A graph partitioning heuristic for the parallel pseudo-exhaustive logical test of VLSI combinational circuits

Alexandre A. Andreatta and Celso C. Ribeiro

Department of Computer Science, Catholic University of Rio de Janeiro, Rua Marquês de São Vicente 225, Rio de Janeiro 22453, Brazil

E-mail: {andreatta,celso}@inf.puc-rio.br

The logical test of integrated VLSI circuits is one of the main phases of their design and fabrication. The pseudo-exhaustive approach for the logical test of integrated circuits consists in partitioning the original circuits to be tested into non-overlapping subcircuits with a small, bounded number of subcircuits, which are then exhaustively tested in parallel. In this work, we present an approximate algorithm for the problem of partitioning integrated combinational circuits, based on the tabu search metaheuristic. The proposed algorithm presents several original features, such as: the use of a reduced neighborhood, obtained from moves involving only a subset of boundary nodes; complex moves which entail several resulting moves, although the variations in the cost function are easily computable; a bi-criteria cost function combining the number of subcircuits and the number of cuts, which simultaneously adds a diversification strategy to the search; and the use of a bin-packing heuristic as a post-optimization step. The behavior of the proposed algorithm was evaluated through its application to a set of benchmark circuits. The computational results have been compared with those obtained by the other algorithms in the literature, with significant improvements. The average reduction rates have been of the order of 30% in the number of subcircuits in the partition, and of the order of 40% in the number of cuts.

Keywords: Integrated circuits, VLSI design, logical test, circuit partitioning, graph partitioning, tabu search.

1. Introduction

The logical test of integrated VLSI circuits is one of the main phases of their design and fabrication. Testing a circuit amounts to submitting it to different input patterns and checking whether the observed outputs are exactly those expected according to the design of the circuit, in order to evaluate if the logical gates are behaving as expected (i.e. producing the correct, desired outputs associated with each input pattern) and to ensure that physical faults do not occur. Among the several approaches available for the logical test of combinational circuits, we may find: (i) exhaustive test, (ii) fault simulation, and (iii) pseudo-exhaustive test. In the latter approach, the circuit to be tested is decomposed into subcircuits with a relatively small, bounded number of inputs. Subsequently, each subcircuit is exhaustively tested. Although it does not cover all possible logical faults, this approach does not depend on a fault simulation model and ensures a 100% fault coverage for single stuck-at faults (lines always fixed at the same logical level).

The pseudo-exhaustive approach for logical testing was introduced in the literature in the 1980's. The first work on this subject seems to be that of Bozorgui-Nesbat and McCluskey [7]. If the original circuit is conveniently partitioned into non-overlapping subcircuits, this approach may be speeded up by testing all subcircuits in parallel. In that case, the total duration of the test will be the same as that of the subcircuit with the largest number of inputs. Patashnik [30] has shown that the decision version of the problem of optimally decomposing a combinational circuit into testable subcircuits is NP-complete. Suitable algorithms are needed for partitioning the original circuit, in order to obtain as few subcircuits as possible (to ensure a high fault coverage and to minimize the number of testers required) and not too many cuts (which would increase too much the cost of the additional hardware which has to be inserted at each point where the original circuit is cut). Roberts and Lala [32] proposed the first general heuristic for this problem. However, their algorithm has some drawbacks [13], due to the nature of an unsuitable implicit objective function which very often leads to solutions which violate the testability condition. New algorithms have recently been proposed by Davis-Moradkhan and Roucairol [13], with significantly better results in terms of the number of subcircuits in the partition.

The main goal of this work consists in the development of a new algorithm for circuit partitioning based on the tabu search metaheuristic, aiming at its use in the framework of the pseudo-exhaustive approach for logical test. The combinational circuits are modelled as acyclic directed graphs. Let T be the available time for testing all subcircuits in parallel. In addition to the circuit to be partitioned, another input data for this problem is the maximum number L of inputs in each subcircuit, such that 2^L test patterns may be generated and applied to the largest subcircuit, and the results compared with those defined during the design phase, in total parallel time less than or equal to T. Then, the partitioning problem amounts to decomposing the circuit to be tested into non-overlapping subcircuits with no more than L inputs each, subject to some connectivity constraints.

The paper is organized as follows. In section 2, we introduce the main aspects of the logical test of integrated circuits. We also give more details about the pseudo-exhaustive approach for logical test. The circuit decomposition problem is formulated in section 3, where the currently existing algorithms for this problem are reviewed. In section 4, we recall the basic elements of the tabu search metaheuristic and we propose a tailored algorithm for the circuit partitioning problem. Issues such as the definition of solutions, moves and their attributes, tabu and candidate lists, cost function and diversification, aspiration and stopping criteria are discussed in detail. The section concludes with the detailed presentation of the heuristic in algorithmic form. Subsequently, in section 5, we present the computational results obtained through the application of the tabu search algorithm to a set of benchmark circuits. The solutions obtained by this heuristic are compared with those given by the other algorithms found in the literature, showing significant improvement both in the number of cuts and in the number of subcircuits. Finally, some conclusions are drawn in the last section.

2. Pseudo-exhaustive approach for logical test

In this section, we first give an overview of logical test procedures for combinational integrated circuits. A broader vision of this subject may be found in references [8,29]. Next, we describe in detail the pseudo-exhaustive approach for logical test.

Combinational circuits are integrated digital circuits where the output at any time is a function depending only on a combination of the current inputs. They implement Boolean functions such as $z = f(x) : \{0, 1\}^n \rightarrow \{0, 1\}^m$, n and m integers, where x is the Boolean input vector (or pattern) and z is the Boolean output vector. A *sequential circuit* is one implementing a sequential function whose outputs depend not only on the current inputs, but also on previous inputs, i.e. on the current state of the circuit.

The logical test of a combinational integrated circuit is a three-step procedure: (i) generation of input test patterns, (ii) application of the test patterns to the circuit, and (iii) comparison of the output vectors with the expected outputs, previously obtained by the application of the same input test patterns to the model of the circuit. Every discrepancy is an error, whose cause is called a *physical fault*. The physical faults may be classified into *logical faults* and *parametric faults*.

Logical faults are those that change the logical function implemented by an element of the circuit. Many physical faults may be modelled as logical faults, e.g. short-circuits and open circuits between signal lines and stuck-at faults (signal lines permanently stuck at some specific logic value). They are also called DC faults, since they may be detected in a frequency smaller than the operating one. In this work, we consider only the detection of logical faults. The three types of logical faults considered here are: (i) stuck-at-0 faults (a line is fixed at the logical level "0"), (ii) stuck-at-1 faults (a line is fixed at the logical level "1"), and (iii) short-circuits between two lines. *Parametric faults* are those arising from changes in the parameters of the circuit, which depend on the technology used for its construction, such as the speed of signal propagation. They cannot be dealt with as logical faults and are also called AC faults, since they may be detected only at the operating frequency. In most cases, they are originated during the manufacturing process.

Two major issues in the logical test of integrated circuits are their *controllability* and *observability*. Controllability concerns the possibility of accessing and applying a complete set of input test patterns to a circuit through external inputs or control points. Observability is the capability of observing the responses of the circuit to different input vectors at some external outputs or control points. The cost of the required additional hardware and the time needed for testing the circuits are other major issues. We may find the following among the approaches available for the logical test of combinational circuits:

- *Exhaustive test.* In this type of test, all 2^n possible input patterns are applied to a circuit with n inputs. Its main advantage comes from the fact that it allows checking for any possible faults, i.e. it ensures the exhaustive coverage of the whole set of logical faults. However, it is not practical for large size circuits, due to the high number of input patterns which must be applied to the circuit.
- *Fault simulation.* This approach was proposed as an alternative to the previous one, aiming at the reduction in the number of test patterns which have to be applied to the circuit. In this case, a simulation model generates the most important to be detected and/or the most likely to happen faults. Next, a set of input patterns which allows the determination of these faults is computed and the behavior of the circuit is evaluated with respect to these inputs. Among other drawbacks of this approach, we should mention the complexity of the determination of both a suitable fault model and an appropriate set of input patterns with a large fault coverage.
- *Pseudo-exhaustive test.* The circuit to be tested is decomposed into subcircuits, each of which with a relatively small, bounded number of inputs. Subsequently, each subcircuit is exhaustively tested. Although it does not cover all possible logical faults, this approach does not depend on a fault simulation model and ensures the coverage of all single stuck-at faults.

The use of fault simulation models for the generation of test patterns proved to be useless in the case of VLSI circuits [7], firstly because the fault model based on the hypothesis of the inexistence of simultaneous multiple faults is no longer valid, while more complex models dealing with multiple faults substantially increase the complexity of the generation of test patterns. The automatic generation of test patterns becomes very costly and, in typical cases, does not provide a sufficiently high fault coverage. Also, an expensive tester is required, since many test patterns are produced by the test generator. Moreover, many testers should be used, since the tester is tied up to a circuit for a long period of time. Finally, the simulation time increases exponentially as the circuit grows in size.

These concerns are not recent. Design techniques appropriate for dealing with these difficulties have followed the increase in the rate of integration. Among the main ideas, we may find the *design for testability* (DFT) and the *autonomous test.* A guide of design techniques aimed at circuit testability may be found in [12]. In the case of VLSI, these ideas evolved to the so-called *built-in self-test* (BIST) technique for the autonomous test. Additional hardware is placed inside the circuit to be tested, in order to reduce the complexity of the external test. This additional hardware (i.e. the tester) should be small when compared to the circuit to be tested. Moreover, it should itself be testable from outside, and should not degrade the performance of the original circuit. McCluskey [24,25] presented an overview of BIST techniques and structures used to replace functions of the external tester.

BIST and DFT techniques are always recommended when field repair costs and tester costs are relevant issues. Moreover, the cost of testing an integrated circuit represents a very small fraction of its design and fabrication costs on an industrial scale. A cost-benefit analysis, taking into account factors such as the increase in the rate of coverage and the reduction in maintenance costs, points out considerable gains which may be obtained from the concern with circuit testability in VLSI design.

Large combinational integrated circuits should then be decomposed into subcircuits with a small, bounded number of inputs, in such a way that each subcircuit may be tested exhaustively. This approach corresponds to the so-called *pseudo-exhaustive test*, which was first proposed by Bozorgui-Nesbat and McCluskey [7]. Circuit decomposition implies cutting some lines and, consequently, in the creation of new inputs and outputs, the so-called *pseudo-inputs* and *pseudo-outputs*. Lines are cut by selector circuits, as illustrated in figure 1. In operating mode, the test line stays at the logical level "0", allowing signal propagation from gate P to gate Q, while inhibiting the pseudo-input. On the contrary, in test model the test line remains at the logical level "1", habilitating the pseudo-input and inhibiting signal propagation from gate P to gate Q. Both in operating mode and in test mode, the pseudo-output may be externally observed.

Advantages and drawbacks of the pseudo-exhaustive test approach are discussed in [2,6,27]. Among the advantages, we notice that it (i) ensures the coverage of all single stuck-at faults, without making use of any fault simulation model, (ii) detects any short-circuit fault, provided that the defective circuit remains a combinational one and that the short-circuit involves two lines in the same subcircuit, (iii) detects all multiple stuck-at faults, provided that they are non-redundant and internal to the same subcircuit, and (iv) does not depend on any fault model and, accordingly, is not limited to the detection of any specific class of faults.

In order to allow that all subcircuits be tested in parallel, the original circuit has to be decomposed into non-overlapping subcircuits, i.e. it has to be partitioned into subcircuits with no gates in common. Then, the total duration of the test will be the same as that of the largest subcircuit in terms of the number of inputs. The circuit partitioning problem is studied in the next section.

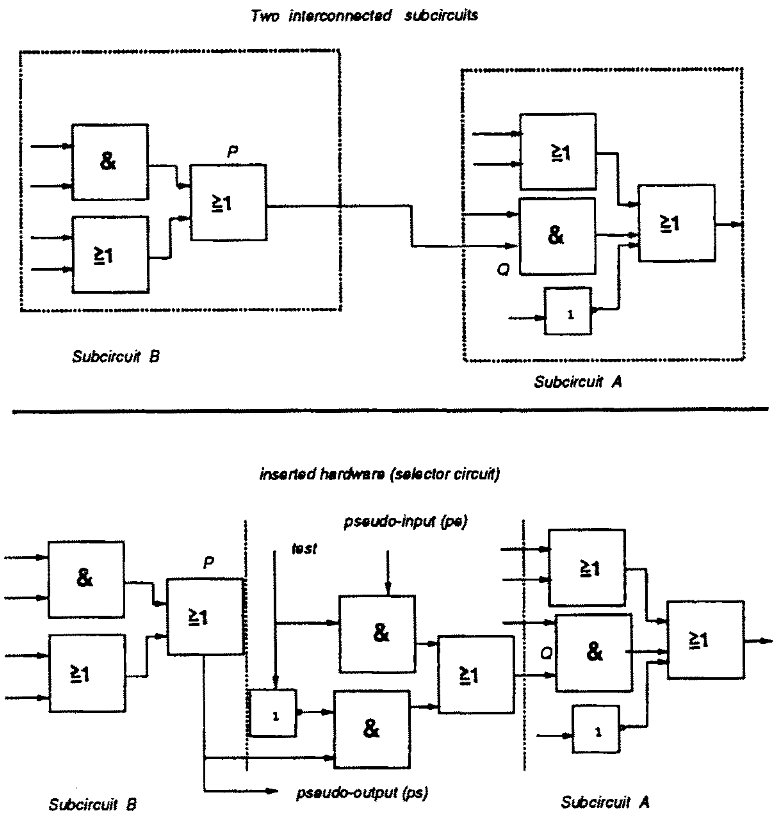

Figure 1. Cut of a line and selector circuit.

3. Circuit partitioning

Given that the total duration of the pseudo-exhaustive test should not exceed a certain time T, let L be a parameter equal to the maximum number of inputs such that 2^L test patterns may be generated and applied to the largest subcircuit, and the outputs compared with the correct ones, in total time less than or equal to T. Then, the circuit partitioning problem consists in finding a decomposition of the circuit to be tested into non-overlapping circuits with no more than L inputs and at least one logical gate each.

Different objective functions may be associated with this decomposition, among them (i) the minimization of the number of cuts, and (ii) the minimization of the number of subcircuits. Very often, a good solution with respect to one criterion is also a good one concerning the other. However, this is not necessarily true and examples illustrating situations where these two objectives are conflicting may be easily constructed [1].

The first criterion corresponds to the minimization of the cost of the additional hardware inserted into the circuit to be tested. We recall that each cut used to partition the original circuit corresponds to the insertion of a selector circuit used for separating the subcircuits while in test mode. Moreover, there is a limit on the maximum numbers of cuts, which depends on the space available on the chip and on the design techniques. There are several reasons for using the second criterion. A smaller number of subcircuits in the partition leads to a higher fault coverage rate. If the pseudo-exhaustive test is performed through external testers, there will be one tester for each subcircuit. On the other hand, if a BIST technique is used in the design of the circuit, each subcircuit will be tested by an embedded linear feedback shift register. In both cases, a smaller number of subcircuits leads to reduced hardware costs.

Let $G = (X, A)$ be the directed acyclic graph associated with a combinational circuit C, where X denotes the set of components (inputs, logical gates, and outputs) and A the set of lines used for signal propagation. The *in-degree* and the *out-degree* of each node $v \in X$ are denoted by $d^-(v)$ and $d^+(v)$, respectively. Given a subset of nodes $V \subset X$, its *input-neighborhood* $\omega^-(V)$ is defined as the set of nodes which are not in V that have at least one successor in V, i.e. $\omega^-(V) = \{v \in X \mid v \notin V$ and $\exists w \in V$ such that $(v, w) \in A\}$. The set of nodes X is formed by three non-empty disjoint subsets E, P, and S, where E is the set of inputs, P is the set of logical gates,

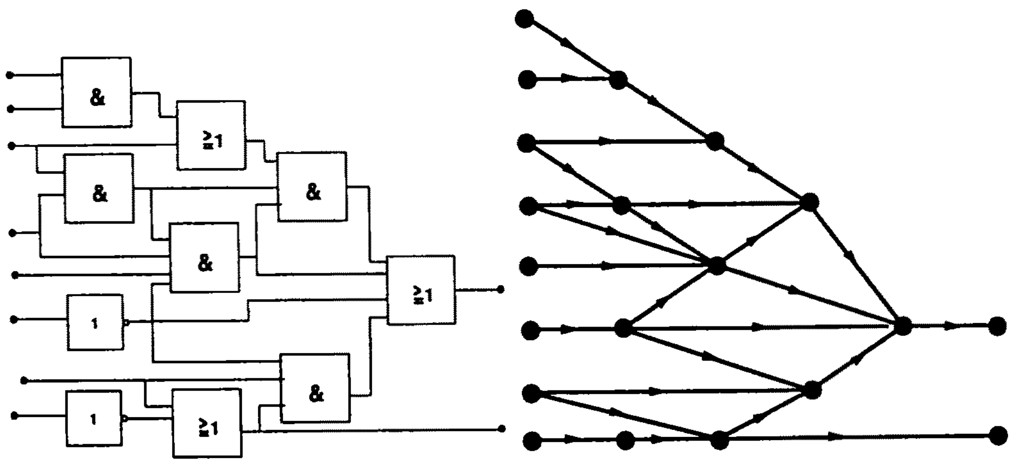

Figure 2. Representation of a combinational circuit by a graph.

and S is the set of outputs of the combinational circuit C. Figure 2 illustrates a combinational circuit and its representation as a directed graph [13].

The problem of partitioning the combinational circuit C, represented by the graph $G = (X, A)$, into testable subcircuits corresponds to finding a partition of X

into a non-fixed number of K subsets X_k, $k = 1, \ldots, K$, such that the induced subgraphs $G_k = (X_k, A_k)$ satisfy the following conditions:

- $X = \cup_{k=1}^{K} X_k$ and $X_k \cap X_\ell = \emptyset$, $\forall k \neq \ell$, $(k, \ell) \in \{1, \ldots, K\}^2$;
- $n_k + c_k \leq L$, $\forall k = 1, \ldots, K$, where $n_k = |X_k \cap E|$ is the number of inputs in G_k and $c_k = |\omega^-(X_k)|$ is the number of gates in the input-neighborhood of X_k which originated pseudo-inputs;
- $X_k \cap P \neq \emptyset$, $\forall k = 1, \ldots, K$; and
- $G_k = (X_k, A_k)$ is either a connected graph or formed by disjoint subgraphs satisfying the above condition, $\forall k = 1, \ldots, K$.

The second condition above ensures the testability of each subcircuit involved in the partition. Each time an arc (i.e. a line of the original circuit) is cut, both a pseudo-input and a pseudo-output are created. Let $G^+ = (X^+, A^+)$ be the augmented graph obtained by the partitioning algorithm, with $X^+ = X \cup E' \cup S'$, where E' is the set of pseudo-inputs and S' is the set of pseudo-outputs. Let $E^+ = E \cup E'$ and $S^+ = S \cup S'$ be, respectively, the set of inputs and outputs of G^+. The graph G^+ consists of K disjoint subgraphs $G_k^+ = (X_k^+, A_k^+)$, $k = 1, \ldots, K$, where the subsets X_k^+ satisfy the above conditions. The testability condition may be represented by the inequality $|X_k^+ \cap E^+| \leq L$, $k = 1, \ldots, K$.

As an example, consider the graph in figure 3. Nodes 1 to 6 are the inputs. Figure 4 illustrates one solution of the partitioning problem for the parameter $L = 4$, with $K = 6$ subcircuits: $X_1 = \{1, 8, 9, 13, 16\}$, $n_1 = 1$ and $c_1 = 2$; $X_2 = \{15, 22, 25\}$, $n_2 = 0$ and $c_2 = 2$; $X_3 = \{23, 26\}$, $n_3 = 0$ and $c_3 = 4$; $X_4 = \{2, 10\}$, $n_4 = 1$ and $c_4 = 1$; $X_5 = \{24, 27\}$, $n_5 = 0$ and $c_5 = 2$; and $X_6 = \{3, 4, 5, 6, 7, 11, 12, 14, 17, 18, 19, 20, 21, 28\}$, $n_6 = 4$ and $c_6 = 0$. The pseudo-inputs and pseudo-outputs are denoted by *pe* and *ps*, respectively, and indexed from 1 to 12.

The graph partitioning problem formulated above may also be modelled as a set partitioning problem with an exponential number of variables, or as a general 0–1 integer programming problem with $O(|X|^3)$ variables. However, as pointed out by Davis-Moradkhan [12], these formulations are not practical for real-size problems. Patashnik [30] has shown through a polynomial transformation from CLIQUE [15] that the decision version of the problem of segmenting a circuit into K testable subcircuits is NP-complete, as well as many restricted versions of it. Several algorithms have been proposed in the literature for circuit decomposition aiming at the pseudo-exhaustive logical test.

The first heuristic for the circuit partitioning problem was proposed by Bhatt et al. [6], who gave a partitioning algorithm for circuits in which the out-degree of every node is less than or equal to its in-degree. Other algorithms for circuits with special structure are also available in the literature. Roberts and Lala [32] have

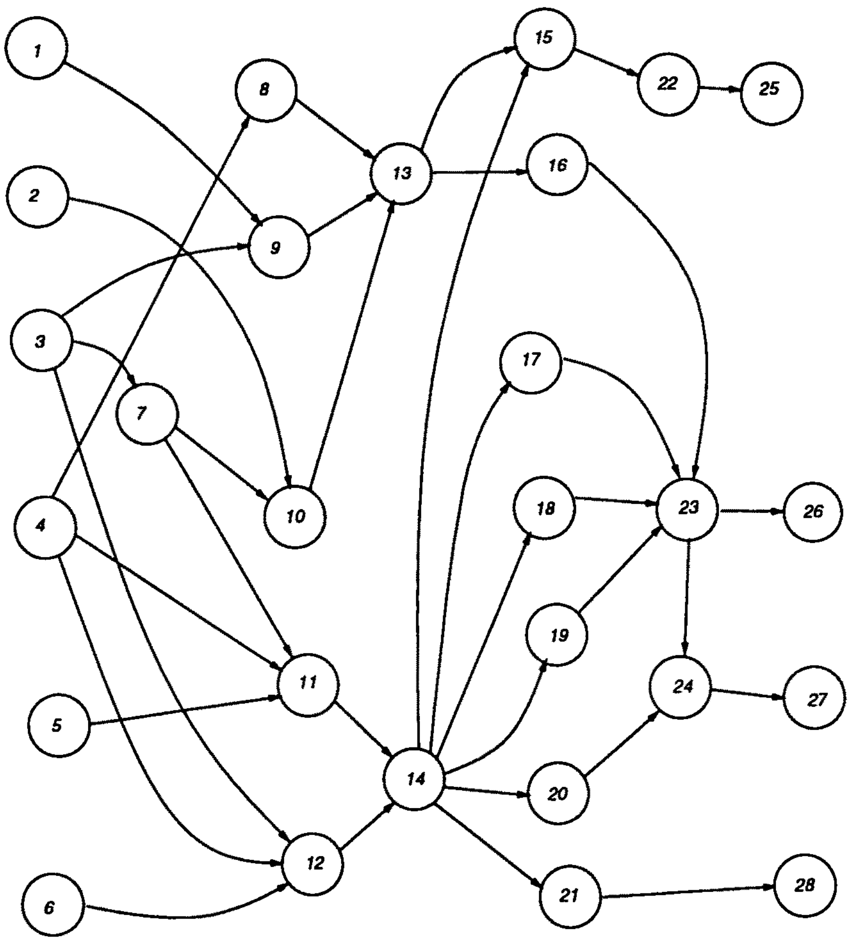

Figure 3. Combinational circuit to be partitioned represented by a graph.

proposed a general heuristic based on the relaxation of the testability condition, in which the total violation $\sum_{k=1}^{K} |L - n_k - c_k|$ is incorporated into the objective function as a penalty. This algorithm very often obtains solutions which greatly violate the testability condition. Moreover, this violation becomes larger, and many small subcircuits with few inputs are created, when the in-degree of the logical gates increases.

Davis-Moradkhan and Roucairol [12–14] have proposed two heuristics, **asp** and **cep**, for this problem. Both heuristics are constructive and perform better than that of Roberts and Lala. Their time complexity is $O(|P| \cdot |A|)$ and the second one is particularly fast. In the next section, we present a new heuristic for the circuit partitioning problem, based on the tabu search metaheuristic and using the **cep** algorithm of Davis-Moradkhan and Roucairol for the generation of initial solutions.

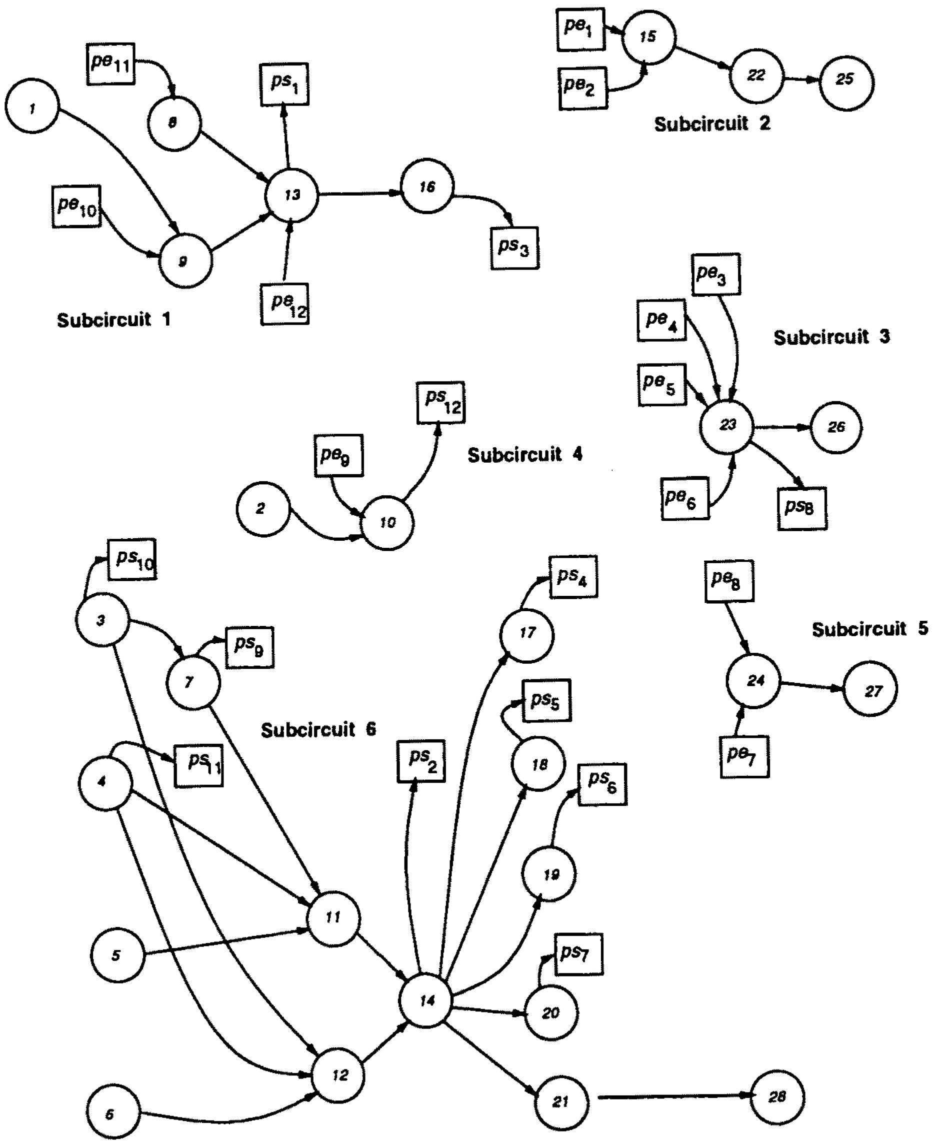

Figure 4. A partition into $K = 6$ subcircuits.

4. A tabu search heuristic

To give a general description of the tabu search metaheuristic, we consider a general combinatorial optimization problem (**P**) formulated so as to

$$\begin{array}{ll} \text{minimize} & c(s) \\ \text{subject to} & s \in S, \end{array}$$

where S is a discrete set of feasible solutions. Local search approaches for solving problem (**P**) are based on search procedures in the solution space S starting from an initial solution $s_0 \in S$. At each iteration, a heuristic is used to obtain a new solution s' in the neighborhood $N(s)$ of the current solution s, through slight changes in s. Every feasible solution $\bar{s} \in N(s)$ is evaluated according to the cost function $c(\cdot)$, which is eventually optimized. The current solution moves smoothly towards better neighbor solutions, enhancing the best obtained solution s^*. The basic local search approach corresponds to the so-called hill-descending algorithms, in which a monotone sequence of improving solutions is examined, until a local optimum is found.

A move is an atomic change which transforms the current solution s into one of its neighbors, say $\bar{s}$. Thus, *move_value* $= c(\bar{s}) - c(s)$ is the difference between the value of the cost function after the move and the value of the cost function before the move. Hill-descending algorithms always stop in the first local optimum. To avoid this drawback, several metaheuristics have been proposed in the literature, namely genetic algorithms, neural networks, simulated annealing, and tabu search [20]. They all have an essential common approach: the use of certain mechanisms which permit that the search for neighbor solutions takes directions of increasing the cost of the current solution in a controlled way, as an attempt to escape from local optima. Among them, tabu search is an adaptive procedure for solving combinatorial optimization problems, which guides a hill-descending heuristic to continue exploration without becoming confounded by the absence of improving moves, and without falling back into a local optimum from which it previously emerged [17–19,23]. At every iteration, an admissible move is applied to the current solution, transforming it into its neighbor with the smallest cost. Moves towards a new solution that increase the cost function are permitted. In that case, the reverse move should be prohibited along some iterations, in order to avoid cycling. These restrictions are based on the maintenance of a short-term memory function which determines how long a tabu restriction will be enforced or, alternatively, which moves are admissible at each iteration. Figure 5 gives a procedural description of the basic tabu search metaheuristic.

The tabu tenure is an important feature of the tabu search algorithm, because it determines how restrictive is the neighborhood search. The performance of an algorithm using the tabu search metaheuristic is intimately dependent on the basic characterizing parameters, namely the time that the short memory function enforces a certain move to be tabu, and the maximum number of iterations *max_moves* during which there may be no improvement in the best solution. If the tabu tenure is too small, the probability of cycling increases. If it is too large, there is a possibility that all moves from the current solution are tabu and the algorithm may be trapped. However, it should be pointed out that cycle avoidance is not an ultimate goal of the search process. In some instances, a good search path will result in revisiting a solution encountered before. The broader objective is to continue to

```
Algorithm Tabu-Search
begin
    Initialize the short term memory function
    Generate the initial solution $s_0$
    $s, s^* \leftarrow s_0$
    while (number of moves without improvement < $max\_moves$) do
    begin
        $best\_move\_value \leftarrow \infty$
        for each ($candidate\_move$) do
        begin
            if ($candidate\_move$ is admissible) then
            begin
                Obtain the neighbor solution $\bar{s}$ by applying $candidate\_move$ to the current solution $s$
                $move\_value \leftarrow c(\bar{s}) - c(s)$
                if ($move\_value < best\_move\_value$) then
                begin
                    $best\_move\_value \leftarrow move\_value$
                    $s' \leftarrow \bar{s}$
                end_if
            end_if
        end_for
        if ($best\_move\_value > 0$) then update the short term memory function
        if ($c(s') < c(s^*)$) then $s^* \leftarrow s'$
        $s \leftarrow s'$
    end_while
end_Tabu-Search
```

Figure 5. Basic description of the tabu search metaheuristic.

stimulate the discovery of new high-quality solutions. One implication of choosing stronger or weaker tabu restrictions is to render smaller or longer tabu tenures appropriate [21].

For large problems, in which $N(s)$ may have too many elements, or for problems where these elements may be costly to examine, the aggressive choice orientation of tabu search makes it highly important to isolate a candidate subset of the neighborhood, and to examine this subset instead of the entire neighborhood [21]. Other advanced features, improvements and extensions to the basic tabu search procedure will be commented on in the next sections, in which the basic tabu search heuristic is specialized into a tailored algorithm for the solution of the circuit partitioning problem.

4.1. SOLUTIONS, NEIGHBORHOOD, AND CANDIDATE LISTS

Each *solution* s of the circuit partitioning problem for the circuit graph $G = (X, A)$ is represented by the augmented graph $G^+ = (X^+, A^+)$, formed by the subgraphs $G_k^+ = (X_k^+, A_k^+)$, $k = 1, \dots, K$, where $X^+ = X \cup E' \cup S'$, with E' being the set of pseudo-inputs and S' the set of pseudo-outputs. The subgraphs $G_k^+ = (X_k^+, A_k^+)$, $k = 1, \dots, K$, satisfy the following conditions:

- $X^+ = \cup_{k=1}^{K} X_k^+$ and $X_k^+ \cap X_\ell^+ = \varnothing, \forall k \neq \ell,\ (k,\ell) \in \{1,\ldots,K\}^2$;
- $X_k^+ \cap P \neq \varnothing, \forall k = 1,\ldots,K$; and
- $G_k^+ = (X_k^+, A_k^+)$ is either a connected graph or formed by disjoint subgraphs satisfying the above condition, $\forall k = 1,\ldots,K$.

The subgraphs G_k^+ are not enforced to satisfy the testability constraint, now written as $|X_k^+ \cap (E \cup E')| \leq L$. Accordingly, the algorithm is allowed to visit infeasible (i.e. non-testable) solutions.

The *neighborhood* $N(s)$ of the current solution s is formed by all solutions $\bar{s}$ which may be obtained from s by transferring one gate from one of its subcircuits to another one. The target subcircuit may be either an existing one or a new subcircuit, characterizing in the latter case the creation of a new subcircuit. Moving a gate from one subcircuit to another entails several resulting moves, which will be detailed in the next section. As noticed before, the neighbor solutions do not necessarily satisfy the testability condition.

We define the *boundary* of the graph $G_k^+ = (X_k^+, A_k^+)$, $k = 1,\ldots,K$, associated with some subcircuit, as the set of gates $\{p \in X_k^+ \cap P \,|\, \Gamma(p) \cap (E' \cup S') \neq \varnothing\}$ (where $\Gamma(p)$ denotes the set of predecessors and successors of node p within graph G^+), i.e. we say that a gate belongs to the boundary of a subcircuit if it has at least one pseudo-input among its predecessors or one pseudo-output among its successors.

The *reduced neighborhood* of the current solution s is then defined as the subset formed by all neighbor solutions in $N(s)$ which may be obtained by moving only boundary gates. The set of moves in the reduced neighborhood is generated at the first iteration and is updated at each next iteration. Only candidate solutions in this reduced neighborhood are examined. This choice to reduce the size of the neighborhood is based on the idea that, most of the time, the reduction in the number of cuts leads to a smaller number of subcircuits. When the number of subcircuits cannot be reduced, we want to reduce the number of cuts. However, moves based on transferring non-boundary gates will necessarily increase the number of cuts, without any effect in the number of subcircuits. For this reason, they may be discarded for the sake of accelerating the neighborhood search. As a nice consequence, it should be noticed that the number of admissible moves vanishes with the number of iterations performed by the algorithm, as far as the number of cuts diminishes.

4.2. MOVES

We have seen before that each *move* is characterized by taking one gate from the boundary of a source subcircuit and transferring it to another subcircuit. The source subcircuit is necessarily one of those in the current partition, while the target subcircuit may be either an existing one or a new subcircuit created with this move.

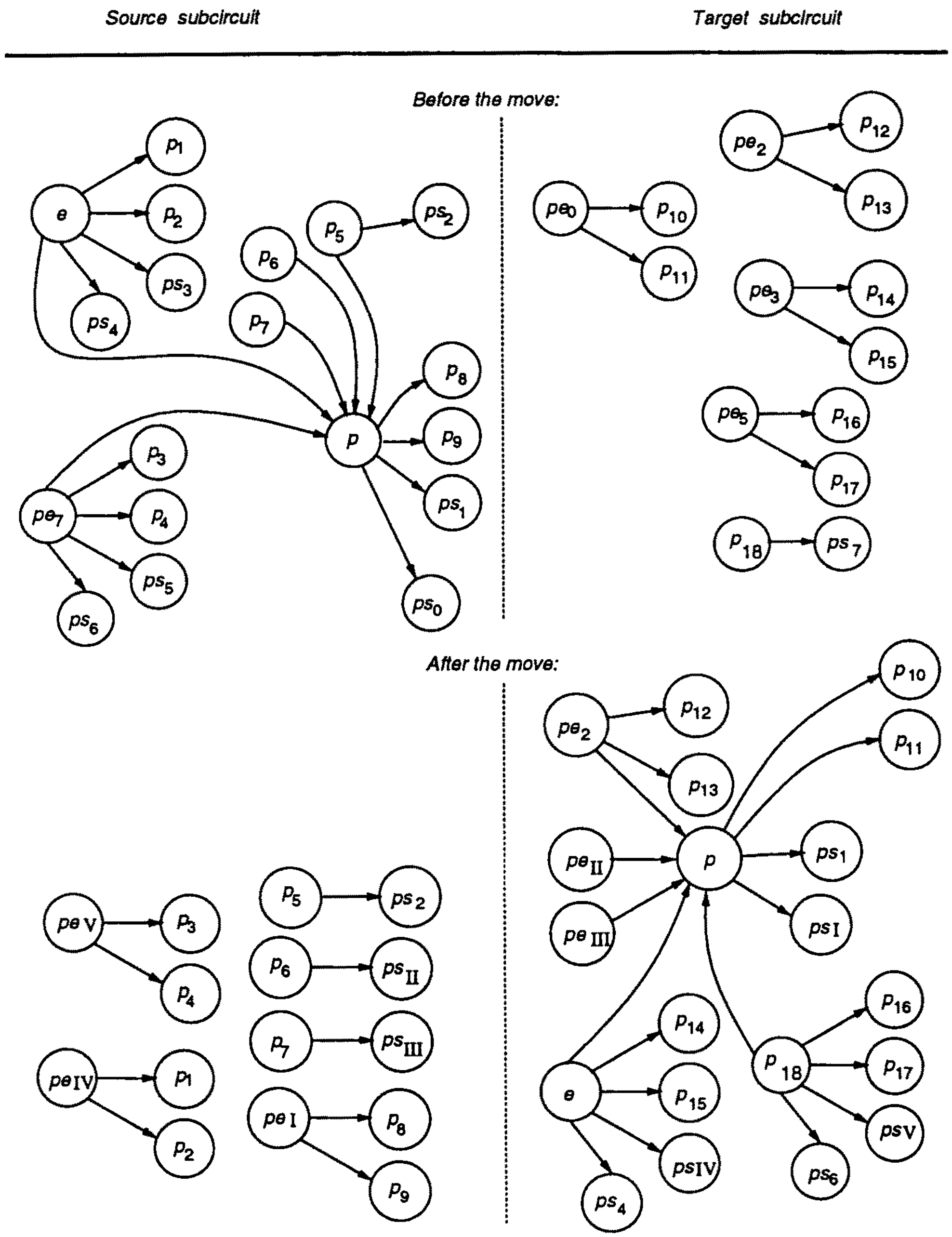

Figure 6. Complete move involving all possible situations.

Whenever a gate p is transferred from the source subcircuit to the target one, it carries to the target subcircuit all its inputs, pseudo-inputs and pseudo-outputs, entailing several resulting moves as a consequence. We illustrate in figure 6 a complete move, in which all the possible situations occur (only the relevant nodes and links are represented in this figure). By the end of the move, five cuts are created (I, II, III, IV, V), while four others are eliminated (0, 3, 5, 7). The steps described below must be carried out during the evaluation of a move (refer to the specific figure for each case, as well as to figure 6 for all examples).

- Analysis of the successors of gate p
  - *Successor pseudo-outputs*: each of them is moved together with p to the target subcircuit. If the corresponding pseudo-input already belongs to it, then the cut is eliminated and the original link is restored (e.g. the old cut (ps_0, pe_0) is eliminated, see figure 7).

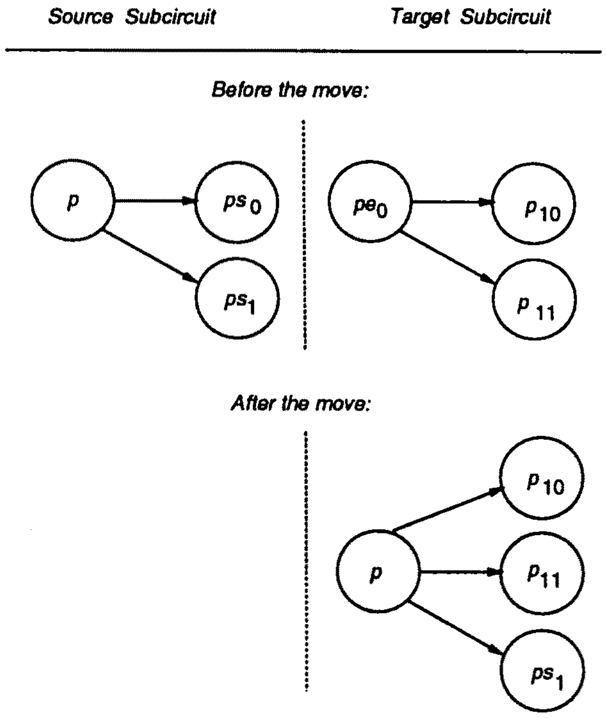

Figure 7. Successor pseudo-outputs.

  - *Successor gates*: they should remain in the source subcircuit. The links between p and each of its successor gates in the source subcircuit are broken, leading to the substitution of each successor gate by a pseudo-output in the target subcircuit, while the corresponding pseudo-input will feed the successor gates in the source subcircuit (e.g. the new cut $(ps_{\mathrm{I}}, pe_{\mathrm{I}})$ separates p from its successor gates p_8 and p_9, which remain in the source subcircuit, see figure 8).

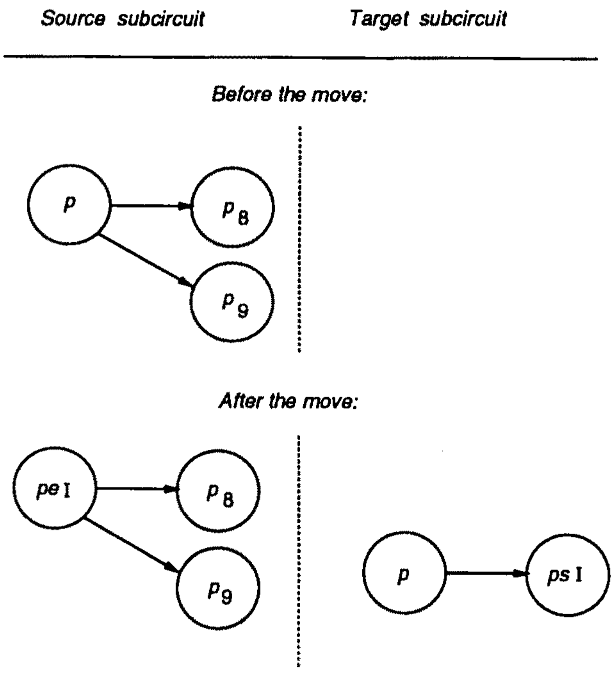

Figure 8. Successor gates.

- Analysis of the predecessors of gate p
  - *Predecessor gates*: as in the previous case, they should remain in the source subcircuit. The links between p and each of its predecessor gates in the source subcircuit are broken, leading to the substitution of each predecessor gate by a pseudo-input which will feed p in the target subcircuit, while the corresponding pseudo-output will be fed by the predecessor gate in the source subcircuit (e.g. the new cuts $(ps_{\mathrm{II}}, pe_{\mathrm{II}})$ and $(ps_{III}, pe_{\mathrm{III}})$ separate p, respectively, from its predecessor gates p_6 and p_7, which remain in the source subcircuit, see figure 9). If any predecessor gate of p already has a successor pseudo-output whose corresponding pseudo-input belongs to the target subcircuit, then the link may be established through this old cut, without it being necessary to create a new one (e.g. gate p_5 and the old cut (ps_2, pe_2)).
  - *Predecessor inputs and their successor gates and pseudo-outputs*: each predecessor input of p is moved together to the target subcircuit (see figure 10). If some predecessor input has other successors different from p itself, they are treated as the successors of p (e.g. input node e, whose move leads to the creation of the new cut $(ps_{\mathrm{IV}}, pe_{\mathrm{IV}})$, as well as to the elimination of the old cut (ps_3, pe_3)).

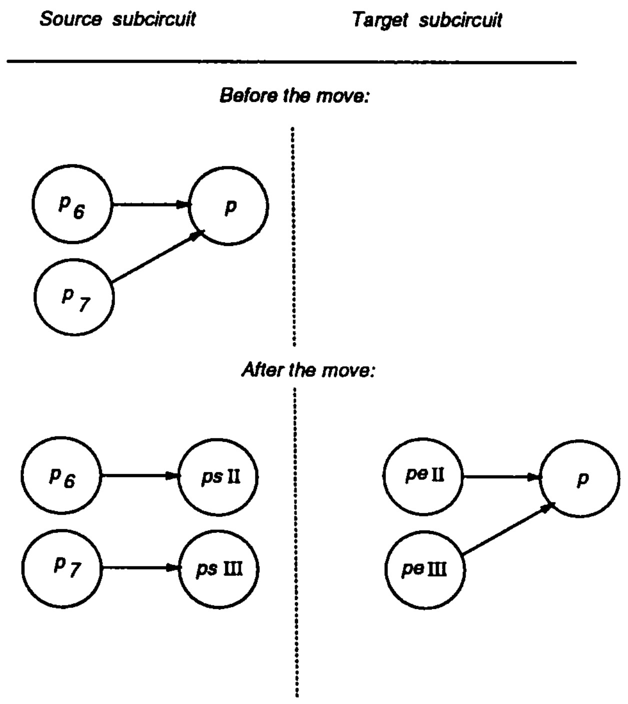

Figure 9. Predecessor gates.

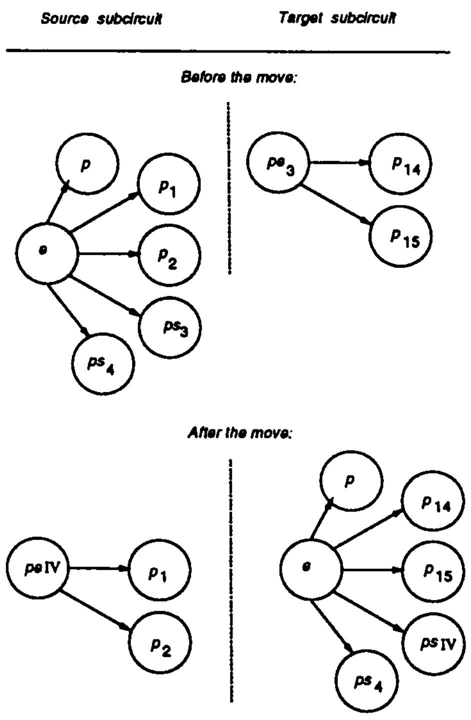

Figure 10. Predecessor inputs and their successor gates and pseudo-outputs.

- *Predecessor pseudo-inputs and their successor gates and pseudo-outputs*: the case of the predecessor pseudo-inputs is very similar to the previous one. The only difference occurs when the corresponding pseudo-output belongs to the target subcircuit, when additional steps must be carried out to establish the links between the predecessors of the pseudo-output and the successors of the corresponding pseudo-input (e.g. pseudo-input pe_7, whose move leads to the creation of the new cut (ps_V, pe_V) due to maintaining p_3 and p_4 in the source subcircuit, as well as to the elimination of the old cuts (ps_5, pe_5) and (ps_7, pe_7)).

4.3. MEMORY FUNCTION, ASPIRATION AND STOPPING CRITERIA

We adopted a flexible memory function in our tailored tabu search algorithm for the circuit partitioning problem. A simple, but strongly restrictive attribute is used to determine the tabu status of each move. Every time a non-improving move is performed, all other moves involving the same associated gate p will be made tabu for the next *tabu_tenure* iterations. The computation of *tabu_tenure* is dynamically performed as

$$\mathit{tabu\_tenure} = (\mathrm{maximum}_{v \in X}\{d(v)\} - \mathit{current\_degree}(v)) \cdot \gamma,$$

where *current_degree*(v) denotes the current degree of node v in the extended graph $G^+ = (X^+, A^+)$, which may vary as long as new pseudo-inputs and pseudo-outputs are created. The tabu tenure of each move then depends not only on the gate itself which was moved, but also on the current iteration. The term $\mathrm{maximum}_{v \in X}\{d(v)\}$ gives the maximum degree among all nodes in the original graph to be decomposed. The larger the degree of a gate in the current extended graph, the larger its potential to affect the search, since many cuts may be created or destroyed when a move involving this gate is performed. Accordingly, moves involving gates that are likely to more affect the search are made tabu for a shorter number of iterations than those involving gates with few adjacent nodes. The parameter γ must be tuned and it is likely to assume larger values for larger graphs.

One implication of choosing stronger or weaker tabu restrictions is to render shorter or longer tabu tenures appropriate [21]. Other search strategies could be entailed by taking other move attributes to define its tabu status, such as the less restrictive ones defined by the pairs (gates, target subcircuit) or (gate, source subcircuit), or those more restrictive defined only by the target subcircuit or the source subcircuit.

Occasionally, it may be interesting that the tabu status of some moves be overriden as a result of more information gathered during the search. Two such situations have been identified in the framework of the circuit partitioning problem. First, the so-called aspiration criterion: a tabu move may be applied whenever it leads to a solution improving the best one found so far. Second, it may happen that the set of admissible moves at some iteration be empty, i.e. all moves are tabu: in this case, the adopted solution corresponds to the reinitialization of the short-term memory function, getting rid of the complete tabu list and restarting the algorithm again with no restrictions.

The overall computational time necessary for obtaining a good partition is not a major issue, since it is just a small fraction of the total time spent in the design of the circuit. Accordingly, here we are interested in developing a good algorithm in terms of solution quality, even if the computational times are long. The stopping criteria will then be verified whenever the number of moves without improvement in the best solution or the overall number of iterations attain some maximum limits. Those limits have been empirically set as, respectively, $max\_moves = 3 \cdot |P|$ and $max\_iterations = \lceil |P|^{(1.2)} \rceil$.

4.4. COST FUNCTION AND DIVERSIFICATION

It was shown in section 3 that the number of subcircuits and the number of cuts are the basic criteria to be optimized in the circuit partitioning problem. As fault coverage and the cost of the testers are the most relevant issues, the number K of subcircuits in the partition is the primary criterion to be minimized. In order to guide the search when improving moves with respect to the primary criterion do not exist, the number *n_cuts* of cuts is also incorporated into the objective function.

The number of subcircuits is weighted by a large constant coefficient α, in order to give a larger weight to the primary criterion and in such a way that improving moves with respect to it are not discarded due to the existence of moves reducing the number of cuts which increase the number of subcircuits. Andreatta [1] has shown that the number of cuts may be reduced by at most $2d^-_{max} + 1$ due to a single move, where $d^-_{max} = \text{maximum}_{v \in X}\{d^-(v)\}$ is the largest in-degree among all nodes representing gates. Then, we should take $\alpha > 2d^-_{max} + 1$ in order to ensure that the first criterion be always privileged with respect to the second one.

Small violations of the testability condition $|X^+_k \cap E^+| \leq L$ may be allowed for some subcircuits $k = 1, \dots, K$, as far as they can be largely compensated by the possible reduction in the number of subcircuits. Allowing the algorithm to visit infeasible solutions also introduces a diversification component into the search. An exponential penalization term $\sum_{k=1}^{K} 2^{deviation^+(k)}$ is incorporated into the objective function as the third criterion, where $deviation^+(k) = \text{maximum } \{0, |X^+_k \cap E^+| - L\}$ is the amount by which the testability condition of subcircuit G^+_k is violated.

The use of an exponential penalty term is coupled with its multiplication by a constant weight β, in order to completely avoid large violations. The ratio between the coefficients α and β determines the maximum violation of the testability condition allowed for each subcircuit in the partition. A move leading to a non-testable solution may only be accepted if other moves reducing the number of subcircuits do not exist. Andreatta [1] has shown that one should take $\alpha \geq 2\beta$ if the maximum allowed deviation from L is fixed as equal to two. Therefore, the cost function to be globally minimized throughout the search is

$$c(s) = \alpha K + n\_cuts + \beta \sum_{k=1}^{K} 2^{deviation^{+}(k)},$$

where s is any solution to the circuit partitioning problem, be it feasible or not. If s is a feasible solution, then its cost is $c(s) = (\alpha + \beta)K + n\_cuts$.

4.5. POST-OPTIMIZATION

A complete description of the tabu search algorithm **TS-CPP** for the circuit partitioning problem is given in figure 11, incorporating all aspects previously discussed in this section. Two additional procedures are incorporated as post-optimization steps, following the application of the tabu search heuristic:

- Procedure **make_feasible** is used whenever the tabu search strategy ends the search failing to find an improving feasible solution, with respect to the initial one. This procedure builds a feasible (i.e. testable) solution from the best infeasible solution minimizing $c(s)$ visited during the search.
- Procedure **pack_together** is used to pack together small subcircuits appearing in the best feasible solution, which globally do not violate the testability condition. A list bin-packing heuristic is used, coupled with a mechanism to evaluate the possible reduction in the number of cuts whenever two subcircuits are packed together.

5. Computational results

The tabu search algorithm **TS-CPP** was applied to nine benchmark combinational circuits presented by Berglez and Fujiwara [4]. The objective was twofold: first, to tune the parameter values for the tabu search algorithm; second, to compare and evaluate its efficiency with respect to other algorithms proposed in the literature. In table 1, we give the basic description of each circuit: the number of inputs, gates, outputs, and links, as well as the maximum in-degree d^{-}_{max} and the maximum out-degree d^{+}_{max} among all gates in the circuit.

Algorithm **TS-CPP** was coded in C. The codes of algorithms **asp** and **cep**, also in C, are those kindly given by their authors, M. Davis-Moradkhan and C. Roucairol. Extensive numerical results obtained on a Sun SPARCstation-2 and reported by Andreatta [1] are available upon request from the authors. We also notice that the weights of the cost function have been fixed throughout all computational experiments at $\alpha = 50$ and $\beta = 20$. These values satisfy the conditions established in section 4.4 and their ratio ensures that no subcircuit will have more than $L + 2$ inputs and pseudo-inputs.

```
Algorithm TS-CPP
begin
   Initialize the short term memory function
   Generate the initial solution $s_0$
   $s, s^* \leftarrow s_0$
   $non\_improving\_moves \leftarrow 0$
   $current\_iteration \leftarrow 0$
   $best\_unfeasible \leftarrow \infty$
   Determine the set of candidate moves in the reduced neighborhood of the current solution (boundary gates)
   while ($non\_improving\_moves < max\_moves$ and $current\_iteration < max\_iterations$) do
   begin
      $best\_move\_value \leftarrow \infty$
      for each ($candidate\_move$) do
      begin
         if ($candidate\_move$ is admissible or $candidate\_move$ satisfies the aspiration criterion) then
         begin
            Obtain the neighbor solution $\bar{s}$ by applying $candidate\_move$ to the current solution $s$
            $move\_value \leftarrow c(\bar{s}) - c(s)$
            if ($move\_value < best\_move\_value$) then
            begin
               $best\_move\_value \leftarrow move\_value$
               $s' \leftarrow \bar{s}$
            end_if
         end_if
      end_for
      if ($best\_move\_value \geq 0$) then update the short term memory function
      if ($c(s') < c(s^*)$) then
      begin
         $non\_improving\_moves \leftarrow 0$
         if (solution $s'$ is feasible) then $s^* \leftarrow s'$
         else begin
            if ($c(s') < best\_unfeasible$) then
            begin
               $unfeasible\_s^* \leftarrow s'$
               $best\_unfeasible \leftarrow c(s')$
            end_if
         end_else
      else $non\_improving\_moves \leftarrow non\_improving\_moves + 1$
      end_if
      $s \leftarrow s'$
      Update the set of candidate moves in the reduced neighborhood
      $current\_iteration \leftarrow current\_iteration + 1$
   end_while
   if ($s^* = s_0$ and $c(s_0) > best\_unfeasible$) then
   begin
      $s_{feasible} \leftarrow$ make_feasible($unfeasible\_s^*$)
      if ($c(s_{feasible}) < c(s^*)$) then $s^* \leftarrow s_{feasible}$
   end_if
   $s^* \leftarrow$ pack_together($s^*$)
end_TS-CPP
```

Figure 11. Tabu search algorithm for the circuit partitioning problem.

Table 1
ISCAS benchmark circuits.

ISCAS circuits	Size	Inputs	Gates	Outputs	Links	d^-_{max}	d^+_{max}
C-1	small	36	153	7	432	9	9
C-2	small	41	170	32	499	5	12
C-3	small	60	357	26	880	4	8
C-4	medium	41	514	32	1355	5	12
C-5	medium	33	855	25	1908	8	16
C-6	medium	157	1129	140	2670	5	11
C-7	medium	50	1647	22	3540	8	16
C-8	large	32	2384	32	6288	2	16
C-9	large	207	3405	108	7552	5	15

The first part of our computational experiments was devoted to tuning the best parameter values and strategies for algorithm **TS-CPP**. Three aspects have been evaluated:

- *Initial solution*. For all small- and medium-size benchmark circuits, both algorithms **asp** and **cep** from Davis-Moradkhan and Roucairol [14] have been applied to the generation of the initial solution. The quality of the best solution found by the tabu search algorithm does not seem to be too much affected by the choice of either one of them. However, since the computational times observed for algorithm **asp** increase with problem size much faster than those of **cep**, only algorithm **cep** was applied to the large-size circuits.
- *Tabu tenure*. The parameter γ involved in the computation of the dynamic tabu tenure of each move (see section 4.3) characterizes the restrictiveness of the search. Larger circuits, for which a wider choice of moves is available, are likely to be better dealt with by taking larger tabu tenures. Accordingly, for the small- and medium-size circuits we have investigated the behavior of the tabu search algorithm by varying γ in the range from 1 to 12. For the large-size circuits, we took γ in the range from 7 to 18. As a general rule, we observed that the most suitable value for γ (i.e. the one leading to the best feasible solution among all those found with the different parameter values) increases with problem size (i.e. with the number of gates and links).
- *Partition parameter L*. In fact, this is not a parameter characterizing the tabu search algorithm, but rather the problems themselves. Strongly constrained problems with small values of L are likely to be more difficult. However, the behavior of the tabu search algorithm does not seem to be too much affected

by the value of L. The observed computational times have been of the same order for L ranging from 15 to 20 for all benchmark circuits. The algorithm seems to be very robust with respect to the partition parameter.

A sample of the numerical results obtained in this first phase is reported below. The computational results obtained for the medium-size circuits for $L = 15$ and $L = 20$ are given, respectively, in tables 2 and 3. The behavior of the algorithm for the small- and large-size circuits, as well as for the other values of the partition parameter L ranging from 16 to 19, is quite the same, and the corresponding lengthy numerical results are omitted for the sake of space. In each of these tables, we report the results obtained by using each initial solution algorithm (**asp** and **cep**) and each value of the parameter γ (associated with the tabu tenure) in the range from 1 to 12. The results in the first block of rows of these tables concern the initial solutions, i.e. the solutions obtained by the algorithms **asp** and **cep** from Davis-Moradkhan [12]: the number of subcircuits, the number of cuts and the computational time in seconds. Next, we report the results obtained by algorithm **TS-CPP** for each initial solution algorithm and each value of the parameter γ: the number of subcircuits, the number of cuts and the computational time in seconds. The best results (in terms of the number of subcircuits in the partition) among all values of γ are reported in bold face type.

We notice that the results reported in tables 2 and 3 for both the initial solution and the tabu search algorithms do not include the application of the **pack_together** procedure or any other similar scheme. They exactly reflect the quality of the solutions produced by the constructive heuristics **asp** and **cep**, against the quality of those produced by the use of tabu search for the solution of the circuit partitioning problem.

The behavior of algorithm **TS-CPP** is further illustrated through the graphics in figures 12 to 15. The iteration counter is represented along the horizontal axis. For each iteration, the value $c(s)$ of the objective function for the current solution s is plotted. The underlying stepwise curve gives the current weighted value αK of the first criterion, i.e. the number of subcircuits in the current partition. Again, we notice that these results do not include the application of the **pack_together** procedure. These figures illustrate the important role played by the objective function proposed in section 4.5. While the heuristic seems to behave as a hill-descending algorithm with respect to the primary criterion, the use of the more complex three-term objective function $c(s)$ seems to be very appropriate. The latter leads algorithm **TS-CPP** to escape from many local optima, guiding the search towards much better solutions which would not be found if only the first criterion was taken into account.

In most of the cases, the best feasible solution found by the basic tabu search algorithm contains several small subcircuits which may be packed together without violating the testability condition, leading to a smaller number of subcircuits. In tables 4 and 5, we report a sample of the final results obtained through the use of

Table 2

Results for the medium-size circuits with $L = 15$.

$L = 15$		C-4		C-5		C-6		C-7	
		asp	cep	asp	cep	asp	cep	asp	cep
Initial	subcircuits	34	25	36	38	52	53	68	71
solution	cuts	166	137	287	299	447	445	851	762
algorithm	seconds	14	4	32	7	50	16	1021	44
TS-CPP	subcircuits	34	23	34	32	50	50	65	67
$\gamma = 1$	cuts	134	108	188	195	341	320	747	636
	seconds	30	28	82	89	143	146	1530	316
TS-CPP	subcircuits	34	14	34	32	49	51	65	66
$\gamma = 2$	cuts	134	108	187	183	321	306	743	625
	seconds	32	66	81	89	141	209	1906	359
TS-CPP	subcircuits	34	**12**	34	32	49	51	64	63
$\gamma = 3$	cuts	137	**106**	185	179	319	303	717	616
	seconds	30	**68**	85	89	144	154	1726	604
TS-CPP	subcircuits	**13**	12	24	**14**	47	51	64	60
$\gamma = 4$	cuts	**113**	121	160	**158**	307	294	692	580
	seconds	**102**	62	284	**296**	164	208	1970	771
TS-CPP	subcircuits	16	13	**15**	15	47	50	64	58
$\gamma = 5$	cuts	125	119	**145**	164	301	290	675	555
	seconds	103	52	**293**	302	201	158	2905	1284
TS-CPP	subcircuits	16	12	15	18	48	49	64	57
$\gamma = 6$	cuts	121	115	151	160	298	279	646	544
	seconds	73	76	295	301	257	169	2966	1290
TS-CPP	subcircuits	14	13	15	15	47	48	56	59
$\gamma = 7$	cuts	132	117	161	170	301	273	609	552
	seconds	61	110	207	299	149	291	2804	1331
TS-CPP	subcircuits	14	17	15	19	46	48	53	58
$\gamma = 8$	cuts	140	126	164	172	285	268	570	548
	seconds	67	114	281	144	152	190	2938	1307
TS-CPP	subcircuits	15	14	16	16	40	48	**50**	47
$\gamma = 9$	cuts	129	140	166	164	239	271	**561**	498
	seconds	70	83	217	227	414	182	**2869**	1356
TS-CPP	subcircuits	15	15	19	21	39	32	51	**46**
$\gamma = 10$	cuts	122	149	167	159	207	211	595	**480**
	seconds	58	69	320	312	453	482	2918	**1353**
TS-CPP	subcircuits	16	14	20	17	35	36	50	59
$\gamma = 11$	cuts	131	135	170	192	203	222	565	557
	seconds	63	51	313	245	448	448	1460	1312
TS-CPP	subcircuits	22	14	18	17	**34**	**31**	51	48
$\gamma = 12$	cuts	130	125	164	170	**209**	**208**	573	496
	seconds	115	51	203	157	**470**	**366**	2929	1334

Table 3

Results for the medium-size circuits with $L = 20$.

$L = 20$		C-4		C-5		C-6		C-7	
		asp	cep	asp	cep	asp	cep	asp	cep
Initial	subcircuits	32	27	30	33	29	36	47	38
solution	cuts	149	121	258	252	292	370	734	623
algorithm	seconds	13	4	32	7	35	17	763	46
TS-CPP	subcircuits	32	27	29	30	27	33	46	38
$\gamma = 1$	cuts	118	107	166	157	210	244	664	504
	seconds	29	26	91	89	116	136	322	291
TS-CPP	subcircuits	32	20	29	30	27	33	44	38
$\gamma = 2$	cuts	118	83	163	150	202	238	654	495
	seconds	28	91	89	87	119	131	538	294
TS-CPP	subcircuits	10	23	29	30	27	33	44	38
$\gamma = 3$	cuts	109	96	163	147	200	234	652	479
	seconds	106	36	88	92	117	197	421	333
TS-CPP	subcircuits	10	**8**	13	**9**	27	33	44	38
$\gamma = 4$	cuts	110	**99**	126	**121**	198	233	648	427
	seconds	91	**98**	319	**305**	199	189	526	478
TS-CPP	subcircuits	11	9	17	9	27	33	40	38
$\gamma = 5$	cuts	120	111	135	128	195	233	536	412
	seconds	98	107	324	303	246	151	1401	624
TS-CPP	subcircuits	**10**	9	10	19	26	32	41	35
$\gamma = 6$	cuts	**103**	103	115	138	191	221	533	368
	seconds	**78**	72	298	165	139	185	728	1191
TS-CPP	subcircuits	10	9	**9**	11	25	32	36	31
$\gamma = 7$	cuts	126	109	**131**	133	173	220	508	363
	seconds	78	103	**319**	291	284	194	894	636
TS-CPP	subcircuits	10	9	14	11	24	31	33	29
$\gamma = 8$	cuts	121	102	131	138	154	204	479	354
	seconds	63	88	228	305	296	371	1229	1149
TS-CPP	subcircuits	13	11	11	16	**22**	29	34	27
$\gamma = 9$	cuts	129	131	131	146	**137**	192	499	365
	seconds	65	84	309	317	**269**	347	1021	757
TS-CPP	subcircuits	13	10	12	11	25	30	36	27
$\gamma = 10$	cuts	101	115	131	143	181	198	509	355
	seconds	75	73	303	229	146	229	1024	643
TS-CPP	subcircuits	11	10	15	12	24	28	33	**25**
$\gamma = 11$	cuts	122	101	153	143	168	173	480	**370**
	seconds	82	47	327	214	261	410	963	**774**
TS-CPP	subcircuits	10	12	15	11	24	**27**	**31**	26
$\gamma = 12$	cuts	120	118	144	142	175	**180**	**500**	395
	seconds	62	105	345	278	194	**300**	**1257**	698

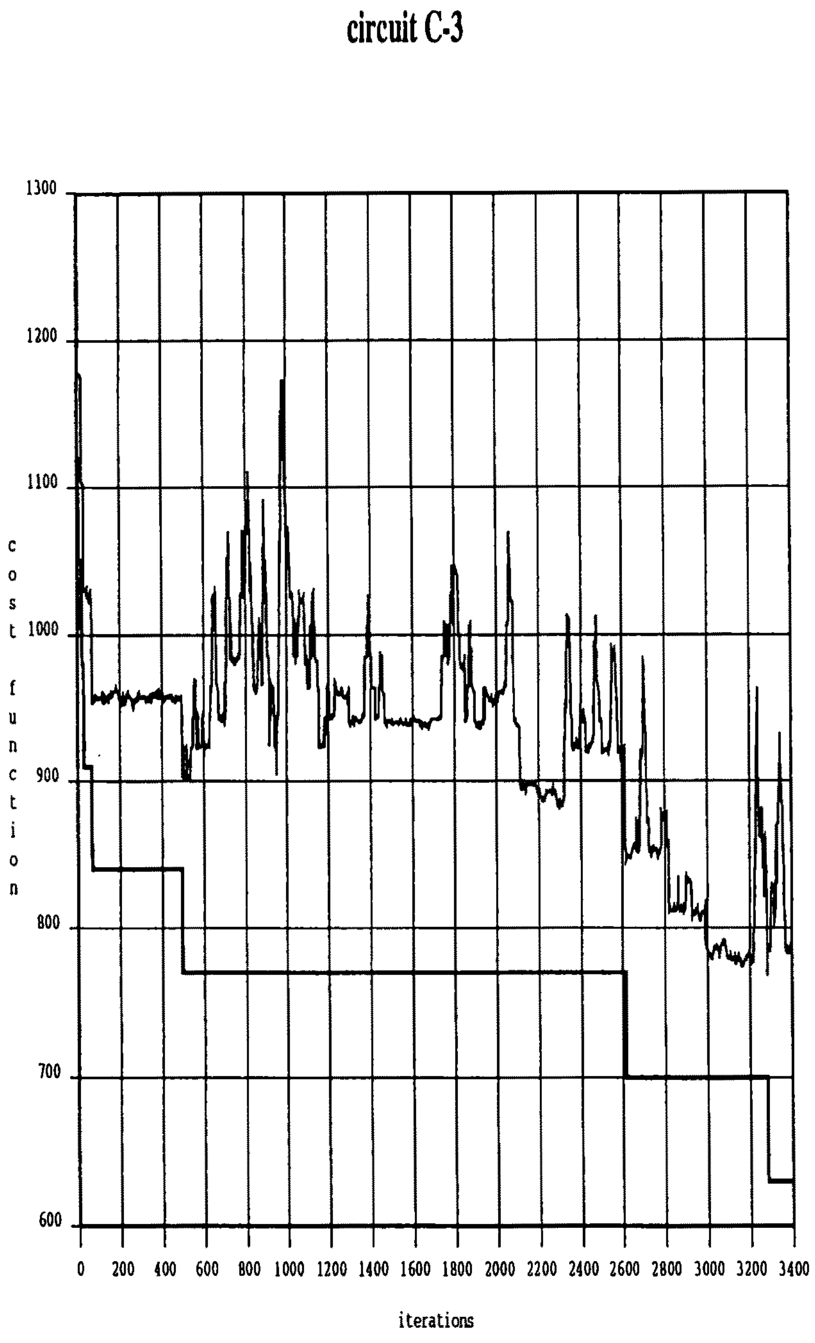

Figure 12. Cost function: circuit C-3, $L = 19$, $\gamma = 3$, initial solution by algorithm **asp**.

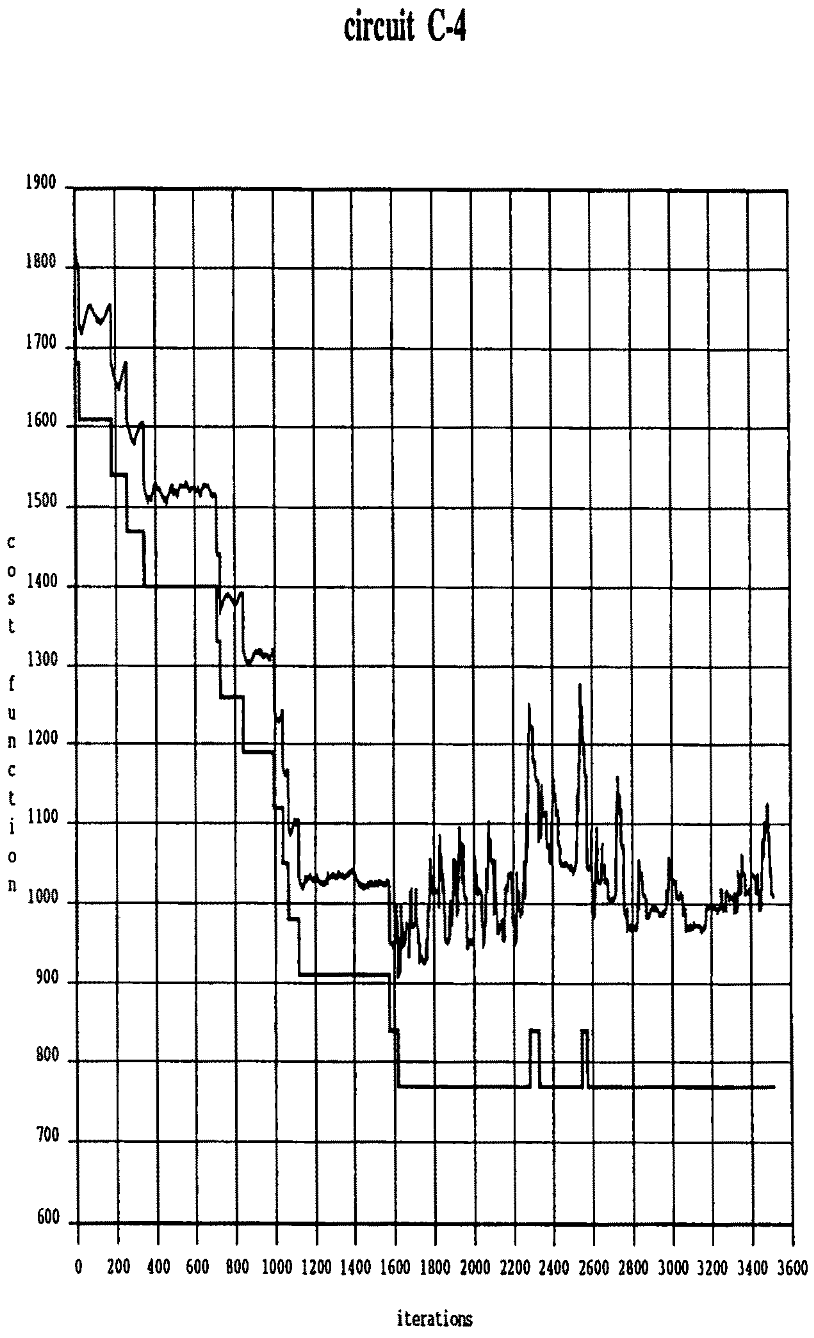

Figure 13. Cost function: circuit C-4, $L = 15$, $\gamma = 3$, initial solution by algorithm **cep.**

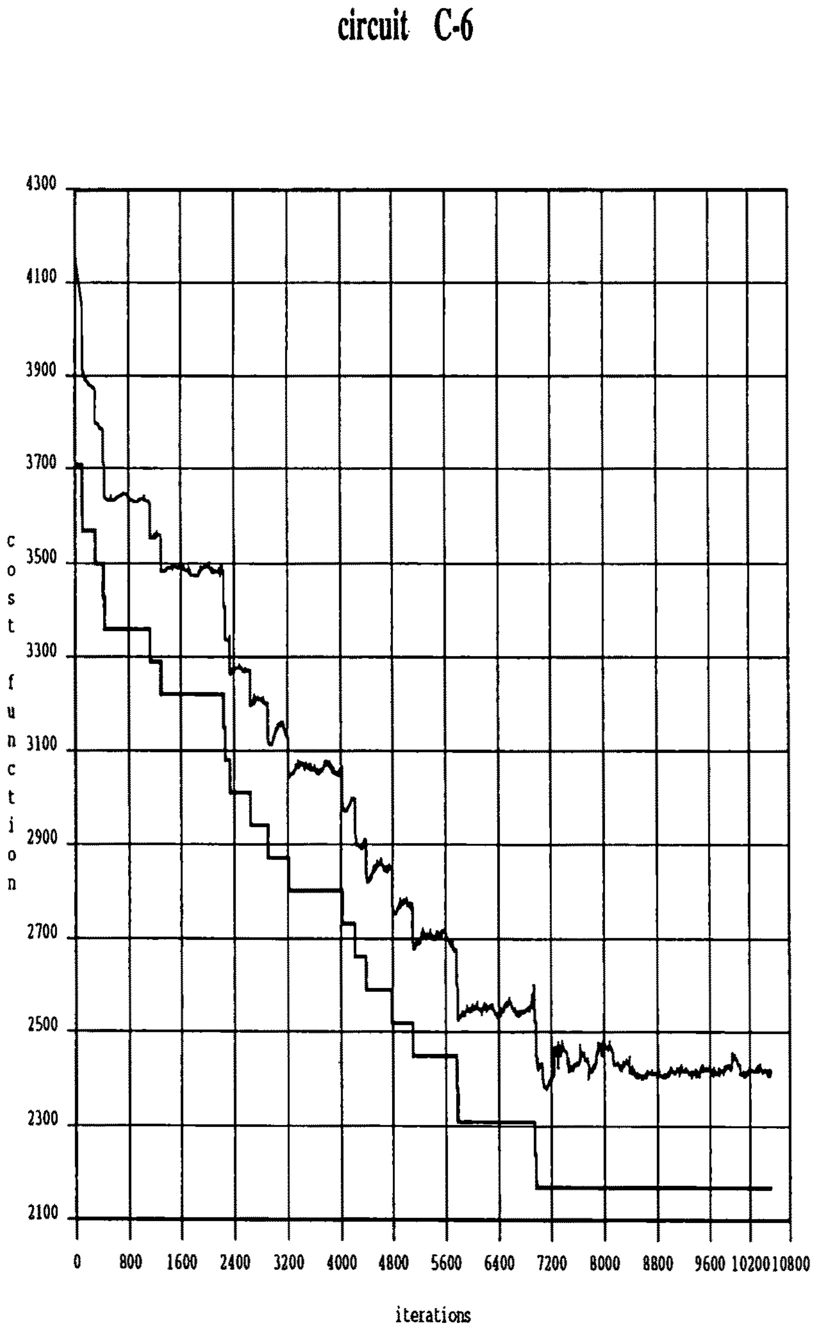

Figure 14. Cost function: circuit C-6, $L = 15$, $\gamma = 12$, initial solution by algorithm **cep**.

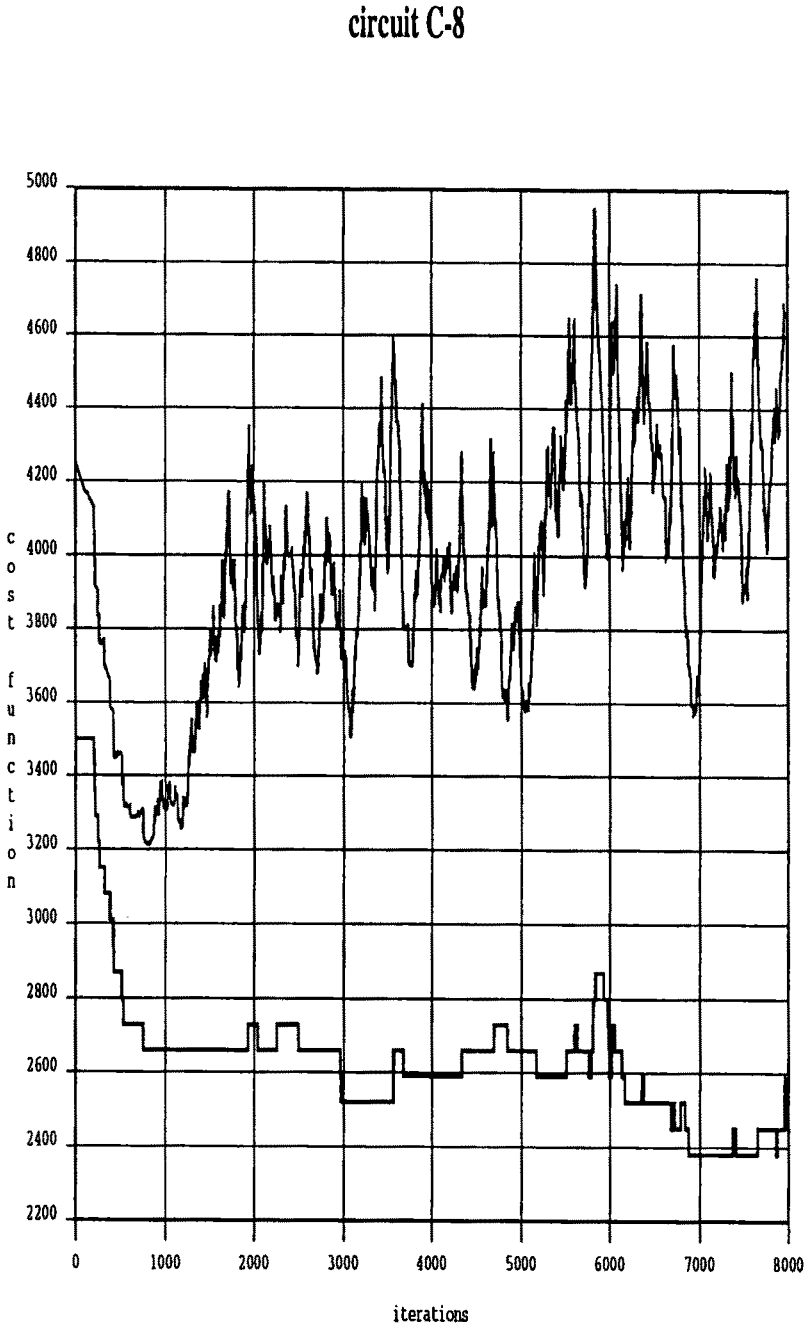

Figure 15. Cost function: circuit C-8, $L = 16$, $\gamma = 8$, initial solution by algorithm **cep.**

Table 4

Final results for the small- and medium-size circuits.

			C-1	C-2	C-3	C-4	C-5	C-6	C-7
$L = 15$	**mrl**	subcircuits	14	16	20	24	37	54	76
		cuts	148	171	215	282	472	578	1031
		cut ratio (%)	93	85	56	52	54	46	62
	cep	subcircuits	11	13	13	12	24	42	55
		cuts	122	138	130	137	299	445	762
		cut ratio (%)	76	68	34	25	34	35	46
	TS-CPP	subcircuits	7	8	11	10	14	26	38
		cuts	61	70	102	106	158	208	480
		cut ratio (%)	38	39	27	19	18	16	29
		cut reduction (%)	50	49	22	23	47	53	37
		subcircuit reduction (%)	36	38	15	17	42	38	31
$L = 17$	**mrl**	subcircuits	12	13	18	19	32	44	65
		cuts	136	168	219	260	445	528	978
		cut ratio (%)	85	83	57	48	51	42	59
	cep	subcircuits	9	11	12	11	19	36	47
		cuts	109	141	126	137	278	432	726
		cut ratio (%)	68	70	33	25	32	34	43
	TS-CPP	subcircuits	6	7	9	10	11	22	31
		cuts	54	65	79	116	138	207	432
		cut ratio (%)	34	32	21	21	16	16	26
		cut reduction (%)	50	54	37	15	50	52	40
		subcircuit reduction (%)	33	36	25	9	42	39	34
$L = 20$	**mrl**	subcircuits	9	13	15	17	24	37	53
		cuts	122	155	205	252	422	512	904
		cut ratio (%)	76	77	54	46	48	40	54
	cep	subcircuits	7	9	10	9	15	28	35
		cuts	98	124	128	121	252	370	623
		cut ratio (%)	61	61	33	22	29	29	37
	TS-CPP	subcircuits	5	5	8	8	9	19	22
		cuts	55	55	86	99	121	177	370
		cut rato (%)	34	27	22	18	14	14	22
		cut reduction (%)	44	56	33	18	52	52	41
		subcircuit reduction (%)	29	44	20	11	40	32	37

procedure **pack_together** coupled with the tabu search heuristic and procedure **make_feasible**, as described in figure 11. These results reflect the effectiveness of procedure **pack_together** as a post-optimization component of algorithm **TS-CPP**.

In table 4, we give the results obtained for the small- and medium-size circuits for $L = 15$, 17, and 20 by algorithms **mrl** (Roberts and Lala [32]), **cep** (Davis-

Table 5

Final results for the large-size circuits.

	$L = 15$	C-8	C-9		$L = 16$	C-8	C-9
cep	subcircuits	52	118	**cep**	subcircuits	50	104
	cuts	742	1527		cuts	753	1427
	cut ratio (%)	31	43		cut ratio (%)	31	41
TS-CPP	subcircuits	41	60	**TS-CPP**	subcircuits	38	52
	cuts	561	657		cuts	552	558
	cut ratio (%)	23	19		cut ratio (%)	23	16
	cut. red. (%)	24	57		cut. red. (%)	27	61
	subcirc. red. (%)	21	49		subcirc. red. (%)	24	50

	$L = 17$	C-8	C-9		$L = 18$	C-8	C-9
cep	subcircuits	46	97	**cep**	subcircuits	34	87
	cuts	743	1407		cuts	567	1334
	cut ratio (%)	31	40		cut ratio (%)	23	38
TS-CPP	subcircuits	36	49	**TS-CPP**	subcircuits	31	43
	cuts	554	540		cuts	491	517
	cut ratio (%)	23	15		cut ratio (%)	20	15
	cut red. (%)	25	62		cut red. (%)	13	61
	subcirc. red. (%)	22	49		subcirc. red. (%)	9	51

	$L = 19$	C-8	C-9		$L = 20$	C-8	C-9
cep	subcircuits	31	83	**cep**	subcircuits	22	77
	cuts	541	1328		cuts	406	1297
	cut ratio (%)	22	38		cut ratio (%)	17	37
TS-CPP	subcircuits	31	42	**TS-CPP**	subcircuits	23	38
	cuts	540	556		cuts	403	506
	cut ratio (%)	22	16		cut ratio (%)	17	14
	cut red. (%)	0	58		cut red. (%)	1	61
	subcirc. red. (%)	0	49		subcirc. red. (%)	0	51

Moradkhan [12], and Davis-Moradkhan and Roucairol [14]) and **TS-CPP** (using algorithm **cep** for the generation of the initial solution). For each circuit and each algorithm, we present the number of subcircuits, the number of cuts, and the ratio between the number of cuts and the total number of logical gates and outputs in the circuit. For algorithm **TS-CPP**, we also give the percentual reduction in the number of subcircuits and cuts with respect to the solution obtained by algorithm **cep**. The same results are reported in table 5 for the large-size circuits for L ranging from 15 to 20. The results of algorithm **mrl** for these circuits were not available in the literature.

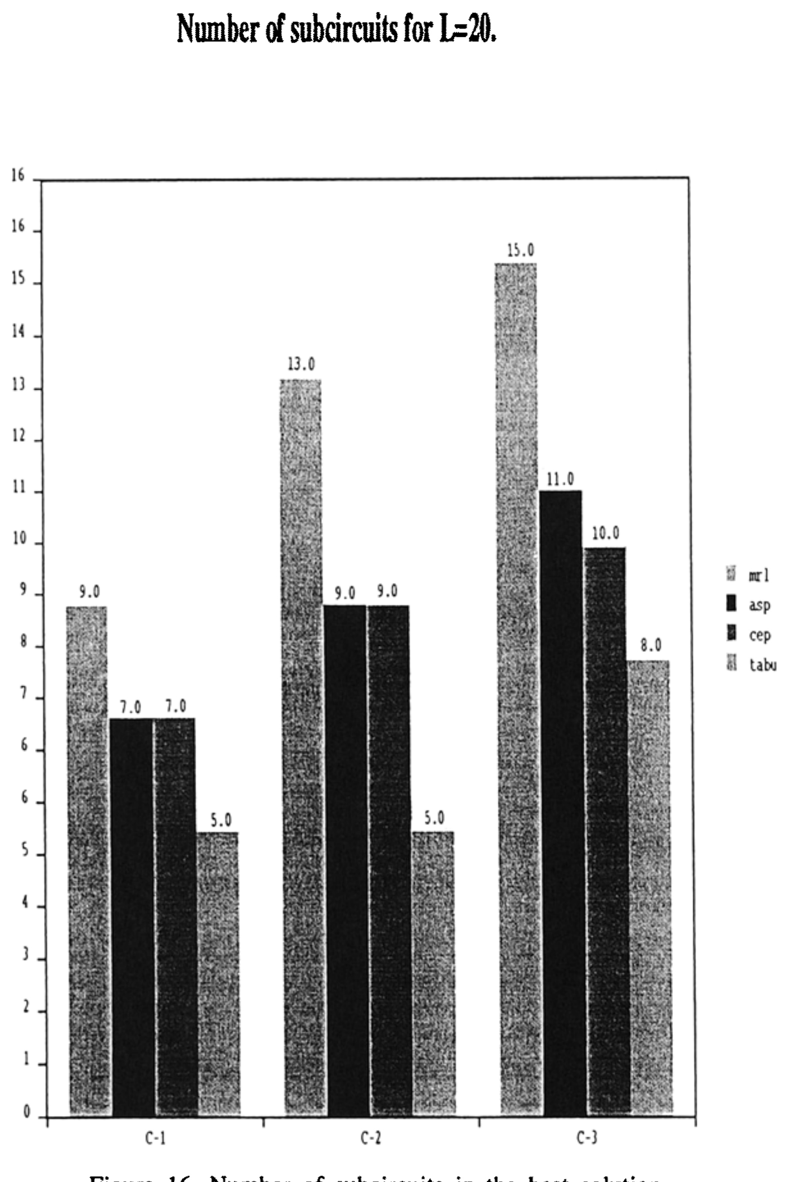

Figure 16. Number of subcircuits in the best solution found by each algorithm for the small-size circuits ($L = 20$).

To further illustrate the typical relative efficiency of the four algorithms investigated in this work (**mrl**, **asp**, **cep**, and **TS-CPP**), figures 16 and 17 are bar graphs indicating, respectively, the number of subcircuits and the number of cuts in the best feasible solutions found for the small-size circuits by the four algorithms for $L = 20$.

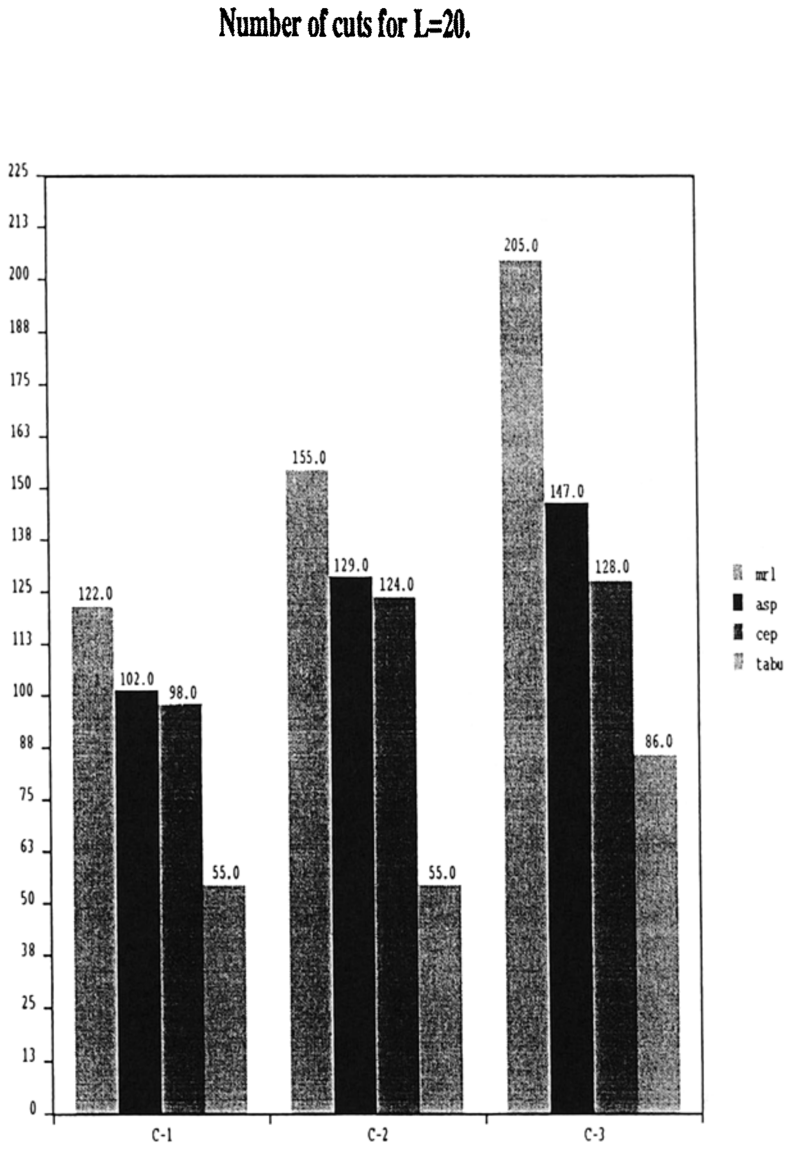

Figure 17. Number of cuts in the best solution found by each algorithm for the small-size circuits ($L = 20$).

6. Conclusions

In this work, we have developed a tabu search algorithm for the circuit partitioning problem in the framework of the parallel pseudo-exhaustive logical test of integrated combinational circuits. The main features of our algorithm are: (i) the use of reduced neighborhoods defined by moves involving only a subset of boundary nodes, whose size vanishes with the increase in the number of iterations; (ii) complex moves which entail several resulting moves, although the variations in the cost function are easily computable; (iii) a bi-criteria cost function combining the number of subcircuits and the number of cuts, which simultaneously adds a diversification strategy to the search; and (iv) the use of a bin-packing heuristic as a post-optimization step.

The numerical results reported in the last section for a set of ISCAS benchmark circuits point out the adequacy of the proposed approach for the solution of the circuit partitioning problem. The search mechanism has systematically guided the algorithm to improve the initial solutions and to escape from local optima.

The solutions obtained by algorithm **TS-CPP** have been compared with those obtained by algorithm **cep**, the best one available so far in the literature. The average reduction in the number of subcircuits was approximately 30% with respect to the latter, while the average reduction in the number of cuts ranged from 45% for the small-size circuits to 37% for the large-size circuits. Especially remarkable are the results observed with $L = 20$ for the largest benchmark circuit C-9, with 3405 gates and 7552 links, when the solution obtained by algorithm **TS-CPP** improved (i.e. reduced) by more than 50% the number of subcircuits and by more than 60% the number of cuts.

The larger computational times, with respect to those observed for other constructive heuristics in the literature, are largely compensated by the improvements in the objective function, in terms of the number of subcircuits and cuts. Moreover, these computational times should not be even considered as large, because they represent a very small fraction of the overall design and fabrication costs on an industrial scale. The critical issue is the total duration of the test, which is kept within reasonable bounds through the testability condition derived from the partition parameter.

References

[1] A.A. Andreatta, A graph partitioning heuristic for the parallel pseudo-exhaustive logical test of VLSI combinational circuits, M.Sc. Dissertation, Department of Electrical Engineering, Catholic University of Rio de Janeiro (1994), in Portuguese.

[2] E.C. Archambeau and E.J. McCluskey, Fault coverage of pseudo-exhaustive testing, *Digest of Papers of the 14th Int. Conf. on Fault-Tolerant Computing* (IEEE, 1984) pp. 141–145.

[3] Z. Barzilai, D. Coppersmith and A.L. Rosenberg, Exhaustive generation of bit patterns with applications to VLSI self-testing, IEEE Trans. Computers C-32(1983)190–193.

[4] F. Berglez and H. Fujiwara, A neutral netlist of 10 combinational benchmark circuits and a target translator in FORTRAN, Paper presented at the special session on *ATPG and Fault Simulation, Int. Symp. on Circuits and Systems*, Kyoto (IEEE, 1985).

[5] Z. Barzilai, J. Savir, G. Mankowsky and M.G. Smith, The weighted syndrome sums approach to VLSI testing, IEEE Trans. Computers C-30(1981)996–1001.

[6] S.N. Bhatt, F.R.K. Chung and A.L. Rosenberg, Partitioning circuits for improved testability, *Proc. 4th MIR Conf. on Advanced Research in VLSI* (MIT, Cambridge, 1986) pp. 91–106.

[7] S. Bozorgui-Nesbat and E.J. McCluskey, Structured design for testability to eliminate test pattern generation, *Digest of Papers of the 10th Int. Symp. on Fault-Tolerant Computing* (IEEE, 1980) pp. 158–163.

[8] M.A. Breuer and A.D. Friedman, *Diagnosis and Reliable Design of Digital Systems* (Computer Science Press, Woodland Hills, 1976).

[9] E. Calia and A. Lioy, Test generation in a distributed environment, Research Report, Instituto Politecnico di Torino (1991).

[10] C.L. Chen, Linear dependencies in linear feedback shift registers, IEEE Trans. Computers C-35(1986)1086–1088.

[11] M. Davis-Moradkhan, The problem of partitioning the nodes of a graph and its applications in VLSI technology: An overview, Rapport MASI 90.08, Laboratoire MASI, Université Paris VI (1990).

[12] M. Davis-Moradkhan, Partitioning problems in VLSI technology, Doctorate Thesis, Université Paris VI (1993), in French.

[13] M. Davis-Moradkhan and C. Roucairol, Comparison of two heuristics for partitioning combinational circuits for parallel pseudo-exhaustive testing, Rapport MASI 92.25, Laboratoire MASI, Université Paris VI (1992).

[14] M. Davis-Moradkhan and C. Roucairol, Graph partitioning applied to the problem of logic testing of VLSI combinational circuits, Rapport MASI 92.41, Laboratoire MASI, Université Paris VI (1992).

[15] M.R. Garey and D.S. Johnson, Strong NP-completeness results: Motivation, examples and implications, J. ACM 25(1978)499–508.

[16] F. Glover, Future paths for integer programming and links to artificial intelligence, Comp. Oper. Res. 13(1986)533–549.

[17] F. Glover, Tabu search – Part I, ORSA J. Comput. 1(1989)190–206.

[18] F. Glover, Tabu search – Part II, ORSA J. Comput. 2(1990)4–32.

[19] F. Glover, Tabu search: A tutorial, Interfaces 20(1990)74–94.

[20] F. Glover and H.J. Greenberg, New approaches for heuristic search: A bilateral linkage with artificial intelligence, Euro. J. Oper. Res. 39(1989)119–130.

[21] F. Glover and M. Laguna, Tabu search, in: *Modern Heuristic Techniques for Combinatorial Problems*, ed. C.R. Reeves (Blackwell, London, 1993) pp. 70–150.

[22] A. Hertz and D. de Werra, Using tabu search for graph coloring, Computing 29(1987)345–351.

[23] A. Hertz and D. de Werra, The tabu search metaheuristic: How we used it, Ann. Math. Art. Int. 1(1990)111–121.

[24] E.J. McCluskey, Built-in self-test techniques, IEEE Design Test Comp. 2(1985)21–28.

[25] E.J. McCluskey, Built-in self-test structures, IEEE Design Test Comp. 2(1985)29–36.

[26] E.J. McCluskey and S. Bozorgui-Nesbat, Design for autonomous test, IEEE Trans. Computers C-30(1981)866–874.

[27] S.D. Millman and E.J. McCluskey, Detecting bridging faults with stuck-at test sets, *Proc. Int. Test Conf.* (IEEE, 1988) pp. 773–783.

[28] Y. Min and Z. Li, Pseudo-exhaustive testing strategy for large combinational circuits, Comp. Syst. Sci. Eng. 1(1986)213–220.

[29] E.I. Muehldorf and A.D. Savkar, LSI logic testing – An overview, IEEE Trans. Computers C-30(1981)1–17.

[30] O. Patashnik, Optimal circuit segmentation for pseudo-exhaustive testing, Doctorate Thesis, Department of Computer Science, Stanford University (1990).

[31] C.V. Ramamoorthy and R.C. Cheung, Design of fault tolerant computing systems, in: *Applied Computation Theory: Analysis, Design, Modeling*, ed. R.T. Yeh (Prentice–Hall, Englewood Cliffs, 1976) pp. 286–296.

[32] M.W. Roberts and P.K. Lala, An algorithm for the partitioning of logic circuits, IEEE Proc. G 131(1984)113–118.

[33] S.C. Seth, B.B. Bhattacharaya and V.D. Agrawal, An exact analysis for efficient computation of random-pattern testability in combinational circuits, *Digest of Papers of the 16th Int. Symp. on Fault-Tolerant Computing Systems* (IEEE, 1986) pp. 318–323.

[34] I. Shperling and E.J. McCluskey, Circuit segmentation for pseudo-exhaustive testing via simulated annealing, *Proc. Int. Test Conf.* (IEEE, 1987) pp. 58–66.

[35] J.G. Udell and E.J. McCluskey, Efficient circuit segmentation for pseudo-exhaustive test, *Proc. IEEE Int. Conf. on Computer-Aided Design* (IEEE, 1987) pp. 148–151.

[36] L.T. Wang and E.J. McCluskey, Condensed linear feedback shift register (LFSR) testing – A pseudo-exhaustive test technique, IEEE Trans. Computers C-35(1986)367–369.