Evolutionary Techniques for Multi-Objective VLSI Netlist Partitioning

by

Raslan Hashim AL-Abaji

A Thesis Presented to the
DEANSHIP OF GRADUATE STUDIES

In Partial Fulfillment of the Requirements
for the Degree of

MASTER OF SCIENCE

IN

COMPUTER ENGINEERING

KING FAHD UNIVERSITY OF PETROLEUM & MINERALS

Dhahran, Saudi Arabia

July 2002

KING FAHD UNIVERSITY OF PETROLEUM AND MINERALS

DHAHRAN 31261, SAUDI ARABIA

DEANSHIP OF GRADUATE STUDIES

This thesis, written by

Raslan Hashim AL-Abaji

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. Sadiq M. Sait (Chairman)

Dr. Aiman H. El – Maleh(CO – Chairman)

Dr. Khan M. Farrukh (Member)

Department Chairman

Dean, College of Graduate Studies

Date

Heartily dedicated to my family and
especially to my dear Father and Mother

Acknowledgements

All praise to Allah, the most Merciful, who enabled me to complete my thesis work. I make a humble effort to thank Allah for his endless blessings on me, as His infinite blessings cannot be thanked for. Then, I pray Allah to bestow peace on his last prophet Muhammad (Sal-allah-'Alaihe-Wa-Sallam) and on all his righteous followers till the day of judgement.

I pay a hearty tribute to all my family members, especially to my parents, who guided me during all my life endeavors. Their love and support motivated me to continue my education and achieve higher academic goals. Without their moral support and sincere prayers, I would have been unable to accomplish this task.

Next, I am deeply grateful to my thesis advisor Dr. Sadiq M. Sait for his valuable guidance throughout my thesis work. At the same time, gratitude is due to my thesis Co-Chairman Dr. Aiman H. El-Maleh and committee member Dr. Khan M. Farrukh. I express my thanks to all of them for their valuable time and support.

I acknowledge the academic and computing facilities provided by the Computer Engineering Department of King Fahd University of Petroleum & Minerals (KFUPM).

I also appreciate the friendly support from all my colleagues at KFUPM. In particular, I want to thank (in alphabetical order) Ahmer, Ala', Atif, Barnawi, Faheemuddin, Junaid, Minhas, Osama, saad, Salman, Sanaullah, Shazli, Wasiq, Yassir, and Yusuf.

Contents

List of Tables

List of Figures

THESIS ABSTRACT

Name:	Raslan Hashim Al-Abaji
Title:	Evolutionary Techniques for Multi-objective VLSI Netlist Partitioning
Major Field:	COMPUTER ENGINEERING
Date of Degree:	May, 2001

The problem of partitioning appears in several areas ranging from VLSI, parallel programming, to molecular biology. The interest in achieving better partitioning of any system is growing rapidly especially in VLSI, and has been a hot issue in recent years. In VLSI circuit partitioning, the problem of obtaining a minimum cut has been of prime importance and most literature available is for this single objective optimization. However, with current technology trends partitioning has become a multi-objective problem (MOP), where power and delay, in addition to minimum cut, need to be optimized. In this thesis, the multi-objective optimization problem at the partitioning phase in VLSI physical design step is addressed. This problem involves multiple, possibly conflicting objectives; hence fuzzy rules have been incorporated in designing the overall cost function. Iterative algorithms, namely Genetic Algorithm (GA), Tabu Search (TS), and Simulated Evolution (SimE) are tailored for finding good quality solutions for the above mentioned MOP problem. Another deterministic algorithm PowerFm which is an extension to Fidducia Mattheyeses (FM) heuristic is proposed and results compared and analyzed.

MASTER OF SCIENCE DEGREE

King Fahd University of Petroleum & Minerals, Dhahran, Saudi Arabia.

July, 2002

Chapter 1

Introduction

1.1 Background

The driving force behind the spectacular advancement of the integrated circuit technology in the past thirty years has been the *exponential scaling* of the feature size, i.e., the minimum dimension of a transistor [4]. It is expected that such exponential scaling will continue for at least another 5-7 years as projected in the 1997 National Technology Roadmap for Semiconductors [1] as shown in Table 1.1. This will lead to a half a billion transistors integrated on a single chip with an operating frequency of 2-3 Ghz in the 0.07 μm technology by year 2009. However the challenges to sustain such an exponential growth to achieve gigascale integration have shifted in a large degree, from the process and manufacturing technologies to the design technology. A great deal of design innovation, in terms of both significant extension of

Technology (μm)	0.25	0.18	0.15	0.13	0.10	0.07
Year	1997	1999	2001	2003	2006	2009
# transistors	11M	21M	40M	76M	200M	520M
Across chip clock (Mhz)	750	1200	1400	1600	2000	2500
Area (mm^2)	300	340	385	430	520	620
Wiring Levels	6	6-7	7	7	7-8	8-9

Table 1.1: Overall technology roadmap [1].

the existing design capability and the development of new design paradigms and methodology, is needed to achieve the projected targets in [1]. This thesis discusses the challenges and proposes some solutions for future IC designs, especially in the area of multiobjective circuit partitioning.

1.2 Motivation and Contribution

Due to substantial advances in VLSI process technology, designers are facing rapid increase in system complexity. One natural approach to designing highly complex systems is to apply the divide-and-conquer methodology to decompose the large system into a set of smaller subsystems recursively and carry out the design hierarchically. Nowadays, it is common to decompose a complex IC into a number of functional blocks, each of them designed by one or a team of engineers with manageable complexity, and then go through a "full-chip assembly" - phase to interconnect these blocks together.

Circuit partitioning divides a given circuit into a collection of smaller subcircuits to achieve certain objectives while satisfying some constraints. In order to keep up

with the rapid increase of system complexity due to substantial advances of VLSI process technology, partitioning is performed at various level of design hierarchy until subproblems become more manageable. During the last decade, the main VLSI circuit design objectives were the minimization of interconnect wire length and the improvement of timing performance. A lot of work targeting either one or both of the above objectives is reported in the literature [2, 5].

1.2.1 Optimizing Power Motivation

In recent years, the focus of portable devices has shifted from low throughput devices (e.g., watches, calculators) to high performance devices like notebook computers, cellular phones, etc. Minimizing power is the primary concern for these battery-powered products as for such products longer battery life translates to extended use and better marketability. Exploring the tradeoffs between power, performance, and other objectives during synthesis and physical design is thus demanding more attention.

Some power optimization techniques that have been proposed in the literature are reviewed in Chapter 2. The optimization for power consumption can be performed at various levels of VLSI design including behavioral level, architectural level, logic level, and physical level. The optimization at each level can be performed subject to degree of actual realization of the circuit. For example, it is not possible to optimize power consumption due to the interconnect capacitance at logic level, because a wire

length estimation cannot be accurately done at this stage. This fact enhances the need to perform the interconnect capacitance optimization at physical level. Another compelling reason for achieving power consumption is the increasing density of VLSI circuits. With the rapid advancement in technology, VLSI circuits are decreasing in size resulting in higher transistor density on a chip. The present technology allows integration of tens of millions of transistors on a single chip and the still advancing technology is allowing further high integration. The excessive power consumption of the circuit results in heating and thus becoming a hindrance towards high integration and hence the feasible packaging of circuits [6, 7]. Also the circuits are operating at much higher clock frequency than before. Therefore, the power dissipation which is a function of clock frequency, is getting significantly prominent. This phenomenon presents an obstacle in further increase of clock frequency. Due to these reasons, there is an emerging need for minimizing the power requirement of VLSI circuits.

1.2.2 Motivation for Optimizing Delay

In current submicron designs, wire delays tend to dominate over gate delays [8], the differences between on-chip and off-chip signal delays and the increasing limited-pin nature of large chips make it desirable to minimize the number of signals traveling off a given chip. Larger die sizes imply that long on-chip global routes between function blocks will more noticeably affect system performance. Other considerations (e.g., design for testability, low power design, etc.,) also require partitioning algorithms

to identify interconnect structure, albeit at more of a functional or communication-based level.

The conventional objective of partitioning is to minimize the interconnect among the partitioned circuits. On the other hand, under the new design paradigms, partitioning is seen as the crucial step that defines local and global interconnects [9]. To meet the performance requirement of today's complex design, performance driven partitioning must consider the amount of interconnect induced by partitioning (measured by its cutset) as well as its impact on performance (measured by its delay). Many proposed cutset partitioning techniques do not consider delay, while many proposed delay driven partitioning techniques do not consider power. Furthermore, many proposed power driven partitioning techniques do not consider delay. As a result, there is a strong need for a performance driven partitioning technique that considers cutset, delay, and power and provide a choice to emphasize on one objective more than the other relatively.

1.2.3 Objective of Thesis

The main goal of this thesis is to design a class of iterative algorithms for VLSI multiobjective partitioning such that circuit delay, power dissipation and interconnect (cutset) are minimized under the balance constraint. The main search algorithms used are Genetic Algorithms (GA), Tabu Search (TS), and Simulated Evolution (SimE). These algorithms are used to find a solution that minimizes all these costs

while satisfying balance constraint. Iterative algorithms are used due to their capability to find optimal or near optimal solutions, and their ability to escape local optima.

Due to the multiobjective nature of the problem, a single aggregating function that represents the properties of all the cost functions is needed. Some techniques to design such an aggregating function are given in [10]. Among these techniques, weighted sum scheme is the most widely used. However, balancing different objectives by weight functions is difficult, or at best controversial. Fuzzy logic is a convenient vehicle for solving this problem. It allows mapping values of different criteria into linguistic values, which characterize the level of satisfaction of the designer with the numerical value of the objectives. All these numerical values operate over values from the interval [0,1] defined by membership functions for each objective.

In this thesis we also propose a new power driven algorithm PowerFM, which is an extension to the well known Fidducia Mattheyeses (FM) heuristic for circuit partitioning.

1.3 VLSI Netlist Partitioning

In this section, VLSI netlist partitioning problem is briefly described. VLSI design is a complex process and is therefore broken down into a number of intermediate

steps [2]. The design cycle starts from an abstract idea, and then each intermediate step continues evolving the design and the process ends with the fabrication of a new chip as illustrated in Fig 1.1.

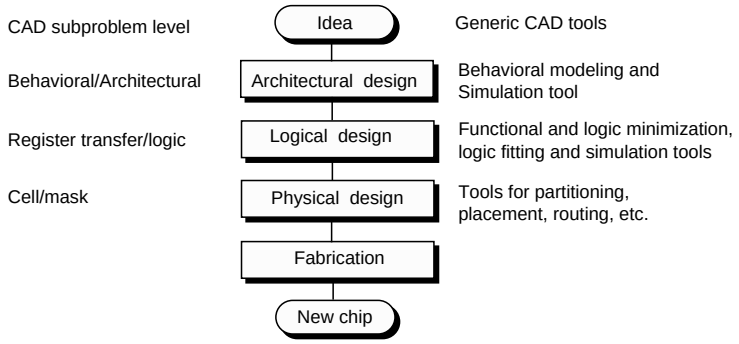

Figure 1.1: Various steps in VLSI design process.

Partitioning is a phase in physical design responsible for the arrangement of cells into a set of partitions while optimizing the costs of certain objectives such as the external wiring of each set (cutset) or power consumption. Generally it is a k-*way* partitioning problem, we refer to $k = 2$ as a bipartitioning in our case. The circuit may be represented as a graph or hypergraph (defined in Chapter 3), whose vertices represent circuit elements, and edges represent the interconnects [2].

Moreover partitioning is considered to be an NP-complete problem [11], which means it is unlikely that a polynomial-time algorithm exists for solving the problem. Therefore one must use heuristic techniques for generating near optimal solutions. One such heuristic is the widely used Kernighan-Lin (KL) algorithm [12]. Another algorithm that is a variation of the Kernighan-lin heuristic is the Fiduccia-Mattheyeses (FM) approach [13]. Details of FM are discussed in the following

section.

1.4 FM Partitioning Heuristic

The FM heuristic is a modification of the Kernighan-Lin group migration method for circuit partitioning. The original Kernighan-Lin heuristic is an improvement procedure, where the current partition is improved by swapping pairs of nodes belonging to the two subsets of the current partition. Selection of the best pair to swap requires searching the space of $O(n^2)$ items (where n is the number of nodes); this search might need to be performed in $O(n)$ times.

Instead, Fiduccia and Mattheyses suggested the following modifications:

1. Only one node to be moved at a time.
2. The FM aims at reducing the cost of nets cut by the partition.
3. The modification can cause imbalance arising from all cells wanting to migrate to a single partitiong. Therefore FM heuristic is designed to handle imbalance, and it produces partitions balanced with respect to size.
4. The algorithm maintains a sorted list of candidate interior nodes (Bucket arrays) for moving to the other subcircuit, and updates it after each move.

The Bucket arrays mentioned is an elegant data structure developed by Fiduccia and Mattheyses to maintain the sorted candidate node lists and to identify the best candidate node in constant time. A Bucket array, shown in Fig 1.2, basically is

a hash table of linked lists, where the index is the gain of the cell (or vertex). A cell is pushed into the linked list according to its gain; all cells in one list have the same gain. Two Bucket arrays, one for each partition, are maintained. Each array is maintained by moving a vertex to the appropriate bucket whenever its gain changes due to movement of one of its neighbors. This move can be done in constant time. To ensure that the algorithm terminates, each node is assigned to one of two sets: those not yet moved belong to the $freeset$, and the others are assigned to the $lockedset$. Only the nodes in the free set can be moved from one subset to another, and a moved node becomes locked. Initially, all nodes belong to the free set; therefore, each node can be moved exactly once. In one single pass, all nodes in the free set are moved once between the partition's subsets. A single pass of the FM heuristic has a linear time complexity with respect to the number of pins in a circuit.

The best candidate node is defined according to the highest cut-gain associated with moving a node from one subcircuit to another. The cut-gain is measured using the *net-cut model* [13]. A net is called a *cut net* if it belongs to the current cutset; otherwise, the net is referred to as a *nocut* net. A net is called critical if it is a *cut net* that, as a result of moving a single node, can become a *nocut* net, or vice versa.

The basic concept of min-cut gain calculation provided with the net-cut model can be explained as follows. Let node i_0 be connected to n cut and critical nets and to m nocut critical nets. The gain associated with the reassignment of a node i_0 is

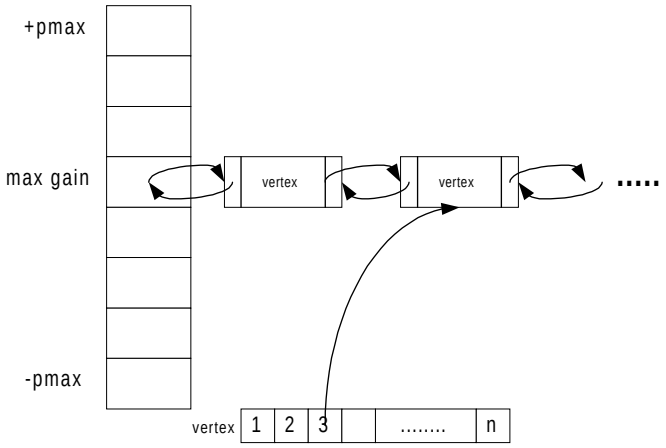

Figure 1.2: Bucket array structure (pmax is the max gain).

defined as the difference:

$$G_{i_0} \triangleq n - m \tag{1.1}$$

The gain can be either positive if moving the node to the other subset of the current partition would reduce the cut value, negative if the cut value would increase, or zero if the cut value would not change after the node move. Fig. 1.3 shows a sample calculation of gains associated with nodes, for example node A movement gain is +2 since 2 edges will be removed from the cutset.

In the FM algorithm, all nodes in the free set are arranged into a bucket array data structure, in which each bucket contains nodes with the same gain. All buckets are sorted according to the decreasing values of gains. For each move, the node with

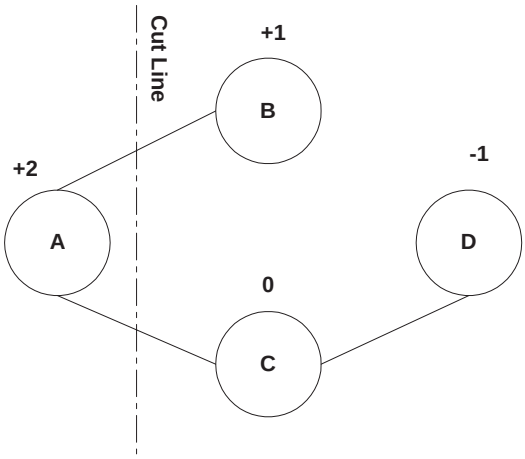

Figure 1.3: Gain calculation associated with a cell move.

the highest gain is considered as the primary candidate. The candidate node must satisfy the balance criterion, used to control the size of subcircuits, e.g., a balance criterion may require that moves causing the size of the subset with the candidate node to fall below a certain minimum must be rejected. If the candidate node does not meet the balance criterion, the node with the highest gain is selected from the free nodes subset and moved. The moved node is locked and eliminated from the bucket array. The move is completed by modifying the gains of all nodes connected to the critical nets. The complexity of the algorithm is $O(P)$, where P is the total number of pins in the hypergraph, i.e., the sum of nets each vertex belongs to. A general description of the heuristic is given in Fig. 1.4.

ALGORITHM *FM*
Begin

Step1: Compute gains of cells;
Step2: $i = 1$;
Select 'base cell' and call it c_i;
If no base cell **Then Exit EndIf;**
A base cell is the one which
(i) has maximum gain;
(ii) Satisfies balance criterion;
IF *tie* **Then** use size criterion or
Internal connections;
EndIf;
Step3: Lock cell c_i;
update gains of cells of those affected critical nets;
Step4: **IF** free_cells $\neq \phi$
Then $i = i + 1$;
select next base cell c_i;
IF $c_i \neq \phi$ then **Goto** step 3;
Step5: Select best sequence of moves c_1,c_2,...,c_k $(1 \leq k \leq i)$
such that $G = \sum_{j=1}^{k} g_i$ is maximum;(g_i is the gain for cell c_i)
IF tie **Then** choose subset that achieve a superior balance;
IF $G \leq 0$ **Then Exit**;
Step6: Make all k moves permanent;
Free all cells;
Goto Step 1
End.

Figure 1.4: Fiduccia-Mattheyeses bipartitioning algorithm [2].

1.5 Organization of Thesis

The rest of this thesis is organized as follows. Chapter 2 presents a survey of techniques reported in literature for VLSI partitioning. Also, a brief review of performance driven partitioning techniques is given.

In Chapter 3, the multiobjective partitioning problem is formulated. The cost functions for objectives, i.e., cutset, timing performance, power consumption as well as for balance constraint are designed. An overview of fuzzy logic is also presented. Finally, the overall cost function used in multiobjective optimization is designed using fuzzy operators. An overview of iterative algorithms used in this thesis is also presented.

Chapter 4 discusses the implementation details of the proposed GA, TS and SimE for multiobjective VLSI partitioning. The settings of various GA and TS parameters are discussed as well as the selection and allocation schemes in SimE. Moreover, the PowerFM is also discussed in detail. Finally, experimental results for application of the proposed techniques to ISCAS-85/89 benchmarks are discussed and compared in Chapter 5.

Chapter 2

Literature Review

2.1 Introduction

The essence of netlist partitioning is to divide a system into clusters such that the number of inter-cluster connections is minimized. The partitioning task is ubiquitous to many subfields of VLSI CAD. Most top-down hierarchical (i.e., divide and conquer) approach in system design must rely on some underlying partitioning technique. There are several reasons why partitioning has recently emerged as a critical step in many phases of VLSI system synthesis, and why the past several years have seen so much research activities on this subject [6, 14].

Partitioning heuristics are used to address the increasing complexity of VLSI design, systems with several million transistors are now common, presenting instance complexities that are unmanageable for existing logic level and physical level

design tools. Partitioning divides a system into smaller, more manageable components; the number of signals which pass between the components corresponds to the interactions between the design sub-problems. In a top-down hierarchical design methodology, decisions made early in the system synthesis process (e.g., at the system and chip levels) will constrain succeeding decisions. Thus, the feasibility - not to mention the quality - of automatic placement, global routing, and detailed routing will somewhat depend on the quality of the partitioning solution.

A bottom-up clustering approach may also be applied to reduce design complexity, typically in cell-level or gate-level layout. The current emphasis on a quick turn-around design cycle reinforces the need for reliable and effective algorithms. Partitioning heuristics also have a great impact on system performance as designs become dominated by interconnects.

Finally, partitioning heuristics affect the layout area; wires between clusters at higher levels of the hierarchy will tend to be longer than wires between clusters at lower levels, and total wirelength is directly proportional to layout area due to minimum wire spacing design rules. The traditional minimum-cut objective is natural for this application, if the layout area is divided into a dense uniform grid. Total wirelength can be expressed in "grid" units or equivalently as the sum over all gridlines of the number of wires crossing each gridline. This view can also improve auto-routability since it suggests reducing the wire congestion in any given layout region. All of these considerations motivate the development of netlist par-

titioning algorithms that identify interconnection and communication structure in a given system design. In the following section, we discuss different approaches to partitioning which considered only cutset as an objective. In Section 2.3 we discuss approaches which explore performance (delay and power).

2.2 Approaches to Partitioning

As the partitioning problem is NP-complete [15, 11], an exact (globally optimal) solution cannot be found in a feasible amount of time. Therefore, heuristics must be used to reach a good solution within reasonable time limits. Major research directions in netlist partitioning can be categorized into four types of approaches:

- Move-based Approaches [16, 17, 18, 19].
- Geometric Representation Approaches [20, 21, 22].
- Combinatorial Approaches [23].
- Cluster-based Approaches [24].

2.2.1 Move-based Approaches

This category explores the solution space by moving from one solution to another. Greedy and iterative exchange [12] approaches are most common. These always try to make the best move, but can easily be trapped in local minima. To avoid this behavior, many other strategies have been proposed including Stochastic Hill-Climbing (Simulated Annealing), Evolutionary Algorithms, and the Multi-start

strategy [25]. A partitioning approach is move-based if it iteratively constructs a new candidate solution based on two considerations:

- A neighborhood structure is defined over the set of feasible solutions.
- the previous history of optimization is maintained.

The first consideration requires the notion of a local perturbation of the current solution; this is the heart of the move-based paradigm. The type of perturbation used determines the topology over the solution space, known as the *neighborhood structure*. For the objective function to be *smooth* over the neighborhood structure, the perturbation (also known as a neighborhood operator) should be *small* and *local*. Typical neighborhood operators for partitioning include swapping a pair of modules or shifting a single module across a cluster boundary. For example, two partitioning solutions are neighbors under the pair-swap neighborhood structure if one solution can be derived from the other by swapping two modules between clusters. In general, the solution space is explored by repeatedly moving from the current solution to a neighboring solution. With respect to previous history, some approaches are *memoryless*, e.g., a simple greedy method might rely only on the current solution to generate the next solution. On the other hand, methods such as Kernighan-Lin [12] or Fiduccia-Mattheyses [13] implicitly remember the entire history of the *pass*. Hybrid genetic-local search or Tabu Search approaches must also remember the lists of previously seen solutions. Move-based approaches dominate in both the literature

and industry practices for several reasons. First, they are generally very intuitive, the logical way of improving a given solution is to repeatedly make it better via small changes, such as moving individual modules. Second, iterative algorithms are simple to describe and implement. For this reason, the bipartitioning method of Fiduccia-Mattheyses [13] and the Multi-way partitioning method of Sanchis [26] are standards against which nearly all other heuristics are measured. Third, the move-based approach encompasses more sophisticated strategies for exploring the solution space e.g., Simulated Annealing, Tabu Search, and Genetic Algorithms which yield performance improvements over greedy iterative methods while retaining the intuitiveness associated with local search. Finally, the move-based approach is independent of the nature of the objective function that is used to measure the solution quality. While other approaches might require the objective to be of a particular form, or a relatively simple function of solution parameters, the move-based approach can flexibly incorporate arbitrary constraints (e.g., on critical path delays or I/O utilization). Thus, the move-based approach has been applied successfully to virtually every known partitioning formulation.

The main algorithms, included in this approach are:

- Fiduccia-Mattheyses Algorithm [13].
- Kernighan-Lin Algorithm [24] .
- Sanchis' Multi-Way Partitioning Algorithm [26].
- Simulated Annealing Algorithm [27] .

- Tabu Search [28].
- Genetic Algorithms [29].

2.2.2 Geometric Representations Approaches

A geometric representation of the circuit netlist can provide a useful basis for a partitioning heuristic. These approaches discuss finding a geometric representation of a graph or hypergraph and applying *geometric* algorithms to find a partitioning solution. This means that the circuit netlist is embedded in some type of *geometry*, e.g, a 1-dimensional linear ordering or a multi-dimensional vector space; the embeddings are commonly constructed using Spectral methods [21]. Spectral methods are of primary importance in constructing geometric representations.

2.2.3 Combinatorial Formulations

An approach is classified under this category if the partitioning problem can be transformed into some other "classic" type of optimization problem (e.g., maximum flow, mathematical programming, graph labeling etc.). These approaches are promising since complex formulations that include timing, module pre-assignment, replication, and other hard constraints can often readily be expressed in terms of a mathematical program or flow networks. In addition, the constantly changing user requirements for solution quality and runtime, and the improved computing platforms, have made such approaches more practical.

It is possible, that the next frontier of optimization strategies for CAD applications will involve large-scale mathematical programming instances, including mixed integer-linear programs that require branch-and-bound search. Following are some of the methods employed under this category:

- Min-Delay Clustering by Graph Labeling, first considered by Lawler *et al.*[30], assumes that the module and intra-cluster delays (i.e., delays between modules in the same cluster) are negligible compared to inter-cluster delay that results from placing clusters onto different chips.

- Mathematical Programming optimizes an objective function subject to inequality constraints on the variables (an equality constraint can be captured by two inequality constraints). A linear program (LP) requires every equation to be linear in terms of each variable. An LP can be solved in an average case polynomial time using the simplex method. An integer linear program (ILP) is an LP with the additional constraint that the variables must take on integer values; solving general ILP instances is NP-Hard. A quadratic program (QP) [23] is an LP with an objective that is quadratic in the variables, and a quadratic boolean program (QBP) additionally restricts the variables to 0-1 values.

- Fuzzy partitioning or the Fuzzy k-means (FKM) [31] algorithm is a well-known optimization technique for clustering problems that arise in such fields as

geological shape analysis, medical diagnosis, etc. The problem formulation generally involves clustering data points in multi-dimensional space. A fuzzy partitioning [32] can partially assign a module to several clusters. FKM begins with an initial fuzzy partitioning X, then iteratively modifies X to optimize the objective function.

2.2.4 Clustering Approaches

A clustering solution is typically used to induce a smaller and more tractable problem instance. Many clustering algorithms utilize a bottom-up approach where each module initially belongs to its own cluster. Clusters are gradually merged or grown into larger clusters until the desired decomposition is found. Bottom-up approaches are *agglomerative*, if new clusters are formed one at a time and *hierarchical*, if several new clusters may be formed simultaneously. The agglomerative approach [33] begins with *n-way* clustering (where each module is a cluster) and iteratively constructs the new clusters by choosing a pair of clusters, and merging them into a new cluster. The criterion for choosing the two clusters is what distinguishes among agglomerative variants, e.g., [33] merges the two clusters that minimize the diameter of the newly formed cluster. This approach is applied to hypergraphs by picking a random net (perhaps with size-dependent probability) and contracting two random incident clusters (or all incident clusters). An alternative greedy approach would be to simply merge the two clusters with high connectivity.

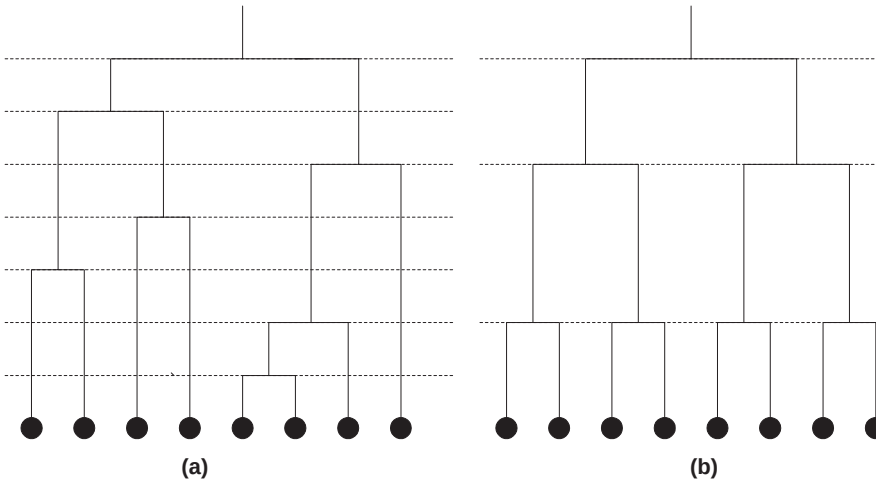

Figure 2.1: An 8-*module* example (a) an agglomerative and (b) a hierarchical construction

Generally, agglomerative methods will not be very efficient: finding the best pair of clusters to merge require $O(k^2)$ time, unless a list of cluster merging costs is stored and updated (which will likely require $O(n^2)$ space). An alternative strategy is to find many good clusters to merge, then perform all merges simultaneously; this is called hierarchical strategy. The difference between agglomerative and hierarchical strategies is illustrated for the 8-*module* example in Fig. 2.1. In (a), the diagram reveals the order in which clusters are merged; each dotted horizontal line is a level in the hierarchy, and an agglomerative algorithm will have n-1 levels. Fig. 2.1(b) shows a hierarchical algorithm that simultaneously merges as many cluster pairs as possible, yielding a hierarchy with $\lceil log\ n \rceil$ levels.

Other intuitive approaches involve random walks, iterative peeling of clusters, vertex orderings, and simulated annealing. Another set of approaches are specific

to (acyclic) combinational Boolean networks [34, 35]. However, move-based approaches, and iterative improvement in particular, are the most common partitioning algorithms in current CAD tools. Clustering techniques are motivated by the fact that a common weakness of move-based approaches is that the solution quality is not *stable*, i.e., unpredictable. It is highly dependent on the starting solution and the choices taken during the optimization process. Hagen *et al.* [16] used a random multi-start approach to FM, where the algorithm is executed many times from random starting points and returning the best solution found. However, it may need hundreds of runs to achieve stable performance.

The hierarchical clustering algorithm groups a set of objects according to some measure of closeness. Two closest objects are clustered first and considered to be a single object for future clustering. Clustering continues by grouping two individual objects, or an object or cluster with another cluster on each iteration. The process stops when a single cluster is generated and a hierarchical cluster tree is formed. A cut-line through the tree indicates a set of segments in a partition. Clustering can be integrated into other move-based algorithms. The simplest way to incorporate a clustering solution into a bi-partitioning heuristic is via the two-phase approach, i.e., to run FM on the contracted netlist, and then use the result as the starting solution of a second run on the flattened netlist [36]. However, more sophisticated techniques may be preferable.

2.3 Performance-Driven Partitioning in Physical Level

This section reviews some recent approaches for performance driven partitioning (power, delay) in CMOS VLSI circuits. Different techniques are applicable and have been reported at different steps of the VLSI design process [6].

In standard CMOS VLSI circuits, switching activity of circuit nodes is responsible for most of the power dissipation. It is reported in [37] that this switching activity contributes up to 90% of the total power dissipation in the circuit. Therefore, most of the reported techniques focus on this aspect [38]. Quite a reasonable number of techniques aiming at low power objective are proposed for all phases in physical design including partitioning of circuit, floorplanning, placement and routing [2].

For the partitioning phase, two low-power oriented techniques based on Simulated Annealing (SA) algorithm have recently been presented in [39]. One of the algorithms uses the Shannon expansion-based scheme and the other uses the Kernel-based scheme. These algorithms partition the circuit into a number of sub-circuits and a single sub-circuit needs to be active at a particular time. In this way, the unnecessary signal transitions are prevented. Circuit partitioning is performed by using an adaptive SA algorithm. The cost function is modeled for low-power consumption under given area constraint. A partitioning solution is obtained by recursive

bi-partitioning of the circuit and the solution space is represented as a binary tree. The stopping criteria used is non-improvement in the solution for a constant number of moves. The performance of the algorithm is evaluated by its application to MCNC benchmark circuits and its comparison with the results of Synopsis design analyzer show an 8.7% power reduction over the latter without allowing any increase in the layout area.

An optimal delay partitioning algorithm targeting low power is proposed in [14] which provides a formal mechanism to implicitly enumerate the alternate partitions and selects a partition that has the same delay but less power dissipation. One disadvantage of this algorithm is that the runtime is one to two orders of magnitude higher than that of Lawler's clustering algorithm [30]. Another disadvantage of this enumeration technique is that as the size of the circuits grows, the algorithm runtime will increase sharply hence this technique is not suitable for industries seeking a faster time to design and market the chips.

A circuit partitioning algorithm under path delay constraint is proposed in [40]. The proposed algorithm consists of the clustering and iterative improvement phases. In the first phase, the problem size is reduced using a new clustering algorithm to obtain a partition in a short computation time. The first phase consist of the following steps:

1. Clustering considering timing constraints

2. Clustering considering timing and area constraints

In step 1, the path which violates the timing constraint (i.e., if the path is cut) is clustered. which means assigning all nodes in the path to the same cluster. In Step 2, clustering is performed again considering the timing and area constraints so as to obtain a better partition in reasonable computation time. This is done by clustering nodes based on a cost function in which the timing and area constraints are considered. Phase 2 is an iterative improvement phase with an extended FM method in which a term to handle the timing constraints was introduced into the gain of the original FM. Phase 2 consists of the following three steps:

1. Initial partitioning

2. Iterative improvement with the extended FM method

3. Removal of timing violations

A detailed description of the timing model that was used is given in section 3.3.2.

Chapter 3

Problem Formulation and Solution Methodology

3.1 Introduction

A typical VLSI design process is divided into several levels of design abstraction as shown in Fig. 1.1. More design complexity is added as we move towards lower abstraction levels. Each level of abstraction has its own importance in the design process i.e., accurate delay calculations are hard or impossible when calculated only in terms of logic gates. Furthermore, other design tasks, such as logic optimization are too cumbersome to be done at physical level [2, 41]. Due to this, efforts have been made to reduce power, delay and area at nearly every level of the design abstraction. In this thesis, the problem of reducing overall power dissipation of the

circuit with improved timing performance, reduced cutset and within acceptable balance at physical design level, or more precisely at partitioning stage is addressed. For this purpose, it is assumed that the circuits are available along with the critical paths and the switching probabilities of the nets. In this chapter, the partitioning problem will be formulated with power, circuit delay and cutset as objectives to be minimized and balance as a constraint.

3.2 Partitioning Formulation and Modeling

Given a set of modules $V = \{v_1,\ v_2, \cdots,\ v_n\}$, the purpose of partitioning is to assign the modules to a specified number of clusters k satisfying prescribed properties. **Definition:** A *k-way* partitioning $P^k = \{C_1,\ C_2, \cdots,\ C_k\}$ consists of k clusters (subset of V), $C_1,\ C_2, \cdots,\ C_k$ such that $C_1 \cup C_2\ \cdots\ \cup\ C_k = V$ and $C_i \cap C_j = \emptyset$ for $i = \{1, \cdots, k\}$ and $j = \{1, \cdots, k\}$ and $j \neq i$. If $k = 2$, we refer to P^2 as bipartitioning.

The objective to be optimized is denoted by $F(P^k)$ i.e., it is a function of the partitioning solution. We generally make the traditional assumption that the clusters are mutually disjoint; note, however, that replication formulations permit a module to be a member of more than one cluster [42]. In general, a circuit can have multi-pin connections (nets) apart from two-pin and therefore it is better to describe it as a hypergraph especially when the net cut is to be minimized. A hypergraph is

defined as $H(V, E)$ where V is a set of nodes and E is a set of hyperedges. Node $v_i \in V$ corresponds to an element (e.g., a gate) in the circuit, and hyperedge $e_i \in E$ corresponds to a net in the circuit. A node corresponding to a register in the circuit is called a register node and denoted by $r_i \in R$, where $R \subset V$. Hyperedge e_i consists of the signal source node $S(e_i)$ and a set of destination nodes $D(e_i)$ and $e_i = (S(e_i), \{D(e_i)\})$. The signal source node $S(e_i)$ of the net e_i corresponds to the output of a gate and the set of destination nodes $D(e_i)$ corresponds to the inputs of the gates. Given a hypergraph $H(V, E)$ with $E = \{e_1,\ e_2,\ \cdots,\ e_m\}$ being the set of signal nets, each net is a subset of V containing the modules that net connects. It is assumed that for each hyperedge $e \in E$, $|e| \geq 2$ (it connects at least two nodes). The equivalence between netlists and hypergraphs is exact if each net has at most one pin on any module. The modules in e may also be called the pins of e. Two-way partitioning of the set of nodes V determines two subsets V_A and V_B such that $V_A \cup V_B = V$ and $V_A \cap V_B = \oslash$.

3.3 Partitioning Objectives and Constraints

Before listing the objectives and constraints, we state some assumptions:

1. The logic level design (or netlist) of the circuit is available.
2. The set of critical paths and switching probabilities of gates is available.
3. The circuit is represented in form of a hypergraph.

Our task is to divide V into 2 subsets (blocks) V_0 and V_1 in such a way that the objectives are optimized, subject to some constraints. We aim at minimizing the following objectives:

- Cutsize.
- Delay.
- Power dissipation.

3.3.1 Cutsize

The set of hyperedges, cut by a cluster C, is given by $E(C) = \{e \in E \text{ s.t. } 0 < |e \cap C| < |e|\}$ i.e., $e \in E(C)$ if at least one, but not all, of the pins of e are in C. The set of nets cut by a partitioning solution p^k can be expressed as $E(p^k) = \bigcup_{i=1}^{k} E(c_i)$ or equivalently $E(p^k) = \{e \in E \text{ s.t. } \exists u, v \in e, h \neq l,\ u \in C_h \wedge v \in C_l\}$. We say that $|E(p^k)|$ is the cutsize of p^k.

The cost function can also be written as follows :

$$f = \sum_{e \in \psi} w(e) \tag{3.1}$$

where $\psi \subset E$ denotes the set of off-chip wires. The weight $w(e)$ on the edge e represents the cost of wiring the corresponding connection as an external wire (outside of its cluster). If all weights equal one, the cost function becomes simpler:

$$f = |\psi| \tag{3.2}$$

where $|\psi|$ denotes the cardinality of the set ψ.

3.3.2 Delay

In order to deal with a signal path, a hypergraph is decomposed into directed edges $e_k = (S(e_k), w)$ for $e_k \in E$ and $w \in D(e_k)$. Let the graph which consists of a set of nodes V and a set of decomposed directed edges E be the directed graph $G' = (V, E)$. A signal path is represented by an alternating sequence of nodes and directed edges $v_1, e_1, v_2, e_2, ..., v_{k-1}, e_{k-1}, v_k$, where $e_l = (v_l, v_{l+1})(1 \leq l \leq k-1)$ and $\{v_i \neq v_j,\ i \geq 1\ ,\ j \leq k\ ,\ i \neq j\}$. The path from node v_i to node v_j is denoted by p_{ij}. Nodes which are included in the path p_{ij} are defined as $V(p_{ij})$. The number of nets cut which are included in the path p_{ij} is denoted by $ncut(p_{ij})$. In the general delay model where gate delay $d(v)$ and constant inter-chip wire delay $d_c \gg d(v)$ are considered, the d_c is actually due to the off-chip capacitance denoted by C_{off}. Let the delay of node $v_i \in V$ be $d(v_i)$ and the delay of the cut net $e_k \in E$ be d_c. Given a partition Φ:$(V_A; V_B)$, the path delay $d(p_{ij})$ between nodes v_i and v_j is the sum of the node delays $d(v_i) \in V(p_{ij})$ and the delay of the cut nets, that is :

$$d(p_{ij}) = \left(\sum_{v_i \in V(p_{ij})} d(v_i) \right) + d_c \times ncut(p_{ij}) \tag{3.3}$$

The delay calculation in this work adopt the linear delay model. It is assumed

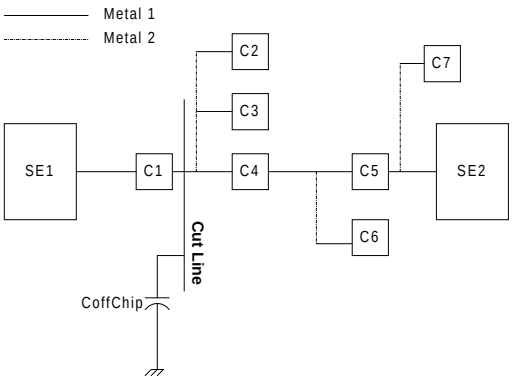

Figure 3.1: Path $SE1$ to $SE2$.

that for a given net, when a signal starts from its source, the sinks will be equally charged at the same time. So, the time needed for the signal to reach any of the sinks is equal. An example of a delay path π is shown in Fig. 3.1, where the delay path $\pi = SE_1 \rightarrow C_1 \rightarrow C_4 \rightarrow C_5 \rightarrow SE_2$.

To carry out the calculation of the delay the following parameters are considered:

CD_i = Overall cell delay for cell i. (in PicoSec)

BD_i = Base delay of cell i.

LF_i = Load factor of cell i (in Ohms).

$CINP_i$ = Total load capacitance on the input pins of cell i (in femtoFarads).

Overall cell delay, CD_i for cell is calculated as follows:

$$CD_i = BD_i + LF_i \cdot CINP_i \cdot 10^{-3} \tag{3.4}$$

Then, the total delay of the path shown in Fig. 3.1 can be calculated as follows:

$$Delay(\pi) = CD_{SE_1} + CD_{C_1} + CD_{C_4} + CD_{C_5} + CD_{SE_2} \tag{3.5}$$

Notice that the delay for the cell C_1 which has a cutline across its output is calculated as follows :

$$CD_{C_1} = BD_{C_1} + LF_{C_1} \cdot (C_{off} + CINP_{C2} + CINP_{C3} + CINP_{C4}) \tag{3.6}$$

3.3.3 Power

The average dynamic power consumed by a CMOS logic gate in a synchronous circuit is given by:

$$P_i^{average} = 0.5 \frac{V_{dd}^2}{T_{cycle}} C_i^{load} N_i \tag{3.7}$$

where C_i^{load} is the load capacitance, V_{dd} is the supply voltage, T_{cycle} is the global clock period, and N_i is the number of gate output transitions per clock cycle. N_i is calculated using the symbolic simulation technique of [43] under a zero delay model. C_i^{load} in Eq. 3.7 consists of two components: C_i^{basic} which accounts for the load capacitances driven by a gate before circuit partitioning, and the extra load C_i^{extra} which accounts for the additional load capacitance due to the external connections of the net after circuit partitioning. Then, the total power dissipation of any circuit

ζ is:

$$P_\zeta = \beta \frac{V_{dd}^2}{T_{cycle}} \sum_{i \in \zeta} (C_i^{basic} + C_i^{extra}) N_i \tag{3.8}$$

where β is a constant that depends on technology. When a circuit partitioning corresponds to a physical partitioning, C_i^{extra} of a gate that is driving an external net is much larger than C_i^{basic}. The power model given in Eq. 3.8 can be further simplified if it is assumed that the power dissipation contribution due to variations of C_i^{basic} under different partitioning solutions is negligible. Furthermore, considering that the fixed overhead capacitance for an external net is dominant within C_i^{extra}, it can be assumed that C_i^{extra} is identical for each net. This leads to the following objective function [14].

$$O_\zeta = \sum_{i \in \zeta_v} N_i \tag{3.9}$$

where ζ_v corresponds to the set of visible gates i.e., the set of gates that drive a load external to the partition.

3.3.4 Area or Balance Constraint

The area/balance constraint is defined as follows: let $A(v_i)$, denote the area of $v_i \in V$ which corresponds to the area of a gate i and $A(S) = \sum_{v_i \in s} A(v_i)$, denote the area of a subset $S \subset V$. The area constraint is that given a balance factor α $(0.5 < \alpha < 1.0)$, we have to partition V into V_A and V_B so that both $A(V_A)$ and $A(V_B)$ should be smaller than $\alpha \times A(V)$, that is $A(V_A)$, $A(V_B) \leq \alpha \times A(V)$. If we

assume that the area of all cells is identical then the problem reduces to balancing the two partitions in terms of the number of cells; the balance constraint is given as follows:

$$\frac{|\beta_1 - \beta_2|}{\phi} \leq \alpha \tag{3.10}$$

where β_i is the number of cells in partition i and ϕ is the total number of cells in the circuit.

3.4 Multi-objective Optimization

Many real-world optimization problems involve two types of difficulties: a) multiple possibly conflicting objectives, and b) a highly complex search space. On one hand, instead of a single optimal solution, competing goals give rise to a set of compromise solutions, generally denoted as pareto-optimal (explained later in this section). 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. VLSI netlist partitioning 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 param-

eters (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,

$$minimize \quad y = f(x) = (f_1(x), f_2(x), ..., f_k(x)) \tag{3.11}$$

$$subject\ to \quad e(x) = (e_1(x), e_2(x), ..., e_m(x)) \leq 0 \tag{3.12}$$

$$Where \quad x = (x_1, x_2, ..., x_n) \in X \tag{3.13}$$

$$and \quad y = (y_1, y_2, ..., y_k) \in Y \tag{3.14}$$

then 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 $e(x)$ [10].

In the partitioning problem the three objectives, cutset ($f_1 = \text{Cost}_{cutset}$) as in Eq. 3.2, power ($f_2 = \text{Cost}_{power}$) as in Eq. 3.9 and delay ($f_3 = \text{Cost}_{delay}$) as in Eq. 3.3, are to be minimized under balance constraint ($e_1 = Balance_{max}$) as given in Eq. 3.10. Then an optimal solution might be a partition which achieves minimal cutset, minimal power dissipation, with minimal delay and does not violate the balance criterion. If such a solution exists, we actually have to solve only a single objective optimization (SOP) because the optimal solution for any objective is also the optimum the others [10] (this is in case the objectives are not conflicting).

However, what makes MOP 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 [44]. It is possible that for a certain change in the partitioning solution there would be a decrease in one cost while producing an increase in other costs for e.g., it is possible that a certain change decreases the overall cutset, while increasing the power dissipation, or the overall delay. Therefore, it is needed to solve the partitioning problem as an MOP.

3.4.1 Goal Programming

In this aggregation method, the decision maker has to assign targets or goals that one 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,

$$minimize \quad f(x) = \sum_{i=1}^{k} w_i \left| f_i(x) - T_i \right|, \qquad subject\ to\ x \in X_f \tag{3.15}$$

where T_i denotes the target or goal set by the decision maker for the i^{th} 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 the goal programming objective function is to consider the weighted sum of the p^{th} power of the deviation $|f_i(x) - T_i|$. Such a formulation called *generalized goal programming* [45].

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.

3.4.2 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 both true or false with some degree.

Fuzzy Sets

A classical crisp set is normally defined as a 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). Unlike this a fuzzy set can be defined as,

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

where $\mu_A(x)$ 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 [46]. 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 propositions based on linguistic variables and inference rules [47]. By linguistic variable we mean a variable whose values are words or sentences in natural or artificial language [48]. The linguistic variables can be composed to form propositions using connectors like AND, OR and NOT. Formally, a linguistic variable comprises five elements [49].

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.

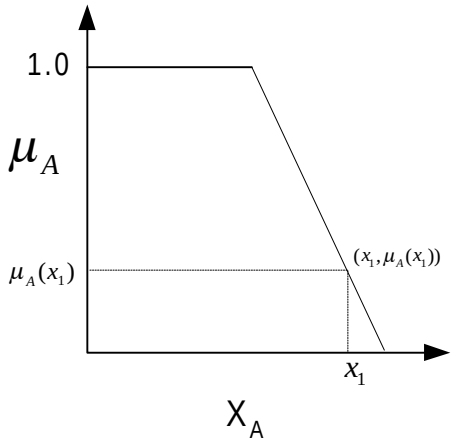

Figure 3.2: Membership function of a fuzzy set A.

5. A set of semantic rules that assign each element in the term set a linguistic meaning.

For example wire-length can be used as linguistic variable for VLSI placement problem. According to the syntactical rule, the set of linguistic values of wire-length may be defined as very short, short, medium, long, very long and very very long. The universe of discourse for linguistic variable is positive range of wire-length of a design, e.g., $[25\mu m,\ 80\mu m]$. 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. 3.2. It shows the designer's knowledge of the problem [47].

Fuzzy Operators

There are two basic types of fuzzy operators: operators for intersection, interpreted as the logical "AND," and union, interpreted as the logical "OR". The intersection operators are known as triangular norms (t-norms), and union operator as triangular conorms (t-conorms or s-norms) [46]. Normally "OR" logic is implemented using maximum operator defined as

$$\mu(x) = \ max\{\mu_A(x), \mu_B(x)\} \tag{3.16}$$

whereas, "AND" logic is normally implemented using minimum operator defined as

$$\mu(x) = \ min\{\mu_A(x), \mu_B(x)\} \tag{3.17}$$

Also the fuzzy complementation operator is defined as

$$\bar{\mu}_B(x) = \ 1 - \mu_B(x) \tag{3.18}$$

Ordered Weighted Averaging Operator

Generally, formulation of multi criteria decision functions do not desire pure "AND-ing" 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 submis-

sion 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 [50, 51]. This operator allows easy adjustment of the degree of "ANDing" and "ORing" embedded in the aggregation. According to Yager [50, 51], "ORlike" and "ANDlike" OWA for two fuzzy sets A and B are implemented as given in Eq. 3.19 and Eq. 3.20 respectively,

$$\mu_{A\cup B}(x) = \beta \times \max(\mu_A, \mu_B) + (1-\beta) \times \frac{1}{2}(\mu_A + \mu_B) \tag{3.19}$$

$$\mu_{A\cap B}(x) = \beta \times \min(\mu_A, \mu_B) + (1-\beta) \times \frac{1}{2}(\mu_A + \mu_B) \tag{3.20}$$

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, and then these combined into linguistic rules using ("and" and "or" logic). 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 function of some base value based on the numerical value of the actual cost. All the membership values are combined into one, using t-norm or s-norm operators. The selection of these operators depends upon the predefined linguistic rule. The combined membership value is now used as an aggregating function. The best solution is that, which results in the highest such combined value.

3.5 Fuzzy Goal Based Aggregation for VLSI Partitioning Problem

In this method it is assumed that there are Γ pareto-optimal solutions. A solution is called pareto-optimal (or efficient) solution, if there exist no other solution for which at least one criterion has a better value without adversely affecting other criteria. In other words, one can not improve any criterion without deteriorating a value of at least one other criterion. Also a p-valued cost vector $C(x) = (C_1(x), C_1(x), ..., C_p(x))$, where $x \in \Gamma$ is given. There is a vector $O = (O_1, O_2, ..., O_p)$ that gives the lower bounds on the cost for each objective such that $O_j \leq C_j(x)\ \forall j$, and $\forall x \in \Gamma$. These lower bounds are normally not reachable in practice. There is another user defined goal vector $G = (g_1, g_2, ..., g_p)$ that represents the relative acceptance limits for each objective. It means that x is an acceptable solution if $C_j(x) \leq g_j \times O_j,\ \ \forall j$ where $g_j \geq 1.0$. For a two dimension problem, Fig. 3.3 shows the region of acceptable solution.

In order to solve the multiobjective partitioning problem, the linguistic variables are defined as cutset, power dissipation, delay and balance. The following fuzzy rule is used to combine the conflicting objectives.

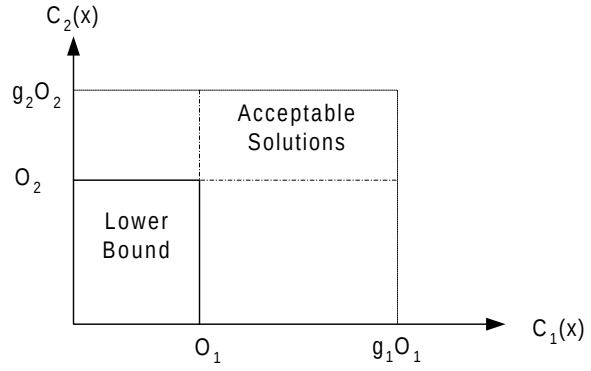

Figure 3.3: Range of acceptable solution set.

Rule R1:
IF a solution is within
acceptable cutset
AND
acceptable power dissipation
AND
acceptable delay
AND
good balance
THEN it is an acceptable solution.

The above mentioned linguistic variables are mapped to the membership values in fuzzy sets within *acceptable cutset*, within *acceptable power dissipation* and within *acceptable delay* . These membership values are computed using the fuzzy membership functions as shown in Fig. 3.4(a). Partitioning aim to produce a balanced solution. where the balance cost refers to the absolute difference in number of cells between the two partitions. Balance can be considered as a constraint in this case; its membership function is shown in Fig. 3.4(b). If the balance is within acceptable range (tolerance), defined by $g_{balance}$ then its membership is 1 otherwise 0. However,

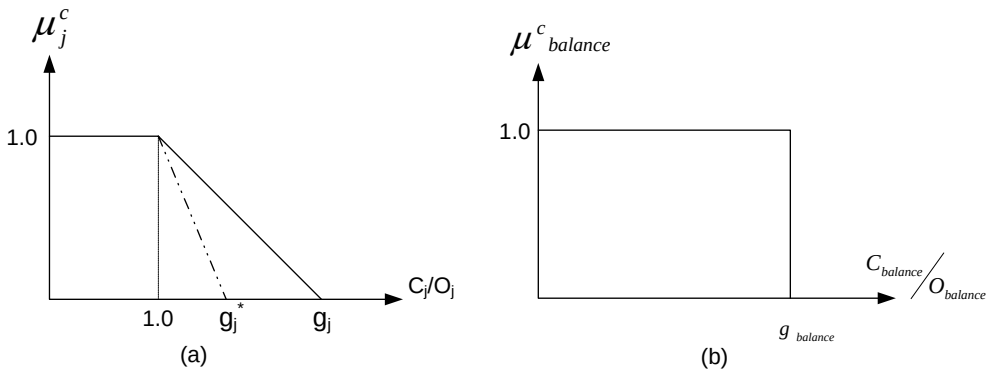

Figure 3.4: Membership functions within acceptable range.

the balance can also be regarded as an objective and its membership function will be as shown in Fig. 3.4(a). For maximizing the performance, it was experimentally found that for Genetic Algorithm and Tabu Search, balance was better considered an objective thus allowing more flexibility in the search process, while for SimE and PowerFM algorithms it was taken as a constraint. For each objective the goal is to have membership value equal to 1. The lower bounds O_j (shown in Fig. 3.4) for different objectives are computed as follows:

$$O_b = 1, \qquad \text{(to avoid divide by zero)} \tag{3.21}$$

$$O_p = \sum_{i \in \zeta} (C_i^{basic}) N_i \qquad \forall v_i \in \{v_1, v_2, ..., v_n\} \tag{3.22}$$

$$O_d = \sum_{j=1}^{k} CD_j, \qquad \forall v_j \in \{v_1, v_2, ..., v_k\} \text{ in path } \pi_c \tag{3.23}$$

$$O_c = 1. \tag{3.24}$$

where O_j for $j \in \{b, p, d, c\}$ are the lower bounds on the costs for balance, power dissipation, delay and cutset respectively while n is the number of nets in the circuit. CD_j is the switching delay of the cell j driving net v_j, N_i is the switching probability of net v_i, π_c is the most critical path with respect to optimal interconnect delays (assuming that no net on this path is cut), k is the number of nets in π_c. The minimum power is obtained if no net is cut, which means substituting 0 for C_i^{extra} in Eq. 3.8. The components of the goal vector G are calculated as follows:

$$g_{balance} = |\zeta| \tag{3.25}$$

$$g_{cut} = \frac{n}{O_c} \tag{3.26}$$

$$g_{delay} = \frac{Initial\ delay}{O_d} \tag{3.27}$$

$$g_{power} = \frac{Initial\ power}{O_p} \tag{3.28}$$

Where $|\zeta|$ is the number of cells in the circuit. Interpreting rule $R1$ as per Eq. 3.20:

$$\mu^c_{pdcb}(x) = \beta^c \times min(\mu^c_p(x), \mu^c_d(x), \mu^c_c(x), \mu^c_b(x)) + (1 - \beta^c) \times \frac{1}{4} \sum_{j=p,d,c,b} \mu^c_j(x) \tag{3.29}$$

where $\mu^c(x)$ is the membership of solution x in fuzzy set of acceptable solutions, $\mu^c_{pdcb}(x)$ is the membership in fuzzy set of "acceptable power AND acceptable delay AND acceptable cutet", whereas $\mu^c_j(x)$ for $j = \{p, d, c, b\}$, are the membership values

in the fuzzy sets within acceptable *power*, within acceptable *delay*, within acceptable cutset and within acceptable *balance* respectively. β^c is the constant in the range $[0, 1]$. In general, $\mu^c(x)$ is used as the aggregating function. The solution that results in maximum value of $\mu^c(x)$ is reported as the best solution found by the search heuristic.

Chapter 4

Iterative Algorithms for Multiobjective VLSI Netlist Partitioning

4.1 Introduction

Once the aggregating function for MOP is obtained, it is then possible to use search techniques to find optimal solutions. Several search techniques have been proposed in literature, like Genetic Algorithms (GA), Simulated Annealing (SA), Tabu Search (TS), and Simulated Evolution (SimE). In this thesis GA, TS, and SimE are used as main search heuristic approaches. A new heuristic called PowerFM which is a modification of the well known Fiduccia Mattheyses algorithm is presented. PowerFm

considers minimization of power consumption due to the cut nets. This chapter discusses the implementation details of Genetic Algorithm, Tabu search, SimE for multiobjective VLSI netlist partitioning as well as PowerFM for power optimization.

4.2 Genetic Algorithm (GA) For Timing and Low Power Driven Partitioning

Genetic Algorithm (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 Fig. 4.1.

There have been many efforts involving application of GA to the VLSI partitioning problem. Earliest application of GA for the min-cut bisection was proposed by Akley [52]. Later, Bui and Moon utilized GAs for graph bisection [53]. Recently, use of GA has gained popularity, especially for multiobjective optimization problems (MOP). However, no previous effort was done to use the GA in a MOP partitioning, involving power, delay, and cutset together.

4.2.1 Chromosome Encoding and Initial Solution

GA uses an encoded representation of the solution in the form of a string made up of symbols called *genes*. The string of genes is called *chromosome*. One way to represent the partitioning problem (as seen in Fig. 4.2(a)) is to use *group-number*

```
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 4.1: A typical Genetic Algorithm [3].

encoding where the j^{th} integer $i_j \in \{1, \cdots, k\}$ indicates the group number assigned to object j. This representation scheme creates a possibility of applying standard operators. The second representation scheme is shown in Fig. 4.2(b). Here, the solution of the partitioning problem is encoded as $n + k - 1$ strings of distinct integer numbers (in case of bipartitioning the string will be seperated into two parts). This representation scheme leads to 100% feasible solutions but requires more computation time due to the complexity of the crossover operators involved [54]. In this implementation the first representation was used.

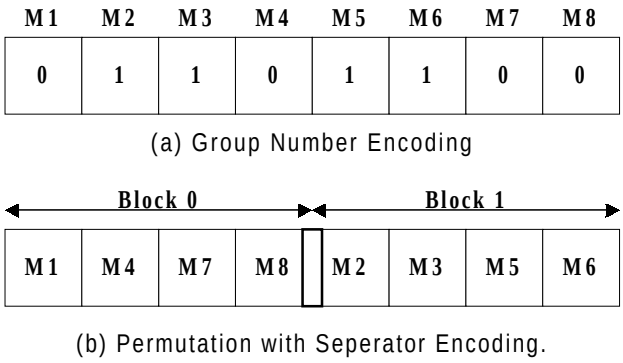

Figure 4.2: Representation schemes.

The algorithm starts with a set of initial solutions called *population* that is generated randomly or taken from the results of a constructive algorithm (e.g., FM). Random initial solutions are usually generated within the limit of the balance constraint.

4.2.2 Fitness Evaluation

For addressing a MOP to minimize many mutually conflicting objectives, a measure is needed which can quantify the overall quality of a solution with respect to all the objectives collectively.

Fuzzy membership functions and fuzzy rules are used for evaluating the fitness of a solution. A fitness value between 0 and 1 is assigned to each solution. The fitness value of a chromosome is its membership value $\mu(x)$ in the fuzzy set of acceptable solution. This membership is computed using Eq. 3.29.

The fitness of a solution is a measure of its proximity to the optimal solution. The higher the fitness value of a solution, the closer it is to the optimal solution. In this thesis experiments, initial random solutions are assigned a membership value of 0 and the optimal solution is assigned a fitness value of 1. This implies that any solution may have a fitness value in the range (0.0-1.0).

4.2.3 Crossover and Mutation

In each iteration (*known as generation*), all the individual chromosomes in the population are evaluated using a *fitness function*. Then, in the *selection* step, two of the above chromosomes are selected from the population. Individuals with higher fitness values are more likely to be selected. After the selection step, different operators, namely, *crossover and mutation*, act on the selected individuals for producing new

individuals called *offsprings*. These genetic operators are described below.

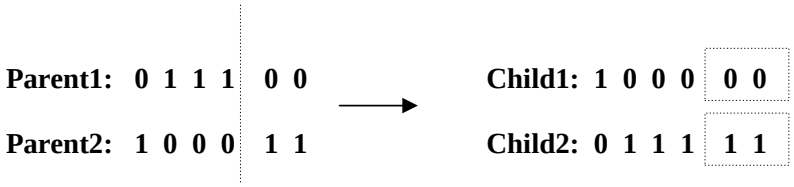

Figure 4.3: Standard one point *crossover* operator (for group number encoding).

One important genetic operator is *crossover*. It is applied on two individuals that are selected in the selection step earlier to generate an offspring. The generated offspring inherits some characteristics from both its parents in a way similar to natural evolution. There are different *crossover* operators, namely, *simple (single point), order, partially mapped,* and *cycle*. The simple *crossover* Fig. 4.3 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 [3]. For description of other *crossover* operators see [3, 5, 55]. Increasing the number of *crossover* points is known to be multi-point *crossover*. For the group number encoding two-point and three-point *crossover*s are favored. The *crossover* operator however may produce children that violate the balance constraint. These can be treated in two ways, either discarding them by giving them a fitness value of zero, or fixing them using some constructive heuristic. This requires too much time overhead. In this implementation the simple single

point crossover is used.

The *mutation* operator is used to introduce new random information in the population. It is usually applied after the *crossover* operator. It helps in producing some variations in the solutions so that the search does not get trapped 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 memoryless search process [5]. Two *mutation* variations are used:

1. Random selection of a cell and swapping its partition.
2. Randomly choosing two cells, one from each partition, and swapping them.

4.2.4 Selection

Individuals for the next population are selected based on the *elitist-random selection (ernd).* $\frac{N_p}{2}$ best chromosomes are selected and remaining $\frac{N_p}{2}$ are selected randomly. Based on experimental results, this scheme offers better choice than other schemes, because it provides a balance between greediness and randomness.

The quality of the solution obtained from GA is dependent on the choice of certain parameters such as the population size, *crossover*, and *mutation* rates, and also the type of *crossover* used. The selection of values for these parameters is

problem specific and there are no hard and fast rules for this purpose. The choice of these parameters is left to the intuition of the person applying GA to a specific problem.

4.3 Tabu search (TS) for VLSI Netlist Partitioning

Tabu search has been applied for solving the problem of partitioning with single objective (cutset) graph bisection by Tao et al. [56], Lim Chee [57], and Areibi et al. [58]. However the multiobjective partitioning problem involving *power*, *delay*, and *cutset* was not approached previously using this algorithm.

Tabu search is an iterative heuristic that has been applied for solving a range of combinatorial optimization problems in different fields [3]. 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

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 : Aspiration 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

Figure 4.4: Outline of Tabu Search algorithm [3].

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 TS heavily depends on the size of Tabu list as well as on the chosen aspiration criterion. 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 Fig. 4.4. The detailed description of Tabu search can be found in [3].

4.3.1 Tabu list and aspiration criteria

Tabu list introduces memory element in the search process. Its purpose is to avoid revisiting a point (solution) in the search space. This is implemented by storing some characteristic of a certain number of previously accepted moves.

Implementation of Tabu list requires two decisions to be made. First, what characteristic of the move should be stored in the list. This decision has a significant effect on search quality as well as memory requirements of TS. The chosen characteristic should identify the move, so that it can accurately be used to restrict the corresponding move and to consequently fulfill the purpose of Tabu list. In the present implementation, the characteristic of the move stored in Tabu list is the index of any cell involved in the move from one block to another.

The second decision that needs to be made is regarding the size of Tabu list. This decision affects time and space requirements of TS. Tabu list management concerns updating the Tabu list i.e., deciding on how many and which moves have to be made within any iteration of the search. The size of the Tabu list can noticeably affect the final results; a long list may give a lower overall minimum cost, but is usually obtained in a long time. Further, a shorter list may trap the routine in a cyclic search pattern. It is deduced experimentally, that Tabu list sizes that provide good results often grow with the size of the problem.

Aspiration Criteria is a rule based on cost values that can temporarily release a solution from its tabu status. The purpose of aspiration criteria is to increase the flexibility of the algorithm while preserving the basic features that allow the algorithm to escape from the local optima and avoid cyclic behavior. There are two types of aspiration criteria that were employed in experiments. The first one is to give an aspiration for each cost reached. Throughout the search procedure, each cost c has an aspiration value that is the current lowest cost of all solutions arrived at from solutions with value c. The actual aspiration rule is that, if the cost associated with a tabu solution is less than the aspiration value associated with the cost of the current solution, then the tabu status of the tabu solution is temporarily ignored and the move is accepted. The second aspiration criterion used is simple, which says that if the best neighbor solution of the current iteration is better than

the global best solution, then Tabu list restrictions are overridden and the solution is accepted. Also the global best solution and the aspiration criterion is updated. The short term memory functions in Tabu Search are fulfilled by the Tabu list and aspiration level.

4.3.2 Intermediate and long term memory

Intermediate term memory operates by recording and comparing features of a selected number of best trial solutions generated during a particular period of search. The method then seeks new solutions that exhibit these features. The long term memory functions, whose goal is to diversify the search, employs principles that are roughly the reverse of those for intermediate term memory. The implementation of the long term memory (search diversification) is based on two different techniques:

- In the first technique, the frequency of moving a module is used as a means of exploring new solutions that have not been visited before. Modules having high frequency are discouraged for further movement. This allows less frequented moves to be explored. This is done either by making the tabu status of module proportional to its frequency, or by locking the module that has very high frequency (simply by using a counter to count the number of times each cells is moved).

- The second form of diversification that allows the meta-heuristic to come out

of local optima is to focus more specifically on producing initial solutions as different as possible from the solutions generated throughout the previous history of the search process. These solutions are then used as a restart to get out of local minimum.

Diversification of the search is usually invoked when a number of consecutive moves occur without an improvement over the best solution or in the case when all the solutions in the current neighborhood are tabu and none of them satisfies an aspiration criterion.

The intermediate term memory for search intensification uses a simple approach that allows the meta-heuristic to come out of local optima. It basically records solutions in a queue with a certain length. As the search proceeds the system identifies a few best solutions and uses them in a restart phase to intensify the search into promising regions not fully explored [58].

4.3.3 Data structure and Stopping Criterion

For many applications, the choice of the proper data structure is really the only major decision involved in the implementation. Once the choice has been made, only very simple algorithms are needed. For the same data, some data structures require more or less space than others; for the same operations on the data some data structures lead to more or less efficient algorithms than others. The saving of

time or space by making a proper choice is of prime interest. In case of TS applied for single objective *cutset* optimization a data structure of linked list is used, similar to that shown in Fig. 1.2. This data structure provides higher speed than simple string data structure. The latter is used in the case of MOP for both GA and TS to avoid implementation complexities.

TS was run for different number of iterations, and depending on experimental observations a fixed run-length was decided as done in the case of GA. This limit on the number of iterations was placed because no significant improvement was observed beyond this point.

4.4 Simulated Evolution Algorithm (SimE)

In this section, Simulated Evolution Algorithm [59] is summarized. The pseudo-code of SimE is given in Fig. 4.5. SimE operates on a single solution, termed as *population*. Each population consists of elements. In case of partitioning problem, these elements are cells to be moved. In the *Evaluation* step *goodness* of each element is measured. Goodness of an element is a single number between '0' and '1', which is a measure of how near the element is from its optimal location. Higher value of goodness means that the element is near its optimal location. The goodness can be calculated as follows,

$$g_i = \frac{O_i}{C_i} \tag{4.1}$$

ALGORITHM *Simulated_Evolution*(B, Φ_i, SC)

B = Bias Value.
Φ = Complete Solution.
Φ_i = Initial Solution.
SC = Stopping Criterion.
e_i = Individual link in Φ.
O_i = Lower bound on cost of i^{th} link.
C_i = Current cost of i^{th} link in Φ.
g_i = Goodness of i^{th} link in Φ.
S = Queue to store the selected links.
Allocate(e_i, Φ_i) allocates e_i in partial solution Φ_i.

Repeat

EVALUATION: **ForEach** $e_i \in \Phi$ **DO**
Begin
Evaluate g_i
End

SELECTION: **ForEach** $e_i \in \Phi$ **DO**
Begin
If random$(1) > \min(g_i + B, 1)$
Begin
$S = S \cup e_i$;
$\Phi = \Phi - e_i$;
End
End
sort(S)

ALLOCATION: **ForEach** $e_i \in S$ **DO**
Begin
Allocate(e_i, Φ_i)
End

Until *SC is satisfied*
Return (Best solution)
End *Simulated_Evolution*

Figure 4.5: Structure of the simulated evolution algorithm.

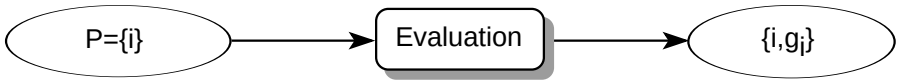

Figure 4.6: Evaluation.

where O_i is the estimate of lower bound on the cost of individual i, and C_i is the actual cost of i. O_i is independent of iterations and therefore it is estimated only once in the beginning. Whereas, C_i has to be calculated in every iteration for every element. The evaluation step is shown in Fig. 4.6.

Selection is the process of selecting those individuals which are unfit (badly placed) in the current solution. For this purpose, goodness of each individual is compared with a random number (in the range [0,1]). If the goodness is less than the random number, then it is selected, otherwise rejected, and it is assigned to an empty location. A selection bias 'B' can also be used to restrict the number of selected individuals to some limit. 'B' is also used to compensate for errors made in the estimation of goodness. Typical range of 'B' is [-0.2, 0.2]. A variable bias scheme is also suggested in [47]. Inputs to the selection process are elements of population P and their respective goodness values, whereas outputs are a selection set P_s, and P_r the set of remaining elements of population P. The selection procedure is illustrated in Fig. 4.7.

The Selection operator is non-deterministic in nature. An individual having high

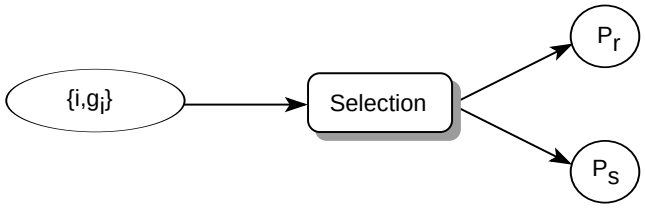

Figure 4.7: Selection.

goodness measure still has a non-zero probability of assignment to set P_s. It is this element of non-determinism that gives SimE the capability of escaping local minima.

After the selection process, the *allocation* operator is used to place cells in P_s to new locations. It is mentioned in [3] that random allocation leads the algorithm into random search. Therefore, it is necessary to adopt some greedy allocation strategy, so as to improve the goodness of cells, which reduces the overall cost of the solution. Inputs to the allocation operator are the sets P_s (selection) and P_r (remaining) and the output is the new population P', as shown in Fig. 4.8.

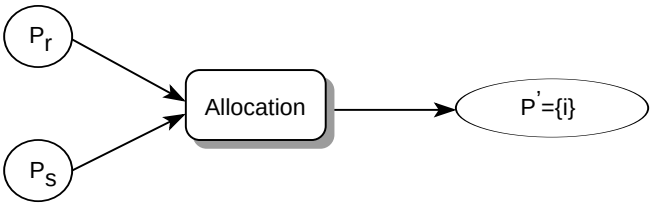

Figure 4.8: Allocation.

The choice of allocation function is problem specific. The decision of the allocation strategy usually requires more ingenuity on the part of the designer than the selection scheme. The choice of proper allocation strategy affects the quality of final

solution considerably. Different constructive allocation schemes have been proposed in literature such as *sorted individual best fit, weighted bipartite matching allocation* and *branch and bound search allocation* [60]. In this work, sorted individual best fit is adopted.

The whole algorithm is repeated until some stopping criteria is met. Stopping criteria may be the number of iterations for which there is no further improvement in the overall cost or total number of iterations are completed. At the end of *Repeat* loop, the algorithm returns the best solution found.

4.5 Simulated Evolution (SimE) for Performance Driven, Low Power VLSI Netlist Partitioning

The earliest implementation of the Simulated Evolution (SimE) algorithm for the partitioning-class problems was done by Saab and Rao [61]. They applied SimE to a graph bisection problem where given an bisection (V_1, V_2) of a hypergraph $H(V, E)$, for each cell $i \in V$ the following were defined:

$$E_i = \begin{cases} \sum_{j \in V_1} c_{ij}, & \text{if} \quad i \in V_2 \\ \sum_{j \in V_2} c_{ij}, & \text{if} \quad i \in V_1 \end{cases} \tag{4.2}$$

$$I_i = \begin{cases} \sum_{j \in V_1} c_{ij}, & \text{if} \quad i \in V_1 \\ \sum_{j \in V_2} c_{ij}, & \text{if} \quad i \in V_2 \end{cases}$$

It is clear from the above definitions that I_i measures how strongly cell i is attached to its partition, and E_i measures how strongly cell i is attached to the complement of its partition. One can think of I_i and E_i as the internal and external attraction of cell i respectively. The "goodness" of a module v_i is taken as $\frac{E_i}{I_i}$, if the goodness of cell i is less than a given random number between 0 and 1, then cell i is judged to be good; otherwise it is bad. Intuitively, a good cell should remain in its current partition since it is likely to have many internal connections while, a bad cell should be moved to the other partition. The *mutation* operator essentially swaps large subsets of bad vertices. In this thesis, we employ SimE for *Hypergraph VLSI paritioning*, in both single objective and MOP optimization.

4.5.1 Proposed Scheme and Implementation Details

In this section, we discuss several important implemetation details of SimE algorithm. The description is combined with fuzzy logic implementation. In the proposed algorithm interconnect power dissipation, overall circuit delay and cutset are used as objectives and balance is taken as a constraint, which effectively makes it a multiobjective optimization problem. In order to solve this MOP, fuzzy logic is used providing a convenient method to combine possibly conflicting objectives. From the

pseudo code of SimE given in Fig. 4.5, it is clear that fuzzy logic can be applied at two different stages of the algorithm. These are *evaluation* and *allocation*. In the proposed algorithm, fuzzy logic is applied to the evaluation stage.

4.5.2 Proposed Fuzzy Goodness Evaluation Scheme

In this stage of the algorithm, goodness of individual cell is computed. Where gd, gp and gc are defined as the delay, power and cut goodness of a cell respectively. A fuzzy membership for each goodness function is then derived, in order to get the overall goodness of the cell in its partition.

Cut Goodness Taking into account the hypergraph representation of the circuit, the goodness function gc is defined and computed as follows,

$$gc_i = \frac{d_i - w_i}{d_i} \tag{4.3}$$

where gc_i is the goodness of cell i, $V_i = \{v_1, v_2, ..., v_k\}$ is the set of nets connected to cell i, U_i is a subset of V_i containing the connected nets to cell i that are cut, the cardinality of V_i is expressed as d_i. The net e_k is said to be cut if and only if cell $u \in e_k$ and cell $e \in v_k$ and $Block(u) \neq Block(v)$. The number of nets connected to i and having the status as cut is expressed as w_i. The cut goodness is simply the

number of uncut nets over the total nets connected. Since gc_i is between 0 and 1, we can take the fuzzy membership μ_c as equal to the goodness $\mu_c = gc_i$. An example of goodness calculation is shown in Fig. 4.9, the goodness of the cell 5 is calculated as follows: $gc_5 = \frac{3-2}{3} = 0.33$.

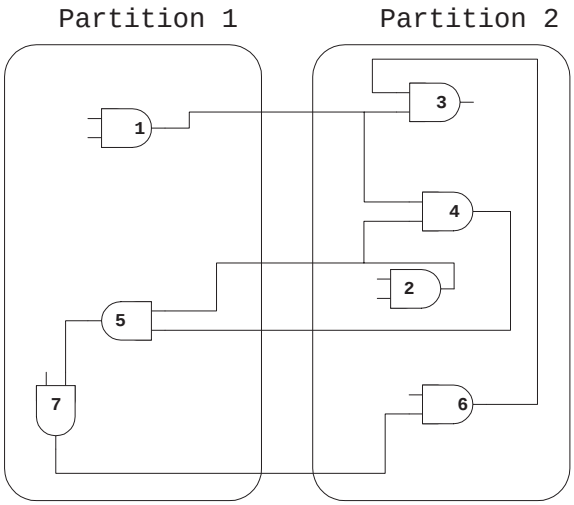

Figure 4.9: Cut Goodness calculation.

Power Goodness The power goodness gp_i of a cell is defined as a measure of how well placed is the cell in its present block according to power consumption and can be computed as follows:

$$gp_i = \frac{\sum_{j=1}^{k} S_j \ (j \ \in \ V_i) - \sum_{j=1}^{k} S_j \ (j \ \in \ U_i)}{\sum_{j=1}^{k} S_j \ (j \ \in \ V_i)} \tag{4.4}$$

S_j is the switching probability of the cell that drives the net. The goodness is equal to the sum of the switching probabilities of the cells that are driving the

uncut nets over the sum of the switching probabilities of the cells that are driving all nets connected. In this way a cell is placed in the partition where the sum of the switching probability of the cut nets is optimized. Results show that this goodness function gives high quality solutions with less power dissipation. Since $0 \leq gp_i \leq 1$ we can take the fuzzy membership $\mu_p = gp_i$. An example of power goodness calculation is shown in Fig. 4.10, the goodness of cell 5 is calculated as follows: $gc_5 = \frac{0.7-0.4}{0.7} = 0.428$.

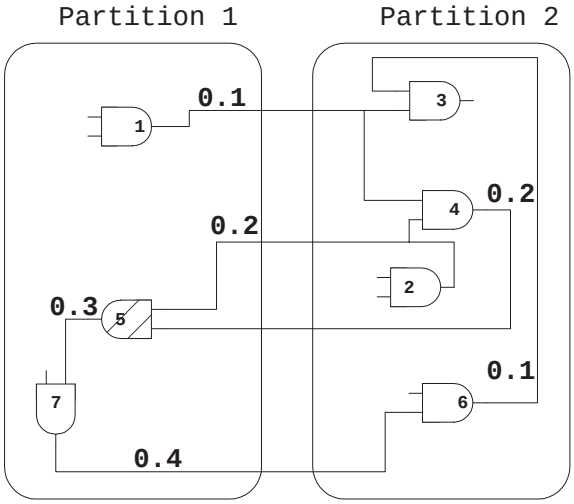

Figure 4.10: Power Goodness Calculation.

The power and cutset objectives are possibly conflicting. Hence it is possible to find alternative solutions for a specific circuit for e.g., there may exist one with high number of cuts and low power consumption (because the nets cut have less switching probability) as well as another lower cuts and higher power consumption.

Delay Goodness In our problem, we deal with multi-pin nets, which makes it hard to design a suitable and simple delay goodness function. We propose the following delay goodness.

$$gd_i = \frac{|K_i| - |L_i|}{|K_i|} \tag{4.5}$$

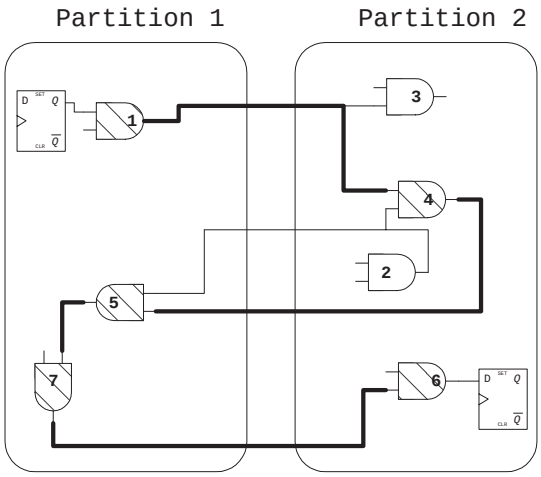

Figure 4.11: Delay Goodness Calculation.

Where gd_i is the delay goodness of cell i. We consider the set of all *critical paths* passing through i and define the set K_i as the set of all cells connected to these paths. We also define L_i as a subset of K_i, containing those cells which are connected to all critical paths passing through i and are not in the same block as i. This goodness function will tend to drive the cells that are connected by the same path to the same block, thus minimizing the delay along the path. A cell is considered good in its

block if the majority of cells connected to all paths passing through it are also placed in the block. An example for delay computation is given in Fig. 4.11. To calculate gd_4 we first compute $|K_4| = 5$ for the critical path {1,4,5,7,6} which is the only one connected to cell 4. $|L_4| = 3$ which are cells $\{1, 5, 7\}$. This gives $gd_4 = \frac{5-3}{5} = 0.4$. However, $gd_5 = 0.6$ and hence is better placed according to the delay consideration.

4.5.3 Proposed Fuzzy Evaluation Scheme

With the classical goodness of cut only, it is possible that a cell having a high goodness with respect to cut may not be selected even thought it is badly placed with respect to circuit delay and power. In order to overcome this problem, it is necessary to include power and delay in the goodness measure along with cut goodness. Also, it is not desirable to select all the cells even if they all have a low goodness value. In this case, it is desirable to select those cells which are far from their lower bounds as compared to other cells in the design. For this purpose the following fuzzy rule is proposed.

Rule R1:

IF cell i is
near its optimal cut-set goodness (*as compared to other cells*)
AND
near its optimal power goodness (*as compared to other cells*)
AND
near its optimal delay goodness (*as compared to other cells*)
OR
$T_{max}(i)$ *is much smaller than* T_{max} (*as compared to other cells*)
THEN it has a high goodness.

Where T_{max} is the delay of most critical path in the current iteration and $T_{max}(i)$ is the delay of the longest path traversing cell i in the current iteration.

The above-mentioned fuzzy rule is interpreted. Using Eq. 3.19 and Eq. 3.20 as follows:

$$g_i = \mu_i^e(x) = \beta^e \times min(\mu_{ic}^e(x), \mu_{ip}^e(x), \mu_{id}^e(x)) + (1 - \beta^e) \times \frac{1}{3} \sum_{j=c,p,d} \mu_{ij}^e(x) \qquad (4.6)$$

where

$$\mu_{id}^e(x) = \beta_d^e \times max(\mu_{dg}^e(x), \mu_{ipath}^e(x)) + (1 - \beta_d^e) \times \frac{1}{2}(\mu_{dg}^e(x) + \mu_{ipath}^e(x)) \qquad (4.7)$$

The superscript e is used here to represent evaluation so that other fuzzy notations in other steps of SimE can be distinguished. The term x represents the block of cell i, $\mu_i^e(x)$ is the membership in the fuzzy set of high goodness and g_i is the goodness of cell i. β^e and β_d^e are constants between 0 and 1 to control OWA operators. $\mu_{ic}^e(x)$ and $\mu_{ip}^e(x)$ represent the membership in fuzzy sets of near optimum cutset and near optimum power as compared to other cells. Moreover, $\mu_{id}^e(x)$ is the overall delay goodness, and represents the membership in fuzzy set of near optimum timing performance. It is obtained after applying "OR-like" OWA to $\mu_{dg}^e(x)$ and $\mu_{ipath}^e(x)$, which are the memberships in fuzzy sets of near optimum cell delay goodness as compared to other cells and $T_{max}(i)$ (most critical path passing trough cell i) is much smaller than T_{max} (current most critical path of the circuit).

$\mu^e_{ipath}(x)$ is included in the computation of $\mu^e_{id}(x)$ because if a cell is not in the critical path then it must have high goodness with respect to delay objective. After considerable number of iterations, it is possible that a cell is in the critical path but is also very near to its optimal delay goodness. In that case it is not possible to optimize it further. At this stage, $\mu^e_{dg}(x)$ overrides $\mu^e_{ipath}(x)$.

The base values for cutset and power are not needed since the membership is directly computed as described in section 4.5.2. As for cell delay goodness it is composed of net delay $\mu^e_{dg}(x)$ which is computed directly by using Eq. 4.5. For computing $\mu^e_{ipath}(x)$ we define base value $X_{ipath}(x)$ for fuzzy set { $T_{max}(i)$ much smaller than T_{max}}, and is computed in Eq. 4.8, the membership function is illustrated in Fig. 4.12. (experimentally we found that a base value of 2 is suitable to quantify that $T_{max}(i)$ is much smaller than T_{max}).

$$X^e_{ipath}(x) = \frac{T_{max}}{T_{max}(i)} \tag{4.8}$$

4.5.4 Selection

In this stage of the algorithm, some cells are selected probabilistically depending upon their goodness values. As proposed by Kling et al [60], for each cell i, a random number in the range [0,1] is generated and compared with $g_i + B$, where B is the fixed selection bias. If this generated random number is greater than $g_i + B$

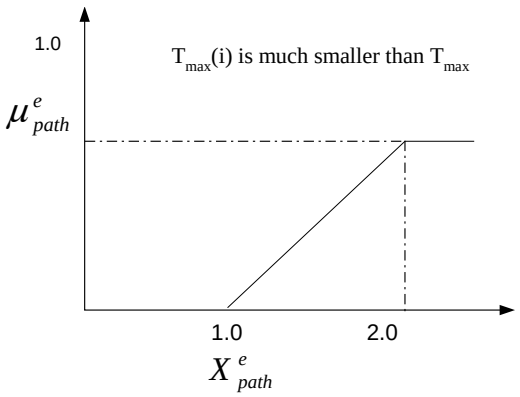

Figure 4.12: Membership function for $T_{max}(i)$ much smaller than T_{max}.

then the cell is selected for allocation. However, determining the value of B is a problem specific issue, varying also from circuit to circuit. For bigger circuits, normally bigger bias values are needed. To overcome this problem, Hussain [47] proposed the idea of a variable bias according to which the value of bias for k^{th} iteration is computed as follows,

$$B_k = 1 - G_k \tag{4.9}$$

where B_k is the bias value for iteration k and G_k is the average goodness of all cells at the end of the k^{th} iteration. The advantages of using variable bias is following.

1. Bias value is not arbitrarily selected and no trial runs are required to find the best value. The variable bias automatically adjusts according to the problem state.

2. For bad quality solutions, the average goodness is low, resulting in a high bias value. This ensures that the size of selection set is not excessively large and thus save the algorithm from making large number of moves.

3. For good quality solutions, the average goodness is high. Therefore, a low bias value is used. It will result in the selection of sufficient number of cells thus protecting the algorithm from early convergence.

Along with these benefits, the following analysis points out some drawbacks of using variable bias.

A cell i will be selected if,

$$Random > g_i + B_k$$

or

$$Random - B_k > g_i$$

or

$$Random - 1 + G_k > g_i$$

or

$$R_{new} > g_i$$

where R_{new} is the new random number in the range $[-(1-G_k), G_k]$. It is clear from this range that probability of selection of cells having goodness $g_i > G_k$ is zero. This

may lead to some local optimal solution because statistically half of the cells have zero probability of selection.

Also probability of no selection is $1 - G_k$. It means that when G_k is low, the size of selection set is small because $1 - G_k$ has a high value, and when G_k is high then selection set is large as $1 - G_k$ gives a low value. This behavior is against the basic idea of SimE according to which size of the selection set has to be decreased with increase in average goodness. It is also against the behavior of any other iterative search algorithm where big perturbations are made when the solution is bad and smaller perturbations are made with improvement in the quality of solution. Another selection criteria where $B_k = G_k$, was also implemented but the experimental results showed that the algorithm converges fast, and does not give better results when compared to "Biasless Selection" discussed in the next section.

4.5.5 Biasless Selection

In this implementation, a scheme proposed by Khan et al [62] where the selection bias B is totally eliminated and a cell is selected if $Random > goodness_i$ is used. This method is as follows:

When number of cells in the problem is large, as in the case of VLSI partitioning, the goodness distribution among the cells is Gaussian, with mean G_m and standard deviation G_σ. In the proposed selection scheme a Gaussian random number is used instead of using uniformly distributed random number. This way the problem of

having cells with zero selection probability can be avoided. The mean R_m and standard deviation R_σ of the random number are calculated as follows

$$R_m = G_m - G_\sigma \tag{4.10}$$

$$R_\sigma = G_\sigma \tag{4.11}$$

If we use G_m as the mean for random number generation, it is most likely that around 50% cells would be selected which is not desirable in case of large number of cells. To avoid such an unfeasible selection set, we change this mean to $G_m - G_\sigma$, so that only $12 - 13\%$ cells will be selected in the initial iterations. These values are determined only in the first iteration of the algorithm. However, with increased number of iterations, average goodness will increase and this may cause very small number of cells to be selected. To avoid this, the mean of random number is updated when the number of selected cells goes down to 5% of total number of cells in the problem as follows:

$$R_m = R_m + 0.1 \times R_\sigma \tag{4.12}$$

this scheme has some significant advantages, (1) there is no cell in the design having zero selection probability, hence avoiding local optima, (2) size of selection set reduces with increase in the average goodness value, and (3) update procedure avoids extremely small selection set.

4.6 Power FM Algorithm

Fidducia Mattheyeses presented an iterative heuristic that takes into consideration multipin nets as well as sizes of circuit elements. As mentioned in Chapter 1, Fidducia-Mattheyeses heuristic (Fig. 1.4) is an efficient and fast technique used in the bipartitioning problem. The move in the Fidducia Mattheyeses is based on the gain due to decrease in the cutset, and is defined as follows: The gain $g(i)$ of a cell i is the number of nets by which the cutset would decrease if cell i were to be moved. A cell is moved from its current block to its complementary block. To avoid having all cells migrate to one block, a balancing criterion is maintained. The algorithm starts by choosing a node which causes maximum gain and does not disturb the balance criteria (to a certain tolerance, usually 10%). The node is moved and the gain of all affected cells is updated. When a cell is moved it is locked; when all cells are moved then this means that we have finished one pass. At the end of a pass, all cells are freed and the process repeated until a point where no further gain can be achieved.

The PowerFM is a modification of the FM algorithm which seeks minimization of the power consumption due to the cut. All concepts of FM are maintained; the major difference is the calculation of the gain due to the sum of the switching probabilities of the cut nets. Also some other necessary modifications are done in some parts of the algorithm and are discussed later next.

4.6.1 Power Gain Calculation

The power gain for a cell i is calculated using Eq. 4.13. X_i is the set of cut critical nets. U_i is the set of uncut critical net. An example of calculation of the power gain is shown in Fig. 4.13, to calculate the gain of moving cell 7 we have $Pgain(7) = 0 - (0.3 + 0.4) = -0.7$ and for cell 1 $Pgain(1) = 0.1 - 0 = 0.1$.

$$Pgain(i) = C_{off}\left(\sum_{j=1}^{k} S_j(j \in X_i) - \sum_{j=1}^{k} S_j(j \in U_i)\right) \tag{4.13}$$

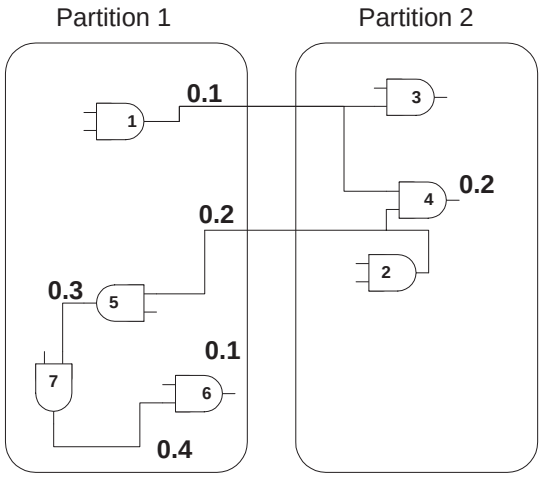

Figure 4.13: Power Gain Calculation.

In each pass, the gain of every free cell is updated according to the Compute_Cell_Gain algorithm shown in Fig. 4.14. Let $F(n)$ be the number of cells connected to net n in the From_block (current block) of the cell i to be moved. Let $T(n)$ be the number of cells connected to net n in the To_block (destination block) of the moved cell i. When computing the gain we consider only the *critical nets* A net

is critical if it has a cell which if moved will change its cutsate. That is if and only if $F(n)$ or $T(n)$ is either 0 or 1. As an example consider cell 1 in Fig. 4.14 and the net denoted as a formed by the connection of cells 1, 3, and 4; hence $a = (1, 3, 4)$. Considering the From_block as partition 1 and the To_block as partition 2; it is clear that $F(a) = 1$ and $T(a) = 2$ then this net contributes positively to the gain by 0.1 (obtained from moving cell 1), and furthermore it is called a critical net. Another example, consider cell 5 and nets $b = (5, 4, 2)$ and $c = (5, 7)$, notice that $F(b) = 1$ and $T(b) = 2$, $F(c) = 2$ and $T(c) = 0$, then net b contributes positively to the gain by 0.2 and net c contribute negatively by 0.3 and both are critical nets. The idea behind the algorithm is simple. It checks if the net is critical, if $F(n) = 1$ then moving cell i will increase the gain by $C_{off} \times (Sw\ prob\ of\ driving\ net)$ and if $T(n) = 0$ then moving the cell i will decrease the gain by $C_{off} \times (Sw\ prob\ of\ driving\ net)$.

4.6.2 General Description

The first step consists of computing the gains of all $free$ cells. Cells are considered to be free if they are not locked either initially by the user, or after they have been moved during this pass. Having computed the gains of each cell, we now choose the "base cell". The "base cell" is one that has the maximum gain and does not violate the balance criterion. If no base cell is found then the procedure stops. When the balance criterion is satisfied, then the cell with maximum gain is selected as the base cell. In some cases, the gain of the cell is non-positive. However, we still move

ALGORITHM *Compute_Cell_gains*;
Begin
For each free cell i **Do**
$g(i) \leftarrow 0$;
F $\leftarrow$ *From_block* of cell i
T $\leftarrow$ *To_block* of cell i
FOR each net n on cell i **DO**
If $F(n) = 1$ **Then** $g(i) = g(i) + (C_{off} \times Sw\ prob\ of\ driving\ net)$
(Cell i is the only cell in the From_block connected to net n.)
If $T(n) = 0$ **Then** $g(i) = g(i) - (C_{off} \times Sw\ prob\ of\ driving\ net)$
(All of the cells connected to net n are in the From_block.)
EndFor
EndFor
End.

Figure 4.14: Procedure to compute gains of free cells.

the cell with the expectation that the move will allow the algorithm to escape out of the local minimum. As mentioned before, to avoid migration of all cells to one block, during each move, the balance criteria is maintained. After each move, the selected cell is locked in its new block for the remainder of the pass. Then, the gains of cells comprising the critical net are updated. If more free cells exist then we search for the next base cell. If found, then we go back and lock the cell, and repeat the update. If no free cells are found then we move on. After all cells have been considered for movement, as in the case of Kernighan-Lin, the best partition encountered during the pass is taken as its output. The number of cells to move is given by the value of k which yields maximum positive gain G_k, where $G_k = \sum_{i=1}^{k} g_i$. Only the cells given by the best sequence, that is $c_1, c_2, ..., c_k$, are permanently moved

to their complementary blocks. Then all cells are freed and the complete procedure is repeated.

Chapter 5

Experiments and Results

5.1 Introduction

This chapter presents the experimental results obtained from the algorithms employed in this thesis, more specifically GA, TS, SimE and PowerFM. Initially the performance of the multiobjective optimization and the single objective optimization of GA and TS is analyzed and compared. Then the GA and TS algorithms are compared in terms of time performance and quality of solution. Simulated Evolution SimE results are presented and compared with TS and PowerFM. Moreover, we study the case of using PowerFM as a starting solution for GA and TS.

Name	Number of cells	number of nets
s298	136	130
s386	172	165
s641	433	410
s832	310	291
s953	440	417
s1196	561	547
s1238	540	526
s1488	667	648
s1494	661	642
s2081	122	121
s3330	1962	1888
s5378	2994	2944
s9234	5845	5822
s13207	8652	8530
s15850	10384	10296

Table 5.1: Circuits details.

5.2 Circuits Details

ISCAS-85/89 benchmark circuits were used in all the experiments. Most of the circuits considered are sequential thus reflecting real life systems where purely combinational circuits are rarely used. However, sequential circuits have more severe delay problems than combinational circuits, as the latter are more often well structured. The key characteristics of these circuits are given in Table 5.1. The details of the cell characteristics are obtained from the 0.25μ MOSIS, TSMC and CMOS technology library [63].

5.3 Single Objective versus Multiobjective Optimization

MOP and SOP are compared to study the relations between the multiple possibly conflicting objectives (delay, cutset, power) and the effects on the overall quality of the final solution. The following three sub-sections present the comparison between MOP and SOP for cutset, delay and power consumption. Each sub-section is divided into both GA, and TS results.

5.3.1 Power-only Optimization versus MOP

Interconnect power dissipation is an important objective to be minimized in VLSI optimization. Hence the first experiment is conducted for power-only optimization and then compared with the multiobjective optimization. The costs of the three objectives for the best solution obtained from GA, and TS are given in Table 5.2, and 5.3 respectively. The implementation specifications are as follows: For GA the population size is 10, probability of crossover is 0.99 and the probability of mutation is 0.09. For TS the number of neighbors is 10 and the Tabu list size is 0.1 times the size of the circuits. C_{off} (off-chip capacitance) is taken as 100 $femtofarad$. Each run is of 10000 generation, and the fuzzy fitness constant $\beta = 0.7$. To save on runtime instead of starting by a completely random solution, this implementation starts by a random solution with a 10% unbalance. The balance criteria is then

taken as an objective in the cost function.

Circuit	For power-only			For all objectives		
	Delay (ps)	Cutset (nets)	power (S.P.)	Delay (ps)	Cutset (nets)	power (S.P.)
S298	407	34	892	233	19	1013
S386	450	37	1327	356	36	1529
S641	1314	72	2195	1043	45	2355
S832	686	59	2861	444	45	3034
S953	685	134	2500	526	96	2916
S1196	550	135	5417	396	123	5443
S1238	471	133	5539	475	127	5713
S1488	747	107	5232	571	104	5648
S1494	181	118	5431	614	102	5474
S2081	292	19	632	302	26	787
S3330	720	358	10331	571	299	10358
S5378	923	653	18509	587	573	18437
S9234	1547	1189	36894	1313	1090	38149
S13207	2243	1908	44801	1399	1683	45611
S15850	2167	2409	51607	1820	2183	51747

Table 5.2: A Comparison between the quality of the best solutions obtained from GA by performing SOP for power consumption and MOP.

As seen in Table 5.2 for all circuits except S5378, it is observed that power-only optimization performs better than MOP in terms of power cost. However, at the same time, the other two objectives are affected adversely for all the circuits. This behavior is easy to understand; when optimization is done for power-only, the target is to minimize the number of the nets cut having high switching probabilities. All other nets may be ignored in this process, and thus overall cutset is not reduced as in the case of multiobjective or cut-only optimization. In Table 5.3 the delay objective is most negatively affected as the process of optimizing power is very limited in terms of number of affected nets and thus it may totally ignore the nets lying on the timing-critical paths. As a result, delay costs are not optimized at all. Furthermore

Circuit	For Power-only			For all objectives		
	Delay (ps)	Cutset (nets)	power (S.P.)	Delay (ps)	Cutset (nets)	power (S.P.)
S298	339	34	870	197	24	926
S386	456	36	1215	386	30	1426
S641	1416	67	2018	889	59	2281
S832	556	51	2619	446	50	2731
S953	765	142	2516	466	99	2518
S1196	541	133	4705	301	106	4920
S1238	584	103	4350	408	79	4597
S1488	705	110	5332	528	98	5529
S1494	620	108	5258	585	101	5339
S2081	431	27	672	225	17	770
S3330	755	339	10239	533	295	10298
S5378	665	538	15847	590	430	16527
S9234	1577	1085	33854	1052	918	34055
S13207	2039	1632	40513	843	1332	41114
S15850	1988	1817	46095	1411	1671	47480

Table 5.3: A Comparison between the quality of the best solutions obtained from TS by performing SOP for power-only and MOP.

when comparing the performance of TS and GA for single objective we see that TS outperformed GA in the case of power-only optimization with respect to the quality of the solution.

5.3.2 Delay-only Optimization versus MOP

In this section, a comparison between SOP for delay and MOP is made. Table 5.4 and Table 5.5 show these comparative values of cutset, power, and delay costs for the solutions obtained from GA and TS respectively.

Delay-only optimization, ignores the other two objectives. The costs of cutset and power consumption are very high in this case for all the circuits. The reason for this behavior is that the optimization for delay targets a limited number of nets

Circuit	For Delay-only			For all objectives		
	Delay (ps)	Cutset (nets)	power (S.P.)	Delay (ps)	Cutset(nets)	power (S.P.)
S298	233	57	1608	233	19	1013
S386	350	61	2537	356	36	1529
S641	1014	158	5030	1043	45	2355
S832	378	108	4945	444	45	3034
S953	485	195	4857	526	96	2916
S1196	392	301	11150	396	123	5443
S1238	385	287	10425	475	127	5713
S1488	552	281	9956	571	104	5648
S1494	610	268	9654	614	102	5474
S2081	252	43	1685	302	26	787
S3330	684	964	22106	571	299	10358
S5378	678	1725	43227	587	573	18437
S9234	1426	3077	84474	1313	1090	38149
S13207	1448	4625	93877	1399	1683	45611
S15850	1920	5520	84449	1820	2183	51747

Table 5.4: A Comparison between the quality of the best solutions obtained from GA by performing SOP for delay-only and MOP.

Circuit	For Delay-only			For all objectives		
	Delay (ps)	Cutset (nets)	power (S.P.)	Delay (ps)	Cutset (nets)	power (S.P.)
S298	189	66	1991	197	24	926
S386	345	65	2568	386	30	1426
S641	872	181	5521	889	59	2281
S832	399	126	5316	446	50	2731
S953	402	172	4615	466	99	2518
S1196	300	106	4919	301	106	4920
S1238	407	79	4596	408	79	4597
S1488	524	286	9912	528	98	5529
S1494	581	275	9750	585	101	5339
S2081	232	61	1792	225	17	770
S3330	654	935	21854	533	295	10298
S5378	590	1682	41845	590	430	16527
S9234	1044	3162	85944	1052	918	34055
S13207	964	4565	92975	843	1332	41114
S15850	1675	5565	90029	1411	1671	47480

Table 5.5: A Comparison between the quality of the best solutions obtained from TS by performing SOP for delay-only and MOP.

which lie on the critical paths; all other nets are totally ignored. As a consequence, the overall cutset and power consumption are not optimized at all. Observing the results in Table 5.4 it can be noticed that for the first ten circuits the delay-only optimization performed better in delay measure when compared with multiobjective. However, when the size of the circuit gets bigger (as in the case of the last five circuits) the multiobjective optimization gives better results. This is because delay-only optimization considers only a small part of the circuit, which will improve the delay at a certain cutset and power value (may be high or low). On the other hand multiobjective optimization decreases also the cutset and the power value, opening a new search space where more optimal delay solutions can be found. This effect is mainly noticed for the bigger and more complex circuits, where the cells that affect the delay comprise a relatively smaller set than the entire circuit. This effect is also noted for TS in Table 5.5 where circuits s3330, s13207 and S15850 have worse delay than multiobjective TS. Comparing the results for TS and GA, we can notice that TS outperformed GA in the case of delay-only optimization.

5.3.3 Cut-only Optimization versus MOP

Comparison between cut-only optimization versus MOP is shown in Table 5.6 and Table 5.7 for GA and TS respectively.

Table 5.6 lists the values of all three objectives in the solutions obtained from GA in both the case of MOP and cut-only optimization. Similarly, Table 5.7 shows the

Circuit	For Cut-only			For all objectives		
	Delay (ps)	Cutset (nets)	power (S.P.)	Delay (ps)	Cutset (nets)	power (S.P.)
S298	284	18	1056	233	19	1013
S386	418	29	1755	356	36	1529
S641	1015	44	2540	1043	45	2355
S832	445	44	3286	444	45	3034
S953	580	93	3196	526	96	2916
S1196	431	107	5859	396	123	5443
S1238	646	116	6369	475	127	5713
S1488	708	99	6271	571	104	5648
S1494	701	82	5743	614	102	5474
S2081	449	16	449	302	26	787
S3330	703	277	11349	571	299	10358
S5378	594	488	20531	587	573	18437
S9234	1557	960	41284	1313	1090	38149
S13207	1572	1538	50364	1399	1683	45611
S15850	1938	2001	54800	1820	2183	51747

Table 5.6: A Comparison between the quality of the best solutions obtained from GA by performing SOP for cut-only and MOP.

Circuit	For Cut-only			For all objectives		
	Delay (ps)	Cutset (nets)	power (S.P.)	Delay (ps)	Cutset (nets)	power (S.P.)
S298	233	13	905	197	24	926
S386	328	18	1453	386	30	1426
S641	1286	51	2853	889	59	2281
S832	453	35	3097	446	50	2731
S953	543	76	3055	466	99	2518
S1196	541	79	5170	301	106	4920
S1238	382	66	5041	408	79	4597
S1488	670	92	6251	528	98	5529
S1494	693	94	5959	585	101	5339
S2081	266	5	592	225	17	770
S3330	670	252	11309	533	295	10298
S5378	773	396	18502	590	430	16527
S9234	1365	899	39330	1052	918	34055
S13207	1135	1240	46789	843	1332	41114
S15850	1965	1545	50629	1411	1671	47480

Table 5.7: A Comparison between the quality of the best solutions obtained from TS by performing SOP for cut-only and MOP.

same comparison for the solutions obtained from TS. It is observed in most of the circuits that cut-only optimization approach results in slightly smaller cutset size but not without an increase in values of other two objectives. The power consumption objective is less affected because it is related to the cutset i.e., if the nets with high switching probability are not cut then the circuit consumes less power. The delay on the other hand may not be minimized when minimizing cutsize because the value of delay depends only on those cells that are on the longest path in the circuit, and these are comprise a small subset of the circuit.

5.4 GA versus TS

The results obtained from GA and TS for MOP are compared in terms of the overall quality of best solution and runtime in Table 5.8. In this table, P represents an estimate of the cost due to power, that is, the sum of the switching probabilities of all the cut nets. It has no unit since it is a probability. D is the delay of the most critical path in pico seconds (ps), $\mu(x)$ is the membership value, T(s) is the total run time in seconds, and $T_{best(s)}$ is the execution time in seconds for reaching the best solution. In case of TS, 10,000 iterations are run, and. for GA the stopping criterion is 10,000 generations.

The results shown are the best case results obtained by best possible tuning of various algorithmic parameters for GA and TS. In case of GA, population size is 10,

	GA						TS					
Circuit	D (ps)	Cut	P (sp)	$\mu(x)$	T (s)	$T_{best(s)}$	D (ps)	Cut	P (sp)	$\mu(x)$	T (s)	$T_{best(s)}$
S298	233	19	1013	0.79	123	43	197	24	926	0.81	62	21
S386	356	36	1529	0.75	163	151	366	30	1426	0.76	82	77
S641	1043	45	2355	0.83	1868	1540	889	59	2281	0.85	939	818
S832	444	45	3034	0.68	289	276	446	50	2731	0.682	148	80
S953	526	96	2916	0.69	618	182	466	99	2518	0.734	313	225
S1196	396	123	5443	0.76	375	373	301	106	4920	0.801	184	134
S1238	475	127	5713	0.72	397	365	408	79	4597	0.75	187	160
S1488	571	104	5648	0.71	1238	1183	528	98	5529	0.72	616	405
S1494	614	102	5474	0.70	1228	1040	585	101	5339	0.71	616	427
S2081	302	26	787	0.73	94	32	225	17	770	0.79	47	16
S3330	571	299	10358	0.75	2096	2074	533	295	10298	0.79	1078	994
S5378	587	573	18437	0.74	2687	2686	590	430	16527	0.79	1338	1100
S9234	1313	1090	38149	0.72	5963	5949	1052	918	34055	0.81	2992	2821
S13207	1399	1683	45611	0.74	8098	8097	843	1332	41114	0.79	4001	3690
S15850	1820	2183	51747	0.74	10214	10206	1411	1671	47480	0.831	5131	5130

Table 5.8: Comparison between costs of the best solutions generated by GA and TS.

crossover used is simple with a probability equal to 0.99. In case of TS, the size of neighborhood is also 10, while Tabu list size is chosen to be 0.1 times the size of the circuit. From the results, it is clear that TS performed better than GA for most of the circuits in terms of quality of best solution as well as run time.

For small circuits like S2081, S298, and S386, the quality of final solutions obtained from GA is comparable with that obtained from TS. However, as the size of circuit increases (and hence size of search space increases), TS consistently performs better than GA. Also, the execution time of GA increases significantly with the increase in circuit complexity. The higher execution time of GA can be attributed to its parallel nature i.e., a population of solutions is to be processed in each generation. Fig. 5.2 and Fig. 5.3 show the trend of solution's cutset, delay, power, balance, average fitness, and best fitness for GA and TS respectively, in case of circuit s13207. It is clear from these plots that TS achieves a better membership that is better than

that reached by GA. The best solution for GA is found after 8097 seconds and has 0.74 fitness value, while for TS, it was found after 3690 seconds and has a higher fitness value of 0.79.

Fig. 5.1 shows the performance of TS and GA against execution time in seconds. It is clearly noticed that TS is by far faster and solutions are of better quality.

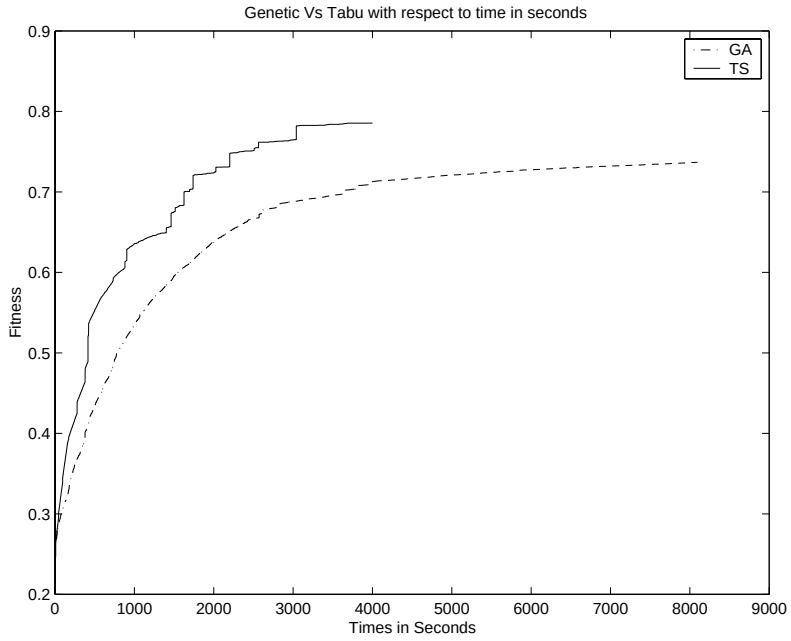

Figure 5.1: Comparison between GA and TS for the circuit S13207 with respect to execution time in seconds.

5.5 Simulated Evolution SimE and PowerFM

The results obtained from SimE when used in the case of MOP are given in Table 5.9. When we compare those results to the ones in Table 5.8 it can be noticed that the SimE outperformed TS for all the circuits except S386, S1238 and S1494. This

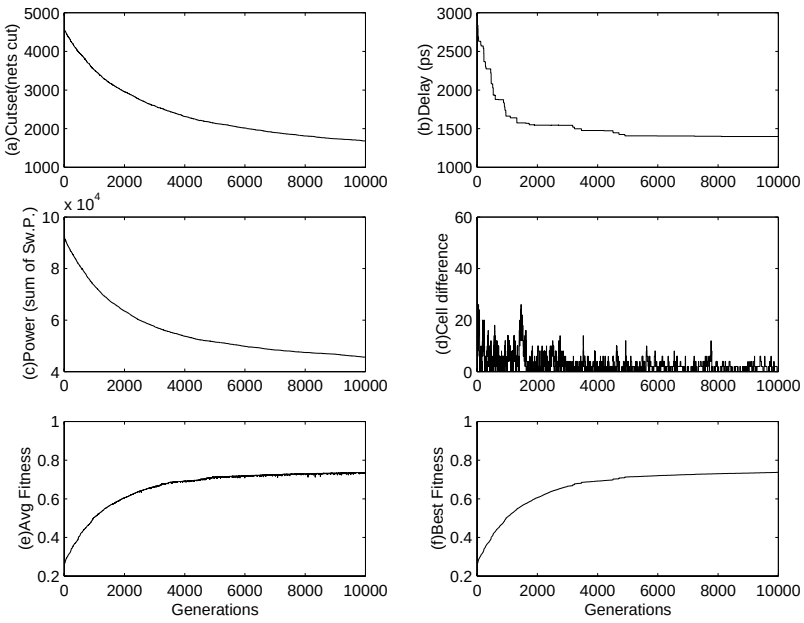

Figure 5.2: Performance of GA for the circuit S13207.

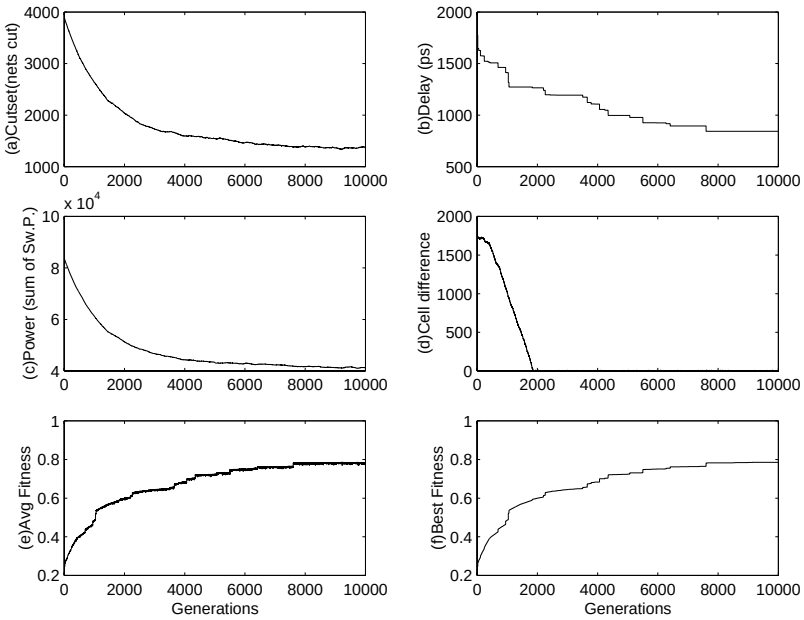

Figure 5.3: Performance of TS for the circuit S13207.

results is a good justification of the use of SimE as an efficient iterative heuristic in the VLSI partitioning problem. And proves the ability of SimE to explore the search space in a more indepth way due to its micro operations (evaluation and selection) on an individual solution.

The results obtained from the PowerFM algorithm are presented in Table 5.10 with a comparison to the results of SimE algorithm for power-only optimization. PowerFM is also compared to SimE in Table 5.11 for the case of multiobjective optimization. The notation in Tables (5.10, 5.11) is as follows: P_{avg} refers to the average power of the results obtained from 100 runs of the PowerFM, $D(ps)$ stands for Delay and is measured in *pico-seconds*, Cut is the number of nets cut, $P(sp)$ is the power dissipation measured in terms of switching probability, $T(s)$ is the total time taken by the whole run, and $T_{best(s)}$ is the time that the algorithm needs to find the best solution. It is noticed that the PowerFM is better in optimizing power for smaller circuits (S386, S832, S1196, S1238, S1494, and S2081) but performs poorly for bigger ones (S3330, S5378, S9234, S13207, and S15850) in terms of the best solution obtained in both Tables (5.10, 5.11). The overall average P_{avg} of the PowerFM is not as good as the results obtained from SimE . However, the execution time of the PowerFM is many orders of magnitude smaller than SimE and varies from 25 to 2000 times faster than SimE. This is due to the fact that the PowerFM is based on passes and it usually finishes after executing several passes where on each pass all the cells are considered for movement. The algorithm stops as soon as it

	Multiobjective Simulated Evolution SimE					
Circuit	D (ps)	Cut	P (sp)	$\mu(x)$	T(s)	$T_{best(s)}$
S298	197	11	837	0.95	102	62
S386	393	28	1696	0.74	156	152
S641	886	16	1738	0.98	1390	966
S832	400	39	3132	0.691	274	257
S953	476	48	2473	0.93	528	249
S1196	415	78	5488	0.82	463	398
S1238	350	77	5960	0.73	417	205
S1488	612	83	5892	0.70	1082	716
S1494	502	71	6250	0.81	1017	802
S2081	325	13	706	0.94	93	89
S3330	394	46	8431	0.98	1328	812
S5378	554	161	14094	0.95	2117	465
S9234	831	196	25672	0.98	4733	3853
S13207	1014	313	35014	0.98	6295	3129
S15850	1188	416	40716	0.96	7978	1850

Table 5.9: Performance of the Multiobjective SimE.

reaches a local minima.

Fig. 5.4 shows the performance of the SimE on the circuit S12307 which gives far better results than GA and TS and better results than PowerFM in terms of the quality of solution. Fig. 5.5 shows the performance of SimE versus TS and GA with respect to time. It can be seen clearly that SimE outperforms both algorithm, in quality and runtime.

5.6 Starting from PowerFM as initial solution to GA and TS.

The idea of using PowerFM as a starting solution for other iterative algorithms is relevant as it proved to be a very fast algorithm with reasonable performance. This

	Power-only Simulated Evolution SimE						PowerFM				
Circuit	D (ps)	Cut	P (sp)	$\mu(x)$	T (s)	$T_{best(s)}$	D (ps)	Cut	P(sp)	T (s)	P_{avg}
S298	301	21	738	0.92	97	26	301	20	732	0.05	828
S386	449	37	1567	0.73	156	152	434	29	1511	0.39	1673
S641	1133	38	1704	0.95	1416	719	1221	44	1667	0.61	1773
S832	527	68	3116	0.601	213	162	527	51	2855	1.97	3338
S953	1120	134	2369	0.643	424	398	902	120	2191	0.60	2422
S1196	598	109	5206	0.71	361	343	612	68	4116	1.81	5289
S1238	658	131	5928	0.60	330	316	544	62	4281	1.80	5358
S1488	655	105	5686	0.65	1009	879	724	70	5228	5.60	5787
S1494	738	125	6201	0.68	800	433	630	80	5354	7.19	6022
S2081	386	15	583	0.94	73	58	335	7	565	0.11	586
S3330	552	228	9354	0.89	1685	1008	593	226	9522	6.37	10180
S5378	738	299	13688	0.91	1582	1142	574	363	14565	19.22	15453
S9234	898	209	25565	0.96	3672	1976	832	389	26784	92.50	29100
S13207	1099	690	34921	0.94	7150	5365	1286	929	37190	273	39155
S15850	1458	688	40686	0.91	8122	5732	1464	919	42521	318.56	43238

Table 5.10: Comparison between SimE and PowerFM for power-only optimization.

	Multiobjective Simulated Evolution SimE					PowerFM				
Circuit	D (ps)	Cut	P (sp)	T (s)	$T_{best(s)}$	D (ps)	Cut	P (sp)	T (s)	P_{avg}
S298	197	11	837	102	62	301	20	732	0.05	828
S386	393	28	1696	156	152	434	29	1511	0.39	1673
S641	886	16	1738	1390	966	1221	44	1667	0.61	1773
S832	500	45	3232	274	257	527	51	2855	1.97	3338
S953	476	48	2473	528	249	902	120	2191	0.60	2422
S1196	415	78	5488	463	398	612	68	4116	1.81	5289
S1238	350	77	5960	417	205	544	62	4218	1.80	5358
S1488	612	83	5892	1082	716	724	70	5228	5.60	5787
S1494	502	71	6250	1017	802	630	80	5354	7.19	6022
S2081	325	13	706	93	89	335	7	565	0.11	586
S3330	394	46	8431	1662	1086	593	226	9522	6.37	10180
S5378	554	161	14094	2117	465	574	363	14565	19.22	15453
S9234	831	196	25672	4733	3853	832	389	26784	92.50	29100
S13207	1014	313	35014	6295	3129	1286	929	37190	273	39155
S15850	1189	416	40716	7978	1850	1464	919	42521	318.56	43238

Table 5.11: Comparison between Multiobjective SimE and PowerFM.

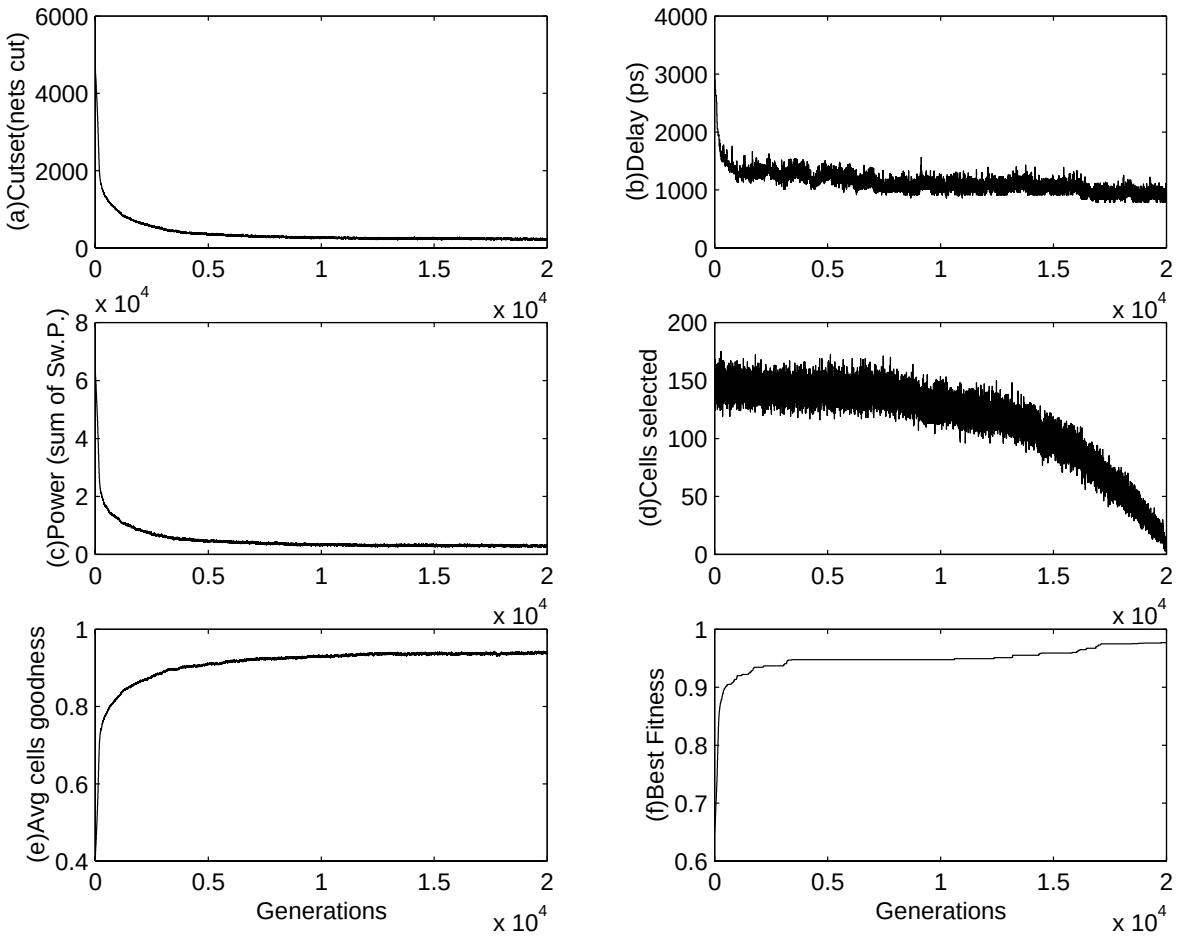

Figure 5.4: Multiobjective SimE performance for the circuit S13207.

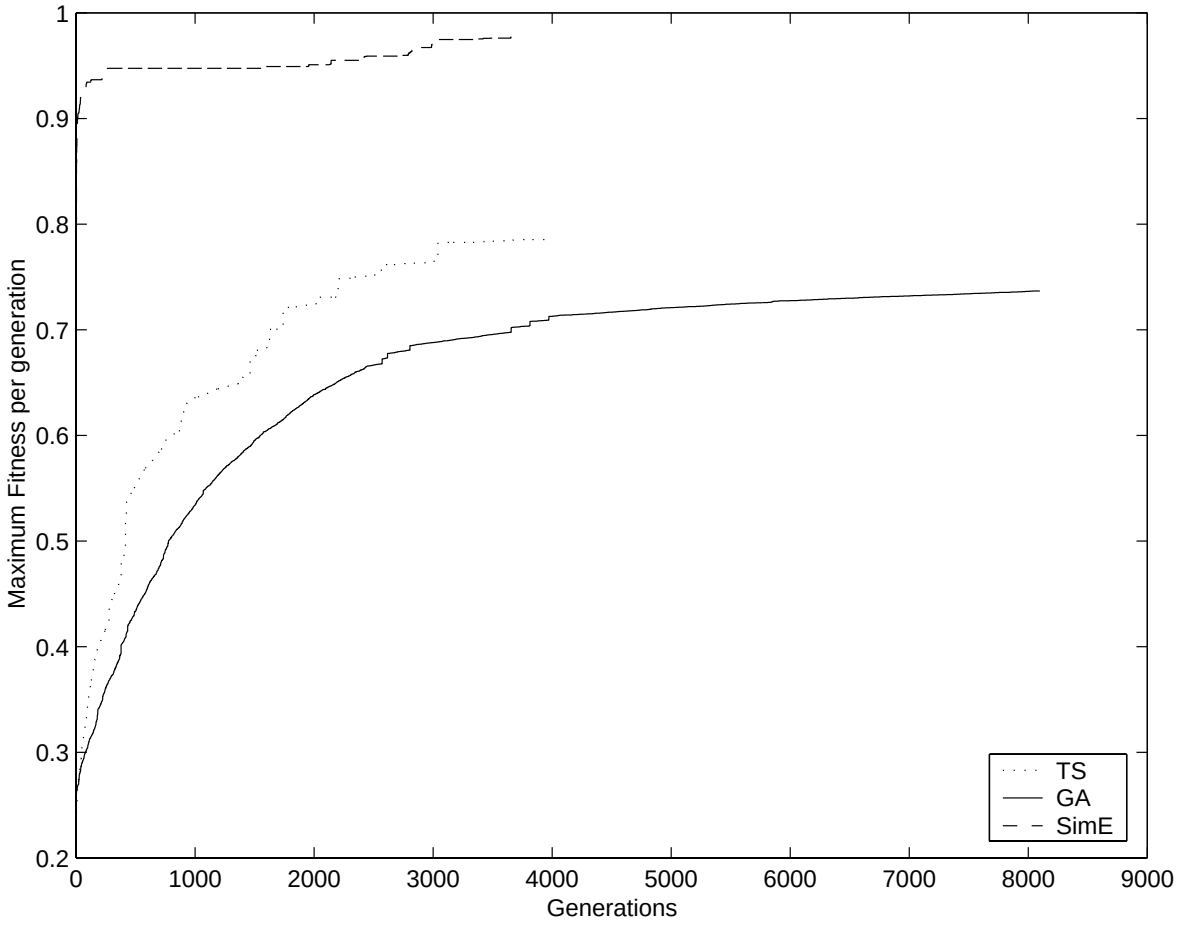

Figure 5.5: Multiobjective SimE versus GA versus TS performance for the circuit S13207 against time.

	TS Random Start			TS Start From PowerFM			Power-only SimE		
Circuit	D (ps)	Cut	P (sp)	D (ps)	Cut	P (sp)	D (ps)	Cut	P (sp)
S298	197	24	926	189	10	849	301	21	738
S386	386	30	1426	333	27	1264	449	37	1567
S641	889	59	2281	844	48	2476	1133	38	1704
S832	446	50	2731	431	40	3135	527	68	3116
S953	466	99	2518	430	85	2999	1120	134	2369
S1196	301	106	4920	335	77	4823	598	109	5206
S1238	408	79	4597	401	74	5190	658	131	5928
S1488	528	98	5529	521	94	6005	655	105	5686
S1494	585	101	5339	534	95	5058	738	125	6201
S2081	225	17	770	244	12	704	386	15	583
S3330	533	295	10298	419	257	9288	552	228	9354
S5378	590	430	16527	432	400	15319	738	299	13688
S9234	1052	918	34055	835	705	31837	898	209	25565
S13207	843	1332	41114	823	1310	40235	1099	690	34921
S15850	1411	1671	47480	1210	1332	45320	1458	688	40686

Table 5.12: Start from PowerFM versus TS and SimE.

will save a lot of time for algorithms like GA and TS. The results showed that GA and TS were able to improve solution provided by PowerFM.

Fig. 5.6 shows the performance of GA on the circuit $S1488$ when starting with a solution provided by PowerFM. Fig. 5.7 shows the performance of TS on the circuit $S1488$ when starting with the same solution. For this circuit GA was able to get delay, power, cutset values better than SimE and PowerFM, with considerably shorter time (compared with the random start GA runtime). For this particular circuit TS results were comparable to SimE but not better.

Table 5.12 shows that for Small circuits (S386, S832, S1196, S1238, S1494, and S2081) the T(s) from PowerFM results are better than SimE. For larger circuits (S3330, S5378, S9234, S13207, and S15850) we have comparable results but still the SimE is better in terms of cutset and power.

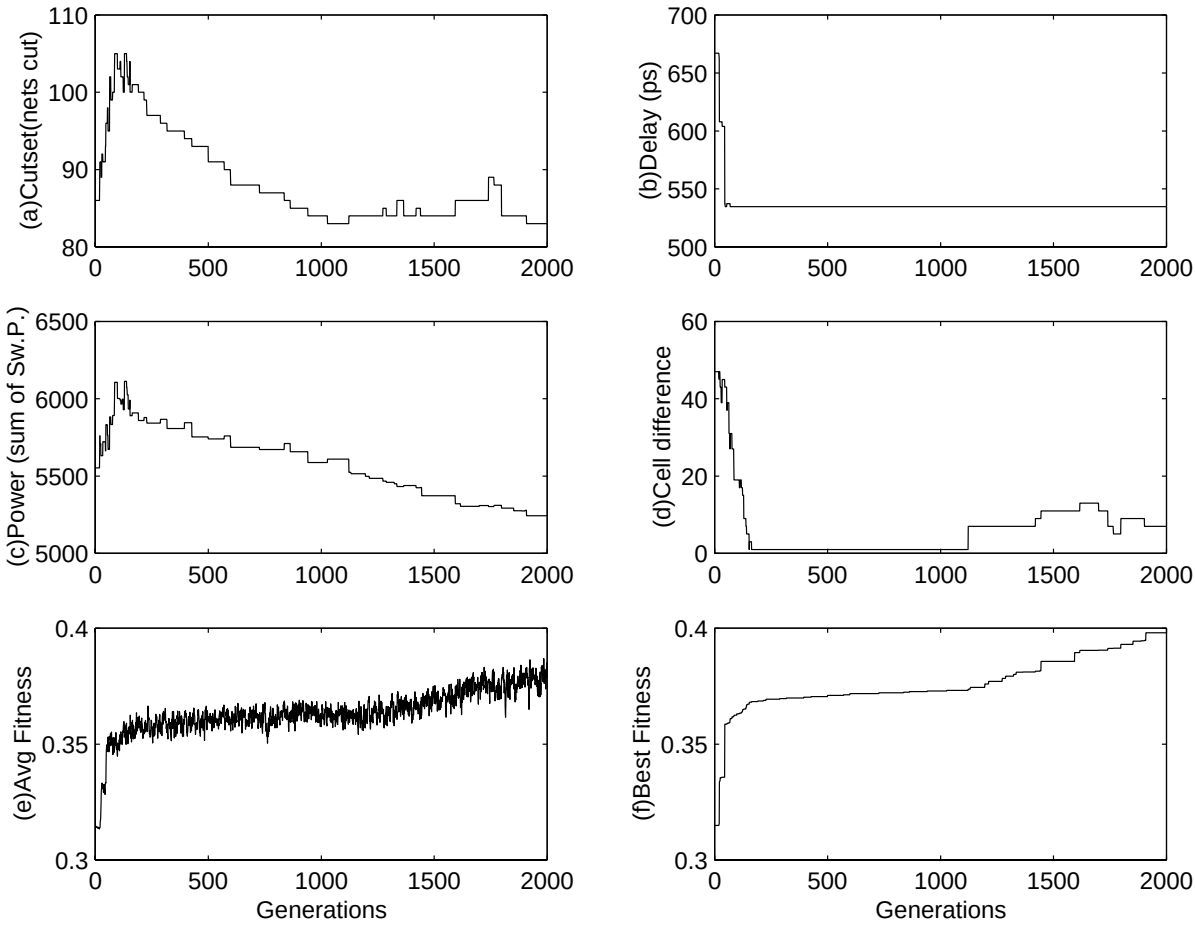

Figure 5.6: Genetic Algorithm starting from PowerFM for circuit S1488.

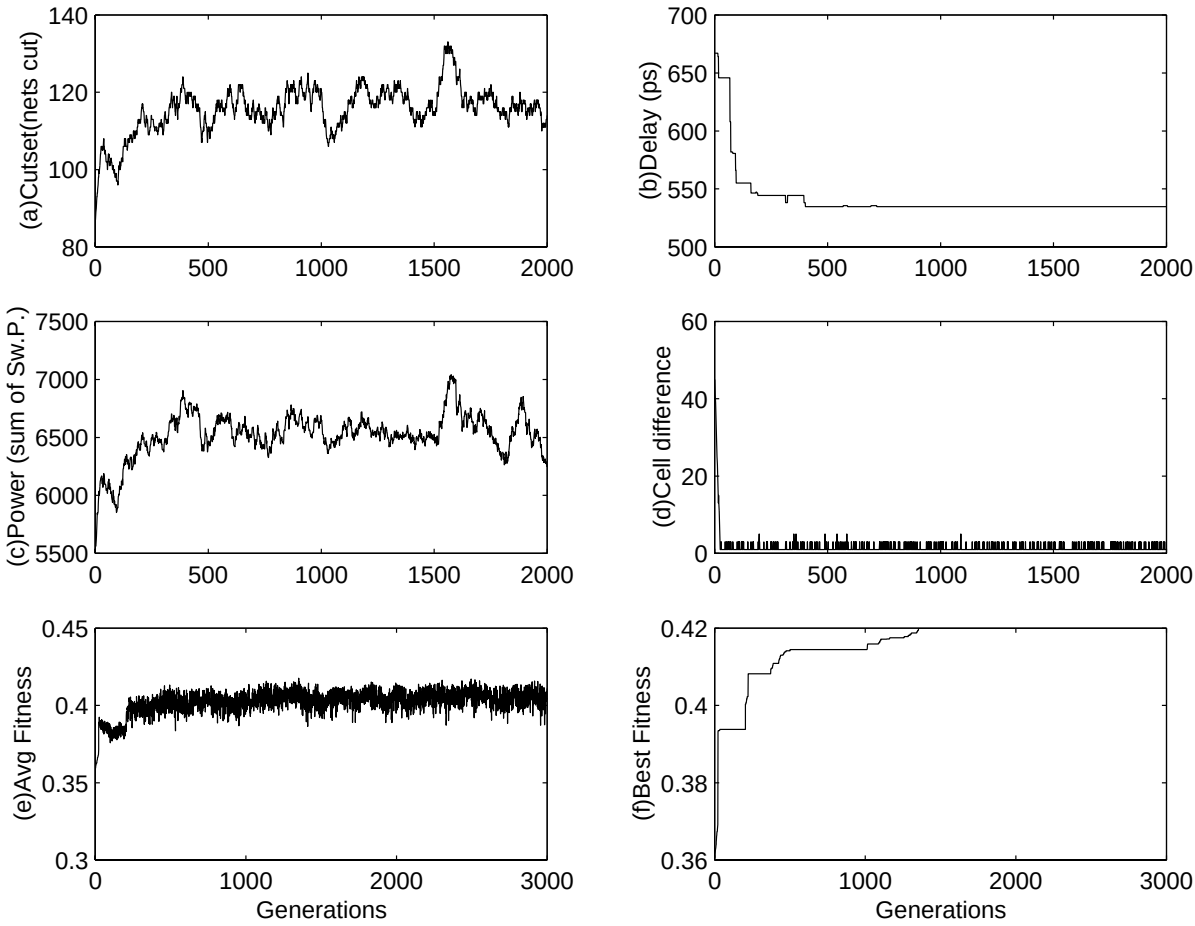

Figure 5.7: Tabu Search algorithm starting from PowerFM for circuit S1488.

Chapter 6

Conclusions and Future Directions

In this thesis, the problem of timing and low power driven VLSI standard cell placement is addressed. This is formulated as a multiobjective optimization problem (MOP) and the use of fuzzy rules is proposed for designing aggregate cost function. Two iterative algorithms, namely, GA and TS are presented for solving this hard problem. In addition, a SimE algorithm was proposed along with a new variation of the Fiduccia-Mattheyeses specifically to optimize power called PowerFM. The results of the above techniques are promising and show that these are well engineered for the problem.

Here are some concluding statements regarding the thesis work:

- The present work successfully addressed the important issue of reducing power consumption in VLSI circuits.

- TS outperformed GA.
- SimE results were better than TS and GA, with faster execution time.
- PowerFM results are comparable to SimE but with a faster runtime.
- PowerFM can be used to provide other algorithms a good starting solution.
- Iterative algorithms provide powerful solutions for solving hard problems with large search space.

Future Directions

Some possible directions of the future work can be following:

- Investigation of some other iterative heuristics for the presented problem.
- Some other hybrid techniques can be proposed and experimented for further improvement of the results.
- The presented standard cell partitioning techniques can be extended to perform the optimization of other steps of VLSI physical design.

Bibliography

[1] Semiconductor Industry Association. *National Technology Roadmap for Semiconductor*, 1997.

[2] Sadiq M. Sait and Habib Youssef. VLSI Physical Design Automation: Theory and Practice. *McGraw-Hill Book Company, Europe*, 1995.

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

[4] J. Cong. Challenges and Opportunities for Design Innovation in Nanomete Technologies. *SRC Design Sciences Concept Paper*, 1997.

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

[6] Massoud Pedram. CAD for Low Power: Status and Promising Directions. *IEEE International Symposium on VLSI Technology, Systems and Applications*, pages

331–336, 1995.

[7] Unni Narayanan, G.I. Stamoulis, and Rabindra Roy. Characterizing Individual Gate Power Sensitivity in Low Power Design. *12th International Conference on VLSI Design*, pages 625–628, January 1999.

[8] H. B. Bakoglu. *Circuits, Interconnections, and Packaging for VLSI.* Addison-Wesley Pub. Co., 1990.

[9] J. Cong. An Interconnect-Centric Design Flow for Nanometer Technologies. *Proc. of the IEEE*, 89(4):505–528, April 2001.

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

[11] M.R. Garey and D.S. Johnson. Computers and Intractability. *Freeman, San Francisco CA*, 1979.

[12] B. W. Kernighan and S. Lin. An Efficient Heuristic Procedure for Partitioning Graphs. *The Bell System Technical Journal*, 29(2):291–307, 1970.

[13] Charles M. Fiduccia and R. M. Mattheyses. A Linear-Time Heuristic for Improving Network Partitions. *Proc. of the 19th IEEE Design Automation Conference*, pages 175–181, 1982.

[14] H. Vaishnav and M. Pedram. Delay Optimal Partitioning Targeting Low Power VLSI Circuits. *IEEE Trans. on Computer Aided Design*, 18(6):298–301, June 1999.

[15] M. Holzrichter and S. Oliveira. New Graph Partitioning Algorithms. *The University of Iowa TR-120.*, 1998.

[16] L. Hagen and A. Kahng. Combining Problem Reduction and Adaptive Multistart: A new technique for Superior Iterative Partitioning. *IEEE Trans. on CAD*, 16(7):709–717, 1997.

[17] S. Areibi and A. Vannelli. A Combined Eigenvector Tabu Search Approach For Circuit Partitioning. *Proc. of the 1993 Custom Integrated Circuits Conference (San Diego)*, pages 9.7.1 – 9.7.4., 1993.

[18] Slawomir Koziel and Wladyslaw Szczesniak. Evolutionary Algorithm for Electronic Systems Partitioning and its Applications in VLSI Design. *IEEE Computing*, pages 1411–1414, 1999.

[19] Shantanu Dutt and Wenyong Deng. A Probabilistic Approach to VLSI Circuit Partitioning. *Proc. Design Automation*, pages 100–105, June 1996.

[20] Charles J. Alpert and So-Zen Yao. Spectral Partitioning: The More Eigenvectors, the Better. *Design Automation Conference*, pages 195–200, 1995.

[21] L. Hagen and A. Kahng. New Spectral Methods for Ratio Cut Partitioning And Clustering. *IEEE Trans. CAD*, 11(9):1074–1085, September 1992.

[22] S. Barnardand and H. Simon. A Fast Multilevel Implementation of Recursive Spectral Bisection for Partitioning Unstructured Problems. *Concurrency: Practice & Experience*, 6(2):101–117, 1994.

[23] S. Areibi and A. Vannelli. Advanced Search Techniques for Circuit Partitioning, in Quadratic Assignment and Related Problems, DIMACS Series in Discrete Mathematics and Theoretical Computer Science edited by P. Pardalos and H. Wolkowicz. 16:77–99, 1994.

[24] H. Shin and C. Kim. A Simple Yet Effective Technique for Partitioning. *IEEE Transaction on VLSI*, pages 380–386, September 1993.

[25] L. Hagen and A. Kahng. Combining Problem Reduction and Adaptive Multistart: A New Technique for Superior Iterative Partitioning. *IEEE Trans. on CAD*, 16(7):709–717., 1997.

[26] L. A. Sanchis. Multiple-Way Network Partitioning. *IEEE Trans. on Computers, IEEE Computer Society, Washington D.C.*, 38(1):62–81, 1989.

[27] Kirkpatrick, C.D. Gelatt and Vecchi M.P. Optimization by Simulated Annealing. *Science*, 220(4598):671–680, May 1983.

[28] Fred Glover. Tabu Search– Part I. *ORSA Journal on Computing*, 1(3):190–206, 1989.

[29] C.J. Alpert and A.B. Kahng. A Hybrid Multilevel/Genetic Approach for Circuit Partitioning. *Physical Design Workshop*, pages 100–105, 1996.

[30] E. Lawler, K. Levitt, and J. Turne. Module Clustering to Minimize Delay in Digital Networks. *IEEE Trans. on Computer-Aided Design*, 47–57, 1969.

[31] J. De Gruijter and A. B. McBratney. A Modified Fuzzy K-Means Method for Predictive Classification. *Proc. of the First Conference of the International Federation of Classification Societies (IFCS)*, 1988.

[32] C. Ball, P. Kraus, and D. Mlynski. Fuzzy Partitioning Applied to VLSI-Floorplanning and Placement. *Proc. IEEE Intl. Symp. Circuits and Systems*, pages 177–180, 1994.

[33] D. S. Johnson. Hierarchical Clustering Schemes. *Psychometrika*, 32(3):241–254, 1967.

[34] S. Dey, F. Brglez, and G. Kedem. Corolla Based Circuit Partitioning and Resynthesis. *ACM/IEEE Design Automation Conf.*, pages 607–612, 1990.

[35] S. Dey, F. Brglez, and G. Kedem. Partitioning Sequential Circuits for Logic Optimization. *IEEE Intl. Conf. Computer Design*, pages 70–76, 1990.

[36] S. Hauck and G. Borriello. An Evaluation of Bipartitioning Techniques. *IEEE Trans. on Computer-Aided Design of Integrated Circuits and Systems*, 16(8):849–866, August 1997.

[37] A. Chandrakasan, T. Sheng and R. W. Brodereson. Low Power CMOS Digital Design. *Journal of Solid State Circuits*, 27(4):473–484, April 1992.

[38] Srinivas Devadas and Sharad Malik. A Survey of Optimization Techniques Targeting Low Power VLSI Circuits. *32nd ACM/IEEE Design Automation Conference*, pages 242–247, 1995.

[39] I.S. Choi and S.Y. Hwang. Circuit Partitioning Algorithm for Low-Power Design Under Area Constraints Using Simulated Annealing. *IEE Proc. Cicrcuits Devices Systems*, 146(1):8–15, February 1999.

[40] Jun'ichiro Minami, Koide Tetsushi and Wakabayashi Shin'ichi. A Circuit Partiotioning Algorithm Under Path Delay Constraints. *IPSJ SIGNotes Design Automation*, 87(4):WT32–1.1 WT32–1.4, 1998.

[41] Wayne Wolf. *Modern VLSI Design.* PTR Prentice Hall Inc, Englewood Cliffs, New Jersey, 1994.

[42] J. Hwang and A. El Gamal. Optimal Replication for Min-Cut Partitioning. *Proc. of IEEE/ACM International Conference on Computer-Aided Design.*, pages 432–435, November 1992.

[43] Abrajit Ghosh, Srinivas Devadas, Kurt Keutzer, and Jacob White. Estimation of Average Switching Activity in Combinational and Sequential Circuits. *Design Automation Conference*, pages 253–259, 1992.

[44] G. Fandel and T. Gal, eds. Multiple Criteria Descision Making Theory and Applications. *Lecture Notes in Economics and Mathematics Systems. Berlin: Springer-Verlag*, 177:468–486, 1980.

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

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

[47] Sadiq M. Sait, Habib Youseff, and Hussain Ali. Fuzzy Simulated Evolution Algorithm for Multi-objective Optimization of VLSI Placement. *Congress on Evolutionary Computation, IEEE Service Center, Washington, D.C.*, pages 91–97, July 1999.

[48] 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.

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

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

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

[52] D. H. Ackley. A Connectionist Machine For Genetic Hillclimbing. *Kluwer Academic Press, Boston*, 1987.

[53] T. Bui and B. R. Moon. A Genetic Algorithm For A Special Class of the Quadratic Assignemnt Problem. *DIMACS Series in Discrete Mathematics and Theoretical Computer Science*, 16:99–116, 1994.

[54] S. Areibi. An Integrated Genetic Algorithm With Dynamic Hill Climbing for VLSI Circuit Partitioning. *In Genetic and Evolutionary Computation Conference (GECCO-2000) IEEE. Las Vegas, Nevada*, July 2000.

[55] J. P. Cohoon and W. D. Paris. Genetic Placement. *IEEE Trans. on CAD*, pages 956–964, 1987.

[56] L. Tao., Y. C. Zhao, K. Thulasiraman and M. N. S. Swamy. An Efficient Tabu Search Algorithm For Graph Bisectioning. *In Proc. First Great Lakes Symposium on VLSI in Kalamazoo, Michigan*, pages 92–95, 1991.

[57] A. Lim and Y. M. Chee. Graph Partitioning using Tabu Search. *In Proc IEEE Intl.symp. Circuits and Systems, Seattle, WA*, pages 1164–7, 1991.

[58] S. Areibi and A. Vannelli. Circuit Partitioning using Tabu Search Approach. *In Proc. IEEE Intl. Symp. Circuits and Systems*, pages 1643–1646, 1993.

[59] R. M. Kling. *Optimization by Simulated Evolution and its Application to Cell Placement.* PhD thesis, University of Illinois, Urbana, 1990.

[60] R. M. Kling and P. Banerjee. ESP: Placement by Simulated Evolution. *IEEE Transaction on Computer-Aided Design*, 3(8):245–255, March 1989.

[61] Y. Saab and V. Rao. Fast Effective Heuristics for the Graph Bisectioning Problem. *IEEE trans. Computer-Aided Design*, 9(1):91–98, January 1990.

[62] Junaid A. Khan, Sadiq M. Sait, and Mahmood R. Minhas. Fuzzy Biasless Simulated Evolution for Multiobjective VLSI Placement. *IEEE CEC 2002, Hawaii USA*, 12-17 May 2002.

[63] Tanner Consulting and Engineering Services. *Digital Low Power Standard Cell Library for MOSIS TSMC CMOS 0.25μ Process Deep Sub-Micron Technology.* Tener Research, Inc.

Vitae

- Raslan Hashim Al-Abaji
- Born in Beirut, Lebanon.
- Received Bachelor of Science in Electrical Engineering Computers Section Beirut Arab University, Beirut, Lebanon.
- Joined Computer Engineering Department, KFUPM, as a research assistant in January 2000.
- Received Master of Science degree in Computer Engineering from KFUPM, Dhahran, Saudi Arabia in 2002.