

**Hidden Markov Models with Patterns to Learn Boolean Vector Sequences
and Application to the Built-in Self-Test for Integrated Circuits**

Laurent Bréhélin Olivier Gascuel Gilles Caraux

Département Informatique Fondamentale et Applications
Laboratoire d'Informatique, de Robotique et de Microélectronique de Montpellier
Université Montpellier II, 161 rue Ada, 34392 Montpellier Cedex 5, France

<http://www.lirmm.fr/~gascuel/MAAS/US-MAAS.html>

Email: {brehelin,gascuel,caraux}@lirmm.fr

Abstract

We present a new model, derived from the Hidden Markov Model (HMM), to learn Boolean vector sequences. Our *Hidden Markov Model with Patterns* (HMMP) is a simple, hybrid and interpretable model that uses Boolean patterns to define emission probability distributions attached to states. Vectors consistent with a given pattern are equiprobable, while inconsistent ones have probability zero to be emitted. We define an efficient learning algorithm for this model, which relies on the maximum likelihood principle, and proceeds by iteratively simplifying the structure and updating the parameters of an initial specific HMMP that represents the learning sequences. Each simplification involves merging two states of the current HMMP, while keeping the likelihood as high as possible, and the algorithm stops when the HMMP has a sufficiently small structure. HMMPs and our learning algorithm are applied to the *Built-in Self-Test* (BIST) for integrated circuits, which is one of the key microelectronic problems. An HMMP is learned from a test sequence set (computed using a specific tool) that covers most of the potential faults of the circuit at hand. Then this HMMP is used as test sequence generator. Our experiments, carried out with classical microelectronic benchmark circuits, show that learned HMMPs have a very high fault coverage. Furthermore, their small sizes combined with their simplicity allow these models to be easily implemented on the circuits for self-testing purposes.

Index terms: Boolean Vector Sequence Modeling; Hidden Markov Models; Hybrid Approach; Structure (and parameters) Learning; Built-in Self-Test for Integrated Circuits.

1 Introduction

The Hidden Markov Model (or HMM) was introduced by Baum and colleagues in the late 1960s [1]. This model is closely related to probabilistic automata (PAs) [2]. A probabilistic automaton is defined by its *structure*, made up of *states* and *transitions*, and by probability distributions over the transitions. Moreover, each transition is associated with a symbol from a finite alphabet that is generated each time the transition is run over. An HMM is also defined by its structure, composed of states and transitions, and by probability distributions over the transitions. For most authors, symbol generation is attached to the states, but it can be also attached to the transitions [3]. Each state (or transition) is then associated with a probability distribution over the alphabet that expresses the probability for each symbol to be generated when the state (or transition) is encountered. These models are said to be *hidden* because the series of states (or transitions) are not apparent in the generated sequences.

When the structure is known, the HMM learning (or training) problem is reduced to estimating the value of its parameters (transition and symbol generation probabilities) from a sample of sequences. The *Baum-Welch* algorithm [4] is a well-known approach, which complies with the maximum likelihood principle and is a special case of the Expectation-Maximization (EM) algorithm [5]. This is an iterative re-estimation algorithm that ensures convergence to a local optimum. Abe and Warmuth [6] studied the training problem from a Computational Learning Theory perspective. They proved that the PA class is not polynomially trainable unless $RP=NP$ while, to the best of our knowledge, the question remains open for HMMs. However, we can reasonably assume that the problem is not easier and that heuristics have to be used.

When it is not possible to infer the HMM structure from the *a priori* knowledge we have about the problem under investigation, the HMM learning problem becomes even more difficult. We have to estimate the parameters of the structure, and also infer this structure from the training sample. A solution is to train a fully connected HMM with the Baum-

Welch algorithm (or one of its variants), leaving this one to determine the probability zero transitions. Various authors have proposed an alternative approach, derived from automata theory [7], which involves generalizing an initial specific automaton that represents the learning sample, by iteratively merging “similar” states until a “convenient” (*e.g.*, sufficiently general or small) structure is obtained. This principle has been applied to Deterministic Finite Automata (DFA) [8], Stochastic Finite Automata (SFA) [9], and HMMs [10]. In the recent Abbadingo One DFA Learning Competition [11], this principle appeared to be the best strategy for learning DFA.

HMMs are intensively used in speech recognition (*e.g.*, [3], [12]) and handwriting recognition (*e.g.*, [13], [14]). In these applications, the usual approach involves using an HMM for each word or character to be recognized. Typically, HMMs have a pre-determined left-to-right structure of fixed size, and they are trained using the Baum-Welch method. A different approach is proposed for speech recognition in [10]: a state merging algorithm learns both the HMM structure and parameters; the structure is not *a priori* determined, but the usual constraints of the application are used, *e.g.* the left-to-right topology. HMMs are successfully used in many other recognition applications, such as computational biology [15] or texture classification [16], and also for various segmentation tasks (*e.g.* [17]). In most of these applications, the learning scheme is similar to that employed for speech and handwriting recognition, *i.e.* a special structure is pre-determined and then trained with the Baum-Welch algorithm or one of its variants.

In this article, we present a new application of HMMs: the *Built-in Self-Test (BIST)* for integrated circuits. Due to the increasing complexity of integrated circuits, in recent years this test technique has become one of the most important research areas in the microelectronic community [18]. Our approach involves learning an HMM from a set of *test sequences*. These Boolean vector sequences (obtained using a specific tool) are able to detect most potential faults in the circuit at hand. Then the learned HMM is used to generate sequences similar to the original test (learning) sequences, and having good fault detection capabilities. Moreover, since this generator is intended to be electronically

implemented, the aim is to obtain an HMM of low size, which may be seen as a compression of the learning sequences. BIST differs from classical HMM applications for several reasons. First we use HMMs for generation purposes, which involves specificities regarding recognition tasks. Secondly, size constraints require appropriate solutions for the model to be efficiently implemented on circuits. This point is particularly important concerning the choice of emission probability distributions attached to states, and led us to develop a new class of HMM, which we called Hidden Markov Model with Patterns (HMMP). Finally, there is no *a priori* knowledge to infer the HMM structure, which is learned from the original test sequences using a state merging algorithm.

The organization of this paper is as follows. In Section 2, we rapidly present the problem of testing integrated circuits (see also [18]); we indicate the main features of the manipulated data and explain how this problem can naturally be dealt with using HMMs. In Section 3, we define HMMPs and present the method for estimating its parameters using the Maximum Likelihood principle. In Section 4, we detail our HMMP learning algorithm. Section 5 is devoted to experimental results. We first present the performance of our method with the classical benchmarks of the test community, then compare the results with those of alternative models and learning algorithms. In Section 5.2, a brief discussion wraps up the paper.

2 Testing integrated circuits

An integrated circuit is composed of three parts:

- inputs;
- outputs;
- the body of the circuit, which is made up of interconnected *logic gates* and *memory cells*.

The manipulated values are Boolean (0 or 1). The logic gates are operators (or functions) such as AND, OR, NOT, NAND and NOR. Memory cells have one input, one output and store a Boolean value. An integrated circuit is *sequential* if it contains memory cells, and *combinational* if it is memoryless. Note that we can apply values to circuit inputs and read its output values, but it is not possible to access its internal elements.

Integrated circuits may be affected by different failures. We usually consider that the logic conception of the circuit is correct, but that an external factor has physically altered one of its components. Such failures are named *faults*. Since the logic structure of the circuit is known, we can infer its set of potential faults. Testing a fault of a circuit involves checking that this fault is not present. Testing a circuit involves testing all (or most) of its potential faults. The *fault coverage* of a test is the ratio of the number of tested faults over the total number of potential faults.

A *test vector* is a set of Boolean values simultaneously applied to the inputs. For a circuit with k inputs, a test vector has length k . Testing a potential fault of a combinational circuit can be achieved using a single test vector. In this article, we deal with testing sequential circuits (general and common case), which requires sequences of test vectors in order to initialize the memory cells. Then the test of a potential circuit fault is achieved by applying such a sequence and comparing the obtained output values with those logically expected. If these values are equal, we conclude that the fault is absent. All inputs do not have to be specified to test a given fault, and only some inputs are determinant. So *Boolean patterns* are used rather than test vectors. A Boolean pattern is a vector defined over $\{1, 0, *\}^k$. A bit is set at $*$ if its value does not influence the result, *i.e.* if the fault occurs, it is detected regardless of the value of this bit. A Boolean pattern defines a set of vectors; a pattern with $n *$ bits represents a set of 2^n vectors. In the same way, a sequence of patterns defines a set of vector sequences. We say that a pattern sequence *tests* a fault if every vector sequence it defines detects this fault. Such sequences are named *test sequences*. In a test sequence, the order of patterns is as important as the patterns themselves. However, one sequence usually tests several faults, and one fault may be tested by several sequences. Moreover,

the sequence length varies depending on the tested fault. Note too that we can easily deduce, by simulation, all potential faults detected by a given test sequence and hence compute the fault coverage of any sequence set. This is used in our experiments (Section 5) to evaluate the performance (fault-coverage) of learned HMMPs and to calibrate their size.

The search of a test sequence for a given circuit and a given fault is a hard task. For combinational circuits, the problem of existence of a test pattern is *NP-Complete* [19]. For sequential circuits, the task is even more difficult. Nevertheless, *Automatic Test Pattern Generators (ATPGs)* are available for such purposes. They try to circumvent the difficulty by using stochastic algorithms (*e.g.*, genetic [20]), or heuristics based on topological analysis [21] or fault simulation [22].

A set of test sequences is shown in Annex. It was produced by the ATPG of *Sunrise*¹, for the classical benchmark² circuit *s820*. We note two essential features of test sequences: their patterns are hollow (they have few fixed bits), and many patterns appear several times (*e.g.*, *010*****0001).

When test sequences with sufficient fault coverage have been produced (which is solved by ATPGs with various degrees of success depending on the circuits), we have to position the test procedure. The classical approach involves using an external tester. Due to the price of these testers and sometimes the fact that there is no physical access to the tested circuit, the *Built-in Self-Test* or *BIST* method is often preferred. The BIST principle is to incorporate a supplementary test structure into the circuit. This structure should be able to generate test sequences and analyse circuit responses. Response analysis is a task that has efficient solutions [18]. This is not the case for test sequence generation. The problem is to find a generator of sequences that combines high fault coverage and small size (in terms of silicon area). We cannot physically stock all ATPG sequences because

¹©Sunrise test system. A Viewlogic Company.

²These benchmarks were created for the *International Symposium on Circuits & Systems (ISCAS)* of 1989. They are available at <http://www.cbl.ncsu.edu/benchmarks/>

of the silicon cost. Otherwise, we can build small generators of pseudo-random sequences, but they have insufficient fault coverage.

Given the fact that different sequences can test the same fault and hence that ATPG sequences are not the only possible sequences, our approach relies on statistical modeling techniques. This involves learning from the ATPG sequences an instance of a new class of HMM (called HMMP) that generates ATPG sequences or similar sequences with sufficiently high probability. We shall see (Section 5) that, at the price of larger but reasonable test sequence sets, relatively small HMMPs effectively generate sequences with fault coverage as high as that of the ATPG.

3 A new class of HMMs: HMMPs

The HMMs we want to infer are intended to generate Boolean vector sequences. There are numerous solutions for modeling such sequences. The most natural is to consider that bits of Boolean vectors are independent and hence to use products of Bernoulli distributions to define the emission probabilities. Unfortunately, due to the great number of Bernoulli distributions involved – one for each bit of each state –, this approach raises size problems when implementing the obtained models as test sequence generators. This becomes particularly important when the vector length (number of inputs of the circuit) is high. To deal with this difficulty, we introduce very simple distributions based on discrete probabilities with low implementation cost. Using these distributions in the HMM framework leads to the *Hidden Markov Model with patterns* or *HMMP*. Section 3.1 defines both of these distributions and the HMMP. Sections 3.2 and 3.3 provide the likelihood of a set of ATPG sequences for a given HMMP and the way to compute this likelihood, while Section 3.4 presents the maximum likelihood estimators of HMMP parameters.

3.1 Definitions

To define the HMMP emission probabilities, we use products of Bernoulli distributions whose parameters cannot handle all possible values between 0 and 1 but are constrained to 0, 1 or 1/2. The advantage of such distributions for the BIST is that these probabilities are easily implemented: probability 1/2 is free, 1 and 0 have very low cost in terms of silicon area [23], while implementing continuous probabilities is much more expensive. A distribution defined in this way can be represented by a Boolean pattern similar to those introduced in the previous section: bits with parameter 0 are set at 0, those with parameter 1 at 1, and those with 1/2 at *. Such patterns will be called *emission patterns* in the following to avoid confusion with the test patterns of ATPG sequences. In distributions defined by emission patterns, the vectors consistent with the pattern are equiprobable, while the others have probability zero. For example, in the distribution *1*, vectors 010, 011, 110 and 111 have probability 1/4, while the four other vectors have probability zero. Such extreme (zero) probabilities are usually avoided in recognition tasks, so as to be able to recognize new unseen sequences. Our aim is different and involves generating sequences as close as possible to ATPG sequences, not new sequences with unseen vectors.

An HMMP H is defined by a triplet $\langle S, P, M \rangle$, where

- S is a finite set of states; S contains two special states *start* and *end* which are used to initiate and conclude a sequence, respectively. Each state of S , except *start* and *end*, is labeled by an emission pattern from P .
- $P = \{p_s, s \in S - \{start, end\}\}$ is the set of emission patterns associated with the states; p_s is the emission pattern associated with state s .
- $M : S - \{end\} \times S - \{start\} \rightarrow [0, 1]$ is the matrix that contains transition probabilities between states. We have: $\forall s, t, M(s \rightarrow t) \geq 0$, and $\forall s \neq end, \sum_{t \in S} M(s \rightarrow t) = 1$.

The *structure* of an HMMP is the set of its states and of its non-zero transitions. Figure 1 gives an example of HMMP with seven states and nine transitions.

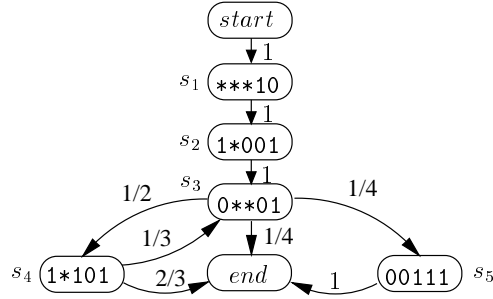


Figure 1: Example of HMMP: each state is labelled with its name and associated emission pattern; transitions are labeled with their transition probability.

3.2 Likelihood of a set of test sequences

We say that two patterns p and p' are *compatible* if the sets of vectors they represent have a non-empty intersection. This is expressed by the fact that their fixed bits (those with value 0 or 1) do not differ. For example, $p = 11^*$ and $p' = 1^*0$ are compatible, while $p = 11^*$ and $p' = 100$ are not compatible. Let p be a test pattern (a set of vectors) and s a state of an HMMP; the *emission probability of p by s* is the probability that in generating a vector with distribution p_s we generate a vector of p . If p and p_s are not compatible, the emission probability of p by s is zero. For a compatible pattern, the emission probability depends on the number of bits which are fixed in p but have the value $*$ in p_s . Let $*_{p_s}^p$ denote this number. For example, if $p_s = 1^{**}$ and $p = 10^{**}$ then $*_{p_s}^p = 1$. Since $*$ has equiprobability of generating a 0 or a 1, the probability of emitting the pattern p on the state s is given by

$$L(s, p) = \left(\frac{1}{2}\right)^{*_{p_s}^p} \tag{1}$$

when p is compatible with p_s , and 0 otherwise. For example, the probability of emitting 1^*01^* with $p_s = 1^*010$ is 1, while the probability of emitting 1^*010 with $p_s = ^{***}10$ is $\frac{1}{4}$.

The *generation probability $P(x|H)$* of a test pattern sequence x by an HMMP H is the probability of generating with H a Boolean vector sequence that belongs to the test

sequence set defined by x . Let x be equal to $p_1 p_2 \dots p_l$, with $p_i \in \{0, 1, *\}^k$, we have:

$$P(x|H) = \sum_{\langle s_0, \dots, s_{l+1} \rangle \in S^{l+2}} \left[\left(\prod_{i=0}^{l-1} M(s_i \rightarrow s_{i+1}) L(s_{i+1}, p_{i+1}) \right) M(s_l \rightarrow s_{l+1}) \right],$$

with S^{l+2} being the set of sequences with $l + 2$ states from S , such that $s_0 = start$ and $s_{l+1} = end$. The *likelihood* of a set of test pattern sequences X for H , represents the probability of generating, with $|X|$ trials, the set X with H :

$$P(X|H) = |X|! \prod_{x \in X} P(x|H). \quad (2)$$

3.3 The Viterbi approximation

A common simplification used to compute the generation probability of a sequence by an HMM is the *Viterbi assumption* that the sequence can only be generated by a unique path (or sequence of states) through HMM. In other words, all paths, except the most likely are assumed to have a negligible probability. Applied to HMMPs, the most likely path in H , named *Viterbi path* [24], of a pattern sequence $x = p_1 \dots p_l$, is defined as

$$V_x = \operatorname{argmax}_{\langle s_0, \dots, s_{l+1} \rangle \in S^{l+2}} \left[\left(\prod_{i=0}^{l-1} M(s_i \rightarrow s_{i+1}) L(s_{i+1}, p_{i+1}) \right) M(s_l \rightarrow s_{l+1}) \right]. \quad (3)$$

For example, consider the set of three sequences below:

Sequence 1	Sequence 2	Sequence 3
***10	***10	***10
1*001	1*001	1**01
0*101	0*101	0*001
10*01	11101	00111
0**01		

The Viterbi path of Sequence 1 for the HMMP depicted in Figure 1 is $start - s_1 - s_2 - s_3 - s_4 - s_3 - end$. Moreover, this is the only path that can generate this sequence, which often occurs in practice, and the Viterbi assumption holds in this case.

Let us now introduce some notations that we shall use in the following. If X is a set of pattern sequences and H an HMMP, we note:

- P_X : the set of test patterns included in sequences from X ;
- V_x : the Viterbi path of sequence x , and $V = \{V_x, x \in X\}$ the set of Viterbi paths of sequences from X ;
- n_s , $n_{s \rightarrow t}$ and n_s^p : the number of times state s is used in V , the number of times transition $s \rightarrow t$ is used (in V), and the number of times state s emits the pattern p , respectively;
- $P_{X,s} = \{p \in P_X, n_s^p \neq 0\}$: the set of patterns from X emitted by state s in V ($P_X = \cup_{s \in S} P_{X,s}$);
- $Out(s)$ and $In(s)$: the sets of out and in states from and to the state s with non-zero transition probability, respectively; by convention, we consider that loops are outgoing transitions, *i.e.* if s loops to itself, then we have $s \in Out(s)$.

Using these notations, the Viterbi approximation of Expression (2) can be written as:

$$P(X|H, V) = |X|! \left(\prod_{s \in S} \prod_{t \in S} M(s \rightarrow t)^{n_{s \rightarrow t}} \right) \left(\prod_{s \in S} \prod_{p \in P_{X,s}} L(s, p)^{n_s^p} \right). \quad (4)$$

Let E_s be defined by:

$$E_s = \prod_{p \in P_{X,s}} L(s, p)^{n_s^p}, \quad (5)$$

and assume $E_{start} = E_{end} = 1$. Expression (4) can then be rewritten as

$$P(X|H, V) = |X|! \prod_{s \in S} \left(E_s \prod_{t \in Out(s)} M(s \rightarrow t)^{n_{s \rightarrow t}} \right). \quad (6)$$

This Formula underestimates the exact likelihood defined by Formula (2). As we shall see, our approach (as others, *e.g.* [10], [26]) involves maximizing Formula (6), instead of

Formula (2). Because (6) is equal to or smaller than (2), but never greater, maximizing (6) actually tends to maximize (2), which explains the efficiency of this (sometimes rough) approximation. Furthermore, we shall see below that using the Viterbi approximation provides an efficient way to simplify and greatly speed up the learning procedure.

3.4 Estimation of transition and emission probabilities

We saw (Section 1) that the HMM training problem, *i.e.*, the problem of estimating parameters of a known structure from a given sample of sequences, is a hard task. Nevertheless, when the Viterbi paths of the sample are known, there is a simple efficient approximation [15]. The same method applies to HMMPs. It uses the maximum likelihood principle and searches the parameter values which maximize Expression (6). This is equivalent to individually maximizing each sub-product E_s and $\left(\prod_{t \in \text{Out}(s)} M(s \rightarrow t)^{n_{s \rightarrow t}}\right)$ of this expression.

We recognize in $\left(\prod_{t \in \text{Out}(s)} M(s \rightarrow t)^{n_{s \rightarrow t}}\right)$ an expression identical to that of a multinomial law distribution. Then, the values that maximize it are

$$M(s \rightarrow t) = \frac{n_{s \rightarrow t}}{n_s}, \quad (7)$$

which gives us a simple estimation of the M parameters.

For emission probabilities, the problem is slightly more difficult and requires additional notations. Let $\delta_*(p, r)$ be the binary function that returns 1 if the bit of rank r in p is equal to $*$, and 0 otherwise. Let $\delta_{\neg*}(p, r)$ be the negation of this function, and

$$n_{\neg*}(P_{X,s}, r) = \sum_{p \in P_{X,s}} n_s^p \cdot \delta_{\neg*}(p, r) \quad (8)$$

the number of bits fixed on position r in the patterns emitted by s .

For every s , we have to find the emission pattern p_s that maximizes E_s . From Expressions (1) and (5), it follows that p_s must be compatible with all patterns of $P_{X,s}$, otherwise the value of E_s is zero. Therefore, if there are patterns in $P_{X,s}$ for which some bits are differently fixed, then p_s must have the value $*$ on these bits. Once this constraint is satisfied,

when incorporating Expression (1) in Formula (5), we obtain:

$$E_s = \left(\frac{1}{2}\right)^{\sum_{p \in P_{X,s}} *_{p_s}^p n_s^p}, \quad (9)$$

that can be rewritten as

$$E_s = \left(\frac{1}{2}\right)^{\sum_{r=1}^k \delta_*(p_s, r) \cdot n_{\neg *}(P_{X,s}, r)}. \quad (10)$$

Maximizing Expression (10) is equivalent to minimizing, for every r , the expression:

$$\delta_*(p_s, r) \cdot n_{\neg *}(P_{X,s}, r). \quad (11)$$

We saw that, for a given position r , if there are incompatible bits in $P_{X,s}$ then p_s has the value $*$ on this position. Now if all bits on a given position r are compatible, two cases are possible. Either some bits have a given fixed value (0 or 1); then, to minimize Expression (11), the corresponding bit of p_s has to be fixed at the same value because we then have $\delta_*(p_s, r) = 0$. Or every bit on this position has the value $*$. Then, we have $n_{\neg *}(P_{X,s}, r) = 0$ and all possible solutions (0, 1 or $*$) minimize Expression (11). However, we shall see in the following that it is more relevant to keep p_s as general as possible, and thus we use the value $*$.

It follows that the emission pattern p_s can be estimated by independently computing each of its bits from the corresponding bits in the patterns from $P_{X,s}$. Now, we shall see that p_s can be computed iteratively, which accelerates our learning algorithm by state merging (Section 4.1). For this purpose, we use an associative and commutative operator denoted as γ . According to the above, for incompatible bits we have $\gamma(1, 0) = *$, while for compatible bits we have: $\gamma(0, 0) = \gamma(*, 0) = 0$, $\gamma(1, 1) = \gamma(*, 1) = 1$, and $\gamma(*, *) = *$. However, γ is not associative as defined. For example, $\gamma(1, \gamma(1, 0)) = \gamma(1, *) = 1$ whereas $\gamma(\gamma(1, 1), 0) = \gamma(1, 0) = *$ which is the right solution because there are incompatible bits. Therefore, when $*$ is obtained from $\gamma(1, 0) = *$, it has to conserve this value during subsequent steps. To solve this difficulty, we mark the $*$ obtained in this way; the marked $*$ are denoted as $\bar{*}$. The γ operator takes this into account and we have: $\gamma(1, 0) = \gamma(\bar{*}, 0) =$

γ	1	0	*	$\bar{*}$
1	1	$\bar{*}$	1	$\bar{*}$
0	$\bar{*}$	0	0	$\bar{*}$
*	1	0	*	$\bar{*}$
$\bar{*}$	$\bar{*}$	$\bar{*}$	$\bar{*}$	$\bar{*}$

Table 1: Response table for the γ operator.

$\gamma(\bar{*}, 1) = \gamma(\bar{*}, *) = \bar{*}$. Table 1 summarizes the response table of the γ operator. For example, when $P_{X,s} = \{10*, 110, *1*\}$ then $p_s = \gamma(10*, \gamma(110, *1*)) = \gamma(10*, 110) = 1\bar{*}0$.

4 HMMP Learning

Let X be a set of pattern sequences (*e.g.* computed by an ATPG). Our aim is to build an HMMP of low size (*i.e.* with a low number of states and transitions) that generates X with as high as possible probability. Since we have no *a priori* knowledge about the structure of this HMMP, a solution could be to train several fully connected HMMPs of different sizes with the Baum-Welch algorithm or one of its variants, and to keep the best obtained model. Unfortunately, as we shall see in Section 5.2, the non-continuous nature of probabilities involved in emission patterns lowers the efficiency of these algorithms. So we chose a state merging approach [10]. The main difficulty of this latter is the computing time, usually higher than that of the classical Baum-Welch algorithm, and emphasis will be given in this section to algorithmic refinement in order to reach acceptable runtimes. This point is critical for practical applications, owing to the size of test sequences. For example, in our experiments (Section 5), one of the largest test sequence sets has a total length of 5616 vectors with size 40. Moreover, circuits are now often bigger than those of the ISCAS'89 benchmarks, and it is common to have test sequence sets of about 100,000 vectors with size 100.

4.1 Outline of the algorithm

The learning algorithm (HMMPLEARNING, summarized below) proceeds in a greedy ascending way. From a specific HMMP that represents sequences from X , and only these sequences, it progressively builds a simpler HMMP using an elementary operation (merging of two states). This simplification usually induces a generalization (more sequences have non-zero probability of being generated) but also a loss of likelihood of the learning sequences. This method builds a series of HMMP: $H_0, H_1, \dots, H_i, \dots$ where H_i is the HMMP obtained by merging two states of H_{i-1} . Figure 2 shows an example of a run with three state mergings for the set of three sequences proposed in Section 3.3. The algorithm stops when the desired number of states (N) is reached. N is just used to stop the algorithm, and the final HMMP is not necessarily the one that will be used and implemented on the circuit. As opposed to recognition tasks, choosing the right size in the BIST framework is relatively easy. A simple method (used in Section 5) involves simulating HMMPs with decreasing sizes (as obtained from the learning algorithm), computing the fault coverage for each one, and selecting the best one according to some size/fault coverage compromise.

In the following, we first (Section 4.2) present the initial HMMP building procedure. Sections 4.3 and 4.4 respectively describe the state merging procedure and the criterion used to select the state pair to be merged at each step. In Section 4.5, we describe a method to efficiently compute the best state pair according to this criterion. The time complexity of the whole learning procedure is discussed in Section 4.6.

Algorithm 1: HMMPLEARNING

```
Data :  $X, N$   
 $H \leftarrow \text{CREATEINITIALHMMP}(X)$ ;  
while  $|S| > N$  do  
   $(s_i, s_j) \leftarrow \text{SELECTBESTSTATEPAIR}(S)$ ;  
   $H \leftarrow \text{MERGE}(H, s_i, s_j)$ ;
```

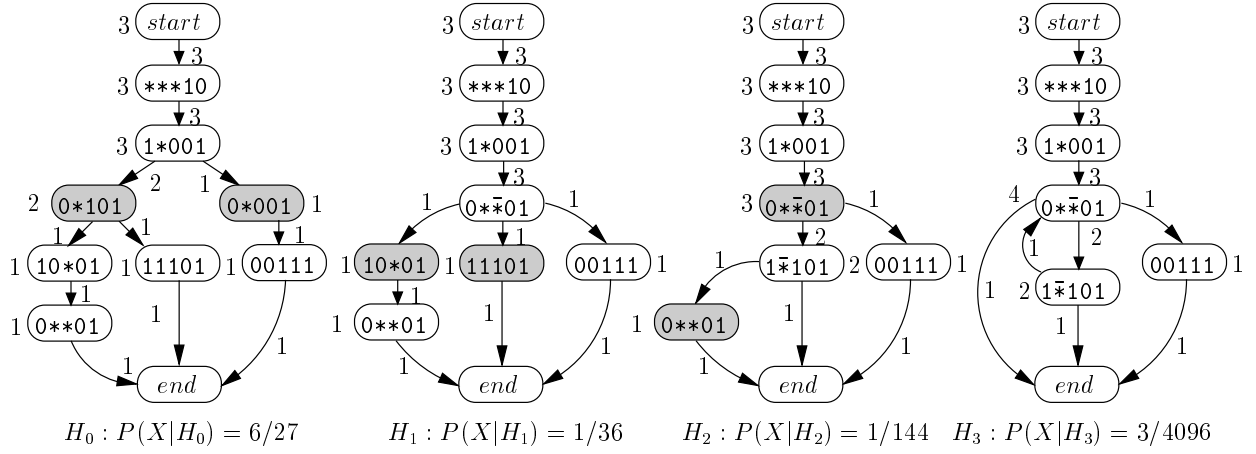


Figure 2: Structure compression achieved by three state mergings. The successive HMMPs are obtained by merging the grey nodes. For each HMMP, we indicate the likelihood of the sample (given in Section 3.3). States and transitions are labeled by the number of times they are used in Viterbi paths, *i.e.* n_s and $n_{s \rightarrow t}$, respectively. The transition probability associated with $s \rightarrow t$ is equal to the ratio $n_{s \rightarrow t}/n_s$ (Formula (7)).

4.2 Building the initial HMMP

The initial HMMP is obtained by building the *prefix tree* of X . In such a tree, each path from the root to a leaf corresponds to a sequence of X , and the common prefixes are not repeated but represented by a unique path starting from the tree root. In our case, the root represents the *start* state. Next, to each state s we attach its emission pattern p_s (except for the *start* with which no emission pattern is associated) and the number of times (n_s) it is used. This number is equal to the number of leaves of the sub-tree with root s . In the same way, to each transition $s \rightarrow t$ we attach the number of times $n_{s \rightarrow t}$ it is run over, *i.e.* the number of leaves of the sub-tree with root t . Finally, we create the *end* state to which every leaf is linked with transition probability 1. The transition probabilities of the initial HMMP are estimated from n_s and $n_{s \rightarrow t}$ parameters using Formula (7). The HMMP obtained is the most specific, as it describes all sequences of X , but only these sequences. Moreover, note that the Viterbi assumption holds, since each sequence of X is generated

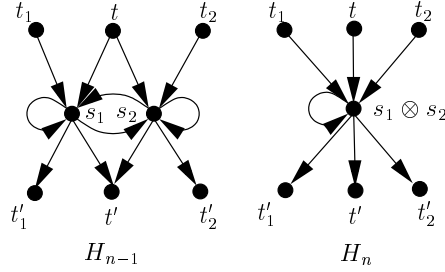


Figure 3: States and transitions before and after merging of s_1 and s_2 .

by a unique path from the root to a leaf.

The HMMP H_0 depicted in Figure 2 is the initial HMMP obtained from the sequence set of Section 3.3. Each sequence has probability $1/3$ of being generated by H_0 . Therefore, $P(X|H_0) = 3! \times 1/3 \times 1/3 \times 1/3 = 6/27$.

4.3 State merging

When a state pair (s_i, s_j) has been selected (the criterion is described in Section 4.4), s_i and s_j are merged. The merging procedure is described by the MERGE algorithm (summarized below), and an example of state merging is provided in Figure 3. This procedure updates the HMMP structure and the parameters of the model. A new state $s_i \otimes s_j$ is created and the *in* and *out* transitions to and from s_i and s_j are connected to $s_i \otimes s_j$. Since this operation has to be made for each merging – and as we will see in Section 4.4 for every state pair considered in the SELECTBESTSTATEPAIR procedure – we need an efficient method to update the model parameters. Fortunately, the Viterbi approximation as used in [10] allows such a procedure. We use the optimistic assumption, which often holds in practice, that state merging does not alter Viterbi paths. Then Viterbi paths are not recomputed using Formula (3), but instead are obtained by uniting the previous paths. This yields the following simple formulae: $n_{s_i \otimes s_j} = n_{s_i} + n_{s_j}$, $n_{t \rightarrow s_i \otimes s_j} = n_{t \rightarrow s_i} + n_{t \rightarrow s_j}$, $n_{s_i \otimes s_j \rightarrow t} = n_{s_i \rightarrow t} + n_{s_j \rightarrow t}$, and $n_{s_i \otimes s_j \rightarrow s_i \otimes s_j} = n_{s_i \rightarrow s_i} + n_{s_i \rightarrow s_j} + n_{s_j \rightarrow s_i} + n_{s_j \rightarrow s_j}$. Note that even when this assumption does not hold, we still maximize an underestimate of the exact

likelihood, which explains the good practical results (see also Section 3.3).

The emission pattern associated with the new state is computed using the γ operator (Table 1). The associativity of γ is relevant: rather than computing $p_{s_i \otimes s_j}$ from the set $P_{X, s_i \otimes s_j} = P_{X, s_i} \cup P_{X, s_j}$, which may yield prohibitive computing time when $P_{X, s_i \otimes s_j}$ contains numerous elements, we compute it more efficiently from the p_{s_i} and p_{s_j} emission patterns. The time complexity of the MERGE procedure is $O(b + k)$, with k being the size of the patterns and b the maximal branching factor of the current HMMP.

Algorithm 2: MERGE

Data : H, s_i, s_j

add a new state $s_i \otimes s_j$ to H ;

$n_{s_i \otimes s_j} \leftarrow n_{s_i} + n_{s_j}$;

$p_{s_i \otimes s_j} \leftarrow \gamma(p_{s_i}, p_{s_j})$;

foreach $t \in In(s_i) \cup In(s_j) - \{s_i, s_j\}$ **do**

add the transition $t \rightarrow s_i \otimes s_j$;

$n_{t \rightarrow s_i \otimes s_j} \leftarrow n_{t \rightarrow s_i} + n_{t \rightarrow s_j}$;

foreach $t \in Out(s_i) \cup Out(s_j) - \{s_i, s_j\}$ **do**

add the transition $s_i \otimes s_j \rightarrow t$;

$n_{s_i \otimes s_j \rightarrow t} \leftarrow n_{s_i \rightarrow t} + n_{s_j \rightarrow t}$;

if $(Out(s_i) \cup Out(s_j)) \cap \{s_i, s_j\} \neq \emptyset$ **then**

add the transition $s_i \otimes s_j \rightarrow s_i \otimes s_j$;

$n_{s_i \otimes s_j \rightarrow s_i \otimes s_j} = n_{s_i \rightarrow s_i} + n_{s_i \rightarrow s_j} + n_{s_j \rightarrow s_i} + n_{s_j \rightarrow s_j}$;

delete s_i and s_j (and their adjacent transitions);

return H ;

4.4 Merging criterion

The aim of the learning algorithm is to reduce the structure of the initial HMMP while keeping the highest possible likelihood. At each step, the algorithm chooses the state pair which, when merged, involves the lowest loss of likelihood. Nevertheless, many pairs

sometimes (especially at beginning of the process) agree with this criterion. Then, from among these pairs we choose that for which emission pattern merging involves fixing the lowest number of bits. The number of bits fixed by merging the two emission patterns p and p' is obtained by

$$\varphi(p, p') = \min(*_{\gamma(p, p')}^p, *_{\gamma(p, p')}^{p'}). \quad (12)$$

For example, if $p = 1*11*$ and $p' = *0\bar{1}***$, then $\gamma(p, p') = 10\bar{1}1*$; we have $*_{\gamma(p, p')}^p = 1$, $*_{\gamma(p, p')}^{p'} = 2$ and then $\varphi(p, p') = 1$. Note that if p is more general than p' , then $\varphi(p, p') = 0$. At the beginning of the learning procedure, using Formula (12) avoids fixing bits in the emission patterns too soon, which would make further mergings more difficult in terms of likelihood. At each step of the algorithm, the selected pair is maximal according to the lexicographic order \succ , defined as:

$(s_i, s_j) \succ (s_k, s_l)$ iff one of the two following clauses is fulfilled:

$$\begin{aligned} P(X|H(s_i \otimes s_j), V) &> P(X|H(s_k \otimes s_l), V), \\ P(X|H(s_i \otimes s_j), V) &= P(X|H(s_k \otimes s_l), V) \text{ and } \varphi(p_{s_i}, p_{s_j}) < \varphi(p_{s_k}, p_{s_l}), \end{aligned}$$

where $H(s_i \otimes s_j)$ denotes the HMMP resulting from the merging of s_i and s_j in H .

4.5 Fast selection of the best state pair

When selecting the best pair relative to \succ , we have to know $P(X|H(s_i \otimes s_j), V)$ and $\varphi(p_{s_i}, p_{s_j})$ for every pair (s_i, s_j) . The value of $\varphi(p_{s_i}, p_{s_j})$ does not change while s_i or s_j are not merged. However, for $P(X|H(s_i \otimes s_j), V)$ this is not the case because its value changes after each merging. Then, we have to compute $P(X|H(s_i \otimes s_j), V)$ $O(|S|^2)$ times at each step, which may yield prohibitive computing time. This difficulty can be overcome by conjointly using the list \mathcal{L} of the state pairs ordered according to \succ (then the best state pair is the first pair of \mathcal{L}) and the criterion $\theta(s_i, s_j)$ defined as

$$\theta(s_i, s_j) = \frac{P(X|H(s_i \otimes s_j), V)}{P(X|H, V)}.$$

This criterion expresses the conservation of likelihood when s_i and s_j are merged. This is a good substitute for $P(X|H(s_i \otimes s_j), V)$ in searching for the best pair, because

$$P(X|H(s_i \otimes s_j), V) > P(X|H(s_k \otimes s_l), V) \iff \theta(s_i, s_j) > \theta(s_k, s_l).$$

Moreover, as we shall see, θ is only altered in some special cases by successive mergings, and it allows the list \mathcal{L} to be rapidly updated.

$P(X|H(s_i \otimes s_j), V)$ and $P(X|H, V)$ are defined by Formula (6), which involves a product of independent factors over the HMMP state set. Between H and $H(s_i \otimes s_j)$, many factors remain identical. This is the case for E_s factors associated with states that differ from s_i , s_j and $s_i \otimes s_j$, and this is also the case for transition probabilities associated with edges non-adjacent to s_i , s_j and $s_i \otimes s_j$. For example, in Figure 3, transition probabilities associated with *out*-transitions from s_1 , s_2 and t are the only ones that change (among all transitions from the current HMMP). Moreover, factors which are not altered by the merging are removed in the ratio $P(X|H(s_i \otimes s_j), V)/P(X|H, V)$. Let $In(s_i, s_j) = In(s_i) \cap In(s_j)$ denotes the *in*-states common to s_i and s_j , and $Out(s_i, s_j) = Out(s_i) \cup Out(s_j)$ be the *out*-states from s_i or s_j (recall that loops are *out*-transitions). Then, the ratio of factors related to transition probabilities that change when merging is

$$T(s_i, s_j) = IT(s_i, s_j) \cdot OT(s_i, s_j), \quad (13)$$

with

$$IT(s_i, s_j) = \frac{\prod_{t \in In(s_i, s_j)} M(t \rightarrow s_i \otimes s_j)^{n_{t \rightarrow s_i \otimes s_j}}}{\left(\prod_{t \in In(s_i, s_j)} M(t \rightarrow s_i)^{n_{t \rightarrow s_i}} M(t \rightarrow s_j)^{n_{t \rightarrow s_j}} \right)},$$

and

$$OT(s_i, s_j) = \frac{\prod_{t \in Out(s_i, s_j)} M(s_i \otimes s_j \rightarrow t)^{n_{s_i \otimes s_j \rightarrow t}}}{\left(\prod_{t \in Out(s_i)} M(s_i \rightarrow t)^{n_{s_i \rightarrow t}} \right) \left(\prod_{t \in Out(s_j)} M(s_j \rightarrow t)^{n_{s_j \rightarrow t}} \right)}.$$

And we have:

$$\theta(s_i, s_j) = T(s_i, s_j) \cdot \frac{E_{s_i \otimes s_j}}{E_{s_i} E_{s_j}}. \quad (14)$$

The first term of Expression (14) depends on the transition probabilities associated with transitions adjacent to s_i , s_j and $s_i \otimes s_j$. The second term depends only on emission patterns associated with s_i , s_j and $s_i \otimes s_j$. Therefore, $\theta(s_i, s_j)$ keeps the same value while no child or parent state of s_i or s_j is merged with another state. So, by using the θ criterion, the list \mathcal{L} can be updated by only computing (and sorting) the performances of pairs which have a state adjacent to the merged pair. Then we only have to deal with $O(|S|b)$ state pairs at each step, with b being the maximal branching factor of the current HMMP.

Let us now describe the time complexity required to compute Formula (14). The $T(s_i, s_j)$ term is computed using Formula (13), that ran over transitions adjacent to s_i and s_j , and its time complexity is $O(b)$. For E_s terms, at the beginning of the learning procedure, we have $E_s = 1 \forall s \in S$ since emission patterns associated with states have not yet been merged. Next, the $E_{s_i \otimes s_j}$ terms can be computed using Formula (9), but this computation may be too time-consuming when $P_{X, s_i \otimes s_j}$ contains numerous patterns. A better approach involves using the $n_{\neg*}(P_{X, s}, r)$ defined in Equation (8). Indeed, if s_i and s_j are merged to produce the state $s_i \otimes s_j$, we obviously have:

$$n_{\neg*}(P_{X, s_i \otimes s_j}, r) = n_{\neg*}(P_{X, s_i}, r) + n_{\neg*}(P_{X, s_j}, r), \quad (15)$$

and this equality allows $E_{s_i \otimes s_j}$ to be computed in $O(k)$ using Formula (10). At the beginning of the procedure, $P_{X, s}$ is reduced to the singleton $\{p_s\}$, and the $n_{\neg*}(P_{X, s}, r)$ terms are initialized as:

$$n_{\neg*}(P_{X, s}, r) = n_s \cdot \delta_{\neg*}(p_s, r). \quad (16)$$

To create the list \mathcal{L} , for each state s of H_0 , we have to compute and store every $n_{\neg*}(P_{X, s}, r)$ using Formula (16) and initialize E_s to 1. Then we consider all possible ($O(|S|^2)$) state pairs; for every pair (s_i, s_j) we compute $p_{s_i \otimes s_j}$, $n_{\neg*}(P_{X, s_i \otimes s_j}, r)$, $E_{s_i \otimes s_j}$ and $T(s_i, s_j)$ and finally $\theta(s_i, s_j)$ and $\varphi(p_{s_i}, p_{s_j})$; this is done in $O(b+k)$ (with b being the maximal branching factor of H_0). Inserting a pair in \mathcal{L} is done in $O(\log(|S|))$. Then the total time complexity of the \mathcal{L} creating procedure is $O(|S|^2(b+k+\log(|S|)))$. The procedure for

updating \mathcal{L} is very similar, but involves $b|S|$ pairs instead of $|S|^2$, so its time complexity is $O(|S|b(b + k + \log(|S|)))$.

4.6 Time complexity of the whole learning procedure

The time complexity of the list creating procedure is $O(|S|^2(b + k + \log(|S|)))$, with k the size of the patterns and b the maximal branching factor of the initial HMMP. The time complexity of the updating procedure is $O(|S|b(b + k + \log(|S|)))$, and merging two states requires $O(b + k)$, with b now being the branching factor of the current HMMP. Both latter procedures have to be executed at each step of the learning procedure, *i.e.* $O(|S|)$ times. Therefore, the total time complexity strongly depends on the evolution of the maximal branching factor b . Fortunately, b remains relatively low in practice. This is essentially due to the pair selection criterion that tends to keep the likelihood of the training sequences high. Indeed, the more the states have *out*-transitions (the higher b), the lower the likelihood. Consider Formula (6). The larger $|Out(s)|$, the lower $\prod_{t \in Out(s)} M(s \rightarrow t)^{n_{s \rightarrow t}}$. For example, with $|Out(s)| = 1$ we have $M(s \rightarrow t) = 1$ and $\prod_{t \in Out(s)} M(s \rightarrow t)^{n_{s \rightarrow t}} = 1$, while with $|Out(s)| = 2$ and $n_{s \rightarrow t_1} = n_{s \rightarrow t_2}$, we have $M(s \rightarrow t_1) = M(s \rightarrow t_2) = 1/2$ and $\prod_{t \in Out(s)} M(s \rightarrow t)^{n_{s \rightarrow t}} = (1/2)^{n_s}$.

Experiments confirm this analysis. Figure 4 shows the evolution of the maximal (mb) and average (ab) branching factor during the learning procedure for the benchmark circuit *s3330* (c.f. Section 5). The number of states in the initial HMMP is 2176, while mb is equal to 260 and ab to 2.2; mb rapidly decreases until about 50, while ab slowly increases but remains lower than 25; throughout the procedure, we clearly have $b \ll |S|$. Therefore, the worst case analysis (mb) as well as the average analysis (ab) confirm the advantage of our approach, which replaces the $|S|^3$ factor of the naive algorithm (computing the performance of every pair at each step) by a factor $|S|^2b$. The CPU learning time with this circuit is about 40 minutes (*PC Pentium III 350 MHz*), while the naive algorithm requires about 40 hours.

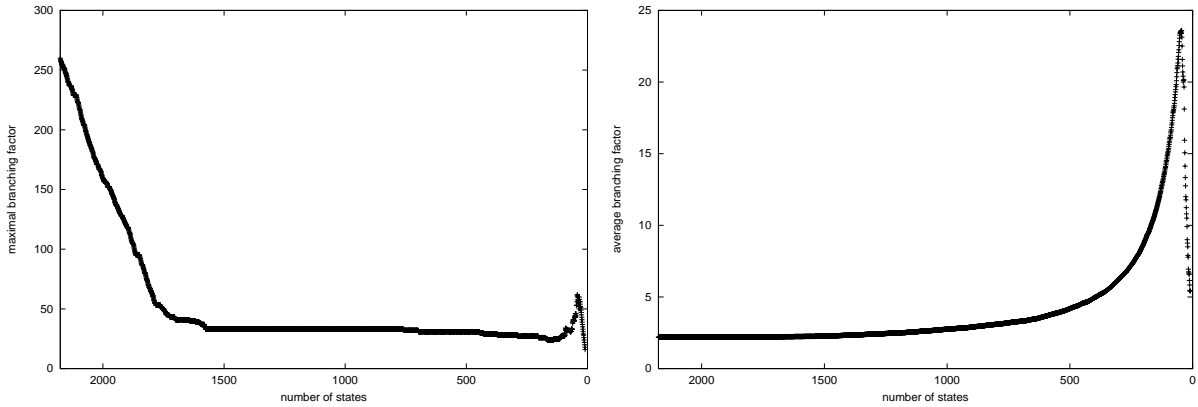


Figure 4: Evolution of the maximal (left) and average (right) branching factor.

5 Experimental results

5.1 Experiments with microelectronic benchmarks

After the learning phase, the HMMP obtained can be physically implemented as a test sequence generator using a natural microelectronic translation [23]. The size of the implementation, a crucial factor for real applications, is closely connected to the size of the HMMP in terms of number of states and transitions.

We tested our method with classical benchmarks of the test community [25]. The results are reported in Table 1:

- The first three columns *Name*, *#I* and *#F* provide the name, the number of inputs and the number of potential faults of the tested circuit, respectively.
- The following two columns *%Cov* and *Leng* indicate the fault coverage and total length of ATPG sequences.
- For each circuit, we simulated 10 sets of sequences generated with the learned HMMP. For comparison purposes, we also simulated 10 long random sequences for every circuit. Only simulating one long sequence is justified here, due to the randomness of

the approach, and no improvement is obtained by decomposing this long sequence into many shorter ones. The *T.Leng* column provides the total length of HMMP sequences; the same length was used for random sequences. When applying sequences generated by a stochastic model such as an HMMP or a fully random process, numerous (easy) faults are detected at the beginning. The frequency of new detected faults then decreases and no new faults seem to be detectable after a certain time. So the total length was manually tuned and limited to the efficient beginning part of HMMP sequences. The following column (*ratio*) provides the ratio of this length over the total length of ATPG sequences.

- The *%Best* columns indicate the best fault coverage achieved in the 10 simulations; the *%Av* columns indicate the average of these 10 fault coverages.
- Columns *#S.* and *#E.* provide the number of states and the number of transitions of the learned HMMP, respectively. As stated in Section 4, the number of states of the HMMPs was manually tuned according to the size and fault coverage constraints.
- On the bottom line, we report the means of all of these quantities.

The fault coverages achieved by our method are much larger than those of the random sequences. They are often equal (s298, s1494), sometimes slightly smaller (s820, s832) and sometimes larger (s444, s526) than the fault coverages of ATPG sequences. On average, our method provides better fault coverage than ATPG sequences, at the expense of larger (about 5 times on average) test sequence sets. This result is surprising and confirms the validity of our approach. It demonstrates that it is possible to infer very efficient construction rules from ATPG sequences. These rules do not ensure accurate generation of the original sequences, but they generate similar sequences that achieve high fault coverage when the sequence set is large enough. The good results obtained with our method could also be explained by the weakness of some ATPG sequence sets (and by the NP-hardness of the task), and by the easiness of achieving relatively high fault coverages for some circuits

CIRCUIT			ATPG				RANDOM		HMMP			
Name	#I	#F	%Cov	Leng	T.Leng	ratio	%Best	%Av	%Best	%Av	#S	#E
s298	3	596	89.9	2408	2.5K	1.03	75.2	68.1	89.9	89.3	5	19
s344	9	670	97.0	427	1K	2.34	96.1	94.2	97.6	97.6	3	7
s382	3	764	85.7	9178	15K	1.63	15.4	15.4	96.7	96.3	5	13
s386	7	772	90.2	754	5K	6.63	74.00	68.8	89.6	89.3	7	20
s444	3	888	75.5	2074	10K	4.82	13.4	13.3	97.1	92.5	4	9
s526	3	1052	52.9	966	10K	10.35	10.8	10.7	82.6	76.6	5	10
s820	18	1640	96.3	4993	30K	6.00	49.4	48.4	94.2	91.8	13	40
s832	18	1664	95.3	5024	30K	5.97	47.3	46.3	93.6	90.6	13	43
s991	65	1948	99.2	1139	6K	5.27	93.8	93.6	96.9	96.6	8	25
s1488	8	2976	95.6	6776	30K	4.42	78.0	75.1	98.6	98.5	8	30
s1494	8	2988	98.1	6723	30K	4.46	78.3	75.1	98.1	98.0	8	29
s3330	40	6660	79.2	5616	30K	5.34	76.5	74.4	78.5	78.1	13	46
Avg:			87.9			4.85	59.0	56.9	92.8	91.3		

Table 2: Fault coverage achieved by ATPG, random sequences and HMMPs.

(see results obtained by the random method). Note also that these fault coverages are obtained with small HMMPs: the maximum number of states is 13, and the number of transitions is low relative to the number of states. This allows HMMPs to be electronically implemented using a reasonable silicon area, which makes HMMPs a suitable solution for the BIST problem [23].

5.2 Comparison with other approaches

As stated in Section 3, emission patterns define very simple probability distributions over $\{0, 1\}^k$. Due to the size constraint, this simplicity is required for microelectronic purposes. However, an issue concerns the effect of this choice on the model accuracy as compared to most conventional distributions such as products of Bernoulli distributions. Another question is the relevance of using a learning algorithm by state merging rather than the classical Baum-Welch algorithm. To answer these questions, another set of experiments summarized

Circuit Name	Bernoulli K -M.		Bernoulli merg.		HMMP K -M.		HMMP merg.	
	F.C.	log lik	F.C.	log lik	F.C.	log lik	F.C.	log lik
s344	97.6%	270	97.6%	287	97.0%	367	97.6%	321
s526	84.3%	905	83.7%	985	12.6%	2117	82.6%	1363
s820	93.4%	3061	95.5%	2281	57.6%	6858	94.2%	2671
s3330	80.3%	6337	80.0%	6931	78.4%	8639	78.5%	7990
Avg. :	88.9%	2643	89.2%	2621	61.4%	4495	88.2%	3086

Table 3: Negative log likelihood of the ATPG sequences and fault coverages achieved by HMMPs and HMMs with Bernoulli distributions trained by the segmental K -Means algorithm and a state merging algorithm.

in Table 3 was carried out for some benchmark circuits. First we trained HMMPs with a variant of the Baum-Welch algorithm known as the segmental K -Means method [26] (column HMMP K -M.). Secondly, we trained HMMs with products of Bernoulli distributions associated with the states. These HMMs were trained in two different ways: using a learning algorithm by state merging as for HMMPs (column Bernoulli merg.) and using the segmental K -Means algorithm (column Bernoulli K -M.). For each of these experiments, the numbers of states of HMMs and HMMPs and the total length of the test sequences were set at the same value as those in Table 2. The initial models trained with the segmental K -Means algorithm were fully connected, and the emission patterns of the initial HMMPs were set at $*^k$ to avoid null likelihood. For each circuit, ten models with different initial parameters were trained, and we kept the model maximizing the likelihood of ATPG sequences. Table 3 includes both the negative log likelihood of the ATPG sequences and the fault coverage achieved by the models.

First we note that, as suspected, the non-continuous nature of HMMP emission probabilities often lowers the accuracy of the classical training algorithm, both for the likelihood of the ATPG sequences and the fault coverage of the learned HMMPs. The obtained fault coverages are then close to those of random sequences (see Table 2). In contrast, the three other approaches provide models that achieve high fault coverages, which are very close

although the likelihoods differ significantly. These results indicate that the merging approach is well suited for HMMP (and HMM), and that HMMP performance for BIST is not very affected by the high simplicity of emission distributions.

6 Conclusion

We presented a new model for learning Boolean vector sequences. This model is close to the classical HMM, but differs in that it defines emission probability distributions with Boolean patterns. We also presented an efficient algorithm to learn HMMPs, which uses the state merging principle. We used HMMP and our algorithm to build test sequence generators for integrated circuits, and observed that, despite their reduced size and simplicity, inferred HMMPs achieve very high coverage of potential faults of circuits at hand. The simplicity of HMMPs could also be relevant in other frameworks. Boolean patterns are very similar to the condition parts of the rules used in symbolic and hybrid classification methods [27], and, just as in these methods, this simplicity is associated with explanatory virtues which should be of interest from a modeling and learning perspective. Moreover, the algorithmic refinements we proposed for the merging algorithm (Section 4.5), which yield a great reduction of the computing time, could be fruitfully used for other models. However, improvements could be done concerning the learning procedure. First, all the ATPG sequences do not have the same testing capabilities, and it could be of interest in the learning phase to weight the sequences by the number of faults they detect in order to improve the fault coverages achieved by the learned HMMPs. Adapted techniques, such as a preliminary clustering step of the patterns of the ATPG sequences or of the bit positions within these patterns, could also enhance the efficiency and accuracy of the learning procedure.

Acknowledgements

We thank Patrick Girard and Christian Landrault of the Microelectronics Department

of the LIRMM for presenting and explaining to us the problem of Built-in Self-Test for sequential integrated circuits, and for their help at the start of this work and throughout the experimental evaluation process.

Annex : Subset (14 among 284) of the test sequence set obtained using the ATPG *Sunrise* for the circuit *s820*.

```

Sequence 1
*****0
*010*****0001
****1*****0001

Sequence 2
*****0
*010*****0001
****0*****1***11001
*****01001
****1*****01001
*1**1*****01001
*010*****0001
*0*1*****010001
****0*1001***11001
****0*****01001
****0*****00***11001
****0*****0001
****0*****001
*0*0*1*****001

Sequence 3
*****0
*010*****0001
****0*****1***11001
*****01001
****1*****01001
*1**1*****01001
*010*****0001
*0*1*****010001
****0*1001***11001
****0*****011001
01**0*****010**11001
****0*****0*11001
0**0*****0001

Sequence 4
*****0
*010*****0001
****0*****1***11001
*****01001
****1*****01001
*1**1*****01001
*010*****0001
*0*1*****010001
****0*1001***11001
****0*****011001
****0*****00***11001
****0*****0001
****0*****0001
****0*****001
*000*****001

Sequence 5
*****0
*010*****0001
****0*****1***11001
*****01001
****1*****01001
*1**1*****01001
*010*****0001
*0*1*****010001
****0*1001***11001
****0*****011001
****0*****00***11001
****0*****0001
****0*****001
*001*****001

Sequence 6
*****0
*010*****0001
****0*****1***11001
*****01001
****1*****01001
*1**1*****01001
*010*****0001
*0*1*****010001
****0*1001***11001
****0*****011001
****0*****00***11001
****0*****0001
****0*****001
*010*****001

Sequence 7
*****0
*010*****0001
****0*****1***11001
*****01001
****1*****01001
*1**1*****01001
*010*****0001
*0*1*****010001
****0*1001***11001
****0*****011001
****0*****00***11001
****0*****0001
****0*****001
*100*****001
0*****0001

Sequence 8
*****0
*010*****0001
****0*****1***11001
*****01001
****1*****01001
*1**1*****01001
*010*****0001
*0*1*****010001
****0*1001***11001
****0*****011001
****0*****00***11001

Sequence 9
*****0
*010*****0001
****0*****1***11001
*****01001
****1*****01001
*1**1*****01001
*010*****0001
*0*1*****010001
****0*1001***11001
****0*****011001
****0*****00***11001

Sequence 10
*****0
*010*****0001
****0*****1***11001
*****01001
****1*****01001
*1**1*****01001
*010*****0001
*0*1*****010001
****0*1001***11001
****0*****011001
****0***1*00*1*11001

Sequence 11
*****0
*010*****0001
****0*****1***11001
*****01001
****1*****01001
*1**1*****01001
*010*****0001
*0*1*****010001
****0*1001***11001
****0*****011001
****0*****1*1**11001
1***1*****0*****1001

Sequence 12
*****0
*010*****0001
****0*****1***11001
*****01001
****1*****01001
*1**1*****01001
*010*****0001
*0*1*****010001
****0*1001***11001
****0*****011001
****0*****1*1**11001
****0*****1*01001

Sequence 13
*****0
*010*****0001
****0*****1***11001
*****01001
****1*****01001
*1**1*****01001
*010*****0001
*0*1*****010001
****0*1001***11001
****0*****011001
01**0*****010**11001
****0*****1*01001

Sequence 14
*****0
*010*****0001
****0*****1***11001
*0*****01001
****1*****01001
*1**1*****01001
*010*****0001
*0*1*****010001
****0*1001***11001
****0*****011001
****0*****1*1**11001
11*****1001

```

References

- [1] L. E. Baum, T. Petrie, G. Soules, and N. Weiss, “A maximization technique occurring in statistical analysis of probabilistic functions in Markov chains,” *The Annals of Mathematical Statistics*, vol. 41, no. 1, pp. 164–171, 1970.
- [2] F. Casacuberta, “Some relations among stochastic finite state networks used in automatic speech recognition,” *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 12, pp. 691–695, July 1990.
- [3] L. R. Bahl, F. Jelinek, and R. L. Mercer, “A maximum likelihood approach to continuous speech recognition,” *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 5, pp. 179–190, Mar. 1983.
- [4] L. E. Baum, “An inequality and associated maximization technique in statistical estimation for probabilistic functions of a Markov process,” *Inequalities*, vol. 3, pp. 1–8, 1972.
- [5] A. P. Dempster, N. M. Laird, and D. B. Rubin, “Maximum likelihood from incomplete data via the EM algorithm.,” *J. Royal Stat. Soc. B*, vol. 39, pp. 1–38, 1977.
- [6] N. Abe and M. Warmuth, “On the computational complexity of approximating distributions by probabilistic automata,” *Machine Learning*, vol. 9(2-3), pp. 205–260, 1992.
- [7] J. Hopcroft and J. D. Ullman, *Introduction to Automata Theory, Language, and Computation*. 1979.
- [8] J. Oncina and P. Garcia, “Inferring regular languages in polynomial updated time.,” *Pattern Recognition and Image Analysis*, pp. 49–61, 1992.

- [9] R. C. Carrasco and J. Oncina, “Learning stochastic regular grammars by means of a state merging method,” in *Proceedings of the Second International ICGI Colloquium on Grammatical Inference and Applications*, vol. 862 of *LNAI*, pp. 139–152, 1994.
- [10] A. Stolcke and S. Omohundro, “Inducing probabilistic grammars by bayesian model merging,” in *Proceedings of the ICGI*, vol. 862 of *LNAI*, (Berlin), pp. 106–118, 1994.
- [11] K. J. Lang, B. A. Pearlmutter, and R. A. Price, “Results of the Abbadingo One DFA learning competition and new evidence-driven state merging algorithm,” in *Proceedings of the 4th International Colloquium on Grammatical Inference (ICGI-98)*, pp. 1–12, July 1998.
- [12] L. R. Rabiner, “A tutorial on hidden Markov models and selected applications in speech recognition,” *Proceedings of the IEEE*, vol. 77, no. 2, pp. 257–285, 1989.
- [13] A. Kundu, Y. He, and P. Bahl, “Recognition of handwritten word: First and second order Hidden Markov Model based approach,” in *CVPR’88 (IEEE Computer Society Conference on Computer Vision and Pattern Recognition, Ann Arbor, MI, June 5–9, 1988)*, pp. 457–462, 1988.
- [14] M. Chen, A. Kundu, and J. Zhou, “Off-line handwritten word recognition using a hidden Markov model type stochastic network,” *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 16, pp. 485–496, May 1994.
- [15] R. Durbin, S. Eddy, A. Krogh, and G. Mitchison, *Biological sequence analysis probabilistic models of proteins and nucleic acids*. Cambridge University Press, 1998.
- [16] B. Povlow and S. Dunn, “Texture classification using noncausal hidden Markov models,” *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 17, pp. 1010–1014, Oct. 1995.

- [17] C. Raphael, "Automatic segmentation of acoustic musical signals using hidden Markov models," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 21, no. 4, pp. 360–370, 1998.
- [18] J. Rajski and J. Tyszer, *Arithmetic Built-In Self-Test*. Prentice Hall PTR, 1998.
- [19] Ibarra and Sahni, "Polynomially complete fault detection problems," *IEEE Transactions on Computers*, vol. 24, 1975.
- [20] F. Corno, P. Prinetto, M. Rebaudengo, and M. Sonza Reorda, "A parallel genetic algorithm for automatic generation of test sequences for digital circuits," *Lecture Notes in Computer Science*, vol. 1067, pp. 454–??, 1996.
- [21] A. Ghosh, S. Devadas, and A. Newton, *Sequential Logic Testing and Verification*. 1992.
- [22] V. D. Agrawal, K. T. Cheng, and P. Agrawal, "A directed search method for test generation using a concurrent simulator," *IEEE Trans. on Computer-Aided Design*, vol. 8, pp. 131–138, Feb. 1989.
- [23] L. Bréhélin, O. Gascuel, G. Caraux, P. Girard, and C. Landrault, "Hidden Markov and Independence Models with Patterns for sequential BIST," in *18th IEEE VLSI Test Symposium*, 2000.
- [24] G. D. Forney, "The Viterbi algorithm," *Proc. IEEE*, vol. 61, pp. 268–278, Mar. 1973.
- [25] F. Brglez, D. Bryan, and K. Kozminski, "Combinational profiles of sequential benchmark circuits," in *International Symposium on Circuits and Systems*, May 1989.
- [26] B. H. Juang and L. R. Rabiner, "The segmental K-means algorithm for estimating parameters of hidden markov models," *IEEE Transactions on Acoustics, Speech, and Signal Processing*, vol. 38, no. 9, p. 1639, 1990.

- [27] O. Gascuel and the SYMENU group, “Twelve numerical, symbolic and hybrid supervised classification methods,” *International Journal of Pattern Recognition and Artificial Intelligence*, pp. 517–572, 1998.