Technical Report 2001-02P(2001/5/17)

A Review of Circuit Partitioning

Shawki Areibi[1]

School of Engineering
University of Guelph,
Albert Thornbrough Bld #159
Guelph, Ontario,
Canada, N1G 2W1

[1]**sareibi@uoguelph.ca**

SOE Technical Reports

This technical report series allows faculty of the School of Engineering at the University of Guelph to publish detailed and recent research results in a timely manner. It is *not* intended that these technical reports duplicate outside publications. However, due to the time lag in publishing results in formal, peer reviewed venues, many of these technical reports will be submitted for review and publication elsewhere. In such cases, it is intended that the technical reports will contain additional details and results that cannot be included elsewhere due to space limitations.

In addition to technical reports pertaining to research conducted within the School of Engineering, the technical report series may also be used to publish "pedagogical" results and methods. Guelph has a strong tradition and committment to high-quality teaching and teaching methods. Many of our faculty are actively engaged in developing new pedagogical techniques, including the use of multi-media and Web-based tools for instructional purposes. We believe that it is equally important to make these results available to the academic and education community.

While all reports will be numbered sequentially, a research report will be identified by the technical report number and the code **R**. Likewise, a pedagogical report will be identified by the technical report number followed by the code **P**.

For more information about this technical report series, please contact:
Shawki Areibi sareibi@uoguelph.ca

Publication History

Any complex system, consisting of a large number of components, cannot be designed efficiently without decomposing it into a set of smaller subsystems. After the decomposition, each subsystem can be designed independently and simultaneously to speed up the design process. Partitioning [1] plays a key role in the design of a computer system in general and VLSI chips in particular. A computer system consists of tens of millions of transistors and to ease the design process, it is partitioned into several smaller modules/blocks. The essence of netlist partitioning is to divide a system specification into clusters such that th number of intercluster connections is minimized. This technical report describes research directions in netlist partitioning during the past decade or so, in terms of both problem formulation and solution apporaches. We discuss the traditional min-cut and ratio cut bipartitioning formulations along with multi-way extensions and newer problem formulations

Contents

List of Figures

List of Tables

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 [2]. 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 [3].

1.1 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 [4]. Currently, the popular VLSI physical design styles are *gate-arrays*, *standard-cells*, *general-cells*, and *full-custom design*.

1.1.1 Gate-Array Design

In gate-array design, the entire wafer is prefabricated with an array of identical gates or cells. As shown in Figure 1a, 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.

1.1.2 Standard-Cell Design

In standard-cell layout as seen in Figure 1b, 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.

1.1.3 General-Cell Design

As seen in Figure 1c, 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.

1.1.4 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 constraints makes synthesis tools difficult to develop. Full-custom design as seen in Figure 1d 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 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, after swapping modules between the

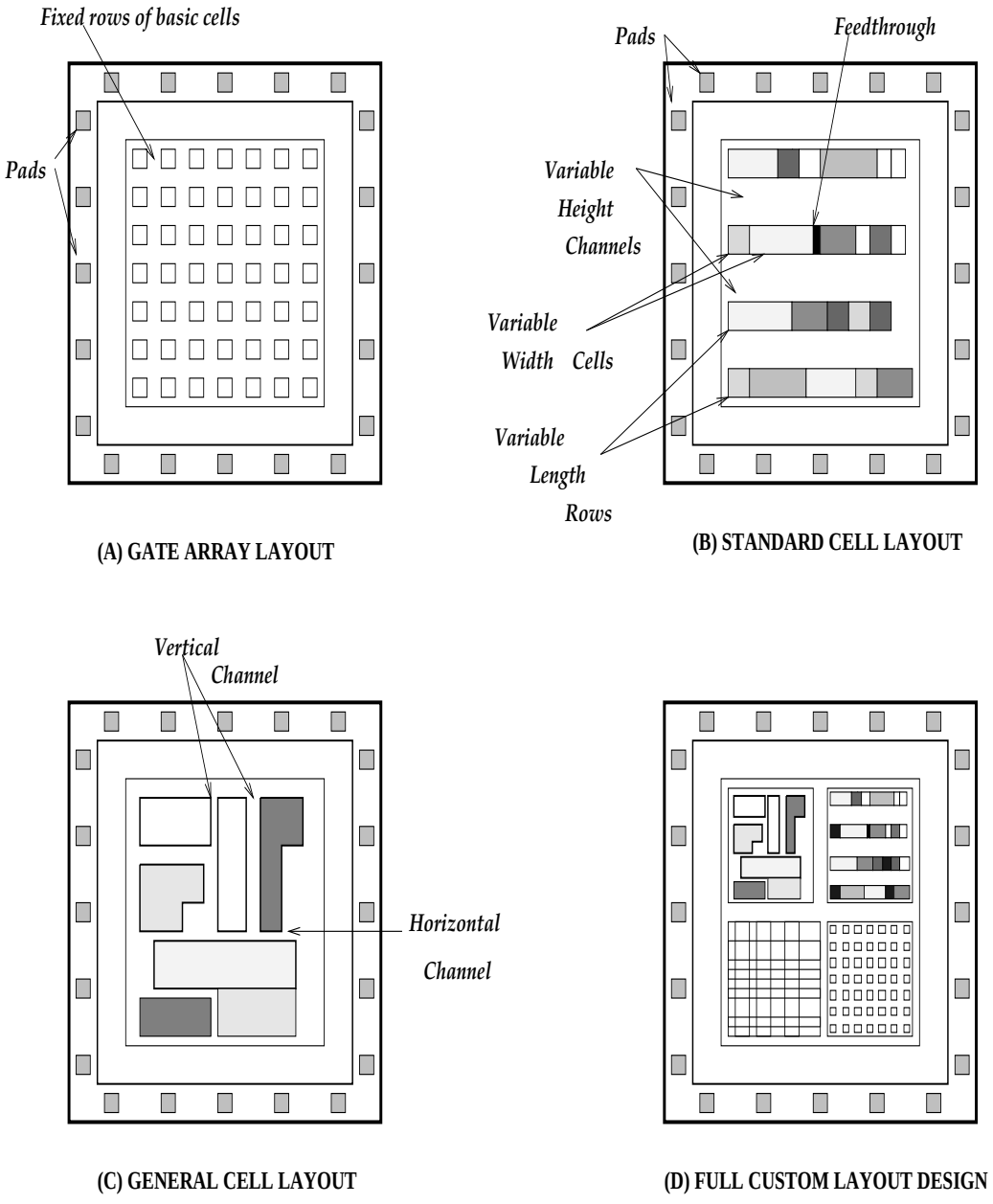


Figure 1: Layout styles

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 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.

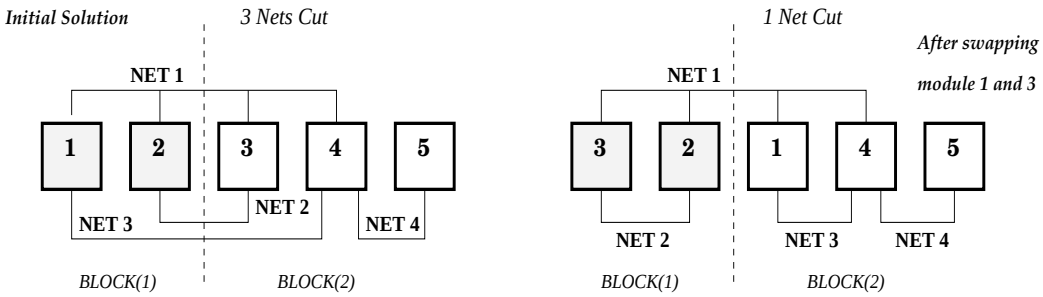


Figure 2: Illustration of circuit partitioning

2.1 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 $\mathbf{A}$ represent the connection matrix (i.e., $\mathbf{A}$ 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

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

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

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

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

Barnes [5] 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 [6, 7], 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 **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 [5] shows that the solution of the following *linear transportation problem* gives an approximate solution to the graph partitioning problem:

$$Max \sum_{i=1}^{n_m} \sum_{k=1}^{n_b} \frac{u_{ik}}{\sqrt{m_k}} x_{ik} \tag{2}$$

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:

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

For the special case of partitioning into two blocks, the above computation can be further simplified. The first two eigenvalues of the connection matrix **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 [6].

2.2 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;

$$Max \quad \sum_{j=1}^{n_n} \sum_{k=1}^{n_b} y_{jk} \tag{3}$$

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:

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

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.

2.3 Lower Bounds

The eigenvector technique of Barnes [5] 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. [6] and [8] 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 [8]:

$$E_c \geq -\frac{1}{2}\sum_{i=1}^{k} m_i \lambda_i(\mathbf{A}+\mathbf{U}), \tag{4}$$

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 $\mathbf{A}+\mathbf{U}$.

2.4 Complexity of Circuit Partitioning

At the basis of all partitioning problems are variations of the following combinatorial problem.
Hypergraph Partitioning [9]

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:	$c(V_1, \ldots, V_m) =$ $\sum_{e \in E}(\lvert\{i \in \{1, \ldots, m\} \rvert V_i \cap e \neq \phi\}\rvert - 1)l(e)$

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 $w \equiv 1, l \equiv 1$ [10]. The problem is only weakly NP-complete if G is restricted to be a tree [11]. 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 [12], then the problem is in **P**.

Currently none of the results cited above has practical impact on the VLSI layout procedures [9]; the instance of the *hypergraph partitioning* problem that occur in circuit layout go beyond the graph classes considered in the restrictions of the problem.

3 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 [12]. 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.

3.1 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 [13]. 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.

3.2 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.*

3.2.1 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 [14].

Kernighan and Lin [15] 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 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 3. fig:kir-lin

The cells to be switched are chosen so that a maximum decrease in cut-set size 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 [16] 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 [17] 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 [18], 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

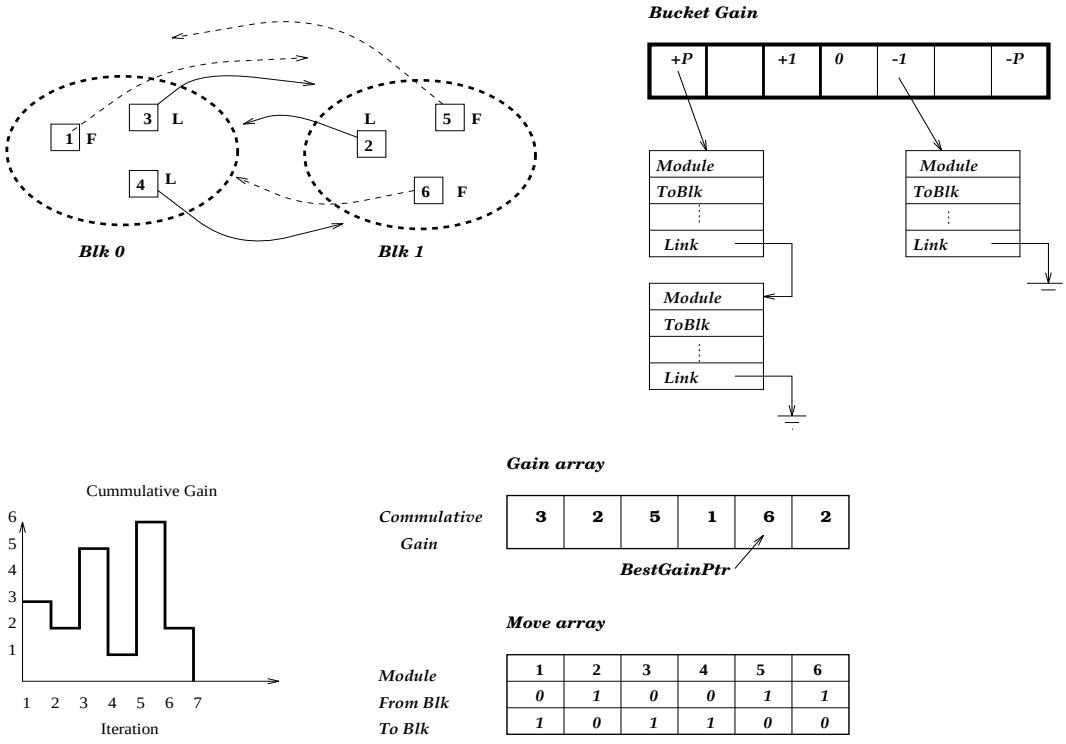


Figure 3: Iterative improvement example based on node interchange

```
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 4: The Kernighan-Lin Algorithm

local in nature and get easily trapped in local minimum.

Recently, several authors reported a ratio cut [19] 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 $\{(u,w) \in E | u \in U \; and \; w \in W\}$). 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 [20].

3.3 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* [21, 22], *Quantum Annealing* [23], and *Genetic Algorithms* [24] have been heralded as new and powerful randomized methodologies for combinatorial problems. Some of these heuristics are introduced in detail in the next chapter.

3.4 Summary

In this Section 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 Section, 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.

4 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 [22] 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 [25], 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 Section, 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 4.1 introduces the main benchmarks that are used to evaluate the different heuristics used in this dissertation. Sections 4.2-4.6 introduce the main concepts of the advanced search heuristics.

4.1 Benchmarks

Some of the benchmarks used throughout this document to evaluate the performance of the hybrid algorithm are presented in Table 1. Chip1-Chip4 circuits are taken from the work of Fiduccia & Mattheyses [16]. The rest are taken from the MCNC gate array and standard cell test suite benchmarks [26]. As seen in the table these netlists (hypergraphs) vary in size from 200 to 15000 nodes and 300 to 20000 nets. Tables 1-2 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. Clustering based heuristics may utilize 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 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 2: Statistical information of benchmarks

Secondly, the statistical information is used as a means to improve the performance of the MIP CPLEX solver [27] in reducing the computation time of the branch and bound heuristic. In addition, the information in Tables 1-2 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 **??**. 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 [20, 19, 28, 29, 30, 31] with ours that are based on equi-sized partitions.

4.2 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 [6, 32, 33]. Sechen [34] 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 Section 3, the main concepts of the Sanchis [18] 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 5 the heuristic continues to explore new regions until either cycling occurs or a certain number of passes have elapsed. Figure 6a 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. Figure 6b 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 7, 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 4.6.4) 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 reason, this heuristic is generally use for clustering as a means to refine the solutions produced by the clustering techniques developed, due to its simplicity and fast convergence.

4.3 Greedy Randomized Adaptive Search

GRASP is a greedy randomized adaptive search procedure that has been successful in solving many combinatorial optimization problems efficiently [35]. The GRASP methodology

```
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 5: A dynamic hill climbing heuristic (SDHC)

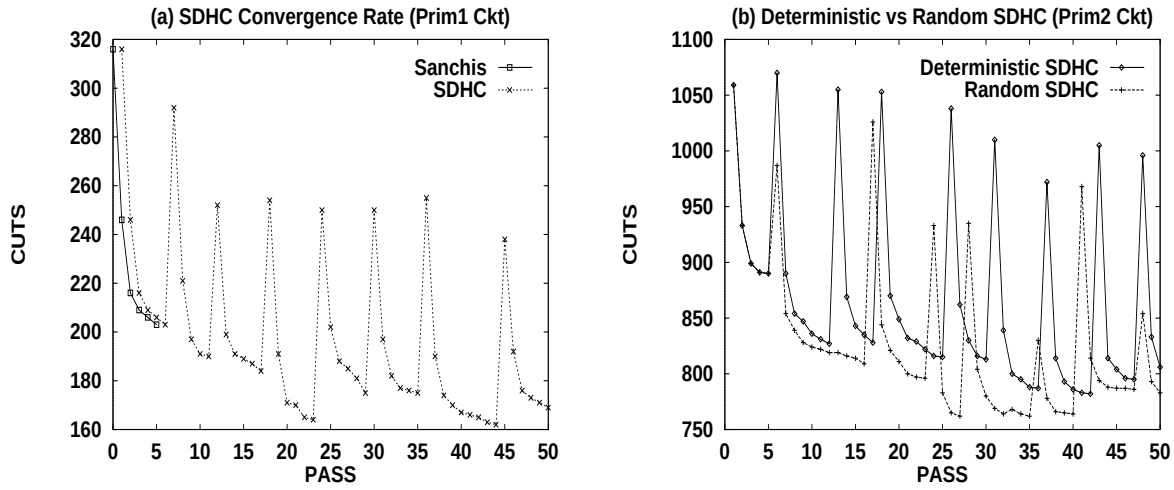


Figure 6: The convergence of SDHC

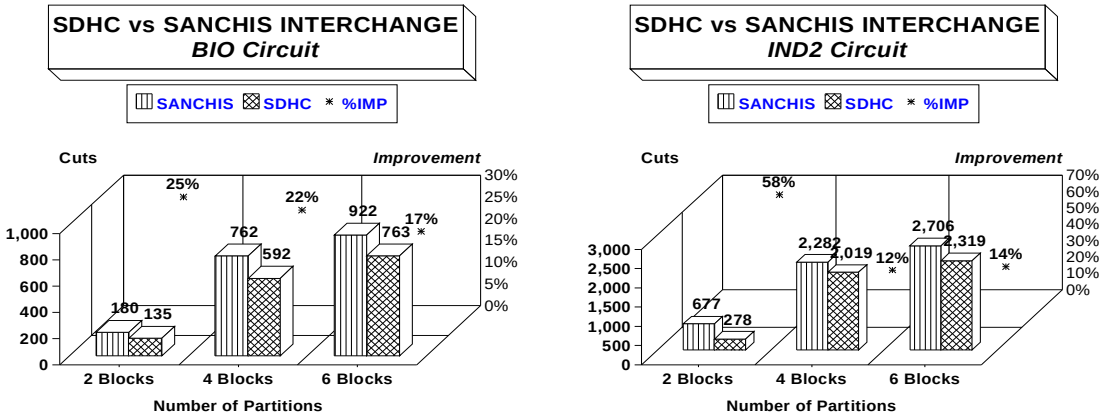


Figure 7: The performance of SDHC

was developed in the late 1980s, and the acronym was coined by Feo [36]. 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.

4.3.1 Implementation

Figure 8 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 8b 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 8c. A more sophisticated local search based on Tabu Search can be implemented instead of the simple local search procedure.

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 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)

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 8: GRASP (Greedy Adaptive Search)

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 9a[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 9b 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 [35] two different restrictions are imposed on the RCL, Value Restriction (*RCL-VR*), and Cardinality Restriction (*RCL-CR*). In RCL-VR a module is allowed to be in the restricted

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

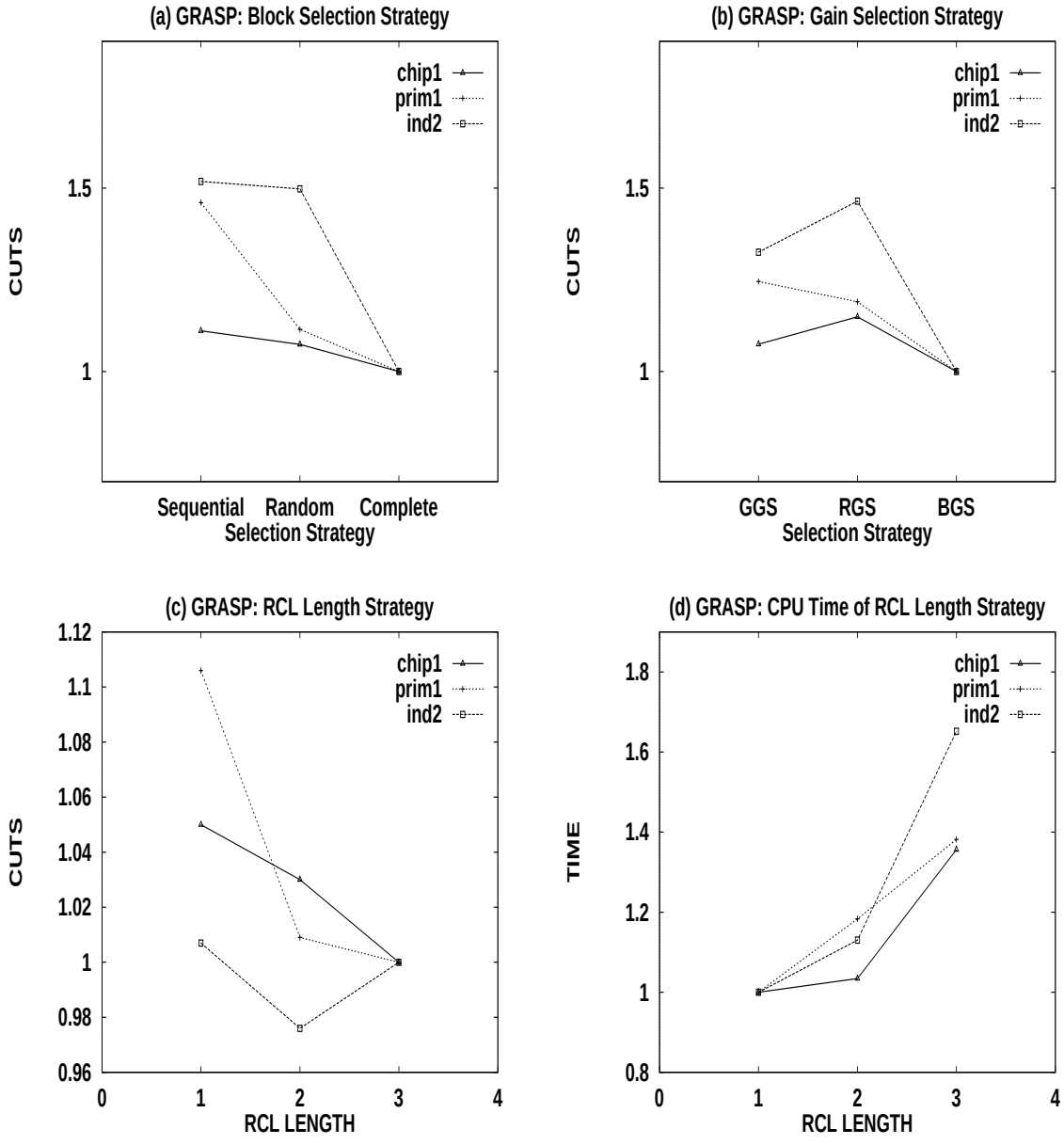


Figure 9: Parameters affecting GRASP performance

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 9c,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 assess the performance of Sanchis Interchange heuristic, to that of the GRASP. It is clear from Table 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: A comparison between GRASP and Sanchis interchange heuristic

4.4 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 [37]. The algorithms are called genetic because the manipulation of possible solutions resembles the mechanics of natural selection. These algorithms which were introduced by Holland [38] 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.

4.4.1 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*.

Characteristics of Genetic Search

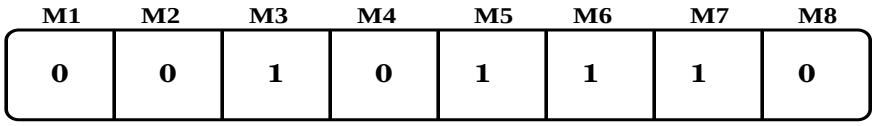
There are many characteristics of Genetic Algorithms which qualify them to be a robust based search procedure. The first feature of Genetic Algorithms is that they are characterized to climb many peaks in parallel. Thus, the probability of finding a false peak is reduced over methods that proceed form 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 [24].

Main Components of Genetic Search

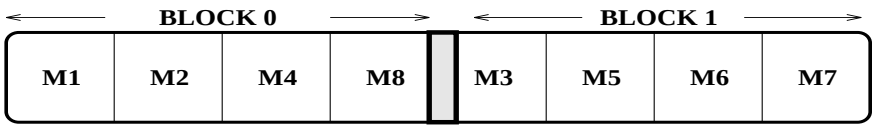
There are essentially four basic components necessary for the successful implementation of a Genetic Algorithm. At the outset, there must be a code or scheme that allows for a bit string representation of possible solutions to the problem. Next, a suitable function must be devised that allows for a ranking 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.

Representation Module

In the original GA's of Holland [38], 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 10a, 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 [39]. 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 4.4.2. The second representation scheme is shown in Figure 10b. 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 [39], but requires more computation time due to the complexity of the unary operator involved.



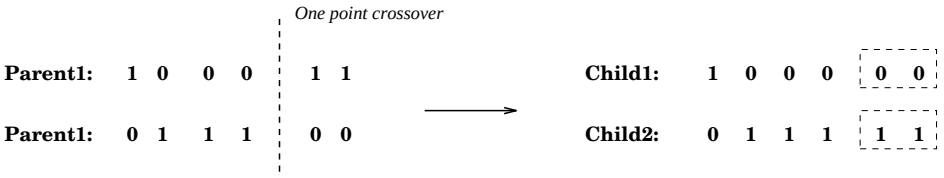
(a) Group Number Encoding



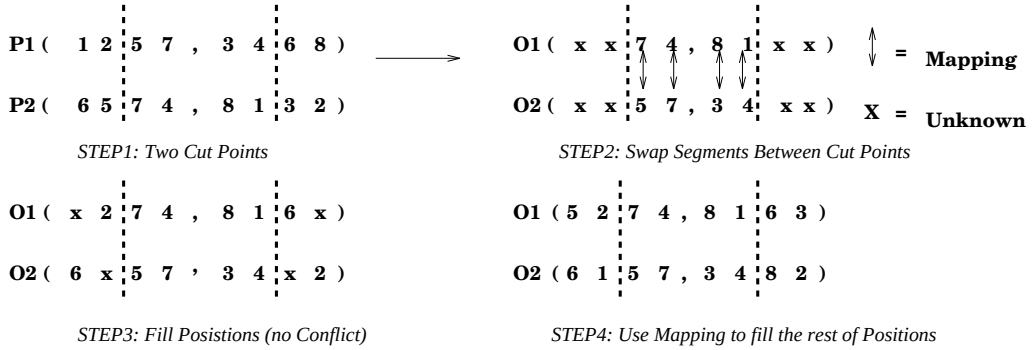
(b) Permutation with Separator Encoding.

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	.005	.840	0	1	1	0	0
0	0	1	0	.760	.473	.894	.001	1	0	0	1	1

(c) Standard Mutation Operator



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



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

Figure 10: Representation schemes and genetic operators

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.

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 10c is simply the introduction 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 10d 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) [39]. As shown in Figure 10e, 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 10e 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 [39].

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.

4.4.2 GA Implementation

Figure 11 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.

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 12 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

```
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 ≥ gen
      Evaluate Fitness
      For (i=1 to pop_size)
        Select (mate1,mate2)
        if (rnd(0,1) ≤ cross_rate)
          child = Crossover(mate1,mate2);
        if (rnd(0,1) ≤ 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 11: A generic Genetic Algorithm

fundamental 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.

Performance of Genetic Algorithm

Section 4.4.1 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

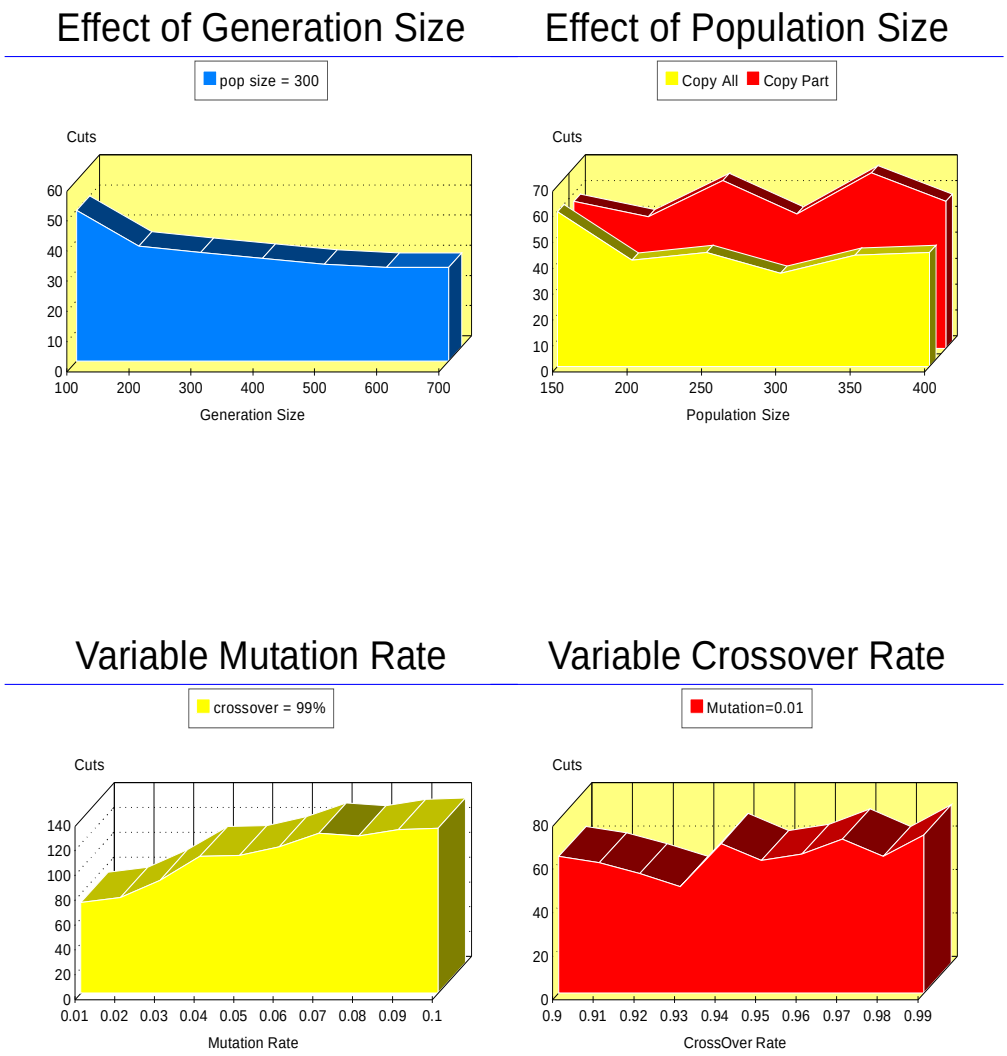


Figure 12: Parameters affecting GA performance

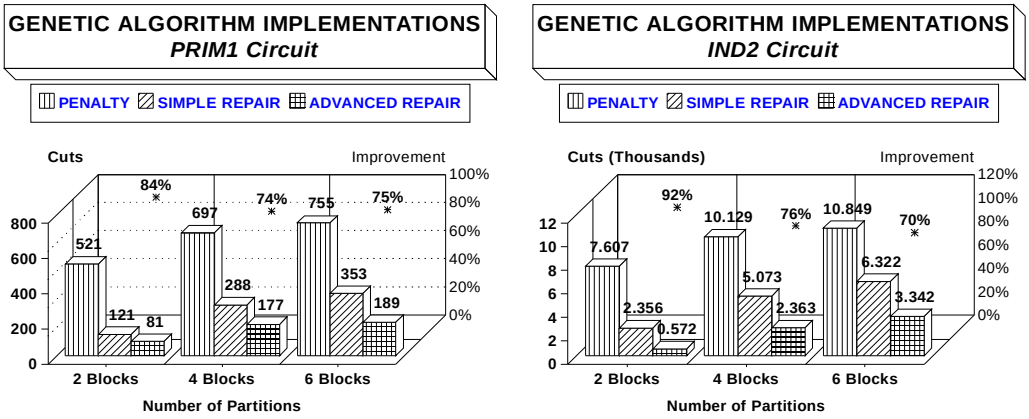


Figure 13: The GA performance

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 13 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 13 also presents the improvement achieved from using the advanced repair heuristic over the penalty function implementation.

Comparison of different Constructive Methods

In this section, the performance of the Genetic Algorithm is compared to results obtained by Barnes's eigenvector technique [5] and the GRASP heuristic [36]. The comparison is based on good initial solutions obtained by each heuristic and the computation time involved to obtain these solutions. Table 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 4: A comparison between constructive techniques

4.5 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 [22]. 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 14. 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.

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* [40]. The choice of annealing schedule has an important bearing on the performance of the algorithm [41].

```
current_solution ← initial_solution
current_cost ← evaluate(current_solution)
T ← T_initial
While (T ≥ T_final)
        for i = 1 to iteration(T) /* neighborhood moves */
                new_solution ← move(current_solution)
                new_cost ← evaluate(new_solution)
                Δcost ← new_cost - current_cost
                if(Δcost ≤ 0  OR  e^(-Δcost/T) > random())
                        /* accept new solution */
                        current_solution ← new_solution
                        current_cost ← new_cost
                EndIf
        EndFor
        T ← next_temp(T)
EndWhile
```

Figure 14: A Simulated Annealing Algorithm

4.5.1 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 [42]:

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 [42]. 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

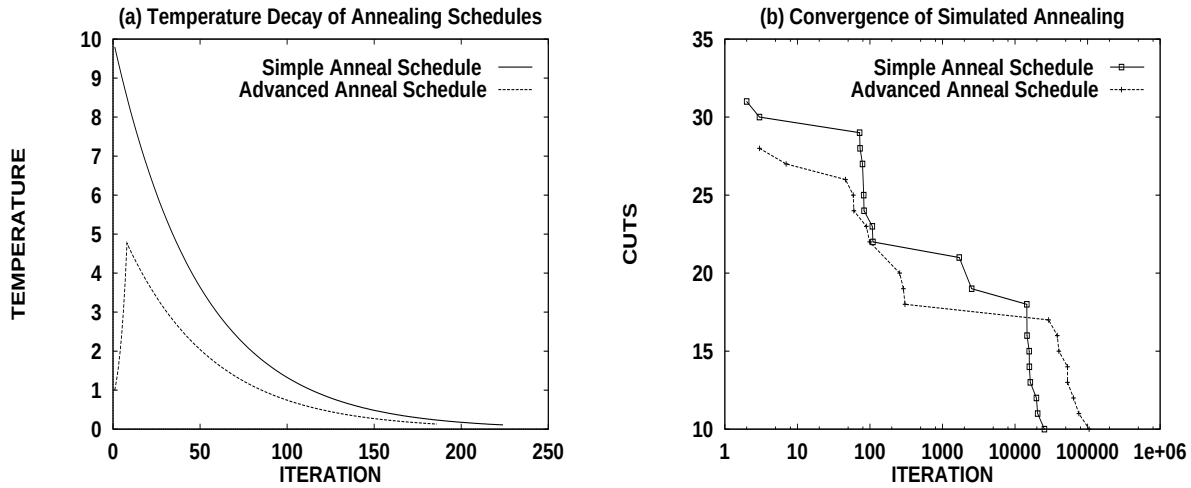


Figure 15: Simulated Annealing with different schedules

of proposed transitions, is less than a given value χ_0 (in [22] $\chi_0 = 0.8$), 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.

Advanced Annealing Schedule

A more advanced annealing schedule proposed by White [43] 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 15, a comparison between the simple cooling schedule proposed by Kirkpatrick [22] and White [43] is shown. Figure 15a illustrates the means of computing the initial temperature in both schedules. Figure 15b shows the convergence rate for a small circuit based on both schedules. Figure 16

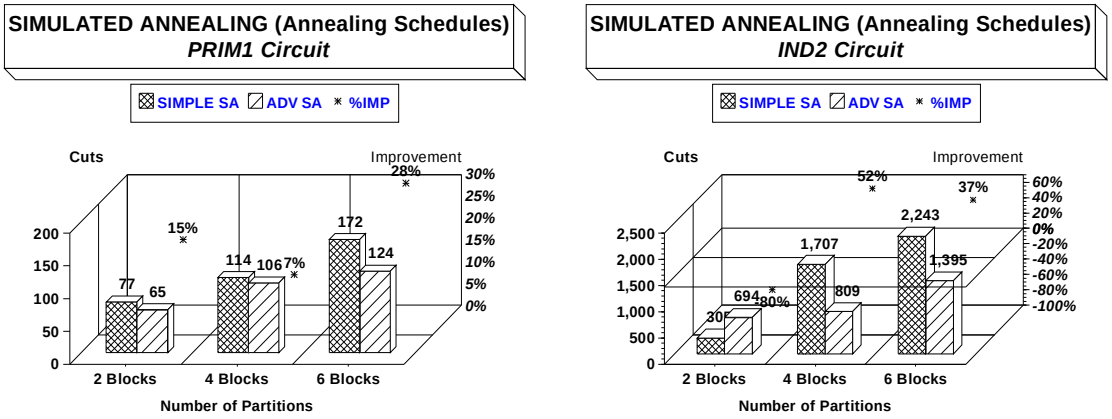


Figure 16: A comparison between annealing schedules

presents results obtained by the two annealing schedules mentioned above. The performance of the Simulated Annealing using the advanced annealing schedule proposed by White [43] 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.

4.6 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 [32] [33] [44], graph coloring [45], to quadratic assignment problems [46]. 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.

4.6.1 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.

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.

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 17a 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 17b 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 4.6.2.

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 17c 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.

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 [47, 48]. Combined with the short term memory functions, intermediate and long term memory functions provide an interplay between "exploitation" and "exploration" of the solution space [47] (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

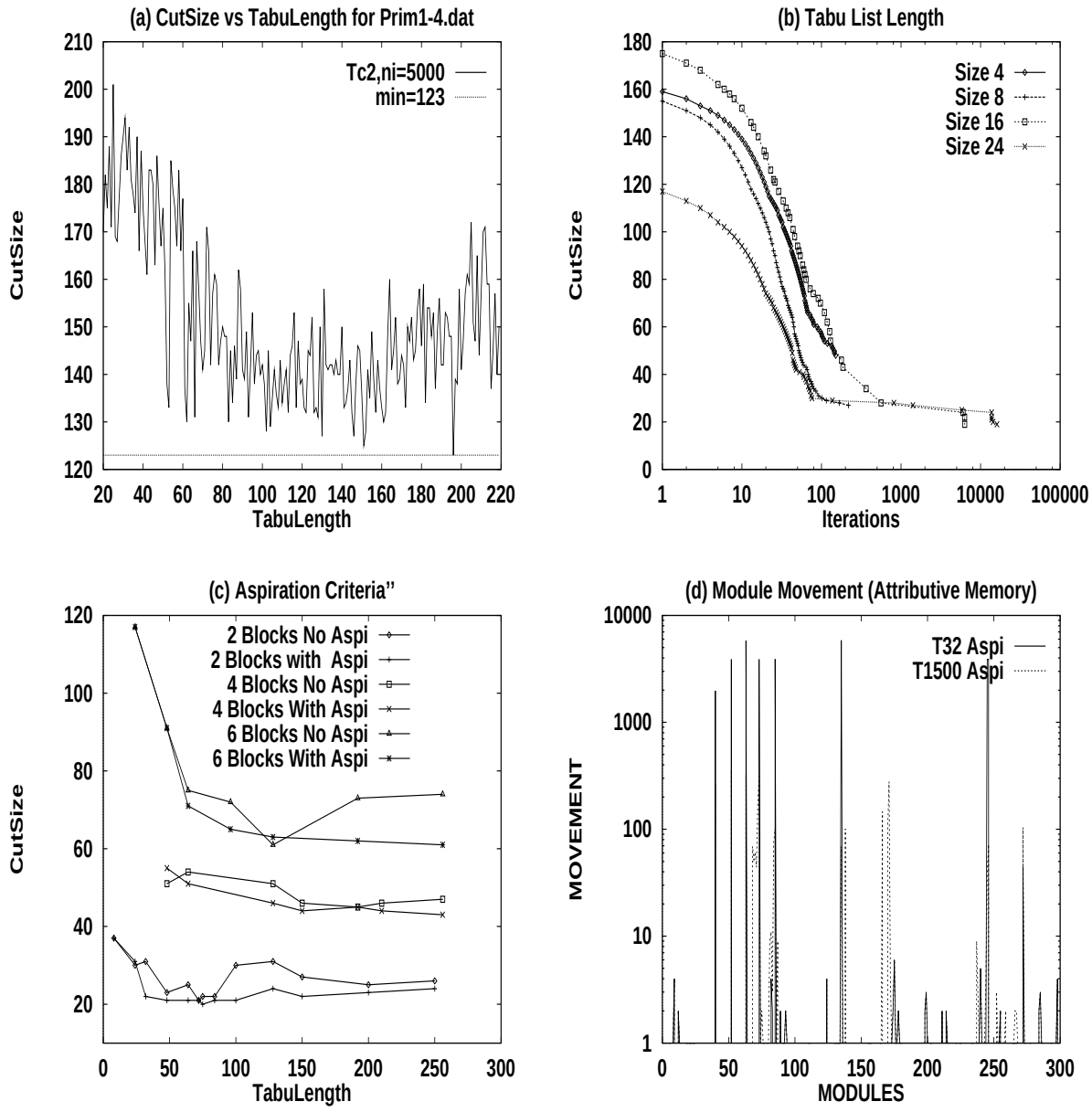


Figure 17: Parameters affecting Tabu Search

principles that are roughly the reverse of those for intermediate term memory. Figure 17d shows a recording of the module movement during the search procedure.

4.6.2 Tabu Search Implementation for Partitioning

The Tabu Search routine described so far can be formulated as shown in Figure 18. 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.

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.

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. 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.

Input:
The net list or the Graph G= (V,E)
K = partitions required; $\mid T \mid$ = 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 18: A Simple Tabu Search implementation

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 Section 3) 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 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 4.6.1). 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.

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 5: A comparison between different Tabu Search settings

4.6.3 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, 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 4.6.1 and 4.6.2 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.

ATS Implementation

In Section 4.6.2, 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 [48]. For example, in the Reverse Elimination Method [47, 48], 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 19 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

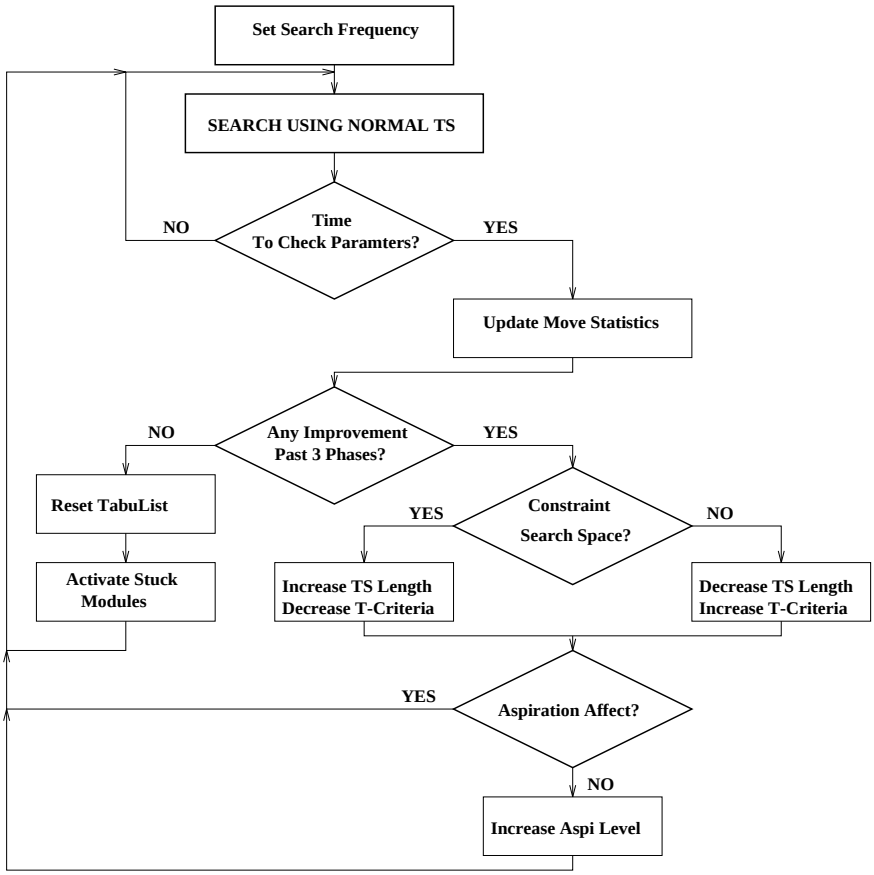


Figure 19: Adaptive Tabu Search

due to invalid moves or Tabu moves. If this is not the case, then the search controller would decide upon the following:

- 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										
Blks	**PRIM1 Circuit**					**IND1 Circuit**				
	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 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 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.

4.6.4 Performance of Advanced Search Techniques

Figure 20 and Table 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 4.5, 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 [32, 33] were based on an annealing schedule similar to that proposed by Kirkpatrick [22]. In this section, solutions obtained by Simulated Annealing are based on results that use an annealing schedule proposed by White [43]. Even with this cooling schedule execution times are still high as seen in Table 7.

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

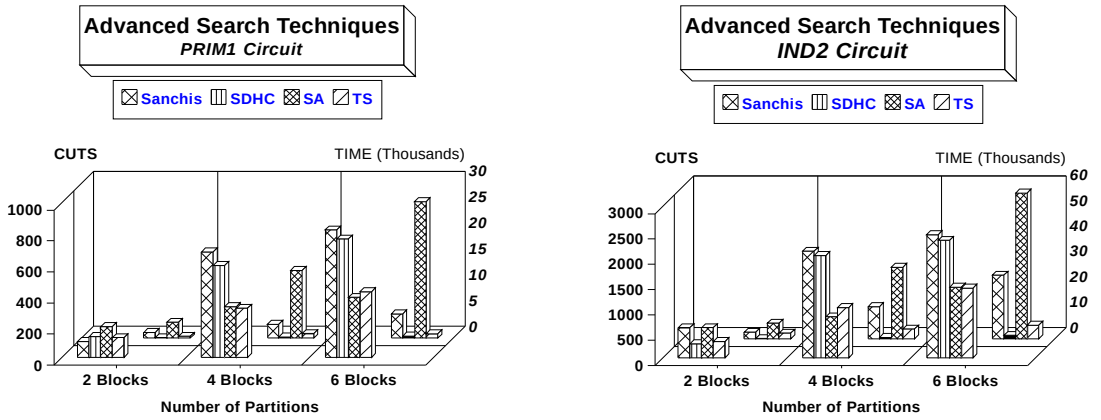


Figure 20: Performance of advanced search heuristics

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 7: 2-Way partitioning of advanced search heuristics

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.

4.7 Summary

This Section 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.

5 Data Structures and Routines

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

5.1 Tabu Search and Data Structures

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 we seek ways of saving time 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 Sanchis [18] 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.
- Make use of first level gain which is the regular gain concept used in earlier algorithms.

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

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

Figures 21a and 21b show the data structures used to read the circuit description. The first structure is a Modlist array, which for each cell contains a linked list of the nets that contain the cell. The second structure is a netlist array, which for each net contains a linked list of the cells on the net. In both cases, each linked list created is regarded as a set, with no duplicates and no implicit order. Each record in each of the arrays also contains several additional fields which the algorithm uses to perform its function.
Figure 21c and 21d present the data structures used for the Aspiration Level criteria and Tabu List respectively. The Aspiration level is a simple array that is indexed by the solution cutstate. The Tabu List is a more complicated structure that is considered to be a random access circular linked list. The Head and Tail pointers are used to easily add and delete nodes at the front and rear of the circular list respectively. The field "next" is used to link a

data item in the list to the next item. The field "link" is used to link only those data items that are related (i.e information related to Cell within the circuit)
Figure 22a shows an array of structures used to implement the net parameters. Figure 22b shows the gain data structure for each cell within the circuit. The bucket list structure is given in Figure 23. This is done using an array BUCKET[-pmax ... pmax], whose k^{th} entry contains a doubly linked list of cells with gains currently equal to k. The fields "next" and "prev" in the bucket structure is used to link the bucket gains, and to easily add and delete an entry in constant time. The "gptr" field in each bucket points to the gains of cells that have equivalent value. The gainlist[] data structure uses the following fields for preserving information:

link field to link "the gains of cell C moving from block A_i to other blocks" together,

prev and next field to link "the gains of all cells having the same gain value",

bptr field to link "the gain of cell moving to block A_k" to the bucket structure

5.2 Algorithm Description

After the network is initialized and an initial partition is obtained, iterations are performed until no more improvements in cutset 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.

Partition-Network

(1) Call **Input-Circuit**

(2) Obtain a random partition P

(3) Call **Init-Partition**

(4) Repeat

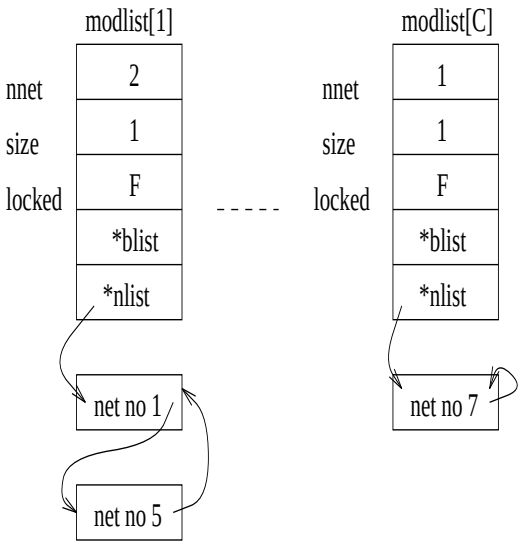
Call **Do-Iteration** to perform an iteration and update the gains

Until (No improvement in cutset size).

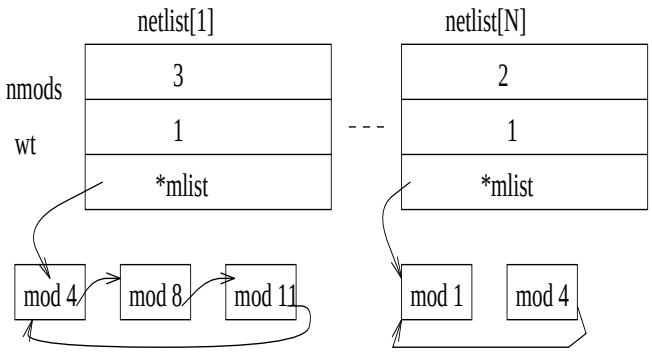
Input-Circuit

(1) For each net = 1 .. n do

(a) For each cell on net(N) do



(a) modlist structure to store cell information

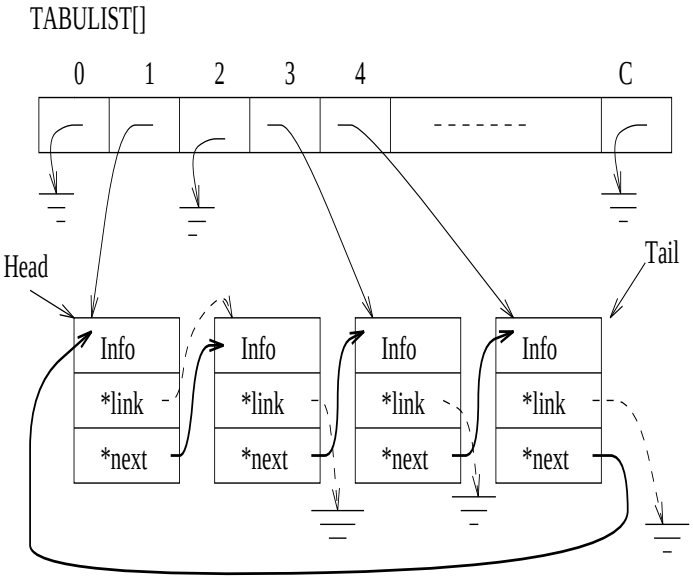


(b) netlist structure to store net information

ASPIRATION[]

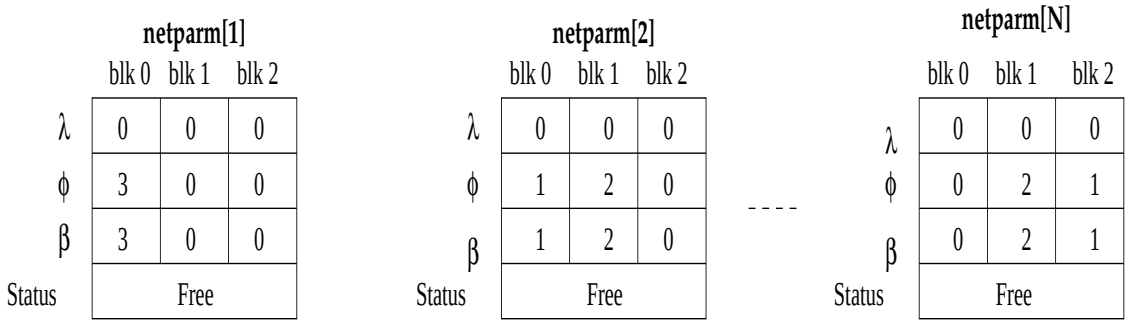
1 cut	2 cuts	3 cuts		MAX cuts
Hi	Hi	Hi		Hi

(c) Table used for Aspiration Level Update

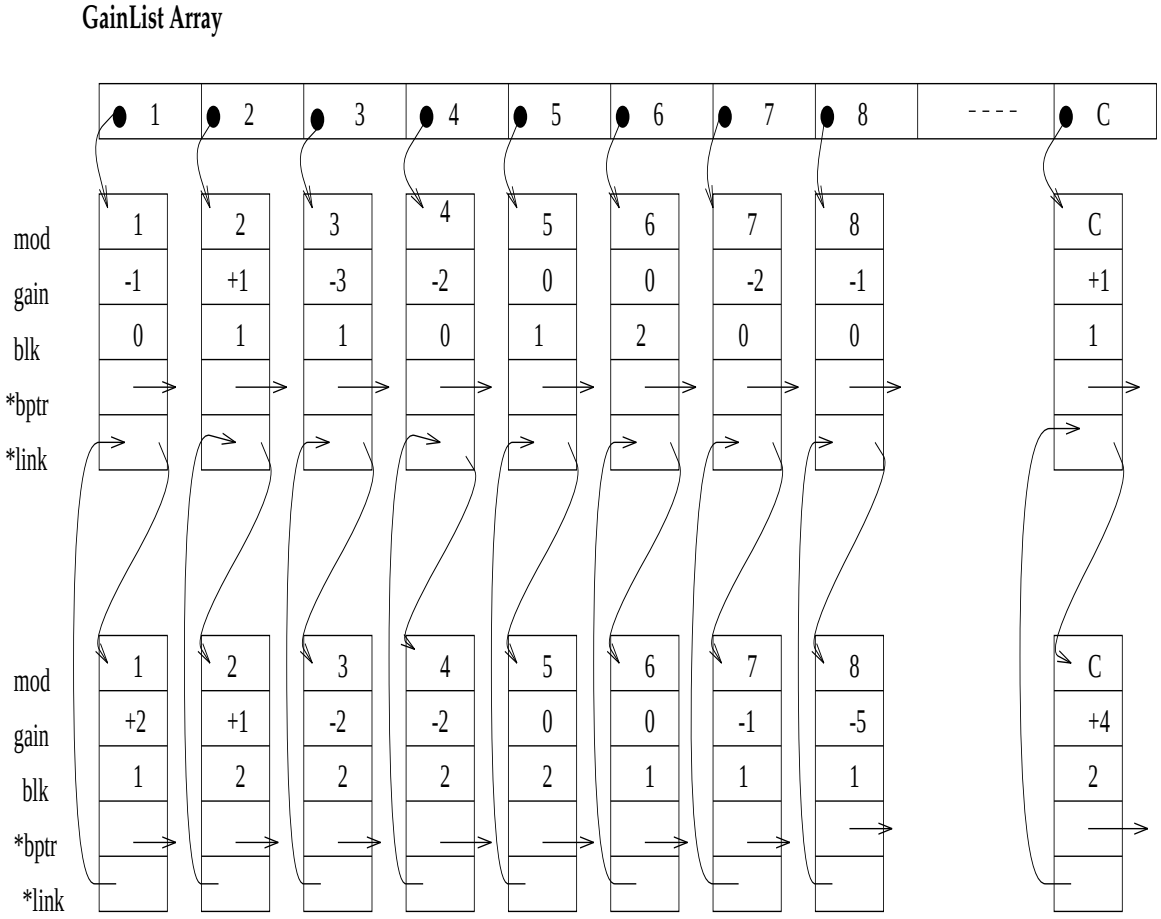


(d) Random access circular list implementation of Tabu List

Figure 21: Netlist, Modlist and TabuList Data Structures



(a) Net Parameters used to efficiently update the gains of the cells.



(b) Gain data structure of cells using a partition of 3 blocks

Figure 22: Net Parameters and Gain Data Structures

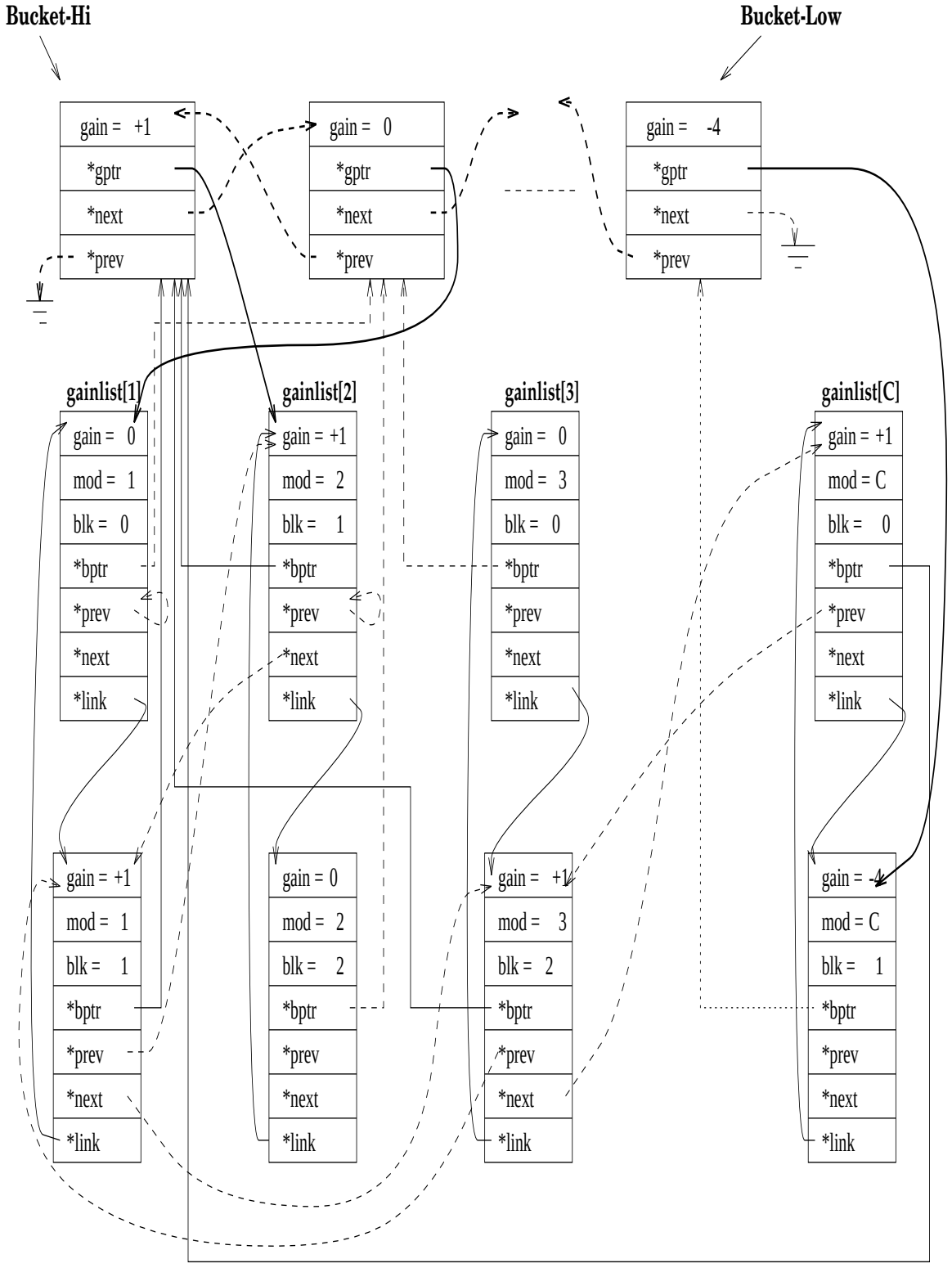


Figure 23: Bucket Gain List

Insert net N into the 'nlist' of cell C
Insert cell C into the 'mlist' of net N
End For

End For

Init-Partition

(1) Set all ϕ 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

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

Return(P)

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

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 $\beta_{A_k} \neq \infty$
  - $\beta'_{A_k} = \infty$

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,

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

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

6 Conclusions

This technical report 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.

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 to exploit and explore the solution space effectively.

All the approaches mentioned above have fundamental strengths but tend to suffer as the problem size increases. 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 remaining part of the paper explored the effectiveness of integrating different heuristics to produce robust, efficient and fast hybrids. Combining GRASP with Tabu Search and Genetic Algorithms not only improved the solution quality but also reduced the computation time in many cases. 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 combined Tabu Search with GRASP (GRASP-TS) gives rise to a heuristic that is capable of finding excellent solutions compared to those obtained randomly. The final generated partitions using search diversification and intensification embedded in the Genetic Algorithm (GAds-is-TS) to obtain initial starting points for the finely tuned search performed by the Tabu Search heuristic are consistently "better" than those obtained by GRASP. However, the computation time of this hybrid was 2-3 times higher than the GRASP-TS hybrid.

In addition, we should also mention that the Genetic Algorithm produces many solutions that vary in quality compared to those obtained by GRASP. Clearly, one trades off the better solutions generated by GA-TS hybrid compared to those solutions that are obtained by the faster GRASP-TS approach. One would use both methods in practice.

References

[1] S. Areibi and A. Vannelli. Advanced Search Techniques for Circuit Partitioning. In *DIMACS Series in Discrete Mathematics and Theoretical Computer Science*, pages 77–98, Rutgers State University, New Jersey, 1994.

[2] G. Hachtel and C. Morrison. Linear Complexity Algorithms for Hierarchical Routing. *IEEE Transactions on Computer Aided Design*, 8(1):64–80, 1989.

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

[4] K. Ueda, R. Kasai, and T. Sudo. Layout Strategy, Standardization, and CAD Tools. *Layout Design and Verification*, pages 1–54, 1986.

[5] E.R. Barnes. An Algorithm for Partitioning The Nodes of a Graph. *SIAM Journal of Algebraic and Discrete Methods*, 3(4):541–550, December 1982.

[6] 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*, 11(7):885–892, July 1992.

[7] A. Vannelli and S.W. Hadley. A Gomory-Hu Cut Tree Representation of a Netlist Partitioning Problem. *IEEE Transactions on Circuits and Systems*, 37(9):1133–1139, September, 1990.

[8] W.E. Donath and A.J. Hoffman. Lower Bounds for The Partitioning of Graphs. *IBM Journal of Research and Development*, 17(5):420–425, 1973.

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

[10] 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.

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

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

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

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

[15] B.W. Kernighan and S. Lin. An Efficient Heuristic Procedure for Partitioning Graphs. *The Bell System Technical Journal*, 49(2):291–307, February 1970.

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

[17] B. Krishnamurthy. An Improved Min-Cut Algorithm for Partitioning VLSI Networks. *IEEE Transactions on Computers*, 33(5):438–446, May 1984.

[18] L.A. Sanchis. Multiple-Way Network Partitioning. *IEEE Transactions on Computers*, 38(1):62–81, January 1989.

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

[20] H. Shin and C. Kim. A Simple Yet Effective Technique for Partitioning. *IEEE Transactions on VLSI Systems*, 1(6):380–386, September 1993.

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

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

[23] C. Carvalho, D. Falco, and B. Apolloni. Quantum Stochastic Optimization. *Stochastic Processes and Their Applications*, 33(2):233–244, 1989.

[24] D.E. Goldberg. *Genetic Algorithms in Search, Optimization, and Machine Learning.* Addison-Wesley Publishing Company, Inc, Reading, Massachusetts, 1989.

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

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

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

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

[29] C.W. Yeh, C.K. Cheng, and T.T. Lin. A General Purpose Multiple-Way Partitioning Algorithm. *IEEE Transactions on Computer Aided Design*, 13(12):1480–1488, 1994.

[30] P.K. Chan, D.F. Schlag, and J.Y. Zien. Spectral K-way Ratio-Cut Partitioning and Clustering. *IEEE Transactions on Computer Aided Design*, 13(9):1088–1096, 1994.

[31] 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*, 14(2):145–153, 1995.

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

[33] S. Areibi and A. Vannelli. Circuit Partitioning Using a Tabu Search Approach. In *IEEE International Symposium on Circuits and Systems*, pages 1643–1646, Chicago, Illinois, 1993.

[34] C. Sechen and D. Chen. An improved Objective Function for Min-Cut Circuit Partitioning. *IEEE Transaction on CAD*, pages 502–505, 1988.

[35] T. Feo, M. Resende, and S. Smith. A Greedy Randomized Adaptive Search Procedure for The Maximum Independent Set. *Operations Research*, 1994. Journal of Operations Research.

[36] T. Feo and M. Resende. A Probabilistic Heuristic for a Computationally Difficult Set Covering Problem. *Operation Research Letters*, 8:67,71, 1989.

[37] V. Venkatasubramanian and I.P. Androulakis. A Genetic Algorithm Framework for Process Design and Optimization. *Computers Chemical Engineering*, 15(4):217–228, 1991.

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

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

[40] M.D. Huang, F. Romeo, and A. Sangiovanni. An Efficient General Cooling Schedule for Simulated Annealing. In *IEEE International Conference on CAD*, pages 381–384, November 1986.

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

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

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

[44] 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*, pages 92–95. IEEE, 1991.

[45] A. Hertz and D. Werra. Using Tabu Search Technique for Graph Coloring. *Computing*, pages 345–351, 1987.

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

[47] F. Glover. Tabu Search Part I. *ORSA Journal on Computing*, 1(3):190–206, 1990.

[48] F. Glover. Tabu Search Part II. *ORSA Journal on Computing*, 2(1):4–32, 1990.