VLSI DESIGN
2000, Vol. 11, No. 3, pp. 259–283
Reprints available directly from the publisher
Photocopying permitted by license only

Tabu Search: A Meta Heuristic for Netlist Partitioning

SHAWKI AREIBI* and ANTHONY VANNELLI†

Department of Electrical and Computer Engineering, University of Waterloo, Waterloo, Ontario, Canada N2L 3G1

(Received 1 March 1999; In final form 1 December 1999)

The main goal of the paper is to explore the effectiveness of a new method called *Tabu Search* [1] on partitioning and compare it with two techniques widely used in CAD tools for circuit partitioning *i.e.*, Sanchis Interchange method and Simulated Annealing, in terms of the running time and quality of solution. The proposed method integrates the well known iterative multi-way interchange method with Tabu Search and leads to a very powerful network partitioning heuristic. It is characterized by an ability to escape local optima which usually cause simple descent algorithms to terminate by using a short term memory of recent solutions. Moreover, Tabu Search permits backtracking to previous solutions, which explore different directions and generates better partitions.

The quality of the test results on MCNC benchmark circuits are very promising in most cases. Tabu Search yields netlist partitions that contain 20% – 67% fewer cut nets and are generated 2/3 to 1(1/2) times faster than the best netlist partitions obtained by using an interchange method. Comparable partitions to those obtained by Simulated Annealing are obtained 5 to 20 times faster.

Keywords: Netlist partitioning; Tabu Search; Adaptive search; VLSI circuit layout

1. CIRCUIT PARTITIONING

Circuit partitioning is the task of dividing a circuit into smaller parts. It is an important aspect of layout for several reasons. Partitioning can be used directly to divide a circuit into portions that are implemented on separate physical components, such as printed circuit boards or chips. Here, the objective is to partition the circuit into parts such that the sizes of the components are within prescribed ranges and the complexity of connections between the components is minimized. As can be seen in Figure 1, after swapping modules between the two blocks, we end up minimizing the number of signal nets that interconnect the components between the blocks. A natural way of formalizing the notion of wiring complexity is to attribute to each net in the circuit some connection cost, and

*Address for correspondence: School of Engineering, University of Guelph, Canada. e-mail: sareibi@cheetah.vlsi.uwaterloo.ca
†The research is partially supported by a Natural Science and Engineering Research Council of Canada (NSERC) operating grant (OGP 0044456) and an Information Technology Research Center (ITRC) of Ontario Operating grant. e-mail: vannelli@cheetah.vlsi.uwaterloo.ca

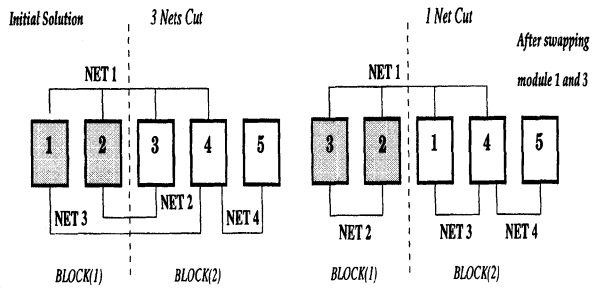


FIGURE 1 Illustration of circuit partitioning.

to sum the connection costs of all nets connecting different components. A more important use of circuit partitioning, is to divide up a circuit hierarchically into parts with divide-and-conquer algorithms for *placement*, floorplanning, and other layout problems. Here, cost measures to be minimized during partitioning may vary, but mainly they are similar to the connection cost measures for general partitioning problems [2].

1.1. An Integer 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 (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.

We define:

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

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

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

$$\max \sum_{j=1}^{n_n} \sum_{k=1}^{n_b} y_{jk} \tag{1}$$

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:

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

(iii) Netlist constraints:

$$y_{jk} \le x_{ik}, \quad \text{where } \begin{matrix} 1 \le j \le n_n \\ 1 \le k \le n_b \\ i \in \text{Net } j \end{matrix}$$

(iv) $0-1$ constraints:

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

The netlist 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.[1]

This paper is divided into five sections. Section 2 gives an overview of the different techniques that have been used by different researchers to solve the circuit partitioning problem. In addition classification of different heuristic and approximation algorithms used to solve the circuit partitioning is presented with a brief overview of the advantages and disadvantages of each technique. Section 3, introduces the Tabu Search as a combinatorial optimization technique and explains its usage as

[1] Results for an MIP formulation are presented in Section 5.

a meta-heuristic to guide a basic interchange method. Detailed implementation of Tabu Search is introduced in Section 4. Test results on several well-known MCNC benchmark circuits are explained in Section 5. Results comparing previous attempts (*i.e.*, module interchange improvement methods, Simulated Annealing) with the Tabu Search implementation are presented. Finally, conclusions and future work are described in Section 6.

2. OVERVIEW

It has been shown that graph and network partitioning problems are NP-Complete [3]. Therefore, attempts to solve these problems have concentrated on finding heuristics which yield approximate solutions in polynomial time. 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 (many real life problems contain more than 100 K modules and nets). For the circuit partitioning problem several classes of algorithms were used to generate good partitions. The techniques can be classified into three classes – *Iterative Improvement algorithms*, *Optimization Techniques* and *Simulated Annealing*.

2.1. Classification of Solution Techniques

Partitioning methods can be classified as being constructive or iterative. Constructive algorithms determine a partitioning from the graph describing the circuit or system, whereas iterative methods aim at improving the quality of an existing partitioning solution. Constructive partitioning approaches are mainly based on clustering [4, 5], spectral or eigenvector methods [6], placement-based partitioning [7], mathematical programming or network flow computations.

2.1.1. Interchange Methods

To date, iterative improvement techniques that make local changes to an initial partition are still the most successful partitioning algorithms in practice. The advantage of these heuristics is that they are quite robust.

In fact, they can deal with netlists as well as arbitrary vertex weights, edge costs, and balance criteria. The heuristics are frequently used in divide-and-conquer algorithms for placement and floor-planning that are variants of the mincut strategy [2].

Kernighan and Lin (KL) [8] described a successful heuristic procedure for graph partitioning which became the basis for most module interchange-based improvement partitioning heuristics used in general. Pseudo-code for the algorithm is given in Figure 2. Their approach starts with an initial bisection and then involves the exchange of pairs of vertices across the cut of the bisection to improve the cut-size as illustrated in Figure 3. The main contribution of the Kernighan and Lin algorithm is that it reduces the danger of being trapped in local minima that face greedy search strategies. The algorithm determines the vertex pair whose exchange results in the largest decrease of the cut-size *or* in the smallest increase, if no decrease is possible. The exchange of vertices is made only tentatively where vertices involved in the exchange are locked temporarily. The locking of vertices prohibits them from taking part in any further tentative exchanges. A pass in the Kernighan and Lin algorithm attempts to exchange all vertices on both sides of the bisection. At the end of a pass the vertices that yield the best cut-size are the only vertices to be exchanged. Computing gains in the KL heuristic is expensive; $O(n^2)$ swaps are evaluated before every move, resulting in a complexity per pass of $O(n^2 \log n)$ (assuming a sorted list of costs).

Fiduccia and Mattheyses (FM) [9] modified the Kernighan and Lin algorithm 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. The

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

FIGURE 2 The Kernighan-Lin algorithm.

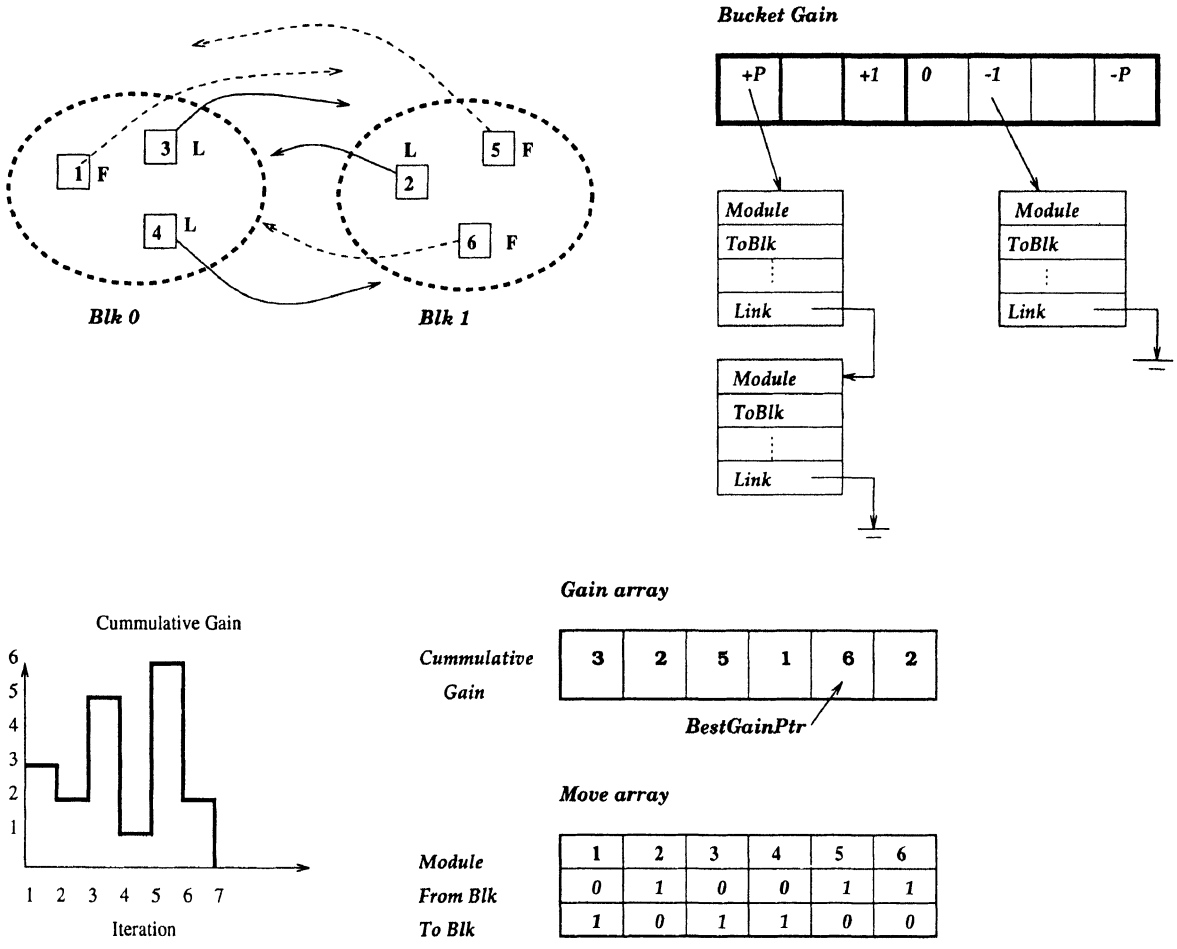


FIGURE 3 Basic techniques for interchange methods.

FM method reduces the time per pass to linear in the size of the netlist (*i.e.*, O(p), where p is the total number of pins) by adopting a single-cell move structure, and a gain bucket data structure that allows constant-time selection of the highest-gain cell and fast gain updates after each move.

Krishnamurthy [10] introduced a refinement of the Fiduccia and Mattheyses method for choosing the best cell to be moved. 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 [11] 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 should be performed until no improvement in cutset size is obtained. This strategy seems to offer some hope of improving the partition in a homogeneous way, by adapting the level gain concept to multiple blocks.

In general, node interchange methods are greedy or local in nature and get easily trapped in local minima. More important, it has been shown that interchange methods fail to converge to "optimal" or "near optimal" partitions unless they initially begin from "good" partitions [12]. Sechen [13] shows that over 100 trials or different runs (each run beginning with a randomly generated initial partition) are required to guarantee that the best solution would be within twenty percent of the optimum solution. Hadley *et al.* [14] also show that starting from good partitions that are generated by an eigenvector approach, using this interchange method on the *one* partition (generated by the eigenvector approach) yields better results than starting from 30 random partitions.

2.1.2. Optimization Methods

The numerical optimization technique by Barnes [15] approximates the graph partitioning problem by a linear programming transportation problem using an eigenvector approach. This technique produces one solution and does not require multiple runs. The advantage of developing an eigenvector approach to solve the partitioning problem is that the generated partitions tend to have many nodes placed in the "right blocks". This is due to observation that eigenvector methods are *global* approaches for solving large-scale optimization problems. Another important advantage of using the eigenvector approach is that it enables us to estimate a lower bound on the weight of any cut of the graph G [14].

2.1.3. Simulated Annealing

Simulated Annealing [16] is widely recognized as a method for determining the global minima of combinatorial optimization problems; that is, it finds the global minimum with probability 1. Its basic feature is that it allows *hill climbing* moves (*i.e.*, the acceptance of moves which increase the cost), thus eliminating chances of getting trapped in local minima. A problem formulated to be solved by Simulated Annealing defines an objective function for assessing the quality of a solution and a set of moves that transform one valid solution into another. Accepting moves (with a decreasing probability) that increase the solution's cost allows Simulated Annealing to explore the design space more thoroughly and move toward a global solution from a local optimum. The major disadvantage of Simulated Annealing is the long computation time, on the order of hours or days for certain realistic problems such as VLSI placement and routing. The integration of Simulated Annealing with a more systematic approach such as Tabu Search [17] has proven to reduce the computation time drastically.

2.2. General Approaches for Improving Circuit Partitions

Figure 4 illustrates the current approaches that are used to improve the performance of algorithms for circuit partitioning based on module interchange.

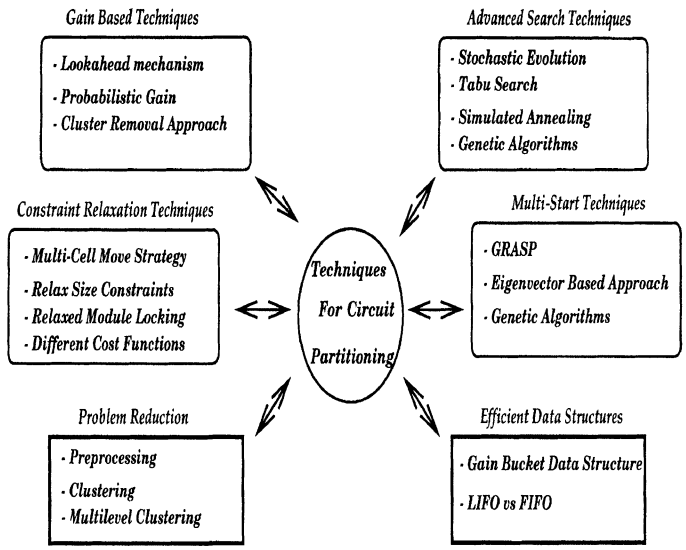


FIGURE 4 General approaches for improving circuit partitioning.

The methods can be classified according to the following criteria:

- **Gain Based Techniques** such as CLIP [5] that selects cells to move with a view to moving clusters that straddle the two subsets of a partition into one of the subsets. Lookahead techniques [10] for tie breaking and PROP [18] that is capable of capturing the global and future implications of moving a node at a certain time.
- **Constraint Relaxation Techniques** where exact bisection constraints [19, 20] are relaxed to improve the resulting cut-sizes. In addition [21] relax the locking mechanism which enforces each cell to move exactly once per pass by allowing each cell to move more than once.
- **Techniques based on Efficient Data Structures** a good example is the FM technique (explained earlier) that reduces the time per pass to linear in the size of the netlist by utilizing an efficient gain bucket data structure that allows fast gain updates after each move. Sanchis's extension of this algorithm to multi-way partitioning also relies on efficient data structures to select the modules to be moved from one partition to the other. Further investigation of LIFO buckets with FIFO based buckets is explored in [22] to improve the solution quality.
- **Problem Reduction Techniques** which are mainly based on preprocessing and clustering [23, 24, 4] have received much recent attention since they are viewed as the most promising method for tackling the increasing problem sizes in VLSI CAD. Clustering involves identifying and merging strongly connected cells into clusters thereby densing the circuit. The result is a two phase heuristic [25]. Recent work [25, 26] has illustrated the promise of *multilevel* approaches for partitioning large circuits. Multilevel partitioning recursively clusters the instance until its size is smaller than a given threshold, then un-clusters the instance while applying a partitioning refinement algorithm.
- **Advanced Search Techniques** [27, 28] are search heuristics that are capable of escaping local minima. 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*, and *Tabu Search*.

- **Multi-Start Techniques** attempt to improve the performance of iterative methods by carefully choosing good initial configurations for each execution of the optimization algorithm [29, 30, 31]. The main disadvantage of iterative improvement techniques 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. Over 100 trials or different runs are required to guarantee that the best solution would be within twenty percent of the optimum solution.

The main goal of the paper is to explore the effectiveness of a new method called *Tabu Search* [1] on partitioning and compare it with two techniques widely used in CAD tools for circuit partitioning *i.e.*, Sanchis Interchange method and Simulated Annealing, in terms of the running time and quality of solution. While it is not the purpose of this paper to compare the Tabu Search implementation with more advanced partitioning techniques such as HMETIS that is based on multilevel clustering and others that utilize the general approaches mentioned in Section 2.2, we present some results in Section 5 evaluating Tabu Search with respect to HMETIS [25] state of the art partitioner.

3. TABU SEARCH

Tabu Search is a general heuristic procedure for global optimization. Based on simple concepts it has been extremely efficient in finding almost optimal solutions for many types of difficult combinatorial optimization problems ranging from graph partitioning [32, 17, 33], graph coloring [34], to quadratic assignment problems [35]. 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.

We first present Tabu Search in a simple form that discloses two of its key components. First, Tabu Search restricts the search by classifying certain moves as forbidden (*i.e.*, Tabu). The second feature of this method is that it frees the search by a short term memory function that provides "strategic forgetting".

3.1. 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.

3.1.1. 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. For example in a 4-way partitioning scheme, if an iteration involved the movement of Module M_1 from Block B_W (source block) to Block B_X (destination block) then the tabu restriction imposed by the system is on M_1 back to B_W but not to other blocks say B_X, B_Y, B_Z. (iii) Restrictions based on module and destination block (**TC3**), *i.e.*, in the previous example the restriction imposed on the system would be on M_1 to move for a certain number of iteration (depending on the Tabu length) back to B_X. (iv) A combination of the above restrictions is implemented using (**TC4**). The fourth restriction is considered to be the most lenient.

3.1.2. 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 5a 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 5b 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.

3.1.3. 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. In Section 3.1.4 the aspiration rule **ASP1** is clearly illustrated through an example. 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 5c shows the effect of the aspiration on the overall solution (cut-net) using the Chip1 circuit. As the number of blocks increase, the effect of the aspiration level has more impact. The Tabu restrictions and aspiration level criterion of Tabu Search play a dual role in constraining and guiding the search process.

3.1.4. Tabu Search Example

Figure 6 provides an example of the Tabu Search mechanism. Figure 6a shows a circuit that is to be partitioned into two blocks with an initial random partition. Figure 6b presents the gains associated with modules in each iteration (always pick the module associated with best gain "steepest descent") Figures 6c, d give the status of the list of Tabu solutions as the routine progresses. Note that moves do not necessarily result in a decrease

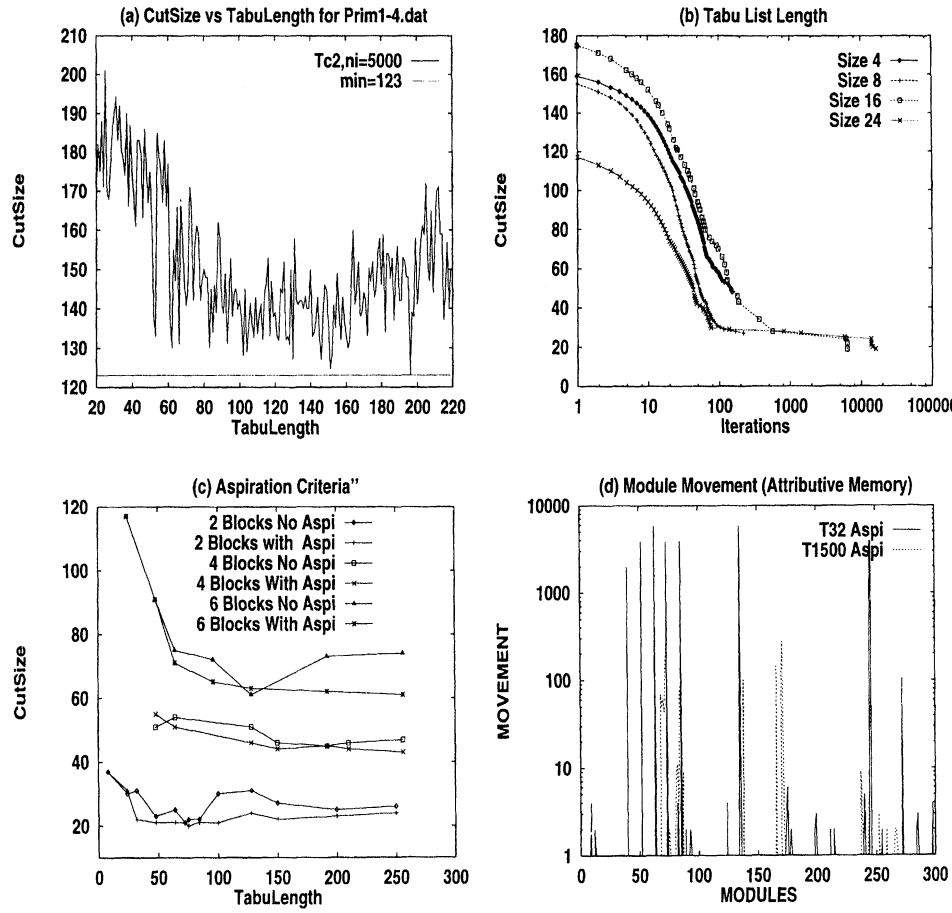


FIGURE 5 Parameters affecting Tabu Search.

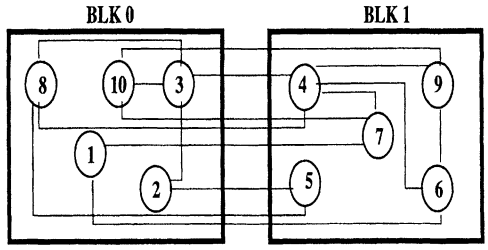
in the associated cost; costs may increase simply as a result of the current solution being Tabu.

Figure 6c shows that only four consecutive solutions are considered as Tabu at any one time. Starting from an initial solution s_0 *in* S_0 a move is made to solution s_1 *in* S_1 on the basis of cost. At the same time, the initial solution s_0[2] is placed in a list of Tabu solutions, which is, in effect, a first-in-first-out queue of length 4. On the basis of accepting the allowable (*i.e.*, non-Tabu) solution with the minimum cost, this process is repeated in subsequent moves.

Note that, in making the move from s_4 *to* s_5, the Tabu status of the initial feasible solution s_0 is dropped; whenever a fifth solution enters the Tabu list, the oldest solution therein is removed and again becomes allowable. The procedure described so far lacks one important feature suggested by Glover [1]. Aspiration effectively provides a method of overriding the Tabu status of moves while preserving to a high degree the ability to avoid becoming trapped within a subset of the solution space. An example is given in Figure 6e, together with the table shown in Figure 6f illustrate the advantageous effects of aspiration.

The table in Figure 6f provides complementary information to that given in Figure 6e. Its contents are the aspiration values associated with each cost

[2] In this case the movement of module 5 from block 0 to block 1 is s_0.



(a) Initial partition for circuit X.

Iteration	Best Gain	Module involved
(1)	+2	Module 5
(2)	-1	Module 4
(3)	+2	Module 3
.....		

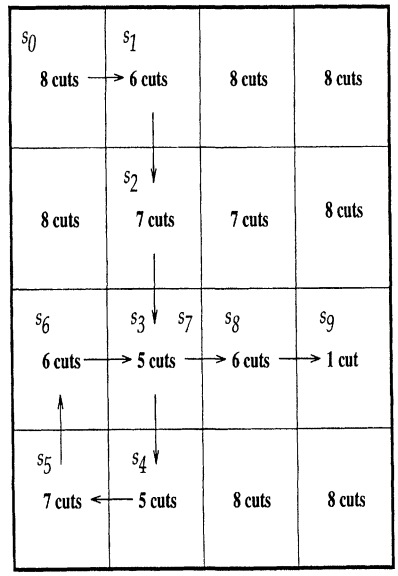
(b) Gains associated with modules in the circuit

4 MAX VALS	1	2	3	4
MODULE	5	4	3	2
FROM BLK	1	1	0	0
TO BLK	0	0	1	1

(c) Tabu List after 4 iterations

4 MAX VALS	1	2	3	4
MODULE	6	4	3	2
FROM BLK	1	1	0	0
TO BLK	0	0	1	1

(d) Tabu List after the fifth iteration



(e) The solution space in terms of cuts

Aspiration Values Associated with Different Cuts in the Circuit

Iteration	Cuts							
	1	2	3	4	5	6	7	8
0	hi	hi	hi	hi	hi	hi	hi	hi
1	hi	hi	hi	hi	hi	hi	hi	6
2	hi	hi	hi	hi	hi	7	hi	6
3	hi	hi	hi	hi	hi	7	5	6
4	hi	hi	hi	hi	5	7	5	6
5	hi	hi	hi	hi	5	7	5	6
6	hi	hi	hi	hi	5	7	5	6
7	hi	hi	hi	hi	5	5	5	6

(f) The Aspiration Table

FIGURE 6 Tabu moves and aspiration.

value (given in the top row) at each step of the search routine. A Tabu list of length 4 is again assumed. Initially, at S_0, all aspiration values are set at a level higher than any possible cost indicated in the aspiration table as ∞ (hi). As moves are made, the appropriate aspiration values are updated, if necessary. For instance, when the first move is made from a solution of cost 8 to a solution of cost 6, the aspiration value associated with the cost value of 8 is updated in the aspiration table to 6. This implies that, for aspiration to release a Tabu solution so that it may become a candidate for a move from any solution with cost 8, the Tabu solutions must have a cost less than 6. Similarly, the second move forces an update of the aspiration value associated with a cost value of 6; the aspiration value is now set at 7. In the example of Figure 6e,

the use of aspiration allows the important seventh move back to a previously accepted solution. As has been demonstrated by the example, aspiration and the short term memory are complementary features, that provide a flexible search procedure. Although the contrived problem may seem simplistic, it nevertheless provides a clear understanding of the power and applicability of aspiration in the basic Tabu Search procedure.

3.2. 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 [1, 36]. Combined with the short term memory functions, intermediate and long term memory functions provide an interplay between "exploitation" and "exploration" of the solution space [1] (fine tuned search *versus* exploration of the solution space). Intermediate term memory operates by recording and comparing features of a selected number of best trial solutions generated during a particular period of search. The method then seeks new solutions that exhibit these features. The long term memory functions, whose goal is to diversify the search, employs principles that are roughly the reverse of those for intermediate term memory. Our implementation of the long term memory (search diversification) is based on two different techniques:

- in the first technique, the frequency of moving a module is used as a means of exploring new solutions that have not been visited before. Modules having high frequency are discouraged for further movement. This allows less frequented moves to be explored. This is done either:
  - by making the Tabu status of module proportional to its frequency.
  - or by locking the module (preventing it from moving any further for a certain number of moves) that has very high frequency.
- the second form of diversification that allows the meta-heuristic to come out of local optima is to focus more specifically on producing initial solutions as different as possible from the solutions generated throughout the previous history of the search process. These solutions are then used as a restart to get out of a local minimum.

Diversification of the search is usually invoked in the following situations:

- For a sequence of θ consecutive moves there has been no improvement of the best solution found.
- All the solutions in the current neighborhood are tabu and none of them satisfies an aspiration criterion.

The intermediate term memory for search intensification uses a simple approach that allows the metaheuristic to come out of local optima. It basically records solutions in a Queue with a certain length. As the search is proceeding the system identifies a few best solutions and uses them in a restart phase to intensify the search into promising regions not fully explored. Figure 5d shows a recording of the module movement during the search procedure.

4. TABU SEARCH IMPLEMENTATION FOR PARTITIONING

The Tabu Search routine described so far can be formulated as shown in Figure 7. The algorithm requires an initial feasible solution (partition) for which an associated cost yielding cut nets 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.

4.1. Tabu List Size

Our Tabu list management techniques are based on static and dynamic approaches. In the static

```
Input:
    The net list or the Graph G= (V,E)
    K = number of partitions; | T | = size of Tabu list
    max_num_iter = maximum number of iterations allowed.
Initialization:
    Initial Partition = Generate a random solution
    s = (V1, V2,...,Vk); 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 Tabulist Status and Accept the move
                Update Tabulist; Update the Aspiration Level;
            Else
                Move not accepted;
        num_iter = num_iter + 1;
    End While
Output:
    Best Partition = bestpart; Best Cut = bestcut
```

FIGURE 7 A simple Tabu Search implementation.

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.

4.2. Stopping Criteria

The stopping conditions used in this implementation are based on the following: (i) The search will terminate when "num_iter" is greater than the maximum number of iterations allowed "max_num_iter". (ii) The search will terminate, if no improvement on the best solution found so far can be made, for a given number "max_num_iter" of consecutive iterations. The maximum number of iterations after which the routine terminates, depends on whether the routine starts from a random starting point or a good initial starting point (as will be explained in part II).

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 × *nodes*". The final solution gives the overall best partition and best cut after the specified maximum number of moves.

5. TESTS AND RESULTS

Table I presents the benchmarks [37] that are used to compare the performance of Tabu Search with those obtained by Simulated Annealing and Sanchis multi-way partitioning interchange method. In all cases, the netlists were partitioned into two, four and six blocks of modules. Each block had the *same* number of modules, *i.e.*, they are *equi-partitioned*.[3] All testing was conducted on Sun SPARC II computers running Solaris Operating System. The programs were written in C and compiled with the Sun C compiler.

5.1. Different Tabu Search Implementations

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

5.1.1. 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 employing a certain number of passes 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.

5.1.2. 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 memories as explained in Section 3.2 are employed to intensify and diversify the search.

5.1.3. Comparison Between Different Settings

Table II presents results obtained using the Tabu Search under the different settings described. **TS-DA** represents Tabu Search with delay activation. The second column of each table represents Tabu Search using the first Tabu restriction. **TS-ASP2** shows results obtained using the second aspiration rule (described in Section 3.1.3). 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

TABLE I Benchmarks used as test cases

				Node degree			Net size		
Circuit	Nodes	Nets	Pins	Max	$\bar{x}$	σ	Max	$\bar{x}$	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

[3] Many results published in the literature are not based on equi-sized partitions, and therefore we cannot make a valid comparison.

TABLE II A comparison between different Tabu Search settings

		TS		TS-DA		TS-TC1		TS-ASP2		TS-DS-IS	
Circuit	Blks	Cut	Time	Cut	Time	Cut	Time	Cut	Time	Cut	Time
Chip1	2	20	13.3	20	3.2	20	4.1	28	3.2	**20**	14.0
	4	49	12.3	56	4.0	53	4.3	55	3.1	**47**	23.5
	6	64	12.5	58	4.6	67	4.3	56	7.9	**58**	37.0
Chip2	2	14	6.9	14	1.7	16	1.7	17	2.0	**14**	10.3
	4	35	8.6	31	2.5	34	3.0	30	2.6	**29**	19.2
	6	38	13.8	40	4.6	37	7.6	42	4.5	**37**	25.5
Chip3	2	7	4.6	8	1.3	7	1.2	7	1.3	**7**	7.4
	4	28	7.4	38	1.4	32	1.8	35	2.6	**29**	15.7
	6	41	10.5	40	1.8	46	2.2	43	2.2	**40**	13.8
Chip4	2	13	5.7	8	1.4	9	1.1	8	2.9	**8**	7.1
	4	21	12.4	25	3.0	26	6.3	21	4.6	**21**	19.2
	6	27	3.0	40	4.7	41	3.8	34	2.8	**28**	20.5
Prim1	2	59	52.6	56	6.4	60	6.6	54	11.6	**54**	47.4
	4	126	50.2	102	29.4	114	20.4	129	14.0	**102**	100.1
	6	159	1:13	139	20.7	136	44.0	146	12.3	**137**	1:15
Prim2	2	240	3:25	240	1:06	239	27.5	220	47.5	**181**	4:44
	4	663	1:52	412	1:09	438	50.2	382	1:40	**357**	4:59
	6	769	1:46	597	1:48	603	6:05	596	2:53	**562**	7:15
Ind1	2	59	1:31	63	0:25	63	0:22	77	1:42	**45**	2:27
	4	135	2:53	121	0:55	161	1:09	125	1:45	**121**	6:22
	6	230	5:14	203	1:16	212	6:50	190	1:33	**193**	12:39
Bio	2	144	3:45	151	0:53	87	1:02	162	3:10	**127**	5:21
	4	509	6:00	387	1:32	389	3:02	367	2:55	**317**	13:56
	6	791	6:29	423	2:05	452	3:36	409	4:09	**423**	13:40
Ind2	2	805	21:10	392	9:45	388	14:09	381	17:06	**323**	37:30
	4	2323	26:23	1189	9:36	1195	20:09	1207	10:46	**991**	65:02
	6	2662	17:58	1375	17:40	1399	14:09	1444	15:30	**1375**	89:33
Ind3	2	462	28:52	1534	8:08	724	9:13	1229	22:31	**305**	47:12
	4	2139	54:57	1817	24:37	1929	24:23	2052	10:36	**1817**	144:28
	6	2623	44:38	2862	29:27	2743	41:23	3094	32:18	**2539**	181:57

considered best among the others but at the expense of extended CPU time.

5.2. An Adaptive Version of 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. The most important parameters that control the performance of Tabu Search are, the aspiration criterion for back-tracking, the lengths of the Tabu lists and the attributes to be used for the Tabu restrictions. Sections 3.1.1 and 4.1 presented some schemes for determining the move attributes and size of the Tabu list respectively. A method proposed here (*and still under development*), 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.

5.2.1. An ATS Implementation

In Section 4.1, the choice of a preferred value for the Tabu list was based either on empirical testing or on variation to the Tabu list size to eliminate cycling. These schemes (static or dynamic) based on a fixed list size (**FIX-TABU**) are not strict and,

therefore, the possibility of cycling remains. Other Tabu Search implementations are based on the fact that cycles are avoided if the repetition of previously visited configuration is prohibited [36]. For example, in the Reverse Elimination Method [1, 36], 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 8 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 (in this Case 3) then the controller decides to reset the Tabu list (removes all items) and activate all modules that have low average movement. The controller attempts to do so either because the landscape of the solution space is flat or no more modules can be moved due to invalid moves or Tabu moves. If this is not the case, then the search controller would decide upon the following:

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

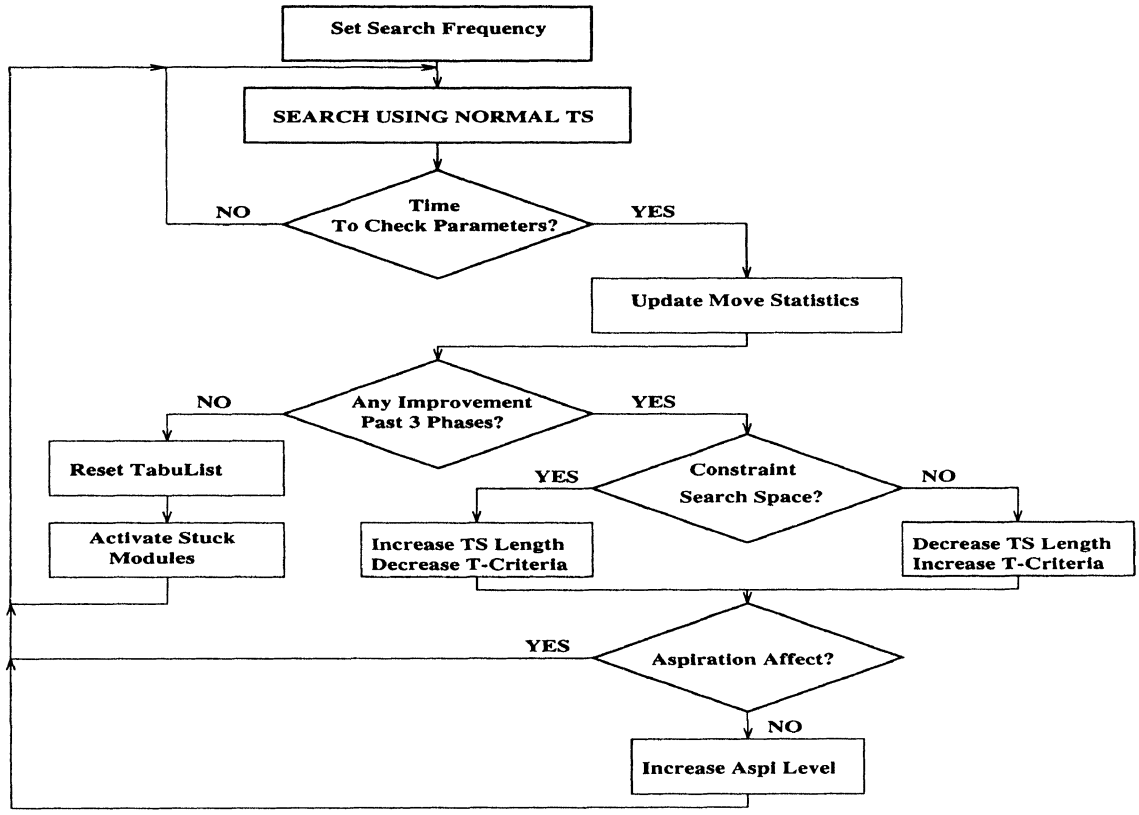


FIGURE 8 An adaptive Tabu Search scheme.

In the first case the search controller would proceed to check the effect 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. The search controller was implemented and tested on some problems, and was effective in improving the search quality within Tabu Search heuristic. Tables III–V show 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 improves steadily for different block partitions and as the problem size increases.

5.3. Tabu Search *Versus* Interchange Methods

Tables VI–VIII 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 [11] and Tabu Search[4] starting from random initial points.[5]

As has been discussed in Section 2.1.3, the Simulated Annealing algorithm's performance depends on the *Annealing schedule* used. The choice of Annealing schedule plays an important

TABLE III 2 way partitioning results using adaptive Tabu Search

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

TABLE IV 4 way partitioning results using adaptive Tabu Search

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

[4] These results are based on pure Tabu Search and not adaptive Tabu Search.
[5] In Section 5.4 we show more results based on GRASP.

TABLE V 6 way partitioning results using adaptive Tabu Search

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

TABLE VI 2-way partitioning using TS, SA and interchange

	SA		IC		TS		% Improve TS *vs.* SA		% Improve TS *vs.* IC	
Circuit	Cut	Time	Cut	Time	Cut	Time	C	T	C	T
Chip1	20	92	20	18	20	14	0	557	0	29
Chip2	14	98	15	19	14	10	0	851	7	84
Prim1	65	346	60	91	54	47	20	630	11	92
ind1	46	959	42	211	45	147	2	552	−6	44
Prim2	224	1314	226	433	181	284	24	362	25	53
Bio	199	3060	102	1058	127	321	57	853	−20	230
ind2	594	6240	593	2661	323	2250	84	177	84	18
ind3	655	9420	514	2294	305	2822	115	234	69	−19
Avg.							38	527	21	66

TABLE VII 4-way partitioning using TS, SA and interchange

	SA		IC		TS		% Improve TS *vs.* SA		% Improve TS *vs.* IC	
Circuit	Cut	Time	Cut	Time	Cut	Time	C	T	C	T
Chip1	45	374	55	27	47	23	−4.3	1491.5	17	16.6
Chip2	29	290	39	24	29	19	0	1410.4	34.5	27.1
Prim1	106	1182	155	96	102	100	3.9	1082	52	−3.6
ind1	141	3807	259	526	121	382	16.6	896.6	114	37.7
Prim2	365	5377	627	787	357	299	2.2	1698	75.6	163.2
Bio	327	13025	680	2627	317	836	3.2	1458	114.5	214.2
ind2	809	28236	2102	12729	991	3902	−18.4	623.6	112.1	226.2
ind3	1668	40085	2183	5874	1817	8668	−8.2	362.4	20.1	−32.2
Avg.							**−0.63**	**1127.8**	**67.5**	**81.2**

role in controlling the effectiveness and efficiency of the algorithm. Our previous circuit partitioning results using Simulated Annealing [32, 17] were based on an Annealing schedule similar to that proposed by Kirkpatrick [16]. In this section, solutions obtained by Simulated Annealing are based on results that use an Annealing schedule proposed by White [38]. Even with this cooling schedule execution times are still high as seen in Tables VI–VIII.

The second and third columns in each table presents the results obtained from Simulated

TABLE VIII 6-way partitioning using TS, SA and interchange

	SA		IC		TS		% Improve TS *vs.* SA		% Improve TS *vs.* IC	
Circuit	Cut	Time	Cut	Time	Cut	Time	C	T	C	T
Chip1	54	651	77	39.4	58	37	−6.9	1659.5	32.8	6.5
Chip2	39	804	63	40.2	37	25.5	5.4	3053	70.3	57.6
Prim1	124	2345	181	133	139	78	−10.8	2906	30.2	70.5
ind1	182	7618	364	769	193	819	−5.7	830.2	88.6	−6.1
Prim2	428	10371	773	1382	562	435	−23.8	2284	37.5	217.7
Bio	387	26427	821	4657	423	820	−8.5	3001	94.1	467.9
ind2	1395	57720	2430	25132	1375	5373	1.5	974.3	76.7	367.7
ind3	2444	80423	2640	13131	2801	10917	−12.7	636.7	−5.7	20.3
Avg.							**−7.7**	**1918.1**	**53.1**	**150.3**

Annealing with a simple *cooling schedule* similar to that proposed by Kirkpatrick [16]. Due to this approximation the algorithm is no longer guaranteed to find a global minimum with probability one. The fourth and fifth columns give the results obtained from using node interchange on *30 random starting partitions*. The results obtained with the new Tabu Search based approach are shown in columns 6 and 7. Columns 8 and 9 show the percentage improvement in both quality of cut nets and time using Tabu Search (TS) *versus* Simulated Annealing (SA). Columns 10 and 11 show the similar results when Tabu Search (TS) is compared to the Sanchis Interchange approach (IC).

Some general observations can be made from the results presented in Tables III–V. The total execution times to obtain a final partition for the different methods indicate that the new Tabu Search based approach takes the least amount of CPU time whereas the Simulated Annealing approach requires large CPU times. Tabu Search execution time is faster than Simulated Annealing by a factor of 5–20. The quality of the test results on MCNC benchmark circuits are very promising in most cases. Tabu Search yields netlist partitions that contain 20%–67% fewer cut nets and are generated 2/3 to 1/3 times faster than the best netlist partitions obtained by using an interchange method. Comparable partitions to those obtained by Simulated Annealing are obtained 5 to 20 times faster.

5.4. Tabu Search *Versus* HMETIS

As has been mentioned in Section 2, the main goal of this paper is to:

- explore the effectiveness of Tabu Search on partitioning,
- explain the different parameter settings involved in designing the Tabu Search heuristic,
- compare the Tabu Search Meta Heuristic to two widely used techniques in CAD tools *i.e.*, Sanchis Interchange and Simulated Annealing.

Nevertheless, we present some results comparing HMETIS [25, 23] the state of the art partitioner based on *multilevel* clustering with an efficient Tabu Search implementation (C-TS) which uses GRASP [29] (for obtaining good initial solutions) and a single level clustering [4] technique. As is evident from the Tables IX–XI the results obtained *via* C-TS approach are comparable to the results obtained by HMETIS (worst by 1% on average). As the number of partitions increase from 2 to 6 the performance of the Tabu Search implementation improves and gives on average better results by 0.5%. The CPU time used by the C-TS approach is about 40% more than that used by HMETIS. This is basically due to the fact that Hmetis uses multilevel clustering which is quite effective in reducing the solution space of the benchmarks (especially ind2 and ind3). A new version of C-TS is currently being developed based on multilevel clustering.

TABLE IX 2 way partitioning (TS *vs* HMETIS)

	C-TS		HMETIS		
Circuit	Cut	Time	Cut	Time	Improvement
Chip1	20	0.8	20	0.7	–
Chip2	14	0.5	14	0.7	–
Chip3	6	0.3	6	0.5	–
Chip4	7	0.3	7	0.6	–
Prim1	54	2.1	53	1.7	−1.8%
Prim2	134	5.3	146	6.2	+8.2%
Ind1	24	6.3	21	3.3	−12.5%
Bio	93	24.5	83	6.7	−10.7%
Ind2	191	34.2	197	25.7	+3.0%
Ind3	291	78.3	282	36.2	−3.1%
Total	834	152.6	829	82.3	−0.6%

TABLE X 4 way partitioning (Ts *vs* HMETIS)

	C-TS		HMETIS		
Circuit	Cut	Time	Cut	Time	Improvement
Chip1	42	1.2	44	1.8	+4.5%
Chip2	28	1.3	26	1.5	−7.1%
Chip3	29	1.8	28	1.4	−3.4%
Chip4	19	1.9	16	1.3	−14.2%
Prim1	86	7.2	90	3.6	+4.4%
Prim2	287	18.2	279	11.0	−2.7%
Ind1	62	14.2	60	7.2	−3.2%
Bio	123	28.9	126	12.6	+2.3%
Ind2	356	1:34	364	46.9	+2.1%
Ind3	744	2:39	746	70.8	+0.2%
Total	1776	327.1	1779	158.1	−0.16%

TABLE XI 6 way partitioning (Ts *vs* HMETIS)

	C-TS		HMETIS		
Circuit	Cut	Time	Cut	Time	Improvement
Chip1	59	1.8	63	2.5	+6.3%
Chip2	35	1.2	31	2.4	−11.4%
Chip3	39	0.5	35	2.0	−10.2%
Chip4	27	0.4	25	2.0	−7.0%
Prim1	119	2.5	112	5.1	−5.8%
Prim2	334	15.2	341	13.9	+2.0%
Ind1	91	8.1	82	10.0	−9.8%
Bio	159	26.8	162	15.9	+1.8%
Ind2	625	2:17	639	58.2	+2.1%
Ind3	1444	2:42	1456	1:34	+0.8%
Total	2932	355.5	2946	205.4	+0.5%

5.5. Solution Optimality

Here, we undertake to study the ability of the developed clustering-partitioning heuristics in finding optimal solutions to the benchmarks presented in Section 5. The CGA-TS approach is a Tabu Search implementation that is similar to C-Ts (*i.e.*, uses GRASP and a single level of clustering) but in addition utilizes a Genetic Algorithm to replace the diversification and intensification modules used in the traditional Tabu Search implementations. Each problem is formulated as a linear integer program (MIP) as presented in Section 1.1 (Eq. (1)). Benchmark problems are optimally solved using CPLEX Version 3.0 [39], and the results are shown in Tables XII, XIII. It is clear from Tables XII, XIII that the strategy of combining clustering with hybrid search techniques achieves near optimal solutions and are within 4% of optimality in the worst case.

Tables XII, XIII also present a lower bound obtained by solving the LP relaxation of the problem. The column in these tables denoted by LP-BOUND reads the solution of the LP relaxation as compared to the total nets of the circuit. The difference indicates the lower bound of the partitioning problem.

5.5.1. Improving CPLEX MIP Performance

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

The branch-and-bound tree may be as large as 2^n nodes, where n is the number of binary variables. A problem containing only 30 binary variables could produce a tree having over one billion nodes. If no other stopping criteria have been set, the process might continue for a long time, until the search is complete or the computer memory is exhausted. Therefore, many issues are to be investigated to improve the performance of the MIP solver.

The first issue is related to different parameter settings of the MIP solver. It is important to

TABLE XII Optimal solutions for 2-way partitioning

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

TABLE XIII Optimal solutions for 4-way partitioning

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

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

Figure 9 summarizes the methodology used to improve the performance of the CPLEX MIP solver for the circuit partitioning problem. We call this method Stat-Heur for the CPLEX MIP solver. As seen in the figure, we make use of our heuristic based techniques in providing a lower bound that is used to cut off any nodes that have an objective value below the value supplied by the heuristic. We also utilize the solutions obtained by our heuristics to fix a few modules in the original formulation. Given that we are maximizing the sum of continuous variables, it seems that setting a binary variable to 1 will have the most favorable impact if that variable can enable a larger number of continuous variables to take on a value of 1. Another important issue worth investigating is the symmetry of the MIP problem. With the current formulation, the problem is extremely symmetric. There is very little to distinguish one variable from another, both in terms of the objective function and the constraints. This feature can slow the MIP performance, because when we branch down on a variable, we can easily shift it's activity level to another variable. Reducing the symmetry of the MIP, reduces the number of equivalent solutions, which will reduce the amount of work needed in the branch and bound process. One way to reduce the symmetry is to formulate with patterns instead of assignments. Another means of accomplishing this is through priority orders. Priority orders provide a very powerful mechanism for adding user-supplied, problem-specific direction to the branching process. The additional information in the form of a priority order file can be used to alleviate the effect of the symmetry if reformulation is not possible. Statistical information of the circuits have been used extensively to provide such information. A factor Λ that determines the relation between modules and nets is used to give priority to the modules (x variables) that have a high percentage of nets incident on them over variables that have less incident nets.

Since priority orders can supply information on integer variables only, and since the y variables are continuous (see Section 1.1), the priority orders were restricted on the x variables. However, because of the nature of our circuit partitioning problems, we managed to declare the y variables to be integer without changing the meaning of our model. Although one's intuition would say it's a mistake to declare more integer variables, this is not always the case, especially if we could provide some priority order information regarding the y variables. For the y variables, Δ, the number of modules incident on the nets is used to provide such information. The higher the Δ factor the higher the priority given to the y variable.

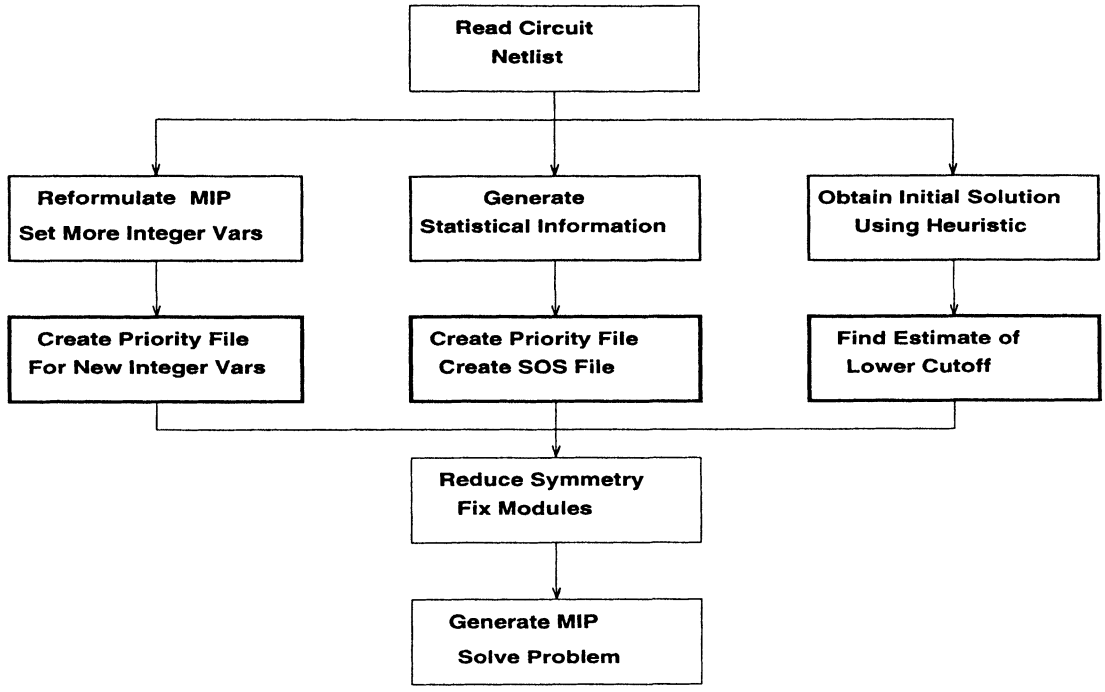


FIGURE 9 Stat-Heur heuristic and CPLEX MIP.

Finally, a highly effective method to improve performance is to take advantage of the presence of *Special Ordered Sets* (SOSs). Special branching strategies are available to take advantage of SOSs. These strategies depend upon the SOS problem definition to include ordering (or weighting) information. If there is no order or weight assigned to individual SOS members, using SOS branching strategies may not improve – and can even degrade-performance. It is important to note in Figure 9 that when SOSs are used after specifying a priority order file on individual variables, CPLEX ignores the order on individual variables because SOS sets involve a branching procedure

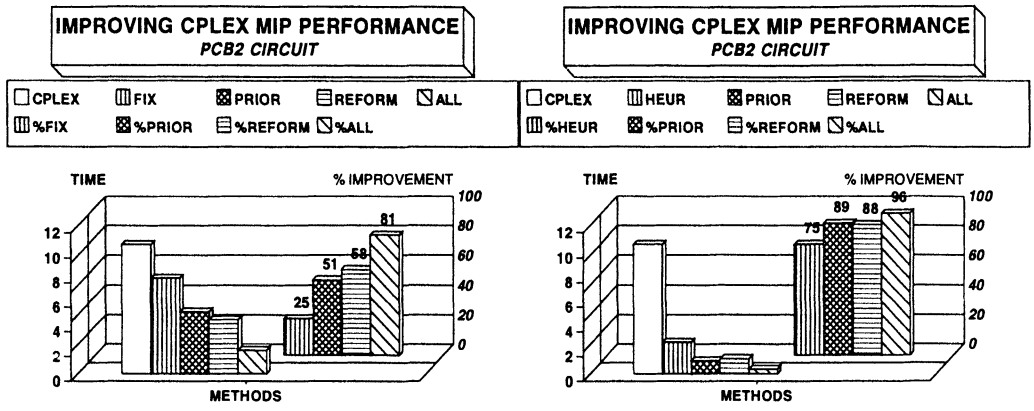


FIGURE 10 Effect of Stat-Heur in improving the CPLEX MIP performance.

TABLE XIV Heur-Stat heuristic used with CPLEX MIP solver

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

where the SOS sets are branched on, not individual variables. Therefore, SOSs are not specified when using priority orders.

Figure 10 indicates clearly the impact of using our heuristic based approaches in improving the performance of the MIP solver. The figure shows that for the PCB2 circuit (a circuit with 24 modules and 32 nets) up to 96% reduction in computation time was achieved using the Stat-Heur technique. Table XIV presents the amount of reduction in computation time for different circuits based on 2 and 4 partitions.

6. CONCLUSION

The new research presented in this paper involved exploring the effectiveness of Tabu Search as a new search methodology to the problem of circuit partitioning in circuit layout VLSI design. The exploration was an attempt to implement and compare Tabu Search technique with other heuristic methods that can produce good solutions for this problem. 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.

The promising results reveal that the Tabu Search method gives a good compromise between the quality of final partitions and time required to provide the solution. The Tabu Search method yields the best final partitions with respect to interchange methods and Simulated Annealing. The quality of the test results on MCNC benchmark circuits are very promising in most cases. Tabu Search yields netlist partitions that contain 20%–67% fewer cut nets and are generated 2/3 to 1(1/2) times faster than the best netlist partitions obtained by using an interchange method. Comparable partitions to those obtained by Simulated Annealing are obtained 5 to 20 times faster. A hybrid clustered Tabu Search implementation C-TS was also presented and compared to HMETIS (state of the art partitioner). Results presented show that C-TS is capable of producing very similar results for 2,4 blocks and could produce better cutset as the number of blocks increased to 6. The final goal of this paper was to asses the quality of solutions obtained by the developed combined Tabu Search and clustering techniques. Results indicate clearly that most real world problems are too complex for any single processing technique to solve in isolation and that the modern philosophy is to use hybrids.

References

[1] Glover, F. (1990). Tabu Search Part I. *ORSA Journal on Computing*, **1**(3), 190–206.
[2] Lengauer, T., *Combinatorial Algorithms for Integrated Circuit Layout*, John Wiley & Sons, New York, 1990.
[3] Garey, M. R. and Johnson, D. S., *Computers and Intractability*, Freeman, San Francisco CA, 1979.
[4] Areibi, S. and Vannelli, A., An Efficient Clustering Technique for Circuit Partitioning. In: *IEEE International Symposium on Circuits and Systems*, pp. 671–674, San Diego, California, 1996.
[5] Dutt, S. and Deng, W., VLSI Circuit Partitioning by Cluster-Removal Using Iterative Improvement

Techniques. In: *IEEE International Conference on CAD*, pp. 194–200. ACM/IEEE, (1996).
[6] Chan, P. K., Schlag, D. F. and Zien, J. Y. (1994). Spectral K-way Ratio-Cut Partitioning and Clustering. *IEEE Transactions on Computer Aided Design*, **13**(9), 1088–1096.
[7] Riess, B. M., Doll, K. and Johannes, F. M., Partitioning very large circuits using analytical placement techniques. In: *Proceedings of 31st DAC*, pp. 646–651, Las Vegas, Nevada, 1994, ACM/IEEE.
[8] Kernighan, B. W. and Lin, S., An Efficient Heuristic Procedure for Partitioning Graphs. *The Bell System Technical Journal*, **49**(2), 291–307, February, 1970.
[9] Fiduccia, C. M. and Mattheyses, R. M., A Linear-Time Heuristic for Improving Network Partitions. In: *Proceedings of 19th DAC*, pp. 175–181, Las Vegas, Nevada, June, 1982, ACM/IEEE.
[10] Krishnamurthy, B., An Improved Min-Cut Algorithm for Partitioning VLSI Networks. *IEEE Transactions on Computers*, **33**(5), 438–446, May, 1984.
[11] Sanchis, L. A., Multiple-Way Network Partitioning. *IEEE Transactions on Computers*, **38**(1), 62–81, January, 1989.
[12] Pothen, A., Simon, H. D. and Liou, K. P. (1990). Partitioning Sparse Matrices with Eigenvectors of Graphs. *SIAM Journal on Matrix Analysis*, **11**, 430–452.
[13] Sechen, C. and Chen, D. (1988). An improved Objective Function for Min-Cut Circuit Partitioning. *IEEE Transaction on CAD*, pp. 502–505.
[14] Hadley, S. W., Mark, B. L. and Vannelli, A., An Efficient Eigenvector and Node Interchange Approach for Finding Netlist Partitions. *IEEE Transactions on CAD/ICAS*, **11**(7), 885–892, July, 1992.
[15] Barnes, E. R., An Algorithm for Partitioning the Nodes of a Graph. *SIAM Journal of Algebraic and Discrete Methods*, **3**(4), 541–550, December, 1982.
[16] Kirkpatrick, S., Gelatt, C. D. and Vecchi, M. P., Optimization by Simulated Annealing. *Science*, **220**(4598), 671–680, May, 1983.
[17] Areibi, S. and Vannelli, A., Circuit Partitioning Using a Tabu Search Approach. In: *IEEE International Symposium on Circuits and Systems*, pp. 1643–1646, Chicago, Illinois, 1993.
[18] Dutt, S. and Deng, W., A Probability-Based Approach to VLSI Circuit Partitioning. In: *Proceedings of 33rd DAC*, Las Vegas, Nevada, June, 1996, ACM/IEEE.
[19] Caldwell, A., Kahng, A. and Markov, I., Relaxed Partitioning Balance Constraints in Top-Down Placement. In: *Proceedings IEEE International Conference on ASIC*, pp. 229–232, June, 1998.
[20] Cherng, J. and Chen, S., A Stable Partitioning Algorithm for VLSI Circuits. In: *IEEE Custom Integrated Circuits Conference*, pp. 9.1.1–9.1.4, San Diego, California, 1996, IEEE.
[21] Dasdan, A. and Aykanat, C., Two Novel Multiway Circuit Partitioning Algorithms Using Relaxed Locking. *IEEE Transaction on Computer Aided Design of Integrated Circuits and Systems*, **16**, 169–178, February, 1997.
[22] Hagen, L., Huang, D. and Kahng, A., On Implementation Choices for Iterative Improvement Partitioning Algorithms. *IEEE Transaction on Computer Aided Design of Integrated Circuits and Systems*, **16**, 1199–1205, October, 1997.
[23] Karypis, G., Aggarwal, R., Kumar, V. and Shekhar, S., Multilevel Hypergraph Partitioning: Application in VLSI Design. In: *Proceedings of 35th DAC*, pp. 526–529, Las Vegas, Nevada, June, 1997, ACM/IEEE.
[24] Hagen, L. and Kahng, A., Fast Spectral Methods for Ratio Cut Partitioning and Clustering. In: *IEEE International Conference on CAD*, pp. 10–13, Santa Clara, California, November, 1991, IEEE.
[25] Alpert, C. J., Huang, J. and Kahng, A., Multilevel Circuit Partitioning. In: *Proceedings of 34th DAC*, pp. 530–533, Anaheim, CA, June, 1997, ACM/IEEE.
[26] Karypis, G. and Kumar, V., Multilevel Graph Partitioning Schemes. In: *Proceedings of the 1995 Intl. Conf. on Parallel Processing*, pp. 113–122. ACM/IEEE, 1995.
[27] Areibi, S., *Ph.D. Thesis*: Towards Optimal Circuit Layout Using Advanced Search Techniques, 1995.
[28] Areibi, S. and Vannelli, A., Advanced Search Techniques for Circuit Partitioning. In: *DIMACS Series in Discrete Mathematics and Theoretical Computer Science*, pp. 77–98, Rutgers State University, New Jersey, 1994.
[29] Areibi, S., GRASP: An Effective Constructive Technique For VLSI Circuit Partitioning. In: *1999 Canadian Conference on Electrical and Computer Engineering*, Edmonton, Alberta, May, 1999, IEEE.
[30] Hadley, S. W., Mark, B. L. and Vannelli, A., An Efficient Eigenvector-Node Interchange Approach for Finding Netlist Partitions. In: *Proceedings of Custom Integrated Circuits Conference*, pp. 28.2.1–28.2.4, San Diego, July, 1991.
[31] Areibi, S. and Vannelli, A., An Efficient Solution to Circuit Partitioning Using Tabu Search and Genetic Algorithms. In: *6th International Conference of Micro Electronics*, pp. 70–74, Istanbul, Turkey, 1994.
[32] Areibi, S. and Vannelli, A., A Combined Eigenvector Tabu Search Approach for Circuit Partitioning. In: *Proceedings of the 1993 Custom Integrated Circuits Conference*, pp. 9.7.1–9.7.4, San Diego, 1993.
[33] Tao, L., Zhao, Y. C., Thulasiraman, K. and Swamy, M. N. S., An Efficient Tabu Search Algorithm for Graph Bi-sectioning. In: *Proceedings/Great Lakes Symposium on VLSI*, pp. 92–95, IEEE, 1991.
[34] Hertz, A. and Werra, D. (1987). Using Tabu Search Technique for Graph Coloring. *Computing*, pp. 345–351.
[35] Skorun-Kapov, J. (1990). Tabu Search Applied to the Quadratic Assignment Problem. *ORSA Journal on Computing*, **2**, 195–202.
[36] Glover, F. (1990). Tabu Search Part II. *ORSA Journal on Computing*, **2**(1), 4–32.
[37] Roberts, K. and Preas, B., Physical Design Workshop. *Technical report*, MCNC, Marriott's Hilton Head Resort, South Carolina, April, 1987.
[38] White, S. R., Concepts of Scale in Simulated Annealing. In: *IEEE Int. Conf. on Computer Design*, pp. 646–651, IEEE, November, 1984.
[39] CPLEX Optimization Inc. *CPLEX Documentation*, CPLEX Optimization, Inc., Tahoe, Nevada, 1993.

Authors' Biographies

Shawki M. Areibi has worked as a research mathematician at Shell International Oil Products in the Hague, The Netherlands. He also served on the faculty of electrical engineering at Ryerson Polytechnic University, Toronto, Canada.

Currently, he is an assistant professor in the School of Engineering at the University of Guelph, Canada. His research interests include distributed processing, VLSI design, Circuit Layout and combinatorial optimization. He is a member of the IEEE, ACM and ISCA.

Tony Vannelli is a professor at the University of Waterloo, Canada. Currently, he is the head of electrical and computer engineering department. His research interests include mathematical programming, Physical Design and graph theory applications. He is a member of IEEE.