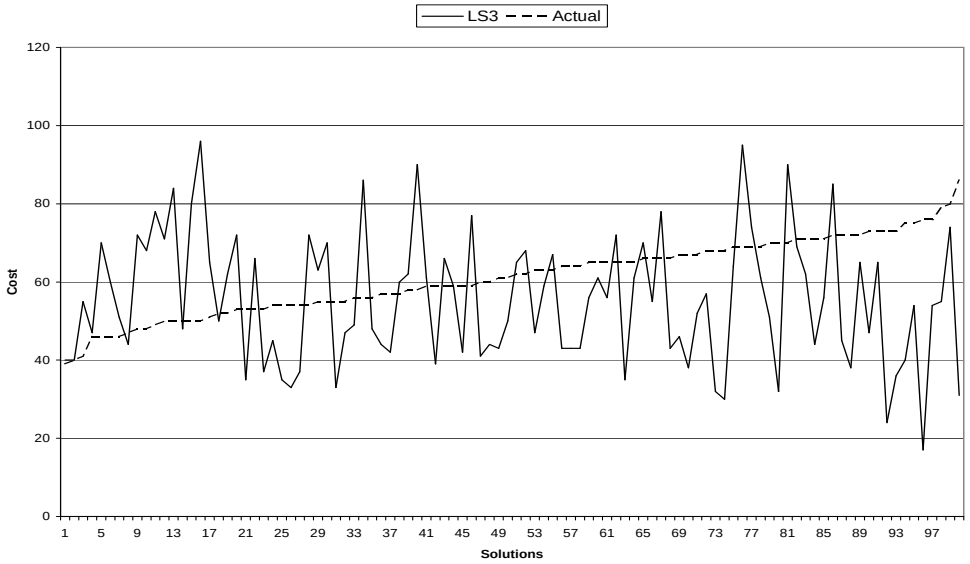


Figure 5.30: LS3 Correlation on train11 circuit

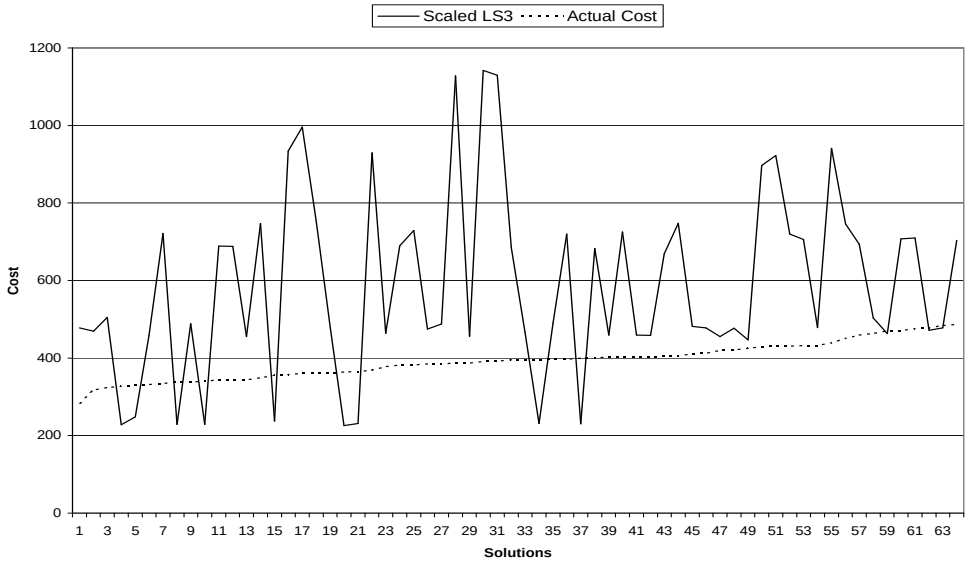


Figure 5.31: LS3 Correlation on keyb circuit

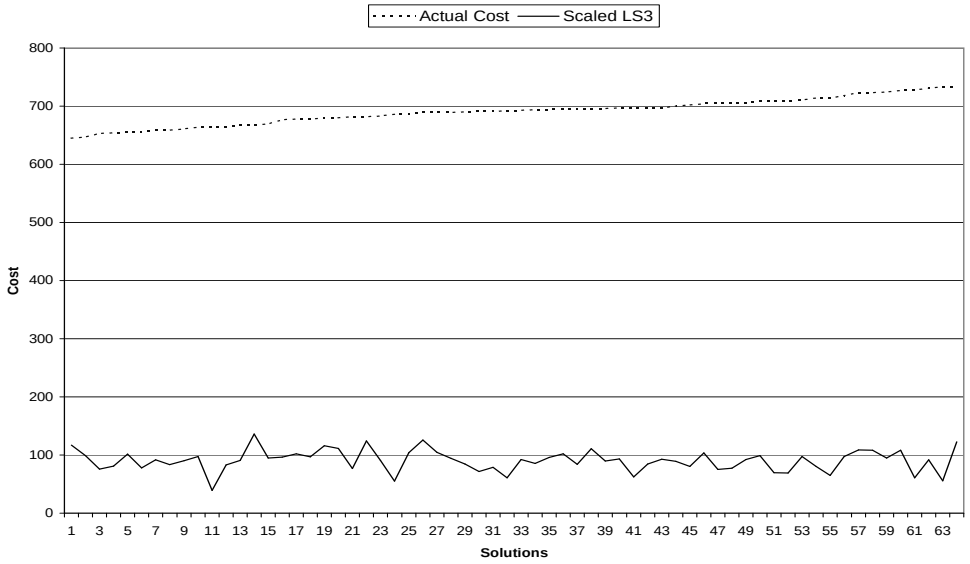


Figure 5.32: LS3 Correlation on planet circuit

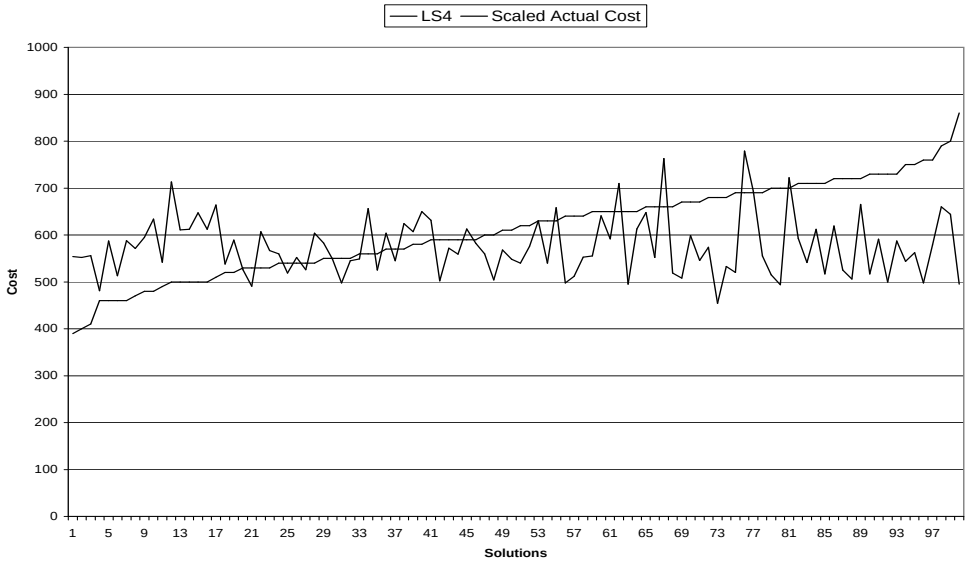


Figure 5.33: LS4 Correlation on train11 circuit

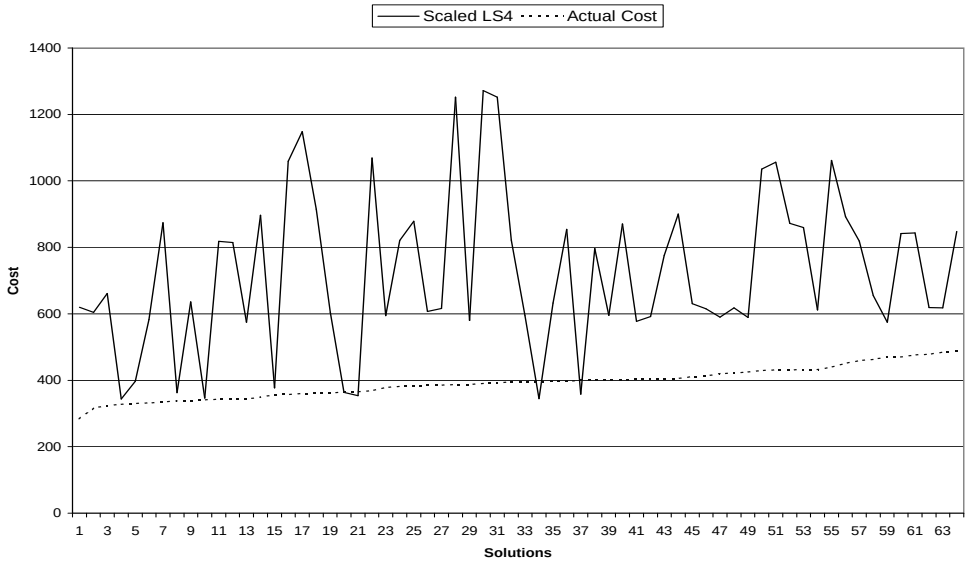


Figure 5.34: LS4 Correlation on keyb circuit

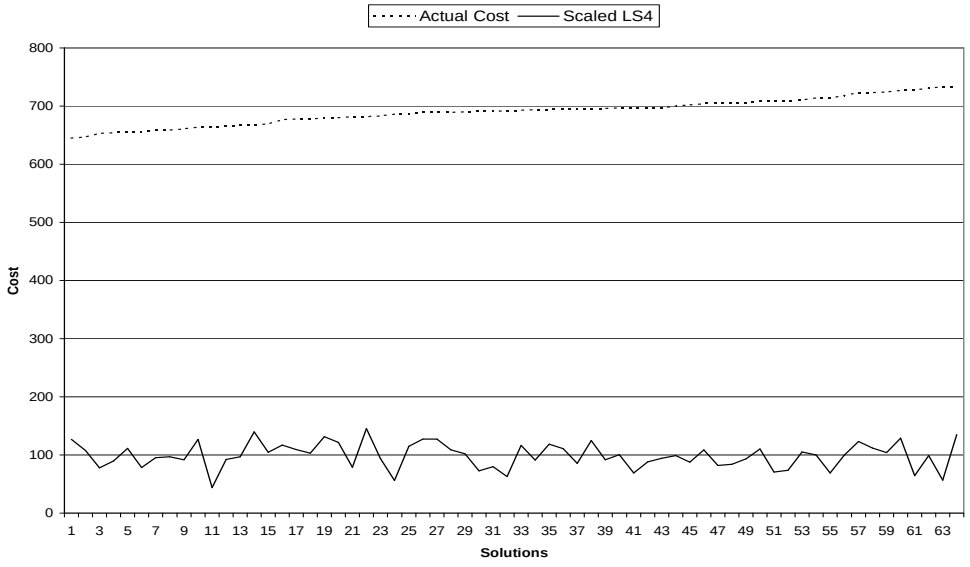


Figure 5.35: LS4 Correlation on planet circuit

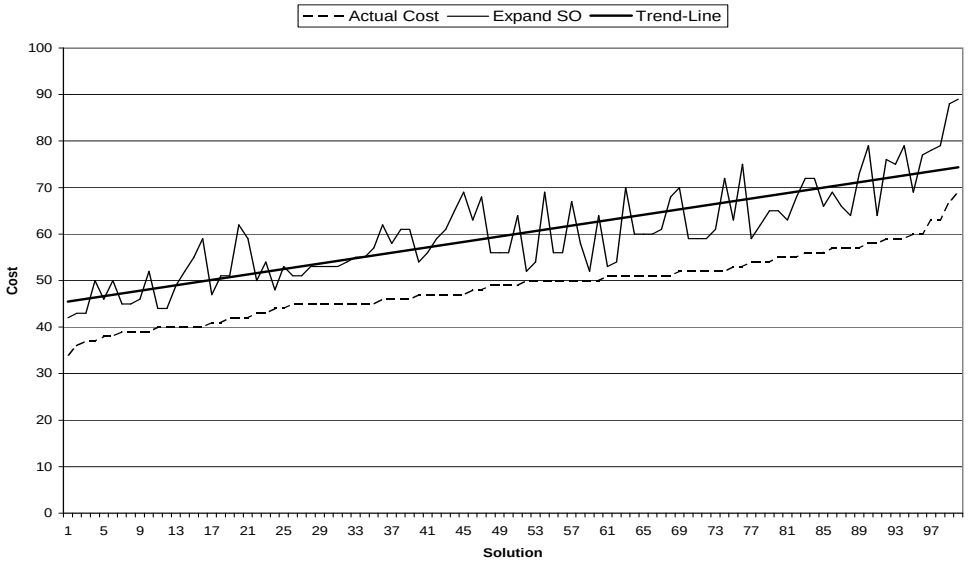


Figure 5.36: Expand-SO Correlation on train11 circuit

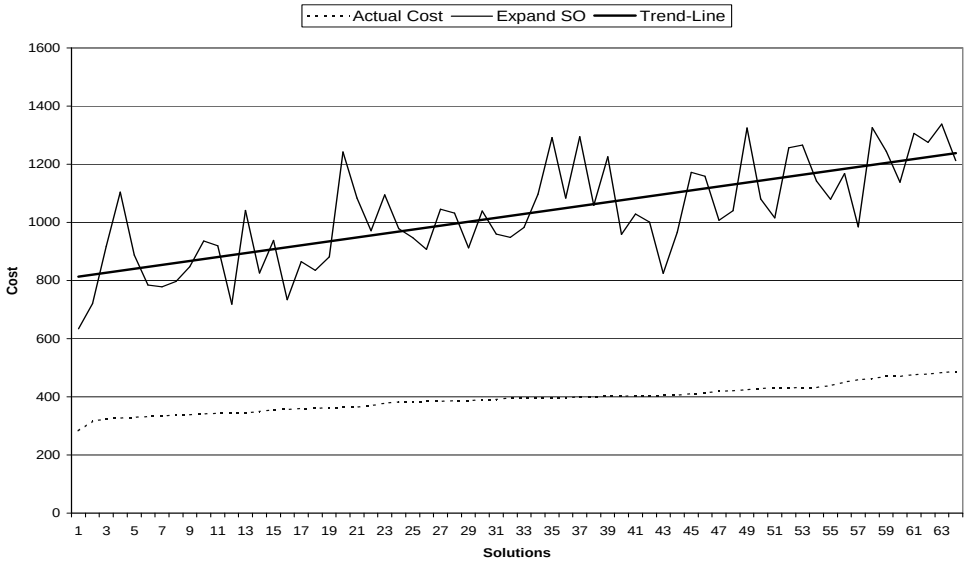


Figure 5.37: Expand-SO Correlation on keyb circuit

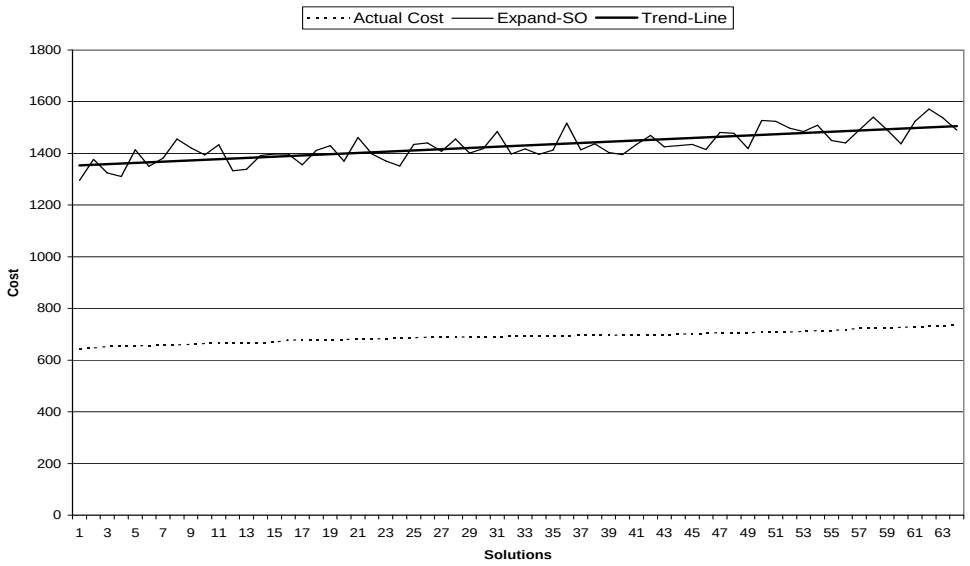


Figure 5.38: Expand-SO Correlation on planet circuit

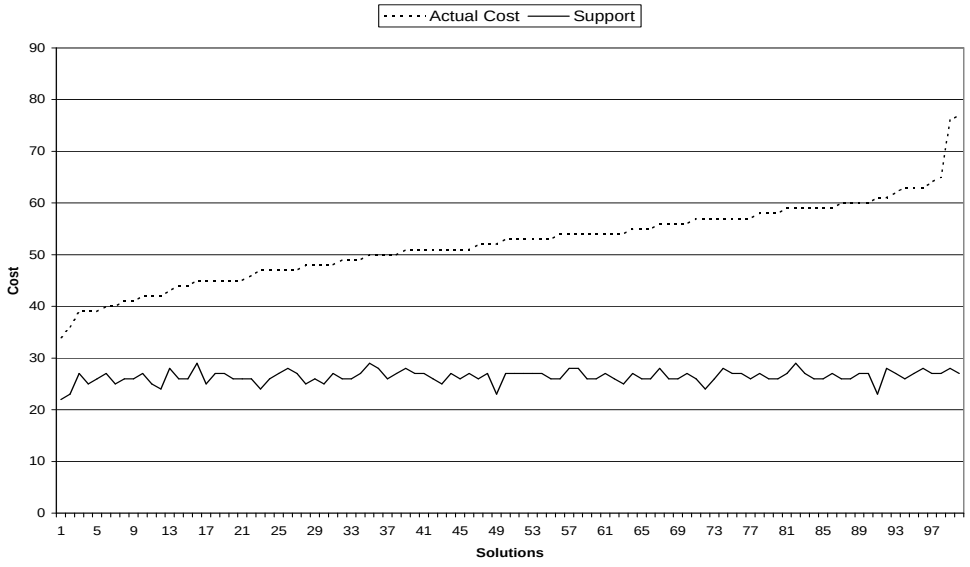


Figure 5.39: Support Correlation on train11 circuit

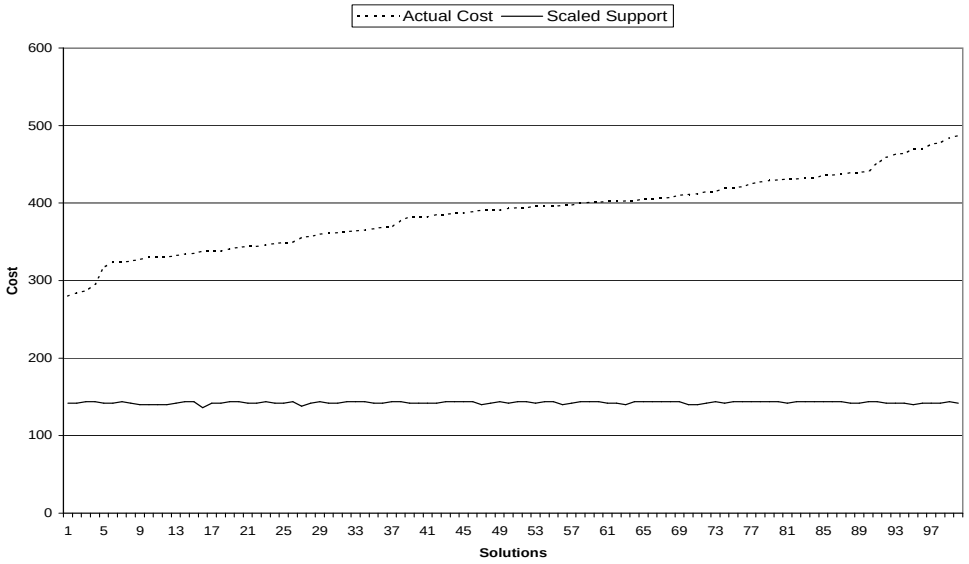


Figure 5.40: Support Correlation on keyb circuit

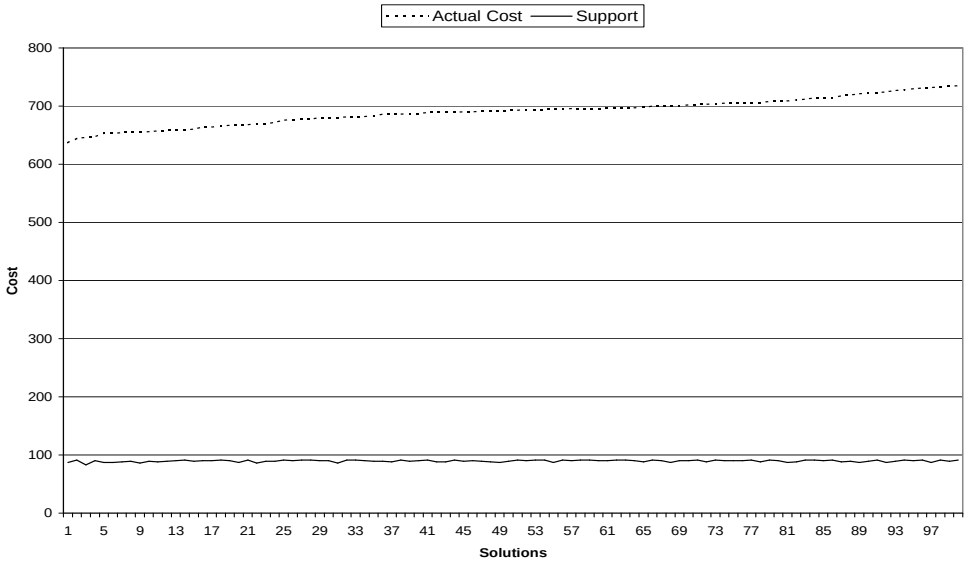


Figure 5.41: Support Correlation on planet circuit

It can be seen from the correlation graphs that except Expand-SO, none of the cost measures correlate well with the final cost. This is especially true as it is very difficult to model FSM state-assignment problem at a higher level of abstraction(see section-4.2)

Convergence of Jedi cost with respect to generations is next compared by plotting Jedi's best cost in a generation with its actual cost. Actual cost is calculated using method used before. The convergence plots thus obtained using 350 generations size are plotted in Figure-5.42 to Figure-5.44.

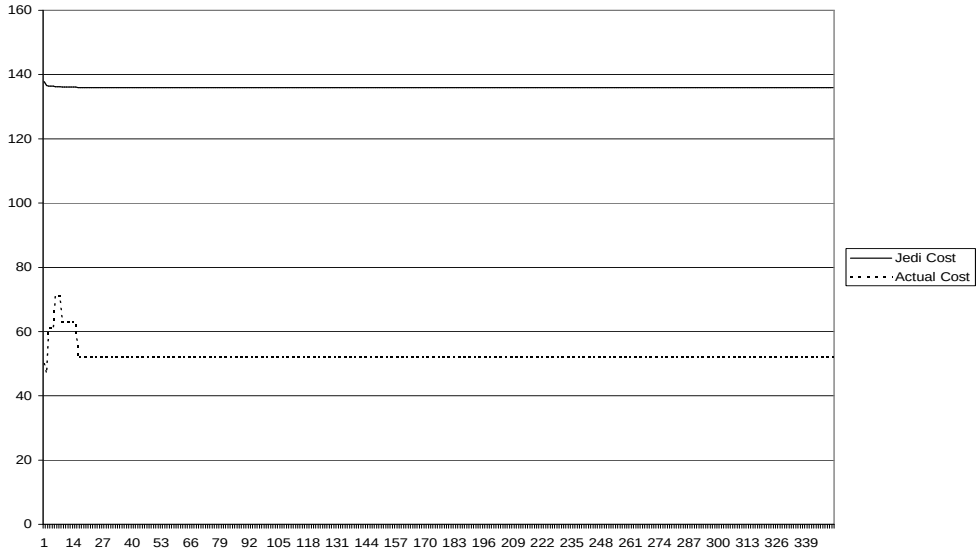


Figure 5.42: Jedi generation convergence graph on train11 circuit

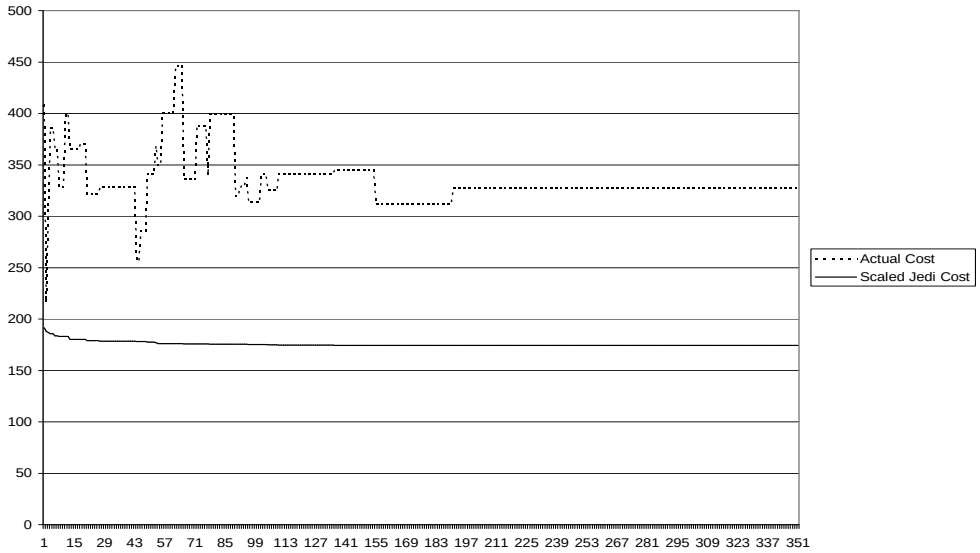


Figure 5.43: Jedi generation convergence graph on keyb circuit

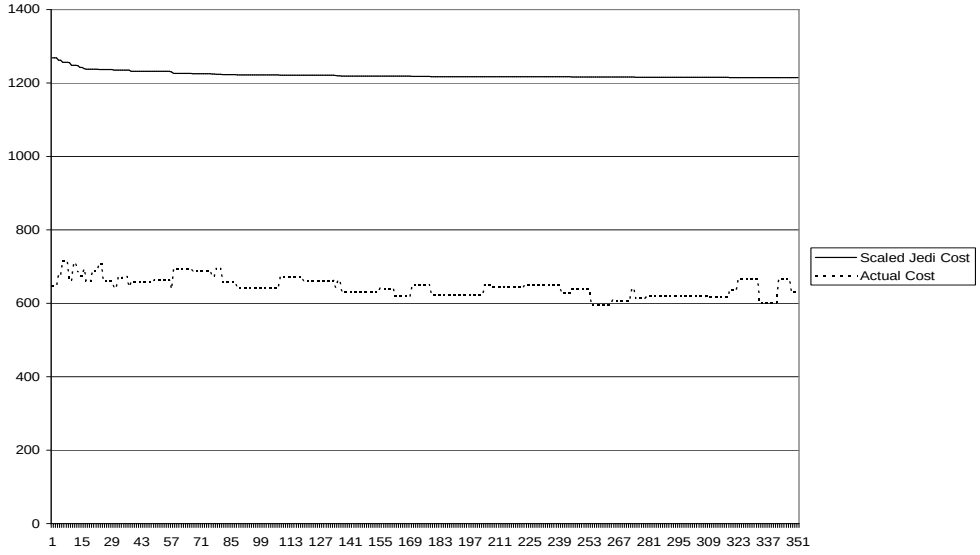


Figure 5.44: Jedi generation convergence graph on planet circuit

A comparison of Expand-SO using GA with other area minimization heuristics is next presented in Table-5.9 and Table-5.10. Table-5.9 presents the comparison of Expand-function with heuristics whose implementations were available. Table-5.10 gives a comparison of Expand-function with results as reported in literature. Values in brackets represent time taken by the GA in seconds to achieve the corresponding solution.

The implementations of Jedi and Nova available in SIS-1.2 are utilized for computing their respective values in Table-5.9. There are various options available for both Jedi and Nova and the best results obtained in all the options are shown in the table. Armstrong cost function was implemented for its comparison. All the values in the table are reported after fast-extraction(Fx).

Benchmark[1]	Expand	Jedi	Nova	Armstrong
bbara	56(113)	73	57	59
bbsse	110(30)	134	140	127
cse	198(219)	240	214	220
dk14	104(8)	108	111	124
donfile	87(388)	82	154	171
ex2	78(224)	123	127	131
ex3	56(14)	65	71	71 68
keyb	199(1838)	260	201	334
lion9	11(7)	19	27	27
planet	486(3259)	603	591	607
pma	165(933)	263	241	218
s1	227(629)	282	340	291
s1494	570(19375)	679	715	696
s832	231(6754)	257	274	301
sand	498(6408)	554	558	619
shiftreg	2(1)	2	2	2
styr	419(4265)	518	502	546
tbk	440(91433)	305	365	711
train11	22(32)	34	32	32
Average	208.37	247.4211	248.5263	278.10252

Table 5.9: Cost functions comparison - I

The comparison given in Table-5.10 uses script.rugged after synthesis except in the case of Armstrong where the reported results are after factorization. However, as the script has evolved with time, care should be taken while interpreting the results. The current work utilizes script.rugged available with

SIS-1.2.

Benchmark[2]	Expand	Jedi [22]	Mustang [21]	Armstrong[3][26]
bbara	52	57	64	86
bbsse	105	111	106	180
cse	228	200	206	NA
dk14	86	106	117	252
donfile	72	76	160	257
ex2	68	122	119	NA
ex3	48	66	71	NA
keyb	205	140	167	NA
lion9	11	13	17	21
planet	438	547	544	NA
pma	152	NA	NA	NA
s1	105	152	183	NA
s1494	624	NA	NA	NA
s832	218	NA	NA	NA
sand	494	437	462	NA
shiftreg	2	2	2	10
styr	429	508	546	NA
tbk	355	278	547	NA
train11	20	27	37	47

Table 5.10: Cost functions comparison - II

The comparison in Table-5.9 and Table-5.10 shows that except with three circuits, keyb, sand and tbk, Expand-SO achieves better literal count as compared to all other area minimization heuristics. The improved averages in Table-5.9 highlight savings of approximately 40 literals with Jedi and Nova while 70 literals with Armstrong's on every circuit. These results along with the previous observation of Expand-SO correlation justifies the use of Expand-SO as an efficient cost measure to be utilized for FSM area optimization.

5.5.1 Tabu Comparison

This section uses TS algorithm for multilevel area minimization and compares the results with those obtained using GA. Expand-SO is used as the cost

[1]Espresso-SO + Fx
[2]Espresso-SO + script.rugged
[3]Espresso-SO + factorization

measure.

Table-5.11 compares GA with TS using 350 iteration (generation for GA) size. LC in the table represent literal count values obtained using the synthesis methodology as used previously. The simulation time reports the time it took for arriving to the best solution.

	GA		TS	
	LC	Time	LC	Time
bbara	56	113	55	18
bbsse	110	30	115	6
cse	198	219	213	40
dk14	104	8	108	1
donfile	87	338	75	1088
ex2	78	224	81	115
ex3	56	14	57	25
keyb	199	1838	154	267
lion9	11	7	12	2
planet	486	3259	434	2508
pma	165	933	154	963
s1	227	629	158	1783
s1494	570	19375	493	8283
s832	231	6754	222	2785
sand	498	6408	494	6403
shiftreg	2	1	2	1
styr	419	4265	412	7729
tbk	440	91433	386	93638
train11	22	32	20	119
Average	208.3684	7151.579	191.8421	6619.6842

Table 5.11: Literal count comparison between TS and GA

It can be seen from the averages that TS is more efficient in exploring the search space. However, as there are two variables, quality and time, such a conclusion is difficult to draw in the previous table from averages alone. A more accurate comparison could be done by fixing either simulation time or solution quality and comparing the other parameter between the two search algorithms.

Simulation time is next fixed to compare solution quality when the search

algorithms are ran for equivalent amount of times. The time at which simulation snapshot is taken is the time to reach the better of the two solutions using either GA or TS for a given circuit. Thus from the above table, TS gives a better solution for bbara circuit at time 20 seconds and is therefore the selected snapshot for quality comparison. Such a snapshot comparison is tabulated in Table-5.12.

	GA	TS
bbara	62	55
bbsse	127	115
cse	257	213
dk14	104	108
donfile	87	84
ex2	94	81
ex3	56	53
keyb	224	154
lion9	16	12
planet	494	434
pma	165	160
s1	227	195
s832	570	493
s1494	250	222
sand	498	494
shiftreg	2	2
styr	419	435
tbk	440	386
train11	22	21
Average	216.5263	195.6316

Table 5.12: Literal count comparison between TS and GA by fixing time

The above table shows that TS on average is reducing 22 more literals per circuit as compared to GA when both are simulated for equal amount of time. There are only four circuits where GA is performing better than TS with literal count difference in three being less than 5 literals. These observation thus further confirm the earlier analysis of TS being more efficient in search space exploration for area minimization.

5.6 Power

This section evaluates cost functions for FSM power minimization as discussed in chapters 2 and 3 and compares their efficiency with those reported in the literature. *Minimum Weighted Hamming Distance* (MWHD) and *Fanout* based measures as discussed previously are used to experiment with FSM power minimization problem. These cost measures are also integrated with Expand-function area estimate for further power-tuning. All the results in this section are calculated using the script file given in Figure-5.45. The power values reported are in microwatts assuming 20 MHz clock and 5 voltage power supply. The steady state probabilities of the set of benchmark circuits used is given in Appendix-B.

```
stg_to_network -e 2
fx
read_library lib2.genlib
map
power_estimate -t SEQUENTIAL
```

Figure 5.45: Power calculation script

Circuits *ex2* and *ex3* have a unique characteristic of having no fanouts from state-0. Thus after infinite execution, these circuits will always be found in state-0. The steady state probability of state-0 is thus 1 with all the rest as 0. Such a probability distribution makes these circuits infeasible with power reduction heuristics used in this work as their power-cost always turn out to be zero. For this reason, ex2 and ex3 circuits are not used as benchmarks for power experimentation.

5.6.1 MWHD

Table-5.13 summarizes the results using MWHD measure on the selected set of benchmark circuits. Results obtained by using genetic (GA) and tabu-search algorithms are compared with those results obtained using Jedi's option that gives best literal count from all of its options.

It can be seen that MWHD performs inferior to area minimizer Jedi in terms of power. This is because MWHD measure is blind to area being synthesized which is a strong function of power.

MWHD measure's correlation with actual area and power of a circuit is next presented by using similar methodology as used in previous section for

	MWHD-GA		MWHD-Tabu		Jedi-Best	
	Power	Area	Power	Area	Power	Area
bbara	214.7	82	158.2	71	156.5	73
bbsse	446.1	140	499.1	144	496.6	134
cse	528.9	217	478.5	217	525.5	240
dk14	661.2	140	649.1	137	628.1	108
donfile	895.9	206	691.6	150	399	82
keyb	655.3	263	572.9	227	767.6	260
lion9	142	20	184	25	145.6	19
planet	1788.6	656	1612	623	1919.1	603
pma	653.4	198	902.1	195	883.7	236
s1	1165.1	406	1349.3	409	1087.2	282
s1494	1376.3	734	1253.7	727	1708.8	644
s832	922.1	368	948	395	1011.5	357
sand	1645.5	599	1475.8	579	1243.9	554
shiftreg	163.3	27	155.6	26	96.3	2
styr	1277.5	540	1223.9	530	1100.7	518
tbk	1682	630	1612.8	612	721.2	305
train11	180.4	38	187.4	37	207.1	34
Average	846.9588	309.6471	820.8235	300.2353	770.4941	261.8235

Table 5.13: Power consumption comparison of MWHD with Jedi

demonstrating area-measures' correlations. The correlation graphs are plotted in Figure-2.24. It can be observed from the graphs that MWHD measure has no correlation with circuit's area and as such it depicts very little correlation with circuit's power as well.

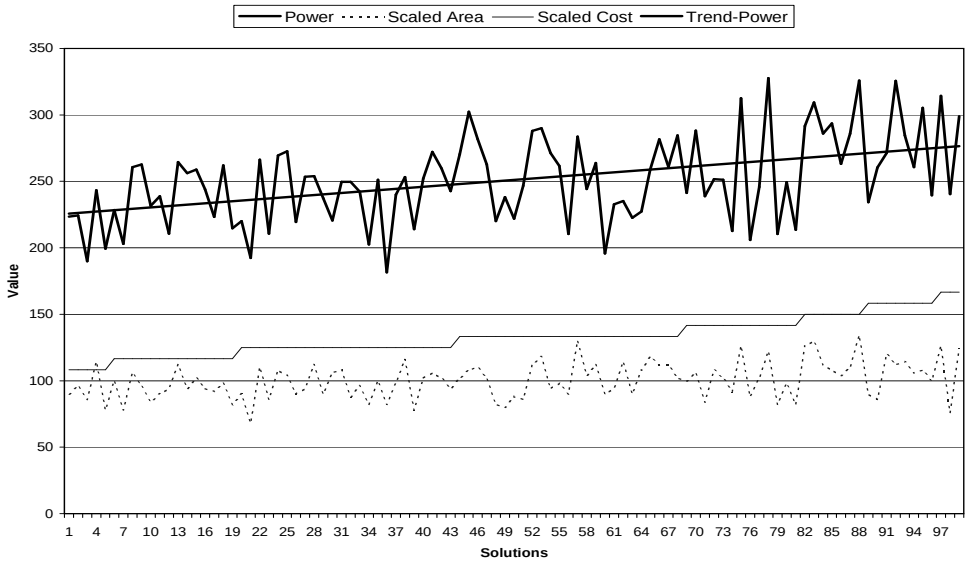


Figure 5.46: MWHD Correlation on train11 circuit

5.6.2 Fanout

Fanout based measure as discussed in chapter-2 is next experimented for power-minimization problem. Table-5.14 summarizes the results using the selected set of benchmark circuits.

The reduced average power consumption of Fanout measure substantiates the benefit of minimizing frequently switched fanout branches of flip-flops than minimizing total transition probability between states as is done in MWHD measure. There is also a reduction in average area as fanout measure incorporates circuit area information available from expanded cover in its search for power optimal solutions. TS is seen to perform better as compared to GA.

The correlation graphs for Fanout measure are next presented in Figure-3.2. Power-trend lines are again plotted to demonstrate the variance in actual power with fanout-cost. The linearly increasing trend lines with fanout-cost

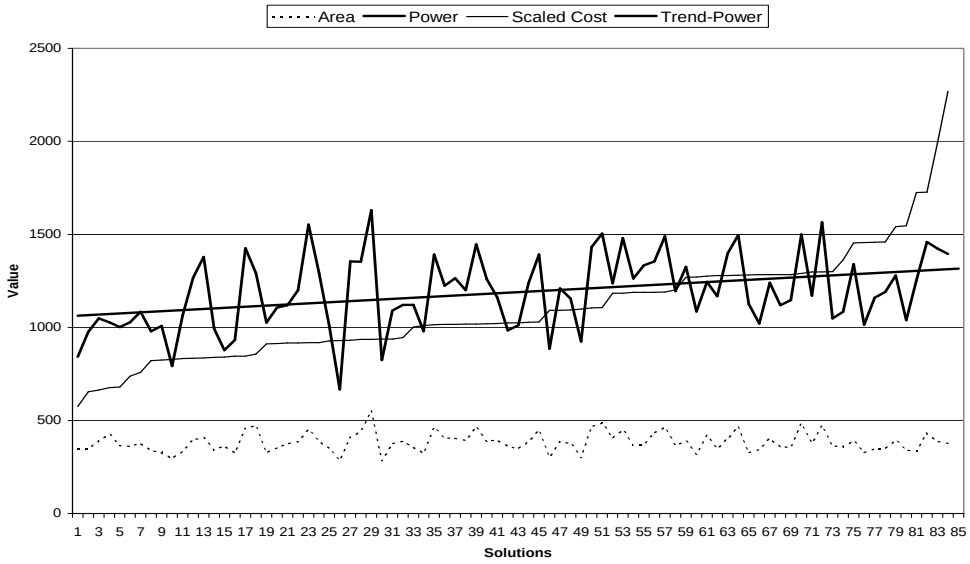


Figure 5.47: MWHD Correlation on keyb circuit

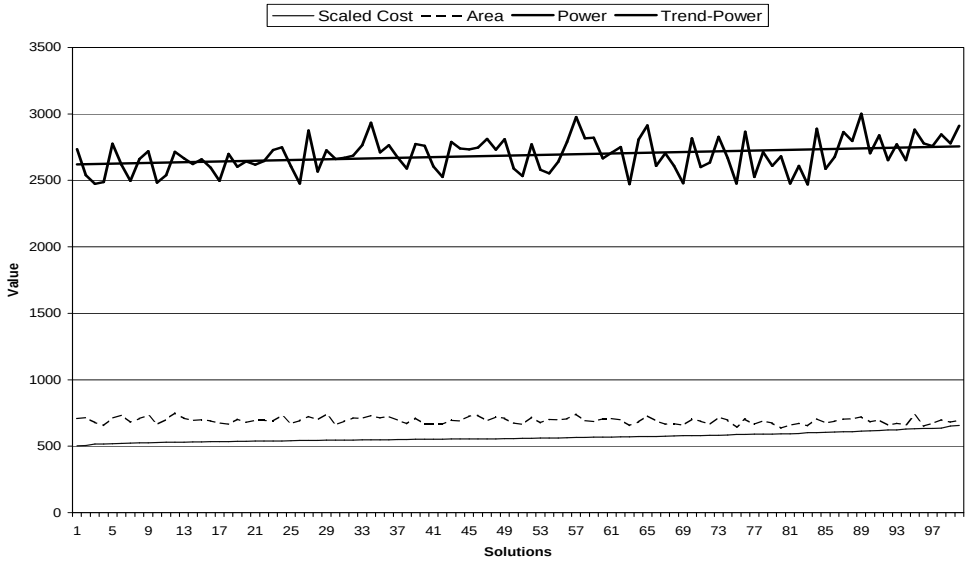


Figure 5.48: MWHH Correlation on planet circuit

	Fanout - GA		Fanout-Tabu	
	Power	Area	Power	Area
bbara	150.5	55	169.7	56
bbsse	412.2	122	489	123
cse	424.8	211	474.9	215
dk14	561.4	103	592.1	109
donfile	513.7	109	236.4	49
keyb	645	237	558.1	222
lion9	116.7	19	123	22
planet	1795.1	553	1523.3	516
pma	778	180	718.1	159
s1	766.5	187	828.6	206
s1494	1553.1	625	1122.4	588
s832	677.5	271	683.4	272
sand	1541.4	559	1346.4	524
shiftreg	96.3	2	96.3	2
styr	1062.9	431	1125.1	409
tbk	1589.3	488	864.6	318
train11	136.3	23	163.6	24
Average	754.1588	245.5882	653.8235	224.3529

Table 5.14: Results for Fanout measure

follow the actual power closely and there are lesser variations along the trend line as compared to MWHD approach. This signifies an increase in level of correlation with actual power using fanout measure that is also evident from the results.

However, fanout trend graphs still show bumps along the trend line that highlight a certain degree of inaccuracy in the estimation. The primary reason behind this is inaccuracy of fanout estimation by utilizing Expand-area estimate that itself is not an accurate estimation of the final circuit. Secondly, logic cone dependant on primary-inputs is not taken into account by the fanout measure. A bigger such portion may offset the savings achieved from minimization of area being switched by flip-flops alone. Another reason is the possibility that the less switching-active (or active in short) area is big in size, thus contributing to a considerable chunk in total power consumption of a circuit due to static power dissipation in it. These inaccuracy yielding factors in fanout measure are next addressed by combining the estimate with previously employed area estimate in next section.

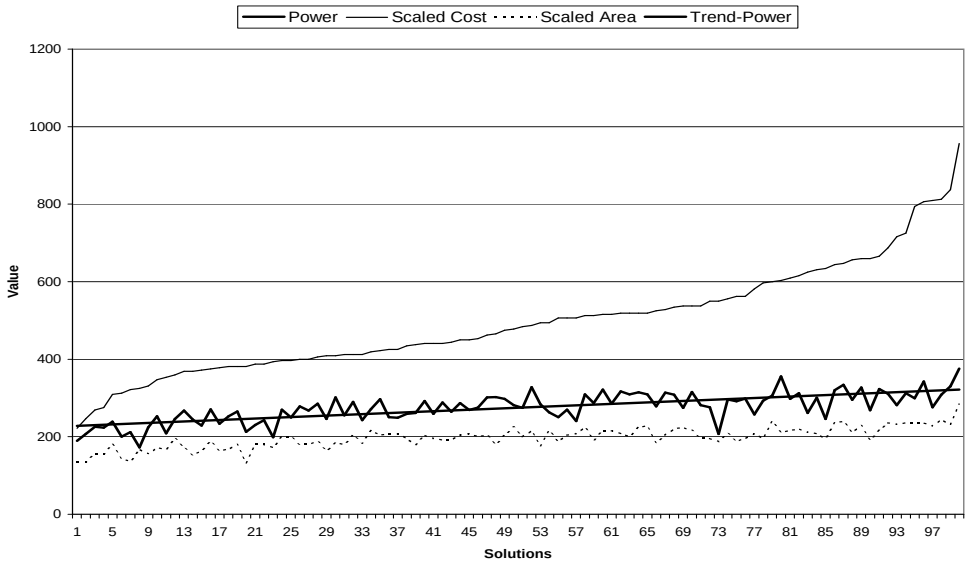


Figure 5.49: Fanout Correlation on train11 circuit

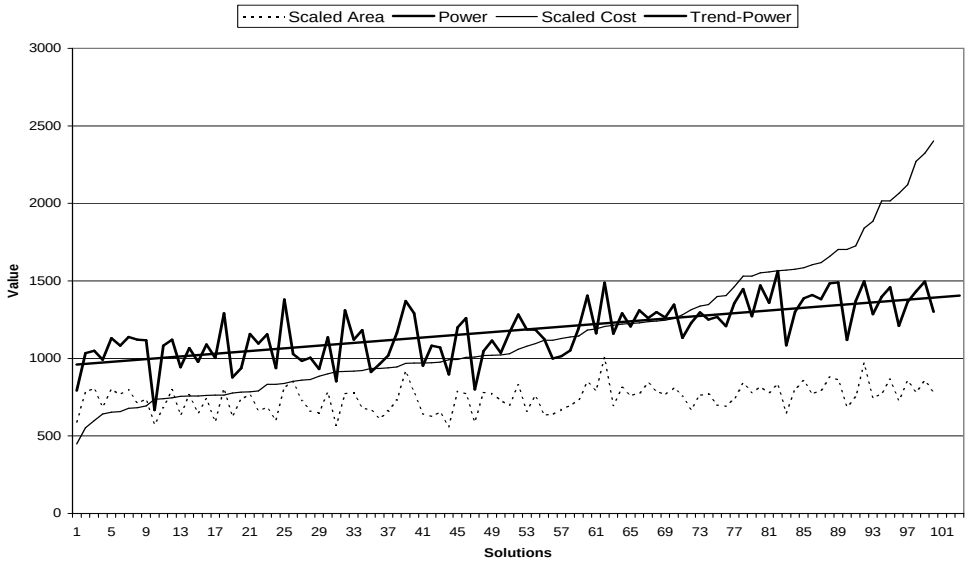


Figure 5.50: Fanout Correlation on keyb circuit

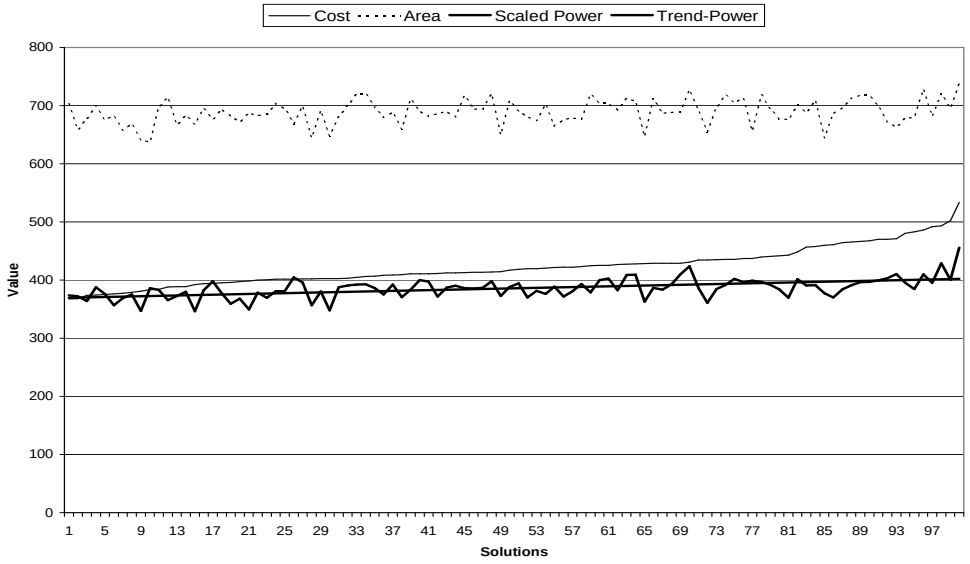


Figure 5.51: Fanout Correlation on planet circuit

5.6.3 Power and Area Estimates Integration

FSM power consumption is a function of its area being switched as discussed previously in chapter-2. Whereas MWHD measure tried to reduce the total switching probability between states of an FSM, Fanout measure also tried to incorporate circuit's area by minimizing total switching on fanout branches out of FSM's sequential elements. However, as earlier analyzed, fanout measure still requires further integration with the area being switched.

This section utilizes two techniques for integrating area estimate from Expand-function with MWHD and Fanout based power estimates. In the first method, product of literal count and power estimates is used as cost measure. The second method integrates area and power measures using fuzzy logic.

Product based integration

Table-5.15 tabulates the results for first integrating mechanism based on product of the measures. The product of **M**WHD and **A**rea estimate is represented as *MA* and with **F**anout as *FA* in the table.

It is observed that power and area costs have consistently improved over MWHD using integrated approach, MA. The only circuit where power decrease occurred, pma, happened in spite of a decrease in area. This can be attributed to the inaccuracy in the power estimate.

A similar decrease is also seen in integrated FA measure from fanout alone. However, there are many cases where both area and power cost has deteriorated. A major reason for this deviation is because of non-compensatory behavior of product measure towards two differently scaled costs. This issue will be addressed next by using fuzzy based integration.

Tabu search is seen to be yet again performing better as compared to genetic algorithm.

Fuzzy-logic Based Integration

Fuzzy logic based integration of area and power measures is carried out by using both *Min* and *Max* operators as discussed in chapter-3. TS is used for search space exploration. Table-5.16 summarizes the results.

It is observed that the shortcomings seen in product based integration in FA approach are much more efficiently handled in fuzzy based integration. There are now only three cases in Fanout(Max) as compared to 9 before where power has deteriorated from original Fanout. Among the three cases, only one comes along with an increase in circuit area as well.

	MA - GA		MA - Tabu		FA - GA -		FA - Tabu	
	Power	Area	Power	Area	Power	Area	Power	Area
bbara	148.7	57	163.1	51	181.2	65	164.3	57
bbsse	435.6	110	382.1	112	394.5	118	427.4	118
cse	526.2	213	442.7	209	391.3	209	425.9	212
dk14	559.8	104	529.6	101	561.4	103	529.6	101
donfile	313.4	65	292.6	68	474.1	100	292.6	68
keyb	601.8	228	570.9	208	517.3	215	510.8	195
lion9	135.6	12	131.3	11	100.8	15	92.9	16
planet	1568.9	540	1646.3	542	1889.7	510	1666.8	475
pma	766.6	172	706.5	172	693.1	165	737.6	157
s1	705.9	205	656.7	178	771.4	197	552.6	162
s1494	1366.8	592	1214.4	534	852.4	569	1033.2	563
s832	638.9	271	577.9	230	665.2	260	612	228
sand	1395.9	537	1367	516	1617.2	585	1498.5	553
shiftreg	96.3	2	96.3	2	98.8	4	96.3	2
styr	1090.2	429	1249.1	436	1086.8	453	1099.6	417
tbk	1281.7	385	706.5	172	1766.6	556	936.7	327
train11	147.6	21	177.1	25	142.4	22	115.2	22
Average	692.94	231.94	622.90	224.18	717.89	243.88	634.84	216.06

Table 5.15: Product based integration of area and power estimates

	MWHD(Max)		MWHD(Min)		Fanout(Max)		Fanout(Min)	
	Power	Area	Power	Area	Power	Area	Power	Area
bbara	163.8	61	166.4	62	181.2	58	181.2	58
bbsse	391.2	120	511.8	130	436.8	122	446.1	127
cse	488.6	207	478.3	214	485.4	206	545.3	218
dk14	580.7	124	554.3	113	529.6	101	584.6	112
donfile	678.6	152	460.8	96	286.3	62	211.6	41
keyb	507.1	176	487.4	177	505.8	193	509.5	191
lion9	131.1	16	152	21	97.4	14	129.9	11
planet	1763.3	496	1693.4	518	1670.7	450	1946.6	470
pma	672.7	163	694.4	158	607.1	153	687.1	145
s1	811.3	224	801.9	222	734.4	181	614.3	157
s1494	1116.3	551	1125.9	522	838.5	531	1056.2	508
s832	625.2	236	620	240	627.4	238	619.1	250
sand	1535.7	576	1499.6	532	1254.2	487	1349.9	496
shiftreg	151.9	26	173.8	20	96.3	2	96.3	2
styr	1002.5	370	1070	426	1016.1	422	1006.8	376
tbk	1181.6	351	998.7	350	886.2	354	1016.3	341
train11	179.8	33	180.7	33	122.2	23	150.1	21
Average	704.79	228.35	686.43	225.53	610.33	211.59	655.93	207.29

Table 5.16: Fuzzy based integration of area and power estimates

Fanout(Min) that tries to optimize both the objectives simultaneously is seen to be performing inferior to Fanout(Max) in terms of power, although the area results are little better. One reason of this could be the unpredictable interaction between the two as fanout influences circuit area as well. Performance of Fanout(Min) is however still comparable in terms of power with fanout alone while substantially better in area.

Fanout(Max) produces the best power results among all the approaches considered so far. The case is thus used for literature comparison in the next subsection

5.6.4 Literature Comparison

Comparison of our techniques with best area yielding option in Jedi has already been reported earlier. A comparison of our best technique (Fanout(Max)) with default output dominant option in Jedi along with ensuing percentage power and area reductions is presented in Table-5.17. It is observed that except with one case, our technique performs better than Jedi in area and as well as power. There is a significant improvement in both power and area using Fanout(Max) approach saving nearly 200uW as well as 66 literals per circuit from Jedi.

Comparison with Jedi above further indicates a strong correlation of circuit's power with its area as savings in power are always being achieved with reduction in circuit's area. An argument can thus be made to optimize circuit's area alone for minimal power. Power consumption of the best area solutions obtained previously in section 5.5 are therefore presented next in Table-5.18.

It can be observed from the results that minimum area solution is no guarantee for minimum power. Average power using minimum area is significantly high as compared to Fanout(Max). It is further observed that Fanout(Max) is competing very closely to minimum area while being significantly better in terms of power.

There has been rich amount of work done to reduce power consumption in an FSM as reviewed in chapter-2. However, the tools and techniques to estimate power consumption have evolved over the years. The method to synthesize a state-assignment, library being employed, and tool being used for power estimation, all greatly vary from one work to another. Apart from these difficulties, some works rely on reporting switching activity only instead of actual power consumption. Due to these reasons, it is very difficult to have an accurate comparison between works.

The above mentioned comparison difficulty is also highlighted in previous works, most of whom only compare their performance with Jedi tool. Thus,

	Fanout(Max)		Jedi - Default		% Reduction	
	Power	Area	Power	Area	Power	Area
bbara	181.2	58	187.7	74	3.462973	21.62162
bbsse	436.8	122	538.8	149	18.93096	18.12081
cse	485.4	206	495.8	251	2.09762	17.92829
dk14	529.6	101	714.4	157	25.86786	35.66879
donfile	286.3	62	380.8	89	24.81618	30.33708
keyb	505.8	193	767.6	260	34.10631	25.76923
lion9	97.4	14	145.6	19	33.1044	26.31579
planet	1670.7	450	2001.5	675	16.5276	33.33333
pma	607.1	153	883.7	236	31.30022	35.16949
s1	734.4	181	1205.3	353	39.06911	48.72521
s1494	838.5	531	1668.9	679	49.75733	21.79676
s832	627.4	238	1068.4	376	41.27668	36.70213
sand	1254.2	487	1458.9	651	14.03112	25.19201
shiftreg	96.3	2	132.5	9	27.32075	77.77778
styr	1016.1	422	1118.6	567	9.16324	25.57319
tbk	886.2	354	721.2	305	-22.8785	-16.0656
train11	122.2	23	218.2	35	43.99633	34.28571
Average	610.3294	211.5882	806.3471	287.3529	23.05589	29.30892

Table 5.17: Comparison between Fanout(Max) and Jedi-Default

Jedi with its default output dominant option, has become a base algorithm for power comparisons. However, there still remain other variables because of which for an accurate comparison, the compared-to algorithm has usually been re-implemented in previous works.

Four recent works reported in literature are used for comparison in this section. A relative comparison is given for three of them ([61], [96] and [97]) whereas actual power value comparison is given for [63] as its authors have provided their final obtained solutions.

A relative comparison compares percentage power reductions achieved from Jedi instead of actual power values. This approach has an advantage of abstracting the various variables in accurate power comparison while also providing an insight on relative efficiencies of different techniques on a given circuit. Relative comparisons with Pedram et al[61], Ciesielski et al[96] and Chattopadhyay and Reddy's, IITG8 [97], are given in Table-5.19 to Table-5.21 respectively. Actual power comparison between this work and Xia and Almaini [63] is given in Table-5.22

	Power	Area
bbara	166.7	55
bbsse	513.1	115
cse	601.9	213
dk14	587.8	108
donfile	380.7	75
keyb	589.7	154
lion9	135.6	12
planet	2212.4	434
pma	691	154
s1	617.5	158
s1494	1622	493
s832	701.9	222
sand	1301.1	494
shiftreg	96.3	2
styr	1201.3	412
tbk	1282.6	386
train11	117	20
Average	754.0353	206.2941

Table 5.18: Power consumption when optimizing for area alone

Pedram's work targeted low power and area FSM solution. It can be seen that our technique not only offers reduced power dissipation but nearly 5 times area improvement as compared to Pedram's. Ciesielski aimed at reducing power and increasing testability. There is again an improvement in percentage power reduction in the case of Ciesielski. Similarly, there is marked improvement in power from IITG8 and Almaini's approaches using Fanout(Max).

	Pedram		Fanout(Max)	
% red.	Power	Area	Power	Area
bbara	17.97	-10.14	3.46	21.62
bbsse	18.37	6.56	18.93	18.12
cse	12.15	-1.41	2.1	17.93
dk14	4.92	-0.98	25.87	35.67
donfile	6.22	22.64	24.82	30.33
sand	10.52	16.12	14.03	25.19
Average	11.69167	5.465	14.8683	24.81

Table 5.19: Power and Area reduction comparison with Pedram et al [61]

%red	Celiski	Fanout(Max)
bbsse	5.66	6.56
keyb	35.56	34.11
s832	7.75	41.28
tbk	5.03	-22.88
s1494	6.89	49.76
Average	12.178	21.766

Table 5.20: Power reduction comparison with Ciesielski et al [96]

%red	IITG8	Fanout(Max)
bbara	17.65369	3.46
cse	18.47769	2.1
dk14	16.19344	25.87
keyb	20.86835	34.11
s1	-22.4591	39.07
s832	26.67556	41.28
shiftreg	-29.075	27.32
styr	-9.00385	9.16
train11	11.6103	44
Average	4.916343	22.79625

Table 5.21: Power reduction comparison with IITG8 [97]

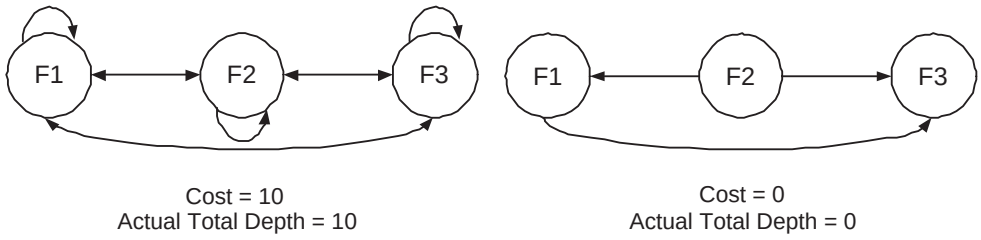
	Almiani	Fanout(Max)
bbara	235.9	181.2
cse	483	485.4
donfile	402	283.6
keyb	748.2	505.8
planet	2386.2	1670.7
s1	1293.6	734.4
sand	1740.3	1254.2
styr	1016.1	1016.1
train11	189.4	122.2
Average	943.8556	694.8444

Table 5.22: Power comparison with Xia and Almaini [63]

5.7 Testability

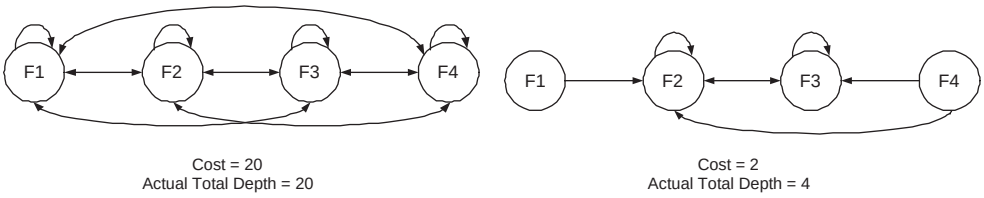
This section presents the results for testability measures discussed in section-3.5. This work uses HITEC to calculate testability of a circuit synthesized using espresso single-output minimization. HITEC is run for two iterations where the maximum time-limit, backtrack limit and state backtrack limits used are 200 seconds, 1,000,000 and 1,000,000 iterations respectively.

The performance of *TDepth* measure is first analyzed by plotting dependency graphs on circuits synthesized with and without using the cost-measure optimization. The dependency graphs for two benchmark circuits obtained from circuit implementations along with the measure's cost and actual total depth are shown in Figure-5.52 and Figure-5.53. Circles in the figures represent sequential elements and pointed arrows describe their dependency requirement. Head of an arrow points to a sequential element dependant on information provided by the element on its tail. A double pointed arrow denotes cross dependency between the sequential elements.



(a) Initial (b) Final

Figure 5.52: Loops reduction on shiftreg circuit



(a) Initial (b) Final

Figure 5.53: Loops reduction on train11 circuit

The loops reduction graphs graphically depict the performance of *TDepth* measure in reducing loops in a circuit. The graphs also demonstrate a high degree of accuracy in the estimation of total depth of a circuit. However, as cover produced by Expand-function can be completely changed by successive

heuristics in synthesis, there is a possibility that the total depth as calculated from the cover may not be exact. This is illustrated in the final implementation of train11 circuit in Figure-5.53 where there are two more loops in actual synthesized circuit than in Expand cover.

Table-5.23 to Table-5.25 tabulate results comparing Fault Coverage, Fault Efficiency and CPU-Time between the proposed measures and other heuristics. Average for all the approaches are also tabulated in the last row. *Int-OC* and *Int-QC* in the table represent the *Integrated* testability measure employing total depth of loops and initializability detection using *original complement* and *quick complement check* respectively.

Circuit donfile is not used in the benchmark set as its only output remains always set due to which all its flip-flops get simplified by testability scripts.

	Tabu			GA			
	Tdepth	Int-OC	Int-QCC	Int-QCC	Jedi	Nova	Amaral
bbara	0.9167	0.8333	0.8333	0.958	0.8412	0.8686	0.9333
bbsse	1	1	1	1	0.9245	0.9527	1
cse	0.9962	1	1	1	0.0121	0.9571	0.998
dk14	1	1	1	1	0.8918	0.8961	1
ex2	0.0053	0.7528	0.7528	0.9271	0	0	0
ex3	0	0.8966	0.5909	0.9302	0.7483	0.0674	0.9071
keyb	0.8917	0.965	0.965	0.9951	0.9273	0.9433	0.9974
lion9	0.0568	0.8689	0.8689	0.0769	0.8235	0	0
planet	0.0047	0.0025	0.0047	0.0047	0.033	0.0027	0.0056
pma	0.9982	0.0036	0.9964	0.0026	0	0	0
s1	0.0019	0.0034	0.0019	0	0	0.0027	0.9774
s1494	0.5	0.9976	0.9976	0.8943	0.8733	0.8592	0.9986
s832	0.6374	0.7523	0.9989	0.9956	0.9644	0.9135	0.9969
sand	0.0026	0.0026	0.0026	0.002	0.0021	0.0027	0.0024
shiftreg	1	1	1	1	0	1	1
styr	0.0042	0.0042	0.0042	0.0031	0.005	0.0027	0.0037
tbk	0.9627	0.9627	0.9627	0.9721	0.9926	0.904	0.9458
train11	0.9205	0.945	0.945	0.8571	0.8553	0	0.9714
Average	0.5499	0.6661	0.7181	0.6455	0.4941	0.4651	0.6521

Table 5.23: Fault Coverage

It is seen that certain circuits remain untestable using *TDepth* measure. The reason for their untestability lies in their uninitializable sequential elements. As earlier discussed in section 3.5, ATPG tools require initialization

	Tabu			GA			
	Tdepth	Int-OC	Int-QCC	Int-QCC	Jedi	Nova	Amaral
bbara	0.9679	1	1	0.986	0.9941	1	0.9778
bbsse	1	1	1	1	1	1	1
cse	1	1	1	1	1	1	1
dk14	1	1	1	1	1	1	1
ex2	0.9813	1	1	1	0.9972	1	1
ex3	1	1	1	1	1	1	1
keyb	1	1	1	1	1	1	1
lion9	1	1	1	1	1	1	0.9853
planet	1	1	1	1	1	1	1
pma	1	1	1	0.9987	1	1	1
s1	0.646	0.7621	0.646	0.782	0.9972	1	0.9907
s1494	1	1	1	1	1	1	1
s832	1	1	1	1	0.9988	0.9976	0.9985
sand	1	1	1	1	1	1	1
shiftreg	1	1	1	1	1	1	1
styr	1	1	1	0.755	1	1	1
tbk	0.9996	0.9996	0.9996	0.9997	1	0.9379	0.9458
train11	1	1	1	1	1	0.9857	1
Average	0.9775	0.9868	0.9803	0.9731	0.9993	0.9956	0.9943

Table 5.24: Fault Efficiency

of sequential elements for higher fault-coverages. Some of these circuits, ex2, ex3 and lion9, show improved fault-coverages using integrated testability approach. This is because integrated-measure further incorporates initializability information in a circuit arising from the state-assignment. These observations show that state-assignment can render a circuit without a dedicated reset line, uninitializable. Incorporating initializability information helps to increase fault-coverage as is evident from the best overall fault-coverage obtained using *Integrated* measure.

However, integrated measure also suffers from certain inaccuracies by the use of Expand-function as discussed in section 3.5. Such an inaccuracy is highlighted in the case of s1 circuit that was predicted initializable by the initializability detection routine. The initialization in s1 circuit initiated from logic-low initialization on one of its sequential elements that rippled further initializations on other sequential elements. However, due to further simplifications during synthesis, the initial sequential element predicted as logic-low

	Tabu			GA			
	Tdepth	Int-OC	Int-QCC	Int-QCC	Jedi	Nova	Amaral
bbara	1.2	2.017	3.133	0.717	2.017	1.2	0.0617
bbsse	0.167	0.117	0.133	0.083	0.1	0.15	0.033
cse	0.383	0.117	0.183	0.167	0.267	0.217	0.1
dk14	0.033	0.017	0	0.017	0.117	0.033	0
ex2	17.283	2.567	2.517	4.483	28.067	35.683	0.733
ex3	0.767	0.15	0.617	0.083	0.583	2	0.1
keyb	6.017	14.95	14.567	1.5	0.867	2.55	1.133
lion9	0.467	0.05	0.033	0.15	0.017	0.017	0.717
planet	2.617	2.183	2.617	2.1	2.417	1.467	0.483
pma	0.017	0.167	0.033	4.617	0.017	0.033	0.017
s1	30.23	28.73	31.25	35.58	28.3	1.617	12.483
s1494	0.017	7.817	7.783	8.9	2.783	4.733	3.167
s832	0.017	11.25	9.117	9.2135	4.083	6.95	1.197
sand	1.2	1.383	1.2	1.367	1	1.667	0.367
shiftreg	0.05	0.017	0.033	0.017	0.017	0.033	0
styr	2.783	2.6	2.783	1	0.4	1.533	0.117
tbk	3.483	3.417	3.7	3.245	0.017	244.817	17.367
train11	0.05	0.017	0.067	0.033	0.067	3.033	0
Average	3.7101	4.3092	4.4314	4.071	3.952	17.0963	2.1153

Table 5.25: CPU Time

initializable did not remain logic-low initializable, rendering s1 circuit uninitializable. A similar erroneous initializability was predicted for lion9 circuit using QCC and GA.

Circuits, planet, sand and styr, that were predicted uninitializable by the initializability routine also remained uninitializable after synthesis. The validity of their uninitializability can also be seen from their low fault-coverages in all the measures.

The inconsistency in initialization estimate can be corrected by using actual synthesized cover in place of Expand-function for the last few iterations. The search process can then be guided to select initializable assignment. As most of the initialization estimates using Expand-cover show a high degree of accuracy, correction using exact cover can be achieved with minimal cost.

There are three cases where integrated-measure using original and QCC differ. Two of the cases, s832 and pma, were ran for lesser number of iterations with OC as they required excessive processing due to their wider (higher

number of inputs) and bigger (higher number of terms) cover sizes. However, as the complexity of QCC is linear with respect to cover size, the two circuits were ran till completion using the heuristic. An improvement in the fault-coverages is thus seen with the two circuits using QCC.

The third case where OC and QCC differ, ex3, was analyzed in more detail. It was seen that QCC sometimes fail to correctly check the presence of complement. For example, a sample cover from the circuit is shown below

a	b
01	10
10	01
01	10
10	10

There is an equal probability that variable-a be selected logic-low or logic-high by the QCC heuristic as all the terms carry equal weights. A selection of logic-low on variable-a may however cause the QCC to incorrectly predict absence of complement from the cover. The situation occurred due to presence of two identical terms in the cover. QCC can thus be modified to take care of such duplicates in the input cover for a more accurate estimation. However, probability of such a situation is fairly low, as can be seen from its occurrence in only one circuit, and is mostly limited to narrow (small number of inputs) sized covers like the one above. Such covers can also be quickly processed using OC and thus another solution could be to do error correction in last few iterations using OC for smaller input sized covers. It should also be noted that QCC prediction is a subset of OC prediction and thus any presence of complement predicted using QCC will always be true using OC.

A comparison between GA and TS using the most efficient integrated quick complement technique shows TS being more efficient in exploring testable solutions.

Integrated measure using QCC is seen to be an efficient heuristic and will now be used in the remaining set of experiments for estimation of testability of a circuit in this thesis.

5.7.1 Literature Comparison

A comparison of Integrated-QCC with those available in literature is next presented. The comparison is done with a recent work by Ciesielski et al [96]. Table-5.26 gives the comparison of respective fault-coverages achieved.

	Int-QCC	Ciesielski
bbsse	1	0.9913
keyb	0.965	0.9203
s1494	0.9976	0.9866
s832	0.9989	0.9598
tbk	0.9627	0.9914
Average	0.98484	0.96988

Table 5.26: Fault Coverage comparison with Ciesielski[96]

The results show an improvement in fault-coverage using our method as compared to Ciesielski. Fault-coverages as compared between the technique validate the benefit of integrated testability measure.

5.8 Area, Power and Testabilty

This section presents integration of all the three objectives, area, power and testability, that are the focus of this thesis. Fuzzy logic is used as integration mechanism. The work employs fuzzy logic with both *Min* and *Max* operators as discussed in chapter-2 using TS and GA as search algorithms. A combined *MaxMin* strategy is also used that initially combines area and power objectives using the better performing Max-operator as seen in section 5.6. The two are later combined with testability estimate using Min-operator. Genetic Algorithm is also used for comparison with the best performing approach. Tables-5.27 to 5.31 summarize the individual area, power and testability results achieved using the various integration mechanisms. The tables also provide comparison with previously obtained best results of the objective of interest of the table when it was optimized as a singe-optimization objective. The comparison is provided in the last column in the tables.

	Tabu			GA	
	Max	Min	MaxMin	Min	Best
bbara	57	57	57	55	55
bbsse	121	120	118	131	115
cse	212	202	212	208	213
dk14	109	109	109	111	108
donfile	72	49	64	97	75
ex2	87	85	85	135	81
ex3	55	58	58	63	57
keyb	176	181	201	241	154
lion9	11	14	13	24	12
planet	480	502	502	545	434
pma	176	153	156	235	154
s1	531	159	169	222	158
s1494	527	539	582	593	493
s832	252	247	249	263	222
sand	730	500	500	527	494
shiftreg	2	2	2	21	2
styr	780	407	376	544	412
tbk	351	347	344	893	386
train11	23	23	26	27	21
Average	250.1053	197.5789	201.2105	259.7368	191.8421

Table 5.27: Integrated Area

The area results show that Min and MaxMin types of integration using TS are performing very closely to the optimal area results. However, the power results clearly hold better for Min type integration using TS. MaxMin type integration, which has the effect of optimizing for both testability and best of area or power, is also seen to be performing below par in terms of testability from Min type integration which is seen to be performing very close to the best achieved testability results. The reason lies in an unpredictable way testability measure interacts with area and power integration in the MaxMin type of integration, effectively giving more weight in optimizing either testability with area or testability with power.

The inaccuracy in testability estimate is reflected once again in the Min type and MaxMin type of integrations. The low fault-coverage of pma circuit in the techniques is because of the inaccuracy and can be handled using the proposed correction mechanism. However, the low fault-coverages in Max-type integration cannot be attributed to the inaccuracy as Max integration is biased towards optimizing either of the three objectives. The biasing has a preference for the most optimized of the objectives which could also be an objective other than testability.

	Tabu			GA	
	Max	Min	MaxMin	Min	Best
bbara	166.6	166.6	166	151.5	181.2
bbsse	484.9	433.6	439.6	471.2	436.8
cse	438.9	501.7	520	443.2	485.4
dk14	592.1	592.1	592.1	560.3	529.6
donfile	375.6	251.7	300.3	451.2	286.3
keyb	536.9	487.9	518.4	605.4	505.8
lion9	132.5	97.4	98.4	178.7	97.4
planet	1852.2	2000.5	2000.5	2145	1670.7
pma	668.2	698.5	708.2	1033.2	607.1
s1	1802.6	605.3	656.3	904.9	734.4
s1494	1021.1	1130.9	1494.5	1374	838.5
s832	711.9	648.5	670.4	672.1	627.4
sand	2088.2	1425	1425	1478.3	1254.2
shiftreg	96.3	96.3	96.3	243.1	96.3
styr	2314.2	1197.3	961.8	1313.6	1016.1
tbk	993.9	958.3	998.2	2456.5	886.2
train11	114.2	113.8	171.1	145.7	122.2
Average	846.4882	670.9059	695.1235	860.4647	610.3294

Table 5.28: Integrated Power

	Tabu			GA	
	Max	Min	MaxMin	Min	Best
bbara	0.8365	0.9538	0.8636	0.9191	0.8333
bbsse	0.9406	1	0.931	1	1
cse	0.0157	0.9912	0.0124	0.9955	1
dk14	0.8598	1	0.8598	1	1
ex2	0	0.0504	0.0504	0	0.7528
ex3	0.9178	0.9576	0.9576	0.9231	0.5909
keyb	0.9821	0.995	0.8935	0.9983	0.965
lion9	0.8333	0.9211	0.9048	0.8475	0.8689
planet	0.007	0.0069	0.0036	0.0065	0.0047
pma	0.0058	0.0032	0.0022	1	0.9964
s1	0.0051	0.9863	0.9231	0.9231	0.0019
s1494	0.9953	0.9972	0.9067	0.8597	0.9976
s832	0.9858	0.9904	0.9944	0.9377	0.9989
sand	0.002	0.0031	0.0045	0.0028	0.0026
shiftreg	1	1	1	0	1
styr	0.0031	0.0051	0.0074	0.0053	0.0042
tbk	0.98	1	1	0.8583	0.9627
train11	1	0.9767	0.9275	0.9375	0.945
Average	0.5761	0.7132	0.6246	0.6786	0.71805

Table 5.29: Integrated Fault Coverage

	Tabu			GA	
	Max	Min	MaxMin	Min	Best
bbara	1	1	0.9805	1	1
bbsse	1	1	1	1	1
cse	1	1	1	1	1
dk14	1	1	1	1	1
ex2	0.9953	1	1	0.9971	1
ex3	1	1	1	1	1
keyb	1	1	1	1	1
lion9	1	1	1	1	1
planet	1	1	1	1	1
pma	0.9971	1	1	1	1
s1	0.7821	1	1	1	0.646
s1494	1	1	0.9994	1	1
s832	0.9878	0.9942	0.9944	0.9967	1
sand	1	1	1	1	1
shiftreg	1	1	1	1	1
styr	1	1	1	1	1
tbk	1	1	1	0.9486	0.9996
train11	1	1	1	1	1
Average	0.986794	0.999678	0.998572	0.9968	0.980311

Table 5.30: Integrated Fault Efficiency

	Tabu			GA	
	Max	Min	MaxMin	Min	Best
bbara	0.5	0.3	2.6	0.217	3.133
bbsse	0.033	0	0.017	0.017	0.133
cse	0.217	0.183	0.1	0.067	0.183
dk14	0.083	0	0.133	0.017	0
ex2	3.533	1.983	1.983	21.483	2.517
ex3	0.05	0.083	0.083	0.483	0.617
keyb	0.333	0.183	1.317	0.683	14.567
lion9	0.017	0.033	0.017	0.017	0.033
planet	0.55	0.683	1.683	0.483	2.617
pma	0.817	0.05	0.033	0.017	0.033
s1	26.24	0.117	0.583	0.467	31.25
s1494	1.833	1.767	2.15	3.717	7.783
s832	1.517	2.017	2	0.883	9.117
sand	1.233	0.083	0.517	0.233	1.2
shiftreg	0	0.017	0.017	0.033	0.033
styr	0.8	0.133	0.5	0.8	2.783
tbk	0.067	0.083	0.017	17.917	3.7
train11	0.033	0.033	0.017	0.083	0.067
Average	2.103111	0.430444	0.764833	2.645389	4.431444

Table 5.31: Integrated CPU Time

The Min type of integration, that tries to optimize all the objectives together, is observed to be the best integration technique for all the three objectives. In the case of area and testability, the Min integration achieves close to previously best found results while depicting a moderate loss in quality of power solutions. This work utilized Min type of integration with equal weights to the objectives for a fair comparison with other integration techniques. In a multiobjective optimization environment, where a designer has a set of relative priorities of the objectives, the above set of experiments can be used to find the relative weights in optimizing individual objectives according to the priorities. For e.g, priority of power objective can be increased to trade it with area and/or testability objectives. The current set of experiments can then be used to guide in the selection of priority weight to be given to the power objective that corresponds to its optimization priority.

5.8.1 Literature Comparison

There is a shortage of work that simultaneously addresses area, power and testability objectives for FSM state assignment problem in literature. Although, a similar work could not be found in literature, a comparison with recent work by Ciesielski et al [96] that addresses power and testability objectives is reused for comparison.

A relative comparison comparing percentage power reduction from Jedi state-assignment heuristic between Ciesielski and our Min-type integration is presented in Table-5.32. Testability results comparing the respective fault-coverages is tabulated in Table-5.33

%red	Celiski	MIN
bbsse	5.66	19.52487
keyb	35.56	36.43825
s1494	6.89	32.2368
s832	7.75	39.30176
tbk	5.03	-32.8758
Average	12.178	18.92518

Table 5.32: Integrated CPU Power

It can be seen from the tables that our aggregation using OWA-MIN operator, that further carries the effect of area estimate in the aggregation, still achieves better results with objectives of Ciesielski et al's work.

	Ciesielski	MIN
bbsse	0.9913	1
keyb	0.9203	0.995
s1494	0.9866	0.9972
s832	0.9598	0.9904
tbk	0.9914	1
Average	0.96988	0.99652

Table 5.33: Integrated CPU Fault-Coverage

5.9 Conclusion

In this chapter, several optimization heuristics for FSM area, power and testability and their search strategies employing non-deterministic algorithms, Tabu-Search and Genetic-Algorithms, are compared. Initially, the search algorithms were experimented with to optimize their exploration capabilities. Various parameters of the search heuristics were empirically evaluated for later experimentations. It was seen that an adaptive TS is more efficient in search space exploration than GA.

Heuristics for FSM multilevel area are next evaluated and compared with results available in literature or obtained from their implementations. It was seen that Expand-SO shows a high level of correlation with FSM's multilevel area. The quality of results are also seen to be close to optimal results obtained using ESPRESSO synthesis and performing fast-extraction. ESPRESSO is an efficient synthesis tool that iteratively utilizes Expand-function along with several different heuristics in synthesizing a circuit. A single application of Expand is thus seen to be achieving close to optimal results while being many times faster than ESPRESSO.

The cover returned by Expand-function is further utilized in optimization for FSM power. A new Fanout approach is proposed that uses the cover along with switching information to minimize logic being switched. The Fanout heuristic is integrated with Expand-SO area estimate and seen to be better optimizing FSM power than recently reported heuristics in the literature. Similarly, Expand cover is reutilized in developing a new efficient Integrated-testability estimate. A new method for quickly checking presence or absence of complement in the Expand cover is also proposed that further enhances the efficiency of the testability measure. The optimized testability estimate, Integrated-QCC, was compared with a recently proposed work and was ob-

served to be more efficient.

Finally, the three objectives of this work are fuzzy-integrated by using combination OWA operators. The OWA parameter β is used as 0.5 to have equal weight distribution between degree of anding/oring and averaging in OWAO aggregation. The OWA aggregation using MIN operator was seen to be best in integration, performing close to optimal for area and testability objectives, while showing a moderate loss in power solutions quality. Although no work combining all the three objectives could be found in the literature, the combination was compared with a recent work employing integration of power and testability objectives and was still observed to be performing better.

This chapter presented a detailed empirical examination of various heuristics and search algorithms for optimizing FSM state assignment problem for multilevel area, power and testability objectives. The results achieved demonstrate the benefit of using the proposed heuristics, in particular Expand-function, whose minimal extra cost is used for significant savings in all the three objectives.

Chapter 6

Conclusion and Future Work

6.1 Conclusion

FSM state assignment problem (SAP) for optimization of area, power and testability objectives has attracted considerable amount of interest in the research community. Most of the work reported in literature addresses SAP for either single or dual-objectives among the set of objectives that are focus of this thesis. This work addresses FSM SAP for multilevel area, power and testability objectives for single as well multiobjective optimization problems (MOP) as one of the first works to combine them.

The objectives of interest are carefully analyzed and heuristics previously addressing them are detailed and experimented with in search for better estimates. Expand-function, that is also utilized in ESPRESSO synthesis tool, was seen to offer both abstraction in estimation as well as improvement of solution quality. Expand single-output (SO) function achieved significant literal savings as compared to previously proposed heuristics. The cost of Expand-function was further amortized over power and testability objectives. Information available in Expand-SO cover was reutilized along with switching information to achieve further savings in FSM power consumption. Similarly, Expand-SO cover is used in initialization estimation for sequential elements to yield a better testability estimate.

As FSM state assignment is NP-Hard problem, the search cannot be performed using exhaustive enumeration. Therefore intelligent heuristics are used to get suboptimal results of the solution in feasible amount of times. In this thesis, Genetic Algorithm (GA) and Tabu-Search (TS) are utilized as search space exploration tools. The work thus further experiments in developing/tuning the search algorithms. An efficient set of search parameters

In order to solve MOP, fuzzy logic based aggregation function is used where user preferences are given in terms of fuzzy goal vectors. Various types of aggregation functions were experimented with involving combination of the objectives.

Following are the conclusions of this research

- The use of adaption mechanism in TS makes the exploration using TS more efficient. TS is further seen to be better with GA in search space exploration.
- Expand-SO provides a good quick estimation for multilevel area. The results obtained using Expand-SO as cost estimate were only 20% deteriorated to savings achieved if a more accurate estimate using ESPRESSO synthesis followed by fast extraction is used. The slight depreciation in quality, however, comes with many times savings in processing cost over the more accurate measure.
- Proposed *Fanout* measure performs better than traditional MWHD measure.
- State assignment can render a circuit without an explicit reset uninitializable and thus untestable. By incorporating initializability information due to an assignment, an increased fault-coverage is obtained by the proposed *integrated-testability* measure.
- Proposed *quick complement check* heuristic can quickly check for presence of input cover-complement with linear complexity and high level of accuracy.

6.2 Future Research

This work can be extended to cover the following issues

- In this work, Fanout measure tries to minimize switched area by minimizing the number of fanouts out of frequently switching sequential elements. In order to have a more accurate switched-area reduction, area being switched by primary inputs can be coupled with with the current fanout-estimate.
- QCC heuristic is observed to have a high level of accuracy. However, QCC may not detect presence of complement in certain input covers.

The probability of such an inaccuracy is inversely proportional to the width of input cover. For a more accurate estimation, QCC heuristic can be improved by running a preprocessing step of removing duplicates in the input cover. Another correction strategy could be to utilize OC for QCC verification either intermittently or in the last few iterations of the simulation.

- Expand-SO, being a quick estimate, is seen to possess certain inaccuracies. The final cover can get altered from Expand-SO cover by successive simplification heuristics used in the synthesis process. Such an inaccuracy was observed to impart a noticeable degree of uncertainty in the testability estimate. The issue can again be addressed by using exact final cover for error detection/correction of Expand-SO cover either intermittently or in the last few iterations of the simulation.

- A state machine may have some don't-care states equal to the difference between the number of valid states of the machine and possible number of states (see equation-2.12). For correct operation of the machine, it is essential that there are no transitions from valid domain to the invalid don't-care domain. Furthermore, it is also required that the machine quickly traverses itself into the valid domain if it gets started in the other one. Although the former is taken care of in the definition of the machine itself, the latter is a subject of state-assignment. Constraining state-assignment to further achieve quick valid domain traversal can thus be an interesting idea to explore.

- The presence of don't-care states along with requirement to have only valid domain traversals may effect testing time for a machine. The testing time may abhorrently increase if justification sequence required by the testability tool needs certain flip-flop initializations that are possible only through traversing to the invalid domain. The issue can be addressed by further constraining the assignment process so to have all possible initializations available through the valid domain.

Appendix A

Solving Discrete-Time Markov Chains

When the number of states in an ergodic, discrete-time Markov chain is finite, we can solve for the steady-state probability vector in several ways. Although the computations could be performed by hand for small problems, Matlab provides simple and efficient operations for finding the solution and can be used even when the number of states is large. Define the state probability vector of a discrete-time Markov chain with m states after the nth transition, given some initial state probability vector p(0), to be

$$\mathbf{p(n)} = \begin{bmatrix} [p(n)]_1 \\ [p(n)]_2 \\ ..... \\ [p(n)]_m \end{bmatrix}$$

where $[p(n)]_i$ is the probability that the system is in state i after transition n, given $p(0)$. Recall that $P = (p_{jk})$ is the single-step transition probability matrix: p_{jk} is the probability that the next state will be k given that the current state is j. If we know the state probability vector at time n, we can compute the ith component of the vector at time n+1 as

$$[p(n+1)]_i = [p(n)]_1 p_{1i} + [p(n)]_2 p_{2i} + \ + [p(n)]_m p_{mi} \tag{A.1}$$

The set of m such equations can be summarized as p(n+1)=p(n)P. These are called *Chapman-Kolmogorov* equations.

As an example, consider the Markov chain described by the state transition diagram

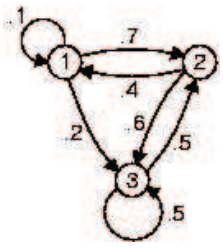


Figure A.1: A state machine

The single-step transition probability matrix is

$$\mathbf{P} = \begin{bmatrix} 0.1 & 0.4 & 0 \\ 0.7 & 0 & 0.5 \\ 0.2 & 0.6 & 0.5 \end{bmatrix}$$

This is clearly an ergodic Markov chain. If the initial state probability vector is $p(0) = (0.3, 0.4, 0.3)^T$, we can compute the evolution of the state probability vector for as many transitions as we want by repeated applications of the Chapman-Kolmogorov equations.

$[p(1)]_1 = (0.3)(0.1) + (0.4)(0.4) + (0.3)(0) = 0.19$
$[p(1)]_2 = (0.3)(0.7) + (0.4)(0) + (0.3)(0.5) = 0.36$
$[p(1)]_3 = (0.3)(0.2) + (0.4)(0.6) + (0.3)(0.5) = 0.45$
$[p(2)]_1 = (0.19)(0.1) + (0.36)(0.4) + (0.45)(0) = 0.163$
$[p(2)]_2 = (0.19)(0.7) + (0.36)(0) + (0.45)(0.5) = 0.358$
$[p(2)]_3 = (0.19)(0.2) + (0.36)(0.6) + (0.45)(0.5) = 0.479$

The resulting sequence of state probability vectors is
$p(0) = (0.3, 0.4, 0.3)^T$
$p(1) = (0.19, 0.36, 0.45)^T$
$p(2) = (0.163, 0.358, 0.479)^T$
$p(3) = (0.1595, 0.3536, 0.4869)^T$
$p(5) = (0.1578, 0.3539, 0.4883)^T$
$p(6) = (0.1573, 0.3546, 0.4881)^T$
$p(7) = (0.1576, 0.3542, 0.4883)^T$
$p(8) = (0.1574, 0.3544, 0.4881)^T$
$p(9) = (0.1575, 0.3543, 0.4882)^T$
$p(10) = (0.1575, 0.3544, 0.4882)^T$
.....
$p(11) = (0.1575, 0.3543, 0.4882)^T$

As expected, the state probability vectors are converging to the steady state probability vector p. After the 11th transition, the state probabilities remain unchanged to four decimal places. The Ergodicity Theorem tells us not only that the state probability vector will converge, but that the steady-state probability vector is unique and does not depend on the initial state. If we start with $p(0) = (1, 0, 0)^T$ and apply the same procedure, we approach the same result. Convergence is a little slower than for the previous initial state probability vector because the new initial vector is farther from steady state

$p(0) = (1, 0, 0)^T$
$p(1) = (0.1, 0.7, 0.2)^T$
$p(2) = (0.29, 0.17, 0.54)^T$
$p(3) = (0.097, 0.473, 0.430)^T$
$p(4) = (0.1989, 0.2829, 0.5182)^T$
$p(5) = (0.1331, 0.3983, 0.4686)^T$
$p(6) = (0.1726, 0.3274, 0.4999)^T$
$p(7) = (0.1482, 0.3708, 0.4810)^T$
$p(8) = (0.1631, 0.3442, 0.4926)^T$
$p(9) = (0.1540, 0.3605, 0.4855)^T$
$p(10) = (0.1596, 0.3505, 0.4898)^T$
$p(11) = (0.1562, 0.3566, 0.4872)^T$
....
$p(20) = (0.1575, 0.3542, 0.4882)^T$

Matlab software can be used to solve the above set of equations to calculate steady-state probabilities. One way for determining the steady-state probabilities is to treat the problem as one of solving the set of linear equations represented by $\pi^P = \pi$. However, P is a stochastic matrix (each of its rows sums to 1) and hence is singular. One other independent equation in the πs is thus required to go with any $m - 1$ of the m equations from $\pi^P = \pi$. Fortunately, we always have one: the *normalization equation*

$$\sum_{i=1}^{m} \pi_i = 1 \tag{A.2}$$

To implement this method, first any column i (for instance, the last column, representing the equation for π_m), is replaced with 1s, corresponding to the normalization equation. P1 can be entered in Matlab like

$$\mathbf{P1} = \begin{bmatrix} 0.1000 & 0.7000 & 1.0000 \\ 0.4000 & 0.0000 & 1.0000 \\ 0 & 0.5000 & 1.0000 \end{bmatrix}$$

or using the following commands, which is equivalent to above if P has been previously entered.

$\gg P1 = P;$

$\gg P1(:, 3) = [111]'$

An mxm identity matrix is next entered with the ith diagonal element replaced by 0.

$$\mathbf{J} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix}$$

which can also be entered using the following Matlab command

$\gg J = diag([110], 0)$

Finally, the system of linear equations $\pi(P1 - J) = (0, 0, , 0, 1, 0, , 0)^T$ is solved where $P1$ is P with the ith column replaced by 1s. J is the diagonal matrix with 1s along the diagonal except for a 0 in the ith diagonal element, and the vector on the right-hand side is all 0's except for a 1 in the ith component. This can be done by Matlab commmand

$\gg \pi = [001]/(P1 - J)$

that gives

$$\pi = \begin{bmatrix} 0.1575 & 0.3543 & 0.4882 \end{bmatrix}$$

which is the same set of steady state probabilities as earlier achieved.

Appendix B

Steady State Probabilities of the Benchmark Circuits

	bbara	**bbsse**	**dk14**	**ex3**	**lion9**	**shiftreg**	**train11**
S1	1.55E-01	2.25E-01	1.88E-01	1.00E+00	1.11E-01	1.25E-01	1.67E-01
S2	2.67E-01	2.14E-01	1.88E-01	0.00E+00	1.11E-01	1.25E-01	8.33E-02
S3	1.33E-01	1.38E-02	2.43E-01	0.00E+00	1.11E-01	1.25E-01	8.33E-02
S4	1.33E-01	3.46E-03	1.25E-01	0.00E+00	1.11E-01	1.25E-01	8.33E-02
S5	1.97E-01	1.45E-02	1.87E-01	0.00E+00	1.11E-01	1.25E-01	8.33E-02
S6	4.92E-02	3.63E-03	5.38E-02	0.00E+00	1.11E-01	1.25E-01	8.33E-02
S7	1.64E-02	2.59E-04	1.57E-02	0.00E+00	1.11E-01	1.25E-01	8.33E-02
S8	3.74E-02	3.24E-05		0.00E+00	1.11E-01	1.25E-01	8.33E-02
S9	9.36E-03	9.25E-06		0.00E+00	1.11E-01		8.33E-02
s10	2.34E-03	1.18E-06		0.00E+00			8.33E-02
S11		3.14E-07					8.33E-02
S12		4.50E-01					
S13		7.50E-02					

	cse	donfile	ex2	keyb	s1
S1	8.65E-01	4.17E-02	1.00E+00	7.22E-01	6.04E-02
S2	2.89E-02	4.17E-02	0.00E+00	4.51E-02	7.06E-02
S3	1.86E-03	4.17E-02	0.00E+00	1.35E-01	1.05E-02
S4	6.21E-05	4.17E-02	0.00E+00	9.02E-02	1.69E-02
S5	4.14E-06	4.17E-02	0.00E+00	3.52E-04	1.02E-02
S6	6.67E-05	4.17E-02	0.00E+00	6.34E-03	1.47E-02
S7	3.38E-02	4.17E-02	0.00E+00	7.05E-04	1.05E-01
S8	6.51E-03	4.17E-02	0.00E+00	5.51E-06	1.16E-01
S9	2.98E-02	4.17E-02	0.00E+00	1.76E-04	8.55E-02
s10	3.19E-02	4.17E-02	0.00E+00	1.10E-05	6.61E-02
S11	2.13E-03	4.17E-02	0.00E+00	1.72E-07	1.09E-02
S12	1.42E-04	4.17E-02	0.00E+00	7.74E-06	7.71E-02
S13	2.03E-05	4.17E-02	0.00E+00	3.44E-07	1.25E-02
S14	1.58E-06	4.17E-02	0.00E+00	2.15E-08	1.14E-01
S15	6.60E-08	4.17E-02	0.00E+00	1.12E-06	3.12E-02
S16	5.11E-08	4.17E-02	0.00E+00	5.38E-09	2.75E-02
S17		4.17E-02	0.00E+00	2.90E-07	1.93E-02
S18		4.17E-02	0.00E+00	2.69E-09	1.20E-01
S19		4.17E-02	0.00E+00	3.51E-08	1.92E-02
S20		4.17E-02			1.27E-02
S21		4.17E-02			
S22		4.17E-02			
S23		4.17E-02			
S24		4.17E-02			

	planet	pma	s832	s1494	sand	styr	tbk
S1	1.35E-02	1.14E-01	6.40E-01	8.10E-01	1.04E-01	6.38E-01	3.42E-01
S2	5.39E-02	9.11E-02	3.79E-03	1.37E-05	1.84E-02	2.04E-02	5.79E-03
S3	4.15E-02	1.37E-01	4.83E-03	3.70E-08	4.80E-03	1.45E-02	5.79E-03
S4	3.58E-02	1.14E-01	6.44E-04	3.85E-06	7.07E-04	6.60E-04	5.79E-03
S5	3.13E-02	5.69E-02	7.32E-04	4.31E-03	7.07E-04	2.16E-05	5.79E-03
S6	3.13E-02	9.49E-02	3.05E-06	4.89E-07	9.90E-02	2.06E-05	5.79E-03
S7	4.98E-02	1.52E-01	2.03E-07	6.51E-08	1.65E-02	3.07E-05	5.79E-03
S8	4.05E-02	3.80E-02	1.70E-09	1.26E-03	4.38E-02	3.30E-04	5.79E-03
S9	4.05E-02	1.52E-02	5.65E-11	3.15E-09	7.01E-02	1.07E-02	5.79E-03
s10	2.96E-02	2.28E-02	3.77E-12	3.45E-07	4.38E-03	1.60E-02	5.79E-03
S11	2.02E-02	8.69E-02	2.51E-13	3.36E-08	8.25E-03	9.45E-02	5.79E-03
S12	2.96E-02	1.45E-02	3.69E-09	4.99E-09	2.06E-03	2.10E-02	5.79E-03
S13	2.02E-02	1.81E-03	4.69E-10	8.76E-04	8.25E-03	1.66E-01	5.79E-03
S14	2.96E-02	4.52E-04	1.25E-10	1.29E-06	6.19E-03	1.28E-02	6.56E-02
S15	2.02E-02	4.52E-04	2.00E-02	1.01E-01	6.19E-03	3.49E-03	1.16E-02
S16	4.98E-02	1.48E-05	1.00E-02	3.16E-04	1.24E-02	2.18E-04	1.16E-02
S17	4.98E-02	2.37E-04	2.13E-01	1.27E-02	1.24E-02	1.96E-04	3.42E-01
S18	4.98E-02	3.16E-05	1.07E-01	2.19E-04	1.86E-02	1.96E-05	5.79E-03
S19	2.80E-02	1.98E-06	8.48E-09	1.30E-07	1.86E-02	1.23E-06	5.79E-03
S20	1.49E-02	1.24E-07	6.02E-11	8.31E-03	2.48E-02	1.36E-05	5.79E-03
S21	7.47E-03	2.85E-02	2.69E-13	1.66E-02	2.48E-02	1.36E-06	5.79E-03
S22	3.73E-03	2.85E-02	6.17E-13	2.74E-05	3.09E-02	2.18E-06	5.79E-03
S23	1.87E-03	3.56E-03	9.25E-12	1.49E-04	3.09E-02	1.25E-04	5.79E-03
S24	1.87E-03	2.22E-04	1.70E-09	1.75E-03	3.71E-02	4.31E-06	5.79E-03
S25	1.12E-02		1.39E-10	3.80E-02	3.71E-02	1.35E-07	5.79E-03
S26	3.73E-02			2.49E-09	4.33E-02	4.04E-06	5.79E-03
S27	3.73E-02			1.38E-06	4.33E-02	1.39E-07	5.79E-03
S28	1.40E-02			2.19E-04	4.95E-02	1.54E-07	5.79E-03
S29	9.34E-03			8.62E-08	4.95E-02	3.47E-04	5.79E-03
S30	9.34E-03			5.47E-05	5.57E-02	1.64E-05	6.56E-02
S31	1.87E-02			1.97E-07	5.57E-02		1.16E-02
S32	1.87E-02			5.47E-05	6.19E-02		1.16E-02

	planet	**s1494**
S33	1.07E-02	1.04E-06
S34	5.34E-03	2.53E-03
S35	4.00E-03	2.16E-08
S36	9.34E-03	2.88E-04
S37	9.34E-03	9.97E-09
S38	9.34E-03	3.72E-05
S39	4.98E-03	1.09E-04
S40	1.87E-03	1.22E-07
S41	2.49E-03	2.60E-07
S42	8.71E-03	1.09E-04
S43	1.85E-02	2.16E-08
S44	1.85E-02	6.32E-04
S45	1.85E-02	2.79E-05
S46	1.85E-02	1.30E-07
S47	6.22E-03	5.47E-05
S48	3.11E-03	8.40E-09

Bibliography

[1] N. H. E. Weste and K. Eshraghian. *Principles of CMOS VLSI Design: A systems perspective, Second edition.* Addison-Wesley, 1993.

[2] M. Pedram. Design technologies for low power VLSI. *In Encyclopedia of Computer Science and Technology, Marcel Dekker, Inc.*, pages 73–96, 1997.

[3] Miron Abramovici, Melvin A. Breuer, and Arthur D. Friedman. *Digital Systems Testing And Testable Design.* IEEE Press, 1990.

[4] Z. Kohavi. *Switching and Finite Automata Theory, Second edition.* McGraw-Hill Book Co., 1978.

[5] Sadiq M. Sait and Habib Youssef. *Iterative Computer Algorithms with Applications in Engineering: Solving Combinatorial Optimization Problems.* IEEE Computer Society Press, California, December 1999.

[6] R. E. Stearns and J. Hartmanis. On the state assignment problem for sequential machines II. *IRE Transaction EC-10*, 1961.

[7] H. Allen Curtis. Multiple reduction of variable dependency of sequential machines. *Journal of ACM*, 9(3):324–344, 1962.

[8] Bernhard Eschermann. State assignment for hardwired VLSI control units. *ACM Computing Survey*, 25(4):415–436, 1993.

[9] R. Amann and U. G. Baitinger. Optimal state chains and state codes in finite state machines. *IEEE Transaction Computer Aided Design*, CAD-8:153–170, 1989.

[10] G. Demicheli, R. K. Brayton, and A. Sangiovanni Vincenteli. Optimal state assignment for Finite State Machines. *IEEE Transaction on Computer Aided Design*, CAD-4:3, 269–285, 1985.

[11] G. DeMicheli. Symbolic design of combinational and sequential logic circuits implemented by two-level logic macros. *IEEE Transaction on Computer Aided Design*.

[12] J. L. Huertas and J. M. Quintana. A new method for the efficient state-assigmnent of PLA-based sequential machines. *In the Internatzonal Conference on Computer-Aided Digital Design*, pages 156–159, 1988.

[13] J. L. Huertas and J. M. Quintana. Efficiency of state assignment methods for PLA based sequential circuits. *IEE Proceedings E, Computer and Digital Techniques*, pages 247–253, 1989.

[14] T. Villa and A. Sangiovanni-Vincentelli. Nova: state assignment of finite state machines for optimal two-level logic implementations. pages 327–332, 1989.

[15] P. Weiner and Smith E. J. On the number of distinct state assignments for synchronous sequential machines. *IEEE Transaction on Elec Cornput. EC-16*, pages 220–221, 1967.

[16] G. DeMicheli, R.K. Brayton, and A. Sangiovanni-Vincentelli. Optimal State Assignment for Finite State Machines. *IEEE Transaction on CAD*, CAD-4(3).

[17] G. DeMicheli. Symbolic Design of Combinational and Sequential Logic Circuits Implemented by Two-level Logic Macros. *IEEE Trans. on CAD*, CAD-5(4).

[18] J. Hartmanis and R. E. Stearns. *Algebraic Structure Theory of Sequential Machines*. Prentice Hall, Englewood Cliffs, 1966.

[19] G. Saucier, M. Crastes de Paulet, and P. Sicard. ASYL: A Rule-based System for Controller Synthesis. *IEEE Transaction on CAD/ICAS*, Vol. CAD-6, No. 6.

[20] P. Ashar, S. Devadas, and A. Newton. *Sequential Logic Synthesis*. Kluwer Academic Publishers, Boston, MA, 1992.

[21] S. Devadas, H.T. Ma, A.R. Newton, and Sangiovanni-Vincentelli. MUSTANG: State Assignment of Finite State Machines for Optimal Multi-Level Logic Implememations. *ICCAD*, November 1987.

[22] B. Lin and A. R. Newton. Synthesis of multiple-level logic from symbolic high-level description languages. *IFIP International Conference on Very Large Scale Integration*, pages 187–196, August 1989.

[23] X. Du, G. Hachtel, B. Lin, and A. R. Newton. MUSE: A Multilevel symbolic encoding algorithm for state assignment. *IEEE Trans. Computer Aided Design*, CAD-10.

[24] A. Almaini, J. Miller, P. Thomson, and S. Billina. State Assignment of state machine using Genetic Algorithm. *IEEE Proc. on Computer and Digital Techniques*, pages 279 – 286, 1995.

[25] J. Amaral, K. Turner, and J. Ghosh. Designing Genetic Algorithm for State Assignment Problem. *IEEE Trans on SMC*, pages 659 – 694, 1995.

[26] D.B Armstrong. A programmed algorithm for assigning internal codes to sequential machines. *IRE Transactions on Electronic Computers*, pages 466 – 472, 1962.

[27] Gary D.; McMullen Curtis T.; Sangiovanni-Vincentelli Alberto L. Brayton, Robert K.; Hachtel. *Logic Minimization Algorithms for VLSI Synthesis*. Kluwer Academic Publishers,, 1984.

[28] Massoud Pedram. Power minimization in IC design: Principles and applications. *ACM Transaction on Design Automation of Electronic Systems*, 1(1):3–56, 1996.

[29] F.N. Najm. A survey of power estimation techniques in VLSI circuits. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 2:446 – 455, 1994.

[30] F.N. Najm. Power estimation techniques for integrated circuits. *IEEE/ACM International Conference on Computer-Aided Design*, 2:492 – 499, 1995.

[31] Chi-Ying Tsui, J. Monteiro, Massoud Pedram, Srinivas Devadas, A.M. Despain, and B. Lin. Power estimation methods for sequential logic circuits. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 3:404 – 416, 1995.

[32] Devadas S., Malik S., Keutzer K., and White J. Estimation of Average Switching Activity in Combinational and Sequential Circuits. *29th DAC*, pages 253–259, 1992.

[33] Devadas S. and Omnteiro J. Techniques for the Power Estimation of Sequential Logic Circuits Under User-Specified Input Sequences and Programs. *Int. Symp. on Low Power Design*, 1995.

[34] Gary D. Hachtel, Enrico Macii, Abelardo Pardo, and Fabio Somenzi. Probabilistic Analysis of Large Finite State Machines. *proceedings of DAC*, pages 270–275, 1994.

[35] E. M. Sentovich, K. J. Singh, L. Lavagno, C. Moon, R. Murgai, A. Saldanha, H. Savoj, P. R. Stephan, R. K. Brayton, and A. Sangiovanni-Vincentelli. SIS: A system for sequential circuit synthesis. Technical report, 1992.

[36] Farid N. Najm, Shashank Goel, and Ibrahim N. Hajj. Power estimation in sequential circuits. In *Proceedings of the 32nd ACM/IEEE conference on Design automation conference*, pages 635–640. ACM Press, 1995.

[37] Joseph N. Kozhaya and Farid N. Najm. Accurate power estimation for large sequential circuits. In *Proceedings of the 1997 IEEE/ACM international conference on Computer-aided design*, pages 488–493. IEEE Computer Society, 1997.

[38] V. Saxena, F. N. Najm, and I. N. Hajj. Monte-Carlo approach for power estimation in sequential circuits. *European Design and Test Conference*, pages 416 – 420, 1997.

[39] Mahadevamury Nemani and Farid N. Najm. Towards a High-Level Power Estimation Capability. *IEEE Transactions on Computer Aided Design of Integrated Cicuits and Systems*, pages 588 – 598, 1996.

[40] M. Nemani and F.N. Najm. High-level area and power estimation for VLSI circuits. *IEEE Transactions on Computer Aided Design of Integrated Cicuits and Systems*, 18.

[41] Gary D. Hatchel, Enrico Macii, Abelardo Pardo, and Fabio Somenzi. Markovian Analysis of Large Finite State Machines. *IEEE Transactions on CAD*, Vol. 15, Num. 12, Dec. 1996.

[42] A. W. Drake. *Fundamentals of Applied Probability Theory.* McGraw-Hill, 1967.

[43] A. Papoulis. *Random Variables and Stochastic Processes.* McGraw-Hill, 1984.

[44] Chi-Ying Tsui, Massoud Pedram, and Alvin M. Despain. Exact and approximate methods for calculating signal and transition probabilities in FSMs. In *Proceedings of the 31st annual conference on Design automation conference*, pages 18–23. ACM Press, 1994.

[45] Jose Monteiro, Srinivas Devadas, and Bill Lin. A methodology for efficient estimation of switching activity in sequential logic circuits. In *Proceedings of the 31st annual conference on Design automation conference*, pages 12–17. ACM Press, 1994.

[46] K. Trivedi. *Probability and Statistics with Reliability, Queuing and Computer Science Applications*. Prentice-Hall, 1982.

[47] Tzong-Dar Her, Wei Kang Tsai, F. Kurdahi, and Yulin Chen. Low-power driven state assignment of finite state machines. *IEEE Asia-Pacific Conference on Circuits and Systems*, pages 454 –459, 1994.

[48] S. K. Hong, I. C. Park, S. H. Hwang, and C. M. Kyung. State assignment in finite state machines for minimal switching power consumption. *IEEE Electronics Letters*, 30 Issue: 8, 1994.

[49] S. J. Wang and M. D. Horng. State assignment of finite state machines for low power applications. *IEEE Electronics Letters*, 32 Issue: 25, 1996.

[50] E. Olson and S.M. Kang. State assignment for low-power FSM synthesis using genetic local search. *IEEE Custom Integrated Circuits Conference*, pages 140 –143, 1994.

[51] V. Vamshi, T. Akhilesh, and R. Suresh. Re-encoding for Low Power State Assignment of FSMs. *ACM International Symposium on Low Power Design*, 1995.

[52] K. Roy and S. Prasad. SYCLOP: synthesis of CMOS logic for low power applications . *ICCD*, pages 464 –467, 1992.

[53] I. Lemberski, M. Koegst, S. Cotofana, and B. Juurlink. FSM non-minimal state encoding for low power. *23rd International Conference on Microelectronics*, pages 605 –608, 2002.

[54] M. Koegst, G. Franke, and K. Feske. State assignment for FSM low power design. *EURO-DAC*, pages 28 –33, 1996.

[55] Chunhong Chen, Jiang Zhao, and Majid Ahmadi. A semi-Gray encoding algorithm for low-power state assignment. *International Symposium on Circuits and Systems, ISCAS*, 5, 2003.

[56] P. Surti, L.F. Chao, and A. Tyagi. Low power FSM design using Huffman-style encoding. *European Design and Test Conference, ED&TC*, pages 521 –525, 1997.

[57] P. Bacchetta, L. Daldoss, D. Sciuto, and C. Silvano. Low-power state assignment techniques for finite state machines. *International Symposium on Circuits and Systems*, 2:641 –644, 2000.

[58] L. Daldoss, D. Sciuto, and C. Silvano. State encoding for low power embedded controllers. *International Symposium on Circuits and Systems, ISCAS*, 2:421 –424, 1998.

[59] L. Benini and G. DeMicheli. State encoding for low power embedded controllers. *IEEE Journal of Solid-State Circuits*, 30, 1995.

[60] T. Dolotta and E. McCluskey. The coding of internal states of sequential machines. *IEEE Trans. Electron. Ecomput.*, EC-13, 1964.

[61] Chi-Ymg Tsui, M. Pedram, Chih-Ang Chen, and A. M. Despain. Low-Power State Assignment Targeting Two And Multi-level Logic Implementations. *IEEE/ACM International Conference on Computer-Aided Design*, pages 82–87, 1994.

[62] I. Bhupathi and L. F. Chao. High-level area and power estimation for VLSI circuits. *ISCAS*, 4.

[63] Y. Xia and A. E. A. Almaini. Genetic algorithm based state assignment for power and area optimisation. *IEEE Proceedings Computers and Digital Techniques*, 149.

[64] G. Venkataraman, S. M. Reddy, and I. Pomeranz. GALLOP: genetic algorithm based low power FSM synthesis by simultaneous partitioning and state assignment. *IEEE 16th International Conference on VLSI Design*, pages 533 – 538, 2003.

[65] O. H. Ibarra and S. Sahni. Polynomially Complete Fault Detection Problems. *IEEE Transaction on Computers*, C-24, No. 3.

[66] J. P. Roth. Diagnosis of Automata Failure: A Calculus and a method. *IBM Journal of Research and Development*, 10, No. 4.

[67] J. P. Roth, W. G. Bouricius, and P. R. Schneider. Programmed Algorithms to Compute Tests to Detect and Distinguish Between Failures in Logic Cicuits. *IEEE Transactions on Electronic Computers*, EC-16, No. 10.

[68] P. Goel and B. C. Rosales. PODEM-X: an automatic test generation system for VLSI logic structures. *Proceedings of the Design Automation Conference, IEEE Computer Society Press*, pages 260–268, 1981.

[69] H. Faujiwara and T. Shimono. On the acceleration of test generation algorithms. *IEEE Transactions on Computers*, C-32, No. 12.

[70] T. Trischler. Incomplete scan pat with an automatic test gernaeration methodology. *Proceedings of the International Test Conference*, pages 153 – 162, 1980.

[71] A. Miczo. *Digital Logic Testing and Simulation.* Harper & Row Publishers, 1986.

[72] A. Lioy, P. L. Montessoro, and S. Gai. A complexity analysis of sequential ATPG. *IEEE International Symposium on Circuits and Systems*, pages 1946 – 1949, 1989.

[73] T. E. Marchok, A. El-Maleh, W. Maly, and J. Rajski. A complexity analysis of sequential ATPG. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 15.

[74] Thomas Edward Marchok. *Modelling the difficulty of Automatic Test Pattern Generation for Sequential Circuits.* PhD thesis, Carnegie-Mellon University, 1995.

[75] I. Pomeranz and K. T. Cheng. State assignment using input/output functions. *29th ACM/IEEE Design Automation Conference, 1992. Proceedings.*, pages 573 – 577, 1992.

[76] K. T. Cheng and V. D. Agrawal. Design of sequential machines for efficient test generation. *ICCAD*, pages 358 – 361, 1989.

[77] C. R. Mohan and P. P. Chakrabarti. EARTH: combined state assignment of pla-based fsm's targeting area and testability. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, pages 727 – 731, 1996.

[78] S. Chiusano, F. Corno, P. Prinetto, M. Rebaudengo, and M. Sonza Reorda. Guaranteeing testability in re-encoding for low power. *Sixth Asian Test Symposium*, pages 30 – 35, 1997.

[79] Eckart Zitzler. Evolutionary Optimization for Multiobjective Optimization: Methods and Applications. *DTS thesis, Swiss Federal Institute of Technology Zurich*, November 1999.

[80] G. Fandel and eds. T. Gal. Multiple Criteria Decision Making Theory and Applications. *Lecture Notes in Economics and Mathematics Systems. Berlin: Springer-Verlag*, 177, 1980.

[81] J. P. Ignizio. The Determination of a Subset of Efficient Solution via Goal Programming. *Computing and Operations Research*, 3(9), 1981.

[82] H.J. Zimmerman. *Fuzzy Set Theory and Its Applications.* Kluwer Academic Publishers, 3rd edition, 1996.

[83] Ali Sayed Hussain. Fuzzy Simulated Evolution Algorithm For VLSI Cell Placement. *Master Thesis, King Fahd University of Petroleum and Minerals, Dhahran*, December 1998.

[84] L. A. Zadeh. Outline of a New Approach to the Analysis of Complex Systems and Decision Processes. *IEEE Transaction Systems Man. Cybern*, SMC-3(1):28–44, 1973.

[85] L. A. Zadeh. The Concept of Linguistic Variable and its Application to Approximate Reasoning. *Information Science*, 8:199–249, 1975.

[86] R. Yager. Multiple Objective Decision-making Using Fuzzy Sets. *International Journal of Man-Machine Studies*, pages 9:375–382, 1977.

[87] R. Yager. Second Order Structures in Multi-criteria Decision Making. *International Journal of Man-Machine Studies*, pages 36:553–570, 1992.

[88] K. Shahookar and P. Mazumder. VLSI Cell Placement Techniques. *ACM Computing Surveys*, 2(23):143–220, June 1991.

[89] J. P. Cohoon and W. D. Paris. Genetic placement. *IEEE Transactions on Computer Aided Design*, pages 956–964, 1987.

[90] R.; Ostapko D. Hong, S; Cain. Mini: A heuristic approach for logic minimzation. pages 443–458, September 1974.

[91] J.H. Patel and T.M. Niermann. HITEC: A test generation package for sequential circuits. *Proceedings of European Design Automation Conference*, pages 214 – 215, 1991.

[92] Giovanni De Micheli. *Synthesis and Optimization of Digital Circuits*. McGraw-Hill Book Co., 1994.

[93] K. T. Cheng and V.D. Agrawal. A partial scan method for sequential circuits with feedback. *IEEE Transactions on Computers*, 39:544 – 548, April 1990.

[94] D.H. Lee and S.M. Reddy. On determining scan flip-flops in partial-scan designs. *IEEE International Conference on Computer-Aided Design, 1990. ICCAD-90*, pages 322 – 325, Nov. 1990.

[95] LGSynth89 Benchmark Suite. Available from World Wide Web:. *http* : $//www.cbl.ncsu.edu/CBL_Docs/lgs89.html$.

[96] Sangju. Park, Sangwook Cho, Seiyang Yang, and Maciej Ciesielski. A New State Assignment Technique for Testing and Low Power. *Annual ACM IEEE Design Automation Conference*, pages 510 – 513, 2004.

[97] S. Chattopadhyay and P.N. Reddy. Finite state machine state assignment targeting low power consumption. *IEEE Proceedings Computers and Digital Techniques*, 151.