Towards Optimal Circuit Layout Using Advanced Search Techniques

by

Shawki Areibi

A thesis

presented to the University of Guelph

in fulfilment of the

thesis requirement for the degree of

Doctor of Philosophy

in

Electrical Engineering

Guelph, Ontario, Canada, 2006

Abstract

A VLSI chip can today contain millions of transistors and is expected to contain more than 100 million transistors in the next decade. This tremendous growth is made possible by the development of sophisticated design tools and software. To deal with the complexity of millions of components and to achieve a turn around time in terms of a couple of months, VLSI design tools must not only be computationally fast but also generate layouts close to optimal. The work in this thesis involves exploring algorithmic solutions to the problem of circuit layout in VLSI design. The exploration is an attempt to evaluate, design, improve and integrate the best combinatorial algorithms to solve the circuit layout problem. Advanced search heuristic techniques in the form of Tabu Search, GRASP and Genetic Algorithms are used extensively to solve most of the problems in circuit layout.

We show in this thesis that new hybrid partitioning techniques based on the above mentioned heuristics outperform traditional heuristic methods. In fact, these novel approaches consistently find better solutions than other methods in a fraction of the time. A new placement algorithm that is suitable for standard cell layout is also presented. The initial placement is obtained using the partitioning algorithm. An efficient clustering based algorithm is developed to further reduce the complexity of circuit partitioning and placement and improve the performance of the design process in terms of quality and computation time. Finally, parallel implementations of the developed heuristics on a network of workstations are presented and significant speedups are reported. The ability of the hybrid heuristics to find near optimal solutions is assessed by comparing their performance with a general purpose mixed integer programming package. Experimental results indicate that our heuristics based on clustering and hybridization schemes give very good results and are suitable for VLSI circuits.

Acknowledgements

My sincere thanks go to my supervisor Professor Vannelli for his support and advice throughout this research. Without his help, this work would never have been possible.

I would like to thank Professor Michel Minoux from the University of Paris for being my external examiner. I am deeply indebted to Professor M. Kamel, C. Gebotys, G. Kesidis from the University of Waterloo, for their advice and helpful discussions.

I would also like to thank Phil Regier the system administrator of the VLSI group, for his patience and help.

Many of my friends helped in one way or another to accomplish this research, thanks to all especially Scott Hadley at Shell Research and Otman Basir and Yousef Berbash at the System Design Department for their helpful discussions and advice.

To

my wife, mother, brother and children

whose love and encouragement helped accomplish this

thesis.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

The last decade has brought explosive growth in the technology for manufacturing integrated circuits. Integrated circuits with several hundred thousand transistors are now commonplace. This manufacturing capability, combined with the economic benefits of large electronic systems, is forcing a revolution in the design of these systems and providing a challenge to those people interested in integrated system design. Since modern circuits are too complex for an individual designer or a group of designers to comprehend completely, managing this tremendous complexity and automating the design process have become crucial issues.

1.1 The VLSI Design Process

Electronic design is carried out in many ways by designers for a variety of purposes. It is impossible to describe one methodology that applies in all cases. Instead of attempting to provide a comprehensive description, this section highlights the design process that places physical design in perspective and defines the interfaces of physical design with other design phases. The VLSI design process is divided for simplicity into five parts [HU85] as seen in Figure 1.1a: System Specification, Functional Design, Logic Design, Circuit Design, and Circuit Layout. Each design phase is further divided into three steps consisting of synthesis, analysis, and verification.

1.1.1 Design Representations

During the design process several different representations, or views, are used to show different aspects of the system under development. Figure 1.1a shows an example of the representation used during each phase of VLSI design. These views are classified as *behavioral, structural and physical*, and they represent various levels of abstraction [Prea88]. Behavioral representations describe a circuit's function. Procedural descriptions and Boolean expressions are behavioral representations; they say nothing about implementation. Structural representations describe the composition of circuits in terms of cells (abstractions of circuit element definitions), components (abstractions of instances of circuit elements) and interconnections among the components. Physical representations are characterized by information used in the manufacture or fabrication of physical systems, such as geometric layout or topological constraints. These representations only implicitly describe how a circuit behaves.

1.1.2 Steps Within A Design Phase

Each design phase is characterized by *synthesis, analysis* and *verification* steps as shown in Figure 1.1b. Synthesis derives a new or improved representation based on the representation derived in the previous phase. Analysis follows synthesis in

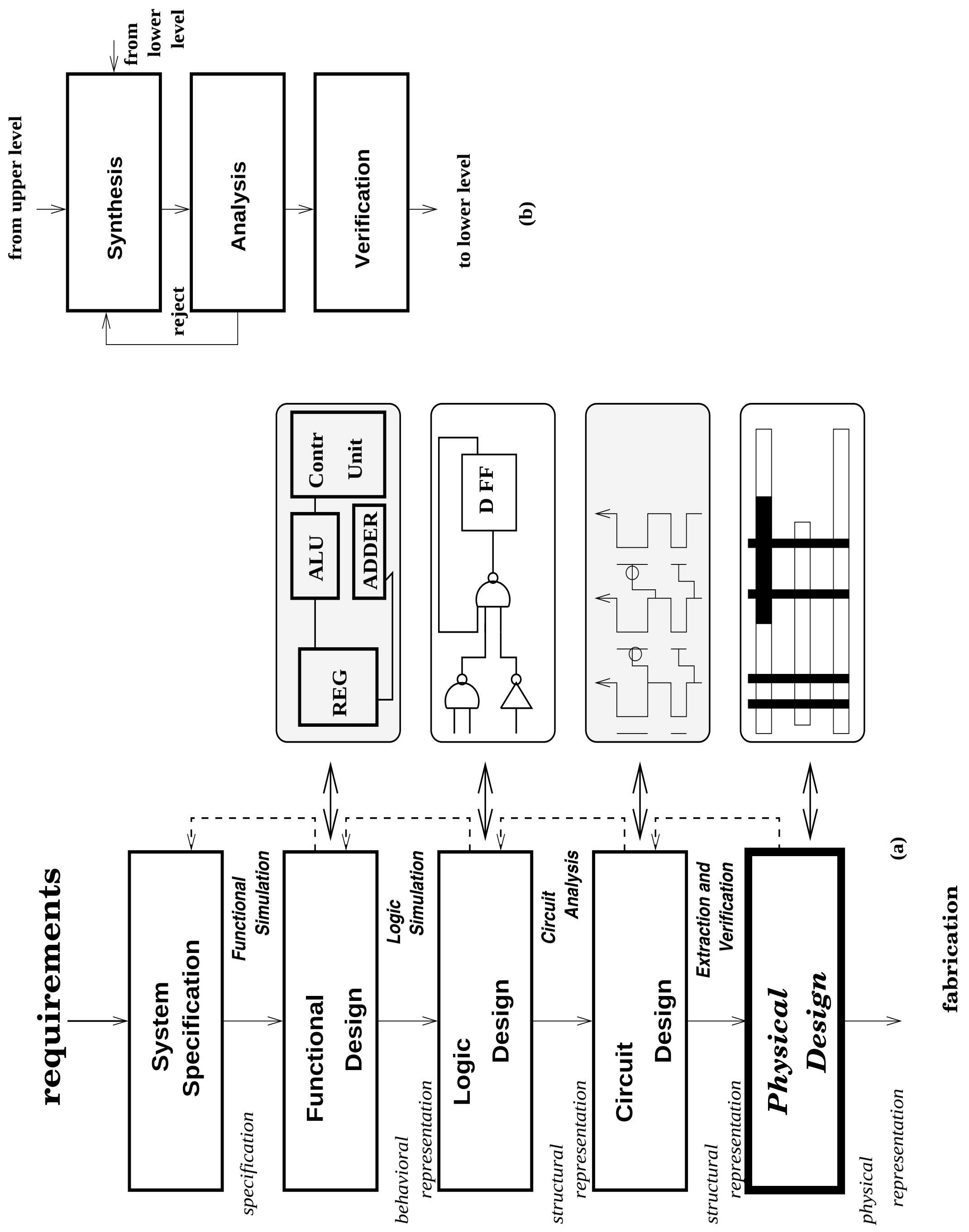


Figure 1.1: The VLSI design process

each design phase and generally takes two forms. First, a design representation must be evaluated against its requirements. For VLSI circuits the requirements are usually specified in terms of die size, performance, and power consumption. A design must also be evaluated for behavioral, structural, and physical correctness and completeness. Verification, the final step within a design phase, demonstrates that the synthesized representation is equivalent under all conditions of interest to representations in other phases.

1.1.3 Electronic Design Phases

At the system specifications level, the goals and constraints of the system are defined; that is, what the system will do, the criteria of optimization, the speed requirement, the space or area requirements, the power requirements, and so forth.

In the functional design phase, the functional relationships among the subunits are decided. The results may be a purely behavioral representation, or it may include structural aspects by partitioning functionality into components. The behavioral simulation is the normal method of analysis.

Logic design, concerns the logic structure that implements the functional design. Here, a set of boolean expressions or a representation of a finite state machine need to be realized. The logic networks have to be converted into electronic circuits. The logic design is validated by comparing the results from the logic level and behavioral level simulations.

The circuit design phase concerns the electrical laws that govern the detailed behavior of the basic circuit elements such as transistors, resistors, capacitors, and inductors. In this phase, transistors are sized to meet signal delay requirements.

Analysis is performed using circuit and timing simulations.

In the physical design or circuit layout phase, the behavioral or structural representations from the previous phases are transformed into geometric shapes that are used in the fabrication of the system. Physical design is a complex process, therefore, it is usually broken down into various sub-steps in order to handle the complexity of the problem. In fact, physical design is arguably the most time consuming step in the VLSI design cycle.

In the final stages of the design cycle (not shown in the figure), a verification phase checks if all design rules are satisfied. Then the chip is tested and debugged for any occurring errors. Finally, a prototype is built and tested, before mass production.

1.1.4 Complexity of Circuit Layout

The input to the physical design phase is a circuit diagram and the output is the layout of the circuit. This is accomplished in several stages using partitioning, floor-planning, placement, routing, and compaction. The main objective of this design process is to position devices on the chip so as to minimize the total area of the layout. In addition, the total interconnection wire is reduced.

The layout design process is a complicated task [Leng90]. There are two aspects of building a layout system that contribute to its complexity. One is the *combinatorial aspect* and the other is the *system aspect*. The combinatorial aspect is that most of the optimization problems that have to be solved during integrated-circuit layout are intractable or NP-hard [Ullm84, Leig83]. The system aspect deals with other problems such as the maintenance of the consistency in the design database in the presence of simultaneous changes of the same design object and other related problems. This thesis is concerned **only** with the combinatorial aspect of circuit layout. The purpose of this work is to introduce efficient heuristic solutions to the underlying combinatorial problems in circuit layout. The developed heuristics for circuit layout are promising and provide an excellent basis to design efficient CAD tools.

1.2 Motivation

A large subset of problems in VLSI CAD is computationally intensive, and future CAD tools will require even more accuracy and computational capabilities from these tools. Table 1.1 shows examples of CAD tools and their run-times on a SUN/4 Sparc workstation [Bane94], rated to operate at about 20 MIPS.

The circuit sizes in each of the problems are quite moderate, about 1000 to 40,000 gates. It is clear that for circuits that are hundred times larger, 100,000 to 1 million gates, the run-time requirements will increase 100 times or more. CAD tools that take hours to run on current designs may take weeks or months to run on future designs. In fact, some of the industrial CAD tools running on the leading

Application	**CAD Tool**	**Circuit Size**	**Runtime**
Extraction	HPEX3.0	1,000,000 rectangles	1 hour
Placement	TimberWolf	2907 cells	1 hour
Logic Synthesis	MIS2.2	7657 gates	2 hours
Test Generation	HITEC2.0	17,793 gates	5 hours
Circuit Simulation	RELAX2.0	40,000 elements	1 month

Table 1.1: Example runtime of VLSI CAD tools on a SUN/4 workstation

edge ASIC gate-array chips containing 500,000 gates take weeks to run tasks such as placement, routing, and layout verification [Bane94]. The VLSI design cycle involves iterations, both within a step and between different steps. The entire design cycle may be viewed as transformations of representation in various steps. In each step, a new representation of the system is created and analyzed. The representation is iteratively improved to meet system specifications. For example, the layout is iteratively improved so that it meets the timing specifications of the system. Another example may be the detection of design rule violations during design verification. If such violations are detected, the physical design step need to be repeated to correct the error. Therefore, one of the objectives of VLSI CAD tools is to minimize the number of iterations and thus reduce the time-to-market. This can be accomplished by having efficient heuristics, that are expected to produce near optimal solutions in reasonable amounts of time.

Another motivation for designing highly efficient heuristics for CAD tools is cost. While the costs of integrated circuits have dropped exponentially, the basic procedure of silicon manufacture is unchanged. A *wafer* is still tested and chopped into *dies* that are packaged. The cost of a packaged integrated circuit is [?]:

$$cost\ of\ IC \;=\; \frac{cost\ of\ die\ +\ cost\ of\ testing\ die\ +\ cost\ of\ packaging}{Final\ test\ yield}$$

$$cost\ of\ die \;=\; \frac{cost\ of\ wafer}{Dies\ per\ wafer \times Die\ yield}$$

The number of dies per wafer is basically the area of the wafer divided by the area of the die. Reducing the die area by half – the parameter that circuit layout tools controls – doubles the die yield, and thus decreases the cost of production.

The final and most important reason to have efficient heuristic solutions to the circuit layout problem is speed. Effective placement and routing routines minimize the die area and the total interconnection wire length. This in place leads to high performance devices. In other words, the sheer size of the VLSI circuit, the complexity of the overall design process, the desired performance of the circuit, and the cost of designing a chip dictate that efficient heuristics are required to solve the circuit layout problem effectively. Despite significant research efforts in this field, the CAD tools still lag behind the technological advances in fabrication. This calls for development of efficient heuristics for physical design automation.

1.3 Dissertation Framework

The main goals of this dissertation are: *efficiency, robustness*, *complexity reduction*, and *speed* as seen in Figure 1.2. Each goal is achieved using a different strategy.

1.3.1 Efficiency

The primary goal of this thesis is to develop **efficient heuristic techniques** to solve the circuit layout problem. This is achieved by evaluating and improving the performance of recent advanced search methods such as *Tabu Search* [**?**, Glov90, Blan91], *GRASP* [Feo94], *Simulated Annealing* [Sech88a], and *Genetic Algorithms* [**?**] on the circuit partitioning problem, and compare them with traditional combinatorial optimization techniques in terms of the execution time and quality of solution. The experience gained from applying these advanced heuristics on the circuit partitioning problem can then be utilized extensively on the more difficult

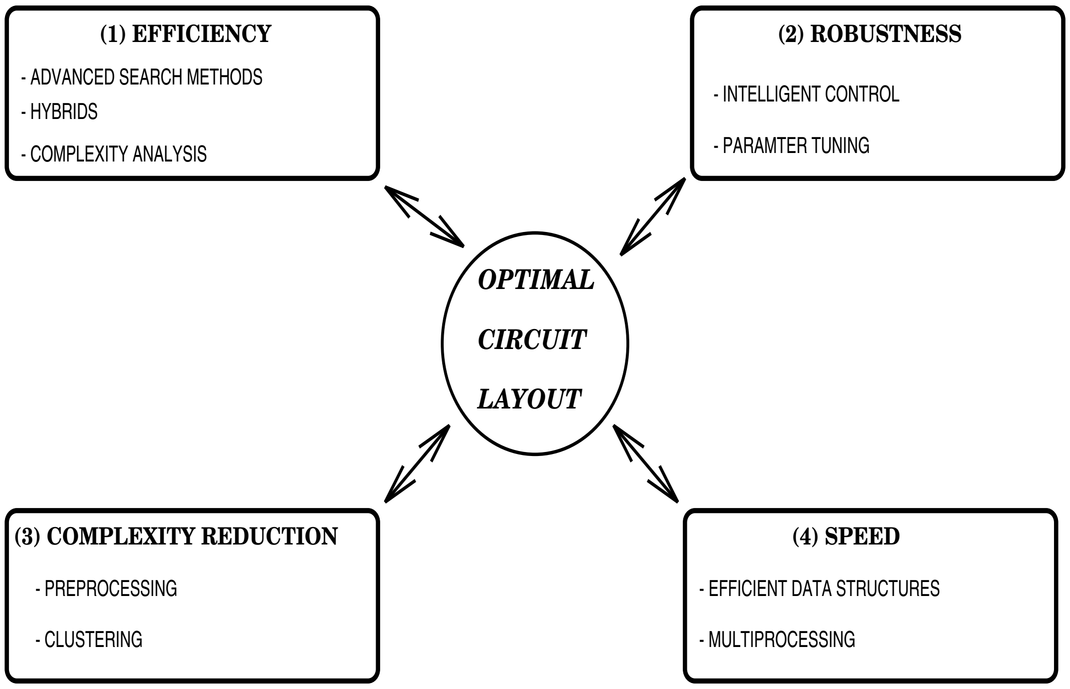


Figure 1.2: Dissertation goals and strategy

placement problem. The second strategy to achieve efficiency, is to emphasize the advantages and disadvantages of these different search techniques, and show the importance of *combining* these distinct models of computation to solve the combinatorial optimization problems in circuit layout. We show that by integrating these fundamentally different approaches, one can avoid many of the weaknesses inherent in each methodology, while capitalizing on their individual strengths. In general, we are interested in finding the most "efficient" algorithm for solving the circuit layout problem. In its broadest sense, the notion of efficiency involves all the various computing resources needed for executing an algorithm. The order of growth of the running time of an algorithm, gives a simple characterization of the algorithm's efficiency and also allows us to compare the relative performance of alternative algorithms. The *worst case analysis* of the above mentioned heuristics is an important measure of the goodness of these heuristics. We evaluate several of the important heuristics that are developed by determining the time and space required to solve large instances of the combinatorial optimization problems of circuit layout.

1.3.2 Robustness

One of the major research goals in this dissertation is to improve the robustness of layout heuristics developed. The heuristics should be adaptable to a wide range of circuit sizes, cost functions, and technology boundary conditions. This important goal can be achieved by *fine-tuning* of parameter settings that affect the performance of the above mentioned heuristics. Selecting and adapting parameter settings that control strategies such as Tabu Search or Simulated Annealing have a drastic effect on the final generated solution.

Another means of accomplishing this goal is through intelligent control of the heuristics involved in solving the problem. The main task of an intelligent controller involves adapting the parameters of the heuristic according to the properties of the solution space being searched.

1.3.3 Complexity Reduction

Reducing the complexity of the design is a further important goal in this dissertation. Complexity reduction is achieved by using *preprocessing* and *circuit clustering*. Preprocessing based on statistical information of the circuit reduces the complexity by eliminating long nets and fixing modules in a particular position through out the search process. Circuit clustering on the other hand plays a fundamental role in reducing the complexity of the circuits, by utilizing a hierarchical approach to shorten the design period. This is in place improves the performance of the design process.

1.3.4 Speed

Finally, our goal is to increase the processing power available to the execution of the developed heuristics. This is accomplished by using *efficient data structures* and *distributed processing*. *Parallelism* is applied using standard equipment to simplify experiment management and reduce the overall CPU time required to provide efficient solutions to the problems. A *Network Multiprocessing Environment* has been developed to parallelize the efficient CAD heuristics. Unlike previous parallel heuristics that required special parallel machines with shared memory or dedicated interconnection networks, the CAD heuristics run on a network of workstations in a robust and efficient manner.

1.4 Research Contributions

The main contributions described in this dissertation are:

- Adapting several advanced search heuristics for solving the circuit partitioning problem. We show the main advantages and disadvantages of each technique, and the means to overcome their weaknesses.

- Intelligent fine tuning of different parameters for Tabu Search, GRASP, and Genetic Algorithms allows these heuristics to be used effectively for circuit layout in particular and other combinatorial optimization problems in general.

- A new hybrid technique based on Tabu Search and Simulated Annealing has been designed. The hybrid technique gives the same high quality of solutions of Simulated Annealing and reduces the computation time by 50% on average.

- A powerful circuit partitioning hybrid based on Tabu Search and Genetic Algorithm is proposed. The uniqueness of this hybrid stems from the ability of a search controller to coordinate the search process efficiently.

- Effective clustering techniques for circuit partitioning in particular and circuit layout in general are presented. These techniques are based on grouping "closely" connected cells and nets within the same partition. The main advantage of these clustering methods is in their capability of providing good initial solutions and reducing the computation time by 60% on average for local search heuristics.

- A novel methodology is used to improve the performance of the MIP CPLEX mixed integer programming package to solve the circuit partitioning problem. Statistical information extracted from the circuit assigns priority to most integer variables, this reduces the symmetry of the MIP and accordingly reduces the number of equivalent solutions.

- Developing and implementing parallel versions of the circuit partitioning heuristics. The parallel heuristics utilize a number of workstations (connected together by a local area network) to distribute the load among the processors, and speed up the execution time of the heuristics.

- A new constructive placement heuristic that is suitable for standard-cell and gate-array designs has been proposed. The algorithm is based on a partitioning algorithm that provides an initial placement for the iterative improvement algorithm.

- A robust and efficient iterative technique for circuit placement has been designed. This technique is mainly based on Tabu Search to improve upon the initial placement provided by the constructive heuristic.
- A worst case complexity analysis of the Tabu Search partitioning heuristic is described and compared to previous iterative improvement techniques.
- A worst case complexity analysis of clustering approaches used for circuit partitioning and placement is presented.

The research in this thesis is hoped to be of great value not only for circuit layout problems but for other combinatorial and engineering optimization problems as well. The developed heuristic techniques can be adjusted to these applications with minimum effort and in some cases with no modification except for the objective function. Some of the applications that can utilize the developed efficient heuristics are:

1. *Distributed Simulation*: Partitioning and mapping for parallel distributed simulation as an attempt to minimize the communication overhead and uniformly distribute the execution load among the processors.
2. *Logic Synthesis Applications*: An obvious way of speeding up the logic synthesis application is to generate a large number of logic partitions of a given circuit and to synthesize each partition independently. The results of the individual partitions can then be merged back together.

3. *Task Scheduling*: The problem here is to schedule a collection of tasks on p processors so that the maximum completion time is minimized. The main idea is to construct a schedule and iteratively modify it by assigning some tasks to more processors.

1.5 Thesis Outline

This thesis is divided into nine chapters. Chapter 2 presents an overview of the VLSI design process and a brief review of heuristic-based search techniques presently used in circuit layout. Advanced search techniques in the form of Tabu Search, Simulated Annealing, GRASP, and Genetic Algorithms are introduced in Chapter 3. Results obtained for the circuit partitioning problem using these advanced techniques are also presented. Chapter 4 introduces new methodologies of combining the above heuristics to tackle the circuit partitioning problem in particular and placement in general. These hybrid systems attempt to combine the power of decentralized characteristics of heuristics that explore the solution space with the more localized features of heuristics that efficiently locate local optimum.

Chapter 5 presents a novel technique to significantly reduce the complexity of the circuits by means of clustering. Techniques for parallel implementation are introduced in Chapter 6 to increase the processing power available to the execution of the developed programs. The placement problem using new constructive and iterative improvement methods is presented in Chapter 7. Analysis of convergence of some of the above hybrid heuristics are presented in Chapter 8.

Finally, the thesis is concluded in Chapter 9 with a summary of present and future work. Figure 1.3 presents the main outline of this dissertation.

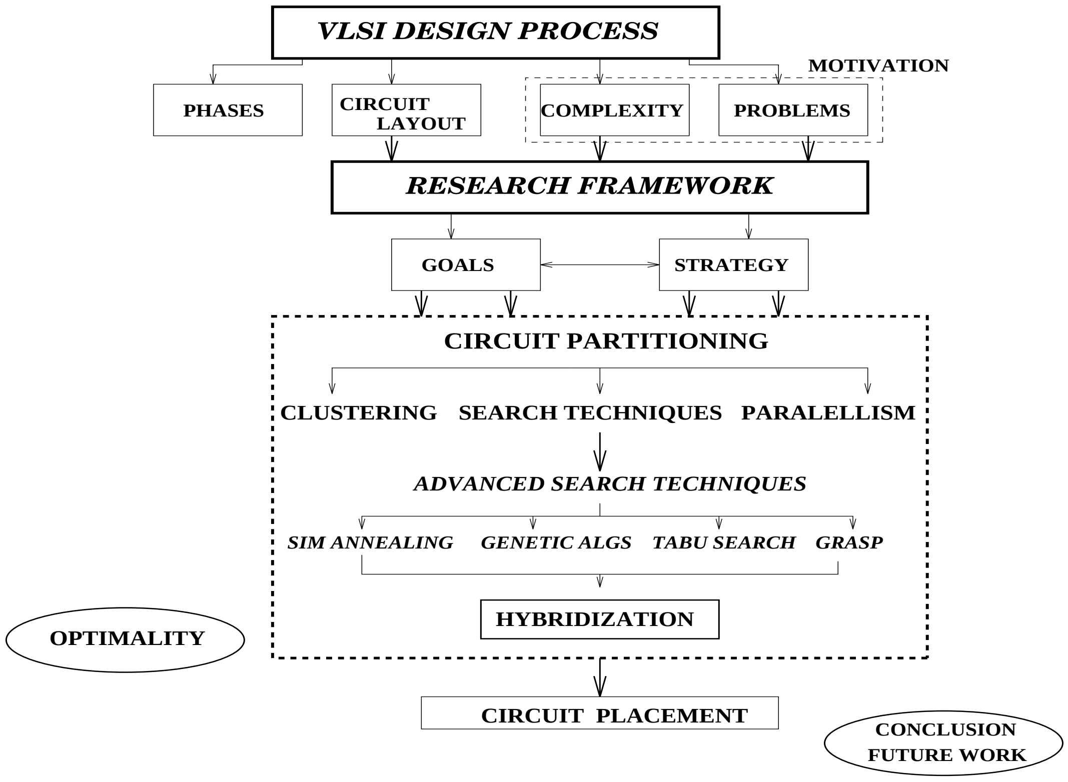


Figure 1.3: Dissertation outline

Chapter 2

Background

2.1 Physical Design and Circuit Layout

In the combinatorial sense, the layout problem is a constrained optimization problem. We are given a circuit (usually a module-wire connection-list called a *netlist*) which is a description of switching elements and their connecting wires. We seek an assignment of geometric coordinates of the circuit components (in the plane or in one of a few planar layers) that satisfies the requirements of the fabrication technology (sufficient spacing between wires, restricted number of wiring layers, and so on) and that minimizes certain cost criteria. Practically, all aspects of the layout problem as a whole are intractable; that is, they are NP-hard [Hach89]. Consequently, we have to resort to heuristic methods to solve very large problems. One of these methods is to break up the problem into subproblems, which are then solved one after the other. Almost always, these subproblems are NP-hard as well, but they are more amenable to heuristic solutions than is the entire layout problem itself. Each one of the layout subproblems is decomposed in an analogous fashion. In this way, we proceed to break up the optimization problems until we reach primitive subproblems.

These subproblems are not decomposed further, but rather solved directly, either optimally (if an efficient polynomial-time optimization algorithm exists) or approximately if the subproblem is itself NP-hard or intractable, otherwise. The most common way of breaking up the layout problem into subproblems is first to do *logic partitioning* where a large circuit is divided into a collection of smaller modules according to some criteria, then to perform component *placement*, and then to determine the approximate course of the wires in a *global routing* phase. This phase may be followed by a *topological-compaction* phase that reduces the area requirement of the layout, after which a *detailed-routing* phase determines the exact course of the wires without changing the layout area. After detailed-routing, a geometric-compaction phase may further reduce the layout area requirement [HU85].

2.1.1 Circuit Partitioning

Circuit partitioning is the task of dividing a circuit into smaller parts. It is an important aspect of layout for several reasons. Partitioning can be used directly to divide a circuit into portions that are implemented on separate physical components, such as printed circuit boards or chips. Here, the objective is to partition the circuit into parts such that the sizes of the components are within prescribed ranges and the complexity of connections between the components is minimized. As can be seen in Figure 2.1, after swapping modules between the two blocks we end up minimizing the number of signal nets that interconnect the components between the blocks. A natural way of formalizing the notion of wiring complexity

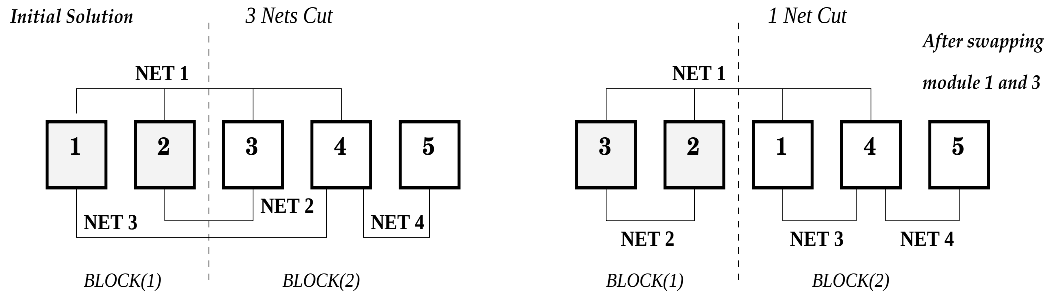


Figure 2.1: Illustration of circuit partitioning

is to attribute to each net in the circuit some connection cost, and to sum the connection costs of all nets connecting different components. A more important use of circuit partitioning, is to divide up a circuit hierarchically into parts with divide-and-conquer algorithms for *placement*, floorplanning, and other layout problems. Here, cost measures to be minimized during partitioning may vary, but mainly they are similar to the connection cost measures for general partitioning problems.

2.1.1.1 Circuit Partitioning as a 0-1 Quadratic Transportation Problem

The problem of optimally partitioning an undirected graph G with n_m nodes into n_b blocks can be formulated as 0-1 quadratic transportation problem. The disjoint blocks have exactly $m_1, m_2, \ldots, m_{n_b}$ nodes per block ($m_1 \approx m_2 \approx m_{n_b} \approx \frac{n_m}{n_b}$) such that

$$\sum_{k=1}^{n_b} m_k = n_m$$

A variable is assigned to each node, in the form x_{ik} where:

$$x_{ik} = \begin{cases} 1 & \text{if node } i \text{ is in block } k \\ 0 & \text{otherwise} \end{cases}$$

If nodes i and j are both in block k, then $x_{ik}x_{jk} = 1$. Let **A** represent the connection matrix (i.e., **A** is the adjacency matrix of G). The weight of the edge joining node i to node j is a_{ij}. If nodes i and j are not connected, then $a_{ij} = 0$. The problem is to partition the graph into n_b blocks, such that, the sum of the weights on the interconnection between the n_b blocks is minimized. In other words, the goal is to maximize the edges inside each block. The objective function is thus

$$Max \sum_{k=1}^{n_b} \sum_{i=1}^{n_m} \sum_{j=1}^{n_m} a_{ij} x_{ik} x_{jk} \tag{2.1}$$

s.t. (i) Module placement constraints:

$$\sum_{k=1}^{n_b} x_{ik} = 1, \quad \forall_i = 1, 2, \ldots, n_m$$

(ii) Block size constraints

$$\sum_{i=1}^{n_m} x_{ik} \le \frac{n_m}{n_b}, \quad \forall_k = 1, 2, \ldots, n_b$$

$$x_{ik} \in \{0, 1\}, \quad 1 \le i \le n_m; \quad 1 \le k \le n_b$$

Barnes [Barn82] developed a polynomial time heuristic for approximating the above 0-1 quadratic transportation problem. The heuristic is based on approximating the netlist or hypergraph by a weighted graph G [Hadl92, Vann90], that tightly estimates the number of cut nets in any netlist partition. The numerical optimization technique used by Barnes transforms the graph partitioning problem into a linear transportation problem, which is solved in polynomial time. This technique can consider 2^k linear transportation problems, where k is a small number of blocks, and does not require multiple runs. Barnes's algorithm first finds the n_b largest *eigenvalues* of the connection matrix $\mathbf{A}$ of the graph and their corresponding orthonormal eigenvectors $u_1, u_2, \ldots, u_{n_b}$. Let u_{ik} be the i^{th} component of the eigenvector corresponding to the kth largest eigenvalue of the adjacency matrix of G. Barnes [Barn82] shows that the solution of the following *linear transportation problem* gives an approximate solution to the graph partitioning problem:

$$Max \sum_{i=1}^{n_m} \sum_{k=1}^{n_b} \frac{u_{ik}}{\sqrt{m_k}} x_{ik} \tag{2.2}$$

s.t. (i) Position constraints:

$$\sum_{k=1}^{n_b} x_{ik} = 1, \quad \forall_i = 1, 2, \ldots, n_m$$

(ii) Block size constraints:

$$\textstyle\sum_{i=1}^{n_m} x_{ik} = m_k, \quad \forall_k = 1, 2, \ldots, n_b$$
$$x_{ik} \geq 0, \quad 1 \leq i \leq n_m; \quad 1 \leq k \leq n_b$$
$$m_1 \geq m_2 \geq \ldots \geq m_{n_b}$$

For the special case of partitioning into two blocks, the above computation can be further simplified. The first two eigenvalues of the connection matrix $\mathbf{A}$ and their corresponding eigenvectors u_1 and u_2 are found; a combined eigenvector

$$u_{1-2} = \frac{u_{i1}}{\sqrt{m_1}} - \frac{u_{i2}}{\sqrt{m_2}} \quad , i = 1, 2, \ldots, n_m$$

is calculated. The largest m_1 terms of the sorted u_{1-2} vector represent the nodes of one block, and the next m_2 terms represent the nodes of the other block [Hadl92].

2.1.1.2 Lower Bounds

The eigenvector technique of Barnes [Barn82] was used to partition the graph G into k blocks of fixed module size. Another feature of this graph underestimation model of the netlist is that it allows one to obtain *lower bounds* on the actual number of cut nets. An underestimation can prove to be useful when either a lower bound or the optimal solution for the graph partitioning problem can be found. The reason the underestimation is useful is that any lower bound for the graph partitioning problem will also provide a lower bound for the netlist partitioning problem. In contrast, if a general estimation is used, bounding results from graph partitioning cannot be exploited.

Recall that the weight of any cut in the generated graph G underestimates the number of generalized edges cut in H. [Hadl92] and [Dona73] introduced an approach that finds lower bounds on the weight of any cut of G. So, we can find a lower bound on the number of cut generalized edges of H.

Consider the matrix $\mathbf{A}$, where a_{ij} is the weight of the edge joining nodes i and j (i.e. $\mathbf{A}$ is the adjacency matrix of G). The matrix $\mathbf{A}$ is symmetric with zeroes along the main diagonal. Consider any diagonal matrix $\mathbf{U}$, where

$$\sum_{i=1}^{n} u_{ii} = -\sum_{i}\sum_{j} a_{ij}$$

then it can be shown that [Dona73]:

$$E_c \geq -\frac{1}{2}\sum_{i=1}^{k} m_i \lambda_i(\mathbf{A}+\mathbf{U}), \tag{2.3}$$

where E_c is the sum of the edges cut by the *optimal* partition and λ_i is the i^{th}

largest eigenvalue of the matrix **A**+**U**.

2.1.1.3 0-1 Linear Programming Formulation of Netlist Partitioning

A standard mathematical model in VLSI layout associates a graph $G = (V, E)$ with the circuit netlist, where vertices in V represent modules, and edges in E represent signal nets. The netlist is more generally represented by a *hypergraph* $H = (V, E')$, where hyperedges in E' are the subsets of V contained by each net (since nets often are connected to more than two modules). In this formulation, we attempt to partition a circuit with n_m modules and n_n nets into n_b blocks containing approximately $\frac{n_m}{n_b}$ modules each; (i.e. we attempt to equi-partition the V modules among the n_b blocks), such that the number of uncut nets in the n_b blocks is maximized.

Defining:

$$x_{ik} = \begin{cases} 1 & \text{if module } i \text{ is placed in block } k \\ 0 & \text{otherwise} \end{cases}$$

$$y_{jk} = \begin{cases} 1 & \text{if net } j \text{ is placed in block } k \\ 0 & \text{otherwise} \end{cases}$$

So the linear integer programming (LIP) model of the netlist partitioning problem is given by maximizing the number of uncut nets in each block;

$$Max \quad \sum_{j=1}^{n_n} \sum_{k=1}^{n_b} y_{jk} \tag{2.4}$$

s.t. (i) Module placement constraints:

$$\sum_{k=1}^{n_b} x_{ik} = 1, \quad \forall_i = 1, 2, \ldots, n_m$$

(ii) Block size constraints:

$$\sum_{i=1}^{n_m} x_{ik} \leq \frac{n_m}{n_b}, \quad \forall_k = 1, 2, \ldots, n_b$$

(iii) Netlist constraints:

$$y_{jk} \leq x_{ik}, \ where \quad \begin{array}{l} 1 \leq j \leq n_n \\ 1 \leq k \leq n_b \\ i \in Net\ j \end{array}$$

(iv) 0-1 constraints:

$$\begin{array}{lll} x_{ik} \in \{0,1\}, & 1 \leq i \leq n_m; & 1 \leq k \leq n_b \\ y_{jk} \in \{0,1\}, & 1 \leq j \leq n_n; & 1 \leq k \leq n_b \end{array}$$

The net placement constraints determine if a net (wire) j is placed entirely in block k or if it is not. In problem (LIP) we maximize the number of uncut nets in the n_b blocks. This is equivalent to the netlist partitioning problem where we minimize the number of wires connecting the n_b blocks.

2.1.1.4 Complexity of Circuit Partitioning

At the basis of all partitioning problems are variations of the following combinatorial problem.

Hypergraph Partitioning [Prea88]

Instance:	An undirected hypergraph G = (V,E)
	with vertex weights $w : V \rightarrow I\!N$,
	edge weights $l : E \rightarrow I\!N$,
	and a maximum cluster size $B \in I\!N$
Configurations:	All partitions of V into subsets $V_1, \ldots, V_m$ *where* $m \geq 2$.
Legal configurations:	All partitions such that
	$\sum_{v \in V_i} w(v) \leq B, \forall_i = 1, \ldots, m.$
Cost functions:	$c(V_1, \ldots, V_m) =$
	$\sum_{e \in E}(\|\{i \in \{1, \ldots, m\} \| V_i \cap e \neq \phi\}\| - 1)l(e)$

The legal configurations are the partitions in which each cluster V_i has a total vertex weight not exceeding B. The weights of the vertices represent the block sizes, and the weights on the edges represent connection costs. The maximum cluster size B is a parameter that controls the balance of the partitions.

The *Hypergraph Partitioning* problem is NP-complete even if $B \geq 3$ is fixed and $w \equiv 1, l \equiv 1$ [Hyaf73]. The problem is only weakly NP-complete if G is restricted to be a tree [Bert82]. In this case there is a pseudo-polynomial time algorithm that solves the problem in time $O(nB^2)$. If G is a tree and all edge weights are identical, or if G is a tree and all vertex weights are identical [**?**], then the problem is in **P**.

Currently none of the results cited above has practical impact on the VLSI layout procedures [Prea88]; the instance of the *hypergraph partitioning* problem that occur in circuit layout go beyond the graph classes considered in the restrictions of the problem.

2.1.2 Circuit Placement

After the circuit partitioning phase, the area occupied by each block can be calculated, and the number of terminals required by each block is known. In addition, the netlist specifying the connections between the blocks is available. In order to complete the layout, we need to arrange the blocks on the layout surface and interconnect their pins according to the netlist. In *general-cell* and *standard-cell* placement, we want to position the components of the circuit such that the layout area is minimized. The area measure used here comprises the area taken up by the circuit components as well as the area needed for wiring the circuit components together (wiring area or routing area). Since hierarchy is employed in practical circuit layout, the circuit components to be placed are not necessarily single devices such as transistors, but may represent large chunks of circuitry, such as complete adders or control sub-circuits. Thus, the placement problem has the dual flavor of a two-dimensional packing problem and a connection-cost optimization problem. The packing problem is concerned with fitting a number of cells of different sizes and shapes tightly into a rectangle. The connection-cost optimization aims at minimizing the amount of wiring necessary.

2.1.2.1 Complexity of Circuit Placement

The placement problem can be stated in terms of several non-equivalent combinatorial problems, depending on what cost measure should be minimized. Each one of these problems is NP-complete. Many placement techniques are based on hypergraph partitioning. This approach is aimed at minimizing wire congestion. The corresponding combinatorial problem is NP-complete (as described in Section

2.1.1.4). Methods that try to minimize a cost measure, representing interconnection cost, look for an assignment of coordinates in the plane, say for the centers of blocks, such that a cost measure estimating the total amount of wiring is minimized.

Two-Dimensional Placement [Prea88]

Instance:	A Hypergraph G = (V,E) and an interconnection cost function $c : E \rightarrow I\!N$, a set $V' \subset V$ of fixed vertices and positions $s_i, t_i \in I\!N \;\; \forall v_i \in V'$
Legal configurations:	All placements $((x_i, y_i)) \;\; i = 1, \ldots, \lvert V \rvert \;\; with \; x_i, y_i \in I\!N$ such that $x_i = s_i$ and $y_i = t_i \;\; \forall v_i \in V'$
Cost functions:	$\sum_{e \in E} c(e) \max_{v_i, v_j \in e} d(x_i, y_i, x_j, y_j)$

The *Two-Dimensional Placement* problem is in **P**. In fact, it can be stated in terms of two identically structured linear programs, one for each dimension. This characteristic remains true even if the *Two-Dimensional Placement* problem is generalized to incorporate block sizes, pin positions in blocks, etc. [Prea88]. In general the solution of the *Two-Dimensional Placement* problem only gives a first approximation of the placement; the blocks have finite dimensions and may still overlap in the placement. Insisting that blocks be non-overlapping amounts to requiring that the coordinates of the placement be integer.

Optimal Linear Arrangement [Prea88]

Instance:	A Hypergraph G = (V,E) and a cost function $c : E \rightarrow I\!N$,
Legal configurations:	All permutations $\pi : \{1, \ldots, n\} \rightarrow \{1, \ldots, n\}$.
Cost functions:	$\sum_{e \in E} c(e) \max_{v_i, v_j \in e} \lvert \pi(i) - \pi(j) \rvert$.

This problem is NP-hard, even if G is only a graph and $c \equiv 1$ [**?**]. It can be solved in polynomial time $O(n^{2.2})$ if G is a tree and $c \equiv 1$ [Shil79]. If the quadratic distance measure is used in the cost measure, then the problem is still NP-hard for graphs with $c \equiv 1$, but no polynomial time algorithm for trees is known.

2.1.3 Routing

Routing follows the placement phase. It determines the course of the wires that connect the cells laid out during the placement. The structure of the routing phase depends greatly on the design and fabrication technology [Kuh83]. There are two approaches to routing; namely, *two-phase* and *area routing*. In area routing, the routing process is carried out in one phase that locates the actual geometric layout of each net within the assigned routing regions.

In two-phase routing, the routing phase is subdivided into *global routing* and *detailed-routing*. In the global (or loose) routing phase [Mowc87], nets or net segments are assigned to specific routing channels in the IC, which determines how wires maneuver around and through cells. As a result of this phase, the interconnection pattern in each channel is defined and is independent of all other channels. In the second phase, called detailed (or local) routing, the interconnection pattern in each channel is implemented by assigning the net segments to tracks and columns in the channel. This phase is commonly referred to as channel routing.

2.1.3.1 Global Routing

The input to the global routing problem is a floorplan that can be represented as a planar graph F = (V,E), whose vertices represent the blocks and whose edges represent possible routing channels. Each edge e has two values associated with it, a capacity *c(e)* and a length *l(e)*. The capacity is a measure of how many wires fit into the corresponding routing channel. The length defines the length of the routing channel. In addition a multi-set N of nets is given—each net n being a subset of the vertices of F. N is a multi-set, because the same subset of vertices can occur several times in N. The nets have to be connected through the routing channels.

Global Routing [Prea88]

Instance:	A Planar graph F = (V,E), a multi-set N of nets on V
	a length function $l : E \rightarrow I\!N$,
	a capacity function $c : E \rightarrow I\!N$.
	let k_n be the multiplicity of n in N.
Legal configurations:	A set $N' \subset N$ or routable nets, and a set of subtrees
	$T_{n,i}$ of F, $n \in N', i \in 1, \ldots, k_n$. Here $T_{n,i}$
	must connect all terminals of net n. $T_{n,i}$ is called
	a Steiner tree for net n. For $e \in E$, let U(e) denote
	the set of pairs (n,i) such that e is an edge in $T_{n,i}$.
	Then for all $e \in E, \|U(e)\| \leq c(e)$ has to hold.
Cost functions:	$\sum_{e \in E} l(e)\|U(e)\| + \lambda\|N - N'\|$.
	Here λ is some large constant.

The constraints $|U(e)| \leq c(e)$ ensure that no edge is used above its capacity. The large constant λ in the cost function ensures that the number of routable nets is maximized with priority.

2.1.3.2 Complexity of Routing

The *Global Routing* problem is NP-hard even if F is a grid and all nets are two-terminal nets and $c \equiv 1$ [Kram84]. If N has only one net n, and $k_n = 1$, this problem becomes the Minimum Steiner Tree problem. The *Minimum Steiner Tree* problem is also NP-hard, but various approximation algorithms exist [Widm86]. If the Minimum Steiner Tree problem is restricted to the case that n is a two terminal net, it becomes the shortest path problem, which is efficiently solvable with Dijkstra's algorithm. All maze routers and several Steiner tree heuristics are based on this observation. Thus, there is a large body of knowledge about the Minimum Steiner Tree problem. The difficulty with solving this problem for global routing is that each instance of the problem considers only one net. Some order has to be chosen in which the nets are processed in order to achieve global routing. The order in which nets are routed has to be chosen heuristically, and finding good heuristics is a major stumbling block for finding good routings. Recently, investigations have been carried out to determine how all nets can be routed simultaneously. Most approaches to simultaneous routing of all wires reformulate the global routing problem as an integer linear program [Vann89].

There are several reasons why the routing task is subdivided into the two phases of global routing and detailed-routing. One is that each phase is more tractable than the whole routing problem in one phase. In fact, many versions of the detailed-routing problem are in **P** [Leng90] (i.e. polynomial in time). Another reason is that, depending on the design and fabrication technology, there are several different detailed-routing models. The detailed-routing model influences the size of the routing space needed, but it is assumed to have only a secondary influence on the global routing. Thus, the global routing is somewhat more technology-independent form of layout description than the detailed routing.

2.1.4 Layout Strategies and Styles

Physical design is an extremely complex process and even after breaking the entire process into several conceptually easier steps, it has been shown that each step is computationally hard. However, market requirements demand a quick time-to-market and high yield. As a result, restricted models and design styles are used in order to reduce the complexity of physical design. The classification and comparison of layout styles is given in [Ueda86]. Currently, the popular VLSI physical design styles are *gate-arrays*, *standard-cells*, *general-cells*, and *full-custom design*.

2.1.4.1 Gate-Array Design

In gate-array design, the entire wafer is prefabricated with an array of identical gates or cells. As shown in Figure 2.2a, the cells are separated by both vertical and horizontal spaces called vertical and horizontal channels. The name 'gate-array' signifies the fact that each cell may simply be a gate, such as a 2 input OR gate. The number of tracks allowed for routing in each channel is fixed. As a result, the purpose of routing phase is simply to complete the connections rather than to minimize the area. Because of the large amount of rigidity imposed both by the design technology and by the prefabrication of the master, gate-arrays do not achieve the same level of performance and amount of density as do full-custom chips.

2.1.4.2 Standard-Cell Design

In standard-cell layout as seen in Figure 2.2b, the cells are small and rectangular; often, all cells have the same height but different widths; and the cells have fixed connections on the left and right side (clocks and/or power) that abut with each other. The placement phase places the standard-cells in horizontal rows. The global routing phase determines where the wires switch between the rows of standard-cells. These locations are called *feedthroughs*. The detailed-routing phase amounts to a set of channel routing problems, one for each routing channel between two adjacent rows. This design style is well-suited for moderate size circuits and medium production volumes. Physical design using standard-cells is more difficult compared to gate-arrays, but much easier than full-custom design.

2.1.4.3 General-Cell Design

As seen in Figure 2.2c, the general-cell design style is a generalization of the standard-cell design style. The cells (available from a library or constructed as required by the design system) may be large and irregularly shaped. Automatic placement of general-cell designs is complicated because the cells must be represented as two dimensional objects and their sizes and shapes can vary widely. Automatic routing is also more difficult (compared to standard-cells and gate-arrays) since the channels may interact in complex ways.

2.1.4.4 Full-Custom Design

This method is characterized primarily by the absence of constraints on the design process. It usually requires a hand-crafted level of automation since the lack of

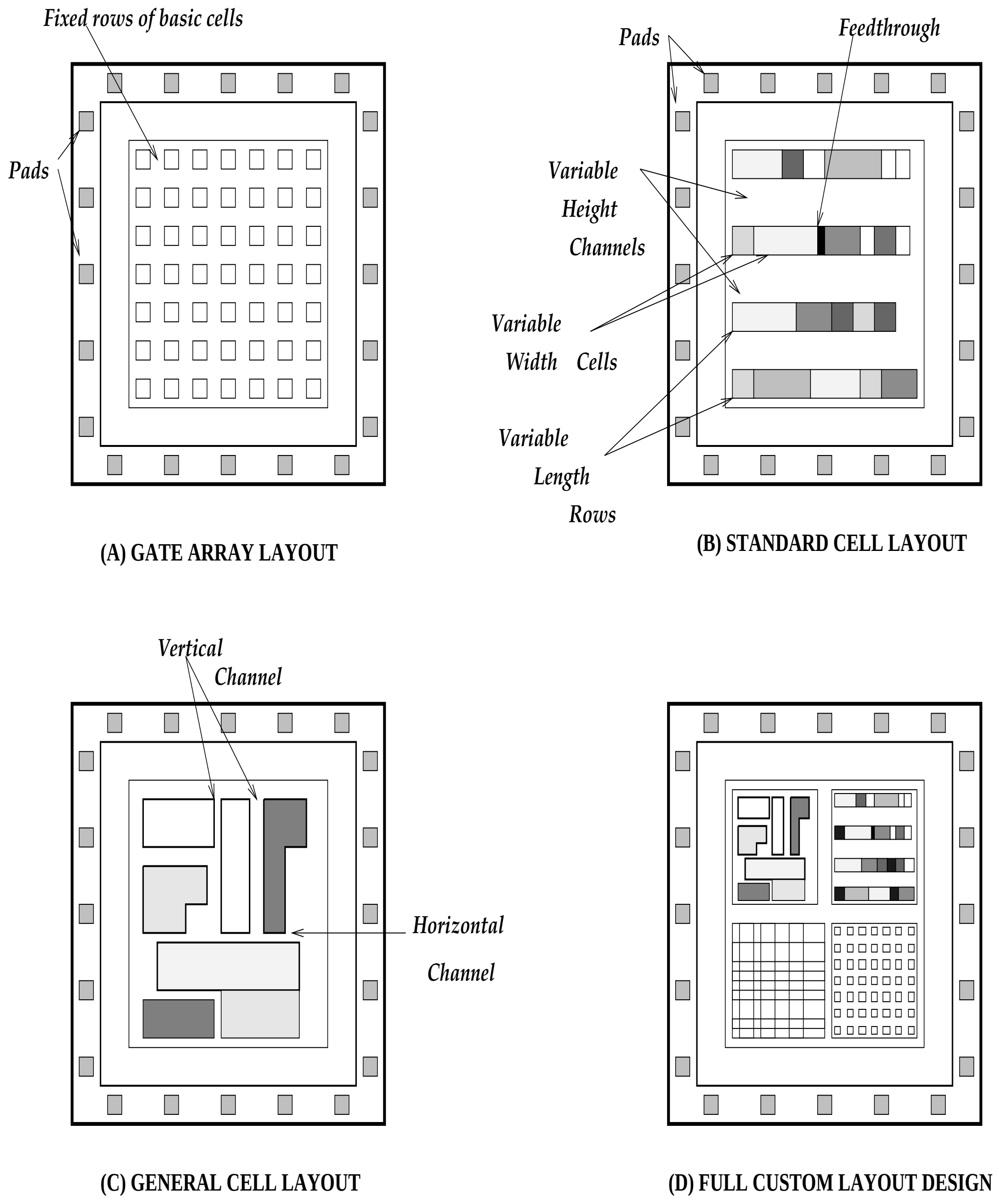


Figure 2.2: Layout styles

constraints makes synthesis tools difficult to develop. Full-custom design as seen in Figure 2.2d is time-consuming; thus the method is inappropriate for large circuits. However, the full-custom method is widely used for smaller cells that are inputs to synthesis tools.

2.2 Optimization Algorithms

Solving a combinatorial optimization problem amounts to finding the "best" or "optimal" solution among a finite or countably infinite number of possible solutions. Considerable effort has been devoted to constructing and investigating methods for solving to optimality or proximity combinatorial optimization problems. Integer, linear and non-linear programming, as well as dynamic programming have seen major breakthroughs in recent years. Over the years it has been shown that many theoretical and practical combinatorial optimization problems belong to the class of NP-complete problems. A detailed overview of problems in this class is given by Garey and Johnson [**?**]. A direct consequence of the property of NP-completeness is that optimal solutions cannot be obtained in reasonable amounts of computation time. However, large NP-complete problems still must be solved, and in constructing appropriate algorithms one might choose between two options. Either one tries to achieve optimality at the risk of very large, possibly impracticable, amounts of computation time, or one chooses quickly obtainable solutions at the risk of sub-optimality. The first option constitutes the class of optimization algorithms. Well known examples of enumeration methods use cutting plane, branch and bound and dynamic programming techniques. The second option constitutes the class of approximation algorithms, also called heuristic algorithms; examples are metric methods, iterative improvement, and calculus-based methods. The division between the two classes falls into a "grey" region.

2.2.1 Exact Solution Techniques

Integer programming, dynamic programming and graph search techniques are designed to produce global extrema for the problems to which they are applied [Gill81]. Unfortunately, many real-world problems including circuit layout are so large and difficult that these methods cannot achieve this extrema efficiently due to their large storage or computational time requirements.

2.2.2 Approximate Solution Techniques

Heuristic methods can produce good solutions (possibly even an optimal solution) quickly. Often in practical applications, several good solutions are of more value than one optimal one. The first and foremost consideration in developing heuristics for combinatorial problems of this type is finding a procedure that is powerful and yet sufficiently fast to be practical. For the circuit partitioning problem three different classes of algorithms were used to generate good partitions. The techniques are, *Iterative Improvement heuristics*, *Numerical Optimization Techniques*, and *Simulated Annealing.*

2.2.2.1 Iterative Improvement Techniques

To date, iterative improvement techniques that make local changes to an initial partition are still the most successful partitioning heuristics used in practice. The advantage of these heuristics is that they are quite robust. In fact, they can deal with netlist as well as arbitrary vertex weights, edge costs, and balance criteria. The heuristics are frequently used in divide-and-conquer algorithms for placement and floorplanning that are variants of the mincut strategy [Leng90].

Kernighan and Lin [?] described the fundamental heuristic procedure for graph partitioning which became the basis for most of the iterative improvement partitioning and placement heuristics generally used. Their heuristic shown in Figure 2.4 dealt with the problem of partitioning a graph with c cells, where c is even, into two blocks of $c/2$ cells each. The basic approach is to start with a given partition and to improve it by iteratively choosing one cell from each of the blocks and exchanging them as seen in Figure 2.3. fig:kir-lin

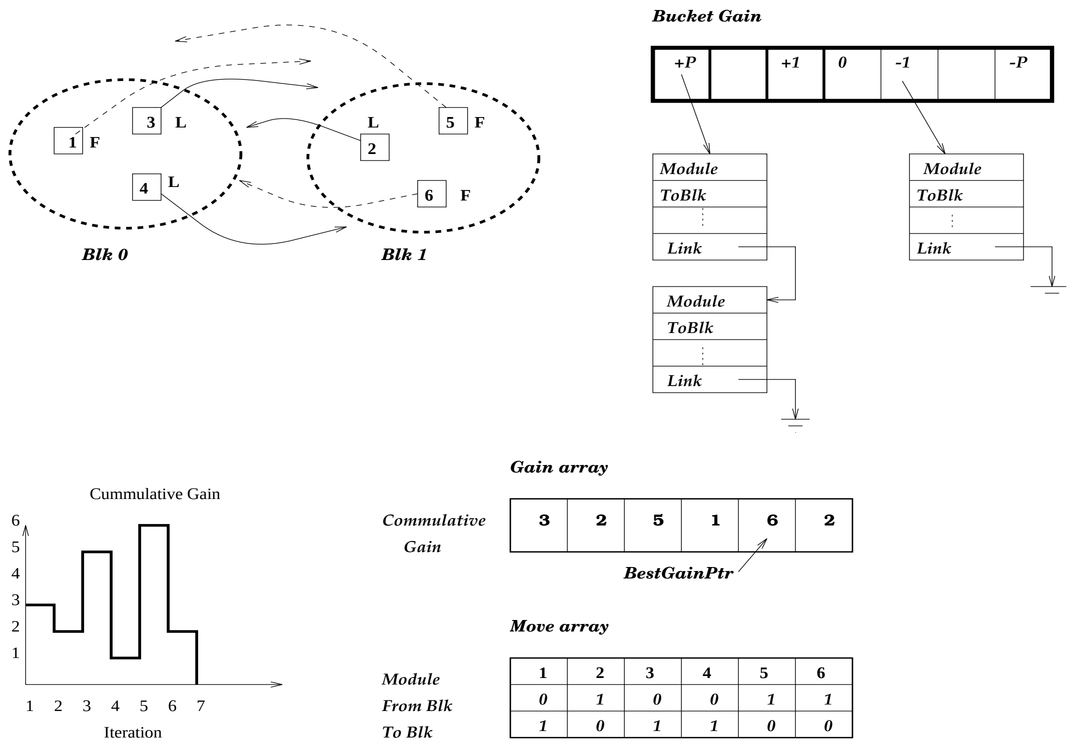


Figure 2.3: Iterative improvement example based on node interchange

The cells to be switched are chosen so that a maximum decrease in cut-set size

```
Pass = 0
While(Cumulative Gain, $G > 0$)
        Pass = Pass + 1
        Mark all nodes as not yet moved
        while (All the nodes have not been selected)
                Select node $a_i$ from Block A
                Select node $b_i$ from Block B
                Which maximize the gain $g_i$ on exchanging the nodes
                Mark $a_i$ and $b_i$ so that they are locked
        end while
        Choose k nodes to be exchanged which maximize $G = \sum_{i=1}^{k} g_i$
        Exchange nodes $a_1$ to $a_k$ with nodes $b_1$ to $b_k$
EndWhile /* end of a run */
```

Figure 2.4: The Kernighan-Lin Algorithm

may be obtained. Formulae are provided for computing and easily updating the gains so that the choice of cell to move next can be done efficiently.

Fiduccia and Mattheyses [Fidu82] modified the Kernighan and Lin heuristic by suggesting to move one cell at a time instead of exchanging pairs of vertices, and also introduced the concept of preserving balance in the size of blocks. This modification permitted a linear running time per pass for the network adaptation of the algorithm.

Krishnamurthy [Kris84] introduced a refinement of the Fiduccia and Mattheyses method for choosing the best cell to be moved. One disadvantage of the previously mentioned heuristics is that there is a large amount of unresolved nondeterminism. The heuristics choose arbitrarily between vertices that have *equal* gain and *equal* weight. In Krishnamurthy's algorithm the concept of look-ahead is introduced. This allows one to distinguish between such vertices with respect to gains they make possible in later moves.

Sanchis [Sanc89], uses the above technique for multiple way network partitioning. Under such a scheme, we should consider all possible moves of each free cell from its home block to any of the other blocks, at each iteration during a pass the best move should be chosen. As usual, passes are performed until no improvement in cut-set size is obtained. This strategy seems to offer some hope of improving the partition in a homogeneous way, by adapting the level gain concept to multiple blocks. In general, node interchange methods are greedy or local in nature and get easily trapped in local minimum.

Recently, several authors reported a ratio cut [Wei89] approach which removes the constraint on predefined subset sizes, and tends to identify natural clusters in the circuit. Given a network $N = (V,E)$, where V is the set of nodes and E is the set of edges, the objective is to partition V into disjoint U and W such that $e(U, W)/(|U| \times |W|)$ is minimized (where $e(U, W)$ is the number of edges in $\{(u, w) \in E | u \in U \text{ and } w \in W\}$). The ratio cut metric intuitively allows freedom to find "natural" partitions: the numerator captures the minimum-cut criterion, while the denominator favors an even partition. The main disadvantage of this approach is that subset sizes may be significantly different when the cut size is reduced. Therefore, ratio-cut may not be used when tight control on the subset sizes is required [Shin93].

2.2.3 Randomization Algorithms

Randomization is perhaps the oldest strategy for overcoming local optimality in combinatorial optimization, and classically takes two forms. The first is the well known "*random restart*" approach, which injects a randomizing element into the generation of an initial starting point to which a heuristic is subsequently applied. Depending on the nature of procedure for obtaining such a starting point, the "randomizing element" may be more systematic than random.

The second classical version of this approach is the "*random shake-up*" procedure which, instead of restarting, periodically generates a randomized series of moves that leads the heuristic from its customary path into a region it would not otherwise reach. In the framework commonly employed, a criterion is established for differentiating a move as improving or non-improving, and the purpose of the randomizing element may be viewed as that of admitting non-improving moves which would normally be disregarded.

Recently refinements of the random shake-up approach have attracted a good deal of attention. *Simulated Annealing* [Davi88, **?**], *Quantum Annealing* [Carv89], and *Genetic Algorithms* [**?**] have been heralded as new and powerful randomized methodologies for combinatorial problems. Some of these heuristics are introduced in detail in the next chapter.

2.3 Summary

This chapter introduced the main subproblems of circuit layout in the form of partitioning, placement and routing. Practically all these subproblems are intractable, they are NP-hard. An overview of the techniques that are based on exact and approximate algorithms to solve the combinatorial problems in circuit layout was introduced. Some of the severe drawbacks that face these traditional methods is the high computation time or poor performance due to locality of solutions. It is not enough to rely on the computing power of high-speed mainframe computers to overcome the combinatorial explosion. The key for dealing with such a problem is to go a step beyond the direct application of exact solution schemes or heuristics, and make recourse to a special procedure (or framework) which monitors and directs the use of these methods.

In the next chapter, we introduce some of the powerful methods that have emerged to handle complex optimization problems such as, Simulated Annealing, Genetic Algorithms, Tabu Search, and GRASP. These methods need not be viewed competitively, but complementary to each other, as we shall see, since they comprise the emergence of promise for conquering the combinatorial explosion in a variety of decision-making arenas including VLSI design.

Chapter 3

Advanced Search Techniques

Some of the problems that are faced by traditional heuristic methods are either, the vast amount of computation time required to solve a combinatorial optimization problem or the inferior quality of solutions due to getting trapped in local optimum. Recently, four approaches have emerged for handling such complex combinatorial optimization problems: *Simulated Annealing*, *Genetic Algorithms*, *Tabu Search*, and *GRASP*. The distinguishing feature for these techniques is the way they attempt to simulate some naturally-occurring process.

The motivation for the Simulated Annealing [**?**] algorithm comes from an analogy between the physical annealing of solids and combinatorial optimization problems. Simulated Annealing is widely recognized as a method for determining the global minima of combinatorial optimization problems. Tabu Search finds some of its motivation in attempts to imitate "intelligent" processes [Reev93], by providing heuristic search with a facility that implements a kind of "memory". Tabu search has been applied across a wide range of problem settings in which it consistently has found better solutions than other methods. GRASP is a random adaptive simple heuristic that intelligently constructs good initial solutions in an efficient manner. Genetic Algorithms on the other hand manipulate possible solutions to the decision problem in such a way that resembles the mechanics of natural selection and offers a number of advantages, the most important being the capability of exploring the parameter space.

In this chapter, the main concepts behind some of the most recent advanced search techniques to be used in this dissertation are introduced. In the description of these approaches in the next sections, the circuit partitioning problem is used as a paradigm for the circuit layout problem. The general strategy is to evaluate the different advanced search techniques in terms of quality of solutions and computational time. The different parameter settings that are used to obtain the tabulated results are presented. Next, the performance of the different heuristics is compared, highlighting their strengths and weakness. Section 3.1 introduces the main benchmarks that are used to evaluate the different heuristics used in this dissertation. Sections 3.2-3.6 introduce the main concepts of the advanced search heuristics.

3.1 Benchmarks

Some of the benchmarks used throughout this thesis to evaluate the performance of the hybrid algorithm are presented in Table 3.1. Chip1-Chip4 circuits are taken from the work of Fiduccia & Mattheyses [Fidu82]. The rest are taken from the MCNC gate array and standard cell test suite benchmarks [Robe87]. As seen in the table these netlists (hypergraphs) vary in size from 200 to 15000 nodes and 300 to 20000 nets. Tables 3.1-3.2 provide some information on the number of nets incident on each cell in the circuit and the number of cells that are contained within a net, and the average and maximum node degree and net sizes. The main purpose of extracting statistical information from the circuits is two fold. First, a clustering based heuristic developed in Chapter 5 utilizes this information to form clusters of cells, thus reducing the complexity of the circuit.

Circuit	Nodes	Nets	Pins	Node Degree			Net Size		
				MAX	$\overline{x}$	σ	MAX	$\overline{x}$	σ
Chip3	199	219	545	5	2.73	1.28	9	2.49	1.25
Chip4	244	221	571	5	2.34	1.13	6	2.58	1.00
Chip2	274	239	671	5	2.45	1.14	7	2.80	1.12
Chip1	300	294	845	6	2.82	1.15	14	2.87	1.39
Prim1	832	901	2906	9	3.50	1.29	18	3.22	2.59
Ind1	2271	2192	7743	10	3.41	1.19	318	3.53	9.00
Prim2	3014	3029	11219	9	3.72	1.55	37	3.70	3.82
Bio	6417	5711	20912	6	3.26	1.03	860	3.66	20.92
Ind2	12142	12949	47193	12	3.89	1.76	584	3.64	11.15
Ind3	15057	21808	65416	12	4.34	1.47	325	2.99	3.23

Table 3.1: Benchmarks used as test cases

Circuit	Nets Incident on Cell					Cells Incident on Net				
	1	2	3	4	≥ 5	2	3	4	5-19	≥ 20
Chip3	20%	31%	14%	27%	8.5%	83%	1.8%	6.8%	8.6%	0.0%
Chip4	23%	47%	7%	20%	3.3%	64%	24%	4.5%	7.2%	0.0%
Chip2	20%	41%	20%	12%	6.6%	57%	17%	18%	8.5%	0.0%
Chip1	11%	37%	17%	30%	5.3%	55%	24%	8.5%	12.1%	0.0%
Prim1	5.6%	18%	25%	33%	19.3%	55%	26%	6.9%	12.1%	0.0%
Ind1	1.5%	21%	35%	20%	21.5%	65%	16%	5.5%	12.9%	0.6%
Prim2	1.4%	15%	42%	17%	23.9%	61%	12%	6.7%	19.9%	0.4%
Bio	0.03%	13%	70%	6.9%	10.5%	69%	16%	7.5%	5.3%	2.2%
Ind2	1.3%	21%	24%	29%	24.3%	71%	14%	2.3%	11.5%	1.2%
Ind3	0.1%	5.8%	27%	21%	46.1%	57%	23%	8.5%	11.2%	0.3%

Table 3.2: Statistical information of benchmarks

Secondly, the statistical information is used as a means to improve the performance of the MIP CPLEX solver [Inc93] in reducing the computation time of the branch and bound heuristic. In addition, the information in Tables 3.1-3.2 indicates clearly that these circuits are well suited as benchmarks to test the partitioning heuristics that are developed.

It is important to note that the it full testing of all techniques discussed in this dissertation are presented in Appendix C. It is also worth mentioning that the results that are introduced in this dissertation are not compared to other results published thus far for the following reason. Most results mentioned in the literature are either based on partitioning techniques with a certain tolerance on the sizes of the blocks, or based on the ratio cut technique. Since, no tight restriction is imposed on the sizes, the quality of solutions vary according to the tolerance indicated. This makes it difficult to compare published results [Shin93, Wei89, Hage92, Yeh94, Chan94, Yeh95] with ours that are based on equi-sized partitions.

3.2 A Simple Dynamic Hill Climbing Heuristic

Iterative improvement techniques based on module interchange are the most robust, simple and successful heuristics in solving the partitioning and placement problems. The main disadvantage of these heuristics is that they mainly focus on the immediate area around the current initial solution, thus no attempt is made to explore all regions of the parameter space. More importantly, it has been shown that interchange methods fail to converge to "optimal" or "near optimal" solutions unless they initially begin from "good" initial starting points [Hadl92, Arei93, **?**]. Sechen [Sech88b] showed that over 100 trials or different runs were required to guarantee that the best solution would be within twenty percent of the optimum solution.

In this section, a modified implementation of the Sanchis iterative improvement heuristic is presented. It is characterized by the ability of escaping local optimum, which usually cause simple descent algorithms to terminate, by dynamically taking a different direction of a steepest ascent. In Chapter 2, the main concepts of the Sanchis [Sanc89] heuristic for multi-way graph partitioning was presented. At each iteration during a pass, the best move is chosen. Passes are performed until no improvement in cut-net size is obtained.

In SDHC, after the termination of the above heuristic, the heuristic considers all possible moves of each free cell from its home block to any of the other blocks, such that the value of the cut-size is increased. This is done such that the direction of the upward slope is different than the last pass performed. As seen in Figure 3.1 the heuristic continues to explore new regions until either cycling occurs or a certain number of passes have elapsed. Figure 3.2a compares the performance of the Sanchis heuristic to that of SDHC. The figure clearly indicates that once the Sanchis interchange technique stops at a local minima, SDHC focuses the search on other parts of the solution space to ensure that other regions are explored. It is worth noting that the complexity of the SDHC heuristic is similar to that of Sanchis (see Chapter 8). Figure 3.2b shows a comparison between a deterministic version of SDHC and a stochastic version. The performance of the simple dynamic hill climbing heuristic is compared to that of Sanchis multi-way partitioning heuristic. As can be seen in the Figure 3.3, the quality of solutions obtained by SDHC are far better than those obtained by the Sanchis heuristic. The quality of solutions obtained by SDHC as will be seen later (in Section 3.6.4) are inferior to that obtained by Simulated Annealing and Tabu Search heuristics. The main objective of SDHC is to explore small regions effectively in relatively short periods of time. For this

```
Pass = 0
**While** (Stopping Criteria is not met)
    Pass = Pass + 1
    *START DESCEND ROUTINE*
        Mark all nodes as not yet moved
        **While** (Modules can be Moved)
            Select node $a_i$ with highest gain
            **if** Balance Criteria is OK
              Move $a_i$ to destination block
              Mark $a_i$ as locked
            **End if**
        **end while**
        Choose k nodes, which maximize $G = \sum_{i=1}^{k} g_i$
        Perform Move on nodes $a_1$ *to* $a_k$
    *END DESCEND ROUTINE*
    *START ASCENT ROUTINE*
        Mark all nodes as not yet moved
        **While** (Modules can be Moved)
            Select node $a_i$ with lowest gain
            **if** Balance Criteria is OK
              Move $a_i$ to destination block
              Mark $a_i$ as locked
            **End if**
        **end while**
        Choose k nodes, which minimize $G = \sum_{i=1}^{k} g_i$
        Perform Move on nodes $a_1$ *to* $a_k$
        **If** (Cycling Occurs)
            Terminate Search
    *END ASCENT ROUTINE*
**EndWhile**
**Record Best Solution**
```

Figure 3.1: A dynamic hill climbing heuristic (SDHC)

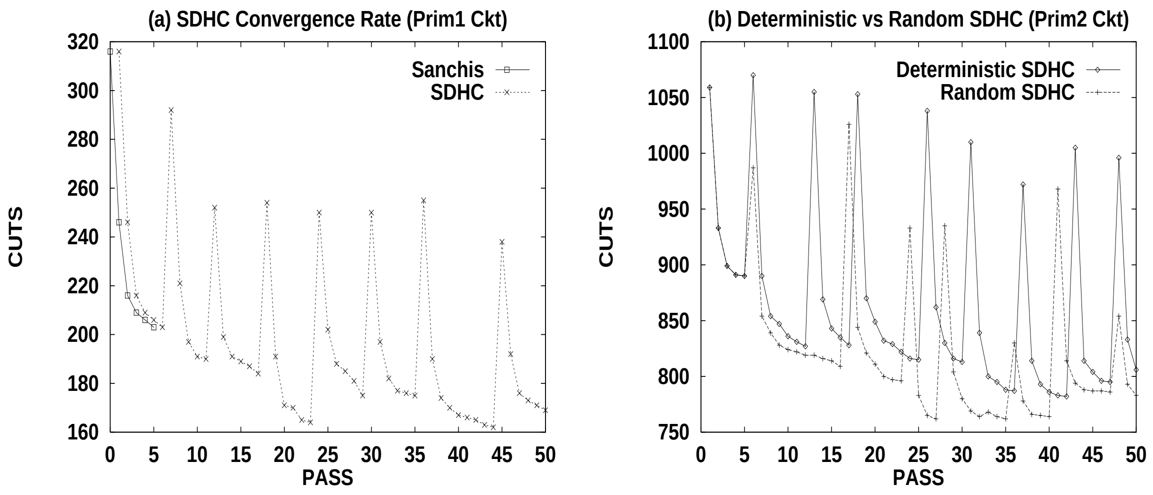


Figure 3.2: The convergence of SDHC

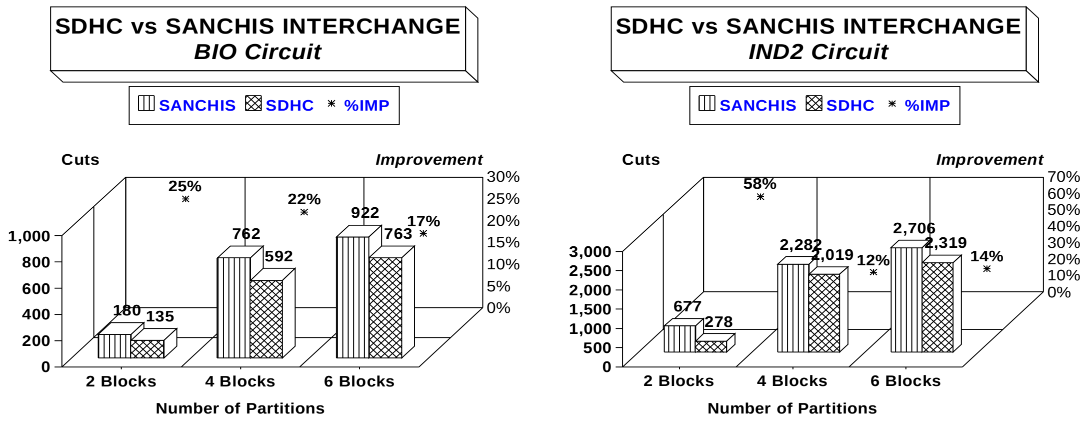


Figure 3.3: The performance of SDHC

reason, this heuristic will be utilized later on (in Chapter 5) as a means to refine the solutions produced by the clustering techniques developed, due to its simplicity and fast convergence.

3.3 Greedy Randomized Adaptive Search

GRASP is a greedy randomized adaptive search procedure that has been successful in solving many combinatorial optimization problems efficiently [Feo94]. The GRASP methodology was developed in the late 1980s, and the acronym was coined by Feo [?]. Each iteration consists of a construction phase and a local optimization phase. The key to success for local search algorithms consists of the suitable choice of a neighborhood structure, efficient neighborhood search technique, and the starting solution. The GRASP construction phase plays an important role with respect to this last point, since it produces good starting solutions for local search. The construction phase intelligently constructs an initial solution via an adaptive randomized greedy function. Further improvement in the solution produced by the construction phase may be possible by using either a simple local improvement phase or a more sophisticated procedure in the form of Tabu Search or Simulated Annealing.

Next, the various components comprising a GRASP are defined, and a demonstration of how to adapt such a heuristic for the circuit partitioning problem is presented.

3.3.1 Implementation

Figure 3.4 shows a generic pseudo-code of the GRASP heuristic. The main body of the GRASP algorithm starts by reading the circuit netlist. The algorithm starts with a construction phase followed by a local improvement phase. The GRASP implementation terminates after a certain number of phases or runs have passed. The construction phase as shown in Figure 3.4b is iterative, greedy and adaptive. It is *iterative* because the initial solution is built by considering one element at a time. The choice of the next element to be added is determined by ordering all elements in a list. The list of the best candidates is called the restricted candidate list (RCL). It is *greedy* because the addition of each element is guided by a greedy function. The construction phase is *randomized* by allowing the selection of the next element added to the solution to be any element in the RCL. Finally, it is *adaptive* because the element chosen at any iteration in a construction is a function of those previously chosen. The improvement phase typically consists of a local search procedure as shown in Figure 3.4c. A more sophisticated local search based on Tabu Search can be implemented instead of the simple local search procedure.

3.3.1.1 Construction Phase

Initially, all modules are placed into the same block and the gains associated with modules are calculated in an efficient manner. The discussion here will be based on four-way partitioning and this can be generalized for the multi-way partitioning case. Assume there are n modules and four blocks A, B, C, and D. The heuristic could either place all modules initially in block A and sequentially fill the other

```
      GREEDY RANDOMIZED SEARCH
(A) MAIN_GRASP()
  1. Read_Circuit_NetList();
  2. do
      3. Construction_Phase(Greedy,Rand,Adapt)
      4. Local_Improvement_Phase(Initial Solution)
      5. Store_Best_Solution(Previous Solutions)
      6. While not Done
  7. Report_Best_Solution()

(B) Construction_Phase()
  1. While  (construction not done)
      2. Greedy: Create Candidate List (RCL)
      3. Random: Module = Select from RCL
      4. Adaptive: Add new element to solution
      5. Feasibility: Check Feasibility of Solution
  6. EndWhile

(C) Local_Improvement_Phase()
  1. Read Initial Solution
  2. While  (local optimum not reached)
      3. NewSolution = Local_Changes(Solution)
  4. EndWhile
  5. Return_Best_Solution()
```

Figure 3.4: GRASP (Greedy Adaptive Search)

blocks by moving the n modules to blocks *B, C*, and *D*, or can create a dummy block (say X) and perform the same operation until block X is empty. At each iteration of the construction phase, the gains for moving modules to the current block being filled are examined, and an RCL list is created using the modules with the highest gains. As a module is moved it is locked to its new position (block) and its associated gain is removed from the bucket list. The gains of the other modules affected are updated accordingly. The construction phase terminates when a feasible solution (partition) is generated; i.e, all blocks contain a certain number of modules. The randomness in the GRASP heuristic is due to the selection strategy that is used to determine the next module to be appended to a certain block. The probabilistic component of the GRASP randomly chooses one of k best candidates in the restricted candidate list (RCL), but not necessarily the top candidate. This randomized selection strategy introduces diversification of initial solutions to the method.

The GRASP Parameters

The GRASP has two characteristics which make it appealing to researchers. First it is easy to implement, as seen from the previous section. Furthermore, only a few parameters need to be set and tuned. Therefore, development can focus on implementing efficient data structures to assure quick GRASP iterations. Some of the parameters that need to be tuned for the circuit partitioning problem are: *Block Selection Strategy*, *Gain Strategy*, and the *RCL Length*. The Block Selection Strategy determines the order by which blocks are filled to obtain an initial solution. In Random Selection Strategy (*RSS*), all modules are placed into the same block and the initial gains associated with moving modules to every other block are calculated.

The other blocks are filled randomly according to the best gain associated with the move involved. In Sequential Selection Strategy (*SSS*), all modules are placed into one block in a similar fashion to (*RSS*), but the other blocks are filled sequentially by removing excess modules from the initially oversized block and placing them into the current block under consideration. In Complete Selection Strategy (*CSS*), all modules are placed in a temporary block say X, and then every other block is filled until completion (all blocks meet the size constraint). As seen in Figure 3.5a[1] CSS gives the best performance with respect to RSS and SSS techniques.

The Gain Strategy determines the highest gain module to be assigned to a certain block. In Greedy Gain Strategy (*GGS*), the module with the highest gain is always selected and assigned to the appropriate block. In Random Gain Strategy (*RGS*) all modules are randomly selected from the RCL and assigned to the blocks according to the Block Selection Strategy used. Finally, the Biased Gain Strategy (*BGS*) is a combination of the above two methods. Figure 3.5b illustrates that BGS strategy works well for most circuits followed by GGS and RGS respectively.

The length of the RCL or restriction imposed on its values is a key success for the implementation of the algorithm. Each GRASP iteration produces a sample solution from an unknown distribution of all obtainable results. The mean and variance of the distribution are functions of the restrictive nature of the candidate list. For example, if the cardinality of the restricted candidate list is limited to one, then only one solution will be produced and the variance of the distribution will be zero. On the other hand if a less restrictive cardinality limit is imposed, many different solutions will be produced implying a larger variance. In [Feo94] two different restrictions are imposed on the RCL, Value Restriction (*RCL-VR*), and

[1]The gain has been normalized to one for the three circuits.

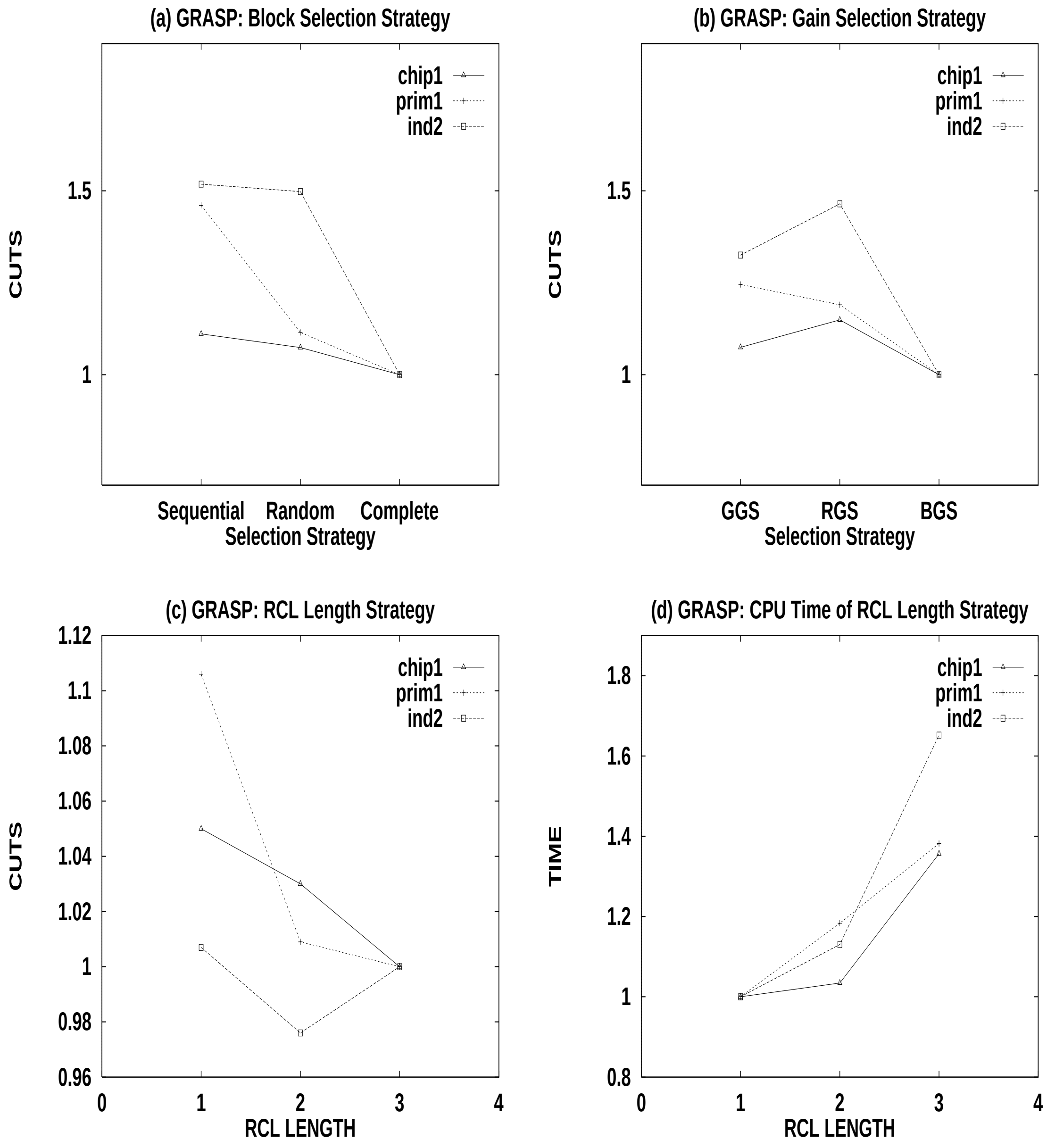


Figure 3.5: Parameters affecting GRASP performance

Cardinality Restriction (*RCL-CR*). In RCL-VR a module is allowed to be in the restricted candidate list if its gain is within some percentage (α) of the maximum gain. In RCL-CR, the candidate list size is limited by including only the (β) best elements. In this implementation, a third type of restriction based on a combination of RCL-CR and RCL-VR is used (RCL-CVR). Figures 3.5c,d show the performance of the GRASP in terms of computation time and quality of solution with different RCL lengths.

The results in table 3.3 assess the performance of Sanchis Interchange heuristic, to that of the GRASP. It is clear from Table 3.3 that the quality of solutions obtained by the GRASP using only 5 different runs are superior than those obtained using the Sanchis heuristic from 50 different starting points. The running times of the GRASP are also much faster since less initial solutions are used. Another reason for the fast computation time is that the local search heuristic has to perform less number of passes, since it is starting from a good initial solution, thus the fast convergence is obtained.

SANCHIS (50 runs) vs GRASP (5 runs)												
Blks	**PRIM1 Circuit**						**IND2 Circuit**					
	SAN		**GRASP**		**%IMP**		**SAN**		**GRASP**		**%IMP**	
	C	T	C	T	C	(T)	C	T	C	T	C	(T)
2 Blks	60	91	56	7.2	6%	92%	593	2661	325	155	45%	94%
4 Blks	155	96	127	12.4	18%	87%	2102	12729	1148	312	45%	97%
6 Blks	181	133	153	18.1	15%	86%	2430	25132	1464	1066	39%	95%

Table 3.3: A comparison between GRASP and Sanchis interchange heuristic

3.4 Genetic Algorithms

Genetic Algorithms **(GA's)** are a class of optimization algorithms that seek improved performance by sampling areas of the parameter space that have a high probability for leading to good solutions [Venk91]. The algorithms are called genetic because the manipulation of possible solutions resembles the mechanics of natural selection. These algorithms which were introduced by Holland [**?**] in 1975 are based on the notion of propagating new solutions from parent solutions, employing mechanisms modeled after those currently believed to apply in genetics. The best offspring of the parent solutions are retained for a next generation of mating, thereby proceeding in an evolutionary fashion that encourages the survival of the fittest.

3.4.1 An Overview of Genetic Search

As an optimization technique, Genetic Algorithms simultaneously examine and manipulate a set of possible solutions. Each candidate solution is represented by a string of symbols called a chromosome. The set of solutions P_j, is referred to as the population of the j^{th} generation. The population evolves for a prespecified total number of generations under the application of evolutionary rules called *Genetic Operators*.

3.4.1.1 Characteristics of Genetic Search

There are many characteristics of Genetic Algorithms which qualify them to be a robust based search procedure. The first feature of Genetic Algorithms is that they are characterized to climb many peaks in parallel. Thus, the probability of finding a false peak is reduced over methods that proceed form point to point in the decision space. Secondly, the operators make use of a coding of the parameter space rather than the parameters themselves. Only objective function information is used, this results in a simpler implementation. Finally, although the approach has a stochastic flavor, it makes use of all the information that has been obtained during the search, and permits the structured exchange of that information [?].

3.4.1.2 Main Components of Genetic Search

There are essentially four basic components necessary for the successful implementation of a Genetic Algorithm. At the outset, there must be a code or scheme that allows for a bit string representation of possible solutions to the problem. Next, a suitable function must be devised that allows for a ranking or fitness assessment of any solution. The third component, contains transformation functions that create new individuals from existing solutions in a population. Finally, techniques for selecting parents for mating, and deletion methods to create new generations are required.

3.4.1.3 Representation Module

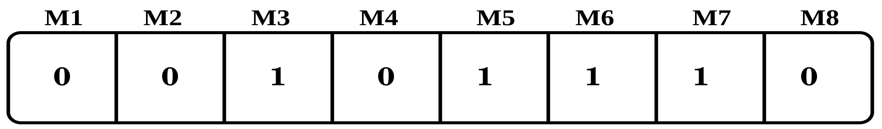
In the original GA's of Holland [?], each solution may be represented as a string of bits, where the interpretation of the meaning of the string is problem specific. As can be seen in Figure 3.6a, one way to represent the partitioning problem is to use *group-number encoding* where the j^{th} integer $i_j \in \{1, \ldots, k\}$ indicates the group number assigned to object j. This representation scheme creates a possibility of applying standard operators [Mich92]. However an offspring may contain less than k groups; moreover, an offspring of two parents, both representing feasible solutions may be infeasible, since the constraint of having equal number of modules in each partition is not met. In this case either special *repair heuristics* are used to modify chromosomes to become feasible, or *penalty functions* that penalize infeasible solutions, are used to eliminate the problem. These schemes will be explained in detail in Section 3.4.2.2. The second representation scheme is shown in Figure 3.6b. Here, the solution of the partitioning problem is encoded as $n + k - 1$ strings of distinct integer numbers. Integers from the range $\{1, .., n\}$ represent the objects, and integers from the range $\{n + 1, \ldots, n + k - 1\}$ represent separators; this is a *permutation with separators* encoding. This representation scheme leads to 100% feasible solutions [Mich92], but requires more computation time due to the complexity of the unary operator involved.

3.4.1.4 Evaluation Module

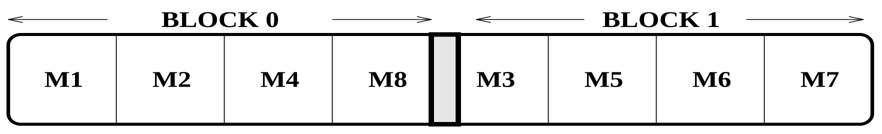
Genetic Algorithms work by assigning a value to each string in the population according to a problem-specific *fitness* function. It is worth noting that nowhere except in the evaluation function is there any information (in the Genetic Algorithm) about the problem to be solved. For the circuit partitioning problem, the evaluation function measures the worth (number of cuts) of any chromosome (partition) for the circuit to be solved.

3.4.1.5 Reproduction Module

This module is perhaps the most significant component in the Genetic Algorithm. Operators in the reproduction module, mimic the biological evolution process, by using unary (mutation type) and higher order (crossover type) transformation to create new individuals. *Mutation* as shown in Figure 3.6c is simply the introduction



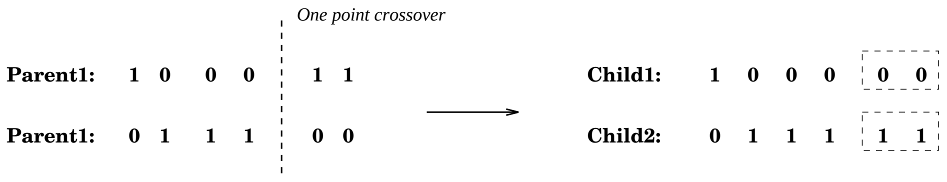
(a) Group Number Encoding



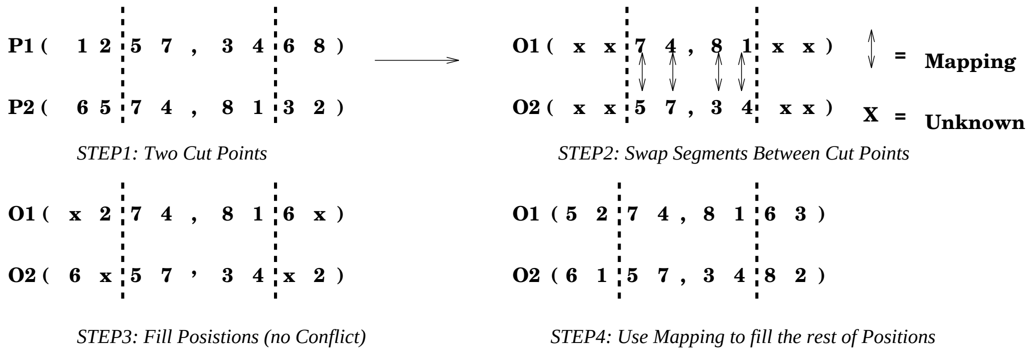
(b) Permutation with Separator Encoding.

Old Chromosome				Random Numbers				New Bit	New Chromosome			
1	0	1	0	.801	.102	.266	.373	-	1	0	1	0
1	1	0	0	.120	.096	$\underline{.005}$	.840	0	1	1	$\boxed{0}$	0
0	0	1	0	.760	.473	.894	$\underline{.001}$	1	0	0	1	$\boxed{1}$

(c) Standard Mutation Operator



(d) Standard Crossover Operator (for group number encoding)



(e) PMX Operator (for permutation with separators encoding)

Figure 3.6: Representation schemes and genetic operators

of a random element, that creates new individuals by a small change in a single individual. When mutation is applied to a bit string, it sweeps down the list of bits, replacing each by a randomly selected bit, if a probability test is passed. On the other hand, *crossover* recombines the genetic material in two parent chromosomes to make two children. It is the structured yet random way that information from a pair of strings is combined to form an offspring.

Crossover begins by randomly choosing a cut point K where $1 \leq K \leq L$, and L is the string length. The parent strings are both bisected so that the leftmost partition contains K string elements, and the rightmost partition contains $L - K$ elements. The child string is formed by copying the rightmost partition from parent P_1 and then the leftmost partition from parent P_2. Figure 3.6d shows an example of applying the standard crossover operator (sometimes called one-point crossover) to the group number encoding scheme. Increasing the number of crossover points is known to be multi-point crossover. The mutation and crossover operators as described above, apply for the first representation scheme "group number encoding". These operators are modified for the "permutation with separator encoding" scheme. A mutation in this case, would swap two objects (separators excluded). The crossover operator considered is the partially matched crossover (PMX) [Mich92]. As shown in Figure 3.6e, PMX builds an offspring by choosing a sub-partition of a solution from one parent, and preserving the position of as many modules as possible from the other parent. A sub-partition of the solution is selected by choosing two random cut points, which serve as boundaries for swapping operations. Figure 3.6e illustrates this process in detail. Generally, the results of the Genetic Algorithms based on permutation with separators encoding are better than those based on group-number encoding, but take a longer time to converge

[Mich92].

3.4.1.6 Population Module

This module contains techniques for population initialization, generation replacement, and parent selection techniques. The initialization techniques generally used are based on pseudorandom methods. The algorithm will create its starting population by filling it with pseudorandomly generated bit strings.

Strings are selected for mating based on their fitness, those with greater fitness are awarded more offspring than those with lesser fitness. Parent selection techniques that are used, vary from stochastic to deterministic methods. The probability that a string i is selected for mating is p_i, "the ratio of the fitness of string i to the sum of all string fitness values", $p_i = \frac{fitness_i}{\sum_j fitness_j}$. The ratio of individual fitness to the fitness sum denotes a ranking of that string in the population. The Roulette Wheel Selection method is conceptually the simplest stochastic selection technique used. The ratio p_i is used to construct a weighted roulette wheel, with each string occupying an area on the wheel in proportion to this ratio. The wheel is then employed to determine the string that participates in the reproduction. A random number generator is invoked to determine the location of the spin on the roulette wheel. In Deterministic Selection methods, reproduction trials (selection) are allocated according to the rank of the individual strings in the population rather than by individual fitness relative to the population average.

Generation replacement techniques are used to select a member of the old population and replace it with the new offspring. The quality of solutions obtained depends on the replacement scheme used. Some of the replacement schemes used are based on: (i) deleting the old population and replacing it with new offsprings

(GA-dop), (ii) replacing parent solutions with sibling (GA-rps), (iii) replacing the most inferior members (GA-rmi) in a population by new offsprings. Variations to the second scheme use an incremental replacement approach, where at each step the new chromosome replaces one randomly selected from those which currently have a *below-average* fitness. The quality of solutions improve using the second replacement scheme. The reason is that this replacement scheme maintains a large diversity in the population.

3.4.2 GA Implementation

Figure 3.7 shows a simple Genetic Algorithm. The algorithm begins with an encoding and initialization phase during which each string in the population is assigned a uniformly distributed random point in the solution space. Each iteration of the genetic algorithm begins by evaluating the fitness of the current generation of strings. A new generation of offspring is created by applying crossover and mutation to pairs of parents who have been selected based on their fitness. The algorithm terminates after some fixed number of iterations.

3.4.2.1 Parameters affecting the performance of Genetic Search

Running a Genetic Algorithm entails setting a number of parameter values. Finding settings that work well on one's problem is not a trivial task. If poor settings are used, a Genetic Algorithm's performance can be severely impacted. Central to these components are questions pertaining to appropriate representation schemes, lengths of chromosome strings, optimal population sizes, and frequency with which the transformation functions are invoked. Figure 3.8 shows some of the parameters that affect the performance of the Genetic Algorithm. As the number of generations increase the quality of solutions improve, but the computation time involved increases also. Choosing the population size for a Genetic Algorithm is a funda-

```
GENETIC ALGORITHM
1. Encode Solution Space
2.(a) set pop_size, max_gen, gen=0;
  (b) set cross_rate, mutate_rate;
3. Initialize Population.
4. **While** max_gen $\geq$ gen
        Evaluate Fitness
        **For** (i=1 to pop_size)
          Select (mate1,mate2)
          if (rnd(0,1) $\leq$ cross_rate)
            child = Crossover(mate1,mate2);
          if (rnd(0,1) $\leq$ mutate_rate)
            child = Mutation();
          Repair child if necessary
        **End For**
        Add offsprings to New Generation.
        gen = gen + 1
   **End While**
5. Return best chromosomes.
```

Figure 3.7: A generic Genetic Algorithm

mental decision faced by all GA users. On the one hand, if too small a population size is selected, the Genetic Algorithm will converge too quickly. On the other hand, a population with too many members results in long waiting times for significant improvement, especially when evaluation of individuals within a population must be performed wholly or partially in serial. Regarding the reproduction module, experimental data confirms that mutation rates above 0.04 are generally harmful with respect to on-line performance. The absence of mutation is also associated with poorer performance, which suggests that mutation performs an important service in refreshing lost values. Good on-line performance is associated with high crossover rate combined with low mutation rate.

3.4.2.2 Performance of Genetic Algorithm

Section 3.4.1.3 introduced two methods for solving the problem of producing infeasible solutions using the Genetic Algorithm. The first is based on a penalty function, where infeasible solutions are penalized such that their fitness is decreased according to the deviation from the feasible solution required. The second method is based on repairing the infeasible solutions produced by crossover and mutation. To repair a corrupted chromosome, one could either use a *simple repair scheme* where extra genes belonging to a certain block are randomly moved to other unbalanced blocks, or a more *efficient repair scheme* is used, where genes are moved to unbalanced blocks such that the gain is increased (cut-net size is decreased).

Figure 3.9 shows the results obtained by the Genetic Algorithm using a penalty function, a simple repair heuristic and an efficient repair heuristic respectively.

The solutions obtained by the simple repair solution are much better than those obtained by the penalty function in terms of quality. The more advanced repair technique gives the best results with respect to the two other approaches mentioned. Figure 3.9 also presents the improvement achieved from using the advanced repair heuristic over the penalty function implementation.

3.4.2.3 Comparison of different Constructive Methods

In this section, the performance of the Genetic Algorithm is compared to results obtained by Barnes's eigenvector technique [Barn82] and the GRASP heuristic [**?**]. The comparison is based on good initial solutions obtained by each heuristic and

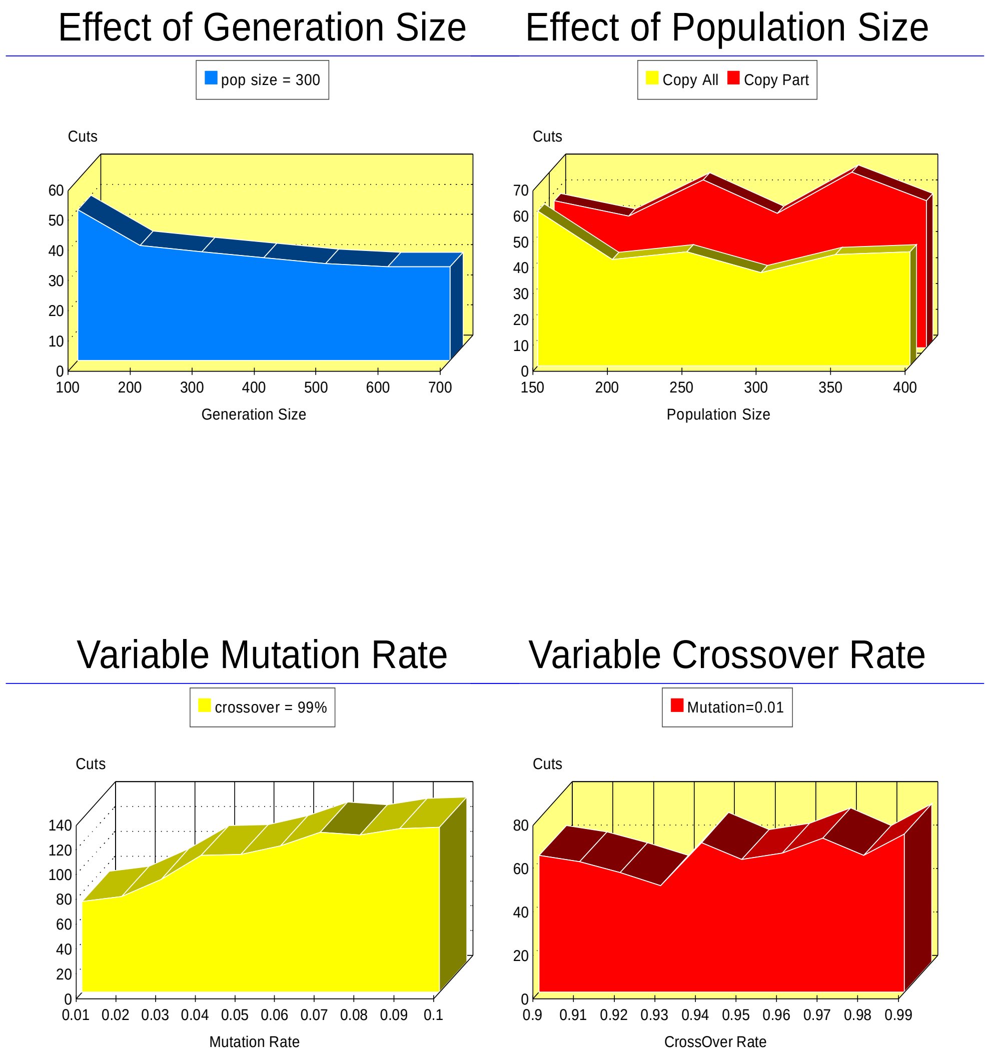
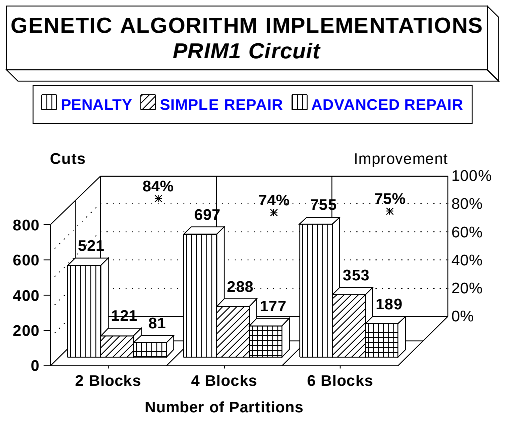


Figure 3.8: Parameters affecting GA performance



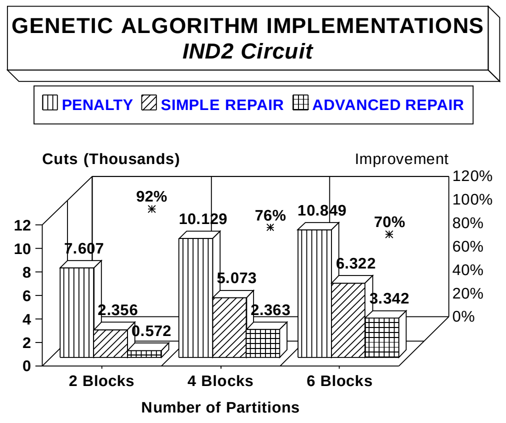


Figure 3.9: The GA performance

the computation time involved to obtain these solutions. Table 3.4 presents results of Barnes's eigenvector algorithm, GRASP and Genetic Algorithms for producing good initial solutions. The CPU time for the partitions obtained by the Barnes's algorithm include the time for forming the graph adjacency matrix, finding the eigenvalues and eigenvectors. It is interesting to note that most solutions obtained by the GRASP and Genetic Algorithm are superior to those obtained by the eigenvector approach except for *Bio* circuit. The computation time used by the GRASP is the least compared to the two other methods. The last column in the table presents the improvement of solution quality using the eigenvector approach and the Genetic Algorithm. On average the Genetic Algorithm reduces the amounts of cuts by 60% compared to the eigenvector approach. Also, eigenvector techniques can be slow to converge, whereas GA's are more stable and give superior results.

CONSTRUCTIVE BASED METHODS								
Circuit	**Blks**	**Eigenvector**		**GRASP**		**GA**		**%IMP**
		Cuts	Time	Cuts	Time	Cuts	Time	Cuts
Prim1	2 Blks	181	10.4	101	1.0	81	9.8	55%
	4 Blks	298	13.5	185	2.2	177	14.1	40%
	6 Blks	329	22.4	203	2.7	189	18.6	42%
Ind3	2 Blks	3188	151	1819	30.9	930	222	70%
	4 Blks	6627	647	2456	70	2444	365	63%
	6 Blks	7254	653	3521	149	2890	495	60%

Table 3.4: A comparison between constructive techniques

3.5 Simulated Annealing

Simulated Annealing **(SA)** searches the solution space of a combinatorial optimization problem, with the goal of finding a solution of minimum cost value. The motivation for the Simulated Annealing algorithm comes from an analogy between the physical annealing of solids and combinatorial optimization problems [**?**]. Physical annealing refers to the process of finding low energy states of a solid by initially melting the substance, and then lowering the temperature slowly, spending a long time at temperatures close to the freezing point. In the analogy, the different states of the substance correspond to the different feasible solutions of the combinatorial optimization problem, and the energy of the system corresponds to the function to be minimized.

A simple descent algorithm corresponds to "rapid quenching" where the temperature is reduced quickly so that only moves which result in a reduction of the energy of the system are accepted. In Simulated Annealing, the algorithm alternatively attempts to avoid becoming trapped in a local optimum by sometimes accepting a neighborhood move which increases the value of *(f)* as seen in Figure 3.10. The acceptance or rejection of an uphill move is determined by a sequence of random numbers, but with a controlled probability. The probability of accepting a move which causes an increase δ in *(f)* is called the acceptance function and is normally set to $e^{(-\delta/T)}$ where T is a control temperature in the analogy with physical annealing. This acceptance function implies that small increases in f are more likely to be accepted than large increases, and that when T is high, most moves will be accepted, but as T approaches zero most uphill moves will be rejected.

```
current_solution ← initial_solution
current_cost ← evaluate(current_solution)
T ← $T_{initial}$
While ($T \geq T_{final}$)
    for i = 1 to iteration(T) /* neighborhood moves */
        new_solution ← move(current_solution)
        new_cost ← evaluate(new_solution)
        $\Delta cost$ ← new_cost - current_cost
        $if(\Delta cost \leq 0\ \ OR\ \ e^{-\Delta cost/T} > \underline{random()})$
            /* accept new solution */
            current_solution ← new_solution
            current_cost ← new_cost
        EndIf
    EndFor
    T ← next_temp(T)
EndWhile
```

Figure 3.10: A Simulated Annealing Algorithm

In Simulated Annealing, the single loop of a descent algorithm is replaced by a double loop; in the outer loop the temperature is changed and the inner loop determines how many neighborhood moves are to be attempted at each temperature. The determination of the initial temperature, the rate at which the temperature is reduced, the number of iterations at each temperature and the criterion used for stopping is known as the *annealing schedule* [**?**]. The choice of annealing schedule has an important bearing on the performance of the algorithm [Naha85].

3.5.1 Annealing Schedule

The Simulated Annealing algorithm, in its original formulation converges with probability **one** to a globally minimal configuration in either one of the following cases [Laar88]:

1. for each value of the control parameter, T_k, an infinite number of transitions is generated and $\lim_{k\to\infty} T_k = 0$. (the homogeneous algorithm);
2. for each value T_k one transition is generated and T_k goes to zero not faster than $O([\log k]^{-1})$ (the inhomogeneous algorithm).

Certain conditions on the generation and acceptance matrices should also be satisfied to ensure the existence of a stationary distribution of a homogeneous Markov chain [Laar88]. In any implementation of the algorithm, asymptotic convergence can only be approximated. Thus, though the algorithm is asymptotically an optimization algorithm, any implementation results in an approximation algorithm. The number of transitions for each value T_k, for example, must be finite and $\lim_{k\to\infty} T_k = 0$ can only be approximated in a finite number of values for T_k. Due to these approximations, the algorithm is no longer guaranteed to find a global minimum with probability one.

Initial Value of the Control Parameter

The initial value of T is determined in such a way that virtually all transitions are accepted, i.e., T_0 is such that $exp(-\delta cost_{ij}/T_0) \simeq 1$ for almost all i and j. The following empirical rule is proposed: choose a large value for T_0 and perform a number of transitions. If the acceptance ratio χ, defined as the number of accepted transitions divided by the number of proposed transitions, is less than a given value χ_0 (in [**?**] $\chi_0 = 0.8$), double the current value of T_0. Continue this procedure until the observed acceptance ratio exceeds χ_0.

Final Value of the Control Parameter

A stopping criterion, determining the final value of the control parameter, is either determined by fixing the number of values T_k, for which the algorithm is to be executed, or by terminating execution of the algorithm if the last configuration of consecutive Markov chains are identical for a number of chains.

Number of Iteration per Temperature

The simplest choice for L_k, the length of the k^{th} Markov chain, is a value depending (polynomial) on the size of the problem.

Decrement of the Control Parameter

A frequently used decrement rule is given by $c_{k+1} = \alpha \times c_k,\ k = 0, 1, 2, ...$ where α is a constant close to 1.

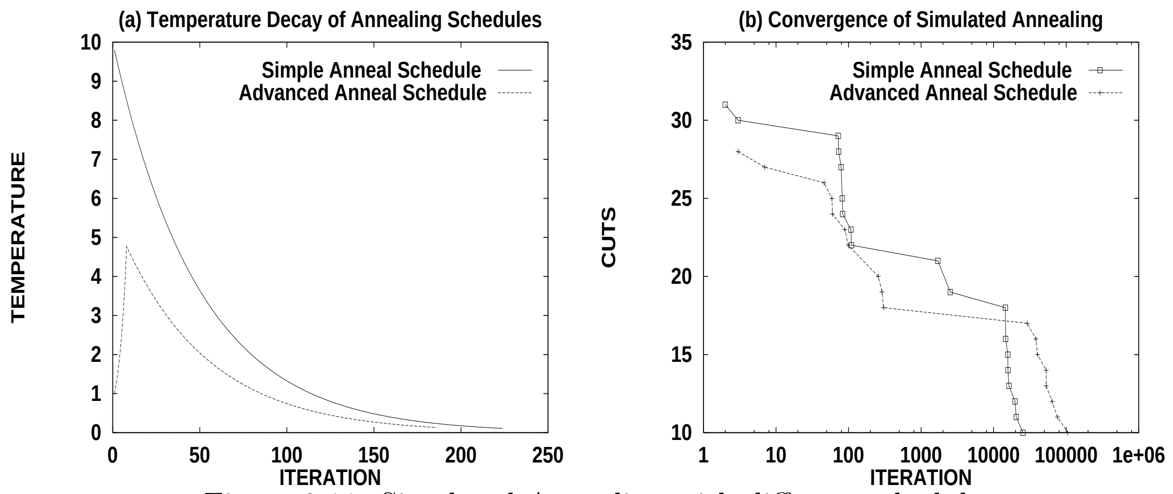
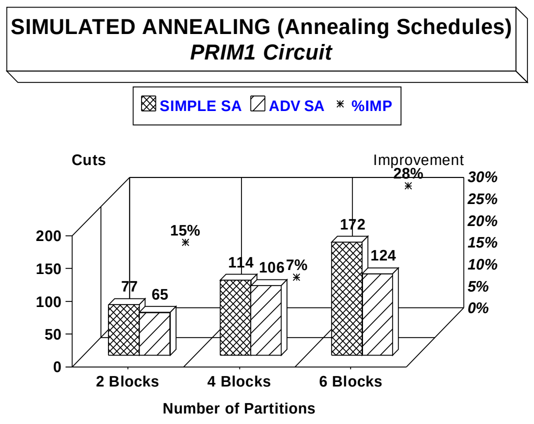


Figure 3.11: Simulated Annealing with different schedules

3.5.1.1 Advanced Annealing Schedule

A more advanced annealing schedule proposed by White [Whit84] is also used to evaluate the performance of Simulated Annealing. The most important settings used are the stopping criterion and the length of chains during a certain temperature. In Figure 3.11, a comparison between the simple cooling schedule proposed by Kirkpatrick [?] and White [Whit84] is shown. Figure 3.11a illustrates the means of computing the initial temperature in both schedules. Figure 3.11b shows the convergence rate for a small circuit based on both schedules. Figure 3.12 presents results obtained by the two annealing schedules mentioned above. The performance of the Simulated Annealing using the advanced annealing schedule proposed by White [Whit84] is superior to that based on a simple annealing heuristic. Thus far, the Simulated Annealing heuristic provides the best results compared to other traditional and advanced search techniques mentioned, but at the expense of huge CPU time.



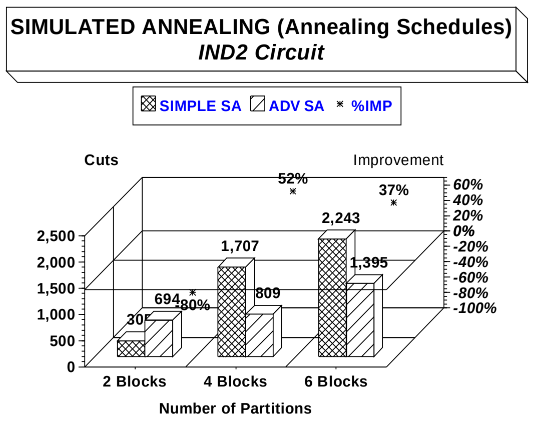


Figure 3.12: A comparison between annealing schedules

3.6 Tabu Search

Tabu Search is a general heuristic procedure for global optimization. Based on simple ideas it has been extremely efficient in getting almost optimal solutions for many types of difficult combinatorial optimization problems ranging from graph partitioning [Arei93] [?] [Tao91], graph coloring [Hert87], to quadratic assignment problems [SK90]. Tabu Search is based on the premise that problem solving, in order to qualify as intelligent, must incorporate adaptive memory and responsive exploration. The use of adaptive memory contrasts with "memoryless" designs, such as those inspired by metaphors of physics and biology (Simulated Annealing), and with "rigid memory" designs, such as those exemplified by branch and bound and its AI-related algorithms.

3.6.1 Tabu Search Main Foundation

The basis for Tabu Search may be described as follows. Given a function $f(x)$ to be optimized over a set X, Tabu Search begins in the same way as ordinary local search, proceeding iteratively from one point (solution) to another until a chosen termination criterion is satisfied. Each $x \in X$ has an associated neighborhood $N(x) \subset X$, and each solution $x^* \in N(x)$ is reached from x by an operation called a **move**. Tabu Search goes beyond local search by employing a strategy of modifying $N(x)$ as the search progresses, effectively replacing it by another neighborhood $N^*(x)$. The key aspect of Tabu Search is the use of special memory structures which serve to determine $N^*(x)$, and hence to organize the way in which the space is explored.

3.6.1.1 Explicit and Attributive Memory

The memory used in Tabu Search is both explicit and attributive. Explicit memory records complete solutions, typically consisting of elite solutions visited during the search. These special solutions are introduced at strategic intervals to enlarge $N^*(x)$, and thereby provide useful options not in $N(x)$. The Tabu Search memory is also designed to exert a more subtle effect on the search through the use of attributive memory, which records information about solution attributes that change in moving from one solution to another. For example in a graph or network setting, attributes can consist of nodes or arcs that are added, dropped or repositioned by the moves executed.

3.6.1.2 Short Term Memory

It is the feature of allowability whereby some moves are not allowed "they are forbidden or made Tabu" which distinguishes Tabu Search from other descent methods. Allowability is managed by a mechanism that involves historical information about moves made as the routine progresses; moves accepted for an arbitrarily defined number of previous iterations are deemed not allowable or Tabu, because to allow one of them may trap the routine into cycling through moves already taken.

Tabu Move

There are different attributes that can be used in creating the short term memory of Tabu lists for the circuit partitioning problem. One possibility is to identify attributes of a move based on the module value to be swapped from one block to another. Another way of identifying attributes of a move is to introduce additional information, referring not only to the modules to be moved but to positions (blocks) occupied by these modules. The recorded move attributes are used in Tabu Search to impose the constraints, called Tabu restrictions, that prevent moves from being chosen. Examples of Tabu restrictions employed are as follows: (i) Restrictions based on module movements (**TC1**). This is considered to be the most rigid restriction since once a module moves from one block to another it is not moved until it is released from the Tabu list. (ii) Restrictions based on module and source block (position of module) (**TC2**). Here, the restriction applies to movement of the module and its source block **X**, but is free to move to other blocks. (iii) Restrictions based on module and destination block (**TC3**). Finally, a combination of the above restrictions is implemented using (**TC4**). The fourth restriction is considered to be the most lenient.

Tabu List

Tabu list management concerns updating the Tabu list; i.e., deciding on how many and which moves have to be made Tabu within any iteration of the search. Figure 3.13a shows the quality of solutions obtained as the size of the Tabu list is increased. The size of the Tabu list can noticeably affect the final results; a long list may give a lower overall minimum cost, but is usually obtained in a long time. Further, a shorter list may trap the routine in a cyclic search pattern. Our empirical results show that Tabu list sizes that provide good results, often grow with the size of the problem. Figure 3.13b shows the Tabu search convergence rate as a function of the Tabu list length. The longer lists (16, 24) give a lower overall minimum partition but is obtained in a longer time. The Tabu list of length (4) on the other hand got trapped in a cyclic search pattern. An appropriate list size depends on the strength of the Tabu restrictions employed. The sizes of the Tabu lists will be discussed in Section 3.6.2.

Aspiration

To increase the flexibility of the algorithm, while preserving the basic features that allow the algorithm to escape local optimum, and avoid cyclic behavior, aspiration is used to temporarily release a solution from its Tabu status. The aspiration criterion plays an important role to achieve good performance. The appropriate use of it can be crucial to the success of the Tabu Search algorithm. Different applications employ only simple types of aspiration criterion. In the current implementation, two different methods are used. The first actual aspiration rule (**ASP1**) is that, if the cost associated with a Tabu solution is less than the aspiration value associated with the cost of the current solution, then the Tabu status of the Tabu solution is temporarily ignored. That is, although the Tabu solution is not removed from the Tabu list, its Tabu status is overridden, and a move to the Tabu solution may be made. The second aspiration rule (**ASP2**) that is used consists of removing a move classified as Tabu when the move yields a solution better than the best obtained so far. Figure 3.13c shows the effect of the aspiration on the overall solution (cut-net) using the Chip1 circuit. As the number of blocks increase, the effect of the aspiration level has more impact. The Tabu restrictions and aspiration level criterion of Tabu Search play a dual role in constraining and guiding the search process.

3.6.1.3 Intermediate and Long Term Memory

Intermediate and long term memory functions are employed within Tabu Search to achieve regional intensification and global diversification of the search [**?**, Glov90]. Combined with the short term memory functions, intermediate and long term memory functions provide an interplay between "exploitation" and "exploration" of the solution space [**?**] (fine tuned search versus exploration of the solution space). Intermediate term memory operates by recording and comparing features of a selected number of best trial solutions generated during a particular period of search. The method then seeks new solutions that exhibit these features. The long term memory functions, whose goal is to diversify the search, employs principles that are roughly the reverse of those for intermediate term memory. Figure 3.13d shows a recording of the module movement during the search procedure.

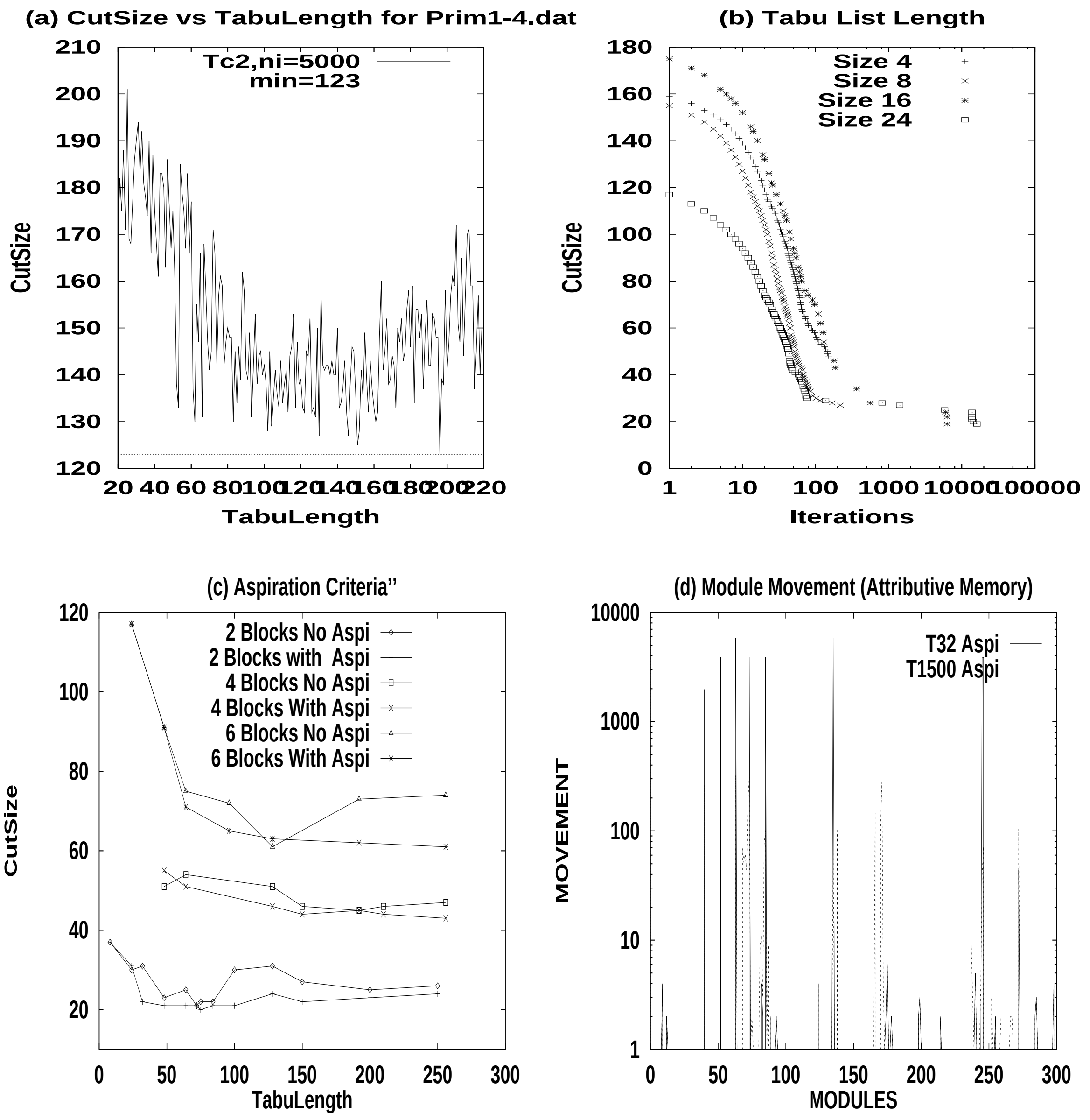


Figure 3.13: Parameters affecting Tabu Search

3.6.2 Tabu Search Implementation for Partitioning

The Tabu Search routine described so far can be formulated as shown in Figure 3.14. The algorithm requires an initial feasible solution (partition) for which an associated cost cut may be calculated. A size for the list of Tabu solutions is also required, (*tabu_list_size*) and a maximum number of moves, (*max_num_iter*) after which the routine terminates.

3.6.2.1 Tabu List Size

Our Tabu list management techniques are based on static and dynamic approaches. In the static approach, the size of the Tabu lists remains fixed for the entire search period. Single or multiple attributes are set Tabu as soon as their complements have been part of a selected move. The attributes stay Tabu for a distinct number of iterations. The efficiency of the algorithm depends on the choice of the Tabu status duration (the size of the underlying Tabu list). The size of the Tabu list is chosen to be a function of the number of nodes within the circuit to be partitioned. Our experimentation with the Tabu Search algorithm indicates that choosing a $tabu\_list\_size = \alpha \times nodes$ (where α ranges from 0.1 to 0.2) yields good results in most cases. The static approach, though successful for some circuits, seems to be a rather limited one. The dynamic implementation allows the size of the Tabu list to oscillate between two ranges. The first range is determined when cycling occurs (Tabu lists are too short). The second range is determined when deterioration in solution quality occurs, which is caused by forbidding too many moves (Tabu lists are too large). Best sizes lie in an intermediate range between these extremes.

```
Input:
    The net list or the Graph G= (V,E)
    K = partitions required; $| T |$ = size of Tabu list
    max_num_iter = max iterations allowed.
Initialization:
    Initial Partition = Generate a random solution
    $s = (V_1, V_2.., V_k)$; num_iter=0;
    bestpart=s; bestcut=f(s);
Main Loop:
   While (num_iter < max_num_iter)
     Pick best module associated with best gain
     If (move not in Tabulist) then
        Accept move ,Update Tabulist;
        Update the Aspiration Level;
     If (move in Tabulist) then
        If (Cost(tabu_sol) < Aspiration($curr\_sol$)) then
          Override the TL Status and Accept the move
          Update TL; Update the Aspiration Level;
        Else
            Move not accepted;
     num_iter = num_iter + 1;
    End While
Output:
    Best Partition = bestpart; Best Cut = bestcut
```

Figure 3.14: A Simple Tabu Search implementation

3.6.2.2 Stopping Criteria

The stopping conditions used in this implementation are based on the following: (i) The search will terminate when "num_iter" is greater than the maximum number of iterations allowed "max_num_iter". (ii) The search will terminate, if no improvement on the best solution found so far can be made, for a given number "max_num_iter" of consecutive iterations. The maximum number of iterations after which the routine terminates, depends on whether the routine starts from a random starting point or a good initial starting point (as will be explained in then next chapter). Experiments performed show the following: For random starting points, the algorithm requires more iterations to converge to good final solutions, so the maximum number of allowable iterations is set to "$max\_num\_iter = 100 \times nodes$", whereas for good starting points "$max\_num\_iter = 20 \times nodes$". The final report gives the solution with the overall best partition and best cut after the specified maximum number of moves.

3.6.2.3 Different Tabu Search Implementations

In this section, the results obtained from the Tabu Search heuristic using different parameter settings are discussed.

Delayed Tabu List Activation (TS-DA)

The Tabu lists in this setting are not activated "no moves are considered to be tabu" until the algorithm hits the first local minima. This is the delay activation (DA). To achieve that without cycling to previous solutions, the interchange method using a certain number of passes (as described in Chapter 2) is used. Once a local minimum is reached, the Tabu lists are activated and the Tabu Search algorithm resumes exploration for better solutions through the short term memory.

Long and Intermediate Term Memories(TS-DS-IS)

In this setting, the Tabu Search algorithm uses the best Tabu criterion and the most suitable aspiration rule. At the same time, intermediate and long term memory are employed to intensify and diversify the search.

Comparison between different settings

Table 3.5 presents results obtained using the Tabu Search under the different settings described. **TS-DA** represents Tabu Search with delay activation. The second column of the table represents Tabu Search using the first Tabu restriction. **TS-ASP2** shows results obtained using the second aspiration rule (described in Section 3.6.1.2). It should be noted that the best results among the implementations using the short term memory are based on **TS-DA** using TC1 and ASP1. Results obtained using search diversification and intensification are considered best among the others but, on the expense of extended CPU time.

3.6.3 Adaptive Tabu Search Heuristic

Advances in the development and refinement of general and advanced search strategies depend in part on identifying the type of adaptation to a specific problem domain that will prove most effective. A worthwhile avenue for research relative to combinatorial algorithm development is the issue of fine-tuning of different parameters that affect the performance of these algorithms. Experimentation indicates that selecting the appropriate parameters that control strategies such as Tabu Search,

Circuit	**Blks**	**TS-DA**		**TS-TC1**		**TS-ASP2**		**TS-DS-IS**	
		Cuts	Time	Cuts	Time	Cuts	Time	**Cuts**	**Time**
Chip1	2	20	3.2	20	4.1	28	3.2	**20**	14.0
	4	56	4.0	53	4.3	55	3.1	**47**	23.5
	6	58	4.6	67	4.3	56	7.9	**58**	37.0
Prim1	2	56	6.4	60	6.6	54	11.6	**54**	47.4
	4	102	29.4	114	20.4	129	14.0	**102**	100.1
	6	139	20.7	136	44.0	146	12.3	**137**	1:15
Ind2	2	392	599	388	14:09	381	17:06	**323**	37:30
	4	1189	9:36	1195	20:09	1207	10:46	**991**	65:02
	6	1375	17:40	1399	14:09	1444	15:30	**1375**	89:33

Table 3.5: A comparison between different Tabu Search settings

Genetic Algorithms and Simulated Annealing has a drastic effect on the final solution acquired. Many attributes of the solution space can affect the ideal Tabu list size, mutation and crossover rates, and the annealing schedule in the above mentioned methods. It is important to identify when and how parameters are stable, and devise methods to adjust these parameters depending on problem size and application. This section introduces a new technique to tune the parameters that control the performance of the Tabu Search algorithm in a more systematic fashion.

The most important parameters that control the performance of Tabu Search are, the aspiration criterion for backtracking, the lengths of the Tabu lists and the attributes to be used for the Tabu restrictions.

Sections 3.6.1.2 and 3.6.2.1 presented some schemes for determining the move attributes and size of the Tabu list respectively. A method proposed here, called *Adaptive Tabu Search* (ATS), is considered more robust in the sense that many parameters adapt according to the properties of the solution space being searched.

3.6.3.1 ATS Implementation

In Section 3.6.2.1, the choice of a preferred value for the Tabu list was based either on empirical testing or on variation to the Tabu list size to eliminate cycling. These schemes (static or dynamic) based on a fixed list size (FIX-TABU) are not strict and, therefore, the possibility of cycling remains. Other Tabu Search implementations are based on the fact that cycles are avoided if the repetition of previously visited configuration is prohibited [Glov90]. For example, in the Reverse Elimination Method [**?**, Glov90], the only movements that are excluded from consideration are those that would lead to previously visited solutions. This method may be realized as a strict tabu implementation. Figure 3.15 shows the adaptive Tabu Search implementation. The main feature of this method is the capability of adapting different parameters according to the nature of the landscape of the solution space being searched. Initially, Tabu Search sets a search period through which it updates statistics regarding the following: (i) Total module moves, average module moves, (ii) Tabu and non-Tabu moves, (iii) Valid and non-valid moves, (iv) Flat and active regions, (v) Rate of change of the objective function, (vi) Inactive modules, (vii) Different number of solutions between search periods. The search controller would decide according to these values on whether to constraint the search or not. If the objective function has not changed for a certain number of phases (case 3) then the controller decides to reset the Tabu list (removes all items) and activate all modules that have low average movement. The controller attempts to do so either because the landscape of the solution space is flat or no more modules can be moved due to invalid moves or Tabu moves. If this is not the case, then the search controller would decide upon the following:

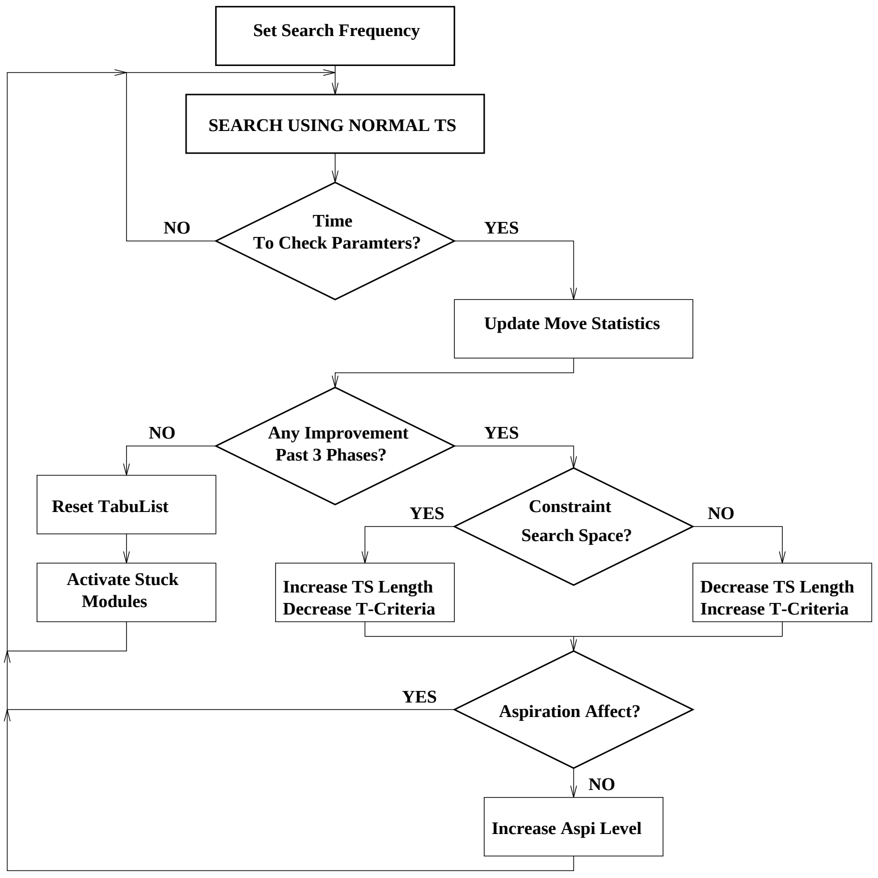


Figure 3.15: Adaptive Tabu Search

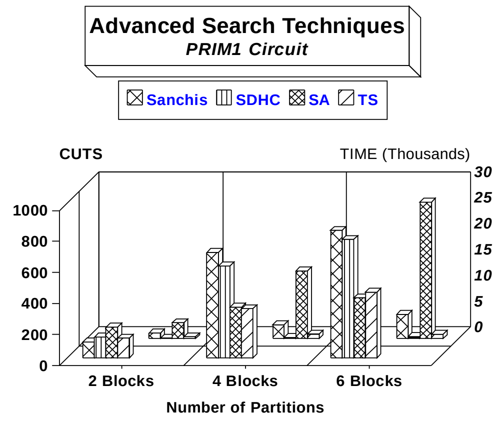
- The search is proceeding well,
- The search is constrained,
- The search is unproductive.

In the first case the search controller would proceed to check the affect of the aspiration criterion used, and would tune it according to the value of the aspiration effect during the previous search period. If the search is constrained or unproductive, the search controller would utilize the length of the Tabu List and the Tabu Criteria used to solve the above mentioned problem.

ADAPTIVE TABU SEARCH HEURISTIC										
	PRIM1 Circuit					**IND1 Circuit**				
Blks	**TS**		**ATS**		**IMP**	**TS**		**ATS**		**IMP**
	Cuts	Time	Cuts	Time		Cuts	Time	Cuts	Time	
2 Blks	59	52.5	55	5.6	7%	59	91	28	202	52%
4 Blks	126	50.4	122	44.4	3%	135	171	124	285	8%
6 Blks	159	91	134	127	16%	230	375	176	606	23%

Table 3.6: The performance of Adaptive Tabu Search

The search controller was implemented and tested on some problems, and was effective in improving the search quality within Tabu Search heuristic. Table 3.6 shows the results obtained with and without the search controller. The results indicate clearly that when the search controller is tuning the parameters, the quality of solutions obtained improve steadily for different block partitions and as the problem size increases.



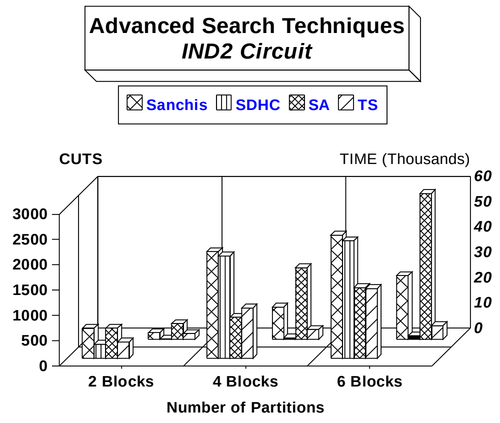


Figure 3.16: Performance of advanced search heuristics

3.6.4 Performance of Advanced Search Techniques

Figure 3.16 and Table 3.7 present a comparison of the solution quality and computation time of partitions obtained for 2,4 and 6 blocks of Simulated Annealing, Sanchis interchange method, Dynamic Hill Climbing and Tabu Search[2] starting from random initial points.

As has been discussed in Section 3.5, the Simulated Annealing algorithm's performance depends on the *annealing schedule* used. The choice of annealing schedule plays an important role in controlling the effectiveness and efficiency of the algorithm. Our previous circuit partitioning results using Simulated Annealing [Arei93, **?**] were based on an annealing schedule similar to that proposed by Kirkpatrick [**?**]. In this section, solutions obtained by Simulated Annealing are based on results that use an annealing schedule proposed by White [Whit84]. Even with

[2]These results are based on pure Tabu Search and not Adaptive Tabu Search.

Circuit	**INTER**		**SDHC**		**SA**		**TS**	
	Cuts	Time	Cuts	Time	Cuts	Time	**Cuts**	**Time**
Chip1	20	18	20	1.3	20	92	**20**	**14.0**
Chip2	15	19	14	2.5	14	98	**14**	**10.3**
Prim1	60	91	75	5.6	65	346	**54**	**47.4**
Prim2	226	433	248	36	224	1314	**181**	**284**
Bio	102	1058	135	61	199	3060	**127**	**321**
ind1	42	211	76	14	46	959	**45**	**147**
ind2	593	2661	278	153	594	6240	**323**	**2250**
ind3	514	2294	440	659	655	9420	**305**	**2822**

Table 3.7: 2-Way partitioning of advanced search heuristics

this cooling schedule execution times are still high as seen in Table 3.7.

The quality of solutions obtained by the Tabu Search method is consistently better than the those obtained by the iterative improvement method (using 50 random starting points), and in many cases better than Simulated Annealing. Moreover, the Tabu Search execution time on average, is faster then the Simulated Annealing approach by a factor of **five**. The quality of partitions obtained by the Tabu Search method are better than those obtained by the iterative method by 40% and yields partitions that are 5% better than those obtained by Simulated Annealing.

3.7 Summary

This chapter considered emerging heuristic search approaches for solving the circuit partitioning problem in circuit layout. The main advantages and disadvantages of each heuristic were discussed, and the best parameter setting for the circuit partitioning problem were presented.

The SDHC heuristic is very simple to implement and is capable of exploring regions of the parameter space in short periods of time. Genetic Algorithms offer a number of advantages : they are robust, they are good at "exploration," and they search from a set of designs and not from a single design. GRASP is effective in producing good starting solutions for local search via an intelligent construction phase that is greedy and adaptive. Simulated Annealing produces near optimal solutions and is easily adapted to any combinatorial optimization problem. Tabu Search is capable of exploiting and exploring the solution space effectively.

We have also demonstrated the importance of intelligently controlling the performance of a search based heuristic. As made explicit in Tabu Search, one may choose any algorithmic framework and superimpose a search controller within this technique as an intelligent adapter.

Still theoretical limits do exist. All the approaches mentioned above have fundamental strengths but tend to suffer as the problem to be solved increases in size. Genetic Algorithms are not well suited to perform finely tuned search. Tabu Search is not certain to find a good solution for every problem without parameter tuning. The practical question is in the realm of heuristic search "*Can these new approaches be more effective at finding good solutions to hard problems?*" The strengths and limitations of Genetic Algorithms, GRASP, Simulated Annealing, and Tabu Search remain to be charted fully. While at present there appear to be many constituents of effective heuristic procedures, the time is perhaps approaching to bind these heuristics together. The next chapter, introduces the means of combining these advanced search techniques and shows the importance of integrating them as a means to offset their weakness and highlight their strengths.

Chapter 4

Hybrid Search Techniques

In general, most real world problems are too complex for any single processing technique to solve in isolation. The modern philosophy for constructing fast globally convergent algorithms is to combine a simple globally convergent algorithm, such as Simulated Annealing or a branch and bound mixed integer programming solver, with a fast locally convergent heuristic to form a more suitable and faster hybrid.

In this chapter, the means of integrating some of the advanced search heuristics that have been discussed in Chapter 3 are described. First, a classification of hybrid techniques will be presented. The advantages and disadvantages of most heuristics will be presented next. Finally, the hybridization strategies used to integrate the most promising heuristics will be discussed.

4.1 Hybridization

The previous chapter introduced the main concepts and advantages of using some advanced search heuristics. Even with these search techniques, theoretical limits exist. A worthwhile avenue for research relative to combinatorial algorithm development involves integrating these fundamentally different approaches as a means to avoid their inherent weakness.

There is currently considerable confusion as to exactly what a hybrid system is. Much of the problem lies in the different interpretation of functionality and

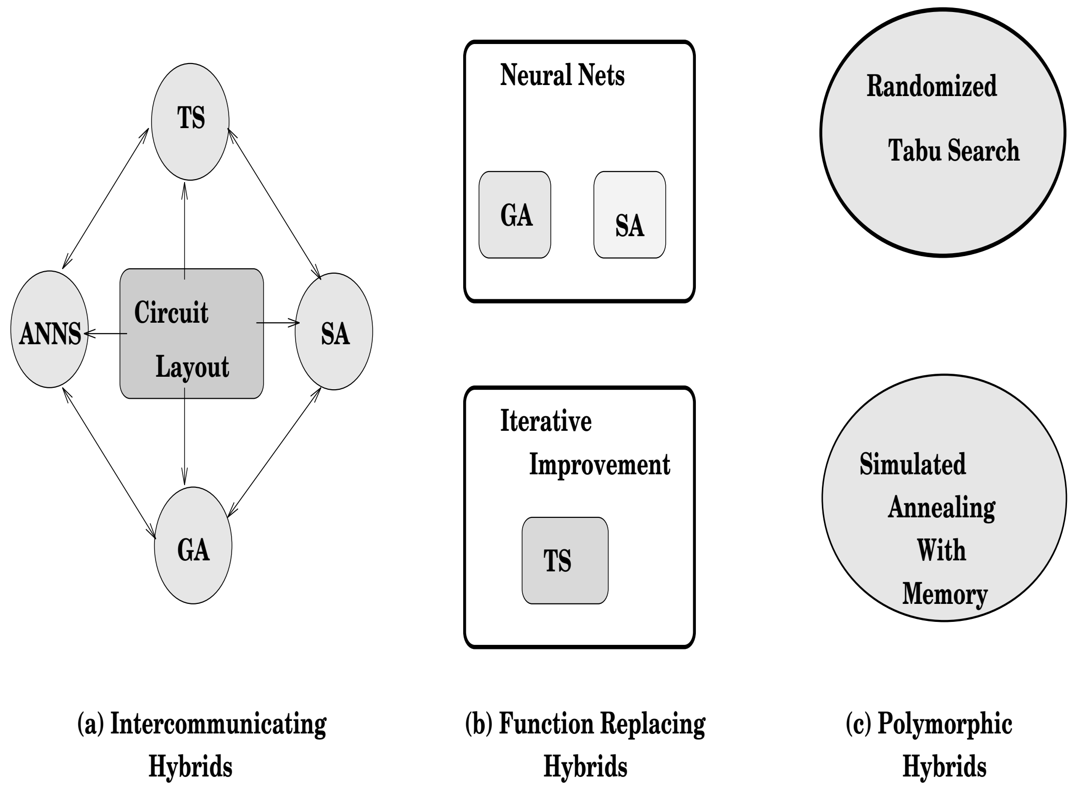


Figure 4.1: Hybrid classes applied to combinatorial optimization

architecture of these systems. Hybridization, in general, couples distinct classes of techniques tightly and switches between them within a design run. As shown in Figure 4.1, hybrid implementations can be classified as following [Goon92]:

- **Intercommunicating hybrids** are independent, self contained, processing modules that exchange information, and perform separate functions to generate solutions. Communication and synchronization is usually performed by the aid of a *controller*. When a problem can be subdivided into distinct processing tasks, then these different independent modules can be used to solve the parts of the problem that they are best at.

- **Function Replacing hybrids** address the functional composition of a single optimization technique. In this class, a principal function of the given technique is replaced by another optimization processing technique. The motivation of replacing these principal functions is to increase execution speed and enhance reliability.

- **Polymorphic hybrids** are systems that use a single processing technique to achieve the functionality of different processing techniques.

4.1.1 Hybridization Strategy

Table 4.1 shows a comparison between different search methods used previously for circuit partitioning. The comparison is in terms of the capabilities of performing finely tuned search "*Fine-S*", diversification of the search "*Div-S*", regional intensification of the search "*Inten-S*", local and global convergence[1] "*L-G-conv*" and finally advantages and disadvantages of each search methods. It is evident that stochastic, adaptive and local search approaches have strengths and weaknesses and that they should be viewed not as competing models but as complementary ones. By integrating these fundamentally different approaches we can avoid many of the weaknesses inherent in each methodology, while capitalizing on their individual strengths.

[1]Sensitivity to initial starting points.

Features of Search Methods					
Method	**Features**				
	Fine-S	Div-S	Inten-S	L-G_conv	Comments
Numerical	no	no	no	local	Provides good starting points.
Interchange	yes	no	no	local	Gets stuck in local minima.
SDHC	yes	no	yes	local	Produce suboptimal solutions.
GRASP	yes	yes	no	local	Constructs many initial points.
GA	no	yes	yes	global	Explore search space.
SA	yes	yes	yes	global	Huge CPU times.
TS	yes	yes	yes	local	Extensive Record Keeping.

Table 4.1: Comparison between different search methods

4.2 Effective Hybrid Schemes

Usually, interchange methods fail to converge to optimal solutions unless they initially begin from good starting points. The choice of starting point is a crucial factor in the performance of the Iterative Improvement heuristics. The performance of some heuristic methods such as Tabu Search and Iterative Improvement based on node interchange can be improved if initial good starting points can be produced by some heuristic. In this section, the solution quality of different local search heuristics that start from good initial solutions are compared. The initial solutions are obtained using GRASP, Barnes eigenvector approach and a Genetic Algorithm.

4.2.1 A Combined GRASP-Genetic Search Heuristic

Chapter 3 introduced the main concept of GRASP as a method for intelligently constructing initial solutions via an adaptive randomized greedy function. In this section, the importance of integrating GRASP with Genetic Algorithms is described as a means to improve the solution quality. The developed heuristic not only produces excellent solutions, but the computation time is reduced drastically. The Genetic Algorithm here is used to obtain many initial starting points instead of a single solution provided by the eigenvector approach. This allows a heuristic such as Tabu Search to concentrate the search effort in the most promising regions. Another advantage of using the Genetic Algorithm approach is that it allows us to further explore and diversify the search through information received by the local search heuristic.

Applications and theory show that GA can be useful in solving difficult problems. However, a pure genetic approach still has its shortcomings for combinatorial optimization problems [Glov90]. The hybrid scheme here, works with subpopulations P_i of solutions. The first half of the population is produced randomly, whereas the second half of the population is produced through a more intelligent scheme using the GRASP. Thus, this method can be seen as a meta-heuristic that improves on existing solution procedures. The method is flexible and can be implemented with many different parameter values.

4.2.1.1 Parameter Setting

In this implementation, the *group-number encoding* representation scheme is used instead of the more popular *permutation with separators* encoding. The reason behind this is due to the high computation time involved to encode and decode the second representation scheme. For the purpose of circuit partitioning, calculation of the objective function "nets cut at every iteration" is expensive. Therefore, it is modified slightly so that it would only update the value of the chromosome instead of completely evaluating all solutions [**?**].

Good on-line performance is associated with high crossover rate $(90 - 98\%)$ combined with low mutation rate $(0.01 - 0.06\%)$. Our empirical study, strongly supports using a multi-point crossover operator over the one-point crossover technique discussed previously (for group number encoding scheme). A 3-point and 4-point crossover works best for circuit partitioning problems.

In this implementation population sizes between 25-50 were used. These population sizes were sufficient in obtaining good initial starting points for a local search heuristic to obtain good solutions in reasonable amounts of time.

It has been noticed that results vary from one setting to another, but generally speaking using the Genetic Algorithm with a deterministic selection method such as *rank select* and a deletion method such as *replace most inferior* gives the desired results for at least obtaining good initial starting points.

4.2.1.2 GRASP-GA Implementation

As in the generic Genetic Algorithm introduced in Chapter 3, the population evolves into a different population for several generations. At the end, the heuristic returns the best member of the population as the solution to the problem. For each generation, the evolution process proceeds as follows. Two members of the population are chosen according to the selection strategy employed (see Section 3.4.1.6). These two members are then combined through the crossover operator to produce an offspring. It is to be noted that the mutation operator is missing. A local optimization technique (described in the next section) is then applied to the offspring to improve it. The offspring is tested to see if it is suitable for the population. The newest generation is then injected by a combination of chromosomes from the old generation and new offsprings that have been produced. The basic structure of the main hybrid heuristic based on GRASP and GA is given in Figure 4.2.

```
        A GRASP-GENETIC ALGORITHM HYBRID
1. Encode Solution Space
2.(a) set pop_size, max_gen, gen=0;
  (b) set cross_rate, mutate_rate;
3a. Initialize 50% of $P$ Randomly.
3a. **Initialize 50% of $P$ using GRASP.**
4. **While** max_gen $\geq$ gen
       Evaluate Fitness
       **For** (i=1 to pop_size)
         Select (mate1,mate2)
         if (rnd(0,1) $\leq$ cross_rate)
           child = Crossover(mate1,mate2);
         Repair child if necessary
           **Local-Optimize(Population)**
       **End For**
       Inject New Generation with most fit offsprings.
       gen = gen + 1
   **End While**
5. Return best chromosomes.
```

Figure 4.2: A GRASP-GA hybrid

4.2.1.3 Local Search

Genetic Algorithms are not well suited for fine-tuning structures which are close to optimal solutions [**?**]. Incorporation of local improvement operators into the recombination step of a Genetic Algorithm is essential if a competitive Genetic Algorithm is desired.

After crossover, GRASP-GA applies a local optimization process on the off-springs. We use a simple variation of the Sanchis heuristic [Sanc89]. The original Sanchis heuristic has several passes after which the heuristic terminates as presented

GRASP-GENETIC SEARCH HYBRID						
Circuit	**Blocks**	**Random-GA**		**GRASP-GA**		**%IMP**
		Cuts	Time	Cuts	Time	Cuts
Prim1	2 Blks	76	113	70	140	8%
	4 Blks	158	163	130	208	17%
	6 Blks	171	216	158	276	8%
Ind2	2 Blks	461	1055	435	866	6%
	4 Blks	1949	1656	1129	1378	42%
	6 Blks	2895	2179	1559	1818	46%

Table 4.2: Results of GRASP-GA hybrid implementation

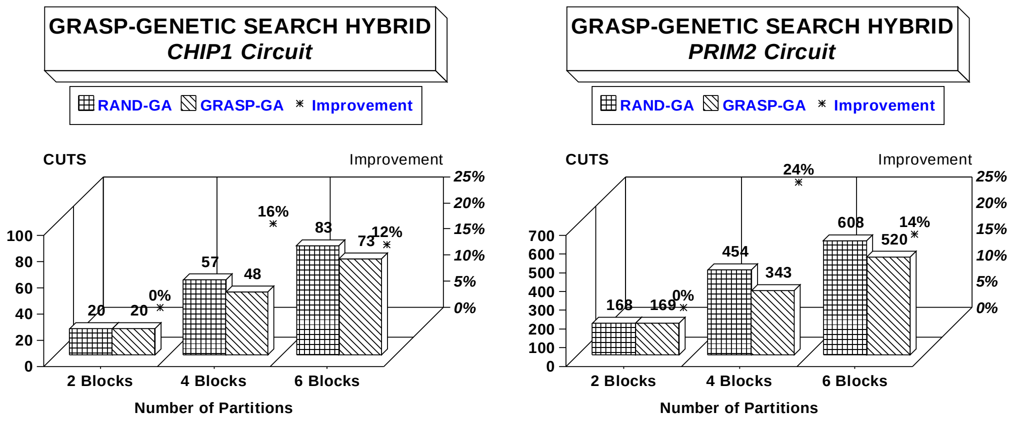


Figure 4.3: The performance of the GRASP-GA hybrid

in Chapter 2. In the local optimization phase, a single pass is allowed, furthermore a restriction on the number of modules to be moved is set to a certain value MAX-MOD. It is to be noted that if local optimization is not strong enough to overcome the inherent disruption of the crossover, more strong local optimization is needed.

4.2.2 A Combined GRASP-Tabu Search Technique

Tabu Search as described in Chapter 3, is capable of guiding local search heuristics that get stuck at a local minima to continue exploration without becoming confounded by an absence of improving moves, and without falling back into a local optimum from which it previously emerged. Yet, the Tabu Search heuristic may take a considerable amount of time (compared to a simple Iterative Improvement heuristic) if it starts from a point that is far away from optimality. The combination of Tabu Search and GRASP leads to a powerful hybrid heuristic. Good initial partitions obtained by GRASP allow Tabu Search to refine that initial partition quality in a reasonable amount of time, thus reducing the computational time and enhancing the solution quality. Figure 4.4 shows the main hybrid heuristic based on GRASP and Tabu Search.

4.2.3 Computational Results

Table C.17 and Figure 4.5 present a comparison of the solution quality and computation time of partitions obtained for 2,4 and 6 blocks of three netlists with different sizes. Some general observations can be made from the results presented in Table C.17. Comparing results based on good initial solutions to those starting from random initial starting points, the following conclusion can be made:

```
        A GRASP-TABU SEARCH HYBRID
(A) GRASP: INITIAL SOLUTION
    Read_Circuit_NetList();
    do
        Construction_Phase(Greedy,Random,Adaptive)
        Store_Best_Solution(Previous Solutions)
    While (not Done)
    Report_Best_Solution()

(B) Tabu Search: Local_Improvement_Phase()
    Read Best Solution
    While  (local optimum not reached)
        NewSolution = Tabu_Search(Solution)
    EndWhile
    Return_Best_Solution()
```

Figure 4.4: A GRASP-TS hybrid heuristic

- Initial solutions obtained using Barnes eigenvector approach, GRASP, and Genetic Algorithms are better than those obtained randomly for most circuits.

- The eigenvector-node interchange approach yields netlist partitions with comparable or fewer cut nets than the best netlist partitions obtained using node interchange heuristics alone on many random initial netlist partitions, in a small CPU time.

- The combined Tabu Search with GRASP gives rise to a heuristic that is capable of finding excellent solutions compared to those obtained randomly, or by a more systematic approach like Barnes eigenvector approach.

- Finally, using the Genetic Algorithm to obtain initial starting points for the finely tuned search performed by the Tabu Search heuristics are comparable to those obtained by GRASP. Also, the computation time of this hybrid is less than other techniques used thus far.

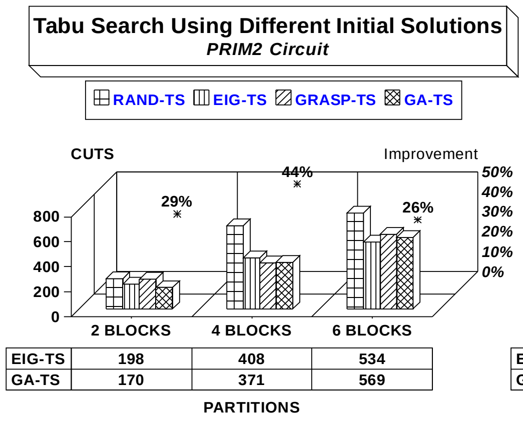
Circuit	**Blks**	**RAND-TS**		**EIG-TS**		**GRASP-TS**		**GA-TS**	
		Cuts	Time	Cuts	Time	Cuts	Time	Cuts	Time
Chip1	2	20	13.4	20	12.6	20	10.2	20	9.4
	4	49	12.3	46	11.8	52	11.4	54	9.0
	6	64	12.4	59	23.8	55	20.5	56	17.1
Prim1	2	72	15.0	61	31.2	60	17.9	58	19.3
	4	126	39.8	119	46.6	102	56.8	115	40.6
	6	168	38.8	140	49.8	140	31.5	136	55.3
Ind2	2	675	1710	520	1809	388	1381	286	821
	4	2259	1752	1804	1318	1189	1602	1403	1272
	6	2682	1733	1803	2171	1371	3038	1537	1132

Table 4.3: Effective hybrid search techniques

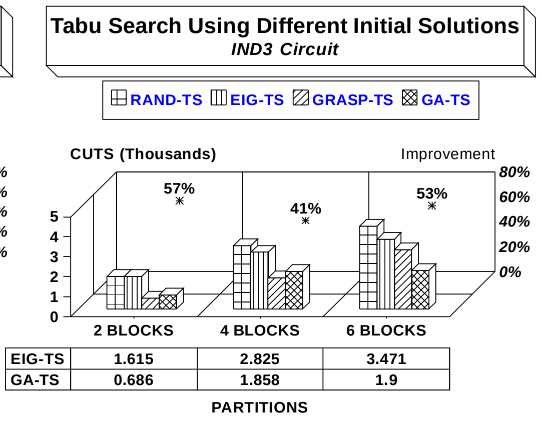
We should also mention that the Genetic Algorithm produces many solutions that vary in quality compared to those obtained by GRASP and the eigenvector approach. Consequently, in Section 4.4, we show the merit of combining Tabu Search with Genetic Algorithms in exploring the solution space effectively.

4.3 A Memory Based Annealing Heuristic

The discussion of Tabu Search in Chapter 3 suggests that it may be considered as a kind of deterministic version of Simulated Annealing in a broad sense. Simulated Annealing achieves diversity in search by randomization without reliance on memory. By this view, the use of randomization, via assigned probabilities, allows a gain in efficiency by obviating extensive record keeping and evaluation operations that a more systematic pursuit of diversity may require. At the same time,



	2 BLOCKS	4 BLOCKS	6 BLOCKS
EIG-TS	198	408	534
GA-TS	170	371	569



	2 BLOCKS	4 BLOCKS	6 BLOCKS
EIG-TS	1.615	2.825	3.471
GA-TS	0.686	1.858	1.9

Figure 4.5: Initial solutions and local search

it entails a loss in efficiency by allowing duplications and potentially unproductive wandering that a more systematic approach would seek to eliminate. One possible effort to improve Simulated Annealing is to replace its rules with those more closely resembling the prescriptions of Tabu Search. We show that the combination of Tabu Search and Simulated Annealing gives rise to probabilistic (hybrid) heuristics that form the basis for approaching combinatorial optimization problems in an intelligent manner.

4.3.1 Simulated Annealing with Memory

The hybridization method in this section will be based on the polymorphic hybrid model as seen in Figure 4.1. In this model of computation, Simulated Annealing emulates Tabu Search by eliminating the unproductive wandering and the duplication of solutions. The Simulated Annealing algorithm that is guided by Tabu Search (shown in Figure 4.6) proceeds by attempting a certain number of neighborhood moves at each temperature, while the temperature parameter is gradually dropped.

The single loop of a descent algorithm is replaced by a double loop; in the outer loop the temperature is changed and the inner loop determines how many neighborhood moves are to be attempted at each temperature. The annealing schedule in this hybrid version of Simulated Annealing and Tabu Search is similar to that proposed by White [Whit84].

```
current_solution ← initial_solution
current_cost ← evaluate(current_solution)
T ← $T_{initial}$
While ($T \geq T_{final}$)
     for i = 1 to iteration(T) /* neighborhood moves */
          new_solution ← move(current_solution)
          if new_solution NOT TABU then
             Update Tabulist;
             Update Aspiration Level;
             new_cost ← evaluate(new_solution)
             $\Delta cost$ ← new_cost - current_cost
             $if(\Delta cost \leq 0 \ \ OR \ \ e^{-\Delta cost/T} > random())$
                 /* accept new solution */
                 current_solution ← new_solution
                 current_cost ← new_cost
             EndIf
          EndIf
          Else If (Move in Tabulist) then
             Check Aspiration Value
             Override Tabulist if Aspiration is met
          End Else
     EndFor
     T ← next_temp(T)
EndWhile
```

Figure 4.6: Simulated Annealing with memory (SAM)

The short term memory functions in Tabu Search, which are fulfilled by the Tabu List and Aspiration Criteria, are used to eliminate visiting neighborhood

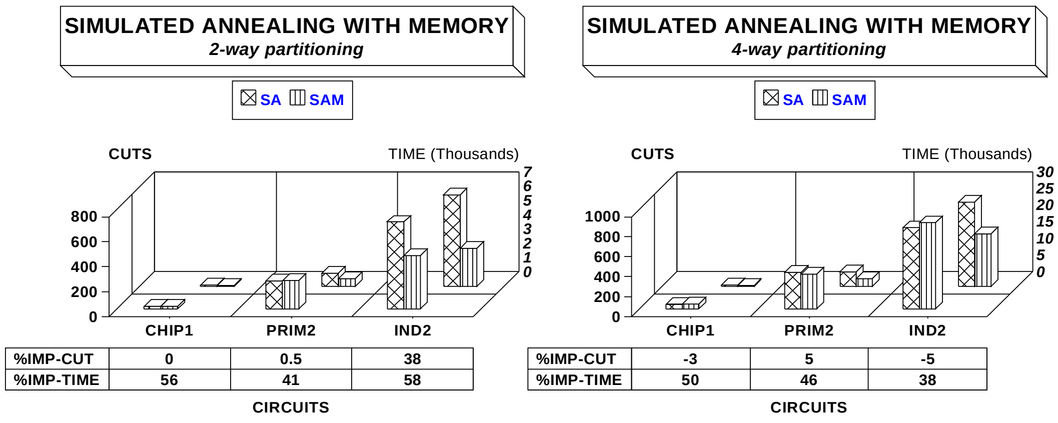


Figure 4.7: The performance of SAM

solutions visited recently, thus preventing duplications of solutions. Aspiration when used effectively, forces Simulated Annealing to backtrack to previous solutions as a means to refine the search in those regions, thus improving the quality of solutions obtained. If a move is randomly chosen and its status is Tabu, another move will be chosen instead, unless the Aspiration Criteria determines otherwise.

SA vs SAM (SA with Memory)												
Blk	**Prim1 Circuit**						**Ind1 Circuit**					
	SA		**SAM**		**IMP**		**SA**		**SAM**		**IMP**	
	C	T	C	T	C	T	C	T	C	T	C	T
2	65	360	56	156	13%	56%	89	1004	37	468	58%	53%
4	106	1311	105	619	1%	52%	141	3922	138	1875	2%	52%
6	124	2440	123	1190	1%	51%	182	7629	183	3740	-1%	51%

Table 4.4: The performance of Simulated Annealing with memory

Table 4.4 and Figure 4.7 present results for the new developed Simulated Annealing with memory hybrid (SAM), and compare its performance with that of pure

Annealing heuristic. The results clearly indicate the computation time of SAM is faster than that of SA by at least 50%, and the quality of solutions obtained by SAM are comparable with those obtained by SA. The reason as indicated before is due to eliminating the duplicated solutions and the concentration on regions that are worth backtracking to using the aspiration criteria of Tabu Search.

4.4 A Tabu-Genetic Algorithm Approach

The hybridization method in this section will be based on the intercommunicating hybrid model as seen in Figure 4.8. In this model of computation, independent processing modules exchange information and perform separate functions to generate solutions. A common means of attempting to improve the performance of heuristic methods is to restart the solution process from different solutions. These solutions are either generated randomly or by a set of favored starting heuristics. The generation of new starting solutions is a key area to be explored by a more systematic diversification strategy. The hybrid model is based on using the Genetic Algorithm as a preprocessor to perform the initial search, before turning the search process over to a system (i.e. Tabu Search) that can employ domain knowledge to guide the local search. The Genetic Algorithm here is used (instead of the numerical eigenvector approach) to obtain many initial starting points. The Tabu Search heuristic is then used to concentrate the search effort in the most promising regions. Another reason for using the Genetic Algorithm approach, is that it is capable of exploring the solution space, thus it is used for further intensification and diversification of search. Figure 4.8a shows the 3 principal stages used in the search strategy. The main task, is divided into three different phases, initial starting points, local search and exploration of the solution space. The initial solutions are obtained either randomly or in a more systematic way, using a numerical eigenvector based approach, Genetic Algorithms or GRASP.

4.4.1 Search Controller

Figure 4.9 shows the means of combining the Genetic Algorithm and Tabu Search into a hybrid technique. This hybrid system thus attempts to combine the power of decentralized characteristics of Genetic Algorithms with the more localized features of Tabu Search and Iterative Improvement heuristic based on node interchange.

The main task of the search controller is to synchronize the search among the three different phases. At each phase, the search controller tunes and sets certain parameters of the Tabu Search heuristic and those of the Genetic Algorithm. Parameter tuning is necessary to achieve good performance in every phase of the search strategy. Once that is achieved it may be useful to invoke either Tabu Search or an interchange based method to perform finely-tuned local search. The search controller sets the population and generation size of the Genetic Algorithm to values of 25 and 10 respectively. A low mutation rate of 0.04 and high crossover rate of 0.96 are also set to achieve good initial starting points. Repair solutions for infeasible chromosomes are used here instead of penalty function approaches. As mentioned in Chapter 3, the Tabu lists in this setting are not activated "no moves are considered to be Tabu" until the heuristic hits the first local minima. To achieve that without cycling to previous solutions, the interchange method using a certain number of passes is performed. Once a local minima is reached, Tabu lists are activated and the Tabu Search heuristic resumes exploration for better solutions

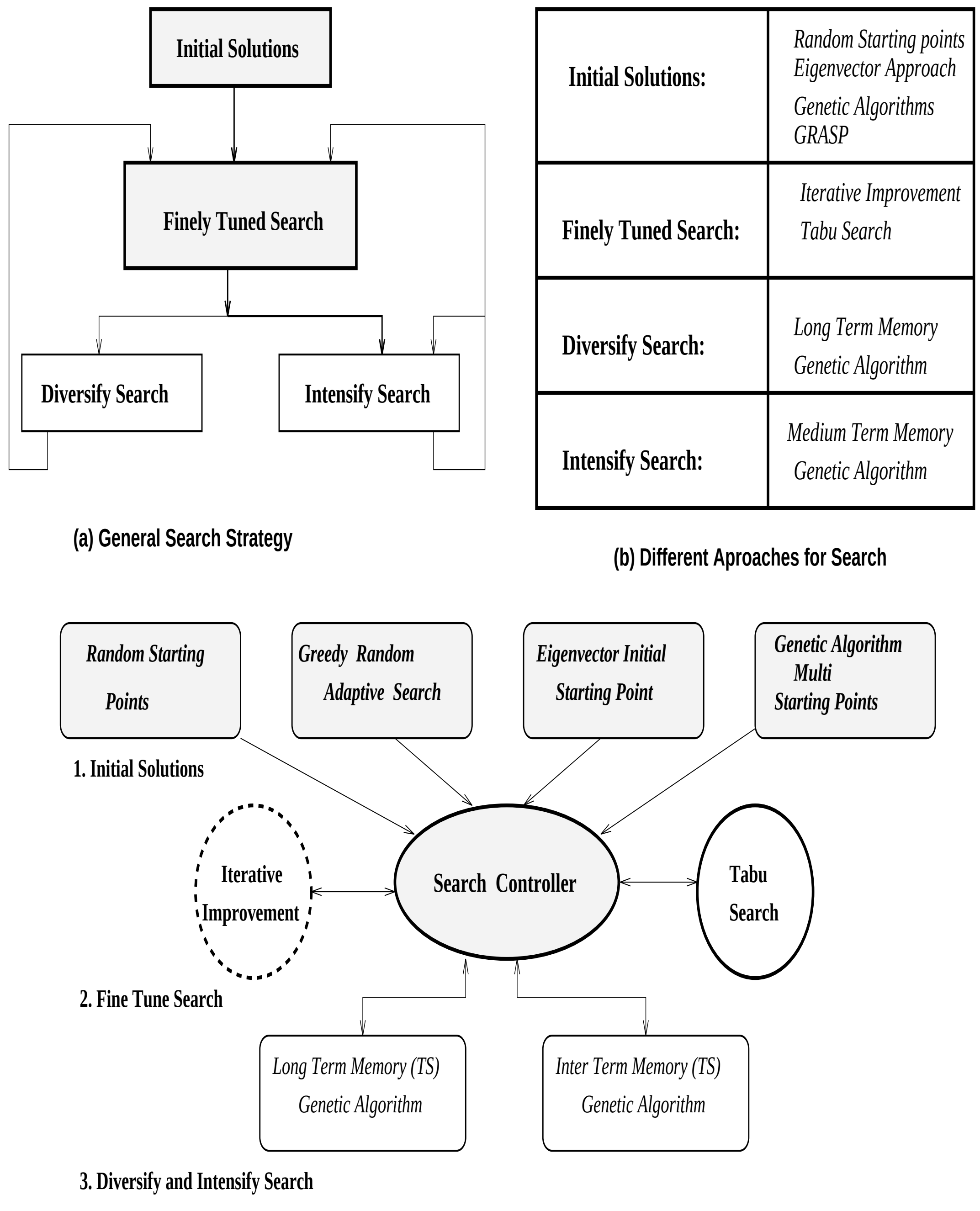


Figure 4.8: Interaction between different search strategies

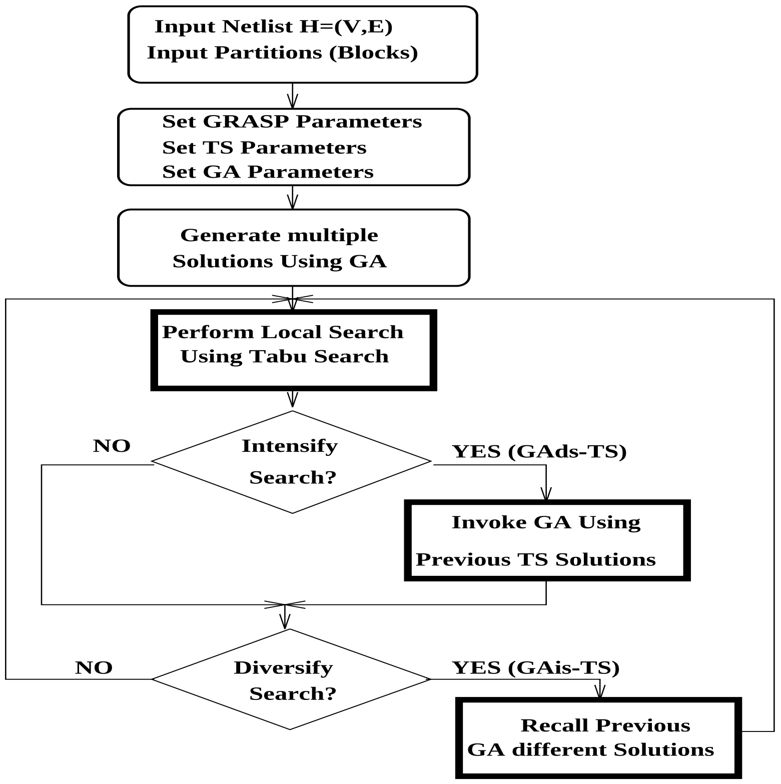

Input Netlist H=(V,E)
Input Partitions (Blocks)
Set GRASP Parameters
Set TS Parameters
Set GA Parameters
Generate multiple Solutions Using GA
Perform Local Search Using Tabu Search
NO
Intensify Search?
YES (GAds-TS)
Invoke GA Using Previous TS Solutions
NO
Diversify Search?
YES (GAis-TS)
Recall Previous GA different Solutions

Figure 4.9: A Tabu Search-Genetic Algorithm hybrid

through the short term memory.

4.4.1.1 Search Diversification

The search controller utilizes the Genetic Algorithm to achieve search diversification (instead of the long term memory usually used with Tabu Search) in two different ways. The first method is based on obtaining multiple starting solution points using the Genetic Algorithm. The Tabu Search Algorithm is then invoked to zoom on the different search points and explore the different regions of the solution space. The search time on each different initial solution is determined by the search controller and is basically terminated when no improvements in solution quality are obtained. In the second approach, the search controller processes information obtained from

Tabu Search during the exploration period. The search controller samples different solutions points and chooses those points that are distinct from each other. These points are then injected as initial populations to the Genetic Algorithm, and new different solutions are formed using crossover and mutation. Since this phase is critical, the search controller attempts to speed the execution time of the Genetic Algorithm. This is achieved by using lower population and generation sizes, in the same time it attempts to repair solutions efficiently instead of penalizing infeasible solutions.

4.4.1.2 Search Intensification

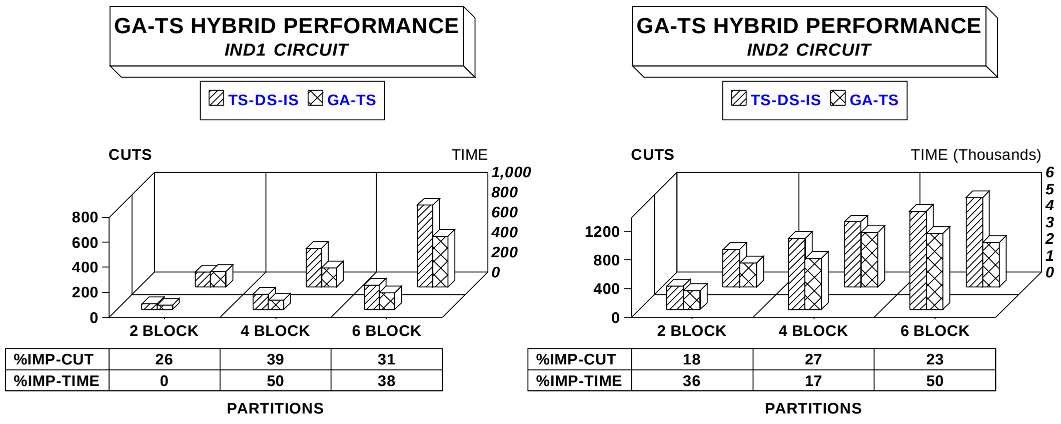
Intensification of the search is achieved in a similar manner to that of the diversification phase, with slight modifications. In this phase, the search controller obtains sample points from the Tabu Search heuristic at the second half of the exploration period. These good quality solutions are compared and only those that have similar qualities are injected in some of the populations of the Genetic Algorithm. Solutions resembling similar structures are created in the same way by crossing notions between different chromosomes. The search controller, attempts to lower the mutation rate to a low value. The replacement of chromosomes in each generation is carried out using GA-rmi method discussed in Chapter 3. The selection method determined by the controller is based on the stochastic roulette wheel method. In summary, the search controller uses a mechanism opposite to that used for diversification to achieve intensification of the search.

4.4.1.3 Related Issues

The parameters that control the performance of the Tabu Search heuristic are controlled in a similar fashion to those of the Genetic Algorithm. The most important are: (i) the aspiration criteria for backtracking, (ii) the lengths of the Tabu lists and (iii) the attributes to be used for the Tabu restrictions. The search controller's performance can be further improved if it can extract some statistical information on the circuits to be partitioned. For instance, the statistics obtained regarding the maximum number of cells on any net could be utilized to have some modules fixed in certain blocks, this in place would reduce the computation time of the heuristics involved in partitioning the circuit.

4.4.2 Computational Results

Figure 4.10 compares the performance of Tabu Search based on long and intermediate term memory with the Genetic Algorithm Tabu Search hybrid. Tables 4.5, 4.6



	2 BLOCK	4 BLOCK	6 BLOCK
%IMP-CUT	26	39	31
%IMP-TIME	0	50	38

	2 BLOCK	4 BLOCK	6 BLOCK
%IMP-CUT	18	27	23
%IMP-TIME	36	17	50

Figure 4.10: The performance of GA-TS hybrid

and 4.7 present a comparison of the solution quality and computation time of partitions obtained for 2,4 and 6 blocks using GRASP, Tabu Search, and Genetic Algorithm as shown in Figure 4.9.

Circuit	**TS**		**GAds-TS**		**GAis-TS**		**GAds-is-TS**	
	Cuts	Time	Cuts	Time	Cuts	Time	**Cuts**	Time
Chip1	20	1.8	20	19.2	20	11.1	**20**	22.9
Chip2	14	2.9	14	15	15	7.2	**14**	16.0
Prim1	72	9.8	58	60	64	32	**57**	73
Prim2	252	43.2	180	230	170	114	**161**	632
Bio	144	89.7	116	462	89	223	**89**	582
Ind1	59	45.2	38	169	40	130	**34**	217
Ind2	805	607	273	3363	306	1505	**273**	3422
Ind3	462	1143	454	3005	664	1565	**300**	2750

Table 4.5: 2-Way partitioning results of hybrid heuristics

Circuit	**TS**		**GAds-TS**		**GAis-TS**		**GAds-is-TS**	
	Cuts	Time	Cuts	Time	Cuts	Time	**Cuts**	Time
Chip1	55	3.1	49	22.5	50	13.2	**47**	36.1
Chip2	35	4.1	28	19.1	26	13.1	**28**	28.2
Prim1	139	16.4	114	126	119	104	**100**	156
Prim2	663	61.6	409	653	410	174	**356**	495
Bio	509	190	411	572	430	414	**342**	884
Ind1	135	117	112	223	109	160	**103**	368
Ind2	2323	1227	1038	3182	1529	2882	**943**	5087
Ind3	2140	4074	1788	6660	1819	4021	**1405**	10279

Table 4.6: 4-Way partitioning results of hybrid heuristics

The first column in each table refers to a Tabu Search implementation based on a short term memory. The next three columns in the tables refer to integrating these heuristic in the following manner:

Circuit	**TS**		**GAds-TS**		**GAis-TS**		**GAds-is-TS**	
	Cuts	Time	Cuts	Time	Cuts	Time	**Cuts**	Time
Chip1	64	7.7	59	28	62	13.1	**56**	45.1
Chip2	39	4.1	37	32	36	18.2	**36**	31.6
Prim1	189	13.0	133	151	133	67	**131**	122
Prim2	769	60.5	547	384	505	296	**501**	563
Bio	791	277	500	1141	500	605	**411**	1415
Ind1	237	241	133	417	177	264	**133**	555
Ind2	2662	743	1583	3784	2058	3365	**1322**	6600
Ind3	2120	2126	1900	6960	1930	2354	**1900**	8848

Table 4.7: 6-Way partitioning results of hybrid heuristics

1. A Genetic Algorithm used to diversify the search (*GAds-TS*).

2. Search intensification based on Genetic Algorithm (*GAis-TS*).

3. Search diversification and intensification using the Genetic Algorithm (*GAds-is-TS*).

It is clear from Figure 4.10 and Tables 4.5-4.7 that the GA-TS hybrid (that is based on exploring the solution space by means of diversification and intensification) improves results by at least 40% over a Tabu Search implementation based only on a short term memory, and by 20% over a Tabu Search method based on long and intermediate term memories.

4.5 Summary

This chapter explored the effectiveness of integrating different heuristics to produce robust, efficient and fast hybrids. The integration of Tabu Search and Simulated Annealing gave rise to a hybrid probabilistic technique that avoids the necessity of unproductive wandering due to randomization while producing good effective solutions. Combining GRASP with Tabu Search and Genetic Algorithms not only improved the solution quality but also reduced the computation time effectively. Finally, the combination of Tabu Search with Genetic Algorithm attempted to combine the strengths of the latter in exploring the solutions space effectively with the former in finely tuning the search in the most promising regions.

The developed hybrid search techniques are effective in producing good quality results for the circuit partitioning problem. The main drawback is the excessive computation time required, which increases with the problem size. In the next chapter, a novel clustering technique is presented for reducing the problem size without compromising the solution quality. In fact, using the clustering based techniques, the quality of solutions produced by many local search heuristics are improved, in fraction of the time.

Chapter 5

Circuit Clustering

As the complexity of VLSI circuits increases, a hierarchical design approach becomes essential to shorten the design period [Hage92]. Circuit clustering plays a fundamental role in hierarchical designs. Identifying highly connected components in the netlist can significantly reduce the complexity of the circuit and improve the performance of the design process. Clustering usually serves as a bottom-up preprocessing stage in a hierarchical partitioning and placement environment as seen in Figure 5.1. A good clustering method should identify groups of cells which will eventually end up together in the final partitioning and placement stages. This can be difficult because clustering decisions are made prior to the start of the hierarchical partitioning (placement) process, and hence these decision are made without a global view of the circuit structure. Because of this difficulty, a top-down global netlist partitioning methodology has been preferred to a bottom-up clustering methodology for circuit placement. But the sizes of today's circuits are so large that a top-down partitioning scheme alone is infeasible. Therefore, an effective bottom-up clustering approach is necessary as a preprocessing stage in a hierar-

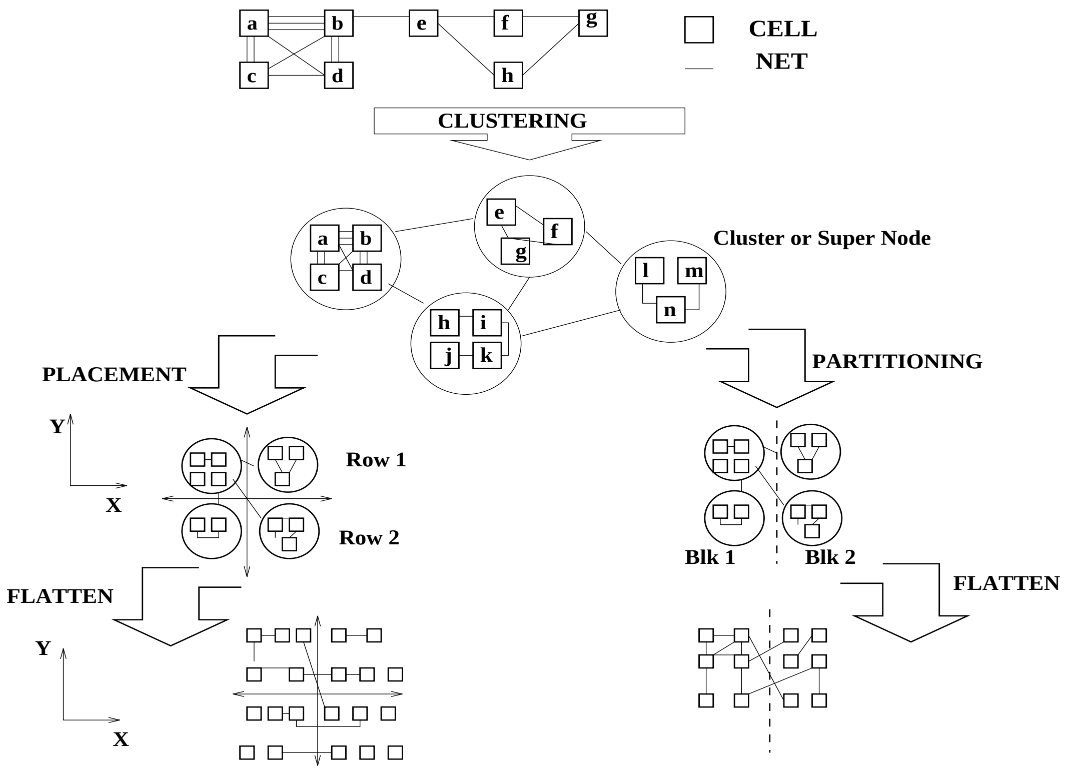


Figure 5.1: Overview of clustering

chical placement approach. In this chapter, new clustering techniques that can be used for circuit partitioning and placement are introduced. Section 5.1 presents the motivation of using clustering in general. Section 5.2 introduces a clustering heuristic based on the GRASP heuristic. The main concepts and terminology used for the second clustering heuristic that is based on statistical information of the circuit are introduced in Section 5.3. The main clustering heuristic is also introduced in Section 5.3. Partitioning based on the clustering heuristic and experimental results are introduced in Section 5.4. Finally, Section 5.5 compares results obtained using a combined hybrid and clustering technique with those based on a branch and bound method to solve circuit partitioning in the form of a mixed integer programming problem.

5.1 Motivation

An intuitive interpretation of a cluster in a circuit is that it is a part of the circuit which is significantly more “complex” or “dense” than the remaining circuit. A cluster is a group of highly connected components in a circuit. The main goal of clustering heuristics is to identify the clusters in a circuit. In VLSI layout design, clustering can be used to construct the natural hierarchy of the circuit. Many existing layout tools generate a circuit hierarchy based on recursive top-down partitioning. Not only does the time and space required by circuit partitioning increase as the circuit size increases, but also the stability and quality of their results deteriorate. A poor result early in the top-down partitioning process imposes an unnatural circuit hierarchy and will likely lead to suboptimal solutions.

Given these difficulties, bottom-up clustering can enable successful top-down partitioning and placement by condensing the circuit netlist and reducing the problem size. Clustering is attractive because it avoids making the far reaching decisions that are inherent in a top-down approach. A bottom-up clustering heuristic can be integrated into the partitioning or placement process (as shown in Figure 5.1) by using clustering as a preprocessing step. First, clustering is performed on the circuit to obtain a *condensed circuit* where each cluster of components is collapsed to form a single component. Partitioning and placement are then performed on the clustered circuit instead of the original circuit. Since the number of components in a clustered circuit is usually much smaller than that of the original circuit, the time and space required by the layout algorithm is reduced significantly. Table 5.1 shows the effect of clustering on the general statistics of three circuits with different sizes.

The average degree of the nodes increases dramatically after clustering the circuit, while the average net size decreases accordingly. The decrease of the net size is due to the fact that many modules that used to be connected to a net are now collapsed to a single cluster (module), thus the net will connect less modules. The local interchange method would yield significantly better results when the circuit has large average node degree [Hage92]. Bui *et al* [Bui87, Bui89] claim that compacting until average degree in netlist ≥ 3 suffices for Interchange to become essentially optimal. This is because there are fewer local minima in the k-interchange neighborhood structure.

Ckt	Type	Nodes	Nets	Pins	Node Degree			Net Size		
					MAX	$\overline{x}$	σ	MAX	$\overline{x}$	σ
chip1	Flat	300	294	845	6	2.8	1.1	14	2.8	1.38
	Clus	37	201	470	20	12.7	4.5	5	2.3	0.61
prim1	Flat	832	901	2906	9	3.5	1.29	18	3.22	2.59
	Clus	104	694	1674	32	16.0	5.08	9	2.4	0.89
ind2	Flat	12142	12949	47193	12	3.89	1.76	584	3.64	11.15
	Clus	1517	11489	29785	65	19.6	8.28	99	2.5	2.6

Table 5.1: The effect of clustering on circuit statistics

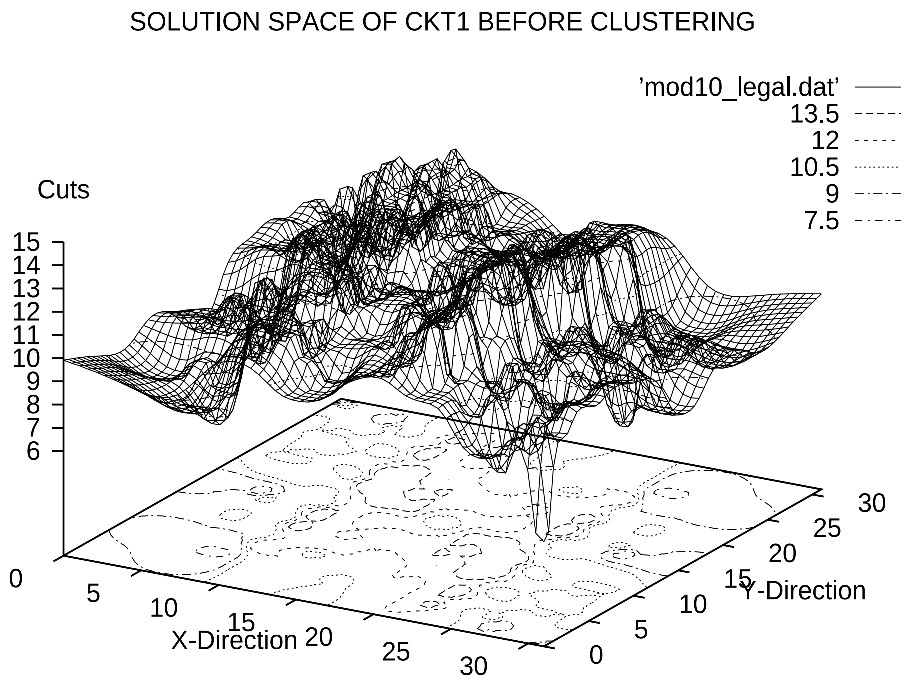
Figure 5.2 shows the effect of clustering in reducing the complexity of a simple circuit with 10 modules and 15 nets. The figure was generated based on a binary representation. The X-axis represents the modules belonging to one partition, whereas the Y-axis represents the modules in the second partition. The Z-axis represents the number of cuts when certain modules are swapped from one partition to the other. It is clear from Figure 5.2 that the number of local minima is reduced after clustering the original circuit. This process enables the partitioning or placement local search heuristics to converge to a neighborhood solution in a short period of time.

5.1.1 Previous Algorithms

Several clustering techniques have been reported in the literature. These approaches are based on circuit partitioning [Chen92], random walk [Hage92],[Cong91] and graph connectivity [Chen92]. In the ratio cut method [Chen92], the number of clusters generated is not known until the heuristic terminates. Moreover, there are generally few resulting clusters, and these clusters vary widely in size. The time complexity of the ratio cut method is more than quadratic which makes it impractical for today's large circuits. In the random walk method [Hage92], the length of the random sequence used to find cycles is $O(n^2)$, where n is the number of cells in the input circuit. The total time complexity is $O(n^3)$. This long run time makes it inappropriate to serve as a bottom-up preprocessing stage in a hierarchical environment. In the (k, l) connectivity method [Garb90], the use of $l \geq 1$ often leads to unnatural results: cells which are farther away from each other (in terms of the number of intervening nets) are more likely to be placed in the same cluster than cells which are closer together. Also, the selection of k and l is not an easy task; they are usually selected based on experiments.

5.2 A GRASP Clustering Heuristic

In this section, a clustering based heuristic that utilizes GRASP to generate clusters of moderate sizes is introduced. For clustering to be used as a practical bottom-up approach, there are two important concerns: (i) the computation time used to generate the clusters must be negligible in comparison to the time needed to



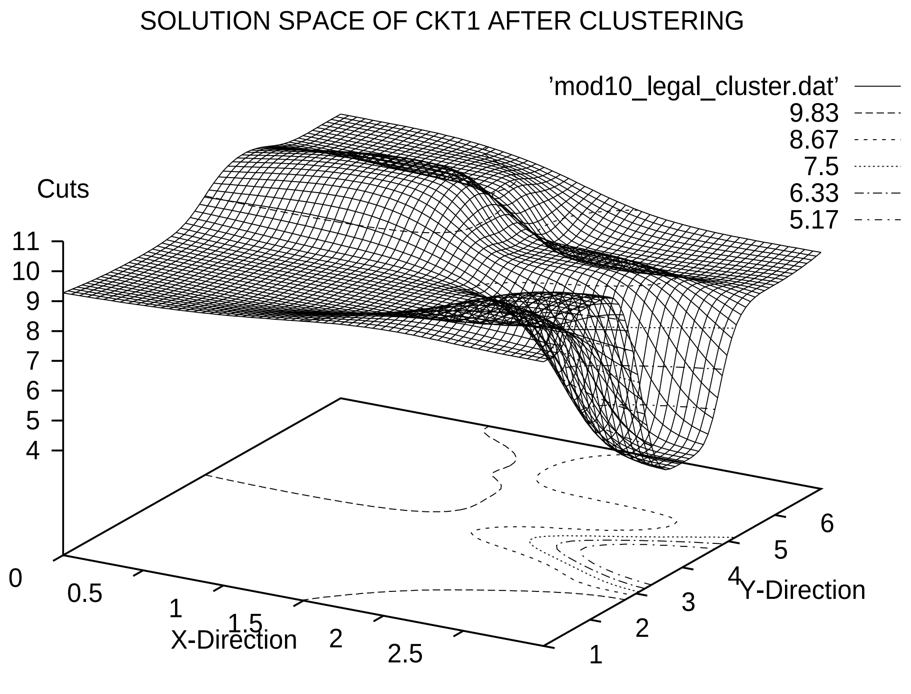


Figure 5.2: 3D view of a clustered network

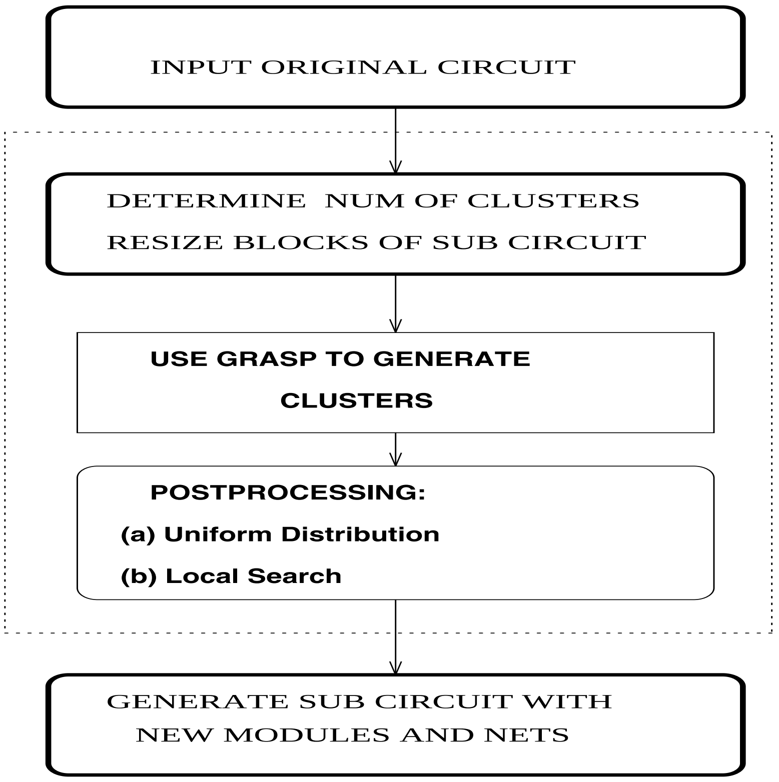

INPUT ORIGINAL CIRCUIT
DETERMINE NUM OF CLUSTERS
RESIZE BLOCKS OF SUB CIRCUIT
USE GRASP TO GENERATE CLUSTERS
POSTPROCESSING:
(a) Uniform Distribution
(b) Local Search
GENERATE SUB CIRCUIT WITH NEW MODULES AND NETS

Figure 5.3: A GRASP based clustering heuristic

partition or place the condensed network and (ii) the sizes of the generated clusters should vary over as small a range as possible.

The basic heuristic is shown in Figure 5.3. The heuristic does not generate natural clusters (the number of clusters has to be predetermined) and the number of clusters usually is set as a function of the number of partitions required. The cluster size C_{sz} plays an important role in the performance of the clustering heuristic to be used. If C_{sz} is too large, information connectivity could be hidden and missed by the preprocessing stage of the clustering technique being used. On the other hand, if the cluster size is small then the performance of subsequent stages of clustering will deteriorate since the clustered circuit would be of the same complexity of the original circuit. Figure 5.4 shows the quality of solutions obtained

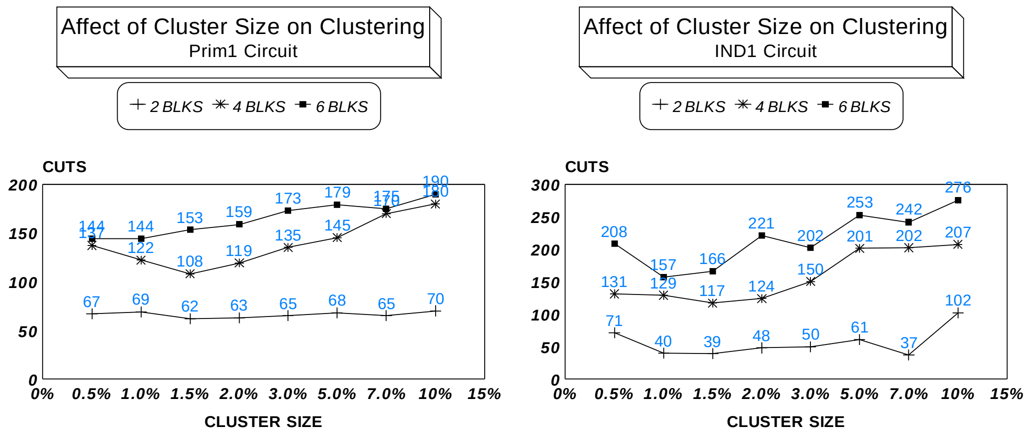


Figure 5.4: Performance of clustering based on cluster size

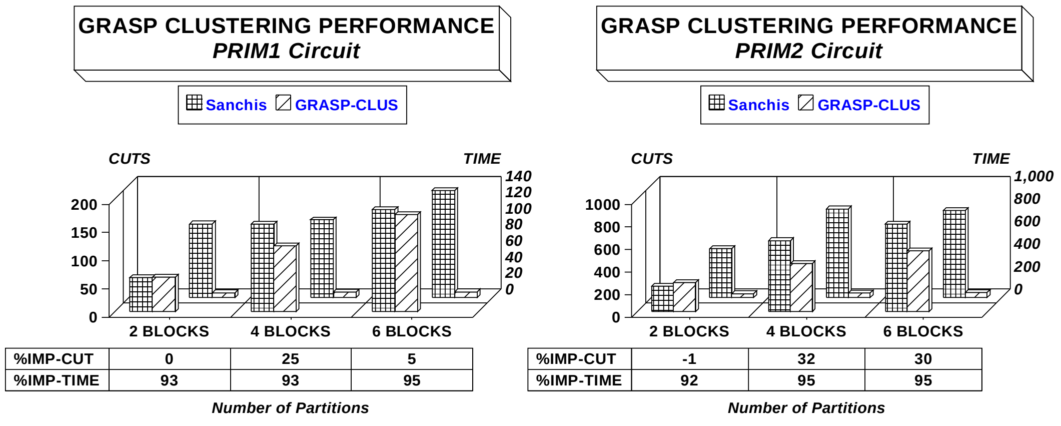
using different cluster sizes. The best quality of solutions obtained were based on sizes between 1% and 5% of the total original circuit size. Initially the heuristic would read in the circuit description, and resize the blocks to be used by the GRASP heuristic. GRASP as has been described in Chapter 3 would utilize only the construction phase to generate the required number of clusters. The parameters used by GRASP are tuned according to the size of the circuit and number of clusters required. A post processing stage follows the GRASP construction phase. In this stage, modules are initially uniformly distributed among partitions, this is followed by a simple dynamic hill climbing interchange pass that improves upon the initial solution generated. The computational results are shown in Table 5.2 and Figure 5.5.

Table 5.2 compares the results based on *Flat Mode*[1] with those based on 2-phase clustering. The results indicate that this simple clustering technique is effective in

[1]This mode refers to the case where the original circuit is partitioned.

GRASP Clustering Using SANCHIS Interchange Heuristic												
Blk	IND1 Circuit						IND2 Circuit					
	FLAT		CLUS		IMP		FLAT		CLUS		IMP	
	C	T	C	T	C	(T)	C	T	C	T	C	(T)
2	42	211	40	27	5%	87%	593	2661	477	209	19%	92%
4	259	526	122	29	52%	94%	2102	12729	1045	243	50%	98%
6	364	769	227	35	37%	95%	2430	25132	1330	280	45%	98%

Table 5.2: Results using Sanchis interchange with GRASP clustering



%IMP-CUT	0	25	5
%IMP-TIME	93	93	95

%IMP-CUT	-1	32	30
%IMP-TIME	92	95	95

Figure 5.5: Performance of the GRASP clustering technique

reducing the computation time, and improving the solution quality. The improvement in cut-sizes obtained for 4 and 6 blocks for most circuits are more profound than those for 2 blocks. This is because the performance of Kernighan-Lin based partitioning heuristics deteriorates as the number of blocks required increases (multiple partitions dramatically increase the solution space). Since clustering reduces the solution space for the problem, this explains the outcome of the tabulated results.

5.3 A Statistical Clustering Heuristic

In the previous section, a clustering technique based on GRASP was introduced. In this section, a novel clustering heuristic that utilizes statistical information of the circuit is presented. The overall clustering heuristic consists of 3 steps, as shown in Figure 5.6. A preprocessing stage is initially performed to eliminate long nets. This decreases the runtime and memory requirements considerably. The second step in the heuristic calculates the statistical information and attributes of the circuit. This is described in Section 5.3.1. Finally, a post processing step is performed to further reduce the number of unclustered nodes. Details of the developed clustering heuristic are described in the following subsections.

5.3.1 Concepts and Terminology

The statistical clustering technique is based on prioritized attributes, where modules are merged according to an attribute list. The attributes include terminal count, common net count, the number of nets localized, common net fan-out, and the cluster size. The main local criteria used to form clusters are:

- Γ, the binding factor of the nets,
- Λ, nets incident on modules,
- Δ, modules incident on nets,
- Φ, the attractive force between modules.

5.3.1.1 Binding Factor of Nets

The *binding factor* (Γ_i *or* Γ_{AB}) of a net, say Net_i that connects modules ($A,\ B$) measures the connectivity (attractiveness) between net i and other nets that are incident on modules $A,\ B$. In other words it represents the sum of pins that are located on modules $A,\ B$ excluding the contribution of pins made by net Net_i. The formula used to calculate this factor is :

$$\Gamma_i = Bind(Net_i) \ = \ Total\_Pins(A, B) - Pins(Net_i) \tag{5.1}$$

5.3.1.2 Relationship of Nets and Modules

Factors that relate nets to modules (Λ and Δ) measure the attractiveness of a module to a net and vice versa. The factor Λ, indicates the number of nets incident on a cell or the number of pins located on that cell. The higher the value of Λ for a certain cell, the more critical the cell becomes. On the other hand, Δ indicates the number of cells incident on a certain net. Nets that have a high value of Δ are usually power or clock nets. The importance of this factor will become evident later on in Section 5.5.1.

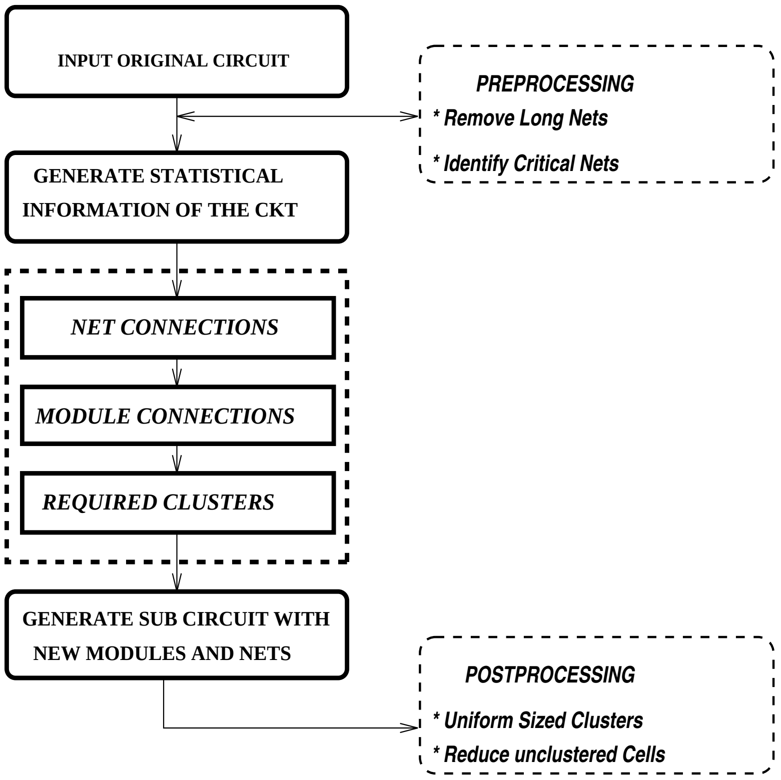


Figure 5.6: A clustering heuristic based on attributes

5.3.1.3 Attraction Force Between Modules

The *attractive force* between two cells represents the closeness factor due to common nets between the cells. The attractive force between two cells or clusters, A and B, is evaluated by the formulae:

$$\Phi_{A,B} = attraction(A,B) \ = \ \Omega_{A,B}/\Psi \tag{5.2}$$

where $\Omega_{A,B}$ is the number of common nets between A *and* B, and Ψ is a function that can have the following values:

$$\Phi^0_{A,B} = \Omega_{A,B}/\Psi^0, \qquad \Psi^0 = 1, \qquad \Omega_{A,B} = 1, \tag{5.3}$$

$$\Phi^1_{A,B} = \Omega_{A,B}/\Psi^1, \qquad \Psi^1 = 1, \tag{5.4}$$

$$\Phi^2_{A,B} = \Omega_{A,B}/\Psi^2, \quad \Psi^2 = \min(\Lambda_A, \Lambda_B) \tag{5.5}$$

$$\Phi^3_{A,B} = \Omega_{A,B}/\Psi^3, \quad \Psi^3 = \Gamma_{AB} + 2 \;\; or \qquad \sum \Lambda_A + \Lambda_B, \tag{5.6}$$

$$\Phi^4_{A,B} = \Omega_{A,B}/\Psi^4, \quad \Psi^4 = 1/\max(\Lambda_A, \Lambda_B) \tag{5.7}$$

Formula 5.3 represents only the connectivity between two modules. On the other hand Formula 5.4 represents the number of nets common between two modules. The last three formulas represent the attraction between two modules based on the common nets between them and the number of pins located on each module.

Figure 5.7 presents two sub-circuits that are extracted from the benchmarks presented in Chapter 3. In Figure 5.7a, we show the effect of each value of Ψ in combining Module **A6** to the three modules A4, A5 and A7 respectively. Table 5.3 shows the preference of each function. $\Phi^0_{A,B}$ indicates only that the four modules are connected. $\Phi^1_{A,B}$ shows that module A5 should be connected first to module A6. $\Phi^2_{A,B}$ and $\Phi^3_{A,B}$ indicate that A5 should be connected first to module A6 followed by module A7. Finally, $\Phi^4_{A,B}$ indicates that module A4 should be connected to module A6 before A7. In this example it is more beneficial to use $\Phi^4_{A,B}$ over the other attraction functions. Figure 5.7b shows an example with six modules connected together. Here, we show the preference of combining modules in this sub-circuit to module B14. The attraction functions $\Psi^0_{A,B}, \Psi^1_{A,B}$ *and* $\Psi^2_{A,B}$ indicate only that module B24 be connected to module B14 but does not show any preference in

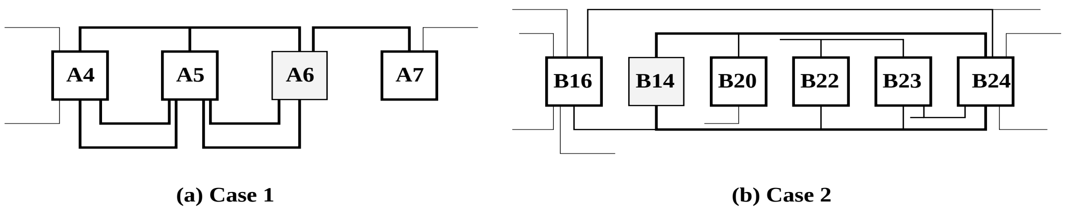


Figure 5.7: Attractive force between modules

Module	$\Phi^0_{A,B}$	$\Phi^1_{A,B}$	$\Phi^2_{A,B}$	$\Phi^3_{A,B}$	$\Phi^4_{A,B}$
A4	1	1	0.25	0.11	5.0
A5	1	3	0.75	0.33	15.0
A7	1	1	0.50	0.17	4.0

Table 5.3: Case1: preference values depending on Ψ

combining other modules to B14. $\Psi^3_{A,B}$ prefers combining B20 to B14 even though only one net connects the two modules. Finally, $\Phi^4_{A,B}$ indicates that module B16 should be merged following module B24. Attraction functions based on different values of Ψ are compared using three circuits. Table 5.5 summarizes the results obtained. The characteristic feature of this scheme is this *look-ahead*, which ensures that when we form a new cluster, it is the best possible with the seed and the candidate at that time. As expected the attraction functions based on Ψ^2, Ψ^3 and Ψ^4 give the best results.

Module	$\Phi^0_{A,B}$	$\Phi^1_{A,B}$	$\Phi^2_{A,B}$	$\Phi^3_{A,B}$	$\Phi^4_{A,B}$
B16	1	1	0.50	0.12	6.0
B20	1	1	0.50	0.25	2.0
B22	1	1	0.50	0.20	3.0
B23	1	1	0.50	0.20	3.0
B24	1	2	1.00	0.22	14.0

Table 5.4: Case2: preference values depending on Ψ

Circuit	$\Phi^0_{A,B}$	$\Phi^1_{A,B}$	$\Phi^2_{A,B}$	$\Phi^3_{A,B}$	$\Phi^4_{A,B}$
Prim2	353	340	334	326	298
Ind2	709	660	658	634	572
Ind3	1133	1154	1008	944	985

Table 5.5: A comparison between attraction functions

5.3.2 Statistical Information and Attributes

During the clustering process, some statistical information is obtained from the circuit under consideration. Some of the statistics that are gathered are: total number of pins in the circuit, average number of pins, maximum fanout, minimum fanout, maximum net binding value, the longest net, the different type of cells (those having X nets incident on them), and the different type of nets (those having Y modules incident on them). All the attributes that were discussed in Section 5.3.1 are generated as well.

5.3.3 Cluster Size Thresholds

The nodes and nets in a cluster must satisfy the area and size thresholds. The size threshold of a cluster depends on the clustering criteria to be used. If the criterion to be used requires *natural clusters* then the size of each cluster is not known until the heuristic terminates. In this case, cluster sizes are not uniform and may vary according to the attractiveness factor between different modules and their corresponding seeds. On the other hand, if the *forced clusters* criterion is used then the size of each cluster is predetermined according to the following:

$$Max\_Clusters = Total\_Cells/2$$

$$Min\_Clusters = 2 \times Total\_Blocks$$

The size threshold is a function of the total number of cells in the original circuit that no cluster can exceed. Possible number of clusters are: Max_Clusters, Min_Clusters, Max_Clusters/Min_Clusters and other values set according to the number of partitions required. Table 5.6 presents a comparison between clustering based on natural sized clusters and forced clusters. The table clearly indicates that forced sized clusters give better results on average by 40%.

Natural vs Predetermined Sizes						
Circuit	**Natural Sizes**			**Forced Sizes**		
	Size	I-Cut	F-Cut	Size	I-Cut	F-Cut
PRIM1	53	370	153	1%	222	124
PRIM2	115	1414	559	0.5%	786	297
IND1	103	857	176	0.5%	437	126
IND2	176	4119	1764	0.1%	3405	935

Table 5.6: Performance based on size threshold

5.3.4 Searching and Forming Clusters

During the clustering phase, the sizes of clusters to be formed are set (see Section 5.3.3). The process starts by choosing a seed (free_module). The *free_module* is chosen either randomly or deterministically. In the former case, any module that is not marked yet as being locked to a cluster can be a candidate as an initial seed or free_module. In the latter case, new free_modules or seeds are chosen such that the corresponding value of Λ (nets incident on a cell) is maximum. Finally, modules that have high connectivity (Φ values) are then merged to the current free_module and marked as being locked. This process continues until the cluster formed satisfies the size threshold. Table 5.7 shows results for 4 circuits using the two different seed techniques. The clustering heuristic was run 5 times and the two techniques are compared for obtaining the best, average, and standard deviation values over the five runs. It is clear that the second technique based on MAX-Λ gives better results than the RANDOM seed technique. Another method worth investigating is based on sorting modules according to their Φ values (using the different five measures mentioned previously) and choosing the seed from the top of the list.

Circuit	RANDOM			MAX-Λ		
	BEST	$\overline{x}$	σ	BEST	$\overline{x}$	σ
Prim2	366	425	33	299	326	24
Ind1	137	153	10	92	131	23
Ind2	779	798	13	653	700	25
Ind3	1120	1264	81	893	1055	99

Table 5.7: Effect of seed on statistical clustering

5.3.5 Postprocessing

After clustering, one usually obtains a few big clusters, each containing for example more than 3% of the total number of cells, and many small clusters or even single cell clusters. To obtain a reasonable cell partitioning these small clusters have to be joined to some of the big clusters. Therefore, a post-processing step is executed on the clustered circuit for two reasons. The first, is to reduce the number of single, unclustered cells. The second reason is to uniformly break down clusters that are oversized. That is, as new clusters are formed, they are immediately considered in evaluating whether other clusters should be combined. The heuristic terminates when no more clusters can be combined without violating the constraint on the size. This helps to balance the sizes of the clusters and reduce the number of cells in the clustered circuit. In this step, a simple pass of local search is applied to collapse cells into clusters. The number of passes required by the heuristic is thus limited.

5.4 Partitioning Clustering Heuristics

The main disadvantage of using local search heuristics to solve the circuit partitioning problem in particular, is that, as the circuit size and number of blocks required increase, the heuristic gets trapped easily in local minima. A common way to overcome this problem is to use a clustering based technique prior to the execution of the local search heuristic. Figure 5.8 shows the two stage hierarchical partitioning methodology that combines the clustering technique with any traditional or advanced search partitioning heuristic implementation. In the first stage, the clustering technique described in Section 5.3 is used to condense the input network. After forming clusters, the condensed network is partitioned using a simple dynamic hill climbing search technique that utilizes GRASP for its initial solution. In the second stage, a local search heuristic is used on the flattened network to optimize local partitions of cells. The combined clustering and local search methodology can be viewed as a combined bottom-up and top-down approach.

5.4.1 Clustered Single Processing Techniques

Tables 5.8-5.10 present the results obtained for partitioning a circuit into 2,4 and 6 blocks. The search techniques used are: Sanchis Iterative Improvement heuristic, Genetic Algorithms and Tabu Search respectively. The first part of the tables represents running the heuristic in the *Flat Mode*, whereas the second part of the

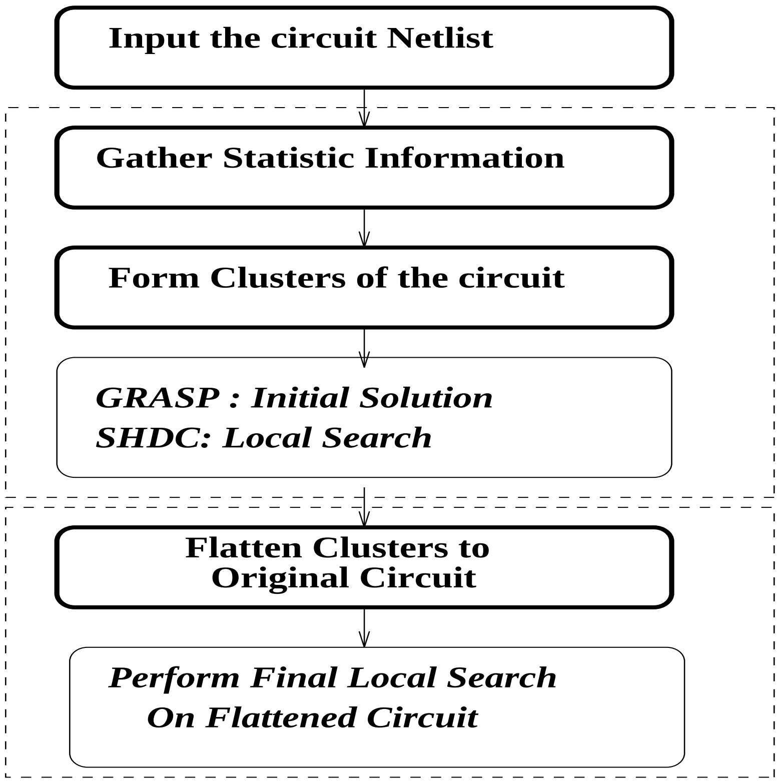

Input the circuit Netlist
Gather Statistic Information
Form Clusters of the circuit
GRASP : Initial Solution
SHDC: Local Search
Flatten Clusters to Original Circuit
Perform Final Local Search On Flattened Circuit

Figure 5.8: A combined clustering-partitioning heuristic

tables represents the 2-phase heuristic based on clustering. It is clear from the tables that clustering not only reduces the computation time to solve the problem, but also improves the quality of solutions drastically compared to solving the original circuit in the flat mode (unclustered network). Table 5.8 compares Sanchis Interchange results based on 50 different initial random runs to a single run based on the clustering-partitioning heuristic. The combined clustering-partitioning technique improves the quality of solutions on average by 40%. The computation time is further reduced by 80%. Table 5.9 indicates clearly the impact of clustering on the performance of the Genetic Algorithm. It is interesting to note that even though the quality of solutions have improved for the two mentioned circuits, the computation time of Industry2 circuit remained the same. The reason is due to the repair solution heuristic involved in this case (see Section 3.4.1.3). Finally, results obtained for the Tabu Search Heuristic (as shown in Table 5.10) indicate that even an advanced search heuristic such as Tabu Search (that avoids getting trapped in local minima) gains from clustering by improving the solution quality, and reducing the computation time. Figure 5.6 compares results obtained by the clustering technique using different heuristic search methods. It is clear that clustering has a more profound affect on the Sanchis Iterative improvement heuristic and Genetic Algorithm than on Tabu Search. The reason, as one would anticipate, is that Tabu Search has the capability of exploring the solution space more effectively than the above mentioned heuristics.

Clustering Using SANCHIS Interchange Heuristic												
Blk	PRIM2 Circuit						IND2 Circuit					
	FLAT		CLUS		IMP		FLAT		CLUS		IMP	
	C	T	C	T	C	(T)	C	T	C	T	C	(T)
2	226	433	176	34	22%	92%	593	2661	337	446	43%	83%
4	627	787	305	24	51%	97%	2102	12729	826	264	60%	98%
6	773	1382	462	26	40%	98%	2430	25132	1041	303	57%	99%

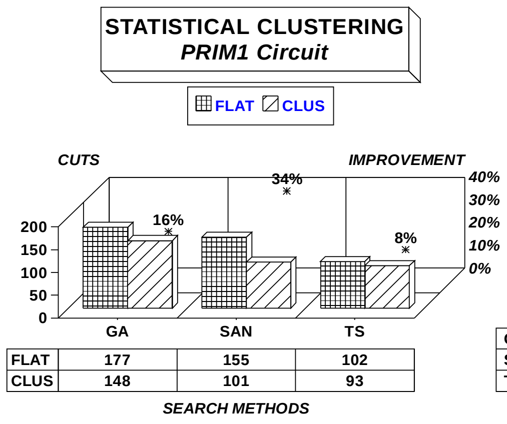
Table 5.8: The Sanchis interchange heuristic and statistical clustering

Clustering Using Genetic Algorithms												
Blk	PRIM2 Circuit						IND2 Circuit					
	FLAT		CLUS		IMP		FLAT		CLUS		IMP	
	C	T	C	T	C	(T)	C	T	C	T	C	(T)
2	190	36.3	183	28.5	4%	21%	572	166	360	266	37%	-
4	529	54.0	371	40.6	29%	26%	2363	274	1139	464	51%	-
6	671	71.1	547	63.2	18%	11%	3342	368	1490	984	55%	-

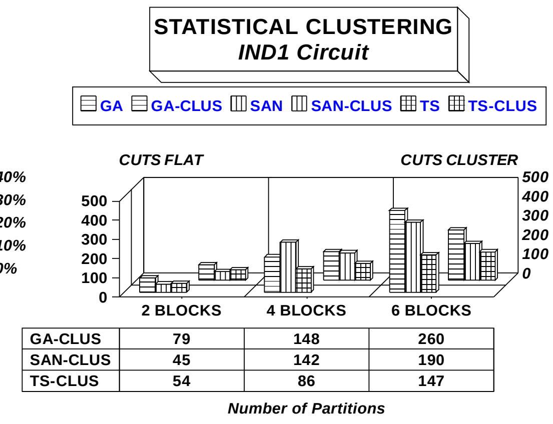
Table 5.9: Genetic Search based on statistical clustering

Clustering Using Tabu Search												
Blk	PRIM2 Circuit						IND2 Circuit					
	FLAT		CLUS		IMP		FLAT		CLUS		IMP	
	C	T	C	T	C	(T)	C	T	C	T	C	(T)
2	181	284	166	139	8%	51%	323	2250	333	1080	-3%	52%
4	357	299	309	206	13%	31%	991	3922	628	1251	36%	68%
6	562	435	459	165	23%	62%	1375	5373	1047	560	24%	89%

Table 5.10: Tabu Search and statistical clustering



	GA	SAN	TS
FLAT	177	155	102
CLUS	148	101	93



	2 BLOCKS	4 BLOCKS	6 BLOCKS
GA-CLUS	79	148	260
SAN-CLUS	45	142	190
TS-CLUS	54	86	147

Figure 5.9: Performance of the statistical clustering technique

5.4.2 A Clustered Hybrid Technique

In this section, results based on the GA-TS hybrid presented in Chapter 4 are discussed. Recall from Chapter 4 that the GA-TS hybrid approach produced better results compared to other hybrid approaches used. The Genetic Algorithm here uses a population of clustered chromosomes. The main advantage of using this technique is twofold. First, the memory requirements is reduced dramatically since the population of solutions used by the GA heuristic deals with a clustered network instead of the original network. The second advantage of using clustering, as mentioned before, is that the solution space is reduced, thus the GA is capable of exploring the solution space more effectively. Table 5.11 and Figure 5.10 present a comparison between GA-TS hybrid and CGA-TS which uses clustering for initial population of solutions. Results obtained indicate that the clustered based technique produces solutions that are at least 20% better than those obtained based on the original circuit representation for medium and large circuits. The clustering based heuristic has less impact on partitioning small circuits since the original representation suffices for the GA heuristic to explore the solution space effectively. In the next section, assessment of the results obtained by the combined clustering-partitioning approach with those obtained using an exact method based on a branch and bound technique is introduced.

Circuit	**Blks**	**GA-TS**		**CGA-TS**		**%IMP**
		Cuts	Time	Cuts	Time	
CHIP1	2 Blks	20	22.9	20	17.2	-
	4 Blks	46	36.1	44	20.2	2%
	6 Blks	56	45.1	55	25.8	1%
PRIM2	2 Blks	161	632	156	360	3%
	4 Blks	406	495	276	587	32%
	6 Blks	501	563	430	540	15%
IND2	2 Blks	273	3422	206	1547	24%
	4 Blks	943	5087	477	2030	49%
	6 Blks	1322	6600	1006	2896	24%
IND3	2 Blks	300	2750	289	3900	4%
	4 Blks	1405	10279	882	7080	37%
	6 Blks	1900	8848	1532	8160	20%

Table 5.11: Performance of GA-TS hybrid with clustering

5.5 Solution Optimality

Here, we undertake to study the ability of the developed clustering-partitioning heuristics in finding optimal solutions to the benchmarks presented in Chapter 3. Each problem is formulated as a linear integer program (MIP) as presented in Section 2.1.1.3 (Equation 2.4). Benchmark problems are optimally solved using CPLEX Version 3.0 [Inc93], and the results are shown in Tables 5.12–5.13. It is clear from Tables 5.12–5.13 that the strategy of combining clustering with hybrid

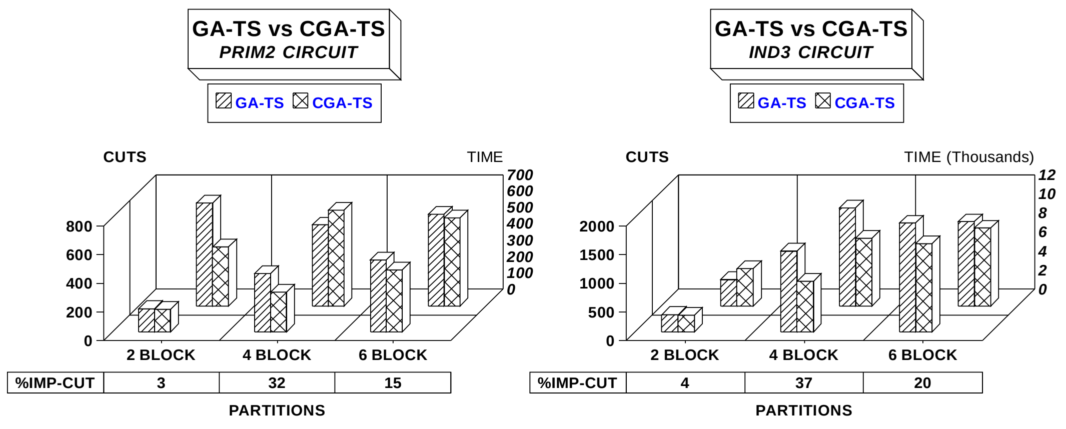


Figure 5.10: Performance of the clustered GA-TS heuristic

search techniques achieves near optimal solutions and are within 4% of optimality in the worst case. Tables 5.12–5.13 also present a lower bound obtained by solving the LP relaxation of the problem. The column in these tables denoted by LP-BOUND reads the solution of the LP relaxation as compared to the total nets of the circuit. The difference indicates the lower bound of the partitioning problem.

5.5.1 Improving CPLEX MIP Performance

One frustrating aspect of the branch-and-bound technique for solving MIP problems is that the solution process can continue long after the best solution has been found. In these situations, the branch-and-bound tree is being exhaustively searched in an effort to guarantee that the current integer feasible solution is indeed optimal.

The branch-and-bound tree may be as large as 2^n nodes, where n is the number of binary variables. A problem containing only 30 binary variables could produce a

Circuit	**Heuristic**			**CPLEX**		**Optimality**	
	Method	Cuts	Time	Cuts	Time	%OPT	LP-BOUND
Pcb1	CGA-TS	5	1.1	5	1.23	100%	4
Pcb2	CGA-TS	7	4.0	7	4.87	100%	2
Chip1	CGA-TS	20	106	20	2782	100%	5
Chip2	CGA-TS	14	2.7	14	728	100%	4
Chip3	CGA-TS	7	2.2	7	153	100%	6
Chip4	CGA-TS	7	2.4	7	104	100%	4

Table 5.12: Optimal solutions for 2-way partitioning

Circuit	**Heuristic**			**CPLEX**		**Optimality**	
	Method	Cuts	Time	Cuts	Time	%OPT	LP-BOUND
Pcb1	CGA-TS	10	0.9	10	329	100%	4
Pcb2	CGA-TS	11	6.2	11	839	100%	6
Chip1	CGA-TS	44	120	44	30.3h	100%	3
Chip2	CGA-TS	25	104	25	153.3h	100%	4
Chip3	CGA-TS	28	207	27	78.1h	97%	13
Chip4	CGA-TS	17	100	17	38.5h	100%	8

Table 5.13: Optimal solutions for 4-way partitioning

tree having over one billion nodes. If no other stopping criteria have been set, the process might continue for a long time, until the search is complete or the computer memory is exhausted. Therefore, many issues are to be investigated to improve the performance of the MIP solver.

The first issue is related to different parameter settings of the MIP solver. It is important to enable any mixed integer programming solver like CPLEX to attempt to reduce the size of the integer program by using a preprocessing phase. This strengthens the initial linear programming relaxation and reduces the overall size of the mixed integer program. The path CPLEX takes through the branch-and-bound tree is determined by a number of user inputs. For example, at each node, CPLEX can either delve deeper into the tree or "backtrack", the setting of the BACKTRACK parameter impacts this decision. Once CPLEX decides to backtrack, there are typically a large number of available, unexplored nodes from which to choose; the NODE-SELECT and VAR-SELECT parameter setting influences this selection. By looking at the solution to the LP relaxation, most variables take on the same value. This combined with the similarity of the variables makes it difficult for CPLEX to differentiate one variable from another in the branching process. The main difference between variables in the formulated problems is that different binaries enable different numbers of continuous variables to take on values. By setting the node select parameter to *best estimate strategy* instead of the depth-first search, the resulting computations implicitly take this difference into account. The variable select parameter is used to set the rule for selecting the branching variable at the node which has been selected for branching. Setting the variable select parameter to *pseudo-reduced costs* causes the variable selection to be based on pseudo-costs which are derived from pseudo-shadow prices.

Figure 5.11 summarizes the methodology used to improve the performance of the CPLEX MIP solver for the circuit partitioning problem. We call this method Stat-Heur for the CPLEX MIP solver. As seen in the figure, we make use of our heuristic based techniques in providing a lower bound that is used to cut off any nodes that have an objective value below the value supplied by the heuristic. We also utilize the solutions obtained by our heuristics to fix a few modules in the original formulation. Given that we are maximizing the sum of continuous variables, it seems that setting a binary variable to 1 will have the most favorable impact if that variable can enable a larger number of continuous variables to take on a value of 1. Another important issue worth investigating is the symmetry of the

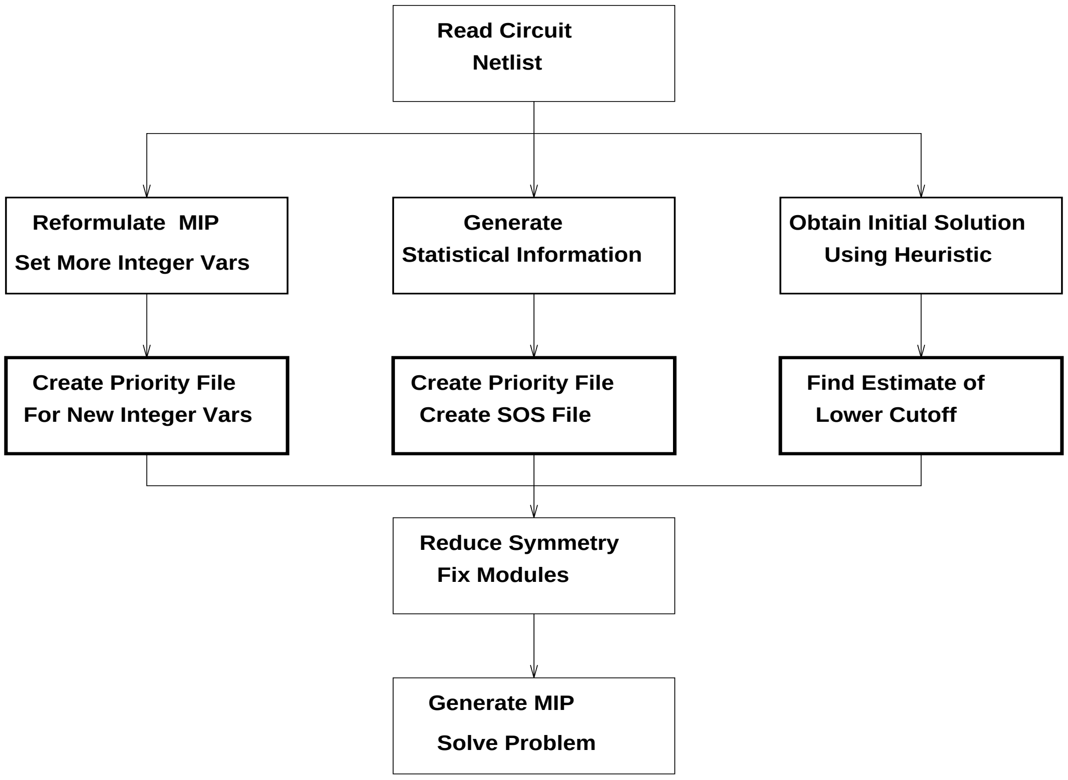

Read Circuit Netlist
Reformulate MIP Set More Integer Vars
Generate Statistical Information
Obtain Initial Solution Using Heuristic
Create Priority File For New Integer Vars
Create Priority File Create SOS File
Find Estimate of Lower Cutoff
Reduce Symmetry Fix Modules
Generate MIP Solve Problem

Figure 5.11: Stat-Heur heuristic and CPLEX MIP

MIP problem. With the current formulation, the problem is extremely symmetric. There is very little to distinguish one variable from another, both in terms of the objective function and the constraints. This feature can slow the MIP performance, because when we branch down on a variable, we can easily shift it's activity level to another variable. Reducing the symmetry of the MIP, reduces the number of equivalent solutions, which will reduce the amount of work needed in the branch and bound process. One way to reduce the symmetry is to formulate with patterns instead of assignments. Another means of accomplishing this is through priority orders. Priority orders provide a very powerful mechanism for adding user-supplied, problem-specific direction to the branching process. The additional information in the form of a priority order file can be used to alleviate the effect of the symmetry if reformulation is not possible. Statistical information of the circuits have been used extensively to provide such information. The factor Λ that determines the relation between modules and nets is used to give priority to the modules (x variables) that have a high percentage of nets incident on them over variables that have less incident nets.

Since priority orders can supply information on integer variables only, and since the y variables are continuous (see Section 2.1.1.3), the priority orders were restricted on the x variables. However, because of the nature of our circuit partitioning problems, we managed to declare the y variables to be integer without changing the meaning of our model. Although one's intuition would say it's a mistake to declare more integer variables, this is not always the case, especially if we could provide some priority order information regarding the y variables. For the y variables, Δ, the number of modules incident on the nets was used to provide such information. The higher the Δ factor the higher the priority given to the y

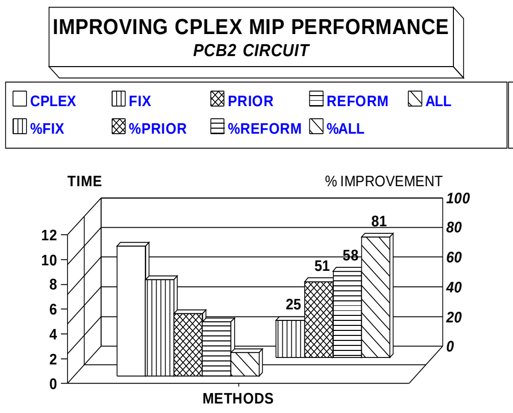
variable.

Finally, a highly effective method to improve performance is to take advantage of the presence of *Special Ordered Sets* (SOSs). Special branching strategies are available to take advantage of SOSs. These strategies depend upon the SOS problem definition to include ordering (or weighting) information. If there is no order or weight assigned to individual SOS members, using SOS branching strategies may not improve–and can even degrade–performance. It is important to note in Figure 5.11 that when SOSs are used after specifying a priority order file on individual variables, CPLEX ignores the order on individual variables because SOS sets involve a branching procedure where the SOS sets are branched on, not individual variables. Therefore, SOSs are not specified when using priority orders.

Figure 5.12 indicates clearly the impact of using our heuristic based approaches in improving the performance of the MIP solver. The figure shows that for the PCB2 circuit up to 96% reduction in computation time was achieved using the Stat-Heur technique. Table 5.14 presents the amount of reduction in computation time for different circuits based on 2 and 4 partitions.

Improving CPLEX MIP Performance						
Circuit	**2 BLKS (Secs)**			**4 BLKS (Hrs)**		
	CPLEX	H-CPLEX	SPEED-UP	CPLEX	H-CPLEX	SPEED-UP
CHIP1	2782	1056	62%	92	6.8	92%
CHIP2	728	329	54%	89	4.4	95%
CHIP3	153	67	56%	78	2.26	97%
CHIP4	104	61	41%	38.5	1.18	96%

Table 5.14: Heur-Stat heuristic used with CPLEX MIP solver



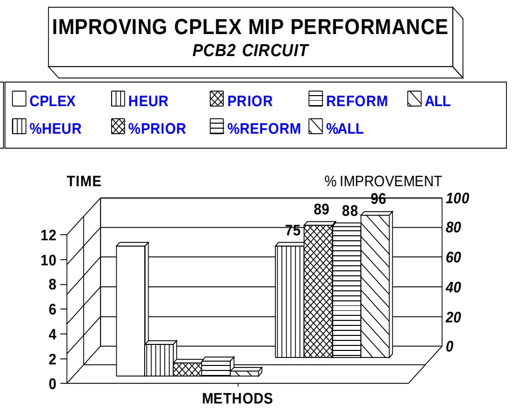


Figure 5.12: Effect of Stat-Heur in improving the CPLEX MIP performance

5.6 Summary

In this chapter, two effective clustering heuristics based on different techniques were introduced. The GRASP based clustering heuristic is efficient and simple to implement. The second scheme based on statistical information of the circuit improves upon the first clustering heuristic in quality of solutions. Both techniques are based on a two stage local search process. Clusters of cells that are highly connected are formed, and then an efficient assignment to their initial location within different blocks is performed. A simple dynamic hill climbing heuristic (SDHC) is then used to improve the solution at the cluster level. Finally, a second local search stage is used to refine the solution at the cell level. This latter process confines the cells to local moves, since they are already in their proper position. Results obtained in the previous sections indicate that the simple clustering heuristics not only reduce the complexity of the problem but also speed up the execution of the partitioning process in a fraction of the time.

The final goal of this chapter was to assess the quality of solutions obtained by the developed combined advanced, hybrid, clustering techniques. Solutions were compared with those obtained by the CPLEX MIP mixed integer programming package. The results indicate clearly that we are truly heading towards optimal solutions for the circuit layout problem. In fact, the developed heuristics were successful in improving the performance of the CPLEX MIP solver by providing information that was necessary to reduce the computation time on average by 70%.

In view of the increasing complexity of VLSI circuits of the future, the requirements on VLSI CAD tools will continuously increase. Parallel processing for CAD applications is becoming gradually recognized as a popular vehicle to support the increasing computing requirements of future CAD tools. The next chapter discusses techniques for distributing the partitioning heuristics developed in this dissertation as a means to reduce the computation time involved.

Chapter 6

Distributed Search Techniques

Parallel and distributed computing systems offer the promise of a quantum leap in the computing power that can be brought to bear on many important problems. The potential for distributed processing exists whenever there are several computers interconnected in some fashion so that a program or procedure running on one machine can transfer control to a procedure running on another. In such an environment we wish to assign optimally the modules of a program to specific processors. The main objective in optimizing is twofold, minimizing the running time of the program and improving the efficiency of the algorithm. Our main task in this work is to develop an environment that allows easy parallelization of the existing sequential algorithms, in which the potential parallelism fits easily into the sequential algorithm.

This chapter discusses techniques to parallelize the circuit partitioning heuristics that were described in the previous chapters. The next section discusses the motivation behind the parallel implementation. Section 6.2 describes the *Network Multiprocessing Environment* [?]. The general approach to parallel partitioning is described in Section 6.3.

6.1 Motivation

Parallelism may be applied in several ways to increase the processing power available to the execution of a program. These approaches can be broadly categorized into two groups, namely, closely coupled or synchronized processors, and loosely coupled or distributed systems. Closely coupled systems have traditionally been more popular since they can be used to speed existing algorithms and programs. A typical CAD environment consists of a number of workstations as seen in Figure 6.1 connected together by a high speed network that allows a team of designers to access the design database. Most individual CAD problems are typically solved on a single workstation. The loosely coupled parallel computing environment provided by the network is not used effectively by CAD algorithms. The workstations typically can support several user tasks, one of which can run in the foreground while the others run in the background. When the workstation is used for a text editing or graphics editing program, it may use its CPU only a fraction of the time and spend most of its cycles waiting for user input. The CAD programs described in this dissertation can make use of the idle cycles on all the networked machines to speed up the solution process.

6.1.1 Multiprocessing Issues

The most promising algorithms used for circuit partitioning and placement are based on iterative improvement methods. Work by [?] and [?] showed that parallel implementation of these pairwise interchange methods can be accelerated in two ways, by performing several moves in parallel and by performing the subtasks for each move in parallel. The amount of parallelism within a move is limited, and good speedup can be achieved only by performing several moves in parallel. However, if moves are performed in parallel, the system can get into an inconsistent state unless the processors are synchronized after every set of noninteracting moves is calculated in parallel. There is a tradeoff between the communication costs and the need to broadcast information after each accepted move. Typically, several moves are accepted before a broadcast, which leads to some errors. Since communication costs are high even on a tightly coupled system, it is not desirable to speed up interchange methods on a distributed system (by performing several moves in parallel) where communication time is in the order of several milliseconds.

6.1.2 Algorithm Development Strategy

Having taken a brief look at different parallel processing issues, it is time look at how parallel processing can be applied to speed up VLSI CAD in general and circuit layout in particular. Developers of applications for parallel processing must choose between implementing their ideas in special-purpose or general-purpose hardware. The primary trade-off is performance gain versus flexibility in implementing applications. Special hardware (such as a network of transputers) usually yields higher performance for specific applications whereas general-purpose (i.e, network of workstations) can be programmed for a variety of applications. Both approaches have been pursued in the past by researchers in the VLSI design automation community. A major limitation with almost all previous work is that the parallel algorithms have been targeted to run on specific machines like an Intel $iPSC^{TM}$ hypercube-based message passing distributed memory multicomputer or an Encore $Multimax^{TM}$ shared memory multiprocessor. Such work, although interesting, is not usable by the rest of the VLSI CAD community since the algorithms are not portable to other machines.

The most important questions that need to be addressed in the development of parallel algorithms are therefore: (i) How can parallel algorithms be designed such that they are truly portable across parallel machines? (ii) How can one exploit good sequential algorithms in the design of parallel algorithms? and (iii) How can parallel algorithms keep pace with future developments in sequential algorithms? These are the main objectives of this work.

Our primary strategy is to utilize the network of workstations (portability) in such a way to minimize the interaction between processors, and at the same time utilize the same sequential powerful heuristics developed with minimum modifications (preserve optimality) on the distributed platform. An attempt will first be made to test (synchronization, communication) the distributed environment. This is carried out by implementing a simple distributed scheme where each processor (workstation) starts from a different initial point in the solution space, iteratively improve it, and ultimately report the final solution to the master processor. The second goal is to improve the performance of the circuit partitioning methodology based on Genetic Algorithms which lends itself naturally to distributed processing, by dividing the initial population of solutions among n processors, and allowing these subpopulations to migrate between processors for further mating. This will attempt to improve solution quality and speed up the convergence rate of the algorithm. Our third goal involves extending the Tabu Search-Genetic Algorithm Hybrid to the distributed platform, by distributing the diversification and intensification phases among different processors. This will attempt to cover a larger portion of the solution space, thus improving the quality of solutions. Finally, we hope to utilize the distributed system to increase the neighborhood search for the placement problem, which is considered to be one of the main limitations of solving this problem.

6.2 A Network Multiprocessor Environment

This section describes an environment designed and built to do experiments in distributed processing, using standard equipment and utilizing the services of a contemporary operating system to reduce hardware and software costs and to simplify experiment management. The Network Multiprocessing Environment is a package intended to make distributed, concurrent applications in BSD UNIX socket network easier to write [?]. It sets up a multiprocessor like structure using processes on a UNIX network. The structure of the interprocess communication paths is an arbitrarily interconnected graph. For distributed applications of the client/server model, the Sun Remote Procedure Call (RPC) package provides a good interface to the socket level of BSD UNIX. RPC, however is not well suited to multiprocessor simulations or applications of a more concurrent nature. NMP provides procedure calls that allow easy parallelization of existing sequential algorithms. The NMP environment is implemented using a set of standard NMP routines that allow the creation of virtual multiprocessors with arbitrary interconnection structures as seen in Figure 6.2. These standard routines are in turn built onto the UNIX networking primitives. Those primitives allow processes to communicate with a variety of protocols and connection strategies. The current implementa-

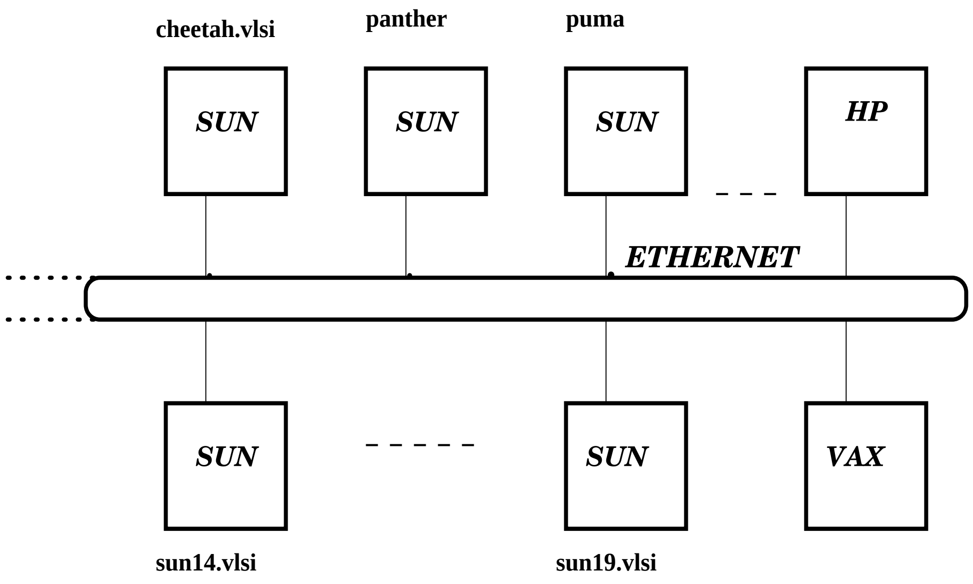


Figure 6.1: Computing facilities

tion of the support routines uses reliable two-way communication channels (called stream-sockets). Stream sockets are similar to UNIX pipes [?], except that the communicating processes need not reside on the same physical machine.

6.2.1 Main NMP Routines

The execution environment consists of a collection of virtual processors (UNIX processes) whose interconnections (UNIX stream-socket connections over an Ethernet) are created on initialization. In the NMP applications, nodes are referred to by their *node ID* (an integer). The node *ID* is simply the cardinal number of the node description line in the configuration file, starting with the root as node 0. The four basic NMP support routines are: **NodeInit, SendNode, RecvNode** and **NodeClose**. These four routines form the core of the system and programs must use (some form) of these routines. To get started a node (process) initialization func-

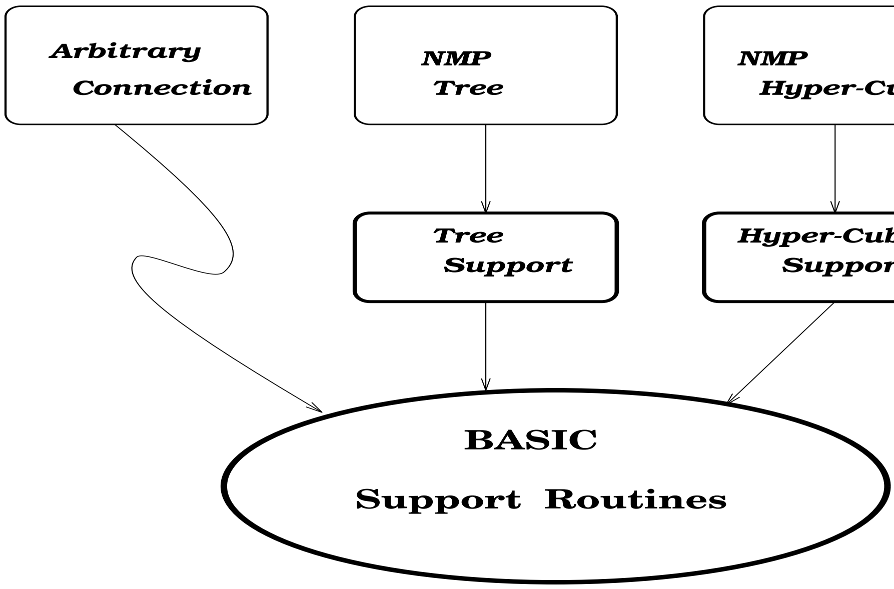


Figure 6.2: NMP interconnection structure

tion must be invoked, as follows: *neighbors = NodeInit(Type, ConfigFile)*, NodeInit initializes the node by reading a ConfigFile and creates other nodes (processes) as specified in that file. There are two types of node initializations, the first Type=0 is used in the node that interacts with the user. All other nodes are of Type=1 and receive the interconnection information from their creator. The configuration file consists of two parts. The first part is an ordered set of descriptor lines, one per node, naming the host process, execution file name and so on, the second part is a connection matrix, showing how the ordered nodes are connected.

All messages passed through the NMP are sent using: *SendNode(NodeID, Message, Length)*. The data in memory at the location pointed to by Message are sent to the node NodeId. The number of bytes to send is given by the Length parameter. SendNode returns the number of bytes successfully sent. To receive information sent from SendNode, the following is used: *RecvNode(NodeId, Message, Length)*.

RecvNode will only return with a partial message if the sender terminated before completing transmission, otherwise, it will always block until the entire message is read. If the application needs to check for incoming messages without using the blocking RecvNode routine, or if it is waiting for messages from more than one sender, the polling routines may be used. To poll all the neighboring nodes, use: *PollAll(Nodes, Block)*. If any data are waiting to be read from nodes, the sender node *IDs* are returned in the array Nodes (which should be large enough to contain all nodes which may have incoming messages). To check an explicit node for pending messages, use: *PollNode(NodeId, Block)*. Other useful routines are found in Appendix B.

6.3 Parallel Partitioning

In most sections comparison are made between a sequential implementation and a distributed implementation that is run using 10 workstations (unless otherwise indicated).

6.3.1 A Simple Distributed Partitioning Heuristic

Iterative improvement techniques that make local changes to an initial partition are still the most successful algorithms used in practice. The advantage of these heuristics is that they are quite robust. But in general, node interchange are greedy or local in nature and get easily trapped in local minima. The performance of these heuristics depend on the initial solution used. As the number of initial solutions increase, this allows the heuristic to diversify the search effectively. In this subsection, a distributed implementation of the iterative improvement technique is presented.

```
Root (Master):
        Read configuration file (Initial Structure)
        Setup connection with all processors.
        Confirm connections with all processors.
        Broadcast arguments to all processors.
        Poll All processors for final result.
        Determine the best solution.
        Report solution quality and timing information

Servers (Slaves):
        Confirm connection to root
        Receive arguments for execution
        Read the circuit and set data structures
        Obtain different initial solution for partitioning
        Perform interchange algorithm
        Send results to Root
        Close Node
```

Figure 6.3: A distributed partitioning heuristic

Figure 6.3 shows the overall algorithm used to implement the simple distributed partitioning scheme. Figure 6.4 shows the execution mechanism using the sequential algorithm versus the parallel implementation. It is important to notice that one should not expect a linear speedup using the distributed network based on the above mentioned scheme. The main reason is that some runs take more time for execution than others. Therefore the distributed version best time depends on the longest run and the communication of solutions among processors (Figure 6.4). Initially the Master processor sets up the connection to all processors that are involved in the configuration file. The slaves in turn would confirm the connections. The master would then broadcast the necessary arguments that are needed to execute the program. Some of these arguments are: the program name, the circuit name, the number of blocks, and other important parameters that are used by the main program. Once these arguments are received the Slaves would then read the input circuit, obtain an initial solution for the circuit partitioning and perform local optimization based on the Sanchis multi-way partitioning scheme. The Master in turn would poll all processors for the final result and report the best solution and timing information.

Table 6.1 and Figure 6.5 present results using the distributed Sanchis heuristic. Results indicate that on average a speedup of 7 can be obtained using the above mentioned heuristic.

6.3.2 A Distributed Genetic Search Heuristic

There are many simple avenues to parallelize a sequential Genetic Algorithm (assuming a global shared memory) [Coho91] e.g., selecting and crossing over pairs

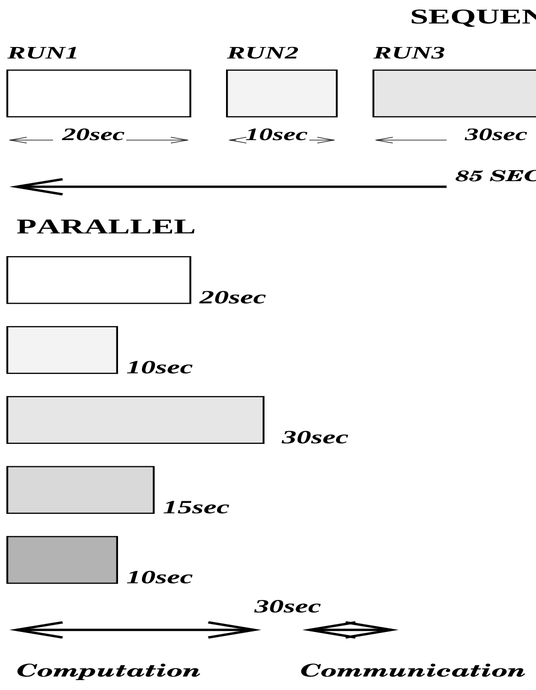


Figure 6.4: Sequential vs parallel computation

Distributed Sanchis Interchange Heuristic										
Blks	**Prim1 Circuit**					**Bio Circuit**				
	SEQ		**DIS**		**Xs**	**SEQ**		**DIS**		**Xs**
	Cuts	T	Cuts	T		Cuts	T	Cuts	T	
2	75	5.8	75	1.1	5	143	75	143	10.7	7
4	155	7.4	155	1.2	6	724	200	724	27.3	7
6	181	9.2	181	1.1	8	821	350	821	60.1	6

Table 6.1: Sanchis interchange heuristic

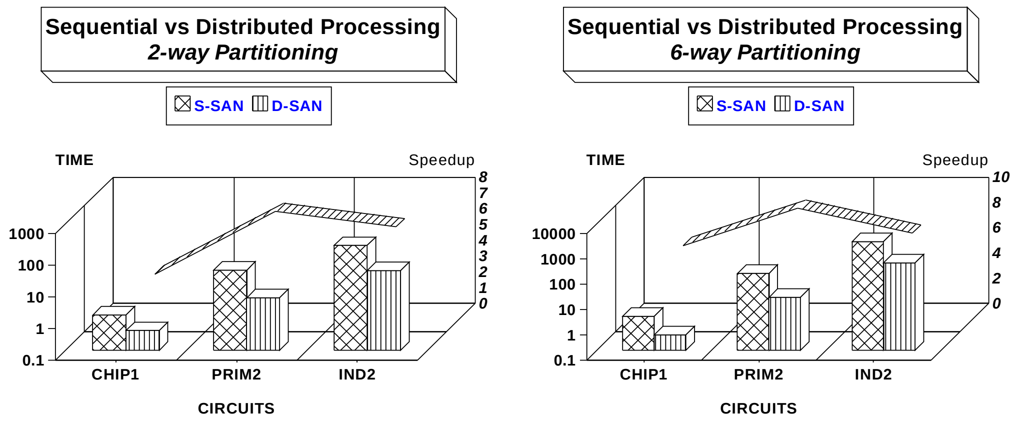


Figure 6.5: The performance of Sanchis distributed heuristic

of solutions in parallel, and mutating solutions in parallel. However, such avenue results in only a simple hardware accelerator, and will not be suitable for the local memory, distributed model of computation. Therefore, the attention is focused on distributing the population of solutions among the processors and allowing the migration of these subpopulations for mating and producing excellent offsprings. This method provides a suitable paradigm to map Genetic Algorithms onto a distributed system. The Genetic Algorithm code used by each processor is shown in Figure 6.6. Table 6.2 shows the results of the distributed Genetic Algorithm. The table clearly shows that the distributed Genetic heuristic achieves good speedup in execution time. The table also shows that there is a difference between the sequential and distributed version in solution quality. In most cases the heuristic achieves speedup with the same or better quality of solutions. Even though the population size is distributed among processors, the infeasible solutions of these subpopulations are repaired using an efficient *repair heuristic*, thus optimized to have a better affect. Since the subpopulations used are small compared to the population used for the sequential heuristic, we can afford to use an efficient repair heuristic. Another reason for the quality of solutions is due to the diversification of initial solutions generated on different processors. It is worth noting that the parallel implementation for the Genetic Algorithm gives better results than the sequential heuristic used.

```
Parallel Genetic Algorithm:
    Read Circuit
    For E iterations do
      For Each Processor i do
        Run GA for G Generations
      End For
      For Each Processor i do
        Send a set of solutions to Master
      End For
    End For

Genetic Algorithm within each Processor:
    For G Iterations do
      While offsprings created $\leq$ X
        Select two solutions
        Crossover to obtain offsprings
      EndWhile
      add offsprings to subpopulation
      Calculate Fitness
      Select Population of n elements
    End For
```

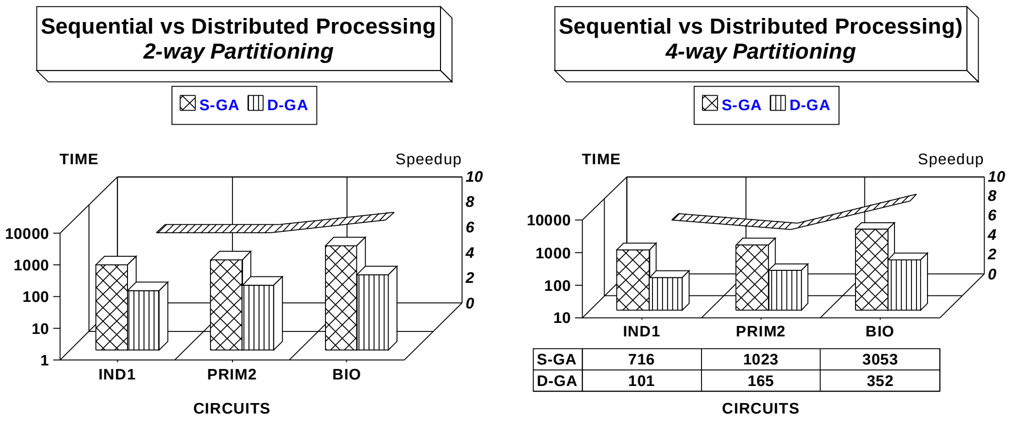
Figure 6.6: A distributed Genetic Algorithm

6.3.3 A Distributed Genetic-Tabu Search Heuristic

Chapter 4 introduced a Tabu Search-Genetic Algorithm Hybrid based on the inter-communicating hybrid model. In this model, the independent processing modules in the form of Tabu Search, GRASP, Genetic Algorithms and simple local search

Distributed Genetic Search										
Blks	Prim1 Circuit					Bio Circuit				
	SEQ		DIS		Xs	SEQ		DIS		Xs
	C	T	C	T		C	T	C	T	
2	76	185	75	22	8	282	1982	284	237	8
4	151	268	155	34	7.7	1164	3053	1146	352	8.6
6	162	356	172	47	7.5	1549	4065	1158	478	8.5

Table 6.2: Sequential and distributed GA



	IND1	PRIM2	BIO
S-GA	716	1023	3053
D-GA	101	165	352

Figure 6.7: The performance of a distributed GA heuristic

exchanged information and performed separate functions to generate near optimal solutions. The main task of the hybrid algorithm was to generate good initial starting points, locally fine-tune the search and finally diversify and intensify the search in the solution space. The distributed Tabu Search Genetic Algorithm (GA-TS) code used is shown in Figure 6.8. Table 6.3 presents the results obtained for the distributed GA-TS hybrid. Even though the speedups obtained are not linear (as expected), we still benefit from using the distributed NMP approach, since the execution time is faster by at least a factor of three.

```
Master Processor:
    Receive Solutions from Servers
      Distribute Solutions for Diversification
      Distribute Solutions for Intensification
    Report Best overall Solution
Servers (Slaves):
    G=Generation Size; PS=Population Size
    P=Total Processors; $P_i$ = Processor i
      For Each Processor $P_i$ (SLAVE):
        Read Circuit
        Initialize PS/P population of solutions
        For G/P Generations
          Crossover;Mutate;Select;.
          Produce New Solutions.
        End For
        Initialize Parameter of TS
        Improve Initial Solutions using TS
        Report Solutions Back to Master
      End For
    Invoke Diversification or Intensification Phase.
    Report solution to Master.
```

Figure 6.8: A distributed GA-TS heuristic

Distributed GA-TS Heuristic										
Blks	Prim2 Circuit					Ind2 Circuit				
	SEQ		DIS		Xs	SEQ		DIS		Xs
	C	T	C	T		C	T	C	T	
2	161	632	161	211	3	273	3422	271	1721	2
4	376	495	366	126	4	943	5087	946	1681	3
6	501	563	501	177	3	1322	6600	1322	2221	3

Table 6.3: GA-TS heuristic

6.4 Summary

In this chapter, we introduced a network multiprocessing environment that was suitable to parallelize the circuit partitioning heuristics used in this dissertation. The main goals of speeding up the execution time and improving the efficiency of the existing heuristics was established by careful design of the parallel heuristics. Results obtained in Section 6.3 indicate that this environment is effective to parallelize the heuristics and consequently to speedup the execution times. The main importance of this work lies in the portability of these heuristics to many sites for experimentation. Another goal that has been achieved is increasing the processing power available to the execution of most heuristics developed thus far.

The main emphasis of this chapter and the previous chapters is on presenting methods for solving the circuit partitioning problems. The different advanced search techniques and hybrids developed so far, have used the circuit partitioning as a paradigm for the circuit layout. In Chapter 7, some of the developed heuristics will be used effectively to solve the circuit placement problem, which is more complex than circuit partitioning.

Chapter 7

Circuit Placement

As the module count on a chip grows, the quality and speed of automatic layout algorithms need to be reevaluated. One of the most critical problems encountered in the design of VLSI circuits is how to assign locations to circuit modules and to route the connections among them such that the ensuing area is minimized. Due to the complexity of this problem, it is partitioned into two consecutive stages. The first deals with assigning locations to individual modules and is commonly referred to as the *Placement* problem. The second involves routing of the connections among the already positioned modules. The quality of the routing obtained at the second stage depends critically on the placement output of the first stage. Hence, the goal of a good placement techniques is to position the cells such that the ensuing area is minimized, while the wire lengths are subject to critical length constraints.

In this chapter, a novel automatic placement method that is suitable for large scale and highly interconnected systems is developed. The method is applied to application-specific integrated circuits (ASIC) placement in general. A *module-net-point* (MNP) model [Song92] is used to improve the estimation of the wire length, especially for nets connected to a large number of modules. The initial placement is obtained by a combination of a *global placement method* (optimizing a quadratic objective function) and a *circuit partitioning scheme.* A Tabu Search heuristic is then adapted to improve the placement quality.

7.1 General Placement

Module placement is an NP-hard problem and, therefore, cannot be solved exactly in polynomial time [**?**]. Trying to get an exact solution by evaluating every possible placement to determine the best one would take time proportional to the factorial of the number of modules. Consequently, it is impossible to use this method for circuits with any reasonable number of modules. To search through a large number of candidate placement configurations efficiently, a heuristic algorithm must be used.

7.1.1 Classification

Placement algorithms can be divided into two major classes: *constructive placement* and *iterative improvement.* In constructive placement, a method is used to build up placement from scratch; in iterative improvement, algorithms start with an initial placement and repeatedly modify it, in search of a cost reduction. If a modification results in a reduction in cost, the modification is accepted, otherwise it is rejected.

7.1.1.1 Constructive Based Algorithms

The constructive algorithms are divided into the following classes: *random placement*, *cluster growth, partitioning based algorithms* [Laut79, Suar88], *global tech-*

niques (analytical algorithms) [Otte82, Blan85, Sha85], and *branch and bound techniques* [Kozm88].

Cluster Growth

Cluster growth placement is a bottom-up method that operates by selecting components and adding them to a partial placement. These methods are differentiated from other placement methods in that cluster growth chooses and places the components independently. The first step in cluster growth is to determine a seed placement. Next, unplaced components are sequentially selected and placed in relation to those components already placed. This process continues until all components are placed. These algorithms are easy to implement but produce poor results because decisions must be made with incomplete information.

Random Placement

Random constructive placement is a degenerate form of cluster growth. It reduces to randomly choosing an unplaced component and placing it in a randomly chosen position. While this method is fast and easy to implement, it utilizes no circuit specific information.

Partitioning Based Placement

Algorithms based on partitioning components into two or more parts while reserving space for the components during the process are widely used in modern layout systems [Breu77]. This top-down approach tends to avoid wiring congestion in the center of the carrier. These algorithms differ from cluster growth in that the former considers all interconnections in parallel and then move the components in steps by partitioning the components into specific areas of the placement surface. The bi-partitioning or min-cut algorithms, divide the components into two sets such that the number of weighted connections is minimized between the sets and the area of the components is approximately equal. This process is repeated until each partition contains only one component. Partitioning based algorithms make use of the interconnection information at a global level and defer local considerations until late in the process. Although these procedures produce good results, they tend to be computationally expensive. They also tend to have difficulty when components are constrained to fixed positions and when natural partitions of the circuit do not correspond to the specified partitioning sequences.

Global Placement

While all techniques keep global notions of better or worse placements, the global techniques are distinguished by moving all of the components simultaneously along an n-dimensional gradient in the placement state space. This contrasts to clustering methods that consider each component sequentially, and to partitioning methods that first divide the components by partitioning and then deal with the components individually. Most of the global placement methods optimize a quadratic objective function [Chen84, Blan85, Fran86, **?**, Klei91]. Some of these methods apply partitioning to recursively generate smaller subproblems. The majority of these methods restrict the simultaneous optimization to the initial step. Maintaining simultaneousness over all optimization steps has been described in [Klei91].

7.1.1.2 Iterative Based Algorithms

Iterative placement manipulates a placement to produce an improved placement. Within one iteration, components are selected and moved to alternate locations. If the resulting configuration is better than the old, the new configuration is restored. This process continues until some stopping criterion is met. The stopping criterion might be relative or absolute improvement in the evaluation metric, or perhaps the time expended. Iterative placement techniques share the same structure and utilize three main functions: *selection*, *movement*, and *scoring*. The selection function chooses the components to move. The function may simply select components in a predefined sequence, or may involve intelligence to select those components which are placed poorly. Once the components are selected, the movement function determines new locations for those components. After the selected components are moved to new positions, a scoring function measures the quality of the new arrangement. The iterative placement improvement procedures known as pairwise interchange (**PI**) [Schw76], neighborhood interchange (**NI**) [Pate77], force-directed interchange (**FDI**) [Agul76], and force-directed pairwise relaxation (**FDPR**) [Schw76] have been effective and widely used in automatic placement of PC cards, PC boards, and LSI chips. The computational step that they all share consists of interchanging the locations of two modules and computing the cost of the new placement. The new placement is retained if its cost is lower than the cost of the old placement. Each algorithm has a different technique by which the pair of modules to be interchanged is selected.

In **PI** all $(n(n-1)/2)$ possible pairs of modules are trial interchanged in one cycle. In **NI**, modules in the neighborhood of the primary module (the module which initiates the trial interchange) are trial interchanged with the primary module. Each module is selected as a primary module in one cycle. In **FDI**, a primary module is selected and its force vector is computed. The primary module is trial interchanged with its three neighbors in the direction of its force vector. A cycle consists of selecting each module as a primary module. In **FDPR**, a primary module is selected and its force vector is computed, which defines its zero force target location. The primary module is trial interchanged with a secondary module in the ϵ-neighborhood of its target location, if the target location of that secondary module is in the ϵ-neighborhood of the primary module. Again, a cycle consists of selecting each module as a primary module.

Most of the computation time in the interchange method is taken up by the common step of these procedures which consists of computing the incremental change in the cost of the new placement resulting from the interchange [**?**]. These algorithms are characterized by accepting trial placements only if the score does not increase. This characteristic may cause the algorithm to get "stuck" in a local minimum rather than finding the global minimum. Recently a randomized version based on Simulated Annealing addressed this problem, and is considered to be widely used and successful [Sech86].

Simulated Annealing for Placement and Routing

Kirkpatrick [**?**] proposed the use of SA for the placement and routing of gate-arrays. The TimberWolf package was developed by Sechen *et al* [Sech86] for the placement of gate-arrays, standard-cells, and macro/custom cells as well as the global routing of standard-cells. Several ways of implementing the basic SA strategy were implemented and satisfactory results were obtained when comparing the quality of the placement by TimberWolf with placements obtained by other programs as well as manual placement.

According to Sechen *et al* [Sech86], a standard-cell placement program places standard-cells into rows and/or columns in addition to allowing user-specified macro blocks and pads. The program was interfaced to the CIPAR standard-cell placement package developed by American Micro-Systems. For larger circuits (800 to 1500 cells), TimberWolf reduced total wire length from 45% to 57% in comparison to CIPAR alone. Furthermore, final chip areas were reduced by at least 30% as a result of the improved placement. For a circuit of 1000 cells, TimberWolf reduced the final chip area by 31% in comparison to CIPAR and by 21% over another commercially available standard-cell placement program [Sech86].

Other algorithms that deserve mentioning are the parallel implementations of Simulated Annealing for chip placement [Rose88, Dare87, Krav87, Caso87].

7.1.2 Placement Cost Functions

Each of the preceding placement methods depends on the cost function employed in order to measure the acceptability of a current placement. Since the two fold goal of cell placement is to minimize the placement area while ensuring the routability of the layout, cost functions have examined various criteria such as *estimated wire length* and *cell congestion*.

7.1.2.1 Wire Length Estimation

To make a good estimate of the wire length, one should consider the way in which routing is actually done by routing tools. Almost all automatic routing tools use

Manhattan geometry; that is, only horizontal and vertical lines are used to connect any two points. Further, two layers are used; only horizontal lines are allowed in one layer and only vertical lines in the other. The shortest route for connecting a set of pins together is a *Steiner tree* (Figure 7.1a). In this method, a wire can branch at any point along its length. This method is usually not used by routers, because of the complexity of computing both the optimum branching point, and the resulting optimum route from the branching point to the pins. Instead minimum spanning tree connections and chain connections are the most commonly used connection techniques. *Minimal spanning tree* (Figure 7.1b) connections allow branching only at the pin locations. Hence the pins are connected in the form of the minimal spanning tree of a graph. *Chain connections* (Figure 7.1c) do not allow any branching at all. Each pin is simply connected to the next one in the form of a chain. These connections are even simpler to implement than spanning tree connections, but they result in slightly longer interconnects. *Source-to-sink connections* (Figure 7.1d) where the output of a module is connected to all the inputs by separate wires, are the simplest to implement. They, however, result in excessive interconnect length and significant wiring congestion. Hence, this type of connection is seldom used.

7.1.2.2 Bounding Box

An efficient and commonly used method to estimate the wire length is the *semi-perimeter method* [Sech86]. The wire length in this method is approximated by half the perimeter of the smallest bounding rectangle enclosing all the pins (Figure 7.2a). For Manhattan wiring, this method gives the exact wire length for all two-terminal and three-terminal nets, provided the routing does not overshoot the bounding

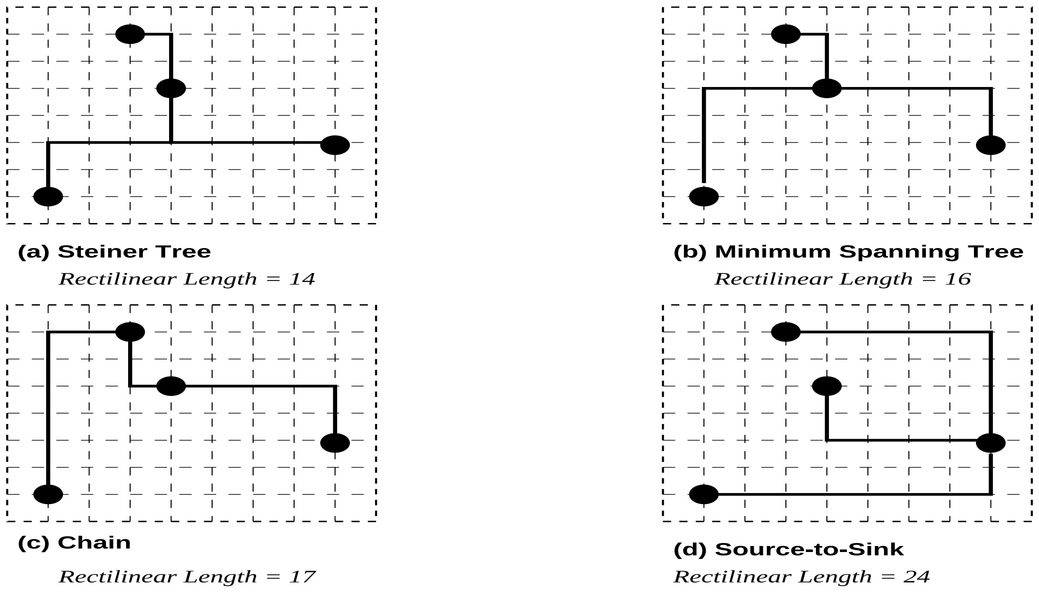


Figure 7.1: Interconnection topologies

rectangle. For nets with more pins and more zigzag connections, the semi-perimeter wire length will generally be less than the actual wire length. Moreover, this method provides the best estimate for the most efficient wiring scheme, the Steiner tree. The error will be larger for minimal spanning trees and still larger for chain connections. In practical circuits, however, two and three terminal nets are most common. Thus, the semi-perimeter wire length is considered to be a good estimate.

7.1.2.3 Cell Congestion

One way to measure cell congestion is to calculate the number of nets that connect separate partitions of the set of cells. The goal is then to minimize the number of nets cut by a line separating the partitions. Figure 7.2b graphically shows a high and low-cost configuration for a small example circuit. Some placement algorithms try to minimize the total wire length, or the congestion, or some combination of both.

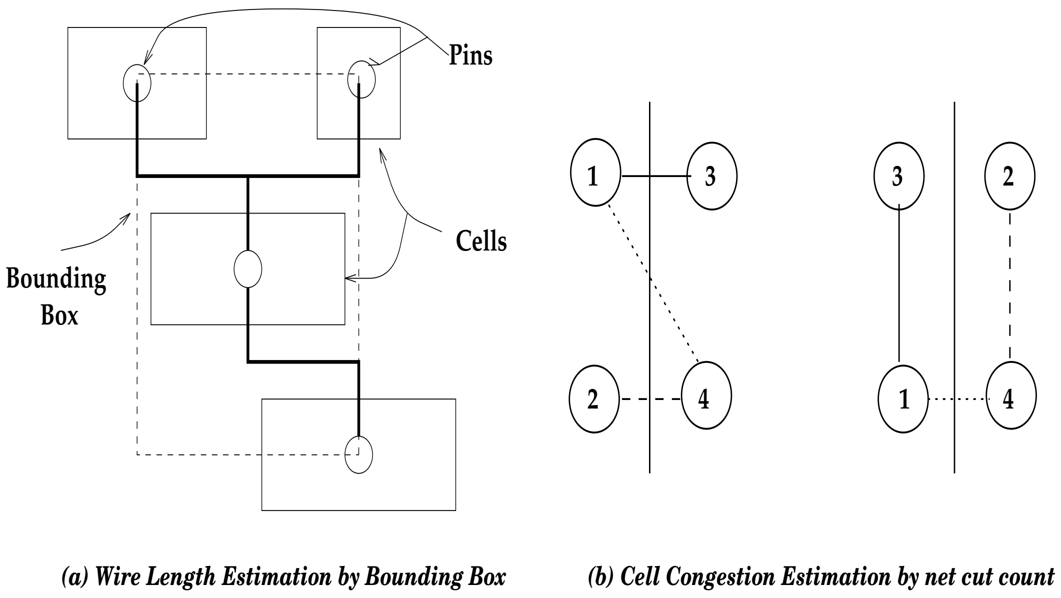

Pins
Cells
Bounding Box
1
2
3
4
3
2
1
4
(a) Wire Length Estimation by Bounding Box
(b) Cell Congestion Estimation by net cut count

Figure 7.2: Approximate wire estimation and congestion

7.1.2.4 Other Cost Measures

Placement algorithms for standard-cell placement usually employ the following cost function consisting of three parts [Sech88a]:

1. Estimate of the wire length of all nets, half-perimeter (W).
2. Overshoot or undershoot of each row over the desired row (R).
3. Area overlap between cell in the same row (O).

$$F = (\alpha \times W) \ + \ (\beta \times R) \ + (\gamma \times O)$$

A detailed discussion of this cost measure will be explained in Section 7.4.2.

7.2 Constructive Phase

7.2.1 A Quadratic Cost Function

For the initial placement problem, a model is needed to estimate the net interconnection length between modules. This model usually provides a quadratic cost metric which when minimized localizes the gates. Figure 7.3 shows some of the metrics used for initial placement. The most common used metric is that formed from the sum of the squared vector distances between gates, modeled as points [Blan85] (Figure 7.3a). This model is not always acceptable (although it provides a quadratic form appropriate for direct minimization) because it tends to overestimate the actual lengths. Figure 7.3b presents the nets-as-points model which has been proposed by [Pill88, Zhan89]. Direct minimization of the generated cost function yields optimal net-point locations which localize the gates. A new *module-net-point* (MNP) model is proposed by [Song92]. This model simultaneously considers the modules and nets as points. The net length is the summation of distances from the centralized net-point to the modules connected to the net, as shown in Figure 7.3c. This model is more realistic in estimating the actual wire lengths than previous models, especially for the multi-pin nets which occur in ASICs.

Let there be M modules located at $(x_i, y_i),\ i = 1, 2, \cdots, M$. Let there be N nets located at $(u_j, v_j),\ j = 1, 2, \cdots, N$.

In practical chip layout, the I/O pads can be considered as fixed modules. Let there be a total of F fixed modules located at $c_i,\ i = 1, 2, \cdots, F$. Let a_{ij} denote the connectivity between module i and net j. Let a'_{ij} denote the connectivity between

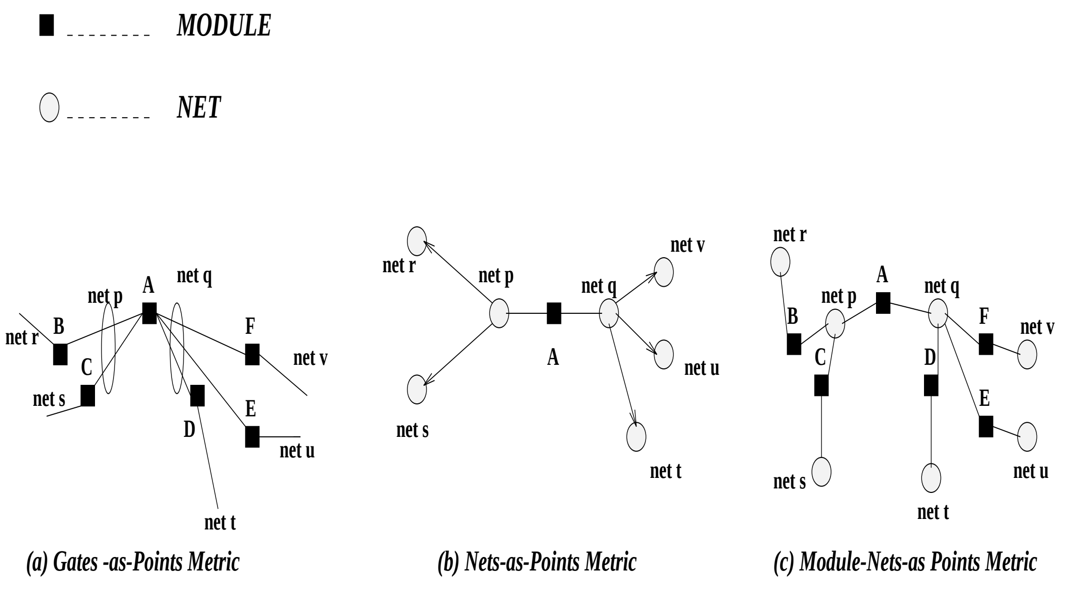

MODULE
NET
net r
B
C
net s
net p
A
net q
F
net v
D
E
net u
net t
(a) Gates -as-Points Metric
(b) Nets-as-Points Metric
(c) Module-Nets-as Points Metric

Figure 7.3: Wire estimation for global placement

fixed module i and net j. Our objective function f_x (f_y is similar) is as following,

$$f_x = \frac{1}{2}\sum_{i=1}^{M}\sum_{j=1}^{N} a_{ij}(x_i - u_j)^2 + \frac{1}{2}\sum_{i=1}^{F}\sum_{j=1}^{N} a'_{ij}(c_i - u_j)^2. \tag{7.1}$$

The matrix form of the quadratic expression (7.1) is:

$$\begin{aligned} Min \quad f_x &= \frac{1}{2}\mathbf{z}^T\mathbf{B}\mathbf{z} + \mathbf{c}^T\mathbf{z} + \mathbf{d} \\ s.t \quad \mathbf{z}^T\mathbf{z} &= 1 \end{aligned} \tag{7.2}$$

Since the matrix $\mathbf{B}$ is symmetric, sparse and positive definite, the solution can be found by a preconditioned conjugate gradient technique, where the preconditioning is done by incomplete factorization. The minimization can also be found by solving for the vector $\mathbf{z}$ at which the gradient of f_x is zero, $\bigtriangledown f_x = 0$. Minimization can be performed by applying the last formula to Equation (7.2) yielding the following linear system of equations to be solved for $\mathbf{z}$.

$$(\mathbf{B} + \lambda \mathbf{I})\mathbf{z} = \mathbf{c}. \tag{7.3}$$

The parameter λ ($\lambda > 0$) is added to make Equation (7.3) more numerically stable without effecting solution quality. The Harwell routine MA31 [Harw79] is used to solve Equation (7.3).

7.2.2 Generating a Legal Placement

Solving the quadratic optimization problem in the previous section produces an optimal continuous placement. The main drawback is that the solution does not put modules on slots (modules are confined to the center of the region). Therefore a mapping heuristic should be used to transform the position of some modules to legal positions on the chip. Also, since optimality is lost when mapping from a continuous placement to an integer grid, downhill local pairwise swaps are performed to regain some of the lost optimality. Figure 7.4 shows different approaches for generating legal placements. Figure 7.4a shows the position of modules after the initial continuous placement. Attempting to move modules to slots that are close to their locations could generate an overlapping placement. Figure 7.4b shows a simple snap-to-grid scheme that is suggested in [Zhan89]. This scheme does not reserve the relative relation of module position in the X-axis, and is only suitable for the case where the number of slots $m \times n$ is close to the number of modules M. Another method suggested by [Song92] strives to uniformly place the modules on the chip so that the routing ultimately can be performed easily. Figure 7.4c illustrates the procedure for uniform mapping. The main disadvantage using this

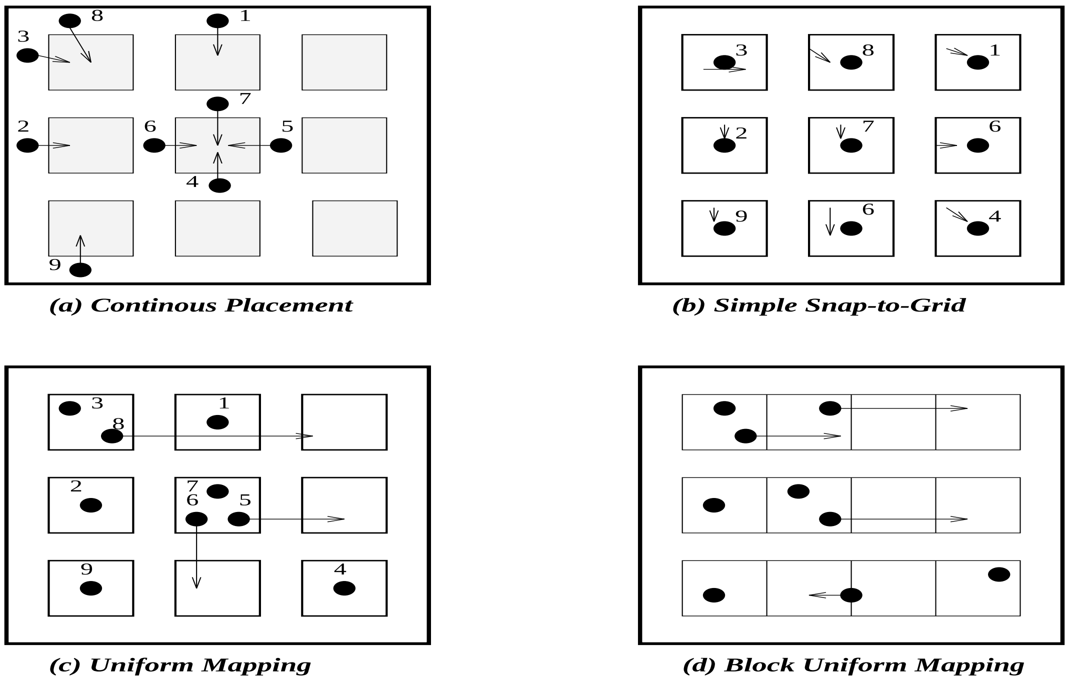


Figure 7.4: The mapping to legal placement

scheme is that in the mapping from the continuous value coordinates into discrete coordinates, overlaps may occur; i.e., some modules have the same coordinates. One way of overcoming this problem is to resort to a heuristic to remove all the overlaps while the total *half perimeter wire length* (HPWL) increases as little as possible (decreases as much as possible). This is achieved using a *neighborhood empty-slot move* (NESM) heuristic.

7.3 A Partitioning Placement Heuristic

In this section, we introduce a placement heuristic that utilizes a constructive phase followed by one or more iterative improvement phases to obtain good solutions. As shown in Figure 7.5, the heuristic tends to minimize the total wire length and congestion of cells at the same time. A global placement method discussed in Section 7.2 is used to generate a good initial placement. The solution does not put the modules on slots, but leads to solutions where modules are confined to the center of the region. This leads to cell congestion and overlapping. As shown in Section 7.1.2.3, partitioning is used to reduce the congestion. The partitioning algorithm based on GRASP and Tabu Search Algorithm introduced in Section 4.2.2 is used to partition the components into rows. Here, the IO pads are fixed to a certain partition. Top IO pads are fixed at block 0, and bottom IO pads are fixed at block with highest value (i.e, number of rows determined in the design), whereas other side IO pads are distributed among other blocks and fixed. Since the number of nets cut between blocks (rows) is minimized, it can be guaranteed that most of the modules within rows are placed in their correct positions. But still their exists overlap between modules within a row. The overlap of modules is dealt with by a snap-to-grid scheme, this scheme has only to deal with module overlap within a row and not between rows. The combined solution of the global approach and circuit partitioning leads to good solutions that are improved using the local search heuristic based on Tabu Search. The next section introduces each phase of the heuristic and explains the details.

7.4 Iterative Improvement Phase

Since optimality is lost when mapping from a continuous placement to the legal placement, iterative improvements have to be made to generate a better placement. A detailed examination of different force-directed heuristics is described with experimental results in [Goto86]. These heuristics are greedy algorithms which stop

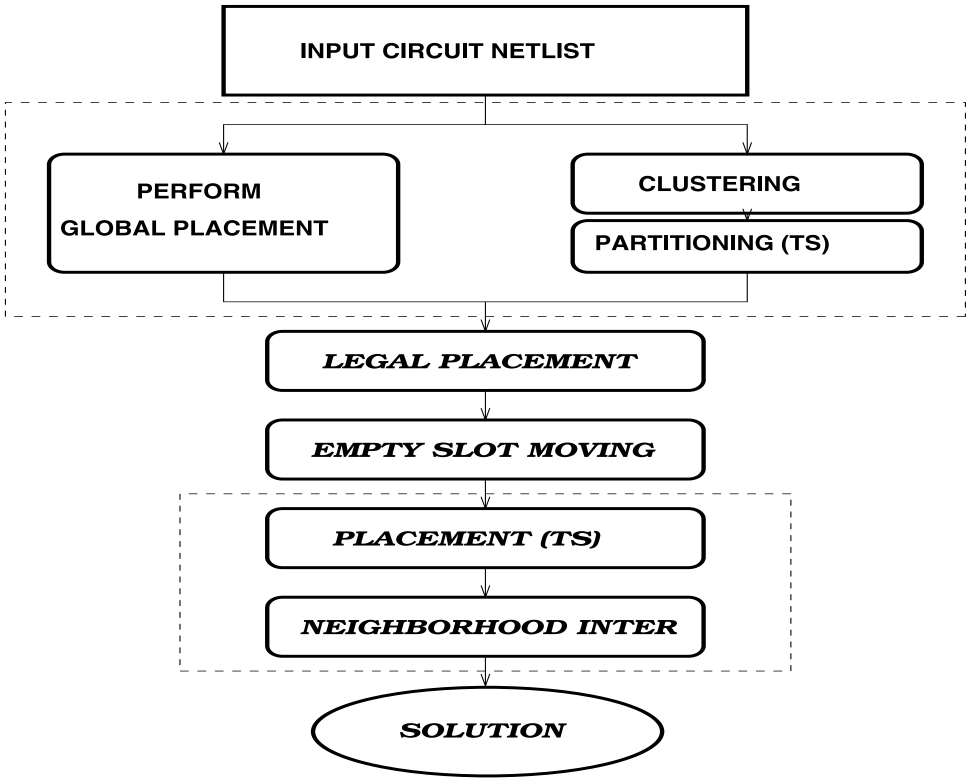


Figure 7.5: The GPPSC heuristic

at the local optimum when no local improving moves are possible. The chief limitation is that the local optimum may not be a global optimum. *Tabu Search* [**?**, Glov90] guides such a heuristic to continue exploration without getting stuck at a local optimum by an absence of improving moves, and without falling back into a local optimum from which it previously emerged. The heuristic using *Tabu Search* technique to iteratively improve the placement quality is presented here. First, the simple *Tabu Search* technique is briefly introduced.

7.4.1 A Tabu Method for Placement

The majority of this section follows closely work by [Song92]. The Tabu Search technique is used to guide the neighborhood interchange heuristic in the iterative placement scheme. We first define the neighborhood of the module.

Definition 1 *A δ-neighborhood $N_\delta(m_i)$ of a module m_i is the set of modules m_j whose X- and Y-coordinate are distance δ apart from m_i, i.e.,*

$$N_\delta(m_i) = \{m_j : |x_j - x_i| \leq \delta \text{ and } |y_j - y_i| \leq \delta,\ j = 1, 2, \cdots, M,\ i \neq j\},$$

where $(x_i, y_i), (x_j, y_j)$ are the coordinates of modules m_i, m_j, and M is the number of modules.

We define the neighborhood $N(s)$ of a solution s as the set of all solutions s' that can be obtained from s by interchanging a pair of modules (m_i, m_j) where $m_j \in N_\delta(m_i)$ for $i = 1, 2, \cdots, M$. The neighborhood interchange technique is to find the best solution in the neighborhood $N(s)$, which exhaustively examines all target modules $m_j \in N_\delta(m_i)$ for module $m_i, i = 1, 2 \cdots, M$. The best pair of

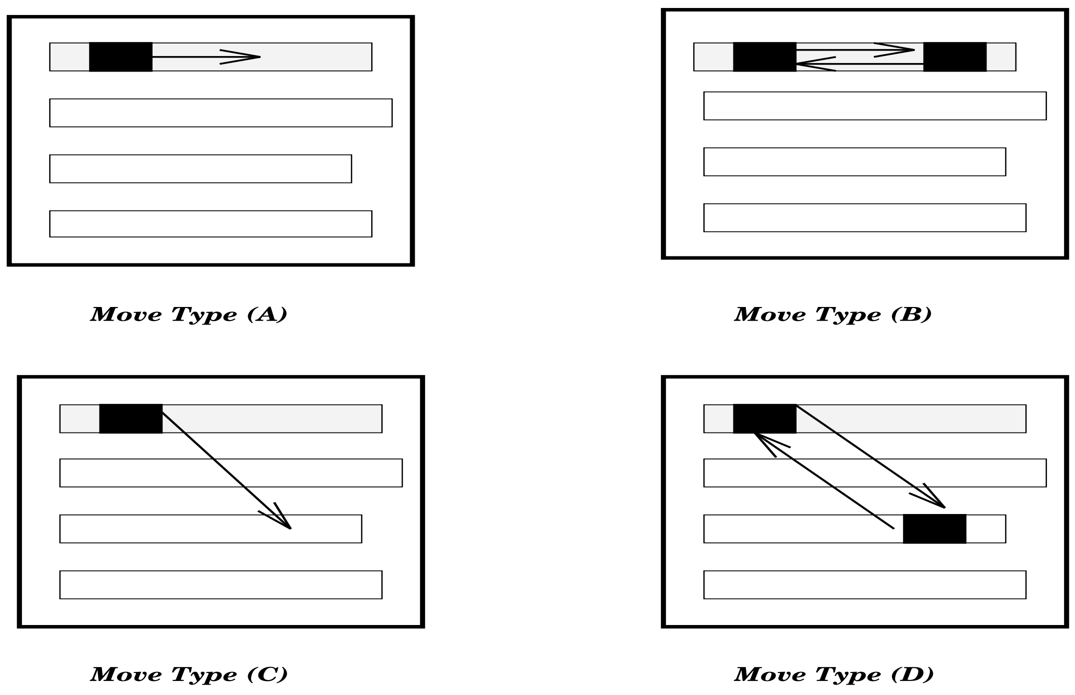


Figure 7.6: Type of possible moves

modules (m_i, m_j) associated with the maximum gain is selected for the interchange if the gain is positive. The interchange procedure is repeated until the gain is negative.

A Tabu method for placement is described in Figure 7.7 which is a combination of the *Tabu Search* technique and the neighborhood interchange heuristic. The new solution s' is obtained from s by the following procedure. First, module m_i is randomly selected. Then all target modules $m_j \in N_\delta(m_i)$ are exhaustively examined. The gains are estimated from the difference of the objective function $f(s)$ before and after the interchange. The best target module m_j associating with the maximum gain is selected for the interchange with module m_i if the pair of modules (m_i, m_j) is not in Tabu list T. The new solution s' is obtained by interchanging a pair of modules (m_i, m_j), even if the gain is *negative*. In determining whether a move is

Tabu or not, only one Tabu list T containing the $|T|$ last module pair interchanges is used. Such a list is sufficient to prevent cycling. The Tabu list T is treated as a circular list. The addition of module pair thus removes the module pair recorded in its position $|T|$ interchanges ago. Note that the key step in the Tabu method for placement is to find a *good solution* in the neighborhood $N(s)$ rather than to find the *best solution* in the neighborhood $N(s)$. Given the parameter δ and module m_i, the complexity for examining the modules $m_j \in N_\delta(m_i)$ is constant. But given the parameter δ, the complexity for examining the modules $m_j \in N_\delta(m_i)$ for all modules $m_i, i = 1, 2, \cdots, M$ is $O(M)$. Therefore, the complexity for finding the best solution in the neighborhood $N(s)$ is $O(M)$. The complexity of the heuristic to generate module pair (m_i, m_j) is constant. Although it needs more Tabu Search steps to obtain a good result, the total computation time is still much less than the case of finding the best solution. On the other hand, the random selection of module m_i contributes to diversify the search, hence the possibility of obtaining a better result increases. A definition of the objective function $f(s)$ for circuit placement will be discussed in the following sections.

7.4.2 The Objective Function

The Tabu Search heuristic used for the placement consists of two stages. Overlap is allowed in the first stage and prohibited in the second stage. The main reason of allowing overlap in the first stage is to increase the solution space being searched. Tabu Search uses two different objective functions for the search stages.

7.4.2.1 First Search Stage

The first stage minimizes the total half-perimeter wire length while making overlap as small as possible and makes row length uniformly distributed. In this stage, the objective function is chosen as:

$$f(x) = c_l(x) + c_o(x) + c_r(x). \tag{7.4}$$

HPWL

The cost $c_l(x)$ is the total half-perimeter wire length (HPWL) given by:

$$c_l(x) = \sum_{i=1}^{N} (H_i + V_i), \tag{7.5}$$

where N is the number of nets, H_i and V_i are the span of the net i in the horizontal and vertical direction separately.

Overlap Penalty

The cost $c_o(x)$ is the overlap penalty function, and given by:

$$c_o(x) = p_o \sum_{i<j} O(i, j), \tag{7.6}$$

where p_o is a penalty parameter. The function $O(i, j)$ returns the total amount of overlap area between cells i and j.

Certainly, by checking every other cell on the same row as cell i it can be determined which of these cells overlap with cell i. However, the complexity is $O(M_i)$, where M_i is the number of cells in the row. The time spent doing overlap computation can be substantial. Let W_{max} denote the maximum cell width. Another way to compute the cost $c_2(x)$ is to search toward W_{max} left away from cell i and toward W_{max} right away from cell i, all other cells which overlap with cell i can be found. As long as the ratio between the maximum and minimum cell width is not too large, this is an efficient method.

The overlap penalty parameter is an empirical parameter. If p_o is too large, the Tabu method will be primarily concerned with the minimization of the overlap penalty function. That is, relatively little attention will be paid to the minimization of the HPWL. If p_o is too small, the Tabu method will concentrate on the minimization of the HPWL. That is, relatively little attention will be paid to the elimination of the cell overlaps. At the end of the first Tabu Search stage, a large amount of the cell overlaps still exists. The HPWL will increase when all cell overlaps are removed by the overlap removing heuristic. The experimental results on five test circuits show that the lower value of HPWL is obtained when p_o is in the range $(0.1, 0.5)$.

Row Length Penalty

The cost $c_r(x)$ is the row length penalty function. It is given by

$$c_r(x) = p_r \sum_{i=1}^{R} |L_{ai} - L_{di}|, \tag{7.7}$$

here p_r is a row penalty parameter, R is the number of rows, L_{ai} and L_{di} are the actual and desired row length for row i. $p_r = 5$ is approximately the smallest value which would yield uniform row lengths without placing excessively emphasis on $c_r(x)$ in the objective function $f(x)$ [Sech88a].

7.4.2.2 Second Search Stage

After the termination of the first search stage, a simple heuristic is used to remove all the overlap among cells by shifting them. At this stage, the placement has no cell overlap, and the row length are changed slightly. The objective here is to minimize the HPWL only, $f(x) = c_l(x)$. The heuristic avoids causing overlap in this stage by choosing cells having the same width and are close to each other. To further refine the search, a simple neighborhood interchange heuristic is used, since Tabu Search does not exhaustively search all possible cells in the neighborhood $N(s)$.

```
Input:           initial placement $s$
                 nbmax: maximum number of iterations
                 $|T|$: size of Tabu list $T$
Initialization:  nbiter:= 0;
                 bestsolution:= s; bestvalue:=$f(s)$; bestiter:=0;
while            nbiter-bestiter $<$ nbmax  do
                 (a) nbiter := nbiter +1 ;
                 (b) Generate the move $s' \in N(s)$ and $s$ is not in Tabu list $T$;
                     —randomly select a module $m_i$
                     —find a best target module $m_j \in N_\delta(m_i)$ for interchange,
                       and module pair $(m_i, m_j)$ is not in Tabu list T
                 (c) if $f(s') <$ bestvalue then
                           bestvalue:= $f(s')$;bestsol:= $s'$; bestiter:= nbiter;
                 (d) Add $(m_i, m_j)$ to Tabu list $T$, remove oldest module pair;
                 (e) $s := s'$;
EndWhile
output:          bestsolution is the best placement found so far
                 and best value is the minimum half perimeter wire length
```

Figure 7.7: A Tabu Search technique for circuit placement

7.5 Experimental Results

The method for the standard-cell placement has been implemented in the C language in the placement program **GPPSC**. The program **GPPSC** has been tested on five examples (Table 7.1). The first three circuits, fnn4, fnn8 and fnn32 are neural network circuits which are highly interconnected circuits. Circuits Prim1 and Prim2 are taken from the MCNC standard-cell test benchmarks [Robe87]. These circuits vary in size from 140 to 3,014 cells and 119 to 3,136 nets. The measure that has been used here is the half perimeter bounding box that is used by most researchers. Minimizing the HPWL in place guarantees the reduction of the area occupied by the components [Sech88a].

Placement Results							
Circuit	**Rows**	**SA TimberWolf**		**TSSC**		**GPPSC**	
		HPWL	Time	HPWL	Time	**HPWL**	**Time**
Ckt1	5	0.0835	171.5	0.0869	6.2	**0.0811**	**8.5**
Ckt2	9	0.3051	580.7	0.3230	105.6	**0.2954**	**123.2**
Ckt3	19	1.6773	3218.8	1.8052	253.5	**1.5344**	**254.9**
Prim1	16	0.9193	1018.3	0.9990	341.7	**0.9191**	**365.7**
Prim2	26	3.9518	6574.0	4.4121	2009.5	**3.9429**	**1992.1**

Table 7.1: Placement results based on HPWL

Table 7.1 presents results based on the developed heuristic GPPSC. The results are compared to those obtained by TimberWolf Version 5.1 for standard-cell placement, and Tabu Search heuristic TSSC [Song92] that uses the global constructive phase only for its initial solution. Results indicate that the solutions obtained by the novel technique improve results by 5-10% over those obtained by TSSC, and by 2-3% over those obtained by TimberWolfSC. Even though the improvement in results obtained by GPPSC are not significant over those obtained by Timber-

WolfSC, most techniques used recently [Klei91] produced results that were inferior to those obtained using TimberWolfSC by a margin of 5%. On the other hand, the computation time of GPPSC are on average 55% faster than TimberWolfSC.

7.6 Summary

This chapter introduced a novel technique for Standard Cell Placement. The initial placement was obtained using a combined global optimization algorithm and a partitioning based heuristic. Results obtained indicate that this initial strategy is successful in placing most modules in their appropriate position. A Tabu Search placement heuristic was used to further improve the initial placement. Since the global placement method was successful to place most of the cells in the "right position", the local search algorithm converged quickly to good solutions.

The asymptotic analysis of many heuristics such as Tabu Search and clustering heuristics is an important measure of the complexity of these heuristics. The next chapter presents the complexity analysis of many heuristics developed in this dissertation, and shows the time and space requirements for solving large instances of combinatorial problems in circuit layout.

Chapter 8

Complexity Analysis

The main characteristics associated with "good" heuristic programming methods are the simplicity of the approach, the simplicity of the computational procedure or operations employed and the low order polynomial growth in computational time required as a function of problem size. The evaluation of the performance of a heuristic method can be carried out either by comparing its performance against those achieved in the past, or by solving massive problems with known solutions, or by determining the computational complexity of the heuristic program.

Computational complexity provides a wealth of exciting results of both theoretical and practical nature. Once an algorithm is developed for solving a problem and decided to be correct, an important step is to determine how much in the way of resources, such as time or space, the algorithm will require. An algorithm that solves a problem but requires enormous computation time is hardly of any use. Likewise, an algorithm that requires a vast amount of main memory is not (currently) useful. So computational complexity seeks to develop general results about the intrinsic difficulty of solving problems where available information is partial or approximate and to apply these results to specific problems. This allows one to determine what is meant by an optimal algorithm in many practical situations, and offers a variety of interesting and sometimes surprising theoretical results.

The previous chapters presented heuristic solutions to the problems of circuit partitioning and placement. In this chapter, we seek to determine the computational performance of the heuristic programs in terms of the solution times (number of operations) required as a function of problem size. Section 8.1 introduces the main notation and definitions used throughout this chapter. A worst case complexity analysis of the Tabu Search heuristic for circuit partitioning is presented in Section 8.2. The computational complexity of the two clustering heuristics (described in Section 5.2) are introduced in Section 8.3.

8.1 Notations and Definitions

This section introduces the notation that is used for computing the complexity of the partitioning and clustering heuristics developed in previous chapters. For simplicity, we borrow some of the notation used in [Sanc89]. The *network* refers to the original circuit that is read initially. The *clustered network* will refer to the circuit after clustering. First the notation associated with the main network is introduced followed by notation used for the clustered network.

8.1.1 Notation Used For Main Network

- b will denote the number of blocks “partitions”.
- c the number of cells in the network.

- n the number of nets in the network.
- N_C, will denote the set of nets incident on cell C,
- n_c will denote the size of N_C,
- C_N will denote the set of cells incident on net N,
- c_n will denote the size of C_N,
- p will denote the maximum number of nets on any cell,
- q will denote the maximum number of cells on any net in the network,
- m will denote the total number of connection points in the network:
  - The variable m may be regarded as a measure of the size of the network.

$$m = \sum_{\forall C} n_c = \sum_{\forall N} c_n$$

- l will denote the concept of level gain.
  - Computing higher level gains enables the algorithm to better distinguish between cells whose first level gains are the same.

8.1.2 Notation Used For Clustered Network

- c' the number of cells in the clustered network.
- n' the number of nets in the clustered network.
- N_{cl} will denote the number of Clusters formed.

- C_{sz} will denote the size of a cluster.
- N'_C will denote the set of nets incident on cell C' in the clustered network,
- n'_c will denote the size of N'_C in the clustered network,
- C'_N will denote the set of cells on net N',
- c'_n will denote the size of C'_N,
- p' will denote the maximum number of nets on any cell in clustered network,
- q' will denote the maximum number of cells on any net in clustered network,
- m' will denote the total number of connection points in the clustered network,
- T_n will denote the net types in the clustered network,
- T_c will denote the cell types in the clustered network,
- C_{cm} will denote the common modules for cell C,
- C_{mcm} will denote the maximum common modules for any cell,

8.1.3 Definitions

8.1.3.1 Free and Locked Cells:

A cell is labeled *free* if it has not yet been moved during the pass; otherwise, it is labeled *locked*.

8.1.3.2 Critical Net:

A net is said to be *critical* if there exists a cell on it which if moved would change the net's cut-state.

8.1.3.3 Network Parameters:

$$\phi_{Ai}(N) = |C|C \in A_i \text{ } and \text{ } C \in C_n and \text{ } C \text{ } is \text{ } free|$$

$$\lambda_{Ai}(N) = |C|C \in A_i \text{ } and \text{ } C \in C_n \text{ } and \text{ } C \text{ } is \text{ } locked|$$

So $\phi_{Ai}(N)$ is the number of free cells on the net N which are in the block A_i, while $\lambda_{Ai}(N)$ is the number of locked cells on the net N which are in the block A_i. For each A_i and each net N, define the binding number β:

$$\beta_{Ai}(N) = \begin{cases} \phi_{Ai}(N) & \text{if } \lambda_{Ai}(N) = 0 \\ \infty & \text{if } \lambda_{Ai}(N) > 0 \end{cases}$$

The binding number of a net with respect to a block of a partition indicates how tightly the net is bound to the block. If there are locked cells from a net in the block then the net will be bound to the block for the rest of the pass, since locked cells can no longer be moved during the pass.

8.1.3.4 Gain Computation:

The i^{th} level gain of C, $\gamma_i(C)$, is defined as

$$\begin{aligned} \gamma_i(C) &= |N \in N_C|\beta_{A1}(N) = i \text{ } and \text{ } \beta_{A2}(N) > 0| \\ &\quad -|N \in N_C|\beta_{A1}(N) > 0 \text{ } and \text{ } \beta_{A2}(N) = i - 1| \end{aligned}$$

Here A_1 and A_2 are the two blocks of the partition and cell C is assumed to be in block A_1. Note that the first level gain is the actual decrease in cut-set size which would result from moving cell C from A_1 to A_2; i.e., the first level gain corresponds to the regular gain concept used in other interchange methods [**?**, Fidu82].

For multiple-way partitioning the function β' is needed, and is defined as follows:

$$\beta'_{A_i}(N) = \sum_{j \neq i} \beta_{A_j}(N)$$

That is $\beta'_{A_i}(N)$ is the sum of all the binding numbers of net N with respect to all of the blocks of the partition except block A_i; it gives a measure of how tightly N is bound to the "side" of the partition "opposite" A_i.

The i^{th} level gain associated with moving cell C from block A_j to block A_k is given by:

$$\begin{aligned} \gamma_i(C) \;\; = \;\; & |N \in N_C|\beta'_{A_k}(N) = i \; and \; \beta_{A_k}(N) > 0| \\ & -|N \in N_C|\beta'_{A_j}(N) = i-1 \; and \; \beta_{A_j}(N) > 0| \end{aligned}$$

The rational behind the above definitions is that in order to remove a net from the cut-set, all cells on the net must end up in a single block.

8.1.3.5 Gain Maintenance:

During each iteration of the algorithm, the gains associated with the cells in the network will be constantly updated as cells are moved from one block to another. Each time a cell C is moved, the β and β' values for the nets connected to C will change, in turn causing changes to the gain vectors associated with C's neighbor cells. For Net N define:

- status(N) = *free* if none of its β values are ∞

- status(N) = *loose* if exactly one of its β values is ∞
- status(N) = *locked* if two or more of its β values are ∞

The status of a net can be easily updated each time one of its binding values changes. The values of β' can be computed easily as following:

- if N is free then
  - $\beta'_{A_k} = c_N - \phi_{A_k}(N)$
- else if N is loose then
  - $\beta'_{A_k} = c_N - \phi_{A_k}(N) - \lambda_{A_k}(N)$
- else if N is locked
  - $\beta'_{A_k} = \infty$
- else if $\beta_{A_k} \neq \infty$
  - $\beta'_{A_k} = \infty$

8.1.3.6 Balancing Criteria:

If no restriction is taken on the size of the final partitions, an empty cut-set is achieved by moving all of the cells to one block of the partition. So the concept of mincut partitioning is meaningless unless a restriction is placed on the sizes of the blocks. The approach taken is to specify a *Tolerance* $TOL = cells/blocks * percent - tolerance$ that can satisfy the following:

$$cells/blocks - TOL \leq Block_i \leq cells/blocks + TOL$$

A cell move from $Block_i$ to $Block_j$ is allowed if it preserves the above relationship.

8.2 Complexity Analysis of Tabu Search

For many applications, the choice of the proper data structure is really the only major decision involved in the implementation. Once the choice has been made, only very simple algorithms are needed. For the same data, some data structures require more or less space than others; for the same operations on the data some data structures lead to more or less efficient algorithms than others. In summary the choice of algorithm and data structure is closely intertwined, and ways are sought for saving time and/or space by making this choice properly.

The Tabu Search algorithm is used as a meta-heuristic to guide the Sanchis multiple-way network partitioning algorithm to achieve better results. Most of the algorithms used in Sanchis [Sanc89] are used within the Tabu Search heuristic except for some modifications that involve the following:

- No concept of a Pass,
- No concept of free and locked cells,
- All nets are either free or loose at a certain move,
- ϕ is the only net parameter used to update the gains,
- New concept of Tabu List and Aspiration Level tables.

The data structures most affected by the k-way Tabu Search partitioning scheme are:

- Circuit Description data structures,

- Gain Vector structures,
- Network Parameter structure,
- Tabu List structure,
- Aspiration Table structure.

After the network is initialized and an initial partition is obtained, iterations are performed until no more improvements in cut-set size results. Network initialization consists of reading in a description of the network and initializing the list of cells incident on each net and the list of nets incident on each cell. A partition P is assumed to be an array indexed by cell numbers specifying which block each cell should be in. The starting partition is achieved using a random number generator. Initializing the partition consists of assigning each cell to its corresponding block and then initializing the gain values and gain structures for the partition.

8.2.0.7 Complexity of Tabu Search Circuit Partitioning

The following propositions show that the time complexity of the Tabu Search heuristic presented in Chapter 3 is $O(b^2 + qb(\log p))$ per iteration.

```
Partition-Network()
    (1) Call Input-Circuit
    (2) Obtain a random partition P
    (3) Call Init-Partition
    (4) Repeat
            (a) Call Do-Iteration()
    Until (No improvement in cutset size).
```

Proposition 8.1 *O(m) time is needed to read in the circuit description (construct the two data structures from the sequence of nets given as input).*

Proof: It is clear that O(m) time will suffice to do all of the above work in the routine **Input-Circuit** , provided that the number of (cell,pin) pairs in the input stream is O(m).

```
Input-Circuit()
  (1) For net = 1 ... n do
      (2) For each cell on net(N) do
          (3) Insert net N into the 'nlist' of cell C
          (4) Insert cell C into the 'mlist' of net N
      End For
  End For
```

Proposition 8.2 *The criticality of a net β'_{A_k} can be computed in constant time, $O(1)$ from the status of N and from the ϕ values for N, independent of b.*

Proof: It is not hard to check that the algorithm that computes β'_{A_k} is done in constant time given by the equations in section 8.1.3.3 and the procedure Beta-Prime().

Proposition 8.3 *The insertion and deletion of a gain node each requires $O(\log p)$ time.*

Proof: Since binary search has complexity of $O(\log p)$), and since its is used implementing the bucket list (with entries ranging from +p to -p), the time complexity of insert-gain is $O(\log p)$. The same applies for delete-gain.

```
Init-Partition()
  (1) Set all $\phi$ values and gain vectors to 0.
  (2) For each cell C
      – For all blocks
          - initialize gains for all free modules
      – End For
      – For each net N incident on C
```

```
        - adjust net parameters
      – End for
  – End For
  (3) For each net N
      – For each block $A_k$
          – For each cell C on net N
              - Call **Update-Gain**
          – End For
      – End For
  – End For
  (4) For each cell C
      - let $A_k$ be the block to which C belongs
      - For each block $A_i \neq A_k$
          - create gain node for C's gain in moving to $A_i$
          - insert gain node in gain list and bucket list
      - End For
  - End For
```

Proposition 8.4 *The routine Init-Partition() requires at most* $O(mb(\log p))$

Proof: Step (1) in **Init-Partition()** is performed in $O(nb) \leq O(mb)$, step (2) is performed in $O(cb)+O(m)$ since the second term is dominant then it is performed in $O(m)$, step (3) is performed in $O(mb)$ since the call to **Update-Gain()** requires constant time. Step (4) of Init-Partition() requires O(cb) time to insert a gain, since Proposition 8.3 states that insertion of a gain requires $O(\log p)$ then step (4) will require $O(cb(\log p))$ time. Hence, Init-Partition() requires at most $O(mb(\log b+p))$

Proposition 8.5 *The routine Can-Move() requires at most* $O(b^2)$

Proof: step (1a) of procedure **Can-Move()** is accomplished in constant time, O(1), since the bucket list is always sorted. Step (1b) which is a call to procedure **Valid-Move()** obviously takes constant time as seen in the procedure in previous section. According to the criteria chosen to update the Tabu list we have:

- criteria (1): store the module number,
- criteria (2): store the module number, the source block and the destination block.

```
Can-Move()
    (1) Do
        (a) Point to cell with highest gain,
        (b) if (Valid-Move())
                If (Move-Not-Tabu())
                Return (Pointer to cell)
        (c) Else
                Advance Pointer to next cell with highest gain,
        While (gain-ptr is valid)
```

If criteria (1) is used, a constant time to check whether a module is Tabu or not can be found in constant time, O(1), whereas if criteria (2) is used, the possible cell directions is given by the following formula:

$$\sum_{i=1}^{b-1} i = \frac{(b-1)\times(b-1+1)}{2} = \frac{b^2-b}{2}$$

so the time needed to test if a move is Tabu is $O(b^2)$. So the time complexity of **Can-Move** is constant, O(1), if criteria (1) is used, and $O(b^2)$ if criteria (2) is used.

Proposition 8.6 *The routine Make-Move() requires at most* $O(qb(\log p))$

Proof: Step (1) of **Make-Move()** requires constant time, O(1). Step (2a) is executed b times, and since each cell incident on N could have its gain updated, and from Proposition 8.3 the insertion and deletion of a gain requires $O(\log p)$ then one can say that step (2a) requires $O(Max(n_c)b(\log p))$ time. Step (2b) takes constant time. Step (2c) requires $O(n_c \log p)$. The overall complexity of Make-Move is $O(qb(\log p))$.

```
Do-Iteration(P)
    (1) Initialize TabuList, Aspiration Table
    (2) While (Can-Move & Stopping Criteria Not Valid)
        (a) Get Nextmove from gain structures
        (b) Call Make-Move to perform Nextmove
        (c) Call Update-TabuList, Update- Aspiration
        (d) Record best partition and best cut
    End While
    (3) Return(P)
```

Proposition 8.7 *The routine Do-Iteration() requires at most $O(b^2 + qb(\log p))$.*

Proof: Step (1) takes constant time. Step (2) takes constant time, $O(1)$, if Tabu Criteria (1) is used and $O(b^2)$ if Tabu Criteria (2) is used. Step (2a) takes constant time. The call to **Make-Move()** as has been described in Proposition 8.6 takes $O(qb(\log p))$. Step (2c) which involves updating the Tabu List and Aspiration Table takes constant time. Recording the best partition and best cut involves constant time since only one value of the partition array is updated. The overall complexity of **Do-Iteration()** is $O(b^2 + qb(\log p))$.

Theorem 8.1 *The multi-way network guided partition algorithm based on Tabu Search requires $O(b^2 + qb(\log p))$ time to complete a move.*

Proof: From Proposition 8.1 **Input-Circuit()** takes $O(m)$ time. The call to **Init-Partition()** which is performed once during the partitioning algorithm takes $O(mb(\log p))$ as had been presented in Proposition 8.4. The loop to perform an iteration is executed η times and varies from one circuit to another. As has been proved in Proposition 8.7 the body of the loop that calls **Do-Iteration()** has a complexity of $O(b^2 + qb(\log p))$ in the worst case. So the overall complexity of the multi-way partitioning algorithm requires $O((b^2 + qb(\log p))$ per iteration, and $O(\eta(b^2 + qb(\log p))$ for the program to end.

8.3 Complexity of Clustering Heuristics

In this section, we attempt to present the complexity of clustering based on (i) GRASP, (ii) Statistical circuit information presented in Chapter 5.

8.3.1 GRASP Based Clustering

The following propositions show that the time complexity of the GRASP based clustering heuristic is $O(N_{cl}(\log N_{cl} + p'))$ per move, and $O(m' N_{cl}(\log N_{cl} + p'))$ for creating complete clusters.

```
GRASP CLUSTER()
  (1) Input-Circuit()
  (2) Determine number of clusters $CL_n$
  (3) Create-Clusters()
  (4) Generate-Clustered-Network()
  (5) Assign-Clusters-Blocks()
  (6) Flatten-Clusters()
  (7) Make-Uniform-Partitions()
```

Proposition 8.8 *O(m) time is needed to read in the circuit description. Constant time is needed to determine number of clusters, O(1).*

Proof: Complexity of **Input-Circuit()** was proved previously in Proposition 8.1. The time needed to determine number of clusters is constant, O(1).

```
Create-Clusters()
  (1) Allocate all modules to a single block.
```

```
(2) Calculate Gain of all modules.
(3) Set MoveArray, GainArray empty.
(4) Set GainPointer to 0.
(5) While Can-Move()
    (a) Call Make-Move().
    (b) Add nextmove to MoveArray
    (c) Add newgain to GainArray
(9) End While
(10) Use Moves in MoveArray upto best ptr.
```

Proposition 8.9 *$O(mN_{cl}(\log N_{cl} + p)$ time is needed to create required clusters from the circuit description. The time per iteration (moving a single module to destination) is $O(N_{cl}(\log N_{cl} + p))$.*

Proof: Step (1) in **Create-Cluster()** requires constant time. Step (2) for calculating gain requires $O(N_{cl}(\log N_{cl} + p))$. This is similar to routine used previously in Proposition 8.4. Steps (3) and (4) take constant time. The most time is occupied by steps (5) and (6). Since a cell becomes locked after it is moved, the while loop is performed at most c times. Choosing the next move, including heap updates requires $O(N_{cl}(\log N_{cl}))$ time, hence all move selections during the execution of the while loop can be performed in $O(cN_{cl}(\log N_{cl})) \leq O(mN_{cl}(\log N_{cl}))$. Step (6) consists of calls to Make-Move, this routine is adapted from Sanchis [Sanc89], whose complexity is $O(lmb(\log b + pl))$ and since b in our case is N_{cl} and *l=1*, then the complexity of this call is $mN_{cl}(\log N_{cl} + p)$. Since steps (7) through (10) take constant time, then the dominant factor is $mN_{cl}(\log N_{cl} + p)$.

Proposition 8.10 *$O(n)$ time is needed for the routine* **Generate-Clustered-Network()***.*

Proof: Steps (1) and (2) in **Generate-Clustered-Network()** takes constant time. The For loop in step (3) inserts information in the net-list and mod-list data structures of the newly generated clustered network, this requires $O(n)$ time. Finally step (7) requires $O(N_{cl})$. Since the previous term is dominant, the complexity is $O(n)$ for Generate-Clustered-Network.

```
Generate-Clustered-Network()
  (1) Allocate Space for netlist and modlist.
  (2) Set new value for nets, modules, pins, etc....
  (3) For main ckt nnets
      (4) If net is cut
          (5) Insert info in netlist, modlist of new ckt
  (6) End For.
  (7) Set Block sizes.
```

Proposition 8.11 *$O(c')$ time is needed for Assign-Clusters-Blocks() to assign clusters to their corresponding blocks.*

Proof: It is clear that assigning clusters to blocks would take $O(c')$ since only a loop that involves the number of modules within the clustered-network would achieve the assignment.

Proposition 8.12 *Flatten-Clusters() requires $O(N_{cl}C_{sz})$ time.*

Proof: **Flatten-Clusters()** assigns modules within clusters to a partition **P**, which is assumed to be an array indexed by cell number, specifying which block each cell should be in. For all clusters N_{cl} a certain number of modules C_{sz} would be assigned to partition **P** in this case.

```
**Make-Uniform-Partitions()**
    (1) Compare sizes of blocks.
    (2) Determine deviation in sizes.
    (3) For sub ckt nblks
        (4) For $CL_n$ in circuit
            (a) If size exceeds target size
                (b) Move Module to under sized block
                (c) Update Size deviations.
        - End For.
    - End For.
```

Proposition 8.13 *Make-Uniform-Partitions() requires $O(bN_{cl})$ time.*

Proof: Steps (1) and (2) each requires $O(N_{cl})$ time. Step (3) is executed b times. Step (4) is executed N_{cl} times. Steps (6) and (7) require constant time. So the overall time required is $O(bN_{cl})$.

Theorem 8.2 *The GRASP Clustering based heuristic requires $O(m'N_{cl}(\log N_{cl} + p'))$.*

Proof: From Proposition 8.8 through 8.13, the dominant factor is in creating clusters which requires $O(m'N_{cl}(\log N_{cl} + p'))$.

8.3.2 Statistical Based Clustering

The following propositions show that the time complexity of the Statistical based clustering heuristic is $O(cq'p')$ for creating complete clusters.

```
**STATISTICAL BASED CLUSTERING()**
    (1) **Input_Main_Circuit()**
    (2) **Circuit_Statistics()**
    (3) **Form_Stat_Clusters()**
    (4) **Generate-Clustered-Network()**
    (5) Assign-Clusters-Blocks()
    (6) Flatten-Clusters()
    (7) Local_Improve_Partition()
    (8) **Make-Uniform-Partitions()**
```

Proposition 8.14 *$O(cp'q')$ time is needed to generate circuit statistics.*

Proof: The routine **Circuit_Statistics()** involves calling four functions, The complexity of each will be proven later on.

- **Generate_Common_Modules()**, complexity is $O(cp'q')$
- **Generate_Nets_Connections()**, complexity is $O(nT_n)$.
- **Generate_Cells_Connections()**, complexity is $O(cT_c)$
- **Generate_Nets_Binding()**, complexity is $O(n)$.

Since the first term is the most dominant, then the complexity of **Circuit_Statistics()** is $O(cp'q')$. The complexity can be reduced further to $O(p'q')$ while the circuit is being read initially.

```
Circuit_Statistics()
  (1) Generate_Common_Modules()
  (2) Generate_Nets_Connections()
  (3) Generate_Cells_Connections()
  (4) Generate_Nets_Binding()
```

Proposition 8.15 *$O(cp'q')$ time is needed for Generate_Common_Modules() to calculate the most common nets between modules in the network.*

Proof: Step (2) in **Generate_Common_Modules()** requires $O(p'q')$ (as will be proven next), and is called c times. Step (3) is computed in constant time. So the overall complexity of this routine is $O(cp'q')$.

```
Generate_Common_Modules()
  (1) For i=0 ... nmods
    (2) Find_Common_Modules(i)
  – End For
  (3) find max common modules;
```

Proposition 8.16 *$O(p'q')$ time is needed for Find_Common_Modules() to locate the common modules and calculate the first measure of attraction between modules.*

Proof: Step (1) in **Find_Common_Modules()** requires $O(N_C) \leq p'$. Step (2) requires $O(C_N \leq q')$. Steps (3) through (5) are computed in constant time, O(1). So the overall complexity of Find_Common_Modules() is $O(p'q')$.

```
Find_Common_Modules(K)
  (1) For i= 0 ... nets connected to mod K
    (2) For all mods connected to net i
      (3) locate common module;
      (4) ++tot_common_mods;
      (5) calculate measure1 = common_nets;
    (6) End For
  (7) End For
```

Proposition 8.17 *$O(cT_c)$ time is needed for Generate_Cells_Connections to determine the type of cells in the network and insert information in the cell lists.*

Proof: Step (1) in **Generate_Cells_Connections()** is executed c times. Steps (2) and (3) are done in constant time, O(1). Step (4) which involves inserting data in the cell_info list requires maximum $O(T_c)$ iterations. So the overall complexity is $O(cT_c)$.

```
Generate_Cells_Connections()
  (1) For i = 0 ... nmods
    (2) find max_nets_per_cell;
    (3) register type_cell;
    (3) insert data in cell_info;
  – End For
```

Proposition 8.18 *$O(nT_n)$ time is needed for Generate_Nets_Connections to determine the type of nets in the network and insert information in the net lists.*

Proof: Step (1) in **Generate_Nets_Connections()** is executed n times. Steps (2) and (3) are done in constant time, O(1). Step (4) requires maximum $O(T_n)$ iterations. So the overall complexity is $O(nT_n)$.

```
Generate_Nets_Connections()
  (1) For i=0 ... nnets
    (2) find max_cells_per_net;
    (3) register type_nets;
    (3) insert data in net_info;
  – End For
```

Proposition 8.19 *$O(N_{cl}C_{mcm})$ time is needed for Form_Stat_Clusters() to create the required number of clusters.*

Proof: Step (1) in **Form_Stat_Clusters()** is executed N_{cl} times. Steps (2) and (3) are done in constant time, O(1). Step (4) is executed $C_{cm} \leq C_{mcm}$ times. Step (4b) and (4c) are also executed in constant time, O(1). So the overall complexity is $O(N_{cl}C_{mcm})$.

```
FORM_STAT_CLUSTERS()
  (1) FOR i=0 ... Clusters
    (2) Find_Free_Module()
    (3) Insert_Seed_In_Cluster()
      (4) FOR j=0 ... Common_Modules
        (a) If Size_Cluster $\leq$ Target
          (b) Merge_Common_Module()
          (c) Update_Common_Modules()

      - End FOR

  - End FOR
```

Theorem 8.3 *The clustering heuristic based circuit statistics requires $O(cq'p')$.*

Proof: From Proposition 8.14 through 8.19, the dominant factor is in locating the common modules in the circuit which requires $O(cq'p')$. Other routines involved are:

- **Generate-Clustered-Network()**, requires $O(n)$ as proved in Proposition 8.10.
- **Assign-Clusters-Blocks()**, requires $O(c')$ as proved in Proposition 8.11.
- **Flatten-Clusters()**, requires $O(N_{cl}C_{sz})$ as proved in Proposition 8.12.
- **Local_Improve_Partition()**, requires $O(m'N_{cl}(\log N_{cl} + p'))$
- **Make-Uniform-Partitions()**, requires $O(bN_{cl})$ as proved in Proposition 8.13.

8.4 Summary

In this chapter a worst case complexity analysis of Tabu Search partitioning heuristic was presented. The computation time required for two clustering techniques based on GRASP and statistical information was also presented in detail. The next chapter gives an overview of the whole dissertation, and summarizes the main contributions made.

Chapter 9

Conclusions and Future Directions

In view of the increasing complexity of VLSI circuits, there is a growing need for sophisticated CAD tools to automate the synthesis, analysis, and verification steps in the design of VLSI systems. CAD tools have to enable designs that are too large or complex to undertake otherwise, shorten design time, improve product quality, and reduce product costs. Despite significant research efforts in this field, the CAD tools still lag behind the technological advances in fabrication. This calls for development of efficient heuristics for physical design automation.

The main goal of this work was to introduce new methodologies to tackle the combinatorial optimization problems in circuit layout as a means to obtain near optimal solutions. The approach relies on four main components in the form of: *efficiency*, *robustness*, *speed* and *complexity reduction*. A strategy was set to meet these goals effectively using advanced search techniques, hybridization, clustering and distributed processing.

The next section summarizes the main accomplishments of this investigation. Section 9.2 summarizes the best methodology that can be used to solve the circuit layout problem in particular and other combinatorial optimization problems in general. Finally, Section 9.3 introduces the possible future directions to further improve upon the developed strategy.

9.1 Accomplishments

In this thesis, a novel methodology was introduced to tackle and solve the VLSI circuit layout problem effectively. The methodology is summarized in Figure 9.1. The primary task of this thesis involved developing and evaluating the performance of recent advanced search methods such as *Tabu Search* [Reev93], GRASP [?], Simulated Annealing [?], and Genetic Algorithms [Venk91] on circuit layout. A comparison was made between advanced search heuristics and traditional techniques in terms of the execution time, quality of solution, and robustness. We showed the main advantages and disadvantages of each heuristic and the best parameter setting for the circuit partitioning problem. Results indicate that advanced search techniques are effective over traditional methods for solving circuit partitioning efficiently.

The importance of intelligently controlling the performance of a search based heuristic was demonstrated. As made explicit in Tabu Search, one may choose any algorithmic framework and superimpose a search controller within this technique as an intelligent adapter.

We also showed the importance of **combining** these distinct models of computation to solve the combinatorial optimization problems in circuit layout. The integration of Tabu Search and Simulated Annealing gave rise to a hybrid probabilistic technique that avoids the necessity of unproductive wandering due to randomization while producing good effective solutions. Combining GRASP with Tabu Search

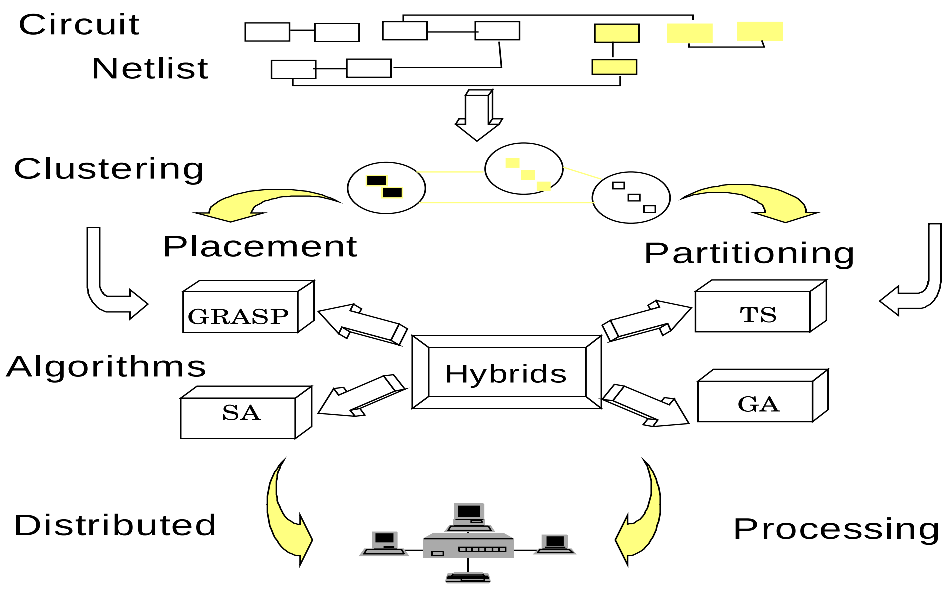


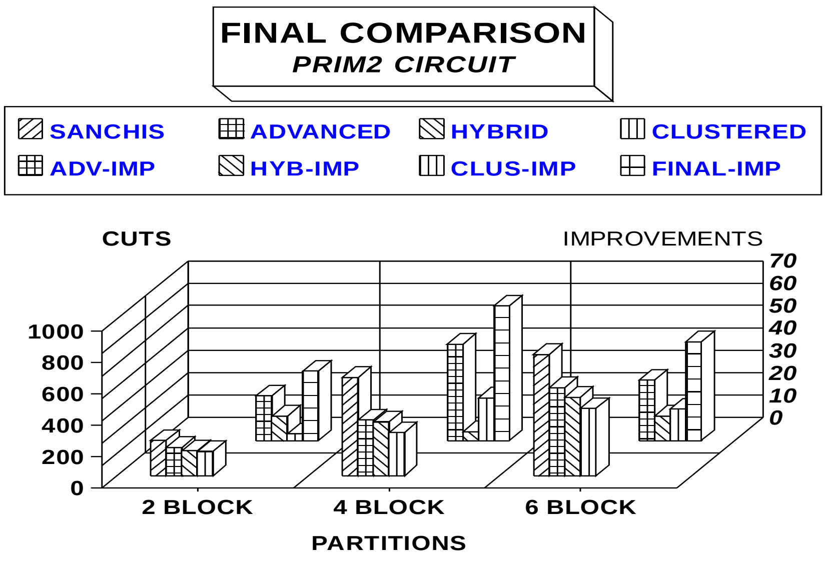
Figure 9.1: Towards optimal solution of circuit layout

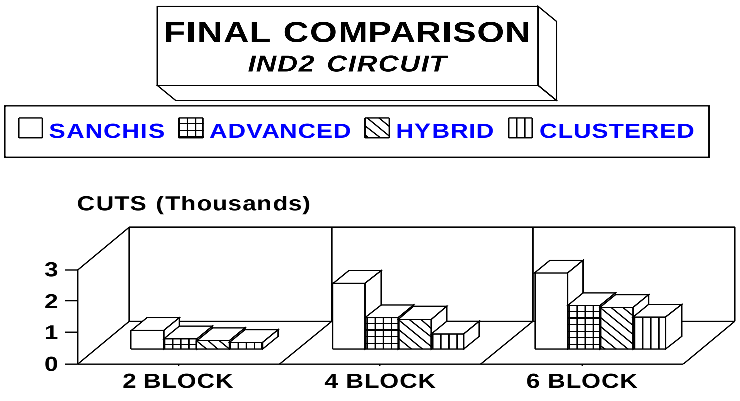
and Genetic Algorithms not only improved the solution quality but also reduced the computation time effectively. Finally the combination of Tabu Search with the Genetic Algorithm attempted to combine the strengths of the latter in exploring the solutions space effectively with the former in finely tuning the search in the most promising regions. Results obtained indicate that the hybridization techniques based on advanced search heuristics generated solutions superior to results obtained from any of the pure single processing heuristics. The running time of these methods are much faster than those obtained using the Simulated Annealing algorithm and Tabu Search implementations based on long and intermediate term memories.

Another main goal of the dissertation was to reduce the complexity of the design by utilizing a hierarchical approach to shorten the design period. This was achieved by using **circuit clustering** which plays a fundamental role in reducing the complexity of the design and improving the performance of the design process.

An attempt was made to assess the quality of solutions obtained by the advanced, hybrid, clustering techniques in finding near optimal solutions. We compared our results with those obtained by the CPLEX MIP package. Results indicate clearly that we are truly heading towards optimal solutions for the circuit layout problem. Another important goal achieved was in improving the performance of the branch and bound algorithm in solving the circuit partitioning problem using the CPLEX MIP package. Some of the developed partitioning and clustering heuristics were used to provide necessary information that helped in reducing the computation time on average by 70%. This strategy helped in solving some of the problems that were difficult to solve using the CPLEX MIP package alone in fraction of the time. and **??** compare circuit partitioning using traditional iterative improvement, advanced each column represents the amount of the previous method. For example in the second column for advanced cut-size reduction over the column in each table represents the overall improvement of clustered-hybrid-advanced search technique interchange methods. It is clear from the tables that the strategy used is layout problems in general.

Figure 9.2 compares the results obtained for circuit partitioning using traditional iterative improvement, advanced search techniques, hybrids and clustering. The Y1 axis in the first figure represents the different solutions obtained using, Sanchis interchange method, Tabu Search, TS-GA hybrid, and clustered TS-GA hybrid respectively. On the other hand the Y2 axis represents the percent of improvement achieved over using the previous method. For example, the first bar in the Y2 axis represents the amount of improvement in cut-size reduction obtained using the





	2 BLOCK	4 BLOCK	6 BLOCK
%ADV-IMP	45	52	43
%HYB-IMP	15	5	4
%CLUS-IMP	24	49	23
%FINAL-IMP	65	77	58

PARTITIONS

Figure 9.2: Effect of strategy on an intermediate and large size circuits

advanced search heuristic over the traditional Sanchis approach. The final bar in the first figure represents the overall improvement of using a clustered-hybrid-advanced search technique over the traditional interchange methods. In the second figure, the table summarizes the amount of improvement achieved using each methodology over the previous one. It is clear from the figures that the strategy used is successful and leads towards near optimal solutions of these layout problems in general.

Our final goal was to increase the processing power available to the execution of the developed heuristics. Unlike previous parallel algorithms that required special parallel machines with shared memory or dedicated interconnection networks, the CAD algorithms ran on a network of workstations in a robust and efficient manner. Tables 9.1, 9.2 and 9.3 summarize the effect of using advanced search techniques, hybridization and distributed processing over the traditional Sanchis interchange method. The tables clearly indicate the advantage of using distributed processing in improving the computation time of the methodology proposed by an average of 60% over sequential implementations.

Hierarchical Comparison of 2-Way Partitioning								
Ckt	Sanchis	Advanced		Hybrid		DISTRIBUTED		Overall
	Time	Time	IMP	Time	IMP	Time	IMP	IMP
Prim2	433	284	34%	632	-55%	161	74%	**62%**
Ind2	2661	2250	15%	3422	-34%	1721	49%	**35%**

Table 9.1: 2-Way Partitioning: CPU TIME COMPARISON

The different advanced search techniques and hybrids developed, have used the circuit partitioning as a paradigm for circuit layout. In Chapter 7, some of the developed heuristics were used effectively to solve the circuit placement problem,

Hierarchical Comparison of 4-Way Partitioning								
Ckt	Sanchis	Advanced		Hybrid		DISTRIBUTED		Overall
	Time	Time	IMP	Time	IMP	Time	IMP	IMP
Prim2	787	299	62%	495	-39%	126	74%	**83%**
Ind2	12729	3902	69%	5087	-23%	1681	66%	**86%**

Table 9.2: 4-Way Partitioning: CPU TIME COMPARISON

Hierarchical Comparison of 6-Way Partitioning								
Ckt	Sanchis	Advanced		Hybrid		DISTRIBUTED		Overall
	Time	Time	IMP	Time	IMP	Time	IMP	IMP
Prim2	1382	435	68%	632	-31%	211	66%	**84%**
Ind2	25132	5373	78%	6600	-18%	1322	79%	**94%**

Table 9.3: 6-Way Partitioning: CPU TIME COMPARISON

which is more complex than circuit partitioning. A novel automatic placement method that is suitable for large scale and highly interconnected systems was developed. The initial placement was obtained using a combined global optimization algorithm and a partitioning based heuristic. Results obtained indicate that this initial strategy is successful in placing most modules in their appropriate position. A Tabu Search placement heuristic was used to further improve the initial placement.

9.2 Summary

Search problems are more like “lock and key” problems in which ideally we want to use a search algorithm to make the right kind of assumption about the landscape it is searching (i.e, right key for a given lock). A search heuristic is considered to be successful in solving a complex optimization problem if it possess the following characteristics:

- Capable of starting the search from good initial points in the search space
- Adapts its parameters according to the solution space being searched
- Efficiently searches local neighborhoods
- Explores all regions of the parameter space
- Utilizes historical information about moves made as the search progresses
- Robustness and speed

This dissertation presented search techniques based on (i) pure advanced methods, (ii) hybrids, (iii) clustering, and a combination of the above mentioned techniques. Each technique as indicated in the previous chapters has its advantages and disadvantages. If we were to decide upon the best heuristic to solve the circuit layout problem in particular, and other combinatorial optimization problems in general (given that feasible solutions are easy to obtain), we would recommend the following technique as shown in Figure 9.3. This technique is based on GRASP, Genetic Algorithms, Tabu Search and clustering. The question may well be asked: how good is this technique in solving a certain combinatorial optimization problem, and what are its distinguishing features?

9.2.1 Clustered GA-TS Approach

Most real world problems in general and VLSI circuit layout problems in particular are too complex for any single processing technique to solve in isolation. The majority of CAD tools available today such as TimberWolf [Sech86], Cadence [**?**],

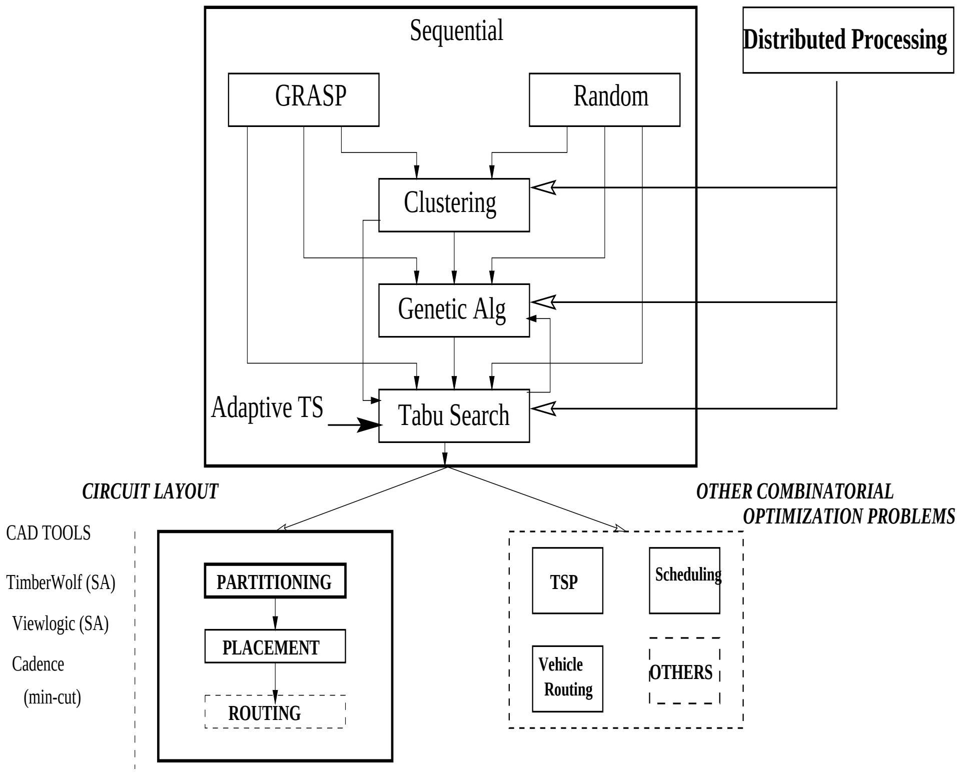


Figure 9.3: CGA-TS: The Best Overall Search Heuristic

and View-logic [?] for circuit placement rely on a single technique in the form of Simulated Annealing and variation of the Kernighan and Lin iterative improvement heuristic. This thesis presented a hybrid methodology that not only produces better results than Simulated Annealing or traditional interchange methods, but also reduces the computation time involved and the complexity of the design process.

The technique we propose in Chapter 5 (CGA-TS), possess all features that a good search technique should acquire. It is capable of obtaining good initial solutions through two efficient mechanisms, using GRASP and Genetic Algorithms. It is very capable of local neighborhood search relying on a powerful technique based on Tabu Search. In addition, embedding an adaptive controller within Tabu

Search, enables this technique to identify when and how parameters are stable, and adjusts these parameters depending on the problem size and structure of the solution space being searched. CGA-TS is also capable of making use of previous information collected during the search process, using the short term memory of Tabu Search and by structured recombination crossover operator of the Genetic Algorithm. The important role of exploring the solution space is accomplished through the use of the Genetic Algorithm which processes information fed back to it from the Tabu Search heuristic. Thus, it is utilized for further intensification and diversification of the search. Finally, the technique further utilizes clustering for the following reasons. First, clustering is used to condense the problem being solved as a means to reduce the complexity of the problem. This is achieved as described in Chapter 5 by smoothing the landscape of the problem and reducing the number of local minima in the k-interchange neighborhood structure. This process enables the local search heuristic to converge to a neighborhood solution in a short period of time. Secondly, clustering is used to reduce the solution space being explored by the Genetic Algorithm thus increasing the efficiency of the latter to examine the solution space effectively.

Distributed processing may be applied to some or all stages of the search process as seen in Figure 9.3 to increase the processing power available to the execution of the search heuristic. The main advantage of the Network Multiprocessing Environment presented in Chapter 6 is in minimizing the running time and improving the efficiency of the proposed CGA-TS heuristic. Parallelism can be applied at each stage of the search process by obtaining many good initial solutions, clustering, performing local search, and finally diversifying the search procedure.

9.3 Future Work

There are several enhancements and extensions that can be applied to further improve upon the methodology used in this dissertation.

First, one can adapt the same strategy used to control and improve the performance of Tabu Search for other advanced search techniques. Determining an adaptive parameter setting through a control mechanism would save a vast amount of time that can be consumed by brute force methods in finding good parameter settings for advanced search algorithms. This method is robust in the sense that many parameters adapt according to the properties of the solution space being searched.

The practical successes of Tabu Search have promoted useful research into ways to exploit its underlying ideas more fully. At the same time, many facets of these ideas remain to be explored. The issues of identifying best combinations of short and long term memory and best balances of intensification and diversification strategies still contain many unexamined corners, and some of them harbor important discoveries for developing more powerful solution methods in the future.

For combinatorial problems whose complexity suggests the use of heuristic methods, such as the partitioning, placement and routing in circuit layout, it is worthwhile to have a lower bound on what can be achieved, regardless of the algorithm, provided the calculation of the bound is itself not too onerous and the bounds derived are not too far from the correct value. The calculation of the bound can be used as a stopping criteria for many of the search techniques presented in Chapters 3–4. This method restricts the continuation of the search in regions of the search space that have been explored, thus cutting the overall time of the heuristic.

The calculation of the bound may itself suggest new approaches to solve or find approximate solutions to the original problem.

The clustering heuristic presented in Chapter 5 was able to significantly reduce the complexity of the circuits and improve the performance of the circuit partitioning process. Yet, the performance of the clustering algorithm can be enhanced by resorting to the following techniques:

- incorporating other statistical measures of the circuits being solved to form highly connected sub-circuits,
- investigate better closeness functions of modules similar to those presented in Chapter 5,
- reducing the complexity of the statistical clustering heuristic, by computing some of the information while the circuit is initially being read,
- utilize advanced search techniques such as Tabu Search in the refinement stage of clustering.

Another possible improvement to the partitioning based heuristics is to use a "net-based move model" instead of the traditional "node-based move model" used by almost all iterative improvement based partitioning heuristics. In the "net-based move model" multiple nodes can be moved since a move is associated with a net rather than a module. Although the net-based move model provides more insight into the improvement of current partition, it is more expensive than the node-based move model, because more nodes are involved in each move.

One way in which we try to improve layout heuristics is to soften the boundaries between layout phases. In other words, rather than just estimating the effect of a succeeding layout phase, we integrate that phase into the current layout phase. An example is the integration between placement and global routing that is being attempted by several researchers [Dona90].

9.4 Epilogue

Recent years have undeniably witnessed significant gains in solving difficult optimization problems, but it must also be acknowledged that a great deal remains to be learned. Research in these areas is full of uncharted and inviting landscapes. Research is an aspect of life which stimulates the birth of many new ideas and directions just by attempting to find a solution for one problem. This research work has broadened my knowledge in many areas which I never envisaged before starting this thesis. New ideas will definitely keep me busy for years to come.

Appendix A

Data Structures, and Algorithms

A.1 Complexity of Partitioning Heuristics

Table A.1 summarizes the running time of a series of iterative improvement algorithms for graph and network partitioning.

A.2 Definitions & Data Structures

In this appendix the main procedures and data structures involved in multiple way Tabu Search partitioning algorithm are presented.

Algorithm	**Complexity**	**Comments**
Kernighan & Lin	$O(c^2 \log c)$	Formulas for computing gain
Fiduccia & Mattheyses	$O(m)$	Advanced data structures
Krishnamurthy	$O(lm)$	Introduced the concept of level gains
Sanchis	$O(lmb(\log b + pl))$	Multi-way uniform partitioning

Table A.1: Computational complexity of circuit partitioning heuristics

Can-Move

```
(1) Do
      (a) Point to cell with highest gain,
      (b) if (Valid-Move)
            If (Move-Not-Tabu)
            Return (Pointer to cell)
      (c) Advance Pointer to next cell with highest gain,
    While(gain-ptr is valid)
```

Valid-Move

```
- size-source-blk = blocksize[source] -1
- size-dest-blk = blocksize[destination] -1
- If (size-source-blk - blocks[source] $\leq$ TOLERANCE)
          AND if (size-dest-blk - blocks[dest] $\leq$ TOLERANCE)
            return(TRUE)
- Else return (FALSE)
```

Move-Not-Tabu

```
(1) PTR = TabuList[Module]
(2) If (PTR $\neq$ NULL)
      (a) Module-In-TabuList = TRUE
      (b) if(Aspiration Condition Met)
                return (TRUE)
          Else return (FALSE)
    Else Return(TRUE)
```

Make-Move()

```
(1) C = Nextmove.cell, $A_j$ = Nextmove.source, $A_k$ = Nextmove.target;
(2) For each net N connected to C
      (a) For (All blocks)
            if(Net is Critical before Move)
                Update gain of all cells on net N
                Adjust Bucket-Gain-List
```

```
        End For
    (b) Update net parameters
    (c) if (Net is Critical after Move)
            Update gain cells affected
            Adjust Bucket-Gain-List
  End For
```

$\boxed{\textbf{Beta-Prime()}}$

- if N is free then
  - $\beta'_{A_k} = c_N - \phi_{A_k}(N)$
- else if N is loose then
  - $\beta'_{A_k} = c_N - \phi_{A_k}(N) - 1$
- else if $\beta_{A_k} \neq \infty$
  - $\beta'_{A_k} = \infty$

$\boxed{\textbf{Update-TabuList}}$

- PTR = TabuList[Module]
- delete entry at head of the list
- link new entry to head of the list
- adjust tail of the list
- link PTR to new entry,

$\boxed{\textbf{Update-Gain}(C, A_k, N)}$

- let A_j be the block to which C belongs
- increment the gain for moving cell C to A_k

$\boxed{\textbf{Reverse-Update-Gain}(C, A_k, N)}$

- let A_j be the block to which C belongs
- decrement the gain for moving cell C to A_k

Appendix B

NMP Routines

B.1 Connection Based Routines

The basic SendNode and RecvNode routines do not check the connectivity of their destination and source nodes. An attempt to send to or receive from a non-existent or non-connected node will return an error code and produce a message on the standard error channel. The following routines are used to determine the status of the NMP connection configuration. To find out if a node is connected to the current node, call *IsConnected(NodeId)* The configured connection between two other nodes may be determined by *AreConnected(NodeId1, NodeId2)* To know if the entire configuration was successfully started, use: *MadeConn()*

B.2 Identification Routines

The following routines are used to access the internal state of the NMP structures: A node can determine the name of the machine it's running on, using *GetHost()*.

To find out the host machine name of another NMP node, use *GetAnyHost(NodeId)* To find out the total number of nodes in the NMP, use *GetNodes()* A node may determine its own node *ID* by calling, *GetMyId()*

B.3 Timing Routines

Two clock routines are provided for system-consistent timing of programs, *clock_start() and clock_stop(rt,ut,st,it)* These calls may be nested, clock_stop terminating the most recent instance of clock_start(). The real time is returned in *rt*, the user time in *ut* the system time in *st* and the idle time *it*. All times are in milliseconds.

B.4 Dynamic Reconfiguration of the System

When the application needs to change its configuration while running, it may add nodes and connections. Nodes can delete themselves by calling NodeClose. The following procedure creates a new node to participate in the NMP and establishes a connection between it and the node issuing the call: *AddNode(HostName, ExecFileName, Sin, Sout, Serr, Bits).* To create a new connection between existing nodes, use *AddConnect(Type, NodeId).* The connection matrix is updated only in the two nodes involved. The other nodes know nothing of this new connection.

Appendix C

Complete Tabulated Results

This appendix presents results tabulated for all benchmarks introduced in Chapter 3. In the next few sections we summarize results obtained for:

- Advanced search techniques.
- Hybrid search techniques.
- Results based on clustering.
- Distributed processing results.

In each section, results for 2,4 and 6 way partitioning are presented. Also, the amount of improvement achieved of using a certain method over previous used methods are indicated.

C.1 Results of Advanced Search Techniques

Circuit	**Sanchis**		**SDHC**		**Solution Quality**	
	Cuts	Time	Cuts	Time	%IMP	Passes
Chip1	20	0.4	20	1.3	0%	3/3
Chip2	16	0.4	14	2.5	12%	4/10
Prim1	106	1.7	75	5.6	29%	5/25
Prim2	322	9.7	248	36.6	29%	9/48
Ind1	97	5.8	76	14.9	21%	8/17
Bio	180	20.9	135	61.2	25%	12/18
Ind2	677	74.4	278	153.1	58%	19/35
Ind3	659	50.5	440	237.3	33%	7/33

Table C.1: 2 Way Partitioning SDHC

Circuit	**Sanchis**		**SDHC**		**Solution Quality**	
	Cuts	Time	Cuts	Time	%IMP	Passes
Chip1	55	0.8	55	3.0	0%	7/7
Chip2	45	0.6	36	3.0	20%	5/47
Prim1	171	2.2	134	11	21%	5/29
Prim2	761	16.9	682	66	10%	6/31
Ind1	259	9.1	215	35	17%	7/45
Bio	762	56.5	592	223	22%	8/43
Ind2	2282	283	2019	579	12%	17/39
Ind3	2583	180	2206	813	15%	10/63

Table C.2: 4 Way Partitioning SDHC

Circuit	**Sanchis**		**SDHC**		**Solution Quality**	
	Cuts	Time	Cuts	Time	%IMP	Passes
Chip1	90	0.8	74	1.7	18%	4/11
Chip2	70	0.9	61	4.3	13%	5/42
Prim1	203	3.3	162	14.3	20%	6/44
Prim2	890	27.7	782	108	12%	5/42
Ind1	431	21.2	411	61	5%	9/27
Bio	922	109	763	379	17%	9/50
Ind2	2706	350	2319	1368	14%	15/49
Ind3	2655	150	2655	300	0%	9/9

Table C.3: 6 Way Partitioning SDHC

Ckt	**INTER**		**GRASP**		**% IMP**	
	Cuts	Time	Cuts	Time	Cuts	Time
Chip1	20	18	20	2.4	-	78%
Chip2	15	19	14	1.7	6%	91%
Prim1	60	91	56	14.0	6%	84%
Prim2	226	433	179	66	21%	84%
ind1	42	211	50	28.8	-16%	86%
Bio	102	1058	89	124	13%	88%
ind2	593	2661	325	155	45%	87%
ind3	514	2294	520	123	-1%	94%
AVG	196	848	156	64	20%	92%

Table C.4: GRASP 2-Way partitioning

Ckt	**INTER**		**GRASP**		**% IMP**	
	Cuts	Time	Cuts	Time	Cuts	Time
Chip1	55	27.4	59	3.4	-6%	87%
Chip2	39	24.4	31	3.5	20%	85%
Prim1	155	96.4	127	12	18%	87%
Prim2	627	787	438	102	30%	87%
Ind1	259	526	160	42	38%	91%
Bio	680	2627	386	213	43%	91%
Ind2	2102	12729	1148	312	45%	97%
Ind3	2183	5874	1898	432	13%	92%
AVG	762	2836	530	140	30%	95%

Table C.5: GRASP 4-Way partitioning

Ckt	**INTER**		**GRASP**		**% IMP**	
	Cuts	Time	Cuts	Time	Cuts	Time
Chip1	77	39.4	67	4.8	13%	87%
Chip2	63	40.2	59	4.6	6%	88%
Prim1	181	133	153	18.8	15%	85%
Prim2	773	1382	644	101	17%	92%
Ind1	364	769	303	85	17%	89%
Bio	821	4657	451	359	45%	92%
Ind2	2430	25132	1464	1066	40%	96%
Ind3	2640	13131	2843	371	-7%	97%
AVG	918	5660	748	251	18%	95%

Table C.6: GRASP 6-Way partitioning

Circuit	**GA (Penalty)**		**GA (Rep_Sim)**		**GA (Rep_Adv)**	
	Cuts	Time	Cuts	Time	Cuts	Time
Chip1	143	0.5	60	0.7	25	3.5
Chip2	117	0.5	61	0.9	22	3.2
Prim1	521	1.4	121	2.0	81	9.8
Prim2	1885	5.3	302	7.2	190	36.3
Ind1	1278	4.0	279	5.3	74	25.2
Bio	3346	10.9	1748	14.6	321	75
Ind2	7607	22.8	2356	29.9	572	166
Ind3	13621	32.1	2307	41.9	930	222

Table C.7: Comparison of GA Techniques for 2 Way Partitioning

Circuit	**GA (Penalty**		**GA (Rep_Sim)**		**GA (Rep_Adv)**	
	Cuts	Time	Cuts	Time	Cuts	Time
Chip1	224	0.5	113	0.6	65	4.9
Chip2	185	0.5	107	0.6	62	4.1
Prim1	697	1.5	288	1.9	177	14.1
Prim2	2389	5.1	900	6.9	529	54.1
Ind1	1705	3.8	616	5.0	183	39.2
Bio	4506	10.9	2846	14.4	1677	105
Ind2	10129	21.1	5073	28.6	2363	243.2
Ind3	17508	28.1	4582	39.5	2444	365

Table C.8: Comparison of GA Techniques for 4 Way Partitioning

Circuit	**GA (Penalty)**		**GA (Rep_Sim)**		**GA (Rep_Adv)**	
	Cuts	Time	Cuts	Time	Cuts	Time
Chip1	240	0.7	141	0.6	86	6.8
Chip2	200	0.4	129	0.7	79	5.4
Prim1	755	1.5	353	1.9	189	18.6
Prim2	2550	5.1	1192	6.8	671	71.1
Ind1	1816	3.8	1043	5.0	427	52.2
Bio	4831	10.5	3050	13.9	1759	142
Ind2	10849	20.9	6322	28.8	3342	368
Ind3	18474	27.8	6959	40.1	2890	495

Table C.9: Comparison of GA Techniques for 6 Way Partitioning

Circuit	**Eigen**		**GRASP**		**GA**	
	Cuts	Time	Cuts	Time	Cuts	Time
Chip1	42	3.9	46	0.3	25	3.5
Chip2	32	3.4	28	0.1	22	3.2
Prim1	181	10.4	101	1.0	81	9.8
Prim2	694	35.4	404	4.8	190	36.3
Ind1	244	15.8	185	3.9	74	25.2
Bio	440	95.5	293	11.7	321	75
Ind2	1334	169	621	24.1	572	166
Ind3	3188	151	1819	30.9	930	222

Table C.10: Constructive Methods for 2-Way Partitioning

Ckt	**Eigen**		**GRASP**		**GA**	
	Cuts	Time	Cuts	Time	Cuts	Time
Chip1	80	7.8	83	0.2	65	4.9
Chip2	51	5.1	43	0.2	62	4.1
Prim1	298	13.5	185	2.2	177	14
Prim2	925	183	569	8.8	529	54
Ind1	359	45.1	216	7.1	183	39
Bio	872	163	426	26	1677	105
Ind2	4643	234	1345	78	2363	243
Ind3	6627	647	2456	70	2444	365

Table C.11: Constructive Methods for 4-Way Partitioning

Circuit	**Eigen**		**GRASP**		**GA**	
	Cuts	Time	Cuts	Time	Cuts	Time
Chip1	91	9.9	85	0.7	86	6.8
Chip2	83	7.9	66	0.3	79	5.4
Prim1	329	22.4	203	2.7	189	18.6
Prim2	1095	333	662	18.9	671	71.1
Ind1	415	81.5	328	10.2	427	52.2
Bio	805	295	472	42.5	1759	142
Ind2	4530	555	1649	173	3342	368
Ind3	7254	653	3521	149	2890	495

Table C.12: Constructive Methods for 6-Way Partitioning

Circuit	**TS**		**ATS**		**Improvement**
	Cuts	Time	Cuts	Time	Improvement
Chip1	20	13.3	20	0.6	-
Chip2	14	6.9	14	7.9	-
Chip3	7	7.4	7	4.8	-
Chip4	13	5.7	8	4.2	38%
Prim1	59	52.6	55	5.6	7%
Prim2	240	3:25	221	5:57	8%
Ind1	59	1:31	28	202	52%
Bio	144	3:45	134	4:52	6%
Ind2	805	21:10	638	51:39	21%
Ind3	462	28:52	357	73:56	23%

Table C.13: 2 Way Partitioning Results Using Adaptive Tabu Search

Circuit	**TS**		**ATS**		**Improvement**
	Cuts	Time	Cuts	Time	Improvement
Chip1	49	12.3	51	11.4	-
Chip2	35	6.9	30	12.3	14%
Chip3	28	4.6	30	14.1	-
Chip4	21	5.7	20	13.0	5%
Prim1	126	50.2	122	44.4	3%
Prim2	663	1:52	541	551	18%
Ind1	135	2:53	124	285	8%
Bio	509	6:0	494	358	3%
Ind2	2323	26:23	1711	48:2	26%
Ind3	2139	54:5	1752	101:1	18%

Table C.14: 4 Way Partitioning Results Using Adaptive Tabu Search

Circuit	**TS**		**ATS**		**Improvement**
	Cuts	Time	Cuts	Time	Improvement
Chip1	64	12.5	61	16.3	5%
Chip2	38	13.8	39	11.3	-
Chip3	41	10.5	40	14.9	2%
Chip4	27	3.0	27	18.6	-
Prim1	159	1:13	134	127	16%
Prim2	769	1:46	664	552	14%
Ind1	230	5:14	176	606	23%
Bio	791	6:29	666	21:0	16%
Ind2	2662	17:5	1827	56:0	31%
Ind3	2623	44:3	1874	105:0	28%

Table C.15: 6 Way Partitioning Results Using Adaptive Tabu Search

C.2 Results of Hybrid Search Techniques

GRASP-GENETIC SEARCH HYBRID						
Circuit	**Blocks**	**Random-GA**		**GRASP-GA**		**%IMP**
		Cuts	Time	Cuts	Time	Cuts
Chip1	2 Blks	20	37.2	20	17.0	-
	4 Blks	57	52.9	48	25.5	16%
	6 Blks	83	72.1	73	42.4	12%
Chip2	2 Blks	16	39.3	17	42.8	-
	4 Blks	47	50.0	43	59.6	9%
	6 Blks	68	62.7	57	69.3	16%
Chip3	2 Blks	7	34.8	7	36.4	-
	4 Blks	37	41.8	37	45.9	-
	6 Blks	46	48.5	46	52.9	-
Chip4	2 Blks	8	35.3	8	37.2	-
	4 Blks	31	51.1	30	49.5	3%
	6 Blks	48	55.7	50	59.5	-4%
Prim1	2 Blks	76	113	70	140	8%
	4 Blks	158	163	130	208	17%
	6 Blks	171	216	158	276	8%
Prim2	2 Blks	168	213	169	236	-
	4 Blks	454	202	343	230	24%
	6 Blks	608	269	520	306	14%
Ind1	2 Blks	68	376	56	393	18%
	4 Blks	117	558	109	609	7%
	6 Blks	272	581	200	683	26%
Ind2	2 Blks	461	1055	435	866	6%
	4 Blks	1949	1656	1129	1378	42%
	6 Blks	2895	2179	1559	1818	46%
Ind3	2 Blks	645	5285	362	5173	44%
	4 Blks	1674	7825	1639	8189	2%
	6 Blks	2832	5506	2381	7182	16%

Table C.16: Results of GRASP-GA hybrid implementation

Circuit	**Blks**	**RAND-TS**		**EIG-TS**		**GRASP-TS**		**GA-TS**	
		Cuts	Time	Cuts	Time	Cuts	Time	Cuts	Time
Chip1	2	20	13.4	20	12.6	20	10.2	20	9.4
	4	49	12.3	46	11.8	52	11.4	54	9.0
	6	64	12.4	59	23.8	55	20.5	56	17.1
Chip2	2	19	1.3	17	3.6	14	4.5	14	5.4
	4	39	4.8	34	5.4	31	6.8	30	7.1
	6	64	7.8	58	8.2	40	12.8	35	13.2
Prim1	2	72	15.0	61	31.2	60	17.9	58	19.3
	4	126	39.8	119	46.6	102	56.8	115	40.6
	6	168	38.8	140	49.8	140	31.5	136	55.3
Ind1	2	59	1:31	37	1:34	63	1:12	45	0:58
	4	135	2:53	135	2:40	129	4:19	98	2:13
	6	230	5:23	144	2:51	185	2:42	157	3:06
Prim2	2	240	3:21	198	2:50	236	2:47	170	1:37
	4	663	1:50	408	3:19	368	7:37	371	2:27
	6	769	1:44	534	5:57	595	2:20	569	2:55
Bio	2	144	3:37	199	3:46	87	3:19	186	2:52
	4	509	6:13	275	4:53	386	4:55	529	5:02
	6	791	6:46	353	6:52	413	5:47	481	9:26
Ind2	2	675	1710	520	1809	388	1381	286	821
	4	2259	1752	1804	1318	1189	1602	1403	1272
	6	2682	1733	1803	2171	1371	3038	1537	1132
Ind3	2	1618	4:47	1615	18:49	521	7:19	686	14:18
	4	3153	39:18	2825	43:26	1556	4:55	1858	16:00
	6	4120	26:17	3471	24:04	2944	9:55	1900	20:59

Table C.17: Effective hybrid search techniques

Circuit	**SA**		**SAM**		**% IMPROVE**	
	Cuts	Time	Cuts	Time	Cuts	Time
Chip1	20	98	20	43	-	**56%**
Chip2	14	93	14	52	-	**44%**
Prim1	65	360	56	156	13%	**56%**
Prim2	224	900	222	528	0.5%	**41%**
Ind1	89	1004	37	468	58%	**53%**
Bio	199	2920	150	1759	25%	**39%**
Ind2	694	6420	425	2634	38%	**59%**
Ind3	655	9425	563	5685	14%	**40%**

Table C.18: 2 Way Partitioning for SAM Hybrid Heuristic

Circuit	**SA**		**SAM**		**% IMPROVE**	
	Cuts	Time	Cuts	Time	Cuts	Time
Chip1	45	393	46	196	-3%	50%
Chip2	29	326	26	167	10%	48%
Prim1	106	1311	105	619	1%	52%
Prim2	365	4312	346	2327	5%	46%
Ind1	141	3922	138	1875	2%	52%
Bio	327	217:0	338	94:4	-3%	56%
Ind2	809	422:3	849	262:5	-5%	38%
Ind3	1668	668:1	1668	285:24	-	57%

Table C.19: 4 Way Partitioning for SAM Hybrid Heuristic

Circuit	**SA**		**SAM**		**% IMPROVE**	
	Cuts	Time	Cuts	Time	Cuts	Time
Chip1	54	715	55	366	-1%	48%
Chip2	32	553	32	298	-	46%
Prim1	124	2440	123	1190	1%	51%
Prim2	428	10736	429	5880	-	45%
Ind1	182	7629	183	3740	-1%	51%
Bio	387	440:27	402	209:3	-3%	52%
Ind2	1395	799:27	1424	519:11	-2%	35%
Ind3	2801	1340:1	2442	714:34	13%	47%

Table C.20: 6 Way Partitioning for SAM Hybrid Heuristic

C.3 Results of Clustering Techniques

Circuit	**ITER FLAT**		**ITER CLUS**		**REDUCTION**	
	Cuts	Time	Cuts	Time	**% Cuts**	**Time**
Chip1	20	17.3	20	1.2	**0%**	**93%**
Chip2	15	19.3	18	0.5	**-16%**	**97%**
Prim1	60	91.1	57	5.2	**+5%**	**94%**
Prim2	226	433	176	34.5	**+22%**	**92%**
Bio	102	1058	111	190	**-8%**	**82%**
ind1	42	211	45	25.4	**-7%**	**88%**
ind2	593	2661	337	446	**+43%**	**83%**
ind3	514	2294	414	300	**+19%**	**87%**

Table C.21: 2 way iterative improvement clustering results

Circuit	**ITER FLAT**		**ITER CLUS**		**REDUCTION**	
	Cuts	Time	Cuts	Time	**Cuts**	**Time**
Chip1	55	27.4	52	0.7	**+6%**	**97%**
Chip2	39	24.4	44	0.7	**-11%**	**97%**
Prim1	155	96.4	101	3.5	**+34%**	**96%**
Prim2	627	787	305	24.3	**+51%**	**97%**
Bio	680	2627	244	90.3	**+64%**	**96%**
ind1	259	526	142	13.4	**+45%**	**97%**
ind2	2102	12729	826	264	**+60%**	**98%**
ind3	2183	5874	1124	310	**+48%**	**95%**

Table C.22: 4 way iterative improvement clustering results

Circuit	**ITER FLAT**		**ITER CLUS**		**REDUCTION**	
	Cuts	Time	Cuts	Time	**Cuts**	**Time**
Chip1	77	39.4	60	0.9	**+22%**	**98%**
Chip2	63	40.2	52	0.8	**+17%**	**98%**
Prim1	181	133	141	4.4	**+22%**	**96%**
Prim2	773	1382	462	26.2	**+40%**	**98%**
Bio	821	4657	531	94.3	**+35%**	**98%**
ind1	364	769	190	18.7	**+47%**	**97%**
ind2	2430	25132	1041	303	**+57%**	**99%**
ind3	2640	13131	1761	390	**+33%**	**97%**

Table C.23: 6 way iterative improvement clustering results

Circuit	**GA FLAT**		**GA CLUS**		**REDUCTION**	
	Cuts	Time	Cuts	Time	**Cuts**	**Time**
Chip1	25	3.5	26	1.6	**-3%**	54%
Chip2	22	3.1	18	1.6	**+18%**	48%
Chip3	8	2.3	9	1.3	**-11%**	43%
Chip4	10	2.7	9	1.3	**+10%**	51%
Prim1	81	9.8	79	6.1	**+3%**	37%
Prim2	190	36.3	183	28.5	**+4%**	21%
Bio	321	75	152	96.7	**+52%**	-
ind1	74	25.2	92	20.2	**-19%**	19%
ind2	572	166	360	266	**+37%**	-
ind3	930	222	458	545	**+50%**	-
Avg %					**14.1%**	**27.3 %**

Table C.24: 2 way genetic clustering results

Circuit	**GA FLAT**		**GA CLUS**		**REDUCTION**	
	Cuts	Time	Cuts	Time	**Cuts**	**Time**
Chip1	65	4.9	55	3.0	**+15%**	**38%**
Chip2	62	4.1	36	2.8	**+42%**	**31%**
Chip3	50	3.2	40	2.5	**+20%**	**21%**
Chip4	50	3.6	32	2.4	**+36%**	**33%**
Prim1	177	14.1	143	9.2	**+19%**	**34%**
Prim2	529	54.0	371	40.6	**+29%**	**26%**
Bio	1677	105	442	255	**+73%**	**-**
ind1	183	39.1	131	23.6	**+28%**	**41%**
ind2	2363	274	1139	464	**+51%**	**-**
ind3	2444	377	1568	902	**+36%**	**-**
Avg %					**+34.9%**	**+22.4%**

Table C.25: 4 way genetic clustering results

Circuit	**GA FLAT**		**GA CLUS**		**REDUCTION**	
	Cuts	Time	Cuts	Time	**Cuts**	**Time**
Chip1	86	6.8	72	5.7	**+17%**	**16%**
Chip2	79	5.4	45	5.3	**+43%**	**-**
Chip3	56	4.1	50	3.7	**+11%**	**10%**
Chip4	68	4.8	41	4.8	**+39%**	**-**
Prim1	189	18.6	164	17.2	**+13%**	**8%**
Prim2	671	71.1	547	63.2	**+18%**	**11%**
Bio	1759	142	609	255	**+65%**	**-**
ind1	427	52.2	253	39.1	**+40%**	**25%**
ind2	3342	368	1490	984	**+55%**	**-**
ind3	2890	495	2356	861	**+18%**	**-**

Table C.26: 6 way genetic clustering resulsts

Circuit	**TS FLAT**		**TS CLUS**		**REDUCTION**	
	Cuts	Time	Cuts	Time	**Cuts**	**Time**
Chip1	20	14.0	20	6.1	**0%**	**56%**
Chip2	14	10.3	14	5.3	**0%**	**48%**
Prim1	54	47.4	64	23.3	**-15%**	**51%**
Prim2	181	284	166	139	**+8%**	**51%**
Bio	127	321	160	200	**-20%**	**37%**
ind1	45	147	54	66.4	**-16%**	**55%**
ind2	323	2250	333	1080	**-3%**	**52%**
ind3	305	2822	399	1884	**-23%**	**33%**

Table C.27: 2 way Tabu Search clustering results

Circuit	**TS FLAT**		**TS CLUS**		**REDUCTION**	
	Cuts	Time	Cuts	Time	**Cuts**	**Time**
Chip1	47	23.5	48	16.2	**-2%**	**31%**
Chip2	29	19.2	30	13.8	**-3%**	**28%**
Prim1	102	100	93	55.2	**+8%**	**45%**
Prim2	357	299	309	206	**+13%**	**31%**
Bio	317	836	258	299	**+18%**	**64%**
ind1	121	382	86	100	**+29%**	**73%**
ind2	991	3902	628	1251	**+36%**	**67%**
ind3	1817	8668	1775	1971	**+3%**	**77%**

Table C.28: 4 way Tabu Search clustering results

Circuit	**TS FLAT**		**TS CLUS**		**REDUCTION**	
	Cuts	Time	Cuts	Time	**Cuts**	**Time**
Chip1	58	37	58	10.5	**0%**	**71%**
Chip2	37	25.5	41	11.9	**-9%**	**63%**
Prim1	139	78	129	51.6	**+7%**	**33%**
Prim2	562	435	459	165	**+18%**	**62%**
Bio	423	820	414	436	**+2%**	**62%**
ind1	193	819	147	228	**+24%**	**72%**
ind2	1375	5373	1047	560	**+23%**	**89%**
ind3	2801	10917	1775	1971	**+36%**	**81%**

Table C.29: 6 way Tabu Search clustering results

C.4 Results of Distributed Processing Methods

Circuit	**SEQUENTIAL**		**PARALLEL (10)**		**Xs**
	Cuts	Time	Cuts	Time	
Chip1	20	1.3	20	0.42	**3**
Chip2	15	0.9	15	0.2	**4**
Prim1	75	5.8	75	1.1	**5**
Prim2	243	33.9	243	4.6	**7**
bio	143	75	143	10.7	**7**
ind1	50	18.7	50	3.1	**6**
ind2	593	206	593	33.1	**6**
ind3	519	199	519	28.4	**7**

Table C.30: 2-way Distributed Local Search

Circuit	**SEQUENTIAL**		**PARALLEL (10)**		**Xs**
	Cuts	Time	Cuts	Time	
Chip1	54	2.0	54	0.5	**4**
Chip2	39	1.4	39	0.3	**5**
Prim1	155	7.4	155	1.2	**6**
Prim2	627	59.5	627	8.7	**7**
Bio	724	200	724	27.3	**7**
ind1	259	26.3	259	5.1	**5**
ind2	2179	995	2179	141.3	**7**
ind3	2101	481	2101	72.3	**7**

Table C.31: 4-way Distributed Local Search

Circuit	**SEQUENTIAL**		**PARALLEL (10)**		**Xs**
	Cuts	Time	Cuts	Time	
Chip1	77	2.4	77	0.5	**5**
Chip2	66	2.2	66	0.4	**6**
Prim1	181	9.2	181	1.1	**8**
Prim2	773	110	773	12.3	**9**
Bio	821	350	821	60.1	**6**
Ind1	364	60.5	364	7.4	**8**
Ind2	2486	1967	2486	280	**7**
Ind3	2689	747	2689	103	**7**

Table C.32: 6-way Distributed Local Search

Circuit	**SEQUENTIAL**		**PARALLEL (10)**		**Xs**
	Cuts	Time	Cuts	Time	
Chip1	20	68.7	20	7.6	9
Chip2	15	56.2	16	6.3	8.9
Prim1	76	185	75	22.4	8
ind1	67	484	66	74	6.5
Prim2	168	697	171	111	6.2
Bio	282	1982	284	237	8.3
ind2	333	2976	328	559	5.3
ind3	649	4007	645	504	7.9

Table C.33: 2-way Distributed Genetic Search

Circuit	**SEQUENTIAL**		**PARALLEL (10)**		**Xs**
	Cuts	Time	Cuts	Time	
Chip1	54	92.3	52	14	6.5
Chip2	45	78.5	48	9.2	8.5
Prim1	151	268	155	34.7	7.7
ind1	120	716	136	101	7.0
Prim2	403	1023	425	165	6.2
Bio	1164	3053	1146	352	8.6
ind2	1812	4984	1964	503	9.9
ind3	1593	6495			

Table C.34: 4-say Distributed Genetic Search

Circuit	**SEQUENTIAL**		**PARALLEL (10)**		**Xs**
	Cuts	Time	Cuts	Time	
Chip1	77	121	76	14.7	8.2
Chip2	66	104	67	17.1	6.1
Prim1	162	356	172	47.9	7.5
ind1	228	991	263	149	6.6
Prim2	564	1350	592	213	6.3
Bio	1549	4065	1158	478	8.5
ind2	2698	6789	2837	1136	5.9
ind3	2536	8417	2742	1033	8.1

Table C.35: 6-way Distributed Genetic Search

Publications Resulting from This Research

1. S. Areibi and A. Vannelli, S. Areibi and A. Vannelli, "A Combined Eigenvector Tabu Search Approach For Circuit Partitioning", In *Proceedings of the 1993 Custom Integrated Circuits Conference*, pp. 9.7.1 – 9.7.4, San Diego, 1993.

2. S. Areibi and A. Vannelli, "Circuit Partitioning Using A Tabu Search Approach", In *1993 IEEE International Symposium on Circuits and Systems*, pp. 1643–1646, Chicago, Illinois, 1993.

3. S. Areibi and A. Vannelli, "Advanced Search Techniques for Circuit Partitioning", In *DIMACS Series in Discrete Mathematics and Theoretical Computer Science*, pp. 77–98, Rutgers State University, New Jersey, 1993.

4. S. Areibi and A. Vannelli, "An Efficient Solution to Circuit Partitioning Using Tabu Search and Genetic Algorithms", In *6th International Conference of Micro Electronics*, pp. 70-74, Istanbul, Turkey, 1994.

5. S. Areibi and A. Vannelli, "Advanced Search Heuristics for Circuit Partitioning and Placement", To be submitted to : *Journal of Heuristics*, March, 1995.

6. S. Areibi and A. Vannelli, "Parallel Implementations of Advanced Search Techniques", To be submitted to : *IEEE Transactions On Parallel and Distributed Systems*, April, 1995.

7. S. Areibi and A. Vannelli, "An Efficient Clustering Technique for Circuit Partitioning", To be submitted to : *IEEE Transactions On VLSI Systems*, May, 1995.

Bibliography

[Agul76] B.J. Agule, P.k. Wolff, and M. Hanan, "Some Experimental Results on Placement Techniques," In *Proceedings of The* 13*th DAC*, pp. 214–224, IEEE/ACM, San Francisco, California, 1976.

[Arei93] S. Areibi and A. Vannelli, "A Combined Eigenvector Tabu Search Approach for Circuit Partitioning," In *Proceedings of The 1993 Custom Integrated Circuits Conference*, pp. 9.7.1 – 9.7.4, San Diego, 1993.

[Bane94] Prithviraj Banerjee, *Parallel Algorithms for VLSI Computer Aided Design*, Prentice Hall, Englewood Cliffs, New Jersey, 1994.

[Barn82] E.R. Barnes, "An Algorithm for Partitioning The Nodes of a Graph," *SIAM Journal of Algebraic and Discrete Methods*, vol. 3, No. 4, pp. 541–550, December 1982.

[Bert82] P.M. Bertolazzi and A.M Spaccamela, "Analysis of a Class of Graph Partitioning Problems," *R.A.I.R.O Theoretical Informatics*, vol. 16, pp. 255–261, 1982.

[Blan85] J.P. Blanks, "Near Optimal Quadratic Based Placement for a Class of IC Layout Problems," *IEEE Circuits and Devices*, vol. 1, No. 6, pp. 31–37, September, 1985.

[Blan91] J. Bland and G. Daswon, "Tabu Search and Design Optimization," *ORSA Journal of Computing*, vol. 23, No. 3, pp. 195–201, April 1991.

[Breu77] M.A. Breuer, "A Class of Min-Cut Placement Algorithms," In *Proceedings of The* 14*th DAC*, pp. 284–290, IEEE/ACM, New Orleans, Louisiana, 1977.

[Bui87] T.N. Bui, S. Chaudhuri, F.T. Leighton, and M. Sipser, "Graph Bisection Algorithms with Good Average Case Behavior," *Combinatorica*, vol. 7, No. 2, pp. 171–191, 1987.

[Bui89] T.N. Bui, "Improving The Performance of the Kernighan Lin and Simulated Annealing Graph Bisection," In *Proceedings of* 26*th DAC*, pp. 775–778, ACM/IEEE, Las Vegas, Nevada, June 1989.

[Carv89] C. Carvalho, D. Falco, and B. Apolloni, "Quantum Stochastic Optimization," *Stochastic Processes and Their Applications*, vol. 33, No. 2, pp. 233–244, 1989.

[Caso87] A. Casotto, F. Romeo, and A. Sangiovanni, "A Parallel Simulated Annealing Algorithm for The Placement of Macro-Cells," *IEEE Transactions on Computer-Aided Design*, vol. 6, No. 5, pp. 838–847, September 1987.

[Chan94] P.K. Chan, D.F. Schlag, and J.Y. Zien, "Spectral K-way Ratio-Cut Partitioning and Clustering," *IEEE Transactions on Computer Aided Design*, vol. 13, No. 9, pp. 1088–1096, 1994.

[Chen84] C. Cheng and E. Kuh, "Module Placement Based on Resistive Network Optimization," *IEEE Transaction on Computer Aided Design*, vol. 3, pp. 218–225, 1984.

[Chen92] C. Cheng, C. Yeh, and T. Lin, "A Probabilistic Multicommodity-Flow Solution to Circuit Clustering Problems," In *IEEE International Conference on CAD*, pp. 428–431, 1992.

[Coho91] J.P. Cohoon, S.U. Hegde, W.N. Martin, and D.S Richards, "Distributed Genetic Algorithms for The Floorplan Design Problem," *IEEE Transaction on Computer Aided Design*, vol. 10, No. 4, pp. 483–492, 1991.

[Cong91] J. Cong, L. Hagen, and A. Kahng, "Random Walks for Circuit Clustering," In *Proceedings IEEE International Conference on ASIC*, pp. 14.2.1–14.2.4, June 1991.

[Dare87] F. Darema, S. Kirkpatrick, and V.A. Norton, "Parallel Algorithms for Chip Placement By Simulated Annealing," *IBM Journal of Research and Development*, vol. 31, No. 3, pp. 391–402, May 1987.

[Davi88] L. Davis, *Genetic Algorithms and Simulated Annealing*, Morgan Kaufmann Publishers, Inc, Los Altos, California, 1988.

[Dona73] W.E. Donath and A.J. Hoffman, "Lower Bounds for The Partitioning of Graphs," *IBM Journal of Research and Development*, IBM, vol. 17, No. 5, pp. 420–425, 1973.

[Dona90] W. Donath, "Timing Driven Placement Using Complete Path Delay," In *Proceedings of The 27th DAC*, pp. 84–89, IEEE/ACM, Orlando, Florida, 1990.

[Feo94] T. Feo, M. Resende, and S. Smith, "A Greedy Randomized Adaptive Search Procedure for The Maximum Independent Set," *Journal of Operations Research*, vol. 42, pp. 860–878, 1994.

[Fidu82] C.M. Fiduccia and R.M. Mattheyses, "A Linear-Time Heuristic for Improving Network Partitions," In *Proceedings of 19th DAC*, pp. 175–181, ACM/IEEE, Las Vegas, Nevada, June 1982.

[Fran86] J. Frankle and R.M. Karp, "Circuit Placement and Cost Bounds By Eigenvector Decomposition," In *IEEE International Conference on CAD*, pp. 414–417, Santa Clara, California, 1986.

[Garb90] J. Garbers, H.J. Promel, and A. Steger, "Finding Clusters in VLSI Circuits," In *IEEE International Conference on CAD*, pp. 520–523, 1990.

[Gill81] P.E. Gill, W. Murray, and M.H. Wright, *Practical Optimization*, Academic Press, New York, New York, 1981.

[Glov90] F. Glover, "Tabu Search Part II," *ORSA Journal on Computing*, vol. 2, No. 1, pp. 4–32, 1990.

[Goon92] S. Goonatilake and S. Khebbal, "Intelligent Hybrid Systems," 1992 To Appear In: Proceedings of the First Singapore International Conference on Intelligent Systems.

[Goto86] S. Goto and T. Matsuda, "Partitioning, Assignment and Placement," In *Layout Design and Verification*, pp. 99–132, North Holland, 1986.

[Hach89] G. Hachtel and C. Morrison, "Linear Complexity Algorithms for Hierarchical Routing," *IEEE Transactions on Computer Aided Design*, vol. 8, No. 1, pp. 64–80, 1989.

[Hadl92] S.W. Hadley, B.L. Mark, and A. Vannelli, "An Efficient Eigenvector and Node Interchange Approach for Finding Netlist Partitions," *IEEE Transactions on CAD/ICAS*, vol. 11, No. 7, pp. 885–892, July 1992.

[Hage92] L. Hagen and A.B. Kahng, "A New Approach to Effective Circuit Clustering," In *IEEE International Conference on CAD*, pp. 422–427, 1992.

[Harw79] Harwell, *MA31A*, Computer Science Department, University of Texas, Austin, 1979.

[Hert87] A. Hertz and D. Werra, "Using Tabu Search Technique for Graph Coloring," *Computing*, Springer Verlag, vol. 39, No. 2, pp. 345–351, 1987.

[HU85] T.C. HU and E. Kuh, "Theory and Concepts of Circuit Layout," In *VLSI Circuit Layout:Theory and Design*, pp. 3–18, IEEE PRESS, New York, 1985.

[Hyaf73] L. Hyafil and R.L. Rivest, "Graph Partitioning and Constructing Optimal Decision Trees Are Polynomial Complete Problems," Technical Report, No. 33, IRIA-Laboria, Rocquencourt, France, 1973.

[Inc93] CPLEX Optimization Inc, *CPLEX Documentation*, CPLEX Optimization, Inc, Tahoe, Nevada, 1993.

[Klei91] J. Kleinhans, G. Sigl, F. Johannes, and K. Antreich, "GORDIAN: VLSI Placement By Quadratic Programming and Slicing Optimization," *IEEE Transaction on Computer Aided Design*, vol. 10, No. 3, pp. 356–365, March 1991.

[Kozm88] K.A. Kozminski and E. Kinnen, "Rectangular Dualization and Rectangular Dissections," *IEEE Transactions on Circuits and Systems*, vol. 35, No. 11, pp. 1401–1416, 1988.

[Kram84] M.R. Kramer and J.V. Leuwen, "The Complexity of Wire Routing and Finding Minimum Area Layouts for Arbitrary VLSI Circuits," *Advances in Computing Research*, JAI Press, vol. 2, pp. 129–146, 1984.

[Krav87] S.A. Kravitz and R.A. Rutenbar, "Placement By Simulated Annealing on a Multiprocessor," *IEEE Transactions on Computer-Aided Design*, vol. 6, No. 4, pp. 534–549, July 1987.

[Kris84] B. Krishnamurthy, "An Improved Min-Cut Algorithm for Partitioning VLSI Networks," *IEEE Transactions on Computers*, vol. 33, No. 5, pp. 438–446, May 1984.

[Kuh83] E.S. Kuh, "The Special Issue on Routing and Micro-electronics," *IEEE Trans CAD of IC*, vol. CAD-2, No. 4, , October 1983.

[Laar88] P.M Van Laarhoven and E.h.L. Aarts, *Simulated Annealing: Theory and Applications*, Kluwer Academic Publishers, Boston, 1988.

[Laut79] U. Lauther, "A Min-Cut Placement Algorithm for General Cell Assemlier Based on a Graph Representation," In *Proceedings of The 16th DAC*, pp. 1–10, IEEE/ACM, San Diego, California, 1979.

[Leig83] F.T. Leighton, *Complexity Issues in VLSI*, MIT Press, Cambridge Ma, 1983.

[Leng90] T. Lengauer, *Combinatorial Algorithms for Integrated Circuit Layout*, John Wiley & Sons, New York, 1990.

[Mich92] Z. Michalewicz, *Genetic Algorithms + Data Structures = Evolution Programs*, Springer-Verlog, Berlin, Heidelberg, 1992.

[Mowc87] J.T. Mowchenko and C.S. Ma, "A New Global Routing Algorithm for Standard Cell ICs," In *IEEE International Symposium on Circuits and Systems*, pp. 27–30, IEEE, Philadelphia,Pa, 1987.

[Naha85] S. Nahar, S. Sahni, and E. Shragowitz, "Experiments with Simulated Annealing," In *Proceedings of The 22nd DAC*, pp. 748–752, IEEE/ACM, Las Vegas, Nevada, 1985.

[Otte82] R.H.J.M. Otten, "Eigen Solutions in Top-Down Layout Design," In *IEEE International Symposium on Circuits and Systems*, pp. 1017–1020, IEEE, Philadelphia,Pa, 1982.

[Pate77] A.M. Patel and K.H. Khokhani, "The Chip Layout Problem: A Placement Procedure," In *Proceedings of The 14th DAC*, pp. 291–297, IEEE/ACM, New Orleans, Louisiana, 1977.

[Pill88] L. Pillage and R. Rohrer, "A Quadratic Metric with a Simple Solution Scheme for Initial Placement," In *Proceedings of The* 25*th DAC*, pp. 324–329, IEEE/ACM, Anaheim, California, 1988.

[Prea88] B. Preas, *Physical Design Automation of VLSI Systems*, Benjamin/cummings Publishing Company, Inc, Menlo Park, California, 1988.

[Reev93] C.R. Reeves, *Modern Heuristic Techniques for Combinatorial Problems*, John Wiley & Sons, Inc, New York, 1993.

[Robe87] K. Roberts and B. Preas, "Physical Design Workshop 1987," MCNC, Technical Report, MCNC, Marriott's Hilton Head Resort,South Carolina, April 1987.

[Rose88] J.S. Rose, W.M. Snelgrove, and Z.G. Vranesic, "Parallel Standard Cell Placement Algorithms with Quality Equivalent to Simulated Annealing," *IEEE Transactions on Computer-Aided Design*, vol. 7, No. 3, pp. 387–396, March 1988.

[Sanc89] L.A. Sanchis, "Multiple-Way Network Partitioning," *IEEE Transactions on Computers*, vol. 38, No. 1, pp. 62–81, January 1989.

[Schw76] D.G. Schweikert, "A 2-dimensional Placement Algorithm for The Layout of Electrical Circuits," In *Proceedings of The* 13*th DAC*, pp. 408–416, IEEE/ACM, San Francisco, California, 1976.

[Sech86] C. Sechen and A. Sangiovanni, "The TimberWolf 3.2: A New Standard Cell Placement and Global Routing Package," In *Proceedings of The* 23*rd DAC*, pp. 432–439, IEEE/ACM, Las Vegas, Nevada, June 1986.

[Sech88a] C. Sechen, *VLSI Placement and Global Routing Using Simulated Annealing*, Kluwer Academic Publishers, Boston, 1988.

[Sech88b] C. Sechen and D. Chen, "An improved Objective Function for Min-Cut Circuit Partitioning," In *Proceedings of ICCAD*, pp. 502–505, San Jose, California, 1988.

[Sha85] L. Sha and R.W. Dutton, "An Analytical Algorithm for Placement of Arbitrarily Sized Rectangular Blocks," In *Proceedings of The* 22*nd DAC*, pp. 602–608, IEEE/ACM, Las Vegas, Nevada, 1985.

[Shil79] Y. Shiloach, "A Minimum Linear Arrangement Algorithm for Undirected Trees," *SIAM Journal of Computing*, vol. 8, pp. 15–32, 1979.

[Shin93] H. Shin and C. Kim, "A Simple Yet Effective Technique for Partitioning," *IEEE Transactions on VLSI Systems*, vol. 1, No. 6, pp. 380–386, September 1993.

[SK90] J. Skorun-Kapov, "Tabu Search Applied to The Quadratic Assignment Problem," *ORSA Journal on Computing*, vol. 2, pp. 195–202, 1990.

[Song92] L. Song and A. Vannelli, “A VLSI Placement Method Using Tabu Search Technique,” *Micro-electronics Journal*, vol. 23, pp. 167–172, April, 1992.

[Suar88] P. Suaris and G. Kedem, “An Algorithm for Quadrisection and Its Application to Standard Cell Placement,” *IEEE Transaction on Circuits and Systems*, vol. 35, pp. 294–303, March 1988.

[Tao91] L. Tao, Y.C. Zhao, K. Thulasiraman, and M.N.S. Swamy, “An Efficient Tabu Search Algorithm for Graph Bi-sectioning,” In *Proceedings/ Great Lakes Symposium on VLSI*, pp. 92–95, IEEE, 1991.

[Ueda86] K. Ueda, R. Kasai, and T. Sudo, “Layout Strategy, Standardization, and CAD Tools,” *Layout Design and Verification*, North-Holland, pp. 1–54, 1986.

[Ullm84] J.D. Ullman, *Computational Aspects of VLSI*, Computer Science Press Inc, Rockville, Maryland, 1984.

[Vann89] A. Vannelli, “An Interior Point Method for Solving the Global Routing Problem,” In *IEEE Custom Integrated Circuits Conference*, pp. 1–22, IEEE, San Diego, California, 1989.

[Vann90] A. Vannelli and S.W. Hadley, “A Gomory-Hu Cut Tree Representation of a Netlist Partitioning Problem,” *IEEE Transactions on Circuits and Systems*, vol. 37, No. 9, pp. 1133–1139, September, 1990.

[Venk91] V. Venkatasubramanian and I.P. Androulakis, “A Genetic Algorithm Framework for Process Design and Optimization,” *Computers Chemical Engineering*, vol. 15, No. 4, pp. 217–228, 1991.

[Wei89] Y.C. Wei and C.K. Cheng, “Toward efficient hierarchical designs by ratio cut partitioning,” In *IEEE International Conference on CAD*, pp. 298–301, Santa Clara, California, 1989.

[Whit84] S.R. White, “Concepts of Scale in Simulated Annealing,” In *IEEE Int Conf on Computer Design*, pp. 646–651, IEEE, November 1984.

[Widm86] P. Widmayer, “A Faster Approximation Algorithm for The Steiner Problem in Graphs,” *Acta Informatica*, vol. 23, pp. 223–229, 1986.

[Yeh94] C.W. Yeh, C.K. Cheng, and T.T. Lin, “A General Purpose Multiple-Way Partitioning Algorithm,” *IEEE Transactions on Computer Aided Design*, vol. 13, No. 12, pp. 1480–1488, 1994.

[Yeh95] C.W. Yeh, C.K. Cheng, and T.T. Lin, “Optimization by Iterative Improvement: An Experimental Evaluation on Two-Way Partitioning,” *IEEE Transactions on Computer Aided Design*, vol. 14, No. 2, pp. 145–153, 1995.

[Zhan89] X. Zhang, L. Pillage, and R. Rohrer, “Efficient Final Placement Based on Nets-As-Points,” In *Proceedings of The 26th DAC*, pp. 578–581, IEEE/ACM, Las Vegas, Nevada, 1989.