Genetic Algorithm for Timing Influenced Floorplanning of VLSI Designs

by

Mohammed Shahid Tanvir Khan

A Thesis Presented to the

FACULTY OF THE COLLEGE OF GRADUATE STUDIES

KING FAHD UNIVERSITY OF PETROLEUM & MINERALS

DHAHRAN, SAUDI ARABIA

In Partial Fulfillment of the
Requirements for the Degree of

MASTER OF SCIENCE

In

COMPUTER ENGINEERING

December, 1994

Genetic Algorithm for Timing Influenced Floorplanning of VLSI Designs

BY

Mohammed Shahid Tanvir Khan

A Thesis Presented to the
FACULTY OF THE COLLEGE OF GRADUATE STUDIES
KING FAHD UNIVERSITY OF PETROLEUM & MINERALS
DHAHRAN, SAUDI ARABIA

In Partial Fulfillment of the
Requirements for the Degree of

MASTER OF SCIENCE
In

Computer Engineering

December 1994

KING FAHD UNIVERSITY OF PETROLEUM AND MINERALS
DHAHRAN, SAUDI ARABIA
COLLEGE OF GRADUATE STUDIES

This thesis, written by

Mohammed Shahid Tanvir Khan

under the direction of his Thesis Advisor, and approved by his Thesis committee, has been presented to and accepted by the Dean, College 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. Habib Youssef (Co-Chairman)

Dr. Muhammed S. T. Benten (Member)

Dr. Samir H. Abdul-Jauwad
Department Chairman

Dr. Ala H. Rabeh
Dean, College of Graduate Studies

Date: 22/3/95

Genetic Algorithm for Timing Influenced Floorplanning of VLSI Designs

MS Thesis

Mohammed Shahid Tanvir Khan

December, 1994

Dedicated to

my

Father and Mother

whose prayers, guidance and inspiration

led to this accomplishment

Acknowledgment

In the name of Allah, the Most Gracious, the Most Merciful. Read: in the name of your Lord and Cherisher, Who created: created man from {*blood clot* }. Read: for your Lord is the Most Bountiful. He, Who taught {the use of} the pen. Taught man that which he knew not. Nay, but man doth transgress all bounds. In that he looketh upon himself as self-sufficient. Verily, to thy Lord is the return {of all}.

(The Holy Quran, Surah 96)

First and foremost, all praise to Allah, *subhanahu-wa-ta'ala*, the Most Forgiving and the Most Merciful. There is no strength to resist evil nor any power to do good except through His grace - the Mighty, the Wise. I depend entirely upon His mercy, in Him I believe and in Him I put my trust. He is sufficient for me in all of my affairs of this world and the world hereafter.

Peace and blessings of Allah be upon Prophet Muhammad, the last of the messengers, the mercy for mankind, the illuminating light and the intercessor on the Day of Judgement. Peace and blessings also be upon his noble family members, companions and all of those who follow him until the Day of Reckoning.

Acknowledgement is due to King Fahd University of Petroleum & Minerals for providing support to this research work.

I am grateful to my thesis chairman, Dr. Sadiq M. Sait for his help and advice. I acknowledge him for encouragement and guidance during the early stages of this work and my MS studies.

I am thankful to my thesis co-chairman. Dr. Habib Youssef for his deep interest. and constructive criticism during the course of this work. Thanks are due to the Dean of CCSE and my thesis committee member. Dr. Muhammed S. T. Benten for providing round the clock access to excellent computing facilities.

I am thankful to the department chairman. Dr. Samir Abdul-Jauwad and other faculty members for their cooperation.

I express my gratitude to brother Arif for his guidance and moral support during my stay at KFUPM. I am thankful to my fellow graduate students and all my friends especially Haroon. Asaf. Sami and Khalid who provided a wonderful company and made my stay at KFUPM a memorable experience.

Last but not the least. acknowledgement is due to the members of my family for

their emotional and moral support throughout my academic career. This thesis is dedicated to my parents who took great pains in order to fulfill my academic pursuits. They played a role model right from my childhood. They shaped my personality and taught me the lesson, "Tough times never last but tough people do", that made me strong enough to face the bitter realities of life. My eldest brother Sayeed deserve special mention for constant inspiration and cultivation of competitive spirit in me. I acknowledge him for his guidance and tremendous support during crucial days of my academic career. I am grateful to brother Mubashir for his moral support during the final stages of this work and brother Iqbal for his magnanimous support to my family during tough times. I acknowledge with gratitude, the affection and encouragement of my sisters (Farhat, Nuzhat) and brothers (Rashid, Javed) that helped me overcome homesickness and concentrate on my work. The real credit of this achievement goes to my family members which I acknowledge most sincerely.

Contents

4 GIFT Floorplanner 52

List of Tables

List of Figures

Abstract

Name: **Mohammed Shahid Tanvir Khan**

Title: **Genetic Algorithm for Timing Influenced Floorplanning of VLSI Designs**

Major Field: **Computer Engineering**

Date of Degree: **December, 1994**

In the wake of state of the art advancements in VLSI technology, speed performance of a circuit is greatly influenced by the interconnect delays. Therefore, it is vital to introduce timing performance as one of the criteria to judge the quality of floorplans. In this thesis, we describe a timing influenced genetic floorplanner for general-cell IC design, called "$\mathcal{GIFT}$". The floorplanner works in two phases. In the first phase, the modules are restricted to be rigid and floorplan to be slicing. The second phase allows modification to the aspect ratios of individual modules to further reduce the floorplan area. The solution quality is measured in terms of overall area, interconnection length and speed performance. Results of experiments on practical VLSI circuits are very good.

Master of Science Degree

King Fahd University of Petroleum and Minerals

Dhahran, Saudi Arabia

December 1994

خلاصة الرسالة

الاسم : محمدشاهدتنويرخان
عنوان الرسالة : خوارزمية جينية للتخطيط السطحي المتأثر بالوقت لتصاميم الدوائر المتكاملة الضخمة.
التخصص : هندسة الحاسب الالى
تاريخ الشهادة : ديسمبر١٩٩٤م

على اثر التقدم الحالي في تقنية الدوائر المتكاملة الضخمة ازداد مدى تأثر سرعة الدوائر بالتوصيلات الداخلية في تلك الدوائر.

نقدم في هذه الرسالة وصفا لخوارزمية جينية للتخطيط السطحى المتأثر بالوقت لتصميم دائرة متكاملة ذات خلايا عامة. ويعمل التخطيط السطحي على مرحلتين حيث يتطلب في المرحلة الاولى ان تكون الوحدات صلبة والمخطط السطحى على شكل شرائع. وتسمح المرحله الثانية بتعديل النسبة الباعية لكل وحدة على حدة لتقليل مساحة مخطط السطح.

ويتم قياسنوعية الحل عن طريق المساحة الكلية وطول التوصيلات الداخلية والسرعة وبدراسة بعض النتائج العملية على بعض الدوائر المتكاملة الضخمة تبين جودة الخوارزمية المقترحة.

درجة الماجستير فى العلوم
جامعة الملك فهد للبترول و المعادن
الظهران - المملكة العربية السعودية
ديسمبر١٩٩٤م

Chapter 1

Introduction

1.1 Overview

Very large scale integration (VLSI) is a design technology that allows the implementation of large circuits comprising millions of transistors on a single silicon chip. The design of complex integrated circuits (ICs) would not have been possible without the help of computers. Computer-aided-design (CAD) helps in automating the process of VLSI design. The automation achieved in IC design is attributed to extensive research work aimed at development of sophisticated and efficient algorithms. Designing an IC chip involves specifying the logical and physical characteristics of a large number of functional sub-systems. Each sub-system performs a specific function. For each sub-system, the designer must develop the associated circuitry which takes into consideration the characteristics of the particular technology and the physical realization of the circuitry for a specific manufacturing process. This design process must ensure that the chip performs the desired function while satisfying all constraints on performance, power, area and testability etc.

The process of IC design has been divided into various phases so as to reduce its complexity (Figure 4.8). The IC design process is composed of three closely related phases. The first phase involves the functional design, where the behavioral and performance requirements of the system are specified. In the second phase, the relation between various logical units is defined to realize the behavior described during the functional design phase. During the last phase called *physical design*, the design is mapped onto the semiconductor surface by determining the precise geometry and position of each constituent unit and the interconnection wires. The various physical design steps are partitioning, floorplanning, placement and routing. After physical design, the design data is converted into a format such as Caltech Intermediate Form (CIF) [1], and submitted to a foundry for the fabrication of the corresponding IC chip.

The physical layout of a circuit greatly influences area, speed, and reliability of the IC chip. For complex VLSI circuits, a structured design technique is used which allows hierarchical decomposition of a circuit. Floorplanning is an essential design step when a hierarchical/building block design method is used. In a hierarchical design, chip area is partitioned into several large blocks, and each block is recursively partitioned into smaller sub-blocks, until each block contains a single circuit module.

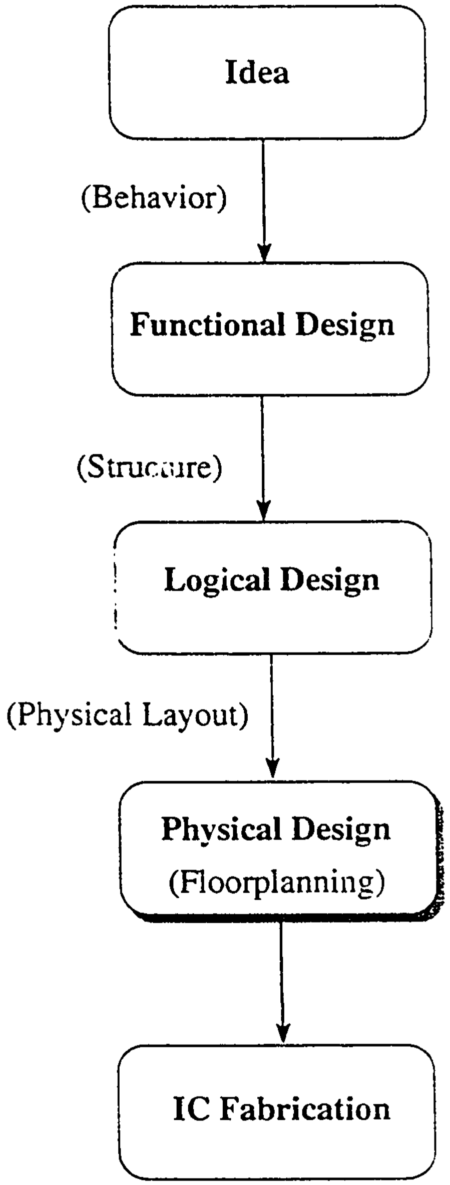


Figure 1.1: Flowchart illustrating phases of IC design process.

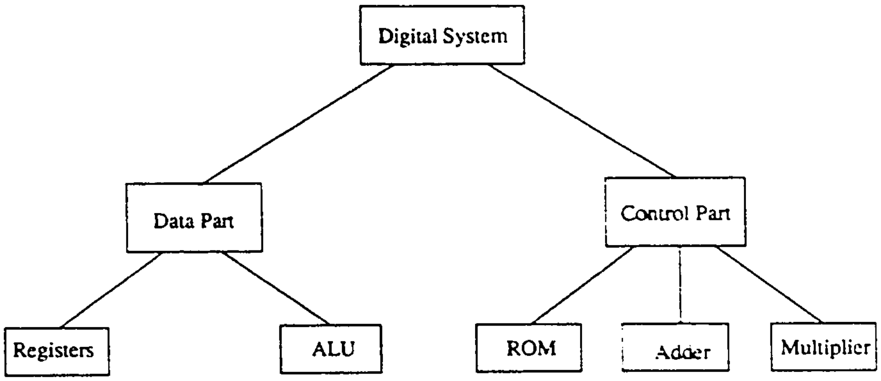


Figure 1.2: Hierarchical representation of a digital system.

The hierarchical representation of a digital system is shown in Figure 1.2 and its floorplan in Figure 1.3.

When a hierarchical building-block design methodology is used for physical design, the sequence of tasks that need to be performed from design specification to layout consists of the following steps:

- defining the layout hierarchy,
- estimating the overall required area,
- determining the aspect ratios for each module,
- assignment of pin and pad locations and finally,
- placement and routing.

Floorplanning is closely related to placement and can be seen as a feasibility study of the layout. Sometimes, it is referred to as topological layout [2]. During the pro-

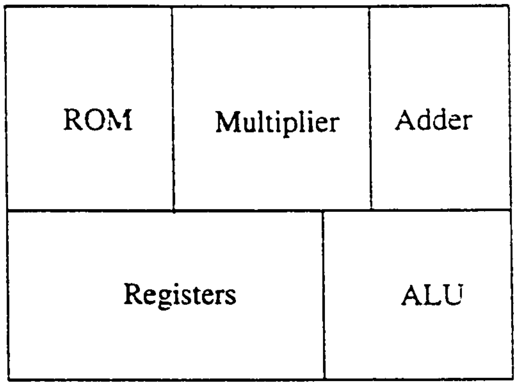


Figure 1.3: A possible floorplan of the circuit in Figure 1.2.

cess of placement, shape and pin positions on the periphery of circuit components are fixed, whereas in floorplanning these have some specified flexibility. The flexibility in the shape of the component represents the designer's freedom to select one among several possible implementations of the circuit components. The designer has additional freedom in terms of chip and component geometries (block orientations, shapes, and may be sizes).

This added flexibility must be captured by the floorplan model. The aspects that need to be modeled should include the components, the interconnection, the flexible interfaces (blocks and chip), the chip carrier (layout surface), any designer stated constraints, and the objectives to be optimized. In this thesis, we are concerned with the problem of floorplanning for general cell layout. In the next section, a formal definition of floorplanning is given.

1.2 Problem Definition

A formal description of the floorplanning problem adopted from the work reported in [3, 4] is as follows.

Given:

(1) a set S of n rectangular modules $S = \{1, 2, \cdots, i, \cdots, n\}$;

(2) S_1 and S_2, a partition of S, where S_1 and S_2 are the set of the modules with fixed and free orientation respectively;

(3) an interconnection matrix $C_{n \times n} = [c_{ij}]$, $1 \leq i, j \leq n$, where c_{ij} indicates the connectivity between modules i and j;

(4) a list of n triplets $(A_1, r_1, s_1), \cdots (A_i, r_i, s_i), \cdots (A_n, r_n, s_n)$, where A_i is the area of block i (i.e., $A_i = w_i \times h_i$, with width w_i and height h_i of block i). r_i and s_i are lower and upper bound constraints on the shape of block i ($r_i \neq s_i$ if the block is flexible, and $r_i = s_i$ if the block is rigid); and,

(5) two additional positive numbers p and q ($p \leq q$), which are lower and upper bound constraints on the shape of the rectangle enveloping the n blocks.

The required output is a feasible floorplan solution, i.e., an enveloping rectangle R subdivided by horizontal and vertical line segments into n nonoverlapping rectangles labeled $1, 2, \cdots, i, \cdots, n$, such that the following constraints are satisfied.

(1) $w_i \times h_i = A_i$, $1 \leq i \leq n$;

(2) $r_i \leq \frac{h_i}{w_i} \leq s_i$ for all modules i with fixed orientation (i is an element of S_1);

(3) $r_i \leq \frac{h_i}{w_i} \leq s_i$ or $\frac{1}{s_i} \leq \frac{h_i}{w_i} \leq \frac{1}{r_i}$ for all modules i with free orientation (i is an element of S_2);

(4) $x_i \geq w_i$ and $y_i \geq h_i$, $1 \leq i \leq n$, where x_i and y_i are the dimensions of basic rectangle i, (every rectangle i is large enough to accommodate module i);

(5) $p \leq \frac{H}{W} \leq q$, where H and W are the height and width of the enveloping rectangle R.

In addition to the above constraints, the objective(s) to be achieved is(are) expressed in the form of a cost function. The cost function usually, consists of one or several of the following objectives [2]:

- area of the bounding rectangle,
- overall wirelength,
- timing performance, and
- routability.

As the yield of IC production is directly related to the area of the IC chip [1], it is highly desirable to minimize the area of the floorplan bounding rectangle. The overall wirelength is a measure of the routing area which should be minimized. The speed performance of a circuit, however, is determined from the delay along its longest path. By minimizing the delay along the longest path, speed performance of the circuit can be improved. A solution is acceptable only when it is completely routable. Thus, routability should be maximized. Generally, a weight is assigned to each of the objectives and a weighted sum of two or several of these is considered as a cost function.

A feasible floorplan optimizing the desired cost function is an optimum floorplan. For example, if the cost function is the area of F, then an optimum feasible solution is a floorplan with the minimum area [2].

1.3 General Cell Floorplanner

In this thesis, a floorplanner has been implemented for general-cell VLSI layouts. The floorplanning problem is solved in two main steps. In the first step, we adopt a simpler floorplan model which has the following two restrictions: (a) Solutions are restricted to slicing floorplans, and (b) all blocks are rigid but can have free orientations. The cost function used is a weighted sum of overall area (functional + routing) and timing performance. The floorplan problem of the first step is solved using the genetic algorithm [5, 6]. The second step is a floorplan refinement phase where blocks are allowed to have flexible shapes, and the floorplan solutions are no longer restricted to slicing structures. The main aim of the second phase described

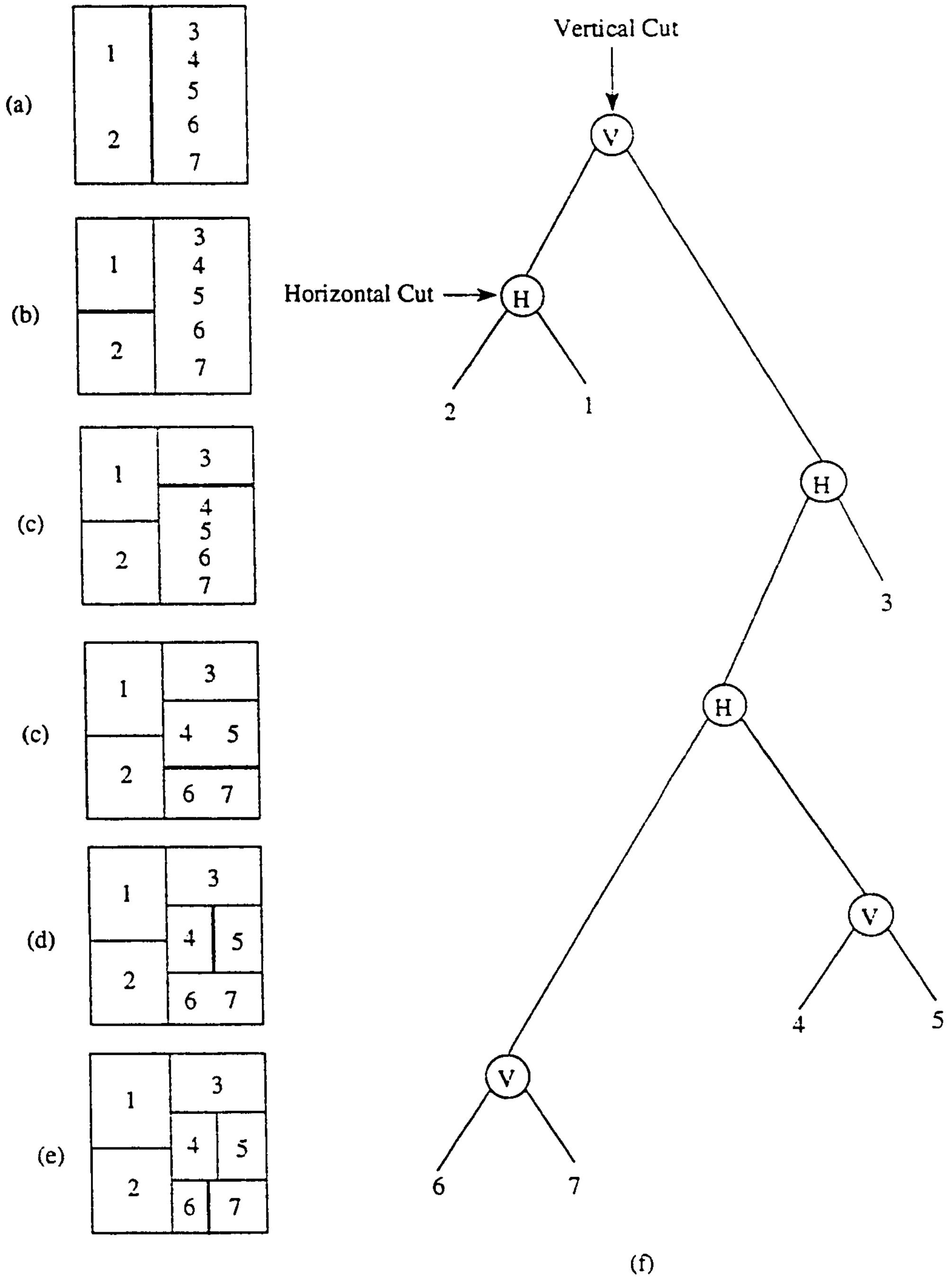


Figure 1.4: Obtaining a slicing floorplan through repeated dissection: In (a) to (e), each cut shown in bold line dissects the previous rectangle into two rectangles: (f) binary tree representation.

in [7], is to minimize the dead space by modifying the aspect ratios of the individual modules and compaction of the bounding box, thus, reducing the overall area of the bounding rectangle. In this step, the relative positions of the modules are kept intact and the non-overlapping feature of floorplan solution produced by the first step is maintained. The novelty of this work lies in the use of a simpler representation during the first step to solve a difficult problem, and using a simpler technique during the second step to refine the solution while easing the restrictions of the first step.

Genetic algorithm [5] has been employed as an optimization and search mechanism due to several reasons. Firstly, it is robust in that it consistently succeeds in locating a desirable solution from any random initial set of solutions. Secondly, it works on a set of solutions allowing a parallel search of the solution space. Finally, it is very convenient for problems with conflicting objectives such as floorplanning. However genetic algorithm is CPU time intensive and has a large memory requirement. This problem is alleviated by choosing a suitable solution encoding, and adopting a simple model of the floorplanning problem.

A floorplan with a slicing structure is called a *slicing* floorplan. Otherwise, it is called a *non-slicing* floorplan. For example, the floorplan in Figure 1.4(e) is a slicing floorplan, whereas those in Figure 1.5 are non-slicing floorplans. They are called *wheels.* A wheel is the smallest non-slicing floorplan. A slicing structure can be modeled by a binary tree called a *slicing tree.* The slicing tree consists of n

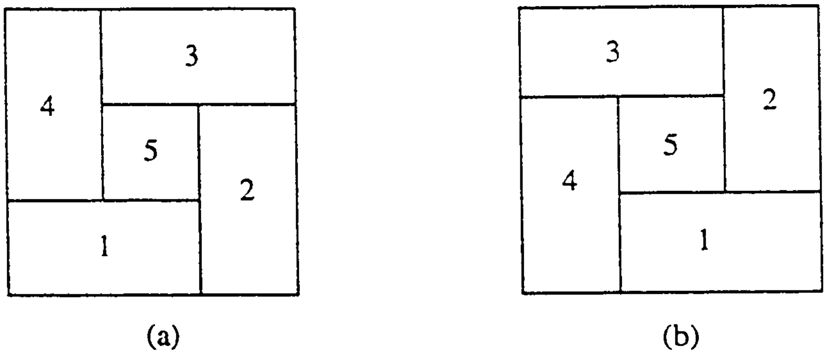
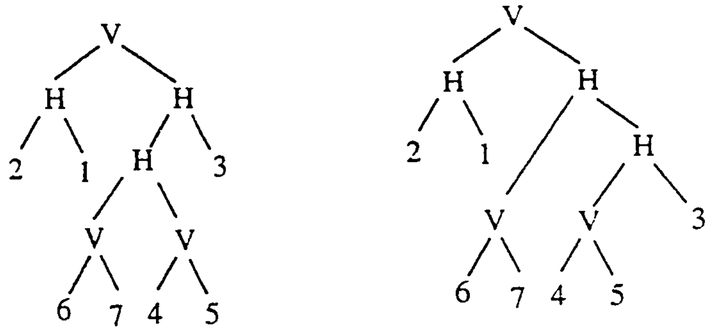


Figure 1.5: Nonslicing floorplans (wheels): (a) clock-wise; (b) anticlock-wise.

leaves and $n-1$ nodes. Each node of the tree represents a vertical or horizontal cut line (operator) and each leaf denotes a basic rectangle (operand). The restriction imposed during the process of bisection is that each cut (horizontal or vertical) must completely bisect one of the rectangles obtained from the previous cut into two rectangles. The steps of cutting the rectangle are depicted in Figure 1.4 (a) to (e) where the bold cut lines represent the most recent cut. A floorplan having such a characteristic is said to have *slicing structure*. When a slicing tree contains no operator such that the operator and its right son are the same. the slicing tree is called *skewed* else it is called *non-skewed*. The string obtained by a post-order traversal of the slicing tree is a *Polish expression*. where the operators (nodes of tree) are the horizontal or vertical cut lines and the operands (leaves of tree) are the basic rectangles (modules). For a floorplan of n modules. the length of the Polish expression is $2 \cdot n-1$ (with n operands and $n-1$ operators). Two possible slicing trees for the floorplan of Figure 1.4(e) are shown in Figures 1.6(a) and (b). This solution representation was proposed by Wong and Liu [4]. It is important to note that there exists a one-to-one correspondence between the set of slicing trees and the set of Polish expressions. A Polish expression corresponding to a skewed slicing tree is called *normalized* and the one for a non-skewed slicing tree is called *non-normalized* (see Figure 1.6). The horizontal and vertical cuts denoted by letters "H" and "V" respectively, act on rectangular sub-floorplans. The left (right) operand of "H" is the sub-floorplan placed below (above) the horizontal cut line. The left (right) operand of "V" is the sub-floorplan placed to the left (right) of the vertical cut line.

The slicing tree is used to determine the area of the floorplan layout, location of each module, and the estimated wirelength etc. A subsequent mapping is required to transform the abstract tree representation into the physical layout.



2 1 H 6 7 V 4 5 V H 3 H V

(a)

2 1 H 6 7 V 4 5 V 3 H H V

(b)

Figure 1.6: For floorplan in Figure 1.4(e): (a) skewed slicing tree and normalized Polish expression: (b) non-skewed slicing tree and non-normalized Polish expression.

1.4 Motivation for Timing Influenced Design

Until recently, the size of the bounding box (comprising functional area and routing area) was a widely used measure of the floorplan quality. However, due to recent advances in VLSI technology, the transistor size has been decreasing and its switching speed increasing. In the last two decades, the scaling has been so drastic that there has been a tremendous increase in the importance of interconnect delays with respect to the overall speed performance of the circuit. Hence physical design stages such as placement, routing etc., are made sensitive to timing.

Optimized for timing, a layout system can permit as much as 20% increase in its speed performance without any changes to the logic or cell design. In this work, we present the design of a timing influenced floorplanner. There are two main reasons that motivate the design of a timing influenced floorplanner. Firstly, it is important to note that in the last ten years, all stages of physical design subsequent to floorplanning such as placement and routing have been made sensitive to timing [8, 9, 10, 11]. One obvious conclusion is that if floorplanning stage is not made sensitive to timing, the previous estimates of area, shapes, positions of pins on modules etc., will be of no use in the later stages. Therefore to maintain the vertical consistency, it is mandatory to incorporate timing information during the floorplanning stage. Secondly, a timing influenced floorplanner gives an estimation of the maximum speed performance of the circuit. This information can be used to tune the circuit early enough in the design process.

1.5 Thesis Organization

Although, a number of strategies have been proposed and successfully applied to solve the floorplanning problem. very little attention has been paid to make floorplanning timing-driven. This thesis is the first attempt to use genetic algorithm for timing-influenced floorplanning of VLSI layouts. It accomplishes implementation of a program called, "Genetically Implemented Floorplanner for Timing (GIFT)". Experiments have been carried out on practical VLSI circuits to validate the performance of GIFT.

In Chapter 2, a survey of the reported work is presented. The general floorplanning techniques are described briefly and the widely used methods are discussed and compared. The previous related work on timing driven design is reviewed.

In Chapter 3, genetic algorithm is described. Various terms that are used in biological sciences are defined in the floorplanning context. The adaptation of genetic algorithm for floorplanning problem is discussed. The effectiveness of genetic operators described is illustrated. Various aspects of this technique are discussed in detail.

In Chapter 4, the core of this thesis is presented. Each of the objectives considered in this work is described and its computation is illustrated with examples. The implementation details of GIFT are given. The experimental results are presented and discussed.

Chapter 5 presents a summary of the thesis and describes some future extensions to this work.

1.6 Conclusion

In this chapter, a brief introduction of the VLSI design process and motivation behind this work is given. A formal definition of the floorplanning problem is presented. An overview of the general-cell floorplanner is described. The reason for including timing objective during floorplanning stage is discussed. Finally, an outline of the thesis is presented.

Chapter 2

Literature Survey

2.1 Introduction

Floorplanning is an $\mathcal{NP}-hard$ problem [12]. This implies that no algorithm exists which can optimally solve floorplanning problem in polynomial time. Due to the combinatorial nature of the problem, enumeration of all possible solutions is impossible for a floorplan. especially when the number of modules is large. To overcome this difficulty, several heuristics and polynomial time approximation algorithms have been reported in the literature. In the next section. the reported approaches are discussed.

2.2 Classification of Techniques

The main techniques to solve floorplanning problem can be classified under three broad categories namely:

- constructive,
- iterative, and
- knowledge-based.

A *constructive* algorithm builds a feasible floorplan starting from a seed module. The other modules are selected one by one or in a group and added to the partial floorplan. This process is repeated until all the modules have been considered. This method is generally very fast but results in poor quality solutions. The various methods belonging to this category are cluster growth, partitioning and slicing, connectivity clustering, mathematical programming and rectangular dualization. Some of these techniques have been discussed in detail in [2].

An *iterative* algorithm starts with a feasible floorplan. An initial solution is normally generated randomly or obtained by a constructive algorithm. A number of perturbation operations are performed on the initial solution(s). This process is repeated until an optimal floorplan is obtained or no further improvements are possible. The iterative techniques include simulated annealing, force-directed interchange/relaxation and genetic algorithm.

In a knowledge-based approach. a knowledge expert system is implemented that comprise the three basic elements namely: (a) a knowledge base that contains data describing the floorplan problem and its current state, (b) set of rules describing manipulation of data in the knowledge base in order to progress towards a solution and (c) an inference engine that controls the application of the rules to the knowledge base.

Another approach has been proposed in [13], that combines knowledge-based system (KBS) with algorithmic techniques. Such an approach permits the definition of a strategy where a KBS is used to embed IC design specialized knowledge. and algorithmic information processing techniques permit the use of fast quantitative evaluations, increased efficiency and increased knowledge modularity. It also enables a factorisation of the solution space and offers an environment that is open to integrate a large amount of information which directs the search process towards practical solutions.

Floorplanning algorithms may also be classified as deterministic or probabilistic. An algorithm that uses fixed connectivity rules, formulas or equations to arrive at a feasible floorplan is called *deterministic.* Such an algorithm will always generate the same floorplan for a certain problem. A *probabilistic* algorithm. on the contrary. may generate a different solution each time it is run. It generates. manipulates. and selects or rejects a solution on a random basis. Usually. constructive algorithms are deterministic. whereas iterative algorithms are probabilistic.

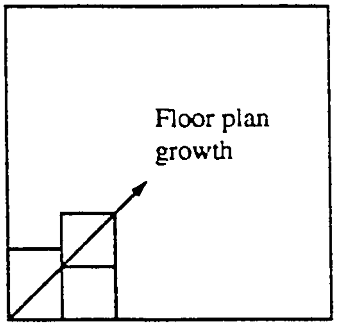


Figure 2.1: Cluster growth floorplanning.

Some of the widely used methods that have been successfully applied to solve floorplanning problem are cluster growth, min-cut, simulated annealing and genetic algorithm.

Cluster Growth

In this approach [2], the floorplan is constructed one module at a time in a greedy fashion until all the circuit modules have been assigned to all the blocks of a floorplan. A seed module is selected and placed into a (lower left) corner of a floorplan. The remaining modules are selected one at a time and added to the partial floorplan. A linear ordering algorithm is used to find the order in which the modules will be placed. For each selected module, a location is chosen so as to grow the floorplan evenly and at the same time, satisfying other criteria. This is followed by routability analysis. The growth of a floorplan is shown in Figure 2.1.

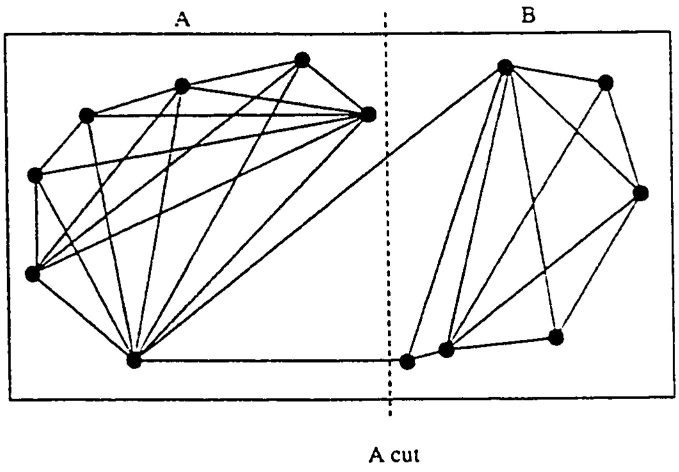


Figure 2.2: Partitioning of the circuit modules into two sets.

Min-cut

The min-cut algorithm clusters the blocks to be placed as per their connectivity [14, 15, 16]. The blocks in a floorplan are divided into two sets in such a way that the number of connections among these sets is minimized as shown in Figure 2.2. The algorithm is repeatedly applied to each set until the resulting sets contain only one block. This algorithm combined with some heuristics. generates locally optimal solutions in an acceptable time. Since only one cut is minimized at a time, the solution obtained is a local optimum. As the number of cuts increase. the connections between the two blocks of different sets may create problems for routing at a later stage. Several improvements to this basic algorithm have been proposed. These improvements. however. are inherently limited by the optimization strategy.

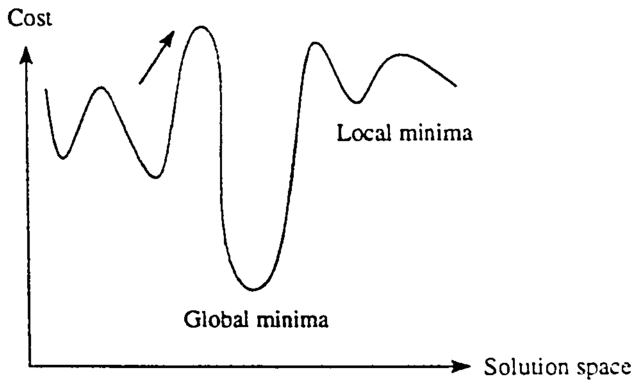


Figure 2.3: Hill-climbing phenomenon in simulated annealing.

Simulated Annealing

Simulated annealing (SA) is an adaptive heuristic that belongs to non-deterministic algorithms [17]. It has analogy with the statistical mechanics of annealing in solids. It allows exploration of global optimum since it has *hill climbing* properties (see Figure 2.3). It requires an initial solution representing some state of the search space. operations to generate local neighborhood of the initial state for better solutions. a cost function(s), and an annealing schedule. The search space is explored until the termination criterion is satisfied.

SA is one of the most widely used heuristics among researchers. Its application to floorplanning has been reported in [4, 18, 19, 20, 21]. There are two main variations of SA based floorplanning, namely direct and indirect. In direct SA, manipulations are done directly on the physical layout of the floorplan. In an indirect SA, an abstract representation (such as a graph) of a floorplan is used for manipulations.

The abstract solution is later mapped to obtain the physical representation of the floorplan. The work reported in [19] is an example of a direct approach whereas [4, 20] discuss an indirect approach.

Genetic Algorithm

Genetic Algorithm (GA) is a robust, stochastic combinatorial optimization search technique [5]. It has analogy with principles of natural selection and genetics. It combines the notion of *survival of the fittest* with random yet directed search and parallel evaluation of the points in the search space. An excellent reference on genetic algorithms is [6]. It has been successfully applied to a wide variety of problems both from academia and industry. The foundations of this strategy were laid down by Holland back in 1975 but its application to the area of VLSI design has been explored only recently. These include cell placement [22, 23], circuit partitioning [24], and floorplanning [25].

Algorithm	Speed	Solution
Cluster growth	Good	Medium-good
Genetic algorithm	Very slow	Near optimal
Min-cut	Medium	Local optimal
Simulated annealing	Very slow	Near optimal

Table 2.1: Table showing comparison of different algorithms.

An overview of GA is given in the next chapter. A comparison of some of the widely used methods is given in Table 2.1.

2.3 Review of Timing Driven Design Approaches

The problem of performance driven floorplanning consists of finding suitable locations of cells so as to minimize the total wirelength while satisfying user specified timing constraints. The problem may also be defined as that of finding a floorplan that minimizes $T_{\max}$, the delay of the longest path in the circuit (which includes the interconnect delays).

During the design process, the propagation delays on the interconnects are not known prior to layout. Long path timing problems registered after layout are very difficult to correct because they may require, not only new iterations of the physical design step, but possibly, many iterations of the logic design step.

Three general approaches have been suggested to correct long path timing problems. The first approach proceeds by making changes to the logic. For example, the delay of a path can be substantially decreased by reducing the loading on some of its circuits elements. Also collapsing some of the logic on the long paths can reduce some of the paths' delays. Representative implementations of this approach are reported in [26, 27, 28, 29, 30].

The second approach relies on transistor re-sizing to speed up some of the circuit elements on the slow paths. By increasing the sizes of some of the driving transistors, the switching delays of the driving elements as well as the propagation delays along the nets driven by the re-sized transistors can be substantially lowered. Examples of implementations using this approach have been described in [29, 26, 31, 32, 33].

The third approach to make a circuit faster without making any changes in its logic design is to reduce the propagation delay due to interconnects to a minimum or well within tolerable limits. This goal can be achieved by imposing timing constraints on the interconnects and paths of the design. That is, wiring delays must be kept in check so that the path delays are kept below a maximum value. Since physical design step (partitioning, floorplanning, placement, and routing) affects the wiring requirements of a layout, the objective of the physical design step is altered to lower the path delays below the latest required arrival time. This approach has been adopted in our work.

Optimized for timing, a layout system can permit as much as 20% increase in the clock rate without any changes in the logic or cell design. Numerous attempts have been reported which tried to make the physical design sensitive to the timing requirements. Work in this direction was initiated as early as the 1970's [33] where ideas from physics were employed to optimize the power and timing of LSI chips. In [34], a system was reported in which a circuit which is completely laid out is simulated on the computer to determine long paths that violated constraints. The layout is then adjusted and simulated again. In [35], performance driven circuit partitioning heuristics that resulted in performance improvements with little loss in wirability were reported.

Methods for generating constraints on sizes of nets to guarantee performance are reported in [36, 37, 38]. These methods consist of distributing slacks on the nets. The final layout that satisfied these net bounds was guaranteed to satisfy path timing constraints for desired performance. In [39], factors which are highly correlated with path timing such as number of nets on the path, path slacks, driving strengths of cell output pins, etc., are combined into a score function. Path criticality is decided on the basis of path scores. The predicted critical paths are used by the placement procedure.

In [40], a linear-programming approach has been attempted to reduce an estimate of cycle time. Timing driven placement that uses the idea of rectilinear distance facility location and partitioning to minimize wire delays to satisfy timing constraints were proposed in [41]. A constructive placement method to sequentially place cells using a cost function that captures the timing behavior was presented in [42]. In [43], the problem of performance driven physical design is formulated as a constrained programming problem. Constraints are placed on total path delays including cell and interconnect delays and the behavior of the paths is captured. Mathematical techniques and heuristics based on Lagrangian relaxation are used to find an approximate solution to the constrained problem. In [44], a description of a placement system based on the fuzzy logic approach is presented. A solution that satisfies multiple objectives, which are, area, routability, and timing, is produced using fuzzy logic rules. In [8], the application of constructive successive augmentation methodology to VLSI placement under constraints on routability, area and timing is presented. The placement algorithm uses adaptive look-ahead procedures to improve the effectiveness of constructive decision making. The methodology was successfully implemented for macro-cell-library based sea-of-gates design style with over the cell routing.

Iterative and non-deterministic techniques have also been employed. In [9], the authors employ simulated annealing [45] to improve both the wirelength and performance. Timing issues were not considered.

Other iterative non-deterministic techniques that have been applied to the placement are genetic algorithm (GA) [22, 46] and simulated evolution [47]. In both these works, however, minimization of wirelength was the sole objective.

2.4 Conclusion

This chapter surveyed various approaches to floorplanning as reported in the literature. A classification of solution approaches was given. Some of the widely used methods were discussed. Several timing-driven design approaches were reviewed.

Chapter 3

Genetic Algorithm and VLSI Floorplanning

3.1 Introduction

This chapter presents an overview of genetic algorithm (GA). The motivation for applying GA to floorplanning problem is discussed. Several technical terms are introduced. Various genetic operators are explained in the context of floorplanning and illustrated with examples.

3.2 An Overview of Genetic Algorithm

Genetic algorithm has emerged as a robust optimization and search technique [5, 6]. It has analogy with the mechanisms of evolution and natural genetics. It succeeds in locating the global solution optimally in a large search space with a high probability. In nature, best fit individuals win the race for meagre resources such as food, space, mates etc. Only the fittest individuals survive and reproduce, a natural phenomenon called *survival of the fittest*. Adaptability to a rapidly changing environment is necessary for the survival of individuals belonging to various species. While the features that distinctly characterize an individual determine its ability to survive, the features in turn are determined by the individual's genetic material.

As against calculus-based optimization methods such as gradient descent, GA does not use derivatives. Another feature of GA is that it performs a parallel search of the solution space, contrary to the normal point-by-point search found in other optimization algorithms. It can explore many regions of the search space simultaneously. These features make genetic algorithm highly immune to getting trapped in the local minima (maxima). This is very significant considering the fact that the real life data is very noisy and the search space may have many local maxima-minima which may be sub-optimal. This is illustrated in Figure 3.1.

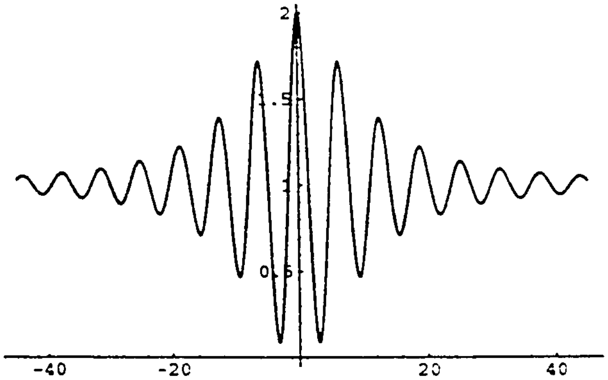


Figure 3.1: A plot of function with many local maxima-minima.

3.2.1 GA Terminology

In the theory of natural evolution, an entity that encodes specification of an organism is called a *chromosome*. One or more chromosomes may be required to represent the organism completely. The complete set of chromosomes is called a *genotype* and the resulting organism is called a *phenotype*. Each chromosome comprises a number of individual structures called *genes*. A gene encodes a particular feature of the organism and the location of the gene determines the characteristic that a gene represents. The different values of a gene are called *alleles*.

In GA, chromosomes are represented by a string of some type. In problems where there is only one chromosome per organism, the chromosome and the genotype are the same. Each position in the chromosome string is a gene which may have different values. The phenotype is the solution decoded by genotype.

3.2.2 Steps of GA

There are several steps involved in genetic formulation of an optimization problem namely representation, evaluation, selection and genetic operations. We explain each of these steps in the following paragraphs.

1. **Representation**

   The primary task is to find a suitable encoding or representation of feasible solutions which can be handled by the computer. In the simplest case, a string of binary digits may be used. In a binary string, each bit is a gene. A solution in general, may have many chromosomes, each one comprising several genes. A population of the solution is a set of some finite number of chromosomes that represent the solution completely.

2. **Evaluation**

   Each chromosome in the population is decoded and evaluated in order to determine how well it solves the problem (referred to as the fitness). The fitness is used to determine the number of offsprings, a particular chromosome will contribute in the next generation of the population. It represents a measure of how close an individual is to the desired solution.

3. **Selection**

   The individuals are selected from a population based on their fitness values. Selection imitates nature's survival of the fittest mechanism. The fitter solutions survive while the weaker ones gets destroyed. In a simple genetic algorithm, a fitter solution inherits higher number of offsprings and thus has a higher rate of survival in the subsequent generations. The selected solutions undergo genetic operations. These operations are described next.

4. Genetic Operations

Crossover and mutation are the two most commonly used genetic operators. These operators are applied probabilistically. Crossover works on a pair of solutions whereas a single solution is subjected to mutation operator. The operation of crossover and mutation depends greatly upon the nature of the problem. These are described below:

a. Crossover

Crossover is a powerful mechanism for probabilistic exchange of useful information. The main idea is that the genetic information of a good solution is spread over the entire population. Thus, the best solution can be obtained by thoroughly combining the individuals in the population.

b. Mutation

Mutation is a means of introducing new information into the population. It is motivated by the possibility that the initial population might not have contained the information necessary to solve the problem.

The design of a genetic algorithm for any problem essentially consists of string representation of the solution, choice of genetic operators, determination of fitness function and determination of the probabilities controlling the genetic operators. Each of these greatly influences the solution obtained and the performance of the genetic algorithm.

3.3 Basic Genetic Algorithm

The structure of a basic genetic algorithm is illustrated in Figure 3.2. Generally, the initial population of individuals is generated randomly. However, some of the individuals in the population may be obtained by constructive approaches. Each individual in the population is assigned a numeric value indicating its merit by a fitness function. The fitness function assigns higher numbers to better fit individuals. Once all the members of the population have been evaluated, their fitnesses are used as the basis for selection. Selection is implemented by eliminating low-fitness individuals from the population, and inheritance is implemented by making multiple copies of high-fitness individuals. Usually, *roulette wheel selection scheme* is used to implement proportionate selection. The genetic operators are then applied probabilistically to produce the offsprings. By transforming the previous set of good individuals to new ones, the operators generate a new set of individuals that have a better than average chance of being good. The cycle of evaluation, selection, and genetic operations (also called *Mating Cycle*) is iterated until the size of new population equals the size of initial population. The individuals in the new population represent improved solutions. In many implementations, the set of offsprings so generated, completely replaces the old population. In some other implementations, however, a replacement policy is employed that may be one of the following:

- advance the best individuals from the set of old population and offsprings,
- advance randomly selected individuals from the set of old population and offsprings,
- advance p% best individuals and select remaining individuals randomly, and
- advance p% best, q% worst and select remaining individuals randomly.

The new population becomes the current solution set for the later generations. The outer loop called *Generation Cycle*, is repeated until no further improvement in the objective function is achieved or some termination condition is satisfied.

3.4 Genetic Floorplanning

The application of genetic algorithm to floorplanning has been reported recently in the literature [25]. In this work, Cohoon et. al. have used slicing floorplans. It is motivated by the concept of punctuated equilibria in evolution theory described in [48], and aims at effectively using large distributed memory, message passing and

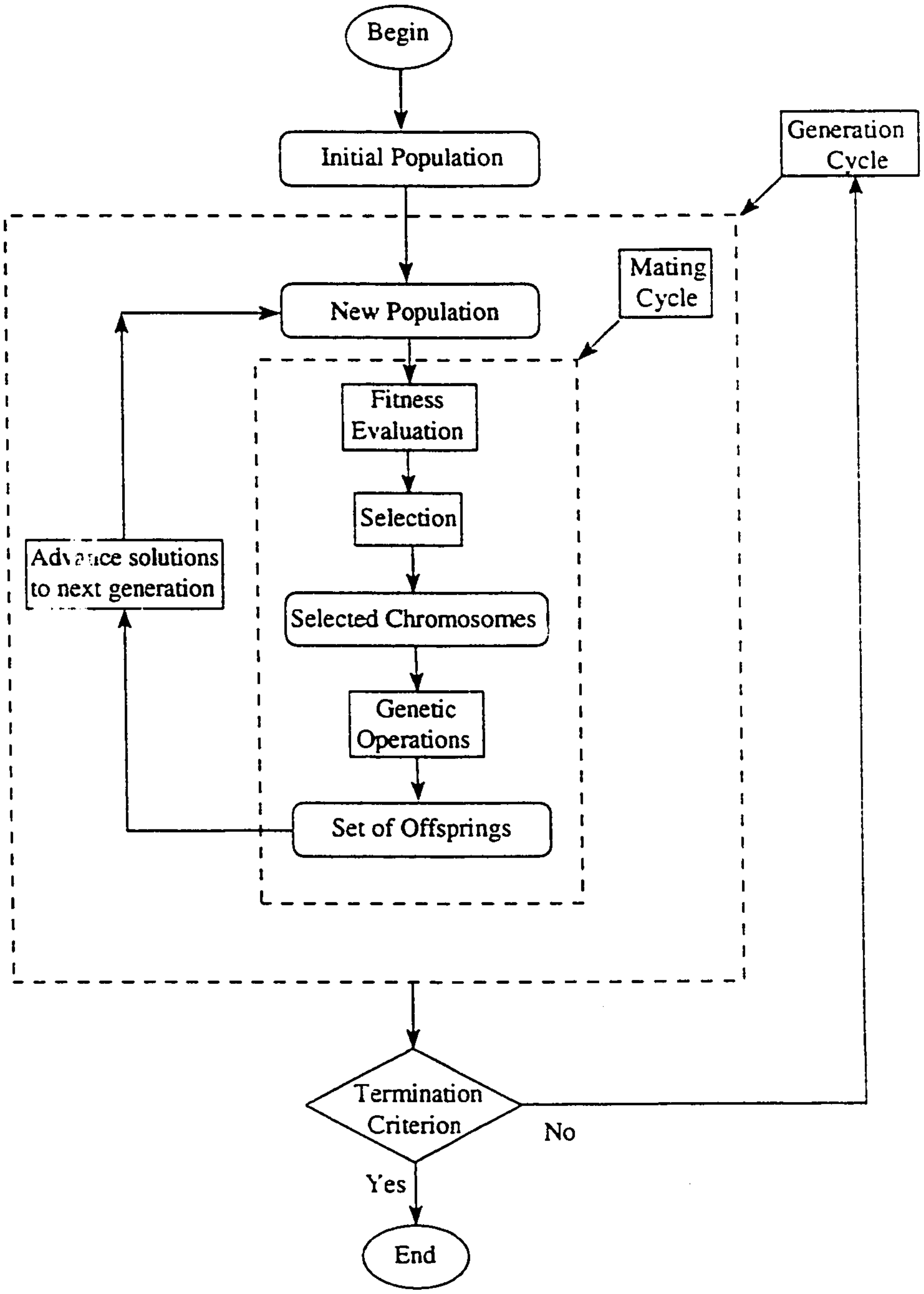


Figure 3.2: A flowchart of basic genetic algorithm.

parallel processing systems. A weighted sum of area and wirelength estimation is used as an objective function. The authors have reported that genetic algorithm approach has generated results better as compared to simulated annealing both in terms of the average cost and the best solution found. The timing issue, however, was not taken into consideration. We follow the guidelines of this work and incorporate timing performance of the circuit besides area and wirelength as a measure to evaluate the floorplan solution. We also aim to explore some more genetic operators suitable for the new cost function. A very good survey on various crossovers for placement problem is found in [23]. These need to be modified for floorplanning application. Some good crossovers have also been reported in [25]. The chromosomal encoding can be done for floorplan configuration, orientation of each module and timing constraints on the circuit floorplan. Regarding the mutation operator, the perturbation moves used in simulated annealing by [4], may be used. We consider various genetic operators for floorplanning problem in the next section.

3.4.1 Initial Population Generators

In this work, an initial population is generated randomly which consists of valid Polish strings. A sample of randomly generated initial solutions for a population of size 10, having 12 modules is shown in Figure 3.3.

{2. 11. V. 3. V. 1. V. 6. H. 7. H. 5. V. 4. H. 10. H. 8. H. 9. H. 12. V}
{7. 5. H. 3. H. 1. V. 9. V. 12. H. 2. H. 6. V. 10. H. 4. H. 8. H. 11. H}
{5. 12. V. 1. V. 4. H. 6. H. 8. H. 10. H. 9. H. 2. H. 7. H. 3. H. 11. H}
{10. 3. H. 2. H. 6, H. 5. V. 4. H. 12. V. 8. H. 1. V. 9. H. 11. V. 7. H}
{3. 10. H. 2. V. 7. H. 4. V. 11. V. 5. V. 1. H. 12. V, 9. H. 8. H. 6. H}
{12. 11. V. 1, H. 7. V. 10. V. 5. H. 9. H. 2. H. 4. H. 3. H. 8. V. 6. H}
{1. 12. V. 4. V, 3, H. 7. H. 11. H. 8, V. 10. H. 5, V. 6. V. 9. H. 2. V}
{8. 7. H. 1. H. 10. H. 12. H. 3. H. 2. V. 6. H. 5. H. 11. V. 4. H. 9. H}
{6. 5. V. 11. V. 7. H. 4. H. 1. V. 12. V. 9. V. 2. H. 10. H. 3. H. 8. H}
{5. 11. V. 4. H. 7, V. 2. V. 12. H. 1. H. 3. H. 6. H. 10. H. 9. V. 8. H}

Figure 3.3: A sample of random population for a floorplan with 12 modules.

3.4.2 Selection

There are two selection methods, one for genetic operations and the other for advancing individuals to next generation. The selection function for genetic operations is based on the *proportionate selection* scheme implemented by roulette wheel method. The algorithm selects two strings, performs crossover and generates the offspring which undergoes mutation depending upon the probability of mutation. This cycle is iterated until the new population size is equal to the initial population size. The selection method for advancing individuals to next generation is described in the next chapter.

3.4.3 Crossover

The basic crossover operator is implemented by choosing a cross-point randomly in the parent strings and copying the sub-strings before the cross-point as they are and exchanging the sub-strings after the cross-point between the two parents as shown in Figure 3.4. This crossover is simple to implement but does not generate valid offsprings in all the chromosomal representations. This is true specially for problems such as placement and floorplanning, where a solution chromosome comprises several genes. Hence, a new set of crossover operators is required for floorplanning. In the following section, some of the possible crossovers that result in valid solutions are discussed and illustrated with examples. For all the examples, the parent chromosomes considered are shown in Figure 3.5.

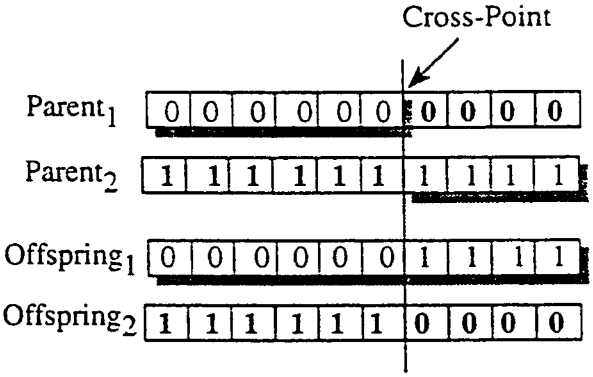


Figure 3.4: Basic crossover operation.

Block Inheritance Crossover (χ_1)

A simple crossover that generates a valid offspring is implemented as follows [48]. Copy the *operands* from a parent (say P_1) insitu and copy the operators from the other parent (say P_2) at the remaining positions. This crossover inherits the building blocks from one of the parents to the offspring. This is shown in Figure 3.6.

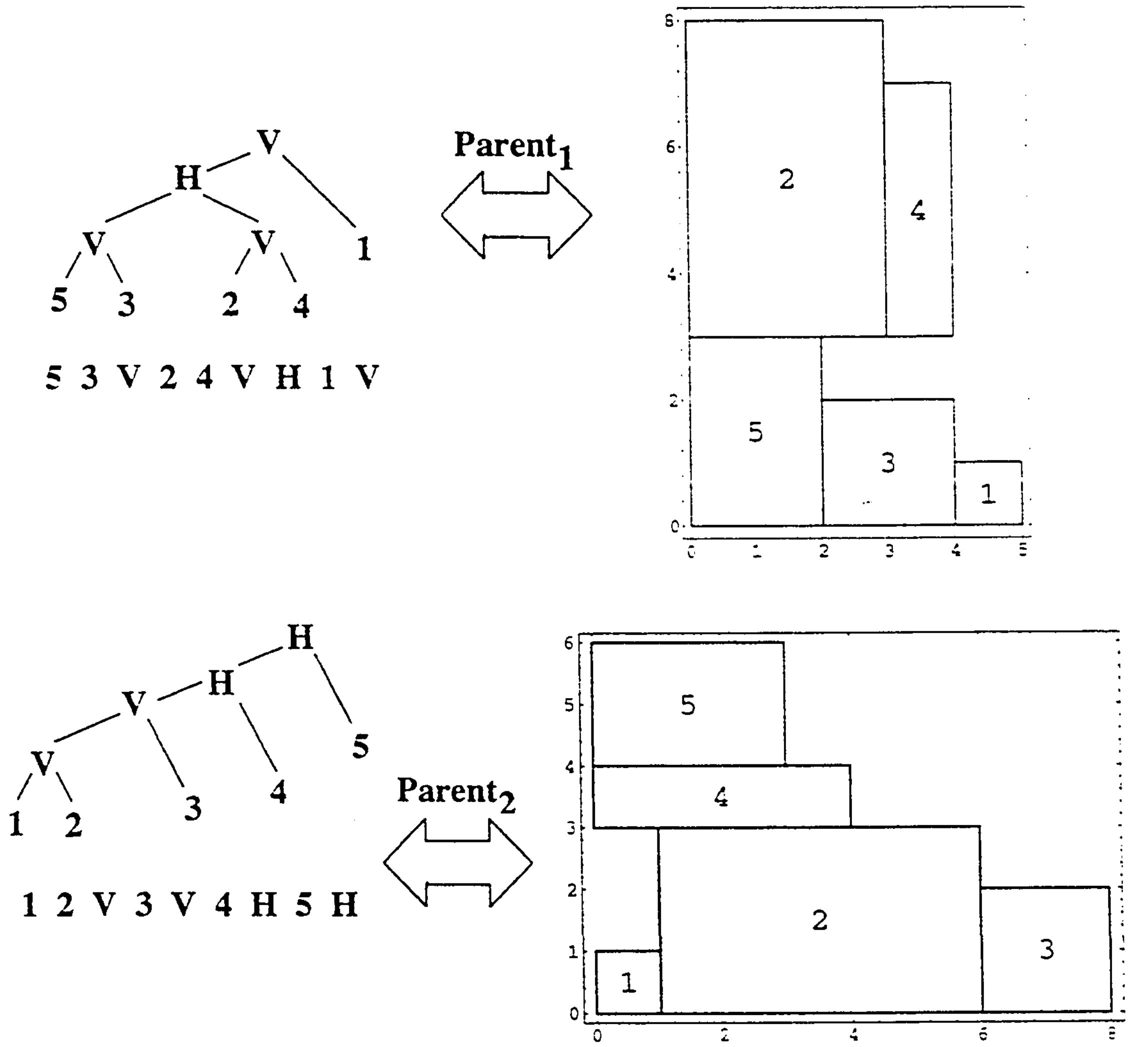


Figure 3.5: Two parent chromosomes and the corresponding floorplans.

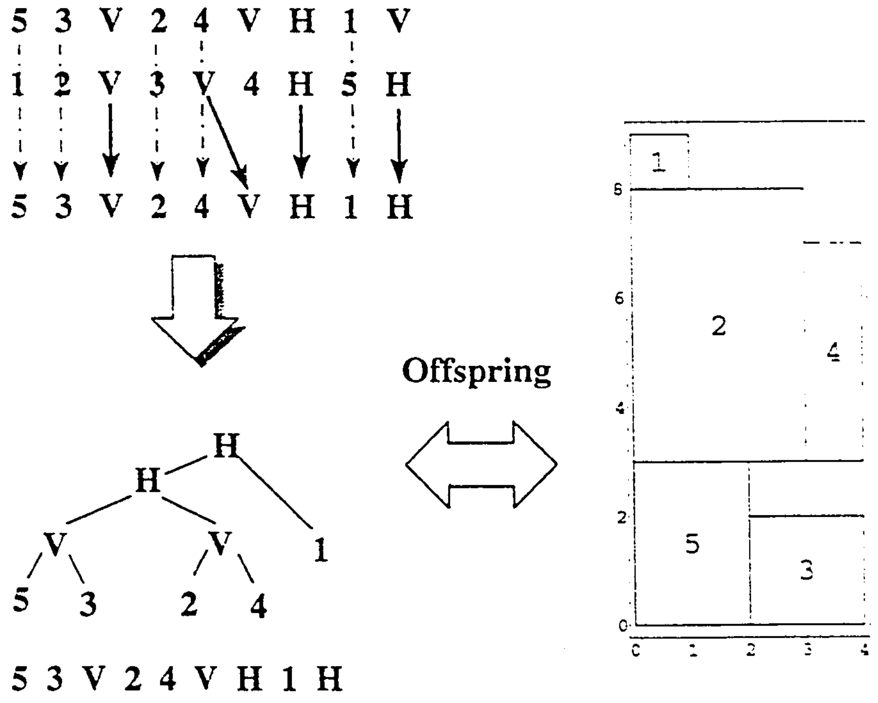


Figure 3.6: Crossover χ_1 that propagates building blocks to the offspring.

Slicing Inheritance Crossover (χ_2)

This crossover inherits the slicing structure from one of the parents. It is implemented by copying the *operators* from a parent (say P_1) insitu and completing the offspring by copying the operands from the other parent (say P_2) at the remaining positions as shown in Figure 3.7. It may be observed that the floorplan corresponding to the offspring has the same slicing structure as that of one of the parents.

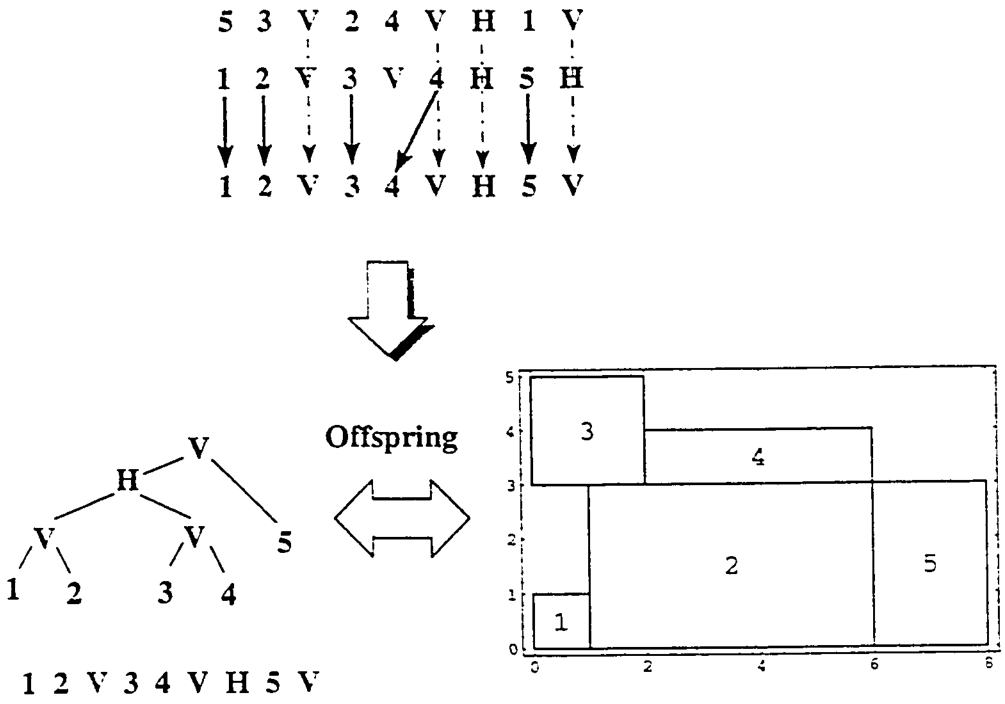


Figure 3.7: Crossover χ_2 that propagates slicing structure to the offspring.

PMX Crossover (χ_3 and χ_4)

PMX stands for *Partially Mapped Crossover* [23]. In PMX, a cross-point is chosen as stated earlier. The sub-string to the right of the cross-point is copied from one parent to the offspring insitu. Next, we start from the first gene of the other parent. Before copying, we check if that gene is already present in the offspring. If so, another gene from the parent having the same position is considered else the gene is copied insitu. This process continues until the complete offspring is obtained.

In floorplanning, a PMX is applied on the string of operands only. The operand sub-string to the right of cross-point (4 and 5 as shown in Figure 3.8) are copied from one parent insitu in the offspring. Next, the first gene (5 in Figure 3.8) in the other parent is considered. Since 5 is present in the offspring, operand 1 that is present in the first parent at the position of 5 is considered. As 1 is not present in the offspring, 1 is copied. Next, operand 3 in the first parent is considered. Since it is not present in the offspring, it is copied insitu. Similarly, operand 2 is copied. This way, an intermediate offspring is obtained. There are two ways in which operators can be inserted in the string of modules to get the offspring. These are:

- copy operators from the first parent insitu (χ_3)
- copy operators from the other parent insitu (χ_4)

This process is illustrated in Figure 3.8.

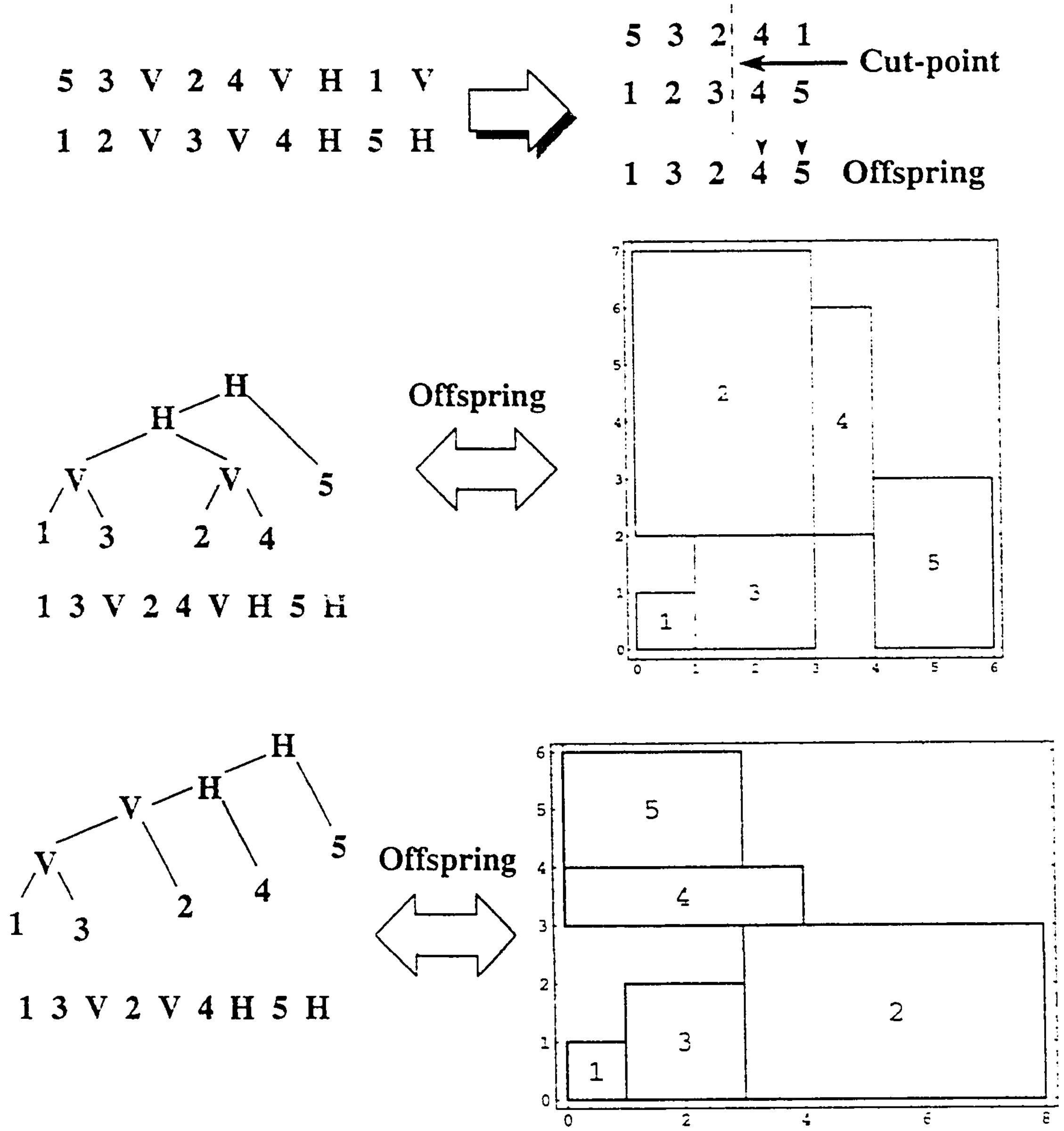


Figure 3.8: Illustration of PMX crossover operators λ_3 and λ_4.

Sub-tree Crossover (χ_5)

In a sub-tree crossover [48], the idea is to allow inheritance of a slicing sub-tree from one of the parents to the offspring. This is achieved as follows:

First, scan the parent chromosome from left to right and identify a sub-tree. Next, copy the sub-string corresponding to the sub-tree insitu in the offspring. Copy all the operators from the same parent and the remaining operands from the other parent to complete the offspring. This is shown in Figure 3.9.

Cycle Crossover (χ_6)

In cycle crossover [23], elements are copied from either parents in a cycle. The cycle is initiated by copying an element (usually first) of one parent (P_1) to the offspring insitu. If the element being copied is similar to that in the other parent (P_2), this cycle will continue by copying more elements from P_1 to offspring. However, if the copied element is different from the one in P_2, it can not be copied from P_2. Hence, that element is also copied to the offspring from P_1. The cycle from P_1 will end when no more elements need to be copied from P_1. Next, a cycle begins from P_2 by copying elements to the offspring. These cycles alternate between P_1 and P_2.

Similar to PMX crossover described above, the cycle crossover is implemented on operands only and operators are inserted in two ways in the string of modules to get the offspring. This is shown in Figure 3.10.

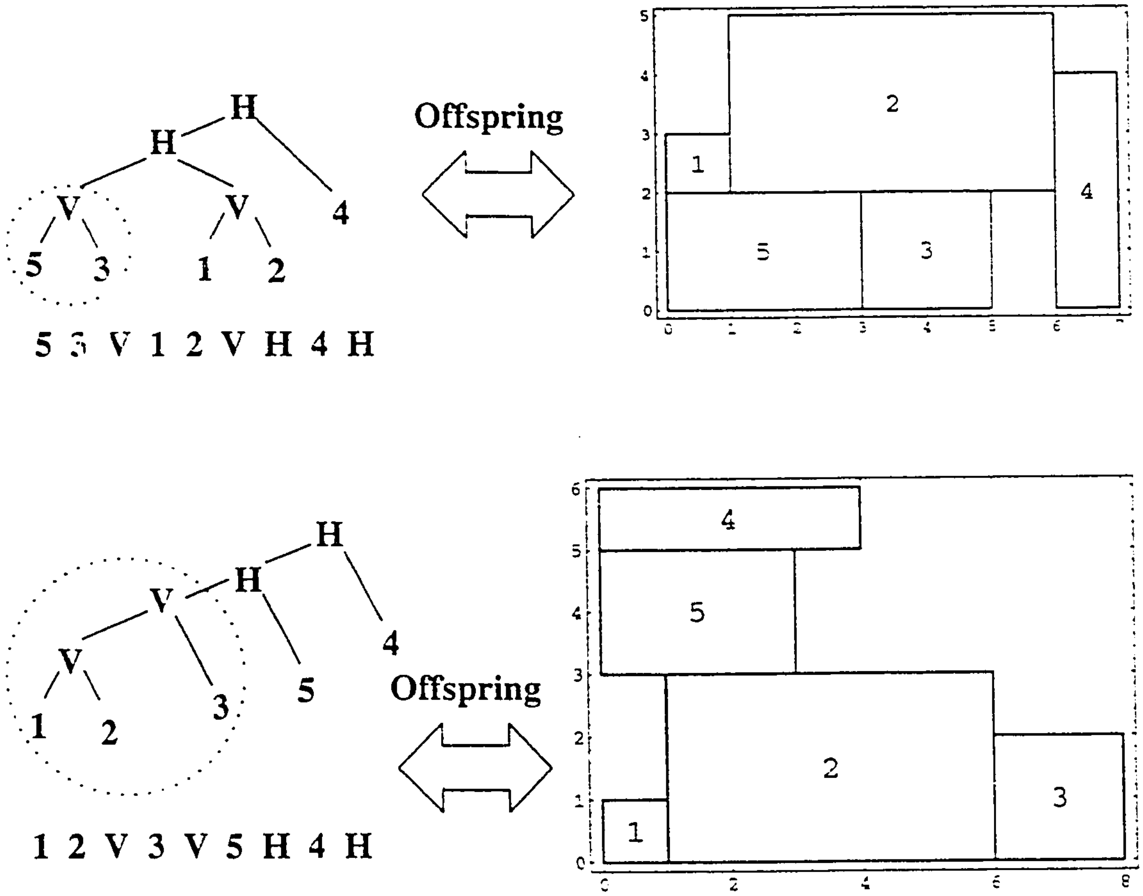


Figure 3.9: A sub-tree crossover operator χ_5.

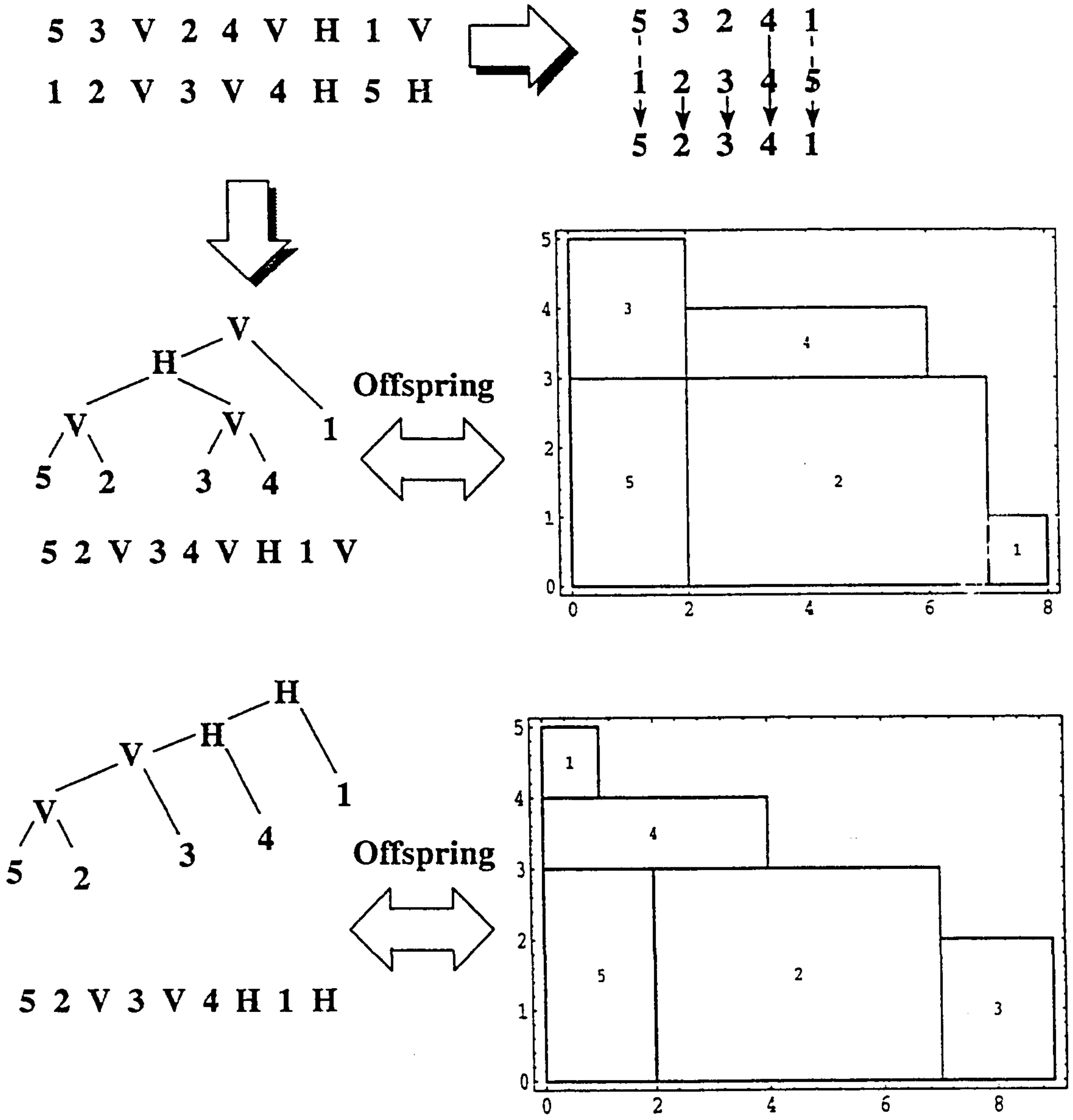


Figure 3.10: Illustration of cyclic crossover operator χ_6.

3.4.4 Mutation

The mutation operators chosen are based on the solution perturbation operators proposed in [49]. In addition to these, two more mutation operators have been implemented. Next, we briefly recall the mutation operators given in [49] (μ_2, μ_3, μ_4) and explain the working of two newly introduced operators (μ_1 and μ_5).

Invert single operator (μ_1): In this case, any operator from a given Polish string is chosen randomly and inverted from "V" to "H" and vice-versa.

Invert a chain of operators (μ_2): This operator inverts a series of adjacent operators.

Swap adjacent operands (μ_3): In this operator, two adjacent operands are swapped.

Swap adjacent operand/operator (μ_4): This move swaps two adjacent operator and operand.

It may be observed that the first three operators result in a valid Polish expression. The last operator, however, may generate an invalid Polish expression. Hence, a condition known as "Balloting Property" needs to be checked before performing such an operation. A Polish expression P is said to have balloting property if and only if for every sub-expression $P_i = p_1 \ldots p_i$, for $1 \leq i \leq 2n - 1$, the number of operands is greater than the number of operators [49]. A simple method explained below is used to check the violation of balloting property.

Let N_k be the number of operators in the Polish expression $P = p_1\ p_2\ \ldots\ p_k$, for $1 \leq k \leq 2n - 1$, where n is the length of expression P. Assume that μ_4 swaps the operand p_i with the operator p_{i+1}, for $1 \leq i \leq k - 1$. Then, the swap will not violate the balloting property if and only if $2N_{i+1} < i$. The mutation operators μ_1 to μ_4 are illustrated in Figure 3.11 for the Polish expression $53V24VH1V$.

Timing biased module exchange (μ_5): This mutation operator is aimed at improving the interconnection delay. It works as follows: first, a net violating the net delay bound on a critical path is identified. Then, modules on this net are selected and brought closer by swapping them with other modules not on critical paths.

3.5 Efficacy of Genetic Operators in Guiding the Search

The genetic operators described in the previous sections result in a valid floorplan solution. Since the operators manipulate the abstract floorplan representation, their role in guiding the search is not obvious. In the following sections we provide some insight in the use of these operators.

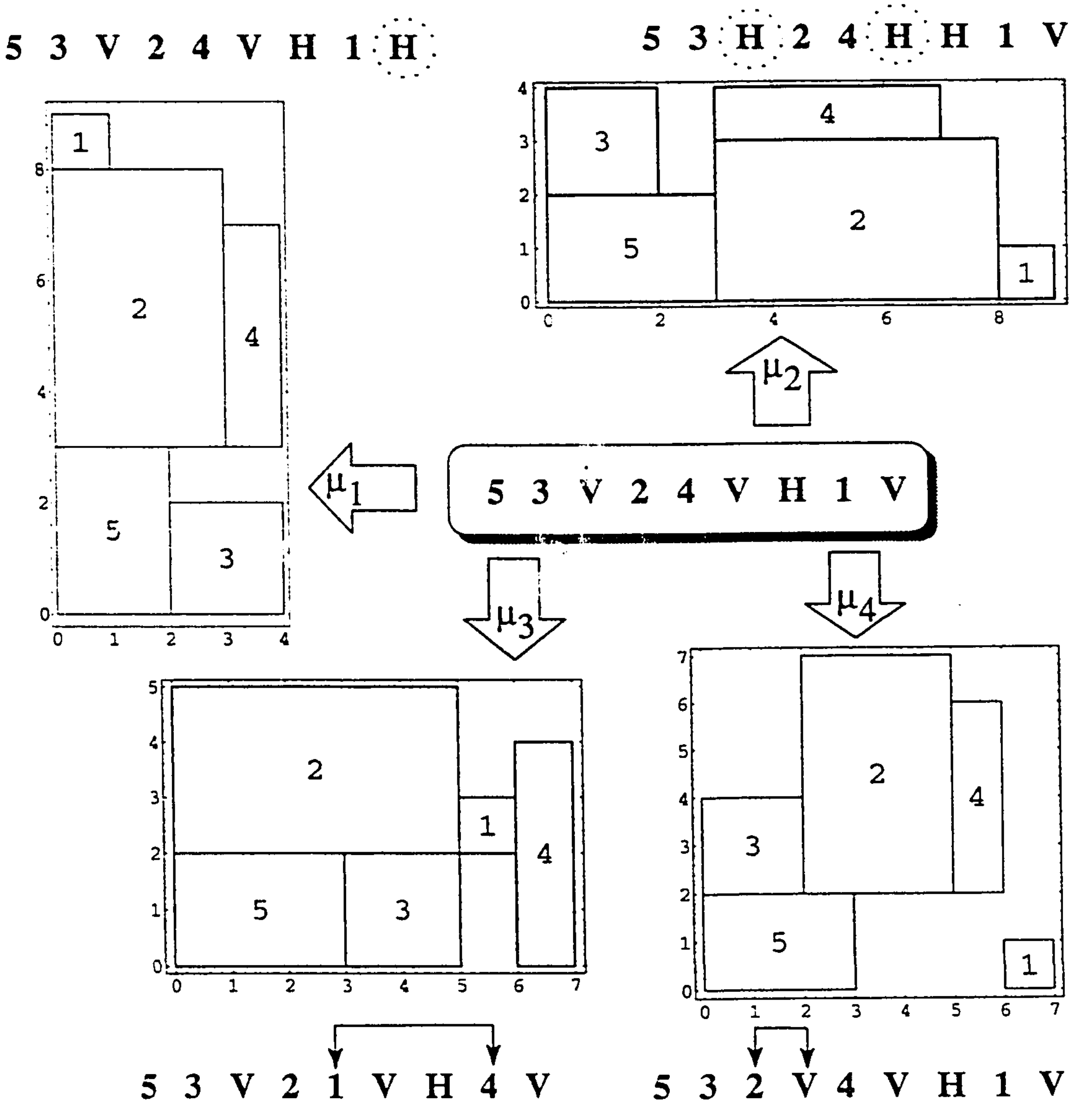


Figure 3.11: Illustrating the effect of various mutation operators.

3.5.1 Effect of Crossover

The crossover operators discussed above fall into two general categories with regard to the characteristics inherited by the offsprings from the parents. These are as follows:

- slicing structure information
- genotype sub-tree besides slicing structure information

The crossovers χ_2, χ_3, χ_4 and χ_6 belong to the first category whereas χ_1 and χ_5 belong to the second category. The crossovers may also be classified with respect to the amount of disturbance or modification they cause to the resulting offspring with respect to the parents. Operators such as block inheritance, PMX, and cyclic are highly disruptive. On the other hand, sub-tree crossover causes lesser disruption.

Empirical and theoretical studies have analyzed the merits and demerits of various crossover operators [50]. Empirical evidence suggests that the population size is related to the type of crossover. It has been observed that the lesser disruptive crossovers preserve genetic material, but they become lesser exploratory when the population becomes dominated by multiple copies of certain individuals. Hence, when the population size is large (50-200), the inherent diversity in the population precludes the need to have a highly explorative crossover. On the other hand, highly disruptive crossovers are capable of performing extensive exploration when the population size is small (10-30). Thus, the choice of a crossover is decided by the complexity of the problem, the string representation, and the size of the population. We have chosen both lesser and highly disruptive crossovers so that for small floorplans, a large population can be maintained and lesser disruptive crossovers can be used effectively. Whereas, for large floorplans consisting of 50 modules and above, a small population is mandatory (taking into account the time and memory constraints), and highly disruptive crossovers can be used.

3.5.2 Effect of Mutation

The mutation operators discussed take a single individual, and modify it in a localized manner. Mutation operators help in the exploration in the beginning when the average fitness is low. As the average fitness increases, the exploratory effect of mutation operators decreases. Contrary to the simulated annealing moves, the random change caused by the mutation operators seems to be always accepted. However, since the selection is performed on a population of solutions, the useless solutions are automatically thrown out.

3.6 Conclusion

The success of genetic algorithms is attributed to the following:

- it manipulates a representation of the potential solutions rather than the solutions themselves.

- it performs a search from a population of feasible solutions rather than a single solution,
- it is probabilistic in nature,
- it has hill climbing property, and
- it accepts the mutated solutions without any criterion unlike in simulated annealing.

In this chapter, we reviewed important aspects of genetic algorithm. The genetic vocabulary was introduced. A basic genetic algorithm was explained. A number of genetic operators were discussed. Finally, the effect of these operators was described.

Chapter 4

GIFT Floorplanner

4.1 Introduction

This chapter describes the implementation details of a performance driven floorplanning program that uses genetic paradigm. The program is named $\mathcal{GIFT}$, an abbreviation for "Genetically Impelemented Floorplanner for Timing". Due to non-availability of standard floorplanning benchmarks, a number of other available circuits were used for the validation of GIFT.

As mentioned earlier, GIFT combines three objectives namely area of the floorplan bounding rectangle, overall wirelength, and the speed performance of the circuit. In the following sections, the computation method for each of the objectives is described and illustrated with examples. The details of implementation are given and experimental results discussed.

4.2 Area

With area as the sole objective, the concern of the designer is to find a feasible floorplan with the smallest overall area. The process of area evaluation for slicing floorplans is illustrated by an example in the next section.

Definition 1 *A continuous curve on a plane denoted by* Γ *is called a bounding curve if it satisfies the following conditions [49]:*

(1) it is decreasing, i.e., for any two points (x_1, y_1) and (x_2, y_2) on Γ, if $x_1 \leq x_2$ then $y_2 \leq y_1$;

(2) Γ lies completely in the first quadrant, i.e., $\forall (x, y) \in \Gamma$, $x > 0$ and $y > 0$; and

(3) it partitions the first quadrant into two connected regions. The connected region containing all the points (x, x) for very large x is called the bounded area with respect to the bounding curve Γ (see Figure 4.1).

Let Γ_1 and Γ_2 be two bounding curves. Two arithmetic operations on bounding curves are defined as follows:

(1) the bounding curve corresponding to $\Gamma_1 H \Gamma_2$ is obtained by summing the two curves along the y-axis, i.e., $\Gamma_1 H \Gamma_2 = \{(u, v+w) | (u, v) \in \Gamma_1 \text{ and } (u, w) \in \Gamma_2\}$;

(2) the bounding curve corresponding to $\Gamma_1 V \Gamma_2$ is obtained by summing the two curves along the x-axis, i.e., $\Gamma_1 V \Gamma_2 = \{(u+v, w) | (u, w) \in \Gamma_1 \text{ and } (v, w) \in \Gamma_2\}$.

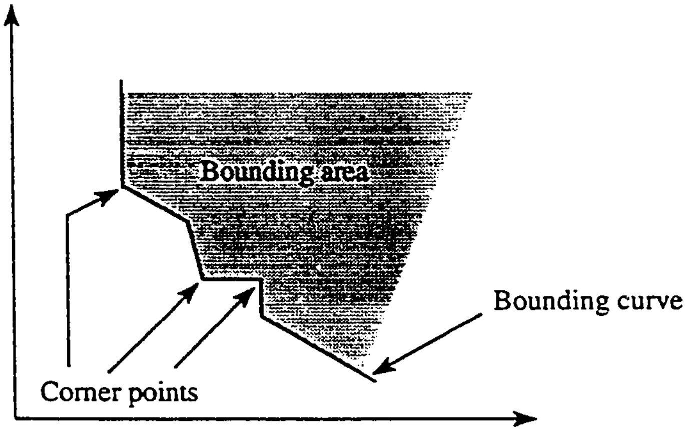


Figure 4.1: A piece-wise linear bounding curve [2].

A piecewise linear bounding curve is completely characterized by an ordered list of its corner points. Moreover, to add two piecewise linear curves along either direction, it is sufficient to sum up the two curves at their corner points.

Module	Width	Height
1	2	3
2	2	2
3	1	3
4	2	3
5	1	2
6	2	2
7	1	5

Table 4.1: Dimensions of floorplan modules.

Assume that the dimensions of floorplan modules are as shown in Table 4.1. It is further assumed that all modules are rigid and can be rotated by 90° with respect

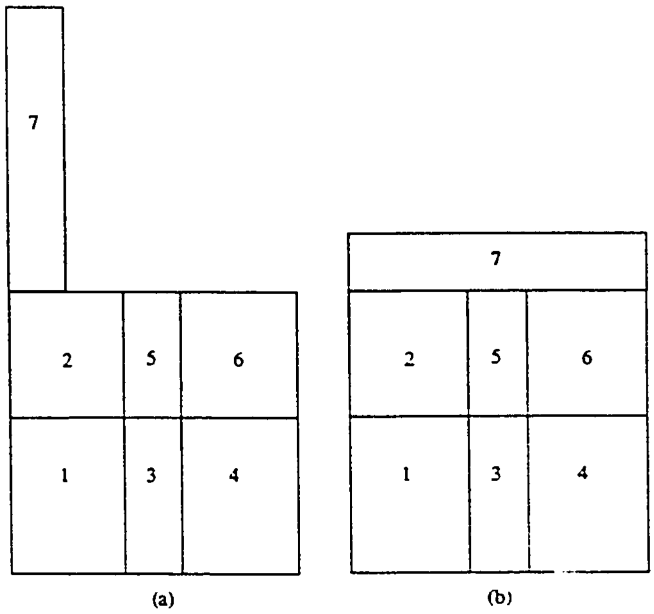


Figure 4.2: Floorplan for illustration of area computation: (a) original orientation of modules; (b) new orientation of modules.

to their original orientations.

The normalized Polish expression corresponding to the floorplan shown in Figure 4.2(a) is $P = 12H34V56VHV7H$, and its tree is shown in Figure 4.3. The goal is to determine a minimum area enveloping rectangle R_P corresponding to P.

In the slicing tree of Figure 4.3, the set of points enclosed between curly braces and appearing next to each node (leaf or internal) is the bounding curve of that node. For example, the leaf node labeled "1" corresponds to module 1 whose width and height are $w_1 = 2$ and $h_1 = 3$. Since the module can be rotated then for basic rectangle 1 to enclose module 1, its height and width must satisfy the following inequalities (refer to Figure 4.3):

$x_1 \geq 2$ and $y_1 \geq 3$ (normal orientation of module 1) or

$x_1 \geq 3$ and $y_1 \geq 2$ (module 1 rotated by 90°).

The bounding curves for the remaining leaf nodes are obtained in a similar fashion. Let Γ_{34} be the bounding curve corresponding to the subtree "34V". Then, Γ_{34} is computed as follows: $\Gamma_{34} = \Gamma_3 V \Gamma_4$, where the operation V is the summation of the two curves along the x-axis; $\Gamma_3 = \{(1,3);(3,1)\}$ and $\Gamma_4 = \{(2,3);(3,2)\}$, therefore $\Gamma_{34} = \{(3,3);(4,3);(5,3);(6,2)\}$. Since points (4,3), and (5,3) are not corner points, they are eliminated. Hence $\Gamma_{34} = \{(3,3);(6,2)\}$. The points (1,3) and (2,3) that were used in the computation of the unique corner points (3,3) and (6,2) of Γ_{34} are marked (encircled in the figure). This marking is needed during the downward traversal of the tree to identify the module sizes, orientations, and shapes that are consistent with the computed minimum enveloping rectangle R_P.

The bounding curves of the remaining nodes are determined in a similar manner until we reach the root. The bounding curve associated with the root gives the set of points whose coordinates are the sizes of possible enveloping rectangles of the rectangular slicing structure. For this example the bounding curve of the root as shown in Figure 4.3, is $\Gamma_P = \{(5,10);(9,9);(5,6);(9,5)\}$. Since $5 \times 6 = 30$ is less than $5 \times 10 = 50$, $9 \times 9 = 81$, and $9 \times 5 = 45$, then a minimum area enveloping rectangle corresponding to the given Polish expression is a 5×6 rectangle. Now, traversing the slicing tree top-down from the root to the leaves and tracing the size/orientation

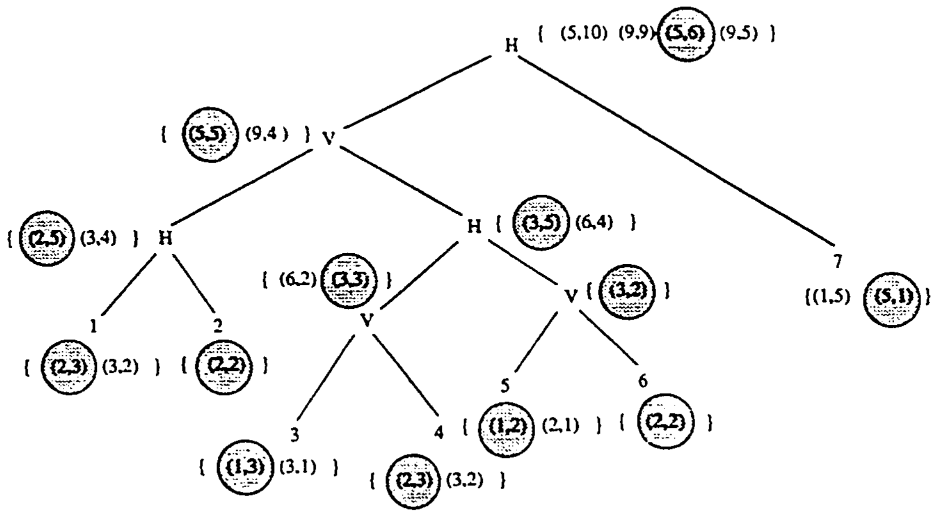


Figure 4.3: Figure showing slicing tree for area computation.

choices that led to the minimum area, we find that the modules must have the sizes and orientations indicated in Table 4.2. The floorplan with the new orientation is shown in Figure 4.2(b).

Module	(w,h)	Orientation
1	(2,3)	Original
2	(2,2)	Original
3	(1,3)	Original
4	(2,3)	Original
5	(1,2)	Original
6	(2,2)	Original
7	(5,1)	Rotated

Table 4.2: Module sizes and orientation for the floorplan solution.

Satisfying Aspect Ratio Constraints

Besides minimization of area, any user specified constraints on the aspect ratio (denoted by ρ) of the enveloping rectangle needs to be satisfied. As mentioned earlier, two positive numbers p and q denoting the lower and the upper bounds are given. At the root of the slicing tree, different (w,h) pairs of the enveloping rectangle may violate either of the bounds on ρ. If the lower bound is violated ($\rho < p$) then h needs to be incremented by some amount say a, so as to make ρ greater than or equal to p. This is shown below:

$$\begin{aligned} p &= \frac{h+a}{w} \\ a &= p \times w - h \end{aligned}$$

If the upper bound is violated ($\rho > q$) then w needs to be incremented by some amount say b, in order to make ρ less than or equal to q. This is achieved as follows:

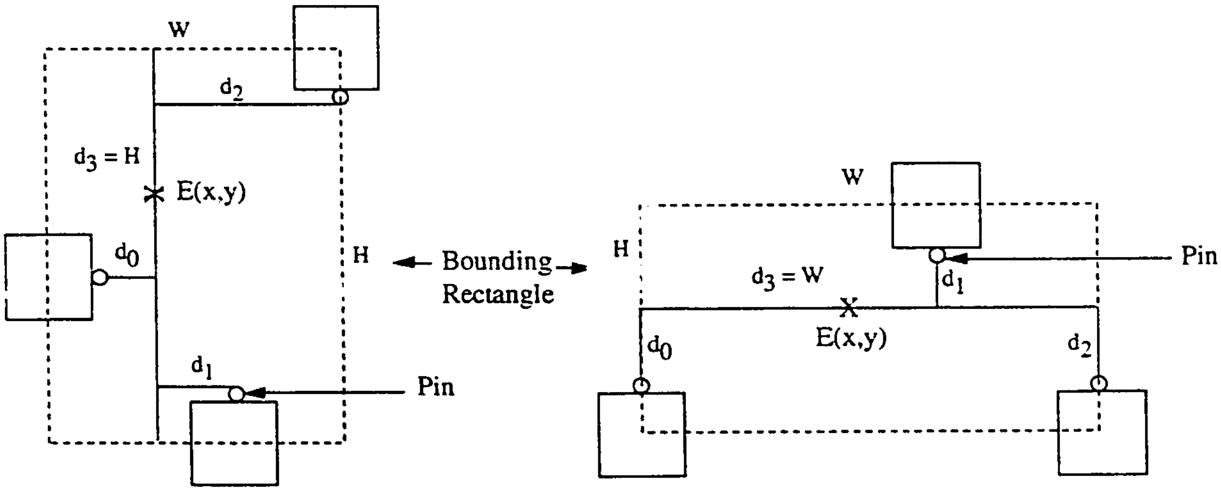
$$\begin{aligned} q &= \frac{h}{w+b} \\ b &= \frac{h}{q} - w \end{aligned}$$

This process is repeated for all the (w,h) pairs and the pair giving the minimum area is selected.

4.3 Wirelength

The overall wirelength is an estimate of the routing area. The estimated wirelength is also used in the delay analysis for computation of interconnect capacitance. Hence, accuracy of estimation method is very important. One of the most popular and simple methods is semi-perimeter estimation. In this method, the smallest bounding rectangle that encloses all the pins connected to a net is computed. The wirelength is obtained by taking half the perimeter of this rectangle. This technique works for 2-3 pins nets but underestimates the wirelength for nets consisting of more than 3 pins. Hence, a correction factor is introduced to improve the accuracy of estimation. Another possible estimate is to determine for each net the length of the minimum spanning tree which covers all the pins of a net. Then, a measure of wirelength will be the sum of the length of all these minimum spanning trees.

In this thesis, the estimation technique used is an approximation of Steiner tree method. The idea is to compute a bounding rectangle that encloses all the pins connected to a net. The height and width of this rectangle is calculated. An equilibrium point with respect to the centers of the modules connected by the net is calculated. If width W of the rectangle is greater than its height H, a horizontal line passing through the equilibrium point is drawn. Otherwise, a vertical line passing through the equilibrium point is drawn. Finally, all the pins are joined with this line. For an n-pins net, the equilibrium point $E(x, y)$ shown in Figure 4.4, is calculated



a) H > W : Vertical line through E

b) H < W Horizontal line through E

Wirelength= $d_0 + d_1 + d_2 + d_3$

Figure 4.4: Variation of Steiner tree estimation.

from the following equations.

$$x_{equilibrium} = \frac{x_1 + x_2 + x_3 + \ldots + x_n}{n} \tag{4.1}$$

$$y_{equilibrium} = \frac{y_1 + y_2 + y_3 + \ldots + y_n}{n} \tag{4.2}$$

This method results in a fairly realistic estimation of the wirelength. A comparison of different estimation methods is shown in Figure 4.5.

I/O Pins Assignment

The I/O pins of the circuit that will be connected to the I/O pads on the chip are assigned to the periphery of the chip. The chip is divided into four quadrants. Depending upon the location of the cell to which the I/O pin is connected, the I/O pin is placed at the corner of that quadrant. The following equations are used, one for each corner of the chip.

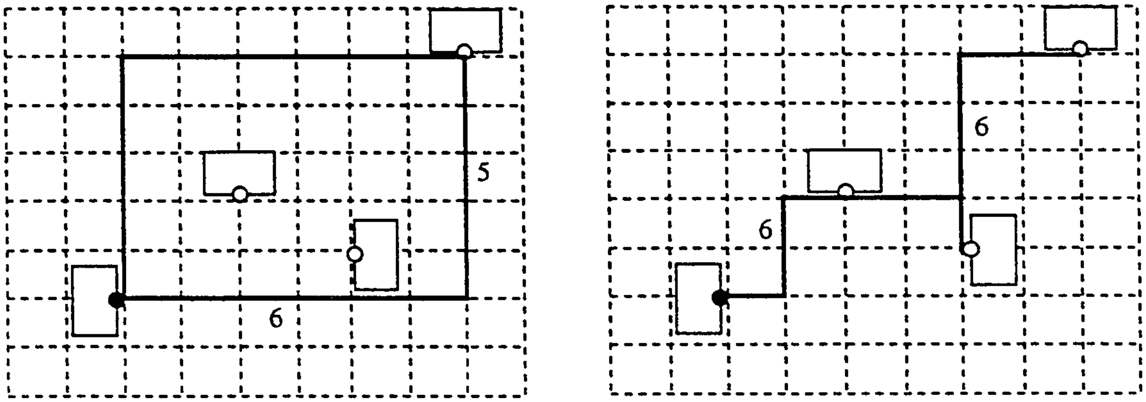
$$\text{First Quadrant} : 0 \leq x \leq \frac{w}{2} \quad \text{and} \quad 0 \leq y \leq \frac{h}{2} \quad \Rightarrow C_{x,y} = (0,0)$$

$$\text{Second Quadrant} : \frac{w}{2} \leq x \leq w \quad \text{and} \quad 0 \leq y \leq \frac{h}{2} \quad \Rightarrow C_{x,y} = (w,0)$$

$$\text{Third Quadrant} : \frac{w}{2} \leq x \leq w \quad \text{and} \quad \frac{h}{2} \leq y \leq w \quad \Rightarrow C_{x,y} = (w,h)$$

$$\text{Fourth Quadrant} : 0 \leq x \leq \frac{w}{2} \quad \text{and} \quad \frac{h}{2} \leq y \leq h \quad \Rightarrow C_{x,y} = (0,h)$$

where, w and h are the width and height of the chip, x and y are coordinates of the centre for each cell, and $C_{x,y}$ denotes the corner point of the quadrant to which the I/O pins will be assigned.

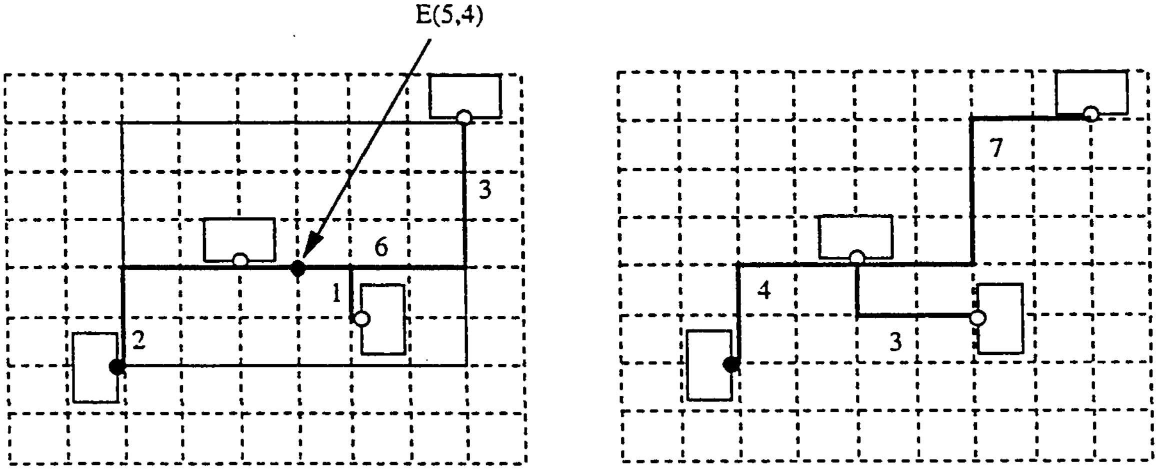


Semi-perimeter
length= 11

(a)

Steiner tree
length= 12

(b)



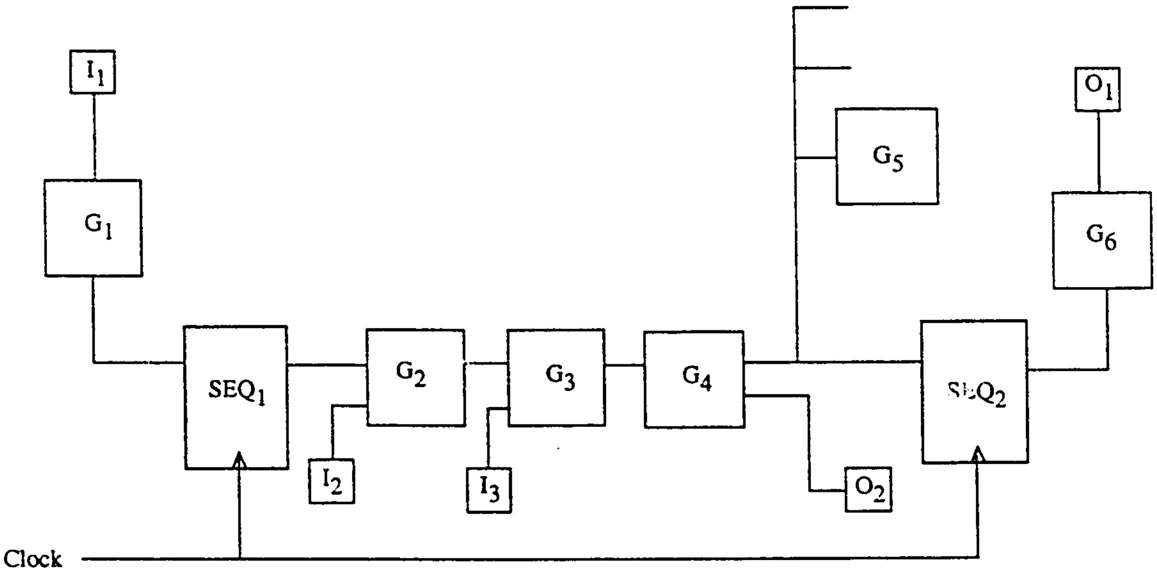
Approximate Steiner tree
length= 12

(c)

Spanning tree
length= 14

(d)

Figure 4.5: Comparison of various estimation methods.



Path 1: $I_1 \rightarrow G_1 \rightarrow SEQ_1$

Path 2: $I_2 \rightarrow G_2 \rightarrow G_3 \rightarrow G_4 \rightarrow O_2$

Path 3: $SEQ_2 \rightarrow G_6 \rightarrow O_1$

Path 4: $SEQ_1 \rightarrow G_2 \rightarrow G_3 \rightarrow G_4 \rightarrow SEQ_2$

Figure 4.6: Illustration of various paths in a sequential circuit.

4.4 Timing Issues

In VLSI floorplanning, the designer is concerned with long path timing problems. A signal starting from a source reaches a destination (sink) after traversing one or more circuit modules. A path is defined as a sequence of such circuit modules. The start point is an input pad or storage element output pin. The end point is an output pad or storage element input pin. Thus, there can be four different paths in a VLSI circuit. These paths are shown in Figure 4.7. All storage elements are *flip-flops.*

In the following section, we recall some of the most important terms used frequently in the context of timing-driven designs.

Long Path Problem

A signal traveling on a path is required to reach the path sink not later than what is known as its *latest required arrival time* (LRAT). LRAT is a function of the circuit clock period. When all the paths in a design are shorter than LRAT, the design is said to be free from long path timing problems. For the path π in Figure 4.7, there can be no long path problems when the following condition is satisfied.

$$T_\pi \leq LRAT_\pi \tag{4.3}$$

Path Slack

The slack of a path π is defined as follows:

$$Slack_{\pi} = (T_{\pi,cells} + T_{\pi,nets}) - LRAT_{\pi} \tag{4.4}$$

For a circuit to be free from long path timing problems the condition $S_{\pi} \geq 0$ must be satisfied. If, for a path π, $S_{\pi} \leq 0$, then it is said to be *critical.* A path having most negative slack among the critical paths is referred to as the *most critical.*

4.4.1 Delay Model

The delay model used in this thesis is the *Linear Model.* As per this model, it is assumed that a signal traveling from the source arrives at all the sinks at the same time. The overall delay for any path π, is given by the following equation:

$$T(\pi) = \sum_{c\in\pi} S_c + \sum_{n\in\pi} I_n \tag{4.5}$$

where S_c is the switching delay of cell c and I_n is the interconnect delay of net n. Based on the lumped RC-model, these quantities are defined as follows:

$$S_c = BD_c + LD_c \tag{4.6}$$

where BD_c and LD_c are the base delay and load delay of cell c. The base delay is obtained as shown below:

$$BD_c = LF_c \times LC_c$$

where the terms LF_c and LC_c denote the load factor of cell c and the total load capacitance at the input pins of cells driven by cell c.

The interconnect delay of net n is computed by the following equation:

$$I_n = LF_c \times C_n + C_n \times R_n + R_n \times LC_c \tag{4.7}$$

where, the terms C_n and R_n represent the capacitance and the resistance respectively of net n. The interconnect capacitance and resistance for net n is computed as given below:

$$C_n = Area\_capacitance + Fringe\_capacitance \tag{4.8}$$

where,

$$Area\_capacitance = [C_{am1} \times L_{m1} + C_{am2} \times L_{m2}] \times w \tag{4.9}$$

and

$$Fringe\_capacitance = 2 \times [(w + L_{m1}) \times Cf_{m1} + (w + L_{m2}) \times Cf_{m2}] \tag{4.10}$$

$$R_n = \frac{R_{m1} \times L_{m1} + R_{m2} \times L_{m2}}{w} \tag{4.11}$$

where,

C_{am1} = Plate Capacitance/Area of metal m_1,

C_{fm1} = Fringe Capacitance/Length of perimeter of metal m_1,

C_{am2} = Plate Capacitance/Area of metal m_2,

C_{fm2} = Fringe Capacitance/Length of perimeter of metal m_2,

R_{m1} = Sheet Resistance of metal m_1.

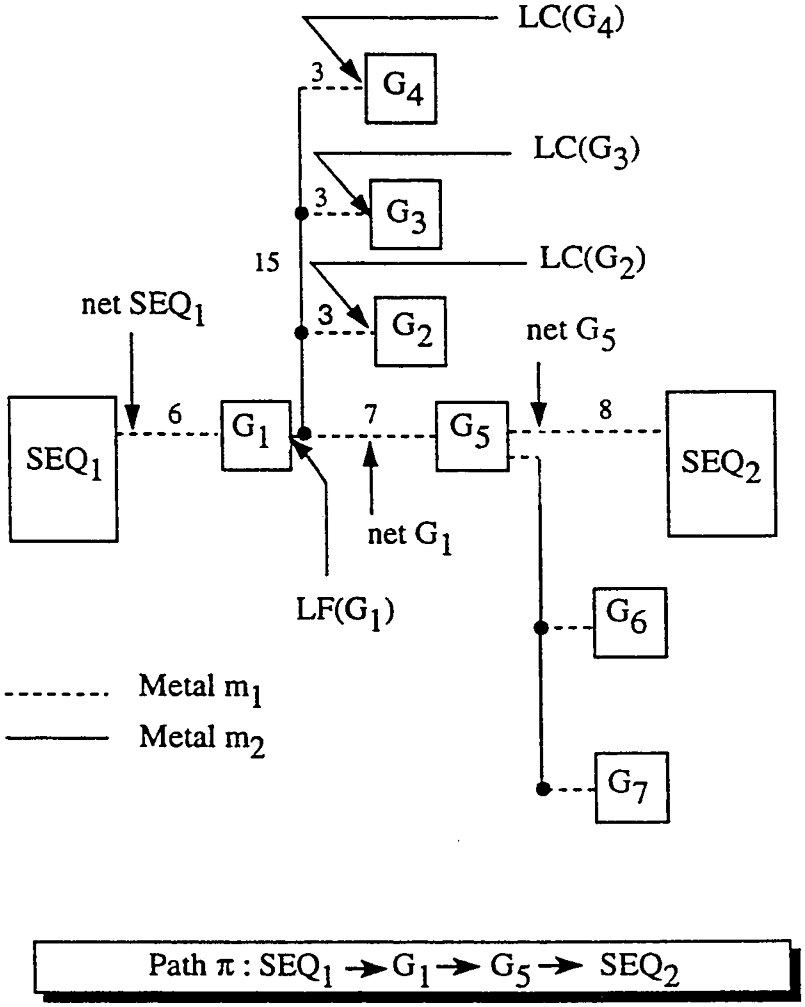


Figure 4.7: Delay model used in this work.

R_{m2} = Sheet Resistance of metal m_2,

L_{m1} = Length of metal m_1,

L_{m2} = Length of metal m_2,

w = Width of interconnects

Next, we illustrate the delay computation for the path π shown in Figure 4.7 through an example. Using Equations 4.6 and 4.7, the switching delay and interconnect delay of the path π can be obtained as follows:

$$\sum_{cell \in \pi} S_c = S_{SEQ_1} + S_{G_1} + S_{G_5} + S_{SEQ_2} \tag{4.12}$$

and

$$\sum_{cell \in \pi} I_n = I_{net_{SEQ_1}} + I_{net_{G_1}} + I_{net_{G_5}} + I_{net_{SEQ_2}} \tag{4.13}$$

The interconnect delay of some net, say net_{G1} in Figure 4.7, is computed by the following equation.

$$I_{net_{G_1}} = LF_{G_1} \times C_{net_{G_1}} + C_{net_{G_1}} \times R_{net_{G_1}} + LC_{G_1} \times R_{net_{G_1}} \tag{4.14}$$

where, the load capacitance of cell G_1 is obtained as follows:

$$LC_{G1} = LC_{G_2} + LC_{G_3} + LC_{G_4} + LC_{G_5} \tag{4.15}$$

Different values for capacitance and sheet resistance are given in Tables 4.3 and 4.4 respectively. The width of interconnects for the circuit technology used is w = 3μ.

Metal_Type	Area_Capacitance $(10^{-4}pF/\mu^2)$			Fringe_Capacitance $(10^{-4}pF/\mu)$		
Range	Min	Typ	Max	Min	Typ	Max
Metal1	0.21	0.23	0.26	0.75	0.79	0.82
Metal2	0.13	0.14	0.15	0.78	0.81	0.85

Table 4.3: Area and fringe capacitance values.

	Sheet_Resistance $(ohms/square)$		
Range	Min	Typ	Max
Metal1	0.050	0.055	0.060
Metal2	0.022	0.028	0.033

Table 4.4: Sheet resistance values.

After combining the constant terms in Equations 4.9, 4.10 and 4.11, these equations may be re-written as follows:

$$Area\_capacitance = 0.000078 \times L_{m1} + 0.000045 \times L_{m2} \tag{4.16}$$

$$Fringe\_capacitance = 0.000164 \times L_{m1} + 0.000170 \times L_{m2} + 0.001002 \tag{4.17}$$

$$R_n = 0.020 \times L_{m1} + 0.011 \times L_{m2} \tag{4.18}$$

After combining all the constant terms, the new equations can be written as follows:

$$C_n = CONST_1 \times L_{m_1} + CONST_2 \times L_{m_2} + CONST_3 \tag{4.19}$$

and,

$$R_n = CONST_4 \times L_{m_1} + CONST_5 \times L_{m_2} \tag{4.20}$$

where L_{m_1} and L_{m_2} are the lengths of metals m_1 and m_2 respectively. The new constant terms are,

$\mathrm{CONST}_1 = 0.000242\mathrm{pF}/\mu$

$\mathrm{CONST}_2 = 0.000215\mathrm{pF}/\mu$

$\mathrm{CONST}_3 = 0.001002\mathrm{pF}$

$\mathrm{CONST}_4 = 0.020\Omega/\mu$

$\mathrm{CONST}_5 = 0.011\Omega/\mu$

From Figure 4.7, L_{m_1}=16μ and L_{m_2}=15μ. Let the values of load capacitance for the cells G_2, G_3, G_4 and G_5 be as given below:

$\mathrm{LC}_{\mathrm{G}_2} = 0.068\mathrm{pF}$,

$\mathrm{LC}_{\mathrm{G}_3} = 0.085\mathrm{pF}$,

$\mathrm{LC}_{\mathrm{G}_4} = 0.091\mathrm{pF}$,

$\mathrm{LC}_{\mathrm{G}_5} = 0.087\mathrm{pF}$,

and the load factor be,

$\mathrm{LF}_{\mathrm{G}_1} = 4.92\mathrm{ns/pF}$,

Then the load capacitance of the driving cell G_1 will be $\mathrm{LC}_{\mathrm{G}_1} = 0.331\mathrm{pF}$. The interconnect capacitance and resistance are found to be $\mathrm{C}_{\mathrm{net}_{\mathrm{G}_1}} = 0.008099\mathrm{pF}$, and $\mathrm{R}_{\mathrm{net}_{\mathrm{G}_1}} = 0.485\Omega$ respectively. By substituting these values in Equation 4.14, the interconnect delay through net_{G_1} is obtained as follows.

$$\mathrm{I}_{\mathrm{net}_{\mathrm{G}_1}} = 4.92 \times 0.008099 + 0.008099 \times 0.485 + 0.485 \times 0.331$$

which is equal to 0.20431 *ns*.

Based on the above calculations and experimental results obtained from actual VLSI circuits, it was observed that the load delay component contributes the most to the interconnect delay. The contribution due to interconnect resistance was found to be ten times smaller than that due to load factor. Before proceeding further, we show that it is safe to ignore the effect of interconnect resistance in the delay computation by the following example.

Let us assume a chip of size $4000 \times 4000\ \mu m^2$ and the interconnect length for a 2-pin net to be $L_{m1} = 2000\mu$ and $L_{m2} = 2000\mu$ for horizontal and vertical connections respectively. Then, the interconnect resistance will be:

$$R_n = 0.02 \times 2000 + 0.011 \times 2000 = 62\Omega$$

Now, let us compute the various components of interconnect delay assuming $LC_c = 0.051pF$, $LF_c = 4.92K\Omega$ and $C_n = 0.012pF$.

$$LF_c \times C_n = 4.92 \times 0.012 = 0.05904ns$$

$$R_n \times LC_c = 62 \times 0.051 = 0.003162ns$$

$$R_n \times C_n = 62 \times 0.012 = 0.000744ns$$

It is evident from the above computation that the delay due to interconnect resistance is very negligible and can be ignored.

4.4.2 Critical Paths

As the number of paths in a VLSI circuit can be in the millions, it is not feasible to enumerate all the paths [51]. The idea is to distinguish a very small subset out of all the paths in such a way that the subset of paths identified, consists of all or a major portion of the paths that will suffer from long path timing problems after layout. In the following paragraphs, we briefly discuss the procedure used to predict the timing critical paths. A detailed description of the timing analyzer/predictor is given in [7].

From past layouts of circuits with similar complexity, the average and standard deviation of net lengths are estimated for each type of net (2 pin-, 3 pin-,...., k pin-nets). These are converted to capacitances for the particular technology of the design at hand. The delay variance on a path π is given by:

$$D_\pi^2 = \sum_{i=1}^{k} LF_i^{\,2} \cdot D_i^{\,2} \tag{4.21}$$

where $D_i^{\,2}$ denotes delay variance on the capacitance of a net i as obtained from the layouts of previous designs.

Let T_{max} be the estimated delay of the longest path in a circuit i.e.,

$$T_{max} = Max(T_\pi) \tag{4.22}$$

A path π is said to be $\alpha - critical$ iff:

$$T_\pi + \alpha \cdot D_\pi \geq T_{max} \tag{4.23}$$

The parameter α [1] above is referred to as the *confidence factor*. The larger the value of α, the more will be the number of paths notified as critical and the higher will be the possibility of identifying most of the prospective critical paths. The $\alpha - critical$ paths will be referred to as the most critical paths.

After layout, the interconnect capacitances are calculated for all the nets and the interconnect delays are computed. Next, actual delays and slack values for each of the most critical paths are computed. The performance of timing-influenced floorplanning depends largely on the accuracy of timing analysis.

4.4.3 Performance Metric

There can be many ways to incorporate timing criterion in the cost function. In this work, the difference between T_{clock} (clock value used for the enumeration of critical paths) and the worst slack among the most critical paths (Π) is used as an objective. We denote this difference by the term T_{period}. For a floorplan solution, the speed of operation can be obtained by subtracting the worst slack among the paths predicted as critical from T_{clock}. This is shown below:

The objective is to minimize $T_{period}(max)$. Its minimum value will determine the speed at which the circuit can run.

---

[1] Desirable values of α are ≤ 3 or 4.

```
Begin
    $T_{period}(max) = 0$
    For each path $\pi \in \Pi$
        Do
            $T_{period} \leftarrow T_{clock} - S_{\pi}$
            $T_{period}(max) \leftarrow Max(T_{period}(max), T_{period})$
        EndDo
    EndFor
End
```

Figure 4.8: Computation of speed performance for a floorplan solution.

4.4.4 Fabrication Technology

The VLSI circuits used in our experiments are implemented in a 2μ CMOS technology. The timing characteristics and other parameters such as dimensions etc., are given in [52]. It is assumed that metal m_1 is used for routing horizontal tracks, whereas metal m_2 is used for vertical tracks. The values of capacitance and resistance for these metals are adopted from the corresponding fabrication foundry manual [53] and shown in Tables 4.3 and 4.4 respectively.

4.5 Implementation Details of GIFT

Floorplan design involves a number of conflicting objectives which are not easy to combine in a single cost function. The first objective is to allocate optimal rectangular regions to cells so as to minimize the area of the bounding rectangle and dead space. The second objective is reduction of the total wirelength denoting the wiring space requirement. Since the wiring consumes a high percentage of the total area, it is very important to have an early estimate of the wiring space. The third objective is concerned with the speed performance of the circuit.

A flowchart of various steps involved in the implementation of GIFT is shown in Figure 4.9. A translator program reads the VLSI circuit in VPNR format and builds the circuit graph and the connectivity. Some intermediate files are also generated that enable the determination of the number of I/O pins, the number of circuit modules, and the area of each module etc. The timing analyzer program (TA) uses intermediate files and generates the most critical paths for the user specified value of clock and confidence factor α. For each of the predicted critical paths, the timing information consists of the total cell delay, delay bounds on each net comprising that path, and the latest required arrival time (LRAT). This information is used to compute the value of interconnect delay of paths. The actual propagation delay is determined after placement of the circuit and compared to the estimated value to check if the circuit is violating any timing constraints.

The information provided by the translator program is used to generate Polish strings representing feasible floorplan solutions for the circuit under consideration. The Polish strings are used to determine area, center, and corners of the floorplan enveloping rectangle, besides computation of centers for each module. After estimation of the overall wirelength, propagation delays of the nets are computed using the delay model described earlier. This gives the actual value of the clock. For

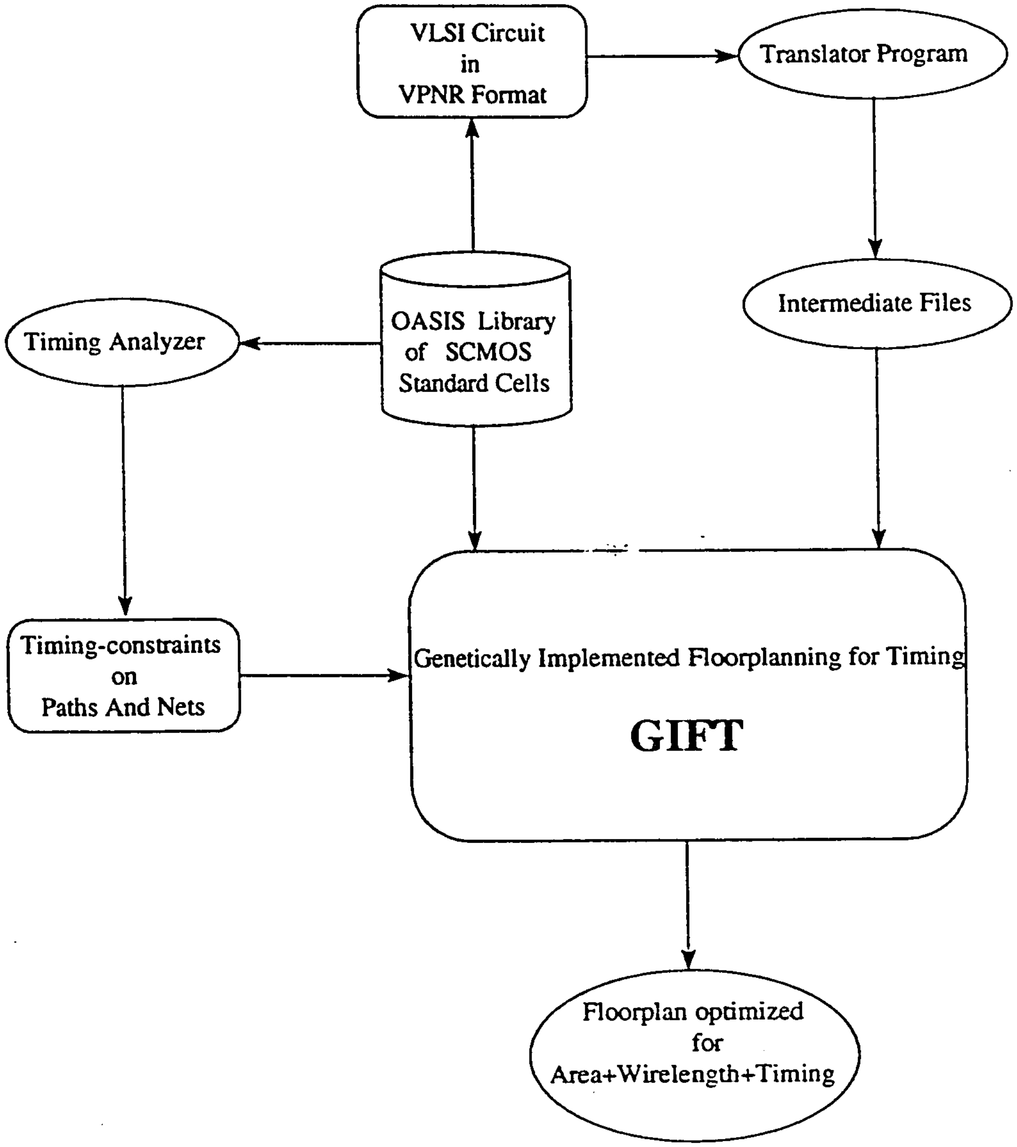


Figure 4.9: Various steps of the GIFT.

each slicing floorplan solution in the population, the three objectives namely area, wirelength and the clock value are output by the floorplanner. The output of GIFT consists of the following information.

- a Polish expression corresponding to the "optimal" floorplan,
- the orientation of each floorplan module,
- the location of each circuit module, and
- the area of the bounding rectangle, the total estimated wirelength, and the speed of the chip.

4.6 Genetic Modeling of Floorplanning

This section discusses various aspects involved in the genetic algorithm based floorplanning approach. These aspects include parameters such as crossover and mutation probabilities, fitness function, and important decisions namely size of the population, the selection scheme for crossover, selection of offsprings for next generation etc. These aspects greatly control the performance of GIFT. These issues are discussed in the following section.

4.6.1 Initial Population (Υ)

Two types of initial population generators were investigated that attempt to produce valid Polish strings. They are:

(1) Generator Υ_1, constructs a Polish string by inserting $n-1$ operators in a random permutation of n; (2) Generator Υ_2, generates a Polish string in such a way that a number of modules are placed in rows.

4.6.2 Fitness Function (Φ)

The cost function consists of three terms that aim at improving the area of floorplan bounding rectangle, the timing performance and the overall interconnection length.

The cost function is transformed to obtain the fitness function such that higher values of fitness means low cost, as is the case with any minimization problem. The fitness may be obtained from the reciprocal of the cost. However, this might cause the range of the fitnesses to be too small requiring a further step of fitness scaling. Another way is to calculate the fitness by subtracting the cost from a constant, which must be large enough to produce positive fitness values. But if it is too large, this may again cause the range of fitnesses to be very small. For these reasons, we have used a normalized fitness value (that varies with time). The fitness of an individual is obtained as explained in the following paragraphs.

For a population of size n and a floorplan solution i for $1 \leq i \leq n$:

$$\begin{aligned} Area(i) &= \text{area of floorplan bounding rectangle} \\ Clock(i) &= \text{required clock period that satisfies timing constraints} \\ Wirelength(i) &= \text{overall estimated wirelength} \end{aligned}$$

The cost of a floorplan solution i is obtained by the following equation:

$$Cost(i) = \frac{Area(i)}{A} \times W_1 + \frac{Clock(i)}{C} \times W_2 + \frac{Wirelength(i)}{W} \times W_3 \tag{4.24}$$

where,

$$\begin{aligned} A &= \max_{i \in (1 \ldots n)} Area(i) \\ C &= \max_{i \in (1 \ldots n)} Clock(i) \\ W &= \max_{i \in (1 \ldots n)} Wirelength(i) \end{aligned}$$

W_1, W_2 and W_3 are relative weights assigned to each term. Since the terms in Equation 4.24 are incompatible, they are normalized to the same mean and standard deviation to obtain a meaningful fitness as shown below:

$$X_{norm}(i) = \overline{X}_{norm} - (X(i) - \overline{X})\frac{\sigma_{norm}}{\sigma} \tag{4.25}$$

where,

X = Value of the term (area, clock, or wirelength) prior to normalization

$\overline{X}$ = Mean of the term prior to normalization

$\overline{X}_{norm}$ = Desired mean of the term after normalization

σ = Standard deviation of the term prior to normalization

σ_{norm} = Desired standard deviation of the term after normalization

The values of $\overline{X}_{norm}$ and σ_{norm} were set equal to 1000 and 100 respectively. The fitness of an individual i is computed as follows:

$$Fitness(i) = \frac{Area_{norm}(i)}{A_{max}} \times W_{area} + \frac{Clock_{norm}(i)}{C_{max}} \times W_{time} + \frac{Wirelength_{norm}(i)}{W_{max}} \times W_{wire} \quad (4.26)$$

The subscript *norm* indicates that the values are normalized and *max* denotes maximum values of the corresponding terms. The constants A_{max}, C_{max} and W_{max} in Equation 4.26 have the following values (see Table 4.5).

$$A_{max} = 1121$$

$$C_{max} = 1118.44$$

$$W_{max} = 1135.42$$

Now, the fitness value of an individual say 1, can be computed as follows:

$$Fitness(1) = \frac{858}{1121} \times W_1 + \frac{981.40}{1118.44} \times W_2 + \frac{959.67}{1135.42} \times W_3$$

Assuming the weight values to be W_1=0.40, W_2=0.30, and W_3=0.30, then

$$Fitness(1) = 0.83$$

	Actual_Quantities			*Normalized_Quantities*		
Floorplan(i)	*Area(i)*	*Clock(i)*	*Wirelength(i)*	*Area(i)*	*Clock(i)*	*Wirelength(i)*
1	428240	21.83	20095.16	858	981.40	959.67
2	425008	22.02	20510.00	908	850.70	847.28
3	416592	21.66	19654.50	1039	1102.54	1079.06
4	414488	21.88	20025.50	1071	946.90	978.54
5	411280	21.63	19446.46	1121	1118.44	1135.42
MEAN	419121.60	21.80	19946.32	1000	1000	1000
SDEV	6436.68	0.14	369.09	100	100	100
MAX	428240	22.02	20510.00	1121	1118.44	1135.42

Table 4.5: An illustration of normalization process for population of 5 floorplans.

The quality of floorplan solution largely depends upon the values of weights. By varying the weight values, the notion of optimality can be changed. For example, if the weight of the clock term is increased to a value greater than that for the area term, then the search will be biased toward floorplan solutions with better timing characteristics than area.

4.6.3 Selection Mechanism for Crossover (η)

This step determines which parents are selected for crossover. In this work we used the selection mechanism based on the *proportionate selection* scheme implemented by the roulette wheel method [6]. In this selection scheme, each string is allocated a sector of a roulette wheel with the angle subtended by the sector at the center of the wheel that equals $\frac{2\pi f_i}{\sum_j f_j}$ where f_i is the fitness of solution i, and $\sum_j f_j$ is the sum of the fitness over the entire population.

4.6.4 Selection Mechanism for Next Generation (ξ)

A number of methods have been proposed to select individuals that can survive and be used in the next generation [6].

- replace-all (ξ_1)
- replace-all except the best (ξ_2)
- combine old population with the offsprings, save best and the rest are chosen probabilistically (ξ_3)
- combine old population with the offsprings, save the overall best and the best with respect to important objectives, and the rest are chosen probabilistically (ξ_4)

The selection schemes ξ_1, ξ_2 and ξ_3 are straightforward. In the last scheme, besides preserving the individual with the best fitness, other individuals that are best with respect to the important objectives such as area, timing, and wirelength are also preserved and advanced to the next generation. In all the schemes, the individuals are chosen probabilistically based on their fitnesses (higher fitness translates to higher survival probability). After experiments, the last technique was found to be better than the others and was adopted in the final experimentation.

4.6.5 Good Genetic Operators

In the initial stages of this work, extensive experimentation was done on the "Highway" test circuit with the purpose of determining the efficacy of different genetic operators by considering these operators one by one in the genetic algorithm. All the crossover operators except "cycle" crossover and all the mutation operators described in the previous chapter were found useful.

The results of above experiments indicated that some genetic operators are better than the others. This information was used in assigning probability values to these operators. The crossover is always performed i.e., the sum of the probabilities of individual crossover operators is 1. Whenever a crossover is to be performed, a random number is generated and depending upon the range in which the number lies, the corresponding crossover operator is invoked. Similarly, all the mutation operators are assigned different probabilities that sum up to 50.5%. This implies that the mutation may or may not be performed.

In the later stages of this work, all experiments were carried out by randomly selecting a genetic operator from the set of crossovers and mutation operators. The results show that the set of operators as a whole perform better than the case when single operators were used. The percentage probabilities of the crossovers and mutations are shown in Tables 4.6 and 4.7 respectively. In the next section, we describe the algorithm used to refine the slicing floorplan produced by the GIFT

Crossover	Probability
λ_1	15%
λ_2	15%
λ_3	20%
λ_4	20%
λ_5	30%

Table 4.6: A summary of the probabilities used for various crossover operators.

Mutation	Probability
μ_1	10%
μ_2	0.5%
μ_3	10%
μ_4	30%

Table 4.7: A summary of the probabilities used for various mutation operators.

floorplanner.

4.7 Floorplan Refinement Phase

In this phase, we remove the slicing restriction on the floorplan structure. Also, modules are allowed to have flexible shapes. The floorplan refinement procedure is due to [7]. In this section, we briefly summarize the main steps of this procedure. The refinement phase consists of two steps:

1. construction of constraint set, and
2. shape optimization.

4.7.1 Graph Construction

Two directed acyclic graphs are used to model the topological constraints between the blocks: a horizontal constraint graph G_H and a vertical constraint graph G_V. The vertex set of G_H is the set of blocks plus two dummy vertices: L, R. Similarly, the vertex set of G_V is the set of blocks plus two dummy vertices: T, B. The dummy vertices L, R, T, B correspond to the left, right, top, and bottom boundaries respectively of the layout. The edge set of G_H models the to-the-left/to-the-right relationships, while that of G_V models the on-the-top/on-the-bottom relationships.

G_H and G_V are constructed as follows. Two blocks are constrained if the center of one block must be to the left/below the center of the other. Vijayan and Tsay [54] introduced the notions of *completeness* and *strong completeness* for a topological constraint set. A constraint set is *complete* if there exists a directed path between every pair of blocks b_i, b_j either in G_H, in G_V, or both. In a *strong complete* set each pair of blocks is *adjacent* either in G_H, in G_V, or both. Two blocks are adjacent if they are connected by an edge. It is clear that a floorplan that satisfies a strongly complete set will have no overlaps. To avoid the overlap of two blocks, only one constraint (either in the horizontal or vertical direction) is necessary and sufficient. Two blocks are called *overconstrained* if they are constrained in both the horizontal and vertical directions. The existence of overconstrained blocks negatively affects the area optimality of the floorplan.

Definition 2 *A constraint set* (G_H, G_V) *is sufficiently constrained if there exists an edge between every pair of blocks* (b_i, b_j) *either in* G_H *or in* G_V.

Clearly, a *sufficiently constrained* set (G_H, G_V) is a strongly complete set. The approach in [54] starts with an overconstrained set, i.e. a set with many overconstrained blocks. This set is then reduced to a *sufficiently constrained* set by removing redundant constraints from only the longest paths in G_H and G_V. A problem with this approach is that the size of the overconstrained set could be very large, and hence may require large computing resources.

In this work, we use the more efficient strategy suggested in [7] which builds directly a *sufficiently constrained* set using a constructive (greedy) procedure. If two blocks (b_i, b_j) are overconstrained then the edge (i, j) is deleted from the graph G_H if the longest path traversing (i, j) is longer than the longest path traversing (i, j) in G_V. Otherwise the edge (i, j) is deleted from G_V. Therefore, the selection is based on which of the constraints will lead to a smaller-area floorplan. If two blocks are constrained in only one direction (i.e., they have the same x or y coordinate), the algorithm in this case has only one choice. This process generates a constraint set (i.e., G_H, G_V) according to Definition 2 and, at the same time, eliminates all redundant constraints right from the beginning. This is in contrast to the algorithm in [54], where only redundant edges belonging to the critical paths in G_H and G_V are examined. Removing all redundant constraints produces a more compact floorplan.

The next step in this refinement phase consists of resizing the variable-shape blocks in order to optimize the floorplan area and satisfy the remaining constraints on block/chip aspect ratios.

4.7.2 Block Reshaping

The reshaping algorithm determines dimensions and positions of the blocks so that the floorplan area is minimized and constraints on block shapes are satisfied. This is achieved by iteratively reshaping flexible blocks on the longest paths.

The *reshaping* algorithm utilizes the constraint graphs G_H and G_V to compute the dimensions of the floorplan and decide on a candidate block for resizing.

Suppose we want to reduce the floorplan dimension in the Y-direction without enlarging the floorplan in the X-direction. This can be achieved as follows. let $\ell(\pi_H)$ and $\ell(\pi_V)$ be the length of the *longest* paths in G_H and G_V respectively. Let block b_i be such that $b_i \in \pi_V$ and $b_i \notin \pi_H$. If π_H^i is the longest path traversing b_i in G_H, then the width w_i of b_i can be increased by an amount $\delta_i^x = \ell(\pi_H) - \ell(\pi_H^i)$ without increasing the overall area of the floorplan. δ_i^x is the maximum block's width increment that is guaranteed not to cause an increase in the length of the critical path π_H in G_H. But, since the block width has an upper bound w_i^{max}, then the legal δ_i^x is given by,

$$\delta_i^{x_{legal}} = \min(\delta_i^x,\ w_i^{max} - w_i) \tag{4.27}$$

Thus, the new dimensions w_i' and h_i' for block b_i are derived as follows:

$$w_i' = w_i + c \times \delta_i^{x_{legal}} \tag{4.28}$$

$$h_i' = a_i / w_i' \tag{4.29}$$

where c is a user specified positive real number ($0 < c \leq 1$) used to control how large the *x-increment* should be. Resizing the blocks in small increments helps achieve a smaller floorplan with the correct aspect ratio. In this work, we set this parameter to 0.5. Optimization in the X-direction is similarly formulated.

The resizing process is terminated if there are no more blocks that can be selected for reshaping

After completing the resizing process, the horizontal and vertical graphs are traced to determine the final xy-locations of the blocks. The lower left corner of the floorplan is at origin (0,0). The lower left corner of block b_i is placed at (x_i, y_i) where x_i is the longest L-to-b_i path in G_H and y_i is the longest B-to-b_i path in G_V.

Finally, the blocks are enclosed inside the smallest bounding rectangle with the desired aspect ratio ρ. The area of the bounding rectangle is the area of the floorplan.

4.8 Experimental Results

The genetic algorithm is a very elaborate and relatively hard to tune algorithm. It is harder than other iterative heuristics such as simulated annealing [17] and simulated evolution [47]. The tuning of genetic algorithm parameters to a particular problem requires extensive experimentation.

There are several key parameters that affect the performance and behavior of GA. These are: (a) size of the population, (b) initial population constructor, (c) selection mechanism for crossover, (d) type of crossovers and their probabilities, (e) mutation operators and their probabilities, (f) selection of individuals for next generation, and finally (g) the fitness function. These parameters are selected based on experimentation. The size of the population depends on the size of the problem (number of blocks). It is recommended to use a large population (30) for small circuits (up to 30 blocks). For larger circuits a population size between 10 and 20 was used. The current implementation applies all crossovers and mutation operators with the probabilities given in Tables 4.6 and 4.7 respectively.

The experiments were performed with five test circuits of sizes varying between 15 and 125 modules both for the standard-cell and the general-cell library. For each circuit, the floorplanner is supplied with a set of the predicted most critical paths. As explained in Section 4.6.2, the fitness function is a weighted sum of the area of the floorplan bounding rectangle, the wirelength of the interconnects, and the circuit clock period.

The results of experiments on the test circuits are tabulated in Tables 4.8 and 4.9 for standard-cell and general-cell respectively. It may be noted that a different cell library is used for experiments with the general-cell case. This library has the same timing characteristics as that of the standard-cell, however, the cell dimensions and thus the area are different. The plots are drawn for three different cases. In the first case, the fitness is a function of *floorplan_area* alone. The behavior of the other two objectives namely overall wirelength and timing performance is plotted against the number of generations. The best solution found is used in the initial population for the remaining two cases. In the second case, 50% weight is assigned to area and wirelength each whereas timing is ignored. The behavior of all three objectives is plotted against the number of generations. The best solutions found in the previous two cases are used in the initial population for the third case. In the third case, all three objectives are combined in the fitness and the weight assignment is 50% for area and 25% each for wirelength and timing objectives.

The experimental results of the "Highway" circuit consisting of 45 modules are plotted against the number of generations. Figures 4.11, 4.12 and 4.13 show the variations in area, wirelength and speed performance for two cases. The part (a) of these plots depict the variation when the fitness function included area objective only. The other two objectives were ignored. The part (b) of these plots show the variation when all three objectives are given weights of ($W_{\text{area}} = 0.5$, $W_{\text{clock}} = 0.25$, $W_{\text{wire}} = 0.25$.)

We tabulate the results corresponding to three different weight assignments. In the first column, 'Area Only', the fitness includes only the area term (wirelength and clock period are given zero weights). In the second column, 'Area+Wire', the fitness

Test Circuits	No. of blocks	Max Delay	Area Only			Area + Wire			Area+Wire+Time		
			Area	Wire	Clock	Area	Wire	Clock	Area	Wire	Clock
Parity1	15	33.2	34800	3538	38.2	34800	3422	37.9	34800	3422	37.2
Parity2	20	36.8	42320	5051	45.8	41088	4579	44.2	47580	4091	42.0
Parity3	30	27.1	43616	7573	46.4	43616	6514	43.3	43616	6124	39.4
Highway	45	15.540	104690	14527	29.3	102200	13805	31.2	108016	13013	25.2
Fract	125	34.1	322392	82299	86.2	319680	76100	78.2	319680	76140	74.4

Table 4.8: Results of experiments considering different objectives for standard-cell circuits. Column 'Max Delay' contains the delay due to logic elements only. Column 'Clock' gives the delay of the longest path which includes logic and interconnect delays.

includes the area and wirelength term with equal weights (50% each). Note that the lower bound on the clock period is given by the maximum path delay due to logic only (column Max Delay in the tables). In the third column, 'Area+Wire+Time', the fitness includes all terms with weights 50%, 25% and 25% for area, wirelength, and clock period respectively.

For the standard-cell circuits, it may be observed that for all the test cases (Table 4.8), the wirelength has consistently improved. A decrease in wirelength between 5% and 20% was achieved at the expense of a slight increase in area of the bounding box by only 3%.

In the third case we observed a maximum decrease in the interconnect delay by a factor of about 43%. For example, for the "Parity2" circuit, the clock period when area was the only objective was 45.87 nano seconds. Since the logic delay on the path is 36.83 nano seconds, the delay due to interconnects is 9.04 nano seconds. When timing was included in the fitness function, the delay due to interconnect was reduced to 5.17 nano seconds. Thus a reduction in the interconnect delay by 42.8% was achieved. The increase in the area of the bounding rectangle in this case when all terms are weighted is up to a maximum of 12%. Note that the dead space introduced in this stage is further reduced by the second phase of the floorplanner.

For the general-cell circuits also, the wirelength has consistently improved for all the test cases (Table 4.9). A decrease in wirelength between 9% and 30% was achieved without any increase in the area of the bounding box.

In the third case we observed a decrease in the interconnect delay between 20% and 75%. For example, for the "Ckt1" test circuit, the clock period when area was the only objective was 37.2 nano seconds. Since the logic delay on the path is 33.2 nano seconds, the delay due to interconnects is 4.0 nano seconds. When timing was included in the fitness function, the delay due to interconnect was reduced to 1.0 nano second. Thus a reduction in the interconnect delay by 75% was achieved. For all the test circuits considered, the reduction in the wirelength and interconnect delay does not result in increase in the area of the bounding rectangle (except for "Ckt4" where the area increase is less than 1%). For the circuit "Ckt5", the area of the bounding rectangle in the third case is reduced by a factor of 8% as compared to the first case.

It is further observed that the overall quality of the optimal floorplans for general-cell circuits are better than that of the standard-cell circuits. Specifically, the wasted space (area of the bounding box and the wiring space) in the general-cell floorplans

Test Circuits	No. of blocks	Max Delay	Area Only			Area + Wire			Area+Wire+Time		
			Area	Wire	Clock	Area	Wire	Clock	Area	Wire	Clock
Ckt1	15	33.2	14605	2561	37.2	16100	2295	36.5	14605	2334	34.2
Ckt2	20	36.8	19712	3235	42.1	20670	3014	41.6	19712	2668	40.7
Ckt3	30	27.1	30880	6565	41.9	29295	5630	40.1	30652	5216	39.1
Ckt4	45	15.5	40186	10377	26.0	39712	7713	23.8	40425	7215	21.9
Ckt5	125	34.1	160114	59788	70.4	150150	56806	68.6	147232	51251	60.6

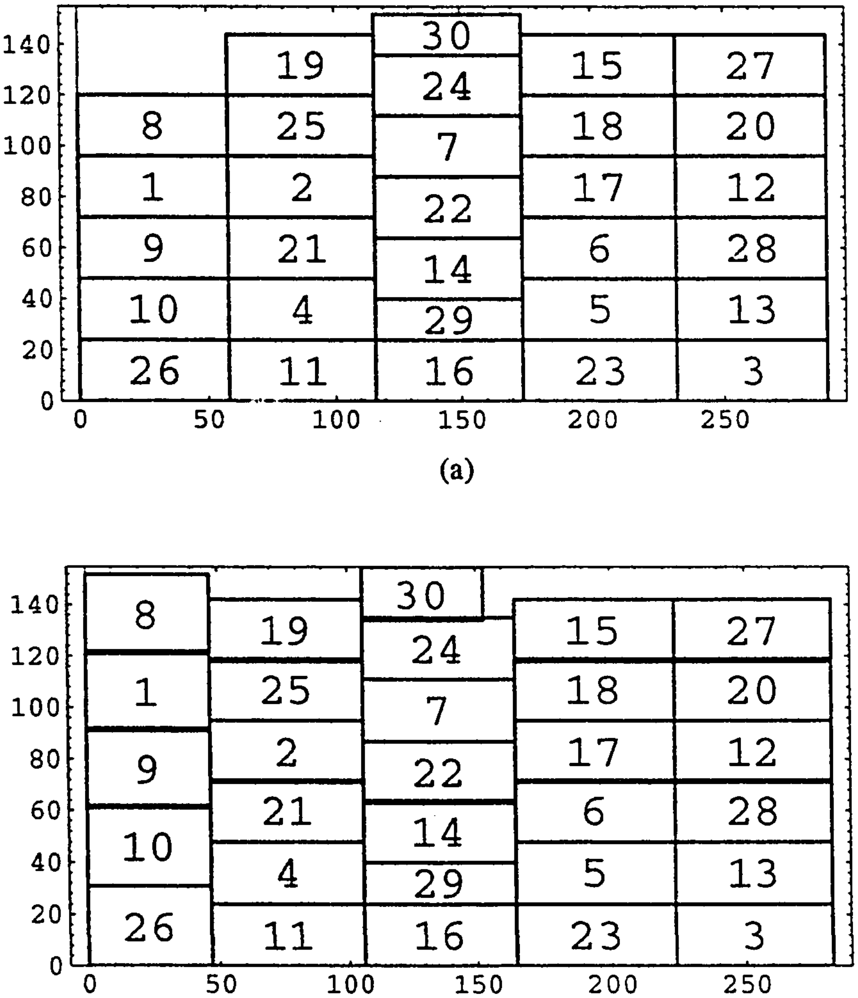
Table 4.9: Results of experiments considering different objectives for general-cell circuits. Column 'Max Delay' contains the delay due to logic elements only. Column 'Clock' gives the delay of the longest path which includes logic and interconnect delays.

is smaller than that in the standard-cell floorplans.

The optimal floorplans for the test circuits "Highway" and "Ckt4" are shown in Figures 4.14, and 4.16 when area was the only term in the fitness. The Figures 4.15 and 4.17 show the optimal floorplans when all the three terms were in the fitness.

As an example, the effect of the refinement phase is illustrated in Figure 4.10 for the floorplan of "Parity3" test circuit. The relative positions of the blocks are retained but the area of the bounding box has been reduced further by about 8%.

In these experiments the interconnect delays were inflated by a factor of 3 in order to make the timing aspects more pronounced. Current implementation does not take into account routability or pin location issues. The reason is that routability is usually not a problem for the general cell design style and can be easily incorporated into our system by integrating a global router with the floorplan sizing procedure of the second phase. The task of the global router is to compute routing space requirements between blocks. The space requirements can be specified as edge weights in



(a)

(b)

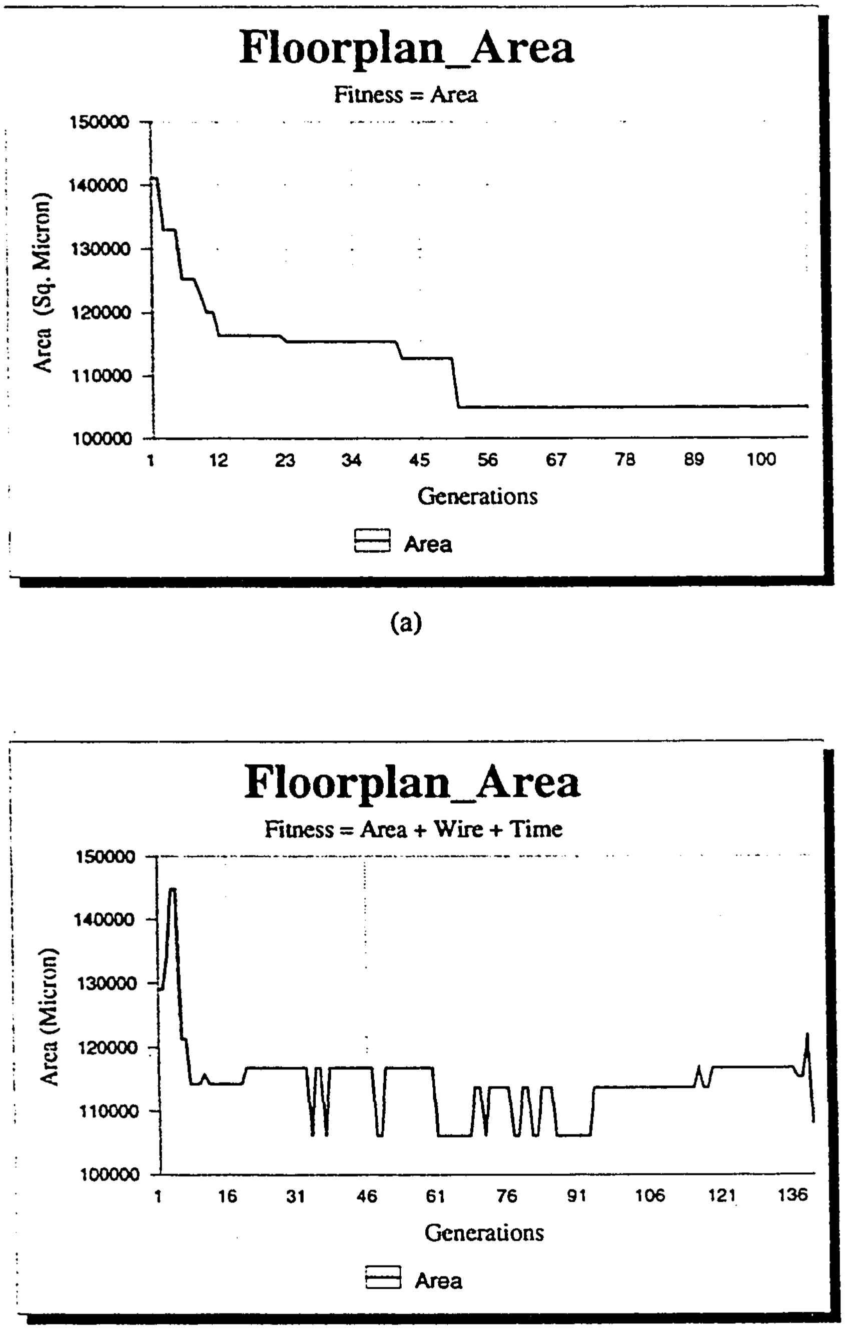
Figure 4.10: (a) floorplan for the test circuit "Parity3" obtained after phase-1 (Area=43,500); (b) floorplan obtained after phase-2 (Area=40,186).

the constraint graphs G_H and G_V. Then, the algorithm will use the edge and node weights to compute correct locations of all the blocks as well as the final dimensions of the floorplan.

4.9 Conclusion

In this chapter, implementation details of a timing-driven floorplanner were described. The floorplanner works in two phases: a simple representation is used during the first phase to solve a difficult problem, and a constructive technique is used during the second phase to refine further the floorplan solution in terms of area objective while easing the restrictions of the first phase. Timing constraints are included in the cost function.

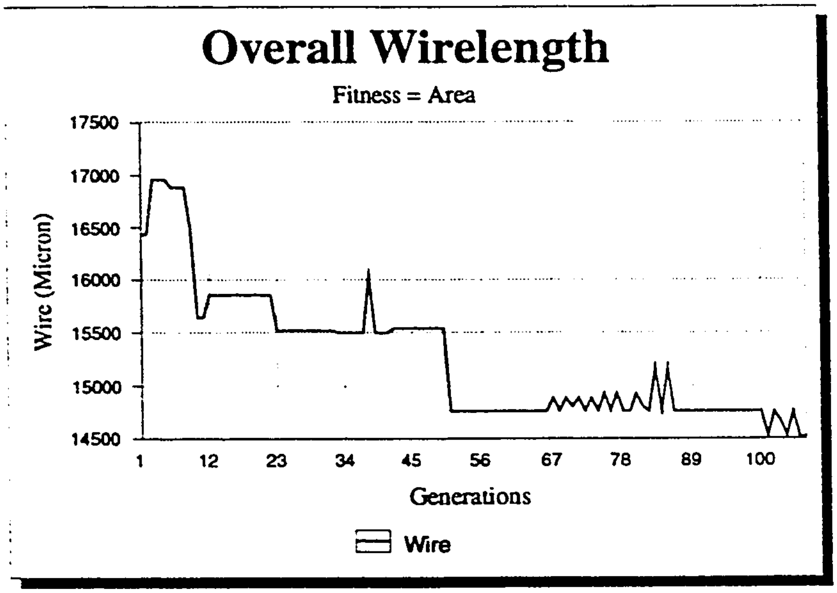
The floorplanning procedure follows the genetic paradigm. The program uses net and path timing data from a timing analyzer. The timing analyzer uses a new criterion (α-*criticality*) to predict the critical paths prior to floorplanning. Extensive experiments were conducted to tune the parameters of the program and evaluate the extent of timing improvement. The experimental results were presented and discussed.



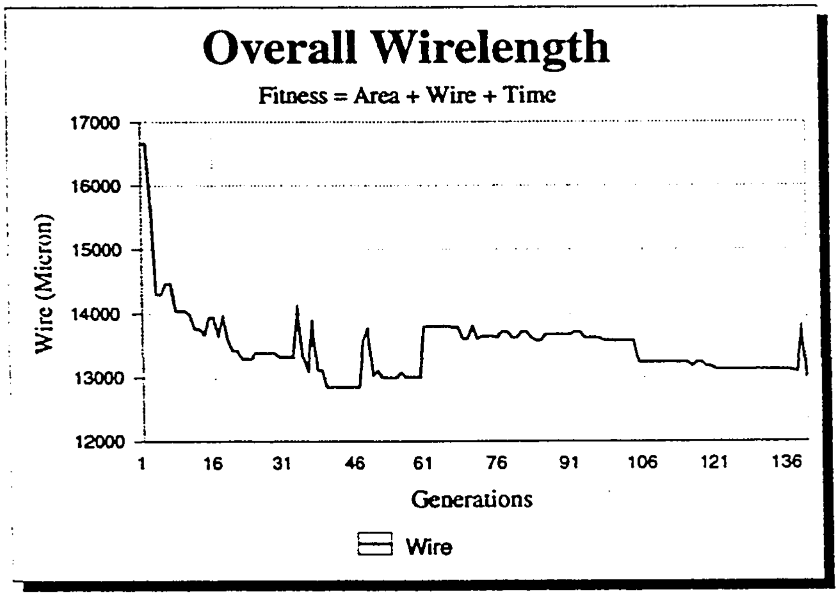
(a)

(b)

Figure 4.11: Plot showing area improvement with the number of generations.

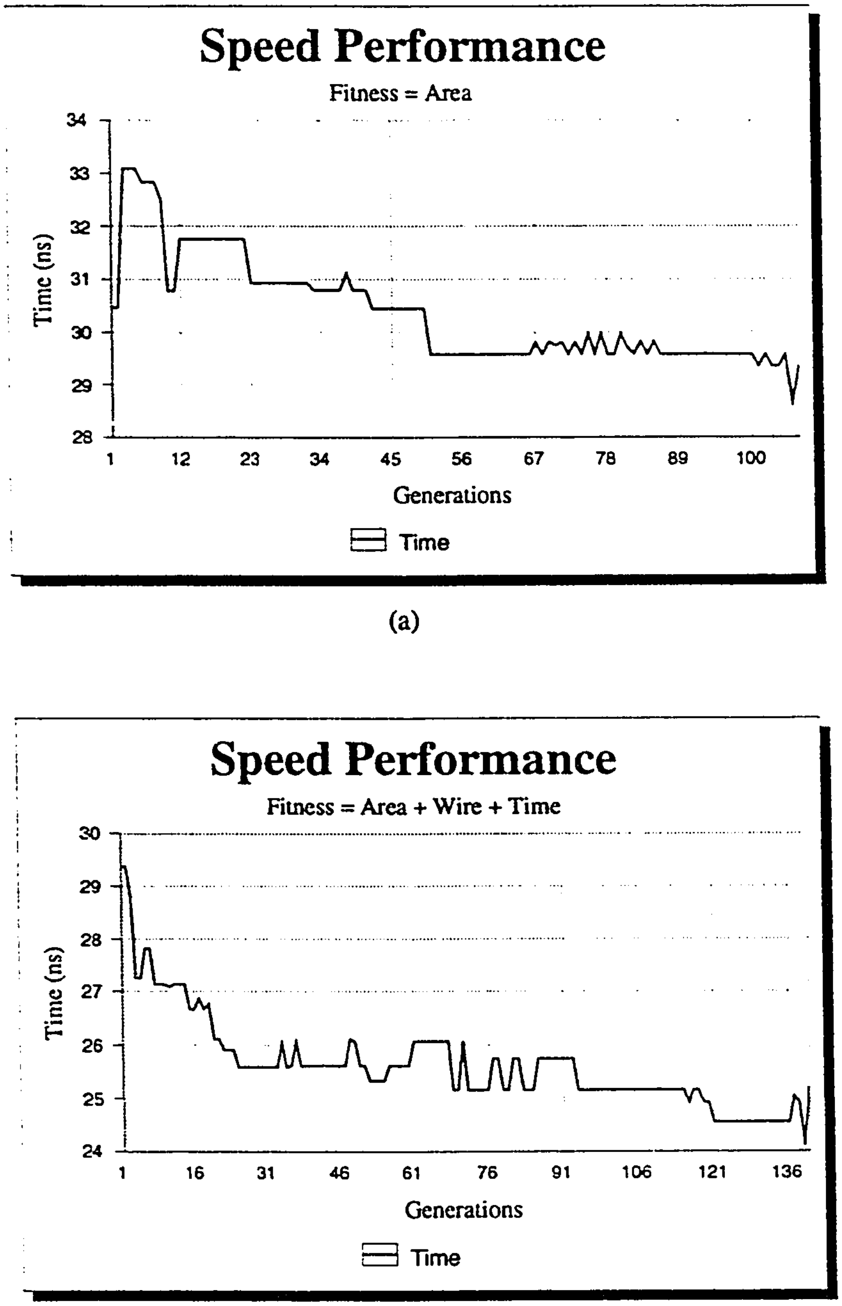


(a)



(b)

Figure 4.12: Plot showing wirelength improvement with the number of generations.



(a)

(b)

Figure 4.13: Plot showing speed performance improvement with the number of generations.

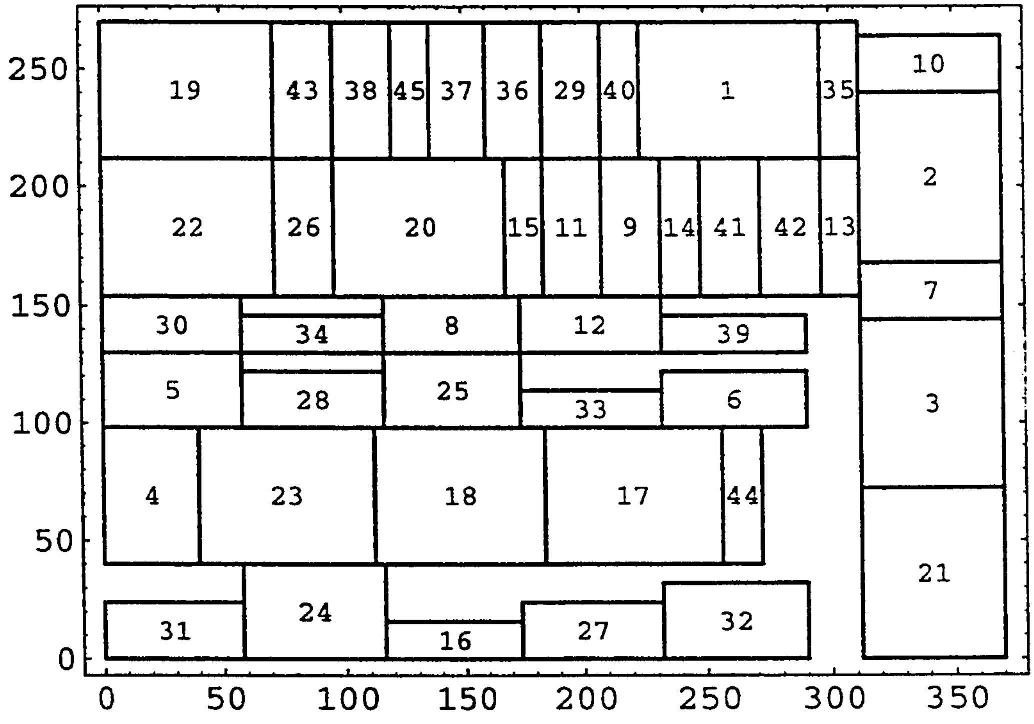


Figure 4.14: Floorplan of the best solution for the test circuit "Highway": fitness consists of only area objective (standard-cells).

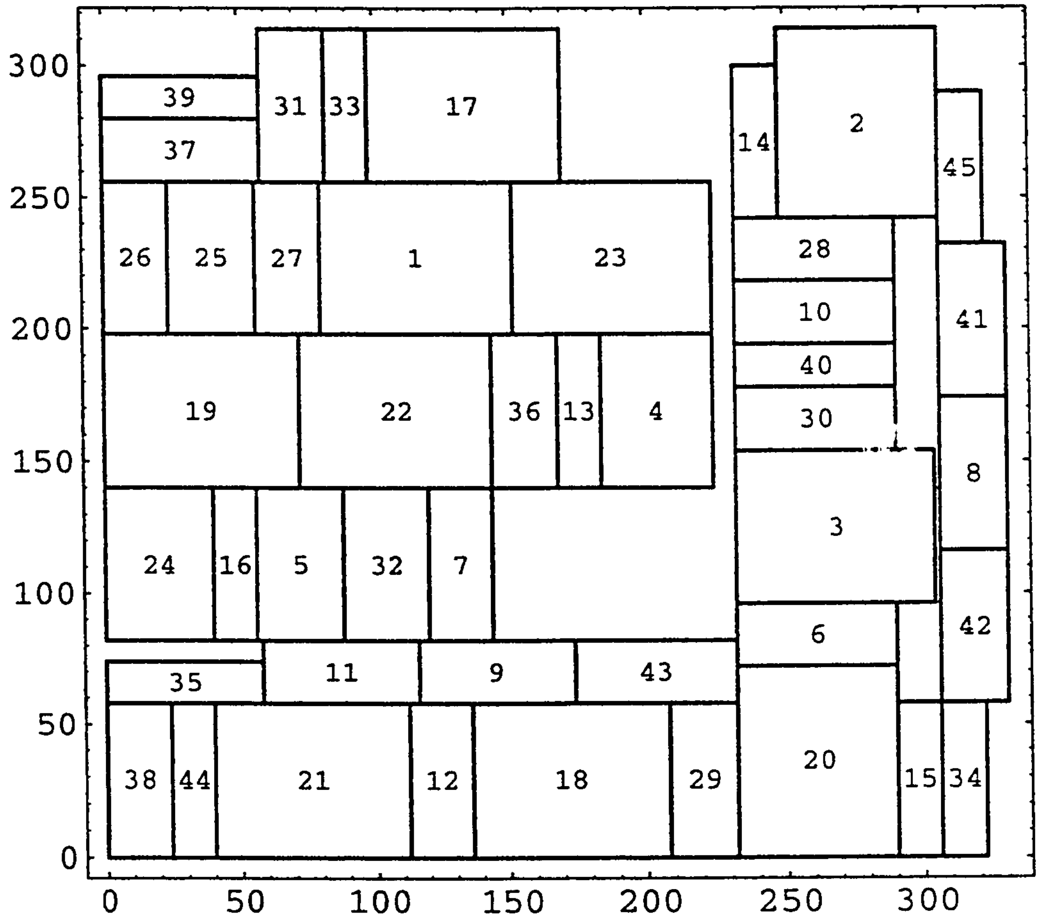


Figure 4.15: Floorplan of the best solution for the test circuit "Highway": fitness consists of all three objectives.

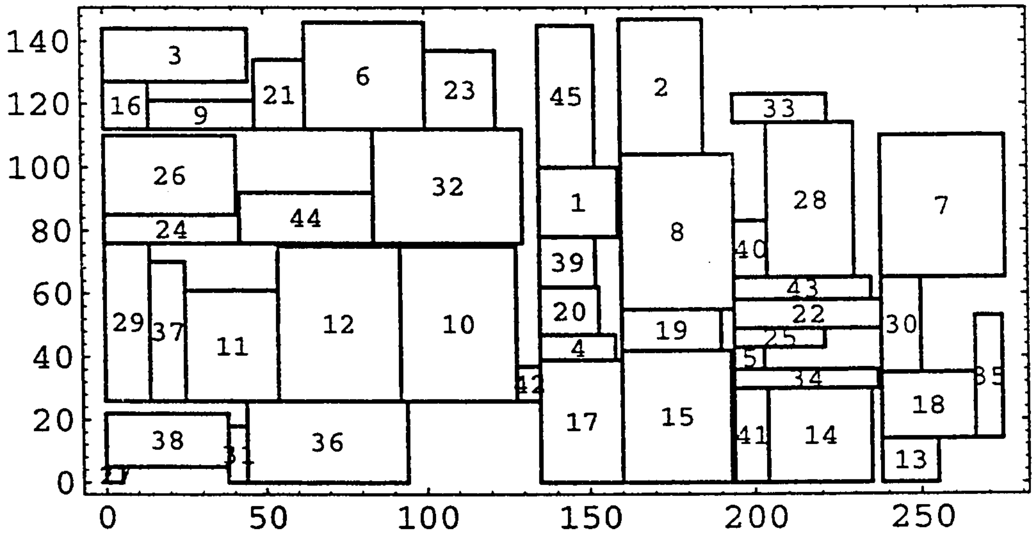


Figure 4.16: Floorplan of the best solution for the test circuit "Ckt4": fitness consists of only area objective (general-cells).

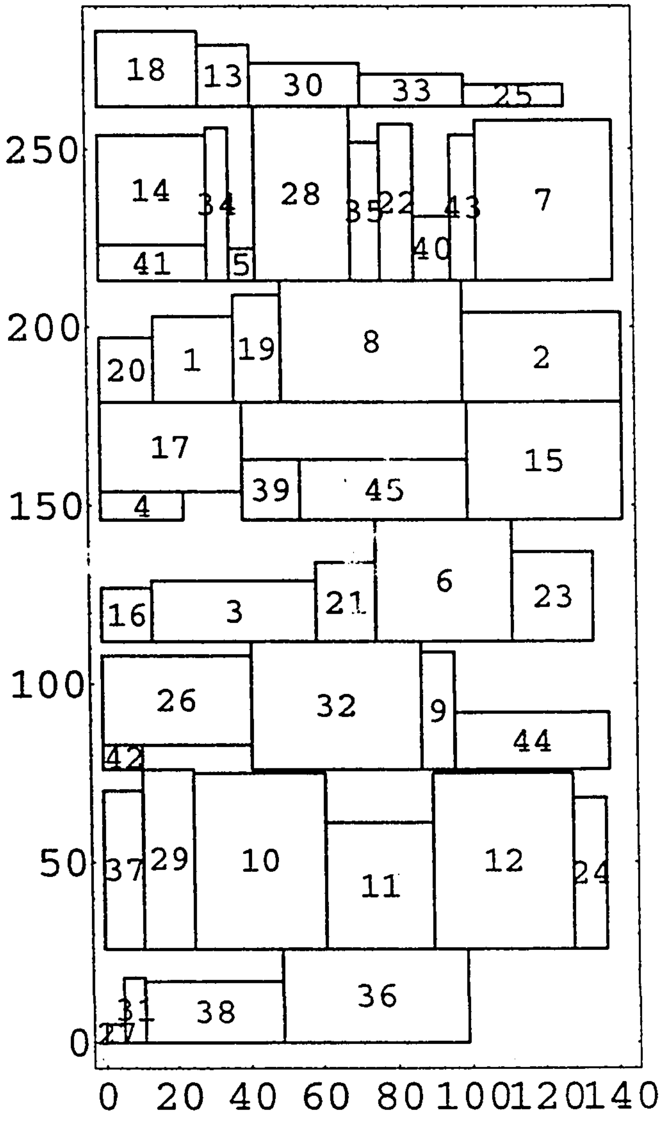


Figure 4.17: Floorplan of the best solution for the test circuit "Ckt4": fitness consists of all three objectives (general-cells).

Chapter 5

Conclusion and Future Work

5.1 Summary

In this thesis, we addressed the problem of timing influenced floorplanning for general-cell VLSI layouts. A general-cell floorplanner called Genetically Implemented Floorplanner for Timing (GIFT) was implemented. The floorplanning problem is solved in two main steps. During the first step, we adopt a simpler floorplan model having slicing structures with rigid blocks and free orientations. The cost function used is a weighted sum of overall area (functional + routing) and timing performance. The floorplan problem of the first step is solved using the genetic algorithm, an iterative technique. The second step is a floorplan refinement phase where blocks are allowed to have flexible shapes, and the floorplan solutions are no longer restricted to slicing structures. The main aim of the second phase, which employs a constructive refinement procedure, is to minimize the dead space. In this step, the relative positions of the modules are kept intact and the non-overlapping feature of floorplan solution produced by the first step is maintained.

In Chapter 1, an overview of the VLSI design process is presented. The description of floorplanning problem is given. The outline of the general-cell floorplanner is described. The motivation for incorporating timing during floorplanning stage is discussed.

In Chapter 2, a survey of the literature on floorplanning problem is given. The general floorplanning techniques are described briefly and the widely used methods are discussed. The related work on timing-driven design reported in the literature is reviewed.

In Chapter 3, an overview of genetic algorithm is given. The main steps in the design of a genetic algorithm are discussed. Various genetic operators namely crossover and mutation are described and illustrated with examples. The selection mechanism used for genetic operations and advancing individuals to the next generation are discussed. A summary of the important features of genetic algorithm is given.

In Chapter 4, the implementation details of the floorplanner are given. The various phases of the floorplanner are explained. The objective functions considered in this work are the *area*, the *speed performance* and the *overall wirelength*. All the objective functions are illustrated with examples. The wirelength estimation method used is an approximation of the Steiner tree method. A comparison of various estimation methods indicated that our method gives an estimate that is very close to the actual wirelength requirement. The timing issues are presented. The delay model and the notion of paths is explained. The performance metric used in the evaluation of floorplan quality is discussed. Extensive experiments were performed to determine the values of various genetic parameters. These include the size of the population, and the values of probabilities for the crossover and mutation operators. Having assigned the suitable values to these parameters, the experiments were carried out to arrive at an appropriate mechanism for advancing the solutions to the next generation. Since the cost function includes three terms, three runs of the floorplanner for the same test circuit were performed. In the first run, the best floorplan considering only *area* in the fitness was determined. In the second run, the best floorplan of the first run was used in the initial population of the second run and two terms namely *area* and *wirelength* were considered in the fitness. The best solution so obtained was used in the third run and all the three terms were included in the fitness. The best solution so obtained is reported as the optimal solution in the tabulated results.

5.2 Future Work

The following issues may be incorporated in this work in future:

- study the effect of dynamic population size on GIFT
- make GIFT driven by routing requirements
- use fuzzy logic for optimizing the process of decision making with regards to multiple objectives

Dynamic Population Size

Since genetic algorithm maintains a population of the feasible solutions, its CPU time requirement is quite high as compared to the other iterative improvement techniques such as simulated annealing and simulated evolution. Initial experiments with GIFT indicated that a large population size results in higher run time and vice-versa. Based on extensive experiments, the size of the population was chosen 30 and 10 for small and large circuits respectively. In all the experiments, the population size was kept fixed throughout the run of the algorithm giving rise to a high run time. If the population size is reduced beyond the values mentioned above, the quality of the final solution deteriorates considerably which may be attributed to insufficient diversity in the population. A clever method to improve the run time without affecting the quality of the final solution is to reduce the population size dynamically with generations. This idea was proposed and experimented in a genetic placer system [55]. With dynamic population size, the population is reduced based on what is termed as a "reduction schedule" which governs when and by how much the population is to be reduced. For example, if timing performance of the best solution does not improve over successive generations by $r\%$, then the population is reduced by $s\%$. The reduction process is continued as long as the population does not decrease beyond $t\%$ of the initial population. The values of parameters r, s, and t can be determined experimentally. The results of the genetic placer indicated an improvement in the run time by 50% with dynamic population size as compared to the fixed population size.

In GIFT, the idea of dynamic population size may be incorporated either by experimenting with aforesaid population size or by doubling it i.e., a size of 60 for the small circuits and 20 for the large circuits. This will have a high diversity in the population in the early generations and depending upon the reduction schedule, the population may be decreased dynamically over successive generations.

Routability

Routability is a very important objective to be met by a design. The issue of routability can be investigated as an extension to this work. The more complex the circuit is, the more dominant becomes the wiring in the final layout. Although, part of the wiring can be realized on top of the active devices (for instance in the sea-of-gates design methodology), a considerable portion of the chip is used exclusively for wiring [56]. This part of the chip may be realized using *wiring cells.* If these wiring cells are realized in rectangular regions, the wiring space can be thought of as the union of non-overlapping rectangular regions. The selection of wiring rectangles affects the efficiency of wiring procedures, determines the type and number of algorithms and the sequence in which wiring can be done.

It has been observed that slicing structures have considerable advantages over the general rectangle dissections for the reasons discussed next. Firstly, they imply a decomposition of the wiring space into the minimum number of rectangles (the slicing lines can be considered as the wiring rectangles). To distinguish these from the ones corresponding to the cells in the functional hierarchy, they are called *junction cells.* Secondly, a feasible sequence can be obtained from the slicing tree for generating the wiring. Thirdly, all the rectangles can be wired by a *channel router.* Finally, any Steiner tree heuristic can be used for routing the nets.

Multi-Objective Optimization by Fuzzy Logic

In this work, we used a mathematical formula as a scoring function to derive the fitness of a floorplan solution that is optimal with respect to overall area of the floorplan, total wirelength, and timing performance. Each objective function was assigned a weight. But there is no obvious method of assigning the weight values. This prompts the need to have a more realistic approach for optimizing the process of decision making with regards to multiple objectives. Unlike the single objective optimization problem, no concept of optimal solution is known to exist for the multi-objective optimization. In reality, the ranking of an individual objective reflects the preference of decision makers. A suitable approach would be to have a compromise between the competing objectives. Fuzzy approach has been found to be useful exactly for such a kind of problem. Fuzzy logic has been successfully applied to placement [57]. It would be worth investigating to apply fuzzy logic for floorplanning optimization.

Bibliography

[1] C. Mead and L. Conway. *Introduction to VLSI Systems.* Addison Wesley, 1980.

[2] S. M. Sait and H. Youssef. *VLSI Design Automation: Theory and Practice.* McGraw-Hill Book Co., Europe, 1994.

[3] S. Sutanthavibul, E. Shragowitz, and J. B. Rosen. An analytical approach to floorplan design and optimization. *Proc. of the 27th DAC*, 3, 1990.

[4] D. F. Wong and C. L. Liu. A new algorithm for floorplan design. *Proc. of the 23rd DAC*, pages 101–107, 1986.

[5] J. H. Holland. *Adaptation in Natural and Artificial Systems.* University of Michigan Press, Ann Arbor, Mich., 1975.

[6] D. E. Goldberg. *Genetic Algorithms in Search, Optimization and Machine Learning.* Addison-Wesley Publishing Company, INC., 1989.

[7] K. Al-Farra. *Timing Driven Floorplanning (MS Thesis).* Computer Engineering Department, KFUPM, Dhahran, 1994.

[8] S. Suthanthavibul, E. Shragowitz, and Rung-Bin Lin. An adaptive timing-driven placement for high performance VLSI's. *IEEE Transactions on Computer Aided Design*, 12(10):1488–1498, October 1993.

[9] W. Donath et al. Timing-driven placement using complete path delays. *Proceedings of 27th Design Automation Conference*, pages 84–89, June 1990.

[10] Y. Ogawa, M. Pedram, and E. S. Kuh. Timing Driven Placement for General Cell Layout Constraints. In *IEEE ISCAS*, pages 872–875. June 1990.

[11] S. Sutanthavibul and E. Shragowitz. Dynamic prediction of critical path and nets for constructive timing-driven placement. *28th Design Automation Conference*, pages 632–635. 1991.

[12] M. R. Garey and D. S. Johnson. Computers and Intractability: A Guide to the theory of NP-Completeness. 1979.

[13] M. A. Jabri. *An Artificial Intelligence Approach to Integrated Circuit Floorplanning.* Springer-Verlag Publishing Company, Inc.. 1991.

[14] U. Lauther. A min-cut placement algorithm for general cell assemblies based on a graph representation. *Proceedings of 16th Design Automation Conference*. pages 1–10, 1979.

[15] M. A. Breuer. A class of min-cut placement algorithms. *Proceedings of 14th Design Automation Conference*, pages 284–290, October 1977.

[16] M. A. Breuer. Min-cut placement. *Journal of Design Automation and Fault Tolerant Computing*, 1(4):343–382, October 1977.

[17] S. Kirkpatrick, Jr. C. Gelatt, and M. Vecchi. Optimization by simulated annealing. *Science*, 220(4598):498–516, May 1983.

[18] R. Otten and L. Van Ginneken. Floorplan design using simulated annealing. *Proc. of ICCAD*, 3, 1984.

[19] C. Sechen. Chip Planning, Placement, and Global Routing of Macro/Custom Cell Integrated Circuits Using Simulated Annealing. In *Proceedings of the 25th Design Automation Conference*, pages 73–80, June 1988.

[20] D. F. Wong and Khe-Sing The. An algorithm for hierarchical floorplan design. *Proc. of the ICCAD*, pages 484–487, 1989.

[21] R. A. Rutenbar. Simulated Annealing Algorithms: An Overview. *IEEE Circuits and Devices Magazine*, 3, 1989.

[22] J. P. Cohoon and W. D. Paris. Genetic placement. *IEEE Transactions on Computer-Aided Design*, 6(6):956–964, November 1987.

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

[24] Lin-Ming Jin and Shu-Park Chan. Analogue placement by formulation of macro-components and genetic partitioning. *International Journal of Electronics*, 73(1):157–173, 1992.

[25] J. P. Cohoon, S. U. Hedge, W. N. Martin, and D. Richards. Distributed genetic algorithms for the floorplan design problem. *IEEE Transactions on Computer-Aided Design*, 10(4):483–492, 1991.

[26] H. C. Chen, D. H. C. Du, and L. R. Liu. Critical path selection for performance optimization. *IEEE Transactions on Computer Aided Design*, CAD-12:185–195, February 1993.

[27] J. Darringer et al. LSS: A system for production logic synthesis. *IBM Journal of Research and Development*, 28(5):259–274, 1984.

[28] B. Kick. Timing correction in logic synthesis. *Proceedings of ICCAD'87*, pages 299–302, 1987.

[29] G. D. Micheli. Performance-oriented synthesis in the yorktown silicon compiler. *Proc. of ICCAD'86*, pages 138–141, 1986.

[30] K. J. Singh et al. Timing optimization of combinational logic. *Proc. of ICCAD'88*, pages 282–285, 1988.

[31] R. Brayton, G. D. Hachtell. and A. L. Sangiovanni-Vincentelli. A survey of optimization techniques of integrated circuit design. *Proceedings of IEEE*, 69(10):1334–1362, October 1981.

[32] N. P. Jouppi. Timing analysis and performance improvement of MOS VLSI design. *IEEE Transactions on Computer Aided Design*. CAD-6(4):650–665. July 1987.

[33] A. E. Ruelhi, P. K. Wolff. and G. Goertzel. Analytical power/timing optimization technique for digital system. *Proceedings of 14th Design Automation Conference*, pages 142–146. 1977.

[34] A. E. Dunlop et al. Chip layout optimization using critical path weighting. *Proceedings of 21st Design Automation Conference*, pages 133–136, 1984.

[35] M. Burstein and M. N. Youssef. Timing influenced layout design. *Proceedings of 22nd Design Automation Conference*, pages 124–130, 1985.

[36] P. S. Hauge, R. Nair, and E. J. Yoffa. Circuit placement for predictable performance. *Proceedings of International Conference on Computer-Aided Design*, pages 88–91, 1987.

[37] R. Nair et al. Generation of performance constraints for layout. *IEEE Transactions on Computer Aided Design*, CAD-8(8):860–874, August 1989.

[38] H. Youssef and E. Shragowitz. Timing constraints on signal propagation in VLSI. *Proceedings of International Conference on Computer-Aided Design*, pages 24–27, 1990.

[39] H. Youssef et al. Critical path issue in VLSI design. *Proceedings of International Conference on Computer-Aided Design*, pages 520–523, 1989.

[40] M. A. B. Jackson and E. S. Kuh. Performance-driven placement of cell-based IC's. *Proceedings of 26th Design Automation Conference*, pages 370–375, June 1989.

[41] J. F. Shapiro. *Mathematical Programming: Structures and Algorithms*. New York, Wiley, 1979.

[42] I. Lin and D. Du. Performance driven constructive placement. *Proceedings of 27th Design Automation Conference*, pages 103–105, 1990.

[43] A. Srinivasan, K. Chaudhary, and E. S. Kuh. RITUAL: a performance-driven placement algorithm. *IEEE Transactions on Circuits and Systems*, pages 825–840, November 1992.

[44] Rung-Bin Lin and E. Shragowitz. Fuzzy logic approach to placement problem. *Proceedings of 29th Design Automation Conference*, pages 153–158, 1992.

[45] M. Vecchi and S. Kirkpatrick. Global wiring by simulated annealing. *IEEE Transactions on Computer Aided Design*, CAD-2:215–222, 1983.

[46] K. Shahookar and P. Mazumder. A genetic approach to standard cell placement using meta-genetic parameter optimization. *IEEE Transactions on Computer Aided Design*, 9(5):500–511, May 1990.

[47] R. M. Kling and P. Banerjee. Empirical and theoretical studies of the simulated evolution method applied to standard cell placement. *IEEE Transactions on Computer Aided Design*, CAD-10:1303 – 1315, October 1991.

[48] J. P. Cohoon, S. U. Hedge, W. N. Martin, and D. Richards. Punctuated equilibria: A parallel genetic algorithm. In *Proceedings of the Second International Conference on Genetic Algorithms*, pages 148–154, 1987.

[49] D. F. Wong, H. W. Leong, and C. L. Liu. *Simulated Annealing for VLSI Design*. Boston, MA, Kluwer Academic, 1988.

[50] M. Srinivas and L. M. Patnaik. Genetic Algorithms: A Survey. *IEEE Computer Magazine*, pages 17–26, June 1994.

[51] H. Youssef. *Timing Issues in cell based VLSI Design*. Computer Science Department, University of Minnesota, 1990.

[52] Open Architecture Silicon Implementation Software, MCNC, 1990.

[53] Foresight Manual. *Orbit Semiconductor Inc., Sunnyvale, California*, July 1992.

[54] G. Vijayan and R. Tsay. A new method for floor planning using topological constraint reduction. *IEEE Transactions on CAD*. 10(12):1494–1501, December 1991.

[55] K. M. W. Nassar. *Timing Driven Placement Algorithm for Standard Cell Design (MS Thesis)*. Computer Engineering Department. KFUPM, Dhahran. 1994.

[56] D. D. Gajski. *Silicon Compilation*. Addison-Wesley Publishing Co. Inc. New Jersey. 1988.

[57] R. B. Lin and E. Shragowitz. Fuzzy Logic Approach to IC Placement Problem. *29th ACM/IEEE Design Automation Conference*, pages 153–158, 1992.

Vitae

- Mohammed Shahid Tanvir Khan
- Born in 1969 at Gondia (Maharashtra), India
- Received Bachelor of Engineering (**B.E.**) degree in Computer Science from Visvesvaraya Regional College of Engineering (**VRCE**), Nagpur University, Nagpur, India in 1991
- Joined the Department of Computer Engineering at King Fahd University of Petroleum and Minerals (**KFUPM**), Dhahran, Saudi Arabia as a Research/Teaching Assistant in May 1992
- Received Master of Science (**M.S.**) degree in Computer Engineering from KFUPM, Saudi Arabia in 1994