ELSEVIER

Discrete Applied Mathematics 62 (1995) 131–165

DISCRETE APPLIED MATHEMATICS

Graph partitioning applied to the logic testing of combinational circuits

Madlaine Davis-Moradkhan, Catherine Roucairol

PRiSM, University of Versailles, 45 Avenue des Etats Unis, 78000, Versailles, France

Received 12 May 1992; revised 14 April 1994

Abstract

Logical testing of integrated circuits is an indispensable part of their fabrication. Exhaustive testing of a VLSI (very large scale integration) circuit, which can detect all its faults, is impractical due to the complexity of such circuits and the number of input patterns that have to be applied. A circuit with n inputs would require 2^n different input patterns to be exhaustively tested. To overcome this problem when n is larger than about 20, the circuit can be partitioned into subscircuits each with fewer inputs, which can then be tested exhaustively. This procedure, known as the pseudo-exhaustive testing has a better fault coverage than classical methods. In this paper we present a graph partitioning model for the problem of partitioning a combinational VLSI circuit represented by a directed acyclic graph. However, any direct approach to finding an optimal solution to the graph partitioning problem will require an inordinate amount of computation. Therefore, we propose a polynomial complexity algorithm, composed of two phases, which has been tested with benchmark ISCAS circuits.

1. Introduction

The application of integrated circuits (IC's) and the process of component miniaturization have fueled the trend to build smaller computers and data processing equipment with higher component densities. The breakthrough offered by large scale and very large scale integration (LSI/VLSI) permit the placement of several thousands of logic gates on a pluggable unit.

The importance of testing (the ability to tell a good machine, one that operates as it was designed, versus a faulty machine), has grown as integration densities have increased. However, the difficulty of testing of LSI/VLSI devices has increased by an order of magnitude.

To cope with the testing problem, new methods have been introduced to permit testing logical responses of thousands of gates. Moreover, the importance of "design for testability" has been emphasized and new design techniques proposed which would facilitate testing procedures. Muehldorf and Savkar [26] give a comprehensive

SSDI 0166-218X(94)00150-2

overview of the state of the art up to and including 1977; Williams [40], [41], and Tsui [35] discuss different design techniques for testability. For detailed description of these techniques see [5].

Next we give some definitions and describe the problem of testing combinational circuits.

1.1. Definitions

A combinational integrated circuit (CIC) is made up of the following elements:

Primary inputs pass binary data from the outside world to the CIC.

Logic gates perform various logical operations. The basic logic gates perform operations AND, OR, NAND, and NOR. There is also the inverter which performs NOT (see Fig. 1). Complex logical functions can be designed using these simple gates. For example, for the two functions $G = \bar{A}B + \bar{E}$ and $F = R + \overline{[\bar{A}B(C + \bar{D})]}$, circuit 1 in Fig. 2 can be designed. This circuit has 6 inputs, and 9 gates.

Primary outputs pass on the binary results of logical operations performed by the CIC to the outside world. The two outputs of circuit 1 are labelled F and G.

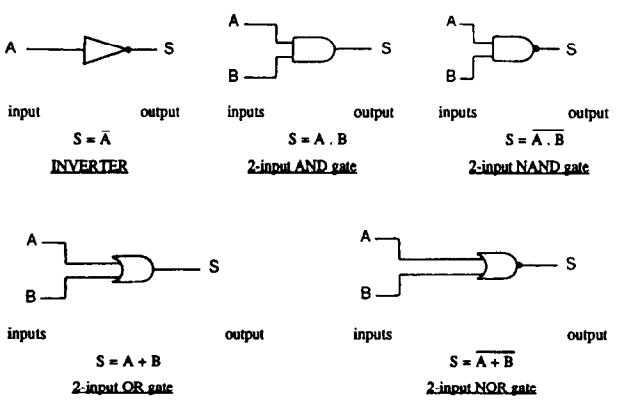

Fig. 1. Simple logic gates (source: MASI89 [18]).

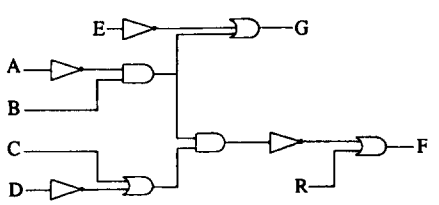

Fig. 2. Circuit 1 (source: Wiatrowski and House [39]).

Logical signals pass one bit of information (0 or 1) between various elements mentioned above.

Logical paths go from the inputs through the gates to the outputs. Every path consists of a number of signals which join an input to a gate, a gate to another gate, or a gate to an output. Since a CIC does not have any memory elements, there are no loops or closed paths in such circuits. In some real CIC's, an input might be connected directly to an output, and there may be signals in series.

Fan-in is the number of predecessors of an element. For example in Fig. 3, gate P has a fan-in equal to two. Inputs have zero fan-in. Gates may have large fan-ins, and outputs have only one predecessor. Generally the term fan-in is used when there are more than one predecessors.

Fan-out is the number of successors of an element. For example in Fig. 3, the fan-out of gate Q is equal to two. Outputs have no successors. Inputs and gates may have large fan-outs. This term is usually employed when an element has more than one successor.

Reconvergent fan-out is the term used to indicate different paths between some pairs of gates. For example in Fig. 3, the fan-out at gate A has reconverged at gates B, C, and D, forming, respectively, 2, 2, and 4 paths from A to B, to C, and to D.

For more details see any textbook on VLSI engineering: for example Wiatrowski and House [39] provide a highly readable handbook.

1.2. Testing a CIC

Testing a CIC means introducing various 0–1 patterns (test vectors) through its inputs and reading the results at its outputs, to verify that logic gates perform correctly and that no physical faults (bridging faults, stuck-at-faults, etc.) exist. For example, test vectors for circuit 1 will be of the form ($ABCDER$), where each input can be either 0 or 1, such as (1 0 0 1 1 1), (1 1 1 1 0 1), etc. In order to insure that a CIC is fault free, it should be tested exhaustively by introducing possible 0–1 patterns. Given a CIC with n input pins, it would take 2^n test patterns to exhaustively test the circuit.

With a tester which could provide a test pattern and read the response from the circuit's output pins in say, 1 μs, a circuit with $n = 100$ would take over $4 * 10^{15}$ years to test. Such a test cycle time is clearly impractical. To reduce the test set a process known as "fault simulation" is performed which determines the faults in a circuit that can be detected by a series of given test vectors. However, this method has been criticized mainly for the following reasons (see e.g. [22, 30]).

(1) A fault model with assumptions about the types of faults that might arise is required. Since the classical simple models with the assumption of single stuck-at-fault are no longer valid for the VLSI circuits, more complex models are required, but they substantially increase the difficulty of test pattern generation.

(2) The automatic test vector generation is costly and typically does not provide sufficiently high fault coverage.

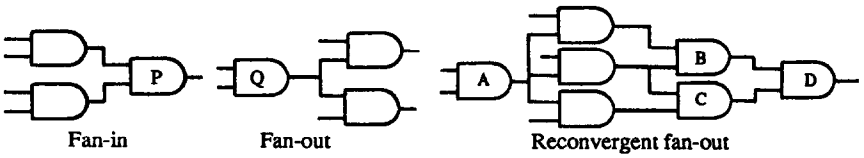

Fig. 3. Fan-ins and fan-outs.

(3) An expensive tester is required. When test generation produces many patterns, the tester is tied up for a long time so that many testers must be used.

(4) As the circuit grows in size, simulation time increases exponentially.

Alternative methods such as built-in-self-testing (BIST), where expensive external testers are no longer required (see e.g. [42]), or locally exhaustive testing (see e.g. [15]) have been proposed in order to overcome some of these problems.

Moreover, decomposing large CIC's into testable overlapping subcircuits, each with fewer inputs has been suggested as an alternative approach to exhaustive testing. This procedure, known as pseudo-exhaustive testing, was introduced around 1980, and the first to discuss it fully were Bozorgui-Nesbat and McCluskey [7]. Then McCluskey and Bozorgui-Nesbat [22] combined the notion of BIST with pseudo-exhaustive testing by suggesting a method for reconfiguring the existing registors of a CIC into modified linear feedback shift registors (LFSR's) which, in test mode, apply the exhaustive test patterns or convert the responses into signatures, and in normal mode, function as before.

Other works on pseudo-exhaustive testing have been presented in the past ten years which in general follow one of two approaches. In the first approach, algorithms to decompose a CIC into subcircuits which may overlap are proposed. Works by Bhatt et al. [6], Min and Li [25], Shperling and McCluskey [32], and Udell and McCluskey [36] fall into this category.

In the second approach, having already been given a segmented CIC, implementation problems such as accessing the subcircuits' inputs and outputs or inserting additional hardware into the CIC are considered. Works by Akers [1], Barzilai et al. [3], Chen [9], Tang and Chen [34], Vasanthavada and Marinos [37], and Wang and McCluskey [38] fall into this category. Patashnik [27] gives a summary of the research done in these two approaches up to and including 1988, where an unrestrictive fault model is used, one assuming that the circuit, if faulty, is still combinational. He also shows that decomposing a CIC into testable subscircuits is an NP-complete problem. More recently, Su and Kime [33] have proposed a computer-aided design tool to decompose semiregular CIC's and performe pseudo-exhaustive BIST.

1.3. Advantages and drawbacks of pseudo-exhaustive testing

The advantages of pseudo-exhaustive testing compared to other methods practiced, and its fault coverage have been discussed by Archambeau and McCluskey [2], and Millman and McCluskey [23]. We briefly mention some of these advantages:

(1) It does not rely on a fault model and is thus not limited to any specific class of faults such as stuck faults.

(2) It guarantees 100% coverage for single stuck faults without any fault simulation being required.

(3) The computation time to determine the test set depends only on the number of inputs and outputs in the circuit, and it is much smaller than for test pattern generation.

Although pseudo-exhaustive testing has a higher performance than other testing procedures, it has, nevertheless, two inconveniences.

(1) It requires longer test cycles because subcircuits overlap and are tested serially (one at a time), and because more test patterns (than in fault-simulation, for example) have to be applied to the circuit. It is for this reason that, in general, the cost function to be minimized is represented in terms of the total test length, which is the sum of test lengths of individual subcircuits. For example, Patashnik [27] defines the cost of pseudo-exhaustive testing as the total number of different test vectors needed, which is equal to $\Sigma_{i=1}^{k} 2^{n_i}$, where n_i is the number of inputs of the ith subcircuit, and k is the number of subcircuits.

(2) It does not provide 100% fault coverage, as does exhaustive testing. Faults that escape detection are faults in different subcircuits that mask each other, and faults that add memory.

The first inconvenience, however, can be removed if the CIC is partitioned into non-overlapping subcircuits, which may then be tested in parallel. In that case test length will be equal to the test time of the subcircuit with the largest number of inputs.

1.4. Parallel testing

Parallel testing and parallel generation of test vectors have recently attracted more attention in literature. Inoue et al. [17] and Matsuda et al. [19] have considered parallel testing technology for VLSI memories which reduces the test time drastically. Patil and Banerjee [28, 29] propose heuristics to partition faults for parallel test-vector generation in fault simulation environment, with the objective of minimizing the overall run time and test length of CIC's.

In pseudo-exhaustive testing, Barzilai et al. [4] have proposed a technique called "syndrome testing" by which they perform a parallel testing of all functions in a subcircuit (which they call a "macro") of a CIC. Moreover, those subcircuits which have disjoint sets of inputs can be syndrome tested in parallel. They do not propose any algorithm to partition the CIC, and suppose that it is already partitioned. McCluskey [20,21] has suggested a method called "verification testing" in which, without partitioning the CIC and only by an appropriate choice of input patterns, it is possible to apply a reduced set of input combinations to each output concurrently rather than serially. His method, however, depends on the structure of the CIC and cannot be applied universally.

Roberts and Lala [30] propose an algorithm to partition a CIC into subscircuits each having more or less the same number of inputs, which can then be tested in parallel.

Their algorithm, however, has certain drawbacks, some of which have been mentioned by the authors themselves, others discussed briefly in [11], where an algorithmic description of their method is also given. Davis-Moradkhan and Roucairol [13] propose a mathematical model to partition a CIC into subscircuits each having at most L inputs, where L is a predefined parameter. They also suggest two algorithms "ESP" (evaluation, segmentation, partition), and "MRL" (modified algorithm of Roberts and Lala), which they compare using ISCAS85 circuits [8].

In this paper we present a polynomial complexity heuristic "CEP" (combinational circuit evaluation and partition), which is compared with "ESP" and "MRL" and is shown to be more efficient. "CEP" is based on an earlier algorithm presented in [12].

2. The problem of partitioning a CIC

2.1. Assumptions

As mentioned earlier, to exhaustively test a combinational circuit 2^n patterns have to be applied to its n input pins. For n larger than about 20 exhaustive testing is impractical. Therefore CIC's with $n > 20$ have to be partitioned into subcircuits each with fewer inputs, which can then be tested with integrated linear feedback shift registers (LFSR's). Clearly if *subcircuit i has n_i inputs*, the LFSR used to test it exhaustively must have at least n_i flip-flops (or bits).

We assume that the *maximum time for the test is set at T*, during which time all subcircuits will be tested in parallel. Let L be an integer such that all 2^L input patterns may be generated in such time that will permit the completion of pseudo-exhaustive testing in time T. Thus, *L is an upper bound on the number of inputs that a subcircuit may have*. We call L the *partition parameter*. The problem is then *to partition the CIC into k (k not fixed) subcircuits*, such that each subcircuit has at most L input pins. Furthermore, *each subcircuit must contain at least one gate*, otherwise there will be nothing to test.

It is intuitively clear that the larger L is, the fewer subcircuits would be created. However, L is bounded above by the testing time. With the available testers, the testing time required is such that, for practical purposes, L should be at most equal to 20. In our experimental results, we have run our partitioning algorithms with L ranging from 15 to 20 to find out any significant dependence of L on the circuit's shape or size. In all cases, the larger L is the better are results, whatever the shape or the size of the circuit.

Partitioning a circuit involves inserting additional logic into signal paths that are to be cut, so that the normal propagation of those signals can be modified. Consider the circuit in Fig. 4, in which the output of gate P is fed into gate Q. If for the partitioning

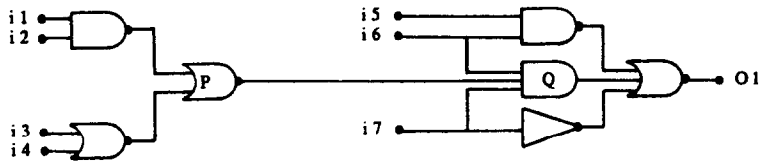

Fig. 4. Circuit 2 before partitioning (source: Roberts and Lala [30]).

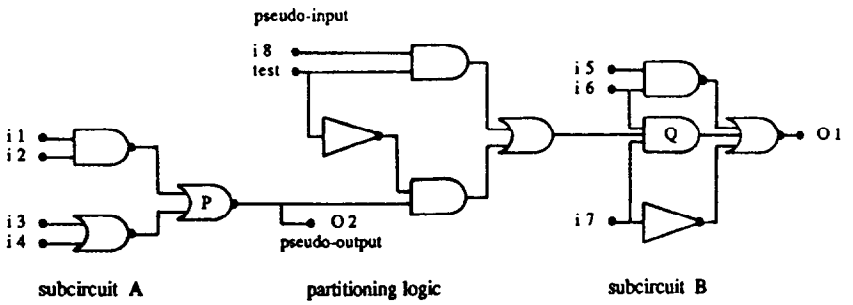

Fig. 5. Circuit 2 after partitioning (source: Roberts and Lala [30]).

it is required to cut the signal PQ, an extra logic along the link between gate P and gate Q is inserted as shown in Fig. 5.

While not in test mode the test signal is at logic zero and a logical connection exists between the output of P and the input to Q. When the test signal is asserted, subcircuit A is disconnected from subcircuit B.

Thus everytime a signal is cut a *pseudo-primary input* (or simply *pseudo-input*) and a *pseudo-primary output* (or simply *pseudo-output*) are created. For simplicity we will assume here that none of these additional input/outputs are regrouped into a single input/output, although in practice it is possible to do so.

Since cutting a signal produces one additional input, when considering potential subcircuits, we will have to account for not only the number of primary inputs they already have, but also for the number of additional inputs that will be created. Thus for example in Fig. 5, subcircuit A will have 4 primary inputs ($i1$ to $i4$), while subcircuit B will have 3 primary inputs ($i5$ to $i7$) plus one pseudo-input ($i8$). Notice that the additional input is part of the subcircuit whose incoming signal was cut. Therefore, if *subcircuit i has n_i inputs, and c_i is the number of pseudo-inputs (cuts) created for partitioning, the relation $n_i + c_i \leqslant L$ must hold.* Needless to say, the outgoing signals from a subcircuit which are cut should not be counted.

2.2. Graph representation of a CIC

Let $G = (X, A)$ be the graph corresponding to the combinational circuit C, called a circuit-graph, where X represents the set of components (inputs, gates, outputs) of

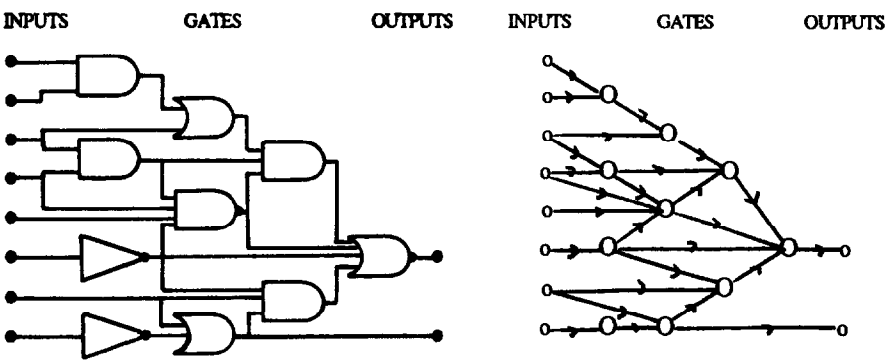

Fig. 6. Circuit 3 and its graph representation.

C and A represents the set of signals. The indegree and outdegree of vertex $v \in X$ are denoted by $d^-(v)$ and $d^+(v)$, respectively. Similarly the set of vertices not in X_j, with at least one successor in X_j is denoted by $\omega^-(X_j)$. The circuit graph G is a directed acyclic graph (DAG) with the following properties:

$$X = E \cup P \cup S,$$

where

$$E \neq \emptyset, \quad P \neq \emptyset, \quad S \neq \emptyset;$$

$$\forall e \in E \quad d^-(x) = 0, \quad d^+(x) \geqslant 1;$$

$$\forall p \in P \quad d^-(x) \geqslant 1, \quad d^+(x) \geqslant 1;$$

$$\forall s \in S \quad d^-(x) = 1, \quad d^+(x) = 0.$$

E, P, and S represent, respectively, the set of primary inputs, the set of logic gates, and the set of primary outputs. Thus vertices in E and S have zero indegree and zero outdegree, respectively, while vertices in P have positive indegree and outdegree. For simplicity, throughout this paper, we call vertices in E, *inputs*, vertices in P, *gates*, and vertices in S, *outputs*.

In some real circuits, the associated graph may consist of two or more disjoint subgraphs $G_1, \ldots, G_b, \ldots, G_B$, where $G_b = (X_b, A_b)$ is connected and $X_b \cap E \neq \emptyset$, $X_b \cap P \neq \emptyset$, $X_b \cap S \neq \emptyset$, $\forall b = 1, \ldots, B$.

Then the problem would be to partition those subgraphs which have more than L inputs. Therefore, without loss of generality, we will assume that G is a weakly connected graph. It should be noted that graphs representing realistic CIC's are non-planar (for definitions concerning graphs see [10]).

It should be added that, just as the associated graph of a realistic CIC may be composed of two or more disjoint subgraphs, after the partitioning of the CIC into subcircuits, the associated subgraphs may in turn consist of two or more disjoint subgraphs. This fact does not effect the testing of the CIC. Circuit 3 and its graph representation are shown in Fig. 6, where orientation of arcs is indicated. In the rest of

this paper, orientations will not be shown, as it is understood that arcs are always orientated from inputs towards gates and outputs (from left to right).

2.3. Graph partitioning model (GPM)

The problem of partitioning G into testable subcircuits is identical to the problem of partitioning X into subsets $X_1, \ldots, X_j, \ldots, X_k$ (k not fixed) with the objective of minimizing k such that:

(i) $X = \bigcup_{j=1}^{k} X_j$ and $X_j \cap X_i = \emptyset$, $\forall j \neq i, j, i = 1, \ldots, k$;

(ii) $X_j \cap P \neq \emptyset$, $\forall j = 1, \ldots, k$;

(iii) $n_j + c_j = |x_j \cap E| + |\omega^-(X_j)| \leqslant L$, $\forall j = 1, \ldots, k$;

(iv) the subgraph $G_j = (X_j, A_j)$, induced by X_j is connected, $\forall j = 1, \ldots, k$; (alternatively G_j has at most H non-empty connected components, where H is given and $X_h \cap P \neq \emptyset$, $\forall h = 1, \ldots, H$).

Proposition 2.1. *The graph partitioning model, GPM, is equivalent to the problem of partitioning a CIC into testable subcircuits.*

Proof. Constraint (i) simply insures that the subsets of vertices are disjoint and that their union is equal to the set X of vertices. In (ii) it is required that there be at least one gate in each subcircuit. Constraint (iii) is the *testability constraint*, in which $n_j = |X_j \cap E| \geqslant 0$ is the number of primary inputs, and $c_j = |\omega^-(X_j)| \geqslant 0$ is the number of pseudo-inputs created due to cuts. Notice that for several incoming arcs from the same vertex, only one pseudo-input is created. In (iv) it is mentioned that associated subgraphs may consist of disjoint subgraphs. This fact does not effect testing procedure, however this condition should be added to avoid subcircuits that have isolated inputs or isolated outputs. Condition (iv) can also be written as: $\forall e \in E_j$, there exists arc (e, v) for some $v \in P_j$, and $\forall s \in S_j$, there exists arc (v, s) for some $v \in P_j$. □

The necessity of constraint (iv) can be seen from the following example. In Fig. 7, if we take the set $X_j = \{e1, e4, p3, s1\}$, condition (ii) is verified since X_j contains a gate.

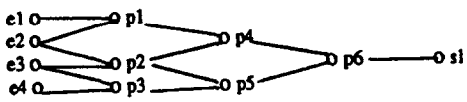

Fig. 7. Circuit 4.

However, X_j is composed of three subsets: $\{e1\}$, $\{e4, p3\}$, and $\{s1\}$. It is clear that $\{e1\}$ and $\{s1\}$ do not satisfy condition (iv); and X_j is not a meaningful subcircuit.

Note: It is important to note that the originality of this graph partitioning model lies in the supplementary constraint on testability of each subcircuit. While in classical partitioning problems, one aims at minimizing the number of cuts or maximizing the number of elements in each part subject to some constraints, here, in addition to these two objectives, we also aim at creating subcircuits that are testable.

The testability constraint is not a usual constraint because the process of creating subcircuits is gradual and each time an arc is cut, the original circuit is augmented by the insertion of an additional pseudo-input/output.

2.4. *Linear integer programming model (LIM)*

We have formulated the graph partitioning model (GPM) as a linear integer programming model with 0–1 variables, denoted LIM, which is presented in this section.

Let $G = (X, A)$ be a circuit graph that we wish to partition, and let J be an upper bound on the number of subcircuits, V_j, that will be created.

For every vertex, $v \in V = E \cup P$, and for every subset V_j we define

$$a_{uv} = \begin{cases} 1 & \text{if arc } (u, v) \in A, \\ 0 & \text{otherwise,} \end{cases}$$

$$y_j = \begin{cases} 1 & \text{if subcircuit } V_j \text{ is created,} \\ 0 & \text{otherwise,} \end{cases}$$

$$x_{vj} = \begin{cases} 1 & \text{if } v \in V_j, \\ 0 & \text{otherwise,} \end{cases}$$

$$z_{uvj} = \begin{cases} 1 & \text{if } x_{vj} = 1, \text{ and } x_{uj} = 0, \\ 0 & \text{otherwise.} \end{cases}$$

Therefore, each time an incoming arc, (u, v), to the subset V_j is cut, z_{uvj} will become equal to 1. Thus $\Sigma_{u, v \in V} a_{uv} z_{uvj}$ will be equal to the number of incoming arcs to subset V_j which are cut for partitioning.

The partitioning problem presented in Section 2.3 can be formulated as follows, where the objective is to minimize the number of subcircuits.

$$
\text{Lim}\ \begin{cases}
\text{Min} \sum_{j=1}^{J} y_j & \\
\text{such that:} & \\
\sum_{j=1}^{J} x_{vj} = 1 \quad \forall v \in V, & (4.1)\\
y_j + \sum_{e \in E} x_{ej} \leqslant \sum_{v \in V} x_{vj} \leqslant N_{y_j} \quad j = 1, \ldots, J, & (4.2)\\
\sum_{v \in E} x_{vj} + \sum_{u,v \in V}^{u \neq v} a_{uv} z_{uvj} \leqslant L, \quad j = 1, \ldots, J, & (4.3)\\
\sum_{v \in P} a_{ev} x_{vj} \geqslant x_{ej} \quad \forall e \in E, \text{ and } \forall j, & (4.4)\\
x_{vj} \geqslant z_{uvj} \quad \forall u,v \in V,\ u \neq v, \text{ and } \forall j, & (4.5)\\
1 - x_{uj} \geqslant z_{uvj} \quad \forall u,v \in V,\ u \neq v, \text{ and } \forall j, & (4.6)\\
x_{vj} - x_{uj} \leqslant z_{uvj} \quad \forall u,v \in V,\ u \neq v, \text{ and } \forall j, & (4.7)\\
y_j, x_{vj}, z_{uvj} \in \{0,1\} \quad \forall v \in V, \text{ and } \forall j. &
\end{cases}
$$

Proposition 2.2. *The graph partitioning model GPM is equivalent to the linear model LIM.*

Proof. (For brevity we have included a shorter version of the proof in this paper.) Constraint (4.1) is equivalent to condition (i) of Section 2.3, which states that every element in the circuit should be included, and that subcircuits should be disjoint.

In constraint (4.2), N represents a large positive number. When $y_j = 1$, and subcircuit V_j is created, constraint (4.2) becomes

$$1 + \sum_{e \in E} x_{ej} \leqslant \sum_{v \in V} x_{vj} \leqslant N.$$

Since the right-hand side inequality is now redundant, constraint (4.2) becomes

$$\sum_{v \in V} x_{vj} \geqslant 1 + \sum_{e \in E} x_{ej}.$$

This is equivalent to condition (ii) of Section 2.3, which states that there should be at least one gate in each subcircuit. And, when $y_j = 0$, i.e., subcircuit V_j is not created, constraint (4.2) becomes

$$0 + \sum_{e \in E} x_{ej} \leqslant \sum_{v \in V} x_{vj} \leqslant 0,$$

which implies $x_{vj} = 0$, $\forall v \in V$, as expected.

Constraint (4.3) is equivalent to condition (iii) of Section 2.3, that the total number of inputs (primary and pseudo) in any subcircuit should not exceed L. The first term in (4.3) is the number of primary inputs, the second term is the number of incoming arcs to V_j which are cut (pseudo-inputs). Notice that here, for simplicity, we assume that for every arc that is cut a pseudo-input is created.

Constraint (4.4) is equivalent to condition (iv) of Section 2.3, which states that there should not exist isolated inputs in a subcircuit, i.e., if input e is in V_j then there should be at least one element in P_j which is a successor of e.

The constraints of the form (4.5)–(4.7) insure that only the incoming arcs to a subcircuit which are cut, are counted and not the outgoing cut arcs. And this completes the proof. □

Methods to find exact solutions to graph partitioning problems have been suggested in the literature. For example, Minoux [24] has proposed a method to solve large-scale set covering/set partitioning problems by linear relaxation and column generation. However, results obtained by this method only set lower bounds which can subsequently be exploited within branch-and-bound procedures to get optimal integer solutions to the problem under consideration. Since these methods are computationally too expensive, due to the size of VLSI circuits, we have not persued them and have contended ourselves with simple and fast heuristics, which will be explained in Section 4.

3. Fundamental properties of subcircuits

Let $G = (X, A)$ be a circuit graph. In the process of partitioning G into k subgraphs, G will be augmented by the addition of pseudo-inputs/outputs. The augmented graph will be denoted by $G' = (X', A')$, with $X' \supseteq X$, and $X' = \boldsymbol{E} \cup P \cup \boldsymbol{S}$, where $\boldsymbol{E}$ is the set of primary and pseudo-inputs, and $\boldsymbol{S}$ represents the set of primary and pseudo-outputs. In the remainder of this paper, unless otherwise stated, the terms *inputs* and *outputs* will indicate both primary and pseudo-inputs/outputs.

G' consists of k disjoint subgraphs $G'_1, \ldots, G'_j = (X'_j, A'_j), \ldots, G'_k$ where:

(i) $X' = \bigcup_{j=1}^{k} X'_j$ and $X'_i \cap X'_j = \emptyset$, $\forall i \neq j$ $i, j = 1, \ldots, k$;

(ii) $|P'_j| \geqslant 1$, $\forall j = 1, \ldots, k$;

(iii) $|X'_j \cap \boldsymbol{E}| \leqslant L$, $\forall j = 1, \ldots, k$;

(iv) the subgraph $G'_j = (X'_j, A'_j)$, induced by X'_j is connected, $\forall j = 1, \ldots, k$; (alternatively G'_j has at most H non-empty connected components, where H is given and $X'_h \cap P' \neq \emptyset$, $\forall h = 1, \ldots, H$).

Example. In Fig. 8 a circuit graph G is represented, where $L = 3$, and the two arcs ($p1$, $p4$) and ($p4$, $p5$) are cut. The resulting augmented graph G' is represented in Fig. 9, where two pseudo-inputs and two pseudo-outputs have been added.

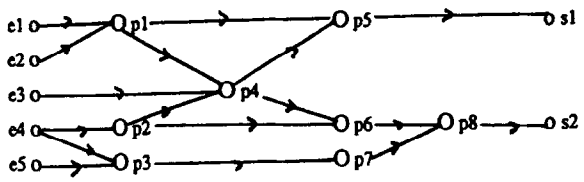

Fig. 8. Circuit graph G.

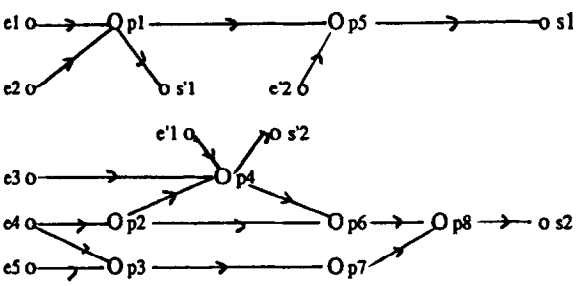

Fig. 9. Augmented circuit graph G'.

Subsets of X' have certain fundamental properties, which we have exploited in our algorithms to partition CIC's. These properties are presented in this section in the form of propositions. Proofs of these propositions are straightforward and follow immediately from conditions (i)–(iv), and from definitions below. Therefore we give proofs only for less obvious propositions.

3.1. Definitions

Correspondence: We suppose that the circuit graph $G' = (X', \Gamma)$ is represented by its set of vertices and correspondence Γ, which shows how the vertices are related to each other.

Thus $d^-(v)$, $\Gamma^-(v)$ and $\gamma_j^-(v)$ denote, respectively, the indegree, the set of predecessors and the jth predecessor of vertex v. Likewise $d^+(v)$, $\Gamma^+(v)$ and $\gamma_j^+(v)$ denote, respectively, the outdegree, the set of successors, and the jth successor of v (we assume that both sets $\Gamma^-(v)$ and $\Gamma^+(v)$ are linearly ordered for all v). When there is no ambiguity the suffix (v) will be dropped. Similarly for a subset of vertices V, $\omega^-(V)$ denotes the set of vertices not in V with at least one successor in V, and $\omega^+(V)$ denotes the set of vertices not in V with at least one predecessor in V.

Reaching set: We note $R(v)$, the reaching set of vertex v, the set of vertices that can reach vertex v, i.e., $R(v) = \{w \in X' \mid w \neq v$ and there exists a path from w to $v\}$. Similarly, the reaching set of a set of vertices, V, will be denoted by $R(V)$, where $R(V) = \bigcup_{v \in V} R(v) - V$.

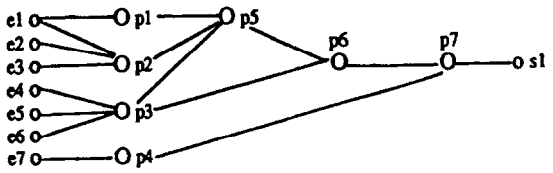

Fig. 10. Reaching sets and ancestor sets.

Example. In Fig. 10, the reaching set of gate $p6$ is: $R(p6) = \{e1, \ldots, e6, p1, p2, p3, p5\}$. And the reaching set of the subset $V = \{p6, p7\}$ will be: $R(V) = \{e1, \ldots, e7, p1, \ldots, p5\}$. Notice that $p6$ which is in $R(p7)$ is not in $R(V)$.

Ancestor set: The ancestor set of a vertex v, denoted by $\mathrm{AS}(v)$ is the set of primary and pseudo-inputs directly or indirectly feeding it, i.e., $\mathrm{AS}(v) = \{e \in R(v) | e \in \boldsymbol{E}\}$. The ancestor set of a set V of vertices is denoted by $\mathrm{AS}(V)$, where $\mathrm{AS}(V) = \bigcup_{v \in V} \mathrm{AS}(v)$. Similarly the ancestor set of subcircuit $G'_j = (X'_j, A'_j)$ will be denoted by $\mathrm{AS}(G'_j)$, where $\mathrm{AS}(G'_j) = X'_j \cap \boldsymbol{E}$.

Example. In Fig. 10, the ancestor set of gate $p5$ is: $\mathrm{AS}(p5) = \{e1, \ldots, e6\}$, and the ancestor set of $V = \{p5, p6\}$ is: $\mathrm{AS}(V) = \{e1, \ldots, e6\}$. And the ancestor set of G'_j is: $\mathrm{AS}(G'_j) = \{e1, \ldots, e7\}$, taking the whole figure as representing subcircuit G'_j.

Value: The value of vertex v, denoted by $u(v)$, is equal to the cardinality of $\mathrm{AS}(v)$, i.e. it is the number of inputs directly or indirectly feeding v, $u(v) = |AS(v)|$. Clearly $u(v)$ is equal to one for inputs because every input is fed by itself only. Similarly the value of set V of vertices will be denoted by $u(V)$, where $u(V) = |\mathrm{AS}(V)| = |\bigcup_{v \in V} \mathrm{AS}(v)|$. And the value of subcircuit G'_j will be denoted by $u(G'_j)$, where $u(G'_j) = |\mathrm{AS}(G'_j)| = |X'_j \cap \boldsymbol{E}|$.

Example. In Fig. 10, $u(p6) = 6$, and $u(V) = 7$, where $V = \{p6, p7\}$. And taking the whole figure as representing subcircuit G'_j, we will have $u(G'_j) = 7$.

Testable subsets and subcircuits: A subset of vertices is testable if $u(V) \leqslant L$. Likewise, a subcircuit G'_j is testable with respect to L, if $u(X'_j) \leqslant L$. This is equivalent to saying that subset X'_j is testable with respect to L.

Example. In Fig. 10, if $L = 5$, then subset $V = \{p1, p2\}$, with $u(V) = 3 < L$ is testable, while subcircuit G'_j is not testable, because $u(G'_j) = 7 > L$.

Maximal testable subsets: A testable subset V is maximal if $u(V) = L$, and no other testable subset W exists, such that $W \supset V$.

Example. In Fig. 10, if $L = 6$, the two subsets $V = \{e1, \ldots, e6, p1, \ldots, p3\}$, and $W = \{e1, \ldots, e6, p1, \ldots, p6\}$ are testable. It is clear that W is maximal while V is not, because $W \supset V$.

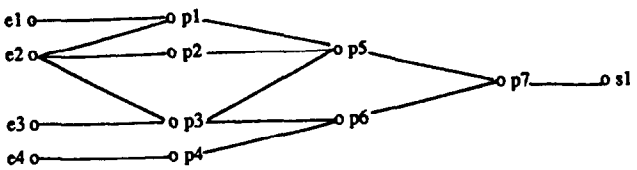

Fig. 11. Circuit 5.

Saturated testable subsets: A testable subset V is saturated with respect to L, if $u(V) < L$, and no other vertex can be added to it to make it maximal, either because $|\omega^+(V)| = 0$, and $|\omega^-(V)| = 0$; or that $u(V \cup v) > L$, $\forall v \in \omega^+(V) \cup \omega^-(V)$.

Example. In Fig. 10, the subset $V = \{e1, p1\}$ is saturated with respect to $L = 2$, since $u(V) = 1 < L$, and if we add $p2$ or $p5$ to it, its value will become greater than L. And if the whole figure is considered to represent a component, G'_j, of a larger circuit graph G', then with respect to $L = 10$, G'_j is saturated since $u(G'_j) = 7 < L$, and it is not connected to any other vertex of G' which could have been added to it.

3.2. Propositions

The purpose of this section is to present rules that we have used in our heuristic to evaluate and partition circuit graphs. Clearly, the simplest heuristic rule that can be used to partition a circuit graph is to isolate all gates by cutting the appropriate arcs so as to create $|P|$ subcircuits. This, in fact, corresponds to the worst case, because for every fan-out that is cut additional gates are added to the circuit, which have to be tested along with the existing gates. Therefore, since our objective is to minimize the number of subcircuits, a more suitable heuristic rule has to be applied.

Another simple heuristic rule is to choose arcs to be cut at random. But it should be clear from constraints imposed on the problem of partitioning a circuit graph that not all arcs in a circuit graph may be cut. Constraints (iv) of Section 2.3, implies that inputs and outputs must not be isolated. Therefore cutting those arcs that will result in isolating inputs/outputs must be avoided. For example in Fig. 11, arcs such as ($e1$, $p1$) or ($e4$, $p4$), or ($p7$, $s1$) cannot be cut. Moreover, as shown in Proposition 3.5, cutting certain arcs is redundant since the value of the vertex fed by them may remain unchanged or may even increase. For example, cutting arcs ($p1$, $p5$), ($p2$, $p5$), or ($p3$, $p5$) in Fig. 11 is redundant, as the value of $p5$ will remain unchanged. *Thus a rule is needed to decide which arcs are to be cut.*

Now suppose that a rule exists, for example suppose that a list of potential arcs for cutting is prepared and arcs are chosen at random from this list. Once an arc is cut, it should be verified if further cuts are required or not. For example, suppose that in Fig. 11 arc ($p3$, $p6$) is randomly chosen from this list to be cut. However, although the value of $p6$ is reduced to 2, the value of $s1$ is increased to 5. *Therefore a rule is needed to correctly evaluate the effect of cutting arcs and to decide whether further cuts are required.*

A simple rule that may be applied is to re-evaluate all vertices in the reachability set of the pseudo-input created after each cut, and check the new values of outputs. Then as long as there are outputs with value superior to L, the above steps of choosing an arc at random from the list, and re-evaluating and checking the values of outputs have to be repeated.

However, as shown in Lemma 3.2 this condition is not sufficient, i.e., even if the values of all outputs of a circuit are less than or equal to L, the circuit may still not be testable. *Therefore a rule is required to define subcircuits that have been created in order to count the number of inputs feeding them and to make them mutually independent by further cuts, when necessary.*

It is obvious that if we create larger subcircuits having as many inputs as possible, then the total number of subcircuits will be reduced. This is why in our algorithm our objective has been to create subsets as close to maximal as possible. And the rules used for cutting arcs, which are presented in Lemma 3.4, are derived from this objective.

In this section we will first present a recursive formula which is used in heuristic "CEP" to evaluate the vertices of a circuit graph, then we will present the rules that we have used in our heuristic to add vertices to a given subset in order to make it either maximal or saturated. Then rules to cut arcs in order to isolate this subset from the rest of the circuit graph are presented.

3.2.1. Evaluating the vertices of a circuit graph

Evaluating the vertices of a circuit graph is the procedure to determine how many ancestors directly or indirectly feed each vertex. There are several ways in which this can be done. One method, which is quite straightforward and has been used in all three heuristics "MRL", "EPS", and "CEP", is explained in the following lemma, where a recursive definition of the ancestor set is given.

Lemma 3.1. *The ancestor set of a vertex v, $AS(v)$, can be determined by forming the union of the AS's of the predecessors of v: $AS(v) = \bigcup_{i=1}^{d^-(v)} AS(\gamma_i^-(v))$.*

If the circuit graph has a tree structure, i.e., it has no reconvergent fan-outs, then evaluation is simpler, as it can be seen by the following proposition and its corollary. Needless to say, only combinational circuit graphs may have a tree structure.

Proposition 3.1. *In a combinational circuit graph without reconvergent fan-outs, the value of every vertex is equal to the sum of the values of its predecessors.*

Proof. In a circuit without reconvergent fan-outs, there is only one path from each ancestor of vertex v to v. Therefore each ancestor of v can be the ancestor of only one predecessor of v. In other words:

$$\mathrm{AS}(\gamma_i^-(v)) \cap \mathrm{AS}(\gamma_j^-(v)) = \emptyset, \quad \text{for all } i \neq j, \quad i, j = 1, \ldots, d^-(v).$$

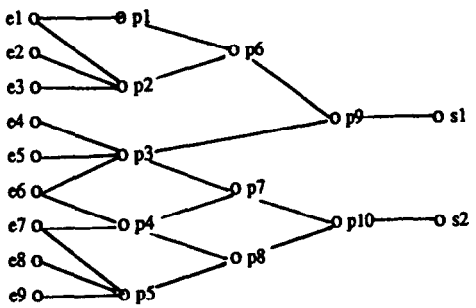

Fig. 12. A combinational circuit graph with reconvergent fan-outs.

As a consequence:

$$u(v) = \left| \bigcup_{i=1}^{d^-(v)} \mathrm{AS}(\gamma_i^-(v)) \right| = \sum_{i=1}^{d^-(v)} u(\gamma_i^-(v)). \quad \square$$

Corollary 3.1. *If all the predecessors of a vertex, v, have distinct ancestors, then even if the circuit graph does not have a tree structure, the value of v is equal to the sum of the values of its predecessors:*

$$u(v) = \left| \bigcup_{i=1}^{d^-(v)} \mathrm{AS}(\gamma_i^-(v)) \right| = \sum_{i=1}^{d^-(v)} u(\gamma_i^-(v)).$$

Example. In Fig. 12, a circuit graph is presented, which has reconvergent fan-outs. Nevertheless, since the predecessors of gate $p9$ have distinct ancestors: $\mathrm{AS}(p3) = \{e4, e5, e6\}$, and $\mathrm{AS}(p6) = \{e1, e2, e3\}$. then it follows that: $u(p9) = u(p3) + u(p6) = 3 + 3 = 6$.

3.2.2. Properties of reaching and ancestor sets

Reaching and ancestor sets of every vertex have important properties summarized in Propositions 3.2 and 3.3. These properties are used in other propositions.

Proposition 3.2. *The reaching set of every vertex $v \in X'$, and of every subset of vertices V, have the following properties:*

(1) $R(v) \supseteq \mathrm{AS}(v)$ $\quad \forall v \in X'$,

and $R(V) \supseteq \mathrm{AS}(V)$ $\quad \forall V$,

(2) $\mathrm{AS}(v) = \mathrm{AS}(R^-(v)) = R(v) \cap \boldsymbol{E}$ $\quad \forall v \in X'$,

and $\mathrm{AS}(V) = (V \cup R(V)) \cap \boldsymbol{E}$ $\quad \boldsymbol{A}V$,

(3) $|\omega^-(R(v))| = 0$ $\quad \forall v \in X'$,

and $|\omega^-(R(V))| = 0$ $\quad \forall V$,

(4) $\forall w \in R(v)$: $R(v) \supset R(w)$, $\mathrm{AS}(v) \supseteq \mathrm{AS}(w) \forall v$, $V \in X'$,

(5) $V' \supseteq V \Rightarrow u(V') \geqslant u(V)$ $\forall V$,

(6) $u(V \cup R(V)) = u(V)$ $\forall V$.

Example. In Fig. 10, the reaching and ancestor sets of gates $p1$, $p5$ and $p6$, are the sets: $R(p1) = \{e1\}$, $\mathrm{AS}(p1) = \{e1\}$, $R(p5) = \{e1, \ldots, e6, p1, p2, p3\}$, $\mathrm{AS}(p5) = \{e1, \ldots, e6\}$, $R(p6) = \{e1, \ldots, e6, p1, p2, p3, p5\}$, $\mathrm{AS}(p6) = \{e1, \ldots, e6\}$.

It is clear that:

(1) $R(p6) \supset \mathrm{AS}(p6)$, and $R(p1) = \mathrm{AS}(p1)$,

(4) $R(p6) \supset R(p5)$, and $R(p6) \supset R(p1)$,

$\mathrm{AS}(p6) = \mathrm{AS}(p5)$, and $\mathrm{AS}(p6) \supset \mathrm{AS}(p1)$,

$u(p6) = u(p5)$, and $u(p6) > u(p1)$.

Proposition 3.3. *The value of a subset of vertices X'_j, is equal to the value of this set augmented by its reaching set.*

$$u(X'_j) = u(X'_j \cup R(X'_j)). \tag{3.1}$$

Proof. From Proposition 3.2, for any subset of vertices V we have

$$\mathrm{AS}(V) = (V \cup R^-(V)) \cap (\boldsymbol{E} \cup M) = \mathrm{AS}(V \cup R(V)).$$

It follows that:

$$\mathrm{AS}(X'_j) = \mathrm{AS}(X'_j \cup R(X'_j)).$$

Therefore, relation (3.1) holds. □

3.2.3. Value of a subset of vertices

In order to determine the value of a subset of vertices, it is not necessary to determine the value of every vertex in that subset. As shown in Proposition 3.4, evaluation of the outputs in a subset will be sufficient to determine its value.

Proposition 3.4. *The value of any subset X'_j is equal to the value of its set of outputs, S'_j*

$$u(X'_j) = u(S'_j).$$

Proof. We create a dummy output, s', and we join arcs (s, s'), $\forall s \in S'_j$ (see Fig. 13). Now $\forall v \in X'_j$, we have $v \in R(s')$. And since $X'_j \supset \Gamma^-(s')$, we have $X'_j = R(s')$. As

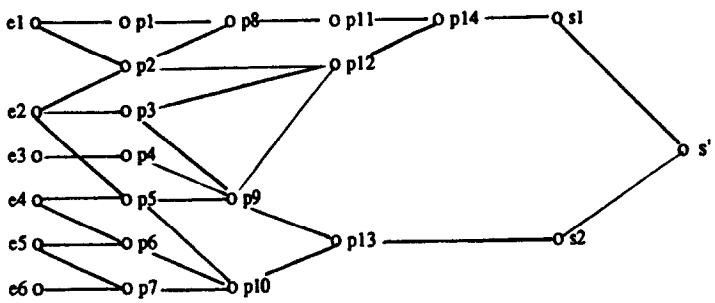

Fig. 13. A subset of vertices.

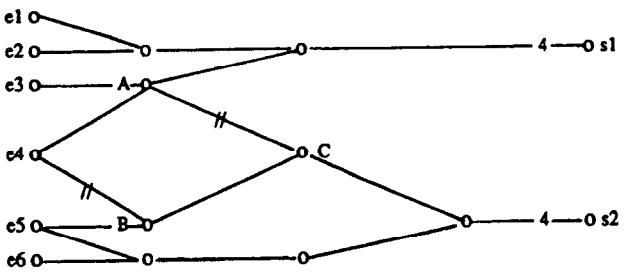

Fig. 14. A subcircuit which must be partitioned if $L = 4$.

a consequence, Proposition 3.2 implies:

$$\mathrm{AS}(X'_j) = \mathrm{AS}(R(s')) = \mathrm{AS}(s') = \bigcup_{i=1}^{d^-(s')} \mathrm{AS}(\gamma_i^-(s')) = \mathrm{AS}(S'_j).$$

It follows that $u(X'_j) = u(S'_j)$. □

Lemma 3.2. *If a subset of vertices is testable, i.e., its value is less than or equal to L, then the value of every vertex in this subset, including the outputs, is less than or equal to L.*

$$\text{If } u(X'_j) \leqslant L, \text{ then } u(s) \leqslant L, \forall s \in S'_j, \text{ and } u(v) \leqslant L, \forall v \in X'_j. \tag{3.2}$$

Proof. The proof is an immediate consequence of Proposition 3.4 and of the monotonicity of the value function (see Proposition 3.2, property (5)). □

Relation (3.2) is not a sufficient condition, i.e., $u(s) \leqslant L$, $\forall s \in S'_j$, does not imply $u(X'_j) \leqslant L$. The following example illustrates this point.

Example. In Fig. 14, a subcircuit has 6 inputs ($u(X'_j) > L = 4$), while every output has a value equal to L (values are indicated on the incoming arcs). In other words, although every vertex in this subcircuit has a value less than or equal to L, nevertheless the subcircuit has too many inputs and is not testable.

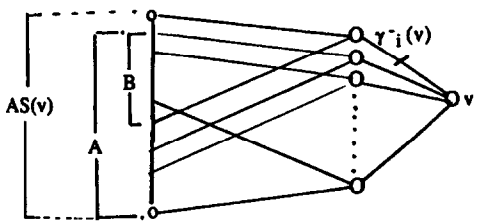

Fig. 15. Vertex v, its predecessors and ancestors.

3.2.5. Effects of cutting an arc

In order to obtain testable subcircuits, some arcs will have to be cut. Intuitively it may seem logical that for a vertex v, if an arc along the paths from inputs in $\mathrm{AS}(v)$ to v is cut, then the value of v will be reduced. This, however, is not true as shown in Proposition 3.5.

Proposition 3.5. *If an incoming arc to vertex v, with $d^-(v) > 2$, is cut, then $u(v)$ may increase, decrease or remain unchanged.*

Proof. Let $\gamma_j^-(v)$ represent the predecessor of v where arc $(\gamma_i^-(v), v)$ has been cut. Let $A = \bigcup_{j=1, j\neq i}^{d^-(v)} \mathrm{AS}(\gamma_j^-(v))$ represent the union of the ancestor sets of all other predecessors of v. Then $B = A \cap \mathrm{AS}(\gamma_i^-(v))$ will represent the common ancestors of $\gamma_i^-(v)$ with other predecessors of v, (see Fig. 15).

Let $u'(v)$ denote the new value of v after arc $(\gamma_i^-(v), v)$ has been cut and a pseudo-input created. Now:

$$u'(v) = |A| + 1. \tag{3.3}$$

In this relation, 1 stands for the value of the new pseudo-input. From definition of value we have

$$\begin{aligned} u(v) &= \left| \bigcup_{j=1}^{d^-(v)} \mathrm{AS}(\gamma_j^-(v)) \right| \\ &= |A \cup \mathrm{AS}(\gamma_i^-(v))| \\ &= |A + \mathrm{AS}(\gamma_i^-(v)) - B| \\ &= |A| + |\mathrm{AS}(\gamma_i^-(v))| - |B|. \end{aligned}$$

Finally

$$u(v) = |A| + u(\gamma_i^-(v)) - |B|. \tag{3.4}$$

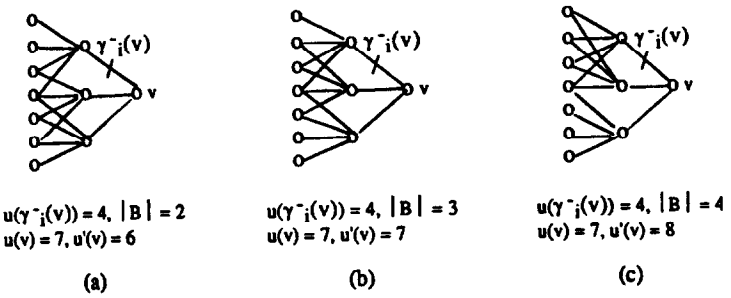

Fig. 16. Examples where the new value of a vertex may increase, decrease or remain unchanged after a cut.

Comparing the right-hand sides of relations (3.3) and (3.4), we conclude:

If $u(\gamma_i^-(v)) > |B| + 1$ then $u'(v) < u(v)$ (see Fig. 16(a)).

If $u(\gamma_i^-(v)) = |B| + 1$ then $u'(v) = u(v)$ (see Fig. 16(b)).

If $u(\gamma_i^-(v)) < |B| + 1$ then $u'(v) > u(v)$ (see Fig. 16(c)). □

3.2.6. Maximality conditions for a subset

The main objective in heuristics presented in the next section is to create testable subcircuits as close to maximal as possible. In Lemma 3.3 we will describe the conditions under which we can add vertices to a given testable subset of vertices so as to make it larger, without violating the testability condition. And in Proposition 3.6 we will give the necessary and sufficient conditions for maximality of a subset of vertices in a circuit graph.

Lemma 3.3. *Let X'_j be a testable subset of vertices. A vertex $v \notin X'_j$ can be added to X'_j without violating testability condition of X'_j,*

either if

$$v \in R^-(X'_j) \tag{3.5}$$

or if

$$\gamma_i^-(v) \in X'_j \quad \text{for some } i, \text{ and } |\mathrm{AS}(X'_j) \cup \mathrm{AS}(v))| \leqslant L. \tag{3.6}$$

Example. In Fig. 17, let $L = 5$, and

$$X'_j = \{e2, e3, e4, e5, p1, p2, p3, p4, p6, p7, p8, p9\}.$$

We may add gates $p5$, $p10$, $p11$, $p12$, $p13$, and input $e6$ to X'_j, without violating testability condition, because then the ancestor set of the new X'_j will be

$$\mathrm{AS}(X'_j) = \{e2, e3, e4, e5, e6\}, \text{ with } |\mathrm{AS}'(X'_j)| = L.$$

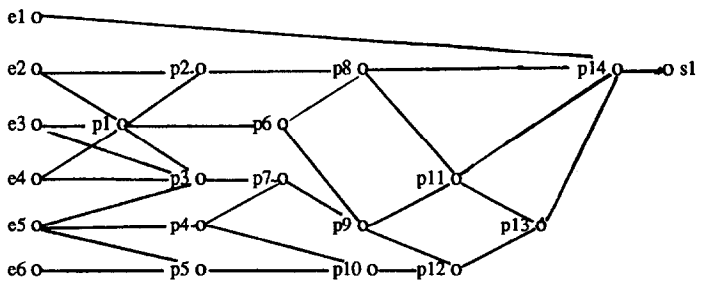

Fig. 17. Creating a maximal subset of vertices with respect to $L = 5$.

Proposition 3.6. *A testable subset X'_j of vertices is maximal with respect to L iff $u(X'_j) = L$ and either $\omega^-(X'_i) = \emptyset$ and $\omega^+(X'_i) = \emptyset$ or for any vertex $v \in X'_j$*

$$u(X'_j \cup \gamma_i^+(v)) > L, \quad \forall i. \tag{3.7}$$

Example. In Fig. 17, let $X'_j = \{e2, \ldots, e6, p1, \ldots, p13\}$, with $u(X'_j) = 5$, which is testable if $L = 5$. Now if we try to add $p14$ as a successor of $p8$ or $p11$ or $p13$, we will get

$$\mathrm{AS}(X'_j) = \{e1, e2, e3, e4, e5, e6\}, \text{ with } |\mathrm{AS}(X_i)| = 6 > L.$$

Since $p14$ is the only successor of vertices in X'_j, therefore now X'_j has become maximal.

3.2.7. Rules to cut arcs

Since our objective in the algorithm "CEP" is to create maximal subsets, we need rules to decide which arcs should be cut so as to achieve our objective. The rules which we use are described in Lemma 3.4.

Lemma 3.4. *If for any $v \in X'_j$, relation $|\mathrm{AS}(X'_j) \cup \mathrm{AS}(\gamma_i^+(v))| > L$ holds for some i, then arc $(v, \gamma_i^+(v))$ should be cut.*

Example. In Fig. 17 let $X'_j = \{e2, \ldots, e6, p1, \ldots, p13\}$, with $L = 5$. Since for gate $p14$, which is a successor of $p8$ we have: $|\mathrm{AS}(X'_j) \cup \mathrm{AS}(p14)| > L$, therefore arc $(p8, p14)$ must be cut, and pseudo-input $e7$ created. Likewise arcs $(p11, p14)$ and $(p13, p14)$ must be cut and pseudo-inputs $e8$, and $e9$ created. A second subset $X'_i = \{e1, e7, e8, e9, p14, s1\}$ is created, which is saturated, with $\mathrm{AS}(X'_i) = \{e1, e7, e8, e9\}$.

4. A polynomial complexity algorithm to partition a CIC

Since finding an optimal solution to the problem of partitioning a CIC is computationally too expensive, we propose a heuristic algorithm "CEP", which is compared with "ESP" and "MRL", presented in [13].

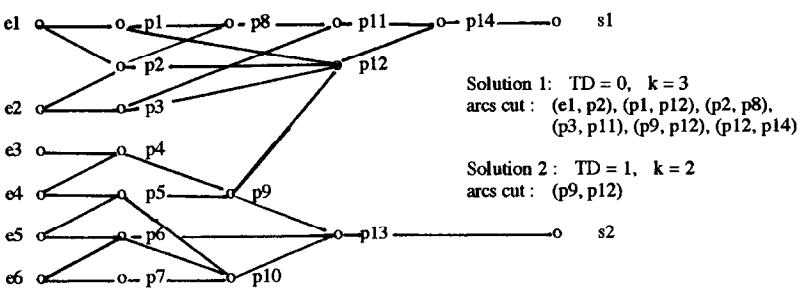

Fig. 18. Two solutions where the better one has a larger TD, with $L = 4$.

This algorithm is based on phases or procedures. *Evaluation* procedure is used either to find the value of every vertex of circuit graph G, or to re-evaluate those vertices whose value has changed after applying the partition procedure. *Partition* procedure is applied to identify and create a testable subcircuit, starting from an input.

The important features of "CEP" which distinguish it from the other two are: (i) in its evaluation procedure only vertices with value at most equal to L are evaluated, which means that "CEP" is faster and uses less static memory than the other two algorithms; (ii) its partition procedure starts from inputs unlike the other two which start from outputs; (iii) it does not need a segmentation procedure because the identification of subcircuits starts from inputs, while the other two algorithms have this procedure.

The objective in all three heuristics is to create subcircuits with as many inputs as possible, i.e., to create subsets as close to maximal as possible. For such subcircuits, constraint (iii) of Section 2.3 is binding. If all subcircuits were maximal, then the total deviation from L, given by $\mathrm{TD} = \sum_{i=1}^{k} (L - n_i - c_i)$ would be zero. Therefore, the objective function chosen is to minimize TD. It is obvious that an optimal solution with respect to this objective function is not necessarily a good solution in terms of number of cuts and number of subcircuits.

This is due to the fact that the term $\sum_{i=1}^{k} c_i y_i$ represents the number of cuts. Since this term has a negative coefficient, in order minimize TD, there may be the tendency to increase the number of cuts. This is illustrated in Fig. 18, where, if L is equal to 4, the solution with 6 cuts and 3 subgraphs gives TD of zero value, while the preferred solution of one cut and 2 subgraphs results in TD equal to one. This is why in the tables of results, we have also reported the number of cuts, and the number of subcircuits created, so as to have a more reliable basis to compare our algorithms. Moreover, it is clear that these results are not optimal. Nevertheless, since they set numerical upper bounds on the number of cuts and subcircuits that can be obtained, they can serve as initial feasible solutions which other algorithms may improve upon.

4.1. Algorithm "CEP": combinational circuit evaluation and partition

In "CEP", after calling evaluation procedure for the first time, a list of all primary inputs, called List, is prepared. Then while List is not empty, its elements are taken one by one. For an input e_i, the corresponding subgraph G_i is identified using the partition procedure. Each identified element of G_i is marked so that it will not be considered again.

During the execution of partition procedure two sets of inputs will be created. Set E_i will contain all inputs included in G_i, which are marked. Set I_i will consist of all pseudo-inputs which have been created by cuts. Therefore, inputs in E_i are eliminated from List, and inputs in I_i are added to List. Then the graph is re-evaluated from pseudo-inputs in I_i onwards, only for a subset X', which contains those vertices whose value is less than or equal to L. An algorithmic description of "CEP" is given below.

Algorithmic description of "CEP"

Step 0: Call evaluation (E, X), set List = {unmarked primary inputs}.
Step 1: While List $\neq \emptyset$ consider next element e_i of List, and do Steps 2–4.
Step 2: Call partition(e_i).
Step 3: Call evaluation(I_i, X').
Step 4: Set List = List $- E_i + I_i$, and go to Step 1.

4.2. Evaluation procedure of "CEP"

Let X^r denote the set of vertices whose AS has not been calculated, and Y^r denote the vertices that will be considered for evaluation during rth iteration. Y^r may contain two types of vertices: vertices with no predecessors, or vertices with predecessors that are either evaluated in previous iterations or whose value had not changed after application of partition procedure.

When evaluation procedure is called for the first time, initially $X^1 = X$, the set of vertices, and $Y^1 = E$, the set of primary inputs. The subsequent calls to evaluation procedure will be made to re-evaluate a subset X' of vertices; therefore $X^1 = X'$, where $X \supset X'$, and $Y^1 = I$, the set of created pseudo-inputs. Then $Z = X - X^1$ will represent the set of vertices whose value has remained unchanged and need not be re-evaluated. Once all the elements of Y^r are considered, it is substracted from X^r. The set NB (not big) consists of those vertices that have been evaluated. A dummy variable, called Union, holds the union of the ancestor sets of the predecessors of vertex v. If the cardinality of Union is less than or equal to L, then the ancestor set of v is set equal to Union, and v is added to NB. Since only vertices whose predecessors have been evaluated, can be evaluated, Y^r may become empty, in which case the algorithm will stop.

An algorithmic description is given below, where from the second step a loop is repeated until either Y^r is empty or $S \supseteq Y^r$, in which case Y^r has no successors. In the first case the algorithm will stop, in the second, after another iteration X^{r+1} will

become empty and the procedure will stop. It should be noted that the first step is superfluous, and is added only to simplify the algorithmic description.

Algorithmic description of evaluation procedure of "CEP"

Step 0: $r = 1$, $X^1 = X$ or X', $Y^1 = E$ or I, $Z = X - X^1$, $\mathrm{NB} = Y^1$.

Step 1: $\forall v \in Y^1$ set $\mathrm{AS}(v) = \{v\}$, and $u(v) = 1$.
Set $X^{r+1} = X^r - Y^r$.
While $X^{r+1} \neq \emptyset$, repeat Steps 2 and 3.

Step 2: $Y^{r+1} = \{v \in X^{r+1} | (\mathrm{NB} \cup Z) \supset \Gamma^-(v)\}$, the set of vertices whose predecessors have either been evaluated or did not need to be re-evaluated.
If $Y^{r+1} = \emptyset$ then stop else
$\forall v \in Y^{r+1}$
Union $= \bigcup_{j=1}^{d^-(v)} \mathrm{AS}(\gamma_j^-(v))$
if $|\mathrm{Union}| \leqslant L$ then set
$\mathrm{As}(v) = \mathrm{Union}$, and $u(v) = |\mathrm{Union}|$, $\mathrm{NB} = \mathrm{NB} \cup \{v\}$.

Step 3: Set $r = r + 1$, and $X^{r+1} = X^r - Y^r$.

This procedure is similar to that of Bellman and Kalaba, for labelling the vertices of a directed graph, for which the complexity is known to be $O(|A|)$, cf. [31]. Therefore, the worst-case complexity of this procedure, when all vertices are evaluated, is $O(|A|)$.

4.3. Partition procedure of "CEP"

Each application of this procedure will create one subcircuit G_i whose elements will have their mark M set equal to i. Subcircuit G_i is created starting from $X_i = \{e_i\}$, where e_i is either an unmarked primary input, or a pseudo-input created in previous calls to partition procedure. Then $\mathrm{AS}(i)$ is set equal to $\{e_i\}$. The conditions of Lemma 3.3 are used in order to add vertices to X_i until it becomes either maximal or saturated and no other vertex can be added to it. Every added vertex is marked and is put in a queue Q to be examined. Elements of Q are taken and examined on the basis of first-in first-out (FIFO). If a successor of a vertex is not evaluated or if adding it to X_i will violate testability condition, then the arc joining them is added to Cutset.

When all successors of a vertex have been considered, all arcs in the Cutset are cut, according to rules given in Lemma 3.4, and one pseudo-input/output is created for them. The psuedo-output is an element of X_i, and is marked, and the pseudo-input is added to the set I_i. This procedure is continued until Q is empty. Then all vertices with mark equal to i constitute the subgraph G_i.

Algorithmic description of partition procedure of "CEP"

Step 0: Let e_i be the unmarked primary input or pseudo-input from where this phase starts. Set $\mathrm{AS}(i) = \{e_i\}$, $v = e_i$, $Q = \{e_i\}$, $M(e_i) = i$, $I_i = \emptyset$, $E_i = \{e_i\}$, $h = 1$.
While $Q \neq \emptyset$, repeat Steps 1–4.

Step 1: Take next element v of Q on the FIFO basis. Set Cutset $= \emptyset$.

Step 2: For all unmarked predecessors $\gamma_j^-(v)$ of v set

$$M(\gamma_j^-(v)) = i, \text{ and } Q = Q \cup \{\gamma_j^-(v)\}$$

if $\gamma_j^-(v)$ is an input then set $E_i = E_i \cup \{\gamma_j^-(v)\}$.

Step 3: Consider all unmarked successors $\gamma_j^+(v)$ of v.
If $\gamma_j^+(v)$ is evaluated and if $|\mathrm{AS}(i) \cup \mathrm{AS}(\gamma_j^+(v))| \leqslant L$, then set

$$M(\gamma_j^+(v)) = i,\ Q = Q \cup \{\gamma_j^+(v)\}, \text{ and } \mathrm{AS}(i) = \mathrm{AS}(i) \cup \mathrm{AS}(\gamma_j^+(v));$$

else Cutset $=$ Cutset $\cup \{(v, \gamma_j^+(v))\}$.

Step 4: If Cutset $\neq \emptyset$, then cut arcs in Cutset, create pseudo-input e'_h, pseudo-output s'_h,

$$\text{set } M(s'_h) = i,\ I_i = I_i \cup \{e'_h\},\ h = h + 1.$$

This procedure is similar to the algorithm of Trémaux to find the connected components of a graph, and elaborated by Tarjan (complexity equal to $O(|A|)$, cf. [18]).

4.4. Complexity of algorithm "CEP"

The number of subcircuits, k, is bounded below by $\lceil |E|/L \rceil$, and it is bounded above by $|P|$, in which case each gate is in a different subcircuit. Therefore, in the worst case, where each subcircuit consists of only one gate, there will be $|P|$ calls to evaluation and partition procedures, where each procedure has a worst-case complexity $O(|A|)$.

In other words, the worst-case complexity of "CEP" will be

$$O(|P| * (|A| + |A|)) = O(|P| * |A|).$$

It should be noted that the above complexity is very pessimistic, and in practice, as shown by experimental results, "CEP" is very fast.

5. Experimental results

5.1. Data structure – ISCAS circuits

We have written a program in *C* for algorithm "CEP". Data structure used in it is identical to that used in programs corresponding to "ESP" and "MRL", which is similar to relational tables. One major difference is that we have used multi-valued as well as single-valued domains. We use three tables, for inputs, outputs, and gates. Domains for these are: the number of predecessors, the set of predecessors if they exist, the number of successors, the set of successors if they exist, the AS, and the value. The advantage of this type of data structure is that each vertex can be accessed directly by

its index number. However, this data structure uses a lot of static memory, and is not suitable for extra large circuits on small computers.

Our data consists of ten ISCAS85 (International Symposium on Circuits And Systems 1985) combinational benchmark circuits [8]. The first part of program is capable to read an ISCAS85 circuit in its neutral netlist format and translate it into the format mentioned above. Domains related to AS and value are left empty. They are calculated later by the evaluation procedure.

5.2. Results

We have run the program corresponding to algorithm "CEP" with various values of partition parameter L, ranging from 15 to 20. Starting points and decisions taken at branching points, when applicable, are identical in all cases.

Results are compared with those obtained for "ESP" and "MRL", and are reported in Tables 1–7. It should be noted that circuit c2670 is reported to have 233 inputs. But, as far as our algorithms are concerned, 76 of these inputs are redundant since they have a zero out-degree, and do not feed any of the vertices.

The first column in each table corresponds to the value of L. As expected, the larger L is, the better are the results. The second column shows the number of cuts. As it can be seen, "CEP" administers fewer cuts than "MRL" in all cases, and "CEP" performs better than "ESP" in most cases. The third column gives the percentage of cuts with respect to the existing number of gates in the circuit. It can be seen that the percentage of cuts by "CEP" decreases rapidly as the circuit grows in size. This result shows that this algorithm is more efficient for the partitioning of larger circuits. Fig. 19 shows the relationship between number of gates and the percentage of cuts by "CEP", "ESP" and "MRL" when L is equal to 20. The fourth and fifth columns represent, respectively, the number of subcircuits, k, and the total deviation from L given by

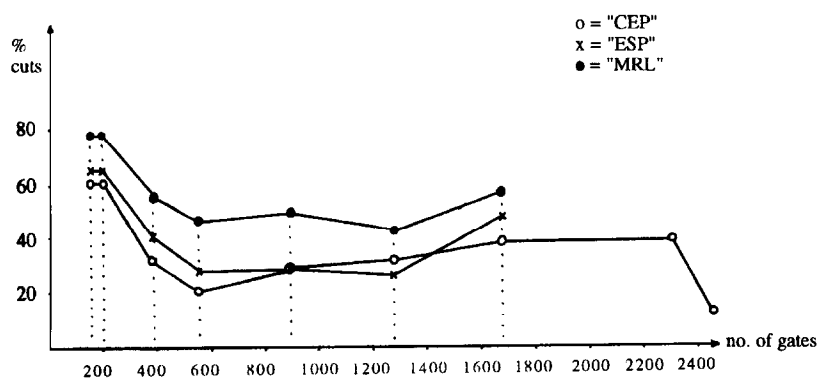

Fig. 19. Curves representing % of cuts with respect to number of gates for "CEP", "ESP" and "MRL" with $L = 20$.

Table 1
Results for C432 with 36 inputs, 160 interior and output gates, 7 outputs, max. fan-in = 9, max. fan-out = 9

Algorithm	L	No. of cuts	Cuts	k	TD	Time (s)
"CEP"	15	122	76	11	7	1.2
"ESP"	15	142	89	13	17	12.8
"MRL"	15	148	93	14	26	15.6
"CEP"	16	116	73	10	8	1.3
"ESP"	16	132	83	11	8	12.4
"MRL"	16	144	90	13	28	14.9
"CEP"	17	109	68	9	8	1.2
"ESP"	17	136	85	11	15	12.8
"MRL"	17	136	85	12	32	14.3
"CEP"	18	104	65	8	4	1.2
"ESP"	18	123	77	9	3	13.0
"MRL"	18	131	82	10	13	13.7
"CEP"	19	102	64	8	14	1.2
"ESP"	19	116	73	8	0	10.9
"MRL"	19	124	78	9	11	13.1
"CEP"	20	98	61	7	6	1.4
"ESP"	20	102	64	7	2	10.8
"MRL"	20	122	76	9	22	13.4

Table 2
Results for C499 with 41 inputs, 202 interior and output gates, 32 outputs, max. fan-in = 5, max. fan-out = 12

Algorithm	L	No. of cuts	Cuts	k	TD	Time (s)
"CEP"	15	138	68	13	16	1.8
"ESP"	15	133	66	12	6	6.1
"MRL"	15	171	85	16	28	8.7
"CEP"	16	141	70	12	10	1.9
"ESP"	16	123	61	11	12	6.6
"MRL"	16	157	78	14	26	7.8
"CEP"	17	141	70	11	5	2.0
"ESP"	17	136	67	11	10	6.6
"MRL"	17	168	83	13	12	7.7
"CEP"	18	152	75	11	5	2.1
"ESP"	18	123	61	10	16	5.8
"MRL"	18	150	74	13	43	6.8
"CEP"	19	132	65	10	17	2.0
"ESP"	19	129	64	10	20	6.9
"MRL"	19	153	76	12	34	6.9
"CEP"	20	124	61	9	15	2.3
"ESP"	20	129	64	9	10	7.2
"MRL"	20	155	77	13	64	7.0

Table 3
Results for C880 with 60 inputs, 383 interior and output gates, 26 outputs, max. fan-in = 4, max. fan-out = 8

Algorithm	L	No. of cuts	Cuts	k	TD	Time (s)
"CEP"	15	130	34	13	5	4.2
"ESP"	15	157	41	16	23	8.7
"MRL"	15	215	56	20	25	10.1
"CEP"	16	135	35	13	13	4.6
"ESP"	16	144	38	14	20	8.7
"MRL"	16	220	57	20	40	10.2
"CEP"	17	126	33	12	18	4.7
"ESP"	17	147	38	14	31	8.6
"MRL"	17	219	57	18	27	9.7
"CEP"	18	144	38	12	12	5.3
"ESP"	18	135	35	12	21	8.5
"MRL"	18	201	52	17	45	7.9
"CEP"	19	121	32	10	9	5.4
"ESP"	19	133	35	11	16	8.6
"MRL"	19	194	51	15	31	8.7
"CEP"	20	128	33	10	12	6.4
"ESP"	20	147	38	11	13	9.0
"MRL"	20	205	54	15	35	9.4

Table 4
Results for C1355 with 41 inputs, 546 interior and output gates, 32 outputs, max. fan-in = 5, max. fan-out = 12

Algorithm	L	No. of cuts	Cuts	k	TD	Time (s)
"CEP"	15	137	25	12	2	6.4
"ESP"	15	166	30	15	18	37.7
"MRL"	15	282	52	24	37	56.9
"CEP"	16	113	21	10	6	6.9
"ESP"	16	163	30	14	20	37.7
"MRL"	16	271	50	22	40	58.7
"CEP"	17	137	25	11	9	6.0
"ESP"	17	151	28	12	12	35.6
"MRL"	17	260	48	19	22	57.9
"CEP"	18	132	24	10	7	7.1
"ESP"	18	157	29	12	18	35.5
"MRL"	18	248	45	17	17	61.1
"CEP"	19	131	24	10	18	7.7
"ESP"	19	156	29	11	12	36.2
"MRL"	19	251	46	17	31	57.8
"CEP"	20	121	22	9	18	7.6
"ESP"	20	149	27	10	10	32.7
"MRL"	20	252	46	17	47	59.0

Table 5
Results for C1908 with 33 inputs, 880 interior and output gates, 25 outputs, max. fan-in = 8, max. fan-out = 16

Algorithm	L	No. of cuts	Cuts	k	TD	Time (s)
"CEP"	15	299	34	24	28	14.1
"ESP"	15	286	33	23	26	85.4
"MRL"	15	472	54	37	50	121.7
"CEP"	16	288	33	21	15	13.6
"ESP"	16	246	28	19	25	85.1
"MRL"	16	481	55	36	62	141.8
"CEP"	17	278	32	19	12	14.6
"ESP"	17	256	29	19	34	79.9
"MRL"	17	445	51	32	66	95.4
"CEP"	18	265	30	17	8	14.1
"ESP"	18	250	28	17	23	76.4
"MRL"	18	450	51	29	39	98.0
"CEP"	19	267	30	17	23	14.7
"ESP"	19	246	28	16	25	71.4
"MRL"	19	450	51	28	49	103.7
"CEP"	20	252	29	15	15	14.7
"ESP"	20	258	29	15	9	85.7
"MRL"	20	422	48	24	25	71.9

Table 6
Results for C2670 with 157 inputs, 1269 interior and output gates, 140 outputs, max. fan-in = 5, max. fan-out = 11

Algorithm	L	No. of cuts	Cuts	k	TD	Time (s)
"CEP"	15	445	35	42	28	37.1
"ESP"	15	447	35	43	41	138.0
"MRL"	15	578	46	54	75	155.4
"CEP"	16	424	33	38	27	38.4
"ESP"	16	367	29	35	36	112.3
"MRL"	16	569	45	51	90	191.1
"CEP"	17	432	34	36	23	41.2
"ESP"	17	368	29	32	19	105.3
"MRL"	17	528	42	44	63	149.4
"CEP"	18	389	31	33	48	39.7
"ESP"	18	307	24	27	22	86.1
"MRL"	18	509	40	42	90	132.2
"CEP"	19	377	30	29	17	41.4
"ESP"	19	320	25	26	17	99.4
"MRL"	19	520	41	38	45	157.3
"CEP"	20	370	29	28	33	41.8
"ESP"	20	292	23	24	31	91.6
"MRL"	20	512	40	37	71	234.8

Table 7
Results for C3540 with 50 inputs, 1669 interior and output gates, 22 outputs, max. fan-in = 8, max. fan-out = 16

Algorithm	L	No. of cuts	Cuts	k	TD	Time (s)
"CEP"	15	762	46	55	13	67.7
"ESP"	15	851	51	62	29	1926.5
"MRL"	15	1031	62	76	59	2712.8
"CEP"	16	720	43	50	30	70.6
"ESP"	16	747	45	51	19	1541.4
"MRL"	16	983	59	70	87	2909.9
"CEP"	17	726	43	47	23	59.7
"ESP"	17	769	46	51	48	1504.8
"MRL"	17	978	59	65	77	2718.3
"CEP"	18	660	40	41	28	59.2
"ESP"	18	743	45	46	35	1757.3
"MRL"	18	944	57	62	122	2499.8
"CEP"	19	640	38	38	32	59.9
"ESP"	19	759	45	44	27	2374.8
"MRL"	19	932	56	58	120	3373.2
"CEP"	20	623	37	35	27	60.7
"ESP"	20	734	44	40	16	2081.6
"MRL"	20	904	54	53	106	3270.1

$TD = \sum_{i=1}^{k} (L - n_i - c_i)$. It is clear that "CEP" minimizes k better than "ESP" and "MRL". A large TD means that many subcircuits with few inputs have been created. Finally, execution times in seconds on a Gould mini-computer are given in the last column. These values confirm that "CEP" is the fastest of all. This is due to the evaluation procedure of "CEP", which at each call evaluates a restricted number of vertices. This method requires less static memory and has enabled us to test "CEP" on the three latest ISCAS85 circuits. Results for these are reported in Tables 8–10.

Table 8
Results for C5315 with 178 inputs, 2307 interior and output gates, 123 outputs, max. fan-in = 9, max. fan-out = 15

L	No. of cuts	Cuts	k	TD	Time (s)
15	1106	48	89	51	98.8
16	1065	46	81	53	101.8
17	997	43	71	32	100.6
18	1047	45	70	35	105.1
19	902	39	59	41	98.3
20	922	40	56	20	101.6

Table 9
Results for C6288 with 32 inputs, 2418 interior and output gates, 32 outputs, max. fan-in = 2, max. fan-out = 16

L	No. of cuts	Cuts	k	TD	Time (s)
15	742	31	52	6	110.3
16	753	31	50	15	124.2
17	743	31	46	7	137.4
18	567	23	34	13	112.0
19	541	22	31	16	108.0
20	406	17	22	2	97.7

Table 10
Results for C7552 with 206 inputs, 3513 interior and output gates, 108 outputs, max. fan-in = 5, max. fan-out = 15

L	No. of cuts	Cuts	k	TD	Time (s)
15	1527	43	118	37	232.5
16	1427	41	104	31	234.5
17	1407	40	97	36	246.0
18	1334	38	87	26	252.7
19	1326	38	83	45	258.6
20	1297	37	77	37	261.0

Indeed the superiority of "CEP" over the other two algorithms is mainly due to its evaluation procedure. Both heuristics "MRL" and "ESP" failed to run on the three largest ISCAS circuits because of the memory requirements of their evaluation procedure. This is why we do not have any results reported for "MRL" and "ESP" on these circuits. Furthermore, as far as the 7 smaller ISCAS circuits are concerned, "CEP" generates definitely less cuts and less subcircuits than "MRL", and it performs better than "ESP" in most cases. Given the ability of "CEP" to run on larger circuits, its speed and its relatively better results, we conclude that it is an improvement over the two previous heuristics.

It should be noted that in the three programs "CEP, "ESP" and "MRL", evaluation procedure is systematically called after each cut. Therefore the values presented for execution times are exagerated, and the actual time needed to execute these algorithms is far less. However, since these programs are fast, we did not attempt to modify them.

6. Conclusions and further research

We have proposed a new graph partitioning model for the problem of pseudo-exhaustive testing of a VLSI combinational circuit. Since solving the theoretical

model would require a considerable amount of computations, we have considered applying heuristic algorithms. A polynomial complexity algorithm "CEP" has been presented and tested using ICSA85 benchmark circuits.

Results confirm that "CEP" is faster and more efficient than algorithms "ESP" and "MRL". Moreover, we find that "CEP" can be used for larger circuits, while the other two failed.

We have extended the proposed model to the general case of sequential circuits, with memory elements. Since "CEP" is not directly applicable to sequential circuits whose corresponding graph is not acyclic, we are at present working on a new version of algorithm "SEP" (sequential circuit evaluation and partition), suggested by Davis-Moradkhan and Roucairol [14], which can be used to evaluate and partition sequential circuits.

Since the objective function used in our algorithm does not sufficiently penalize the number of cuts, results reported here are not optimal. However, they set upper bounds on the number of cuts and subcircuits that may be created for each of the ten ISCAS85 circuits tested, and may be used as initial feasible solutions by other algorithms, such as meta heuristics (Tabu search, simulated anealing, etc.).

From the theoretical point of view, it would be interesting to find the set of problems solvable in polynomial time by studying the structure of the underlying digraph G, in particular when G is an arborescence or when it has no reconvergent fan-outs, or when G is a series-parallel DAG.

Acknowledgements

We are indebted to Mr. Srinivas Patil of the Center for Reliable and High-Performance Computing, University of Illinois at Urbana-Champaign, for his valuable help by kindly sending us the ISCAS circuits which we have used for the experimental results presented in this paper. We are also indebted to Prof. Martine Labbé of "CEME, Université Libre de Bruxelles", and to Prof. Celso Ribeiro, of the Catholic University of Rio de Janeiro, both visiting professors at "Laboratoire MASI", University of Paris 6, for their valuable suggestions, and to Prof. Alain Greiner of "Laboratoire MASI" and his VLSI team for helpful discussions. And most of all we are sincerely grateful to the referees for their valuable comments which have considerably improved the quality of this paper.

References

[1] S.B. Akers, On the use of linear sums in exhaustive testing, digest of papers, The 15th International Symposium on Fault-Tolerant Computing, IEEE (1985) 148–153.

[2] E.C. Archambeau and E.J. McCluskey, Fault coverage of pseudo-exhaustive testing, digest of papers, The 14th International Conference on Fault-Tolerant Computing (1984) 141–145.

[3] Z. Barzilai, D. Coppersmith and A.L. Rosenberg, Exhaustive generation of bit patterns with applications to VLSI self-testing, IEEE Trans. Comput. 32 (1983) 190–194.
[4] Z. Barzilai, J. Savir, G. Markowsky and M.G. Smith, The weighted syndrome sums approach to VLSI testing, IEEE Trans. Comput. 30 (1981) 996–1000.
[5] R.G. Bennets, Design of Testable Logic Circuits (Addison-Wesley, London, 1984).
[6] S.N. Bhatt, F.R.K. Chung and A.L. Rosenberg, Partitioning circuits for improved testability, in: Proceedings of the Fourth MIT Conference: Advanced Research in VLSI (MIT Press, Cambridge, MA, 1986) 91–106.
[7] S. Bozorgui-Nesbat and E.J. McCluskey, Structured design for testability to eliminate test pattern generation, digest of papers, The 10th International Symposium on Fault-Toletant Computing (1980) 158–163.
[8] F. Brglez and H. Fujiwara, A neutral netlist of 10 combinational benchmark circuits and a target translator in fortran, Presented at the special cession on ATPG and Fault Simulation, IEEE International Symposium on Circuits and Systems, Kyoto (1985).
[9] C.L. Chen, Linear dependencies in linear feedback shift registers, IEEE Trans. Comput. 35 (1986) 1086–1088.
[10] N. Christofides, Graph Theory. An Algorithmic Approach (Academic Press, London, 1975) Ch. 1.
[11] M. Davis-Moradkhan and C. Roucairol, Une heuristique pour le partitionnement des circuits logiques, Rapport MAS190.52, Laboratoire de Méthodologie et Architecture des Systèmes Informatiques, Université Pièrre et Marie Curie, Paris (1990).
[12] M. Davis-Moradkhan and C. Roucairol, Pseudo-exhaustive logic testing of VLSI circuits by partitioning, Presented at the International Workshop on Optimal Partitioning of Combinatorial Structures, Rome (1991); also: Technical Report MASI91.49 (1991).
[13] M. Davis-Moradkhan and C. Roucairol, Heuristics to partiton combinational circuits for pseudo-exhaustive logic testing, Rapport MASI92.25, Laboratoire de Méthodologie et Architecture des Systèmes Informatiques, Université Pièrre et Marie Curie, Paris (1992).
[14] M. Davis-Moradkhan and C. Roucairol, Le problème de la partition d'un graphe et ses applications à la technologie des VLSI, Presented at "Rencontres Franco-Suisses de Recherche Operationnelle", AFCET, Paris (1991).
[15] K. Furuya, A probabilistic approach to locally exhaustive testing, digest of papers, The 17th International Symposium on Fault-Tolerant Computing (1987) 62–65.
[16] M. Gondran and M. Minoux, Graphes et algorithmes, Edition Eyrolles, Paris (1985) 14–17.
[17] J. Inoue, T. Matsumura, M. Tanno and J. Yamada, Parallel testing technology for VLSI memories, in: Proceedings International Test Conference (IEEE Computer Society Press, Silver Spring, MD, 1987) 1066–1071.
[18] MASI/CAO and VLSI, Glossaire de Cellules de Bibliothèque et de Plots. Université Pierre et Marie Curie, CMP, Paris (1989).
[19] Y. Matsuda, K. Arimoto, M. Tsukude, T. Oishi and K. Fujishima, A new array architecutre for parallel testing in VLSI memories, in: Proceedings International Test Conference (IEEE Computer Society Press, Silver Spring, MD, 1989) 322–326.
[20] E.J. McCluskey, Built-in verification test, in: Proceedings International Test Conference (IEEE Computer Society Press, Silver Spring, MD, 1982) 183–190.
[21] E.J. McCluskey, Verification testing – A pseudoexhaustive test technique, IEEE Trans. Comput. 33 (1984) 541–546.
[22] E.J. McCluskey and S. Bozorgui-Nesbat, Design of autonomous test, IEEE Trans. Comput. 30 (1981) 866–874.
[23] S.D. Millman and E.J. McCluskey, Detecting bridging faults with stuck-at test sets, in: Proceedings International Test Conference (IEEE Computer Society Press, Silver Spring, MD, 1988) 773–783.
[24] M. Minoux, A class of combinatorial problems with polynomially solvable large scale set covering/partitioning relaxtions, RAIRO 21 (1987) 105–136.
[25] Y. Min and Z. Li, Pseudo-exhaustive testing strategy for large combinational circuits, Comput. Systems Sci. Engrg. 1 (1986) 213–220.
[26] E.I. Muehldorf and A.D. Savkar, LSI Logic testing – an overview, IEEE Trans. Comput. 30 (1981) 1–16.

[27] O. Patashnik, Optimal circuit segmentation for pseudo-exhaustive testing, Report No. STAN-CS-90-1308, Department of Computer Science and Medicine, Stanford University (1990).
[28] S. Patil and P. Banerjee, Fault partitioning issues in an integrated parallel test generation/fault simulation environment, in: Proceedings International Test Conference (IEEE Computer Society Press, Silver Spring, MD, 1989) 718–726.
[29] S. Patil and P. Banerjee, A parallel branch and bound algorithm for test generation, IEEE Trans. Comput.-Aided Des. 9 (1990) 313–322.
[30] M.W. Roberts and P.K. Lala, An algorithm for the partitioning of logic circuits, IEE Proc. 131 (1984) 113–118.
[31] Roseaux, Exercices et Problèmes Résolus de Recherche Opérationnelle Vol. 1 (Masson, Paris, 1991).
[32] I. Shperling and E.J. McCluskey, Circuit segmentation for pseudo-exhaustive testing via simulated annealing, in: Proceedings International Test Conference (IEEE Computer Society Press, Silver Spring, MD, 1987) 58–66.
[33] C.C. Su and C.R. Kime, Computer-aided design of pseudoexhaustive BIST for semiregular circuits, in: Proceedings International Test Conference (IEEE Computer Society Press, Silver Spring, MD, 1990) 680–689.
[34] D.T. Tang and C.L. Chen, Logic test pattern generation using linear codes, IEEE Trans. Comput. 33 (1984) 845–850.
[35] F.F. Tsui, Design for testability in LSI/VLSI systems, in: International Conference on Computers and Applications (IEEE Computer Society Press, Silver Spring, MD, 1987) 855–862.
[36] J.G. Udell and E.J. McCluskey, Efficient circuit segmentation for pseudo-exhaustive test, in: IEEE International Conference on Computer-Aided Design (1987) 148–151.
[37] N. Vasanthavada and P.N. Marinos, An operationally efficient scheme for exhaustive test-pattern generation using linear codes, in: Proceedings International Test Conference (IEEE Computer Society Press, Silver Spring, MD, 1985) 476–482.
[38] L.T. Wang and E.J. McCluskey, Condensed linear feedback shift register testing – a pseudo-exhaustive test technique, IEEE Trans. Comput. 35 (1986) 367–370.
[39] C.A. Wiatrowski and C.H. House, Logic Circuits and Microcomputer Systems (Mcgraw-Hill, New York, 1980).
[40] T.W. Williams, Design for testability, in: T.W. Williams, ed., VLSI Testing (IBM Division of General Technology, Boulder Co., Amsterdam, 1986) 95–160.
[41] T.W. Williams, Trends in design for testability, Presented at ESSCIRC '86, 12th European Solid-State Circuit Conference, Deft (1986) 146–148.
[42] T.W. Williams, Built-in self-testing, in: Proceedings of VLSI and Computers, 1st International Conference on Computer Technology Systems and Applications (1987) 345–348.