Hidden Markov Models with Patterns and Their Application to Integrated Circuit Testing

Laurent Bréhélin, Olivier Gascuel, and Gilles Caraux

LIRMM, Université Montpellier II
161 rue Ada, 34392 Montpellier Cedex 5, France
{brehelin,gascuel,caraux}@lirmm.fr

Abstract. We present a new model, derived from classical Hidden Markov Models (HMMs), to learn sequences of *large* Boolean vectors. Our model – *Hidden Markov Model with Patterns*, or *HMMP* – differs from HMM by the fact that it uses patterns to define the emission probability distributions attached to the states. We also present an efficient state merging algorithm to learn this model from training vector sequences. This model and our algorithm are applied to learn Boolean vector sequences used to test integrated circuits. The learned HMMPs are used as test sequence generators. They achieve very high fault coverage, despite their reduced size, which demonstrates the effectiveness of our approach.

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 letter from a finite alphabet that is generated each time the transition is ran over. An HMM is also defined by its structure, composed of states and transitions, and by probability distributions over the transitions. The difference with respect to PAs is that the letter generation is attached to the states. Each state is associated with a probability distribution over the alphabet that expresses the probability for each letter to be generated when the state is encountered.

When the structure is known, the HMM learning (or training) problem is reduced to estimating the value of its parameters – transition and generation probabilities – from a sample of sequences. A well-known approach is the *Baum-Welch* algorithm [3], which complies with the maximum likelihood principle and is a special case of the *Expectation-Maximization* (*EM*) algorithm [4]. This is an iterative re-estimation algorithm that ensures convergence to a local optimum. Abe and Warmuth [5] 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, for the HMMs the question remains open. However, we can reasonably assume that the problem is not easier, and that heuristics have to be used.

R. López de Mántaras, E. Plaza (Eds.): ECML 2000, LNAI 1810, pp. 75–87, 2000.
© Springer-Verlag Berlin Heidelberg 2000

In many applications, it is not possible to infer the structure of the HMM from the *a priori* knowledge we have about the problem under investigation. In this case, the HMM learning problem becomes even more difficult. We have to estimate the parameters of the structure, and also infer this structure from the learning sample. Various authors have proposed a heuristic approach derived from automata theory. It involves generalizing an initial specific automaton that accurately 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 successfully applied to non-probabilistic automata [6] as well as to probabilistic ones [7], and to HMMs [8].

HMMs have been used as models for sequences from various domains, such as speech signals (e.g. [9]), handwritten text (e.g. [10]) and biological sequences (e.g. [11]). 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 with fixed size, and they are trained using the Baum-Welch method. Moreover, these applications all learn and use HMMs for recognition purposes. In this article, we present a new application: the *Built-in Self Test for integrated circuits*. This application differs in that we use HMMs for generation purposes: an HMM is learned from a sample of sequences, which involves building a convenient structure and estimating its parameters. This HMM is then used to generate sequences similar (eventually identical) to the learning sequences. Another difference is that the alphabet in this application can be extremely large, e.g. 2^{100}, so the emission probabilities cannot be easily defined in the usual way. This led us to develop a new class of HMM that we called *HMMP* (*Hidden Markov Model with Patterns*).

The organization of this paper is as follows. In Section 2, we present the integrated circuit test; 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 in Section 4 we present an HMMP learning algorithm which uses the state merging principle. Section 5 provides experimental results of our method with classical benchmark circuits of the test community.

2 Integrated Circuit Testing

An integrated circuit manipulates Boolean values (`0` or `1`). It is made of *inputs*, *outputs* and internal elements that compose the body of the circuit. It is possible to apply values to the inputs and read the output values, but it is not possible to access the internal elements. These elements may be affected by various physical *faults*. We can infer the set of potential faults of a circuit because we know its logic structure. The test of a potential fault of a given circuit is achieved by using an appropriate sequence – or *test sequence* – of Boolean vectors. The vectors are sequentially applied to the inputs; when the outputs are identical to those logically expected the fault is not present, and conversely, when the fault is present the outputs are erroneous. In a test sequence, the vector application order is as important as the vectors themselves. One sequence can test several

faults, and one fault may be tested by several sequences. Note that the sequence length varies according to the faults (between two and hundreds of vectors) and that we can deduce, by simulation, all the faults detected by a given sequence.

The research of a test sequence, for a circuit and a given fault, is NP-hard [12]. *Automatic Test Pattern Generators* (*ATPGs*) try to circumvent the difficulty by using various heuristics, and usually provide satisfying results. We shall see in Section 5 that the fault coverage – the proportion of faults for which the ATPG finds a test sequence – is usually above 80%.

ATPGs provide sequences of patterns $\in \{0, 1, *\}^k$ (where k is the number of inputs) rather than sequences of vectors. The * character means that the bit value is unimportant, 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 2^n vectors. In the same way, a sequence of patterns defines a set of vector sequences. We present below an example of three test sequences similar to those generated by an ATPG.

Sequence 1	Sequence 2	Sequence 3
****0	****0	****0
1*001	1*001	1**01
0*101	0*101	0*001
10*01	11*01	1*101
0**01		

When test sequences with high fault coverage have been obtained, 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 circuit, we often prefer the method of *built-in self test* or *BIST*. The BIST principle is to incorporate a supplementary test structure in the circuit. This structure should be able to generate test sequences and analyse the circuit responses. Response analysis is a task that has efficient solutions. 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). Indeed, we can not physically stock all the ATPG sequences on ground of silicon cost. On the other side, we can build small generators of pseudo-random sequences, but they have low fault coverage.

Our approach consists of building, with a learning algorithm, 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 relatively small HMMPs effectively generate sequences with fault coverage equivalent to that of the ATPG.

3 The HMMP Class

The HMMs that we want to infer are intended to generate Boolean vector sequences. Then the size of the manipulated alphabet is 2^k (for our test problem, k is the number of inputs of the circuit), which is potentially very large (e.g., there

are circuits with $k > 100$). Common HMMs do not fit the modeling of such sequences. We define here a new class of HMMs, named *Hidden Markov Model with Patterns* (or *HMMPs*), specially adapted to deal with this problem. Owing to the specificity of the test sequences (we learn from patterns, i.e. from sets of vectors and not from simple vectors; moreover, we perform generation and not recognition), we use symbols and operators specific to these data and to the problem. However, HMMPs can also be used to model more conventional vector sequences (Boolean or not). This point is discussed in Section 6.

3.1 Presentation

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 a pattern from P.
- $P = \{p_s, s \in S - \{start, end\}\}$ is the set of patterns associated with the states; p_s is the pattern associated with state s.
- $M : S - \{end\} \times S - \{start\} \to [0, 1]$ is the matrix that contains transition probabilities between states. M defines the probability distributions associated with states of H. We have: $\forall s, t,\ M(s \to t) \geq 0$, and $\forall s,\ \sum_{t \in S} M(s \to t) = 1$.

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 six states and seven transitions.

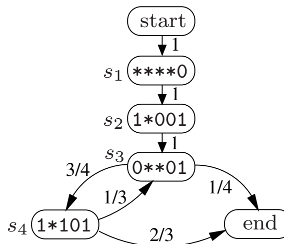

Fig. 1. Example of HMMP: each state is labelled by its name and associated pattern; transitions are labeled by their transition probability

3.2 Generating Vector Sequences with an HMMP

The procedure involves beginning on the *start* state, running over the transitions and generating a test vector for each state encountered by using the pattern associated with that state. The test vectors generated by a given pattern are consistent with this pattern and, moreover, the `*` has equal probability of generating a `0` and a `1`. For example, pattern `*1*` generates, with probability 1/4, each of

the 4 vectors 010, 011, 110 and 111. Once a test vector has been generated, we choose, according to the associated probabilities, one transition starting from the current state, and then go to the targeted state. This procedure is continued until the *end* is reached. A sequence of vectors is thus generated, and another sequence can eventually be generated by going onto the *start* again.

3.3 Generation Probability of a Set of Sequences

The pattern p is said to be *compatible* with the pattern p_s if its fixed bits (those with value 0 or 1) have the same value or the value * in p_s. For example, $p = \texttt{11*}$ is compatible with $p_s = \texttt{1*0}$, while $p = \texttt{11*}$ is not compatible with $p_s = \texttt{100}$. The probability is zero that the state s will generate a pattern p that is not compatible with p_s. The generation probability by a state s of a pattern p compatible with p_s depends on the number of bits which are fixed in p but have the value * in p_s. Let $*^p_{p_s}$ denote this number. For example, if $p = \texttt{10**}$ and $p_s = \texttt{1***}$ then $*^p_{p_s} = 1$. Since * has equiprobability of generating a 0 or a 1, the probability of generating the compatible pattern p on state s is given by the formula:

$$P(p|s) = \left(\frac{1}{2}\right)^{*^p_{p_s}} . \tag{1}$$

For example, if $p = \texttt{1*01*}$ and $p_s = \texttt{1*010}$, then $P(p|s) = 1$. On the other hand, if $p = \texttt{1*010*}$ and $p_s = \texttt{***10*}$, then $P(p|s) = \frac{1}{4}$.

Let $x = p_1 p_2 \ldots p_l$ be a sequence of patterns. A common method for computing the generation probability of x by an HMM H is to make the *Viterbi* assumption [13] that x can only be generated by a unique path (or sequence of states) through H. In other words, all paths except the most likely are assumed to have a negligible (or null) probability of generating x. This path is called the *Viterbi path* of x. For example, the Viterbi path of the first sequence of Section 2 in the HMMP of 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 $V_x = v_{p_0} \cdots v_{p_{l+1}}$ (with $v_{p_0} = start$ and $v_{p_{l+1}} = end$) be the Viterbi path of the sequence $x = p_1 \cdots p_l$. Then, under the Viterbi assumption, the generation probability of x by H is:

$$P(x|H) = \left(\prod_{i=0}^{l-1} M(v_{p_i} \rightarrow v_{p_{i+1}}) P(p_{i+1}|v_{p_{i+1}}) \right) M(v_{p_l} \rightarrow v_{p_{l+1}}).$$

For the above sequence, we have: $M(start \rightarrow s_1) = 1$, $P(p_1|s_1) = 1$, $M(s_1 \rightarrow s_2) = 1$, $P(p_2|s_2) = 1$, $M(s_2 \rightarrow s_3) = 1$, $P(p_3|s_3) = 1/2$, $M(s_3 \rightarrow s_4) = 3/4$, $P(p_4|s_4) = 1/2$, $M(s_4 \rightarrow s_3) = 1/3$, $P(p_5|s_3) = 1$, $M(s_3 \rightarrow end) = 1/4$. It follows that the probability of HMMP of Figure 1 generating this sequence is equal to: $1/2 \times 3/4 \times 1/2 \times 1/3 \times 1/4 = 1/64$.

Let X be a set of sequences and $V = \{V_x, x \in X\}$ the set of Viterbi paths associated with the sequences of X. The probability $P(X|H)$ of generating, with $|X|$ trials, the set X using H, is obtained (under the same assumption) by the following formula:

$$P(X|H) = |X|! \prod_{x \in X} P(x|H),$$

that can be rewritten as:

$$P(X|H) = |X|! \prod_{s \in S} \left(\prod_{p \in P_{X,s}} P(p|s)^{n_s^p} \prod_{t \in Out(s)} M(s \to t)^{n_{s \to t}} \right), \tag{2}$$

where $P_{X,s}$ is the set of patterns generated by s (in V), n_s^p is the number of times s generates the pattern p ($\in P_{X,s}$), $n_{s \to t}$ is the number of times the transition $s \to t$ is used, and $Out(s)$ is the set of states t for which there is a transition $s \to t$.

4 Learning HMMPs

Let X be a set of pattern sequences (for example obtained from an ATPG). Our aim is to build an HMMP of low size (i.e. with a low number of states and transitions), and that generates X with probability as high as possible. We designed a learning algorithm for this purpose which is based on the state merging generalization method.

4.1 Main Algorithm

The main algorithm – HmmpLearning – of the learning procedure proceeds in a greedy ascending way. First, it builds with the InitialHmmpBuilding procedure (Line 1) an initial specific HMMP that represents the sequences of X. Next, at each step of the algorithm, the BestStatePair procedure (Line 2) selects the state pair that, when merged, involves the lowest loss of probability of generating X. If several pairs have the same probability loss, it chooses the pair that involves the nearest patterns. The selected state pair is merged with the Merge procedure (Line 3), which modifies the structure of the HMMP and updates its parameters. The algorithm iterates this procedure until the desired number of states (N) is reached.

Figure 2 details six steps of the algorithm when applied to the three sequences of Section 2.

4.2 Building the Initial HMMP

The initial HMMP H_0 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

Algorithm 1: HmmpLearning(X,N)

```
   Data   : X,N
   Result : H
1  H ← InitialHmmpBuilding(X);
   while Number of states of H > N do
2  |  (s1,s2) ← BestStatePair(H);
3  |_ H ← Merge(H,s1,s2);
   return H;
```

from the tree root. In our case, the root represents the *start* state. Next, to each state s we attach its pattern p_s (except for the *start* with which no pattern is associated) and the number of times (n_s) it is used in the Viterbi paths. In H_0, Viterbi paths are naturally described by sequences and the Viterbi assumption holds. Therefore, n_s is equal to the number of leaves of the sub-tree with root s, and, in the same way, $n_{s\to t}$ is equal to the number of leaves of the sub-tree with root t. Values of both parameters (n_s and $n_{s\to t}$) are stored. Moreover, we set $P_{X,s} = \{p_s\}$ and $n_s^{p_s} = n_s$.

According to the maximum likelihood principle, the values of the transition probabilities associated with the edges of H_0 are estimated by maximizing $P(X|H_0)$. Maximizing Expression (2) is equivalent to maximizing each of its sub-products. Therefore, we estimate the transition probabilities by maximizing the expressions $\prod_{t\in Out(s)} M(s\to t)^{n_{s\to t}}$. Each of these expressions is identical to the probability distribution of a multinomial law and is maximized by

$$M(s\to t) = \frac{n_{s\to t}}{n_s}. \tag{3}$$

Finally, we create the *end* state to which every leave is linked with transition probability 1. The HMMP obtained is the most specific, in that it describes all sequences of X, but only these sequences.

The HMMP H_0 of Figure 2 is the initial HMMP obtained from the sequence set of Section 2. 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 two states s_1 and s_2 have been selected (the criterion used is described in Section 4.4), they are merged. States s_1 and s_2 are deleted and replaced by a new state s. The *in* and *out* edges from s_1 and s_2 are connected to s, and the potential double transitions (e.g., $t\to s_1$ and $t\to s_2$) are also merged. An example of state merging is provided in Figure 3.

The structure of the HMMP is modified and its parameters have to be updated. We assume (as usual) that the Viterbi paths are not altered by merging, and the new Viterbi paths are inferred from the previous ones by replacing s_1 and s_2 by s. This assumption provides an efficicent way of updating parameters associated with the new state and to its adjacent edges. We have: $n_s =$

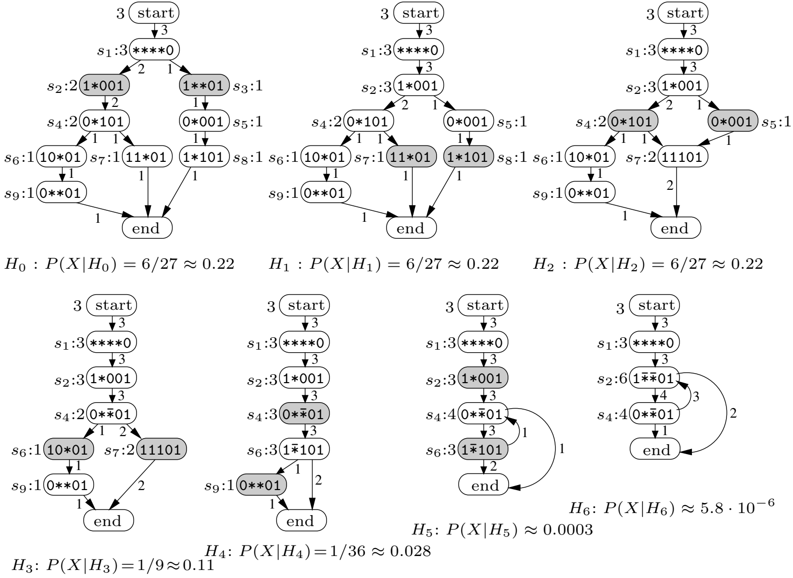

Fig. 2. Generalization achieved by six state mergings. The HMMPs are obtained by merging the grey nodes. For each HMMP, we indicate the probability of generating the learning sequences. Each state is labeled by its name and the number of times n_s it is used in the Viterbi paths. Each edge is also labeled by the number of times $n_{s\to t}$ it is ran over. The transition probability associated with $s \to t$ is equal to the ratio $n_{s\to t}/n_s$ (*c.f.* Formula (3))

$n_{s_1} + n_{s_2}$, $P_{X,s} = P_{X,s_1} \cup P_{X,s_2}$, and $\forall p \in P_{X,s}$, $n_s^p = n_{s_1}^p + n_{s_2}^p$. Moreover, for edges adjacent to s, we have: $n_{s\to t} = n_{s_1\to t} + n_{s_2\to t}$, $n_{t\to s} = n_{t\to s_1} + n_{t\to s_2}$ and $n_{s\to s} = n_{s_1\to s_1} + n_{s_1\to s_2} + n_{s_2\to s_2} + n_{s_2\to s_1}$. Note that only parameters associated with the new state and its adjacent edges need to be updated; the merging has no effect on the other states and transitions. Finally, the transition probabilities attached to the updated edges are computed using Formula (3).

The pattern associated with the new state must generate, with the highest possible probability, all patterns of $P_{X,s}$. It can be computed by merging all these patterns bit after bit, but a more efficient method is to merge bit after bit

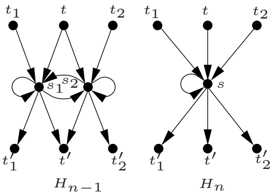

Fig. 3. States and transitions before and after the merging of s_1 and s_2

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

Fig. 4. Response table of the bit merging operator

the patterns p_{s_1} and p_{s_2} attached to s_1 and s_2. Let γ denote the bit merging operator. The character `*` means that the value of the bit is not important. Therefore, $\gamma(*, 0) = 0$ and $\gamma(*, 1) = 1$. But two patterns are not always mutually compatible. In this case, some of their bits differ and the merged pattern must generate `0` or `1` with equal probability on these bits. These bits take the value `*` in the merged pattern and are marked to store the fact that their value results from the merging of a `0` and a `1`. Let $\bar{*}$ denote the marked `*`. During further steps of the learning algorithm, this information is needed so that we do not set this bit at value `0` or `1`. The merging operator takes this into account, and we have: $\gamma(0, 1) = \bar{*}$, and during the further steps, $\gamma(\bar{*}, 0) = \gamma(\bar{*}, 1) = \gamma(\bar{*}, *) = \bar{*}$. Figure 4 summarizes the result of operator γ.

4.4 Selection of the Best State Pair

The aim of the learning algorithm is to reduce the HMMP structure while keeping (as much as possible) a high probability (given by Formula (2)) of generating X. At each step, the algorithm chooses the state pair which, when merged, involves the lowest loss of probability of generating X. Nevertheless, sometimes (especially at the beginning of the process) many pairs agree with this criterion. Then, we choose among these pairs that for which pattern merging involves fixing the lowest number of bits. The number of bits fixed by the merging of two patterns p and p' is obtained by formula

$$\varphi(p, p') = \min(*_p^{\gamma(p,p')}, *_{p'}^{\gamma(p,p')}). \quad (4)$$

For example, if $p =$ `1*11*` and $p' =$ `*0`$\bar{*}$`**`, then $\gamma(p, p') =$ `10`$\bar{*}$`1*`; we have $*^p_{\gamma(p,p')} = 1$, $*^{p'}_{\gamma(p,p')} = 2$ and then $\varphi(p, p') = 1$. Note that if p is more general than p', then $\varphi(p, p') = 0$. Using Formula (4) has two justifications. First, at the beginning of the learning procedure, this avoids fixing bits in the patterns too soon, which would make further mergings more difficult (in terms of likelihood). Second, we can reasonably assume that similar patterns quite likely play the same part in sequences.

In Figure 2, two state pairs of H_0 involve a null loss of generation probability: (s_2, s_3) and (s_7, s_8). However, $\varphi(p_{s_2}, p_{s_3}) = 0$, while $\varphi(p_{s_7}, p_{s_8}) = 1$. Therefore, (s_2, s_3) is selected and merged to obtain H_1.

At each step of the algorithm, this selection criterion involves computing the loss of generation probability for every state pair, i.e. calculating $O(n^2)$ times Formula (2), where n is the number of states in the current HMMP. When the initial HMMP contains numerous states (i.e. when the learning sequence set is large) this yields prohibitive computing time. A more efficient algorithm is proposed in [14]. It make use of the fact that merging has no effect on the states and transitions which are not adjacent to the merged states. It follows that the performance of a pair is not really affected, unless one of its states is adjacent to the merged states. Therefore, we initially compute the $O(n^2)$ criterion values for all pairs, and further we only update the values of the adjacent pairs, that is $O(nb)$, where b is the branching factor of the HMMP. Moreover, b remains relatively low

during the learning process. Indeed, our selection criterion (minimizing the loss of $P(X|H)$) tends to lower the number of out transitions (*c.f.* Formula (2)). For example, with $|Out(s)| = 1$ we have $\prod_{t \in Out(s)} M(s \rightarrow t)^{n_{s \rightarrow t}} = 1$, while with $|Out(s)| = 2$ and $n_{s \rightarrow t_1} = n_{s \rightarrow t_2}$, we have $\prod_{t \in Out(s)} M(s \rightarrow t)^{n_{s \rightarrow t}} = (1/2)^{n_s}$.

5 Experimental Results

After the learning phase, the HMMP can be physically implemented as test sequence generator. We do not describe this procedure (the interested reader can consult [15]). It consists in a natural microelectronic translation of the HMMP structure; the size of the implementation (an crucial factor for the validity of our approach) is strongly connected to the size of the HMMP (in terms of number of states and transitions).

The performances of our method were tested on the classical benchmark[1] circuits of the electronic community. The results are reported in Table 1. After the name of the circuit, the number of its inputs (#I.) and potential faults (#F.), we provide the fault coverage and the total length (in thousands of patterns) of ATPG sequences. For comparison purposes, we include the fault coverage achieved by a *long* Boolean vector sequence generated by a pure random process (only simulating one long sequence is highly justified in the case of a purely random approach, and no improvement is obtained by decomposing this long sequence into many shorter ones).

For every circuit, we computed by simulation the fault coverage of 10 random sequences, and the fault coverage of 10 sets of test sequences generated with the HMMP learned from the ATPG sequences. The T.Len. column provides both the length of the random sequences and the total length of the set of HMMP sequences. This length has been manually tuned for each circuit, in order to obtain a sufficient fault coverage. 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. On the bottom line, we report the means of these quantities.

First, it can be seen that the fault coverage achieved by using learned HMMP is much larger than the fault coverage of the random sequence. Next, we observe that it is often equal (s298, s1494), sometimes little smaller (s820, s832) and sometimes larger (s444, s526 and general mean) than the fault coverage of the ATPG sequences. This result is surprising and is a good confirmation of our method. 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 achieve high fault coverage when the generated sequence set is large enough. The good results obtained with our method could

[1] These benchmarks were created for the *International Symposium on Circuits & Systems* (*ISCAS*) in 1989. They are available at *http://www.cbl.ncsu.edu/benchmarks/*

Table 1. Fault coverage achieved by ATPG, random sequences and HMMPs

CIRCUIT			ATPG		Len. R.H.		RANDOM		HMMP			
Name	#I.	#F.	%Cov.	Leng	T.Len	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	74.0	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	85.2	79.3	5	10
s820	18	1640	96.3	4993	30K	6.00	49.4	48.4	93.3	91.2	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	92.9	91.4		

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 (see the results obtained by the random method).

6 Conclusion

We presented a new probabilistic model for learning Boolean pattern sequences, and its application for testing integrated circuits. This model is close to the classical HMM, but differs in that it defines the emission probability distribution with a boolean pattern. Moreover, we use this model for generation purposes and not for recognition, as in most HMM applications. Experimental results indicate that our model is well adapted for testing integrated circuits. Nevertheless, the fault coverage achieved for some circuits may be improved. A possible solution could be to weight ATPG sequences by the number of faults they detect.

HMMPs were defined to model pattern sequences. This notion of pattern – a set of vectors – is relatively specific to the test problem. Moreover, due to the generation aim of the test problem (with the constraint of generating at least one vector sequence from each pattern sequence computed by the ATPG), our γ operator performs specialization ($\gamma(\mathtt{0},*) = \mathtt{0}$ and $\gamma(\mathtt{1},*) = \mathtt{1}$) and not only generalization as is usually done in the recognition framework.

Nevertheless, as stated in the introduction, HMMPs could be used to model more usual vector sequences (as time series for example). In this case, the non-marked character $*$ does not appear, and γ only performs generalization in the usual sense: $\gamma(\mathtt{0},\mathtt{1}) = \gamma(\mathtt{0},\bar{*}) = \gamma(\mathtt{1},\bar{*}) = \bar{*}$. Moreover, the φ function is useless (it returns 0 for every pair) and the likelihood is the only selection criterion. Therefore, for Boolean vector sequences, the γ operator and our learning algorithm could be used directly. For sequences of vectors with discrete variables

(ordinal or not), slight modifications of the model, and of the learning procedure (especially of γ) are easily conceivable.

The Boolean patterns define very simple probability distributions over $\{0, 1\}^k$. Numerous more expressive distribution classes could be envisaged, such as, for example, using a Bernoulli distribution associated with each bit. However, the simplicity of Boolean patterns is well suited for microelectronic purpose. Indeed, the * is free and the 1 and 0 have very low cost in terms of silicon area overhead [15], while implementing continuous probabilities is much more expensive. Moreover, our Boolean patterns are very similar to the condition parts of the rules used in symbolic and hybrid classification methods [16], and, just as in these methods, the simplicity of these descriptions is associated to explanatory virtues which should be of interest from a modeling and learning perspective.

References

1. Baum L. E., Petrie T., Soules G., Weiss N.: A maximization technique occurring in statistical analysis of probabilistic functions in Markov Chains. The Annals of Mathematical Statistics **41** (1970) 164–171 75
2. Casacuberta F.: Some Relations Among Stochastic Finite State Networks Used in Automatic Speech Recognition IEEE PAMI **12** (1990) 691–695 75
3. Baum L. E.: An inequality and associated maximization technique in statistical estimation for probabilistic functions of a Markov process. Inequalities **3** (1972) 1–8 75
4. Dempster A. P., Laird N. M., Rubin D. B.: Maximum likelihood from incomplete data via the EM algorithm. J. Royal Stat. Soc. B **39** (1977) 1–38 75
5. Abe N., Warmuth M.: On the Computational Complexity of Approximating Distributions by Probabilistic Automata. Machine Learning **9** (1992) 205–260 75
6. Angluin D., Smith C.H.: Inductive Inference: Theory and Methods. ACM Computing Surveys **15** (1983) 237–269 76
7. Carrasco R. C., Oncina J.: Learning Stochastic Regular Grammars by Means of a State Merging Method. Proceedings of the Second International ICGI (1994) 139–152 76
8. Stolcke A., Omohundro S.: Inducing Probabilistic Grammars by Bayesian Model Merging. Proceedings of the ICGI (1994) 106–118 76
9. Rabiner L. R.: A Tutorial on Hidden Markov Models and Selected Applications in Speech Recognition. Proceedings of the IEEE **77** (1989) 257–285 76
10. Schenkel M.,Guyon I., Henderson D.: On-line cursive script recognition using time delay neural networks and hidden Markov models. Proc. ICASSP '94 (1994) 637–640 76
11. Krogh A., Brown M.,Saira Mian I., Sjolander K., Haussler D.: Hidden Markov Models in computational Biology: Applications to protein modelling. UCSC-CRL-93-32 (1993) 76
12. Ibarra, Sahni: Polynomially Complete Fault Detection Problems. IEEETC: IEEE Transactions on Computers **24** (1975) 77
13. Forney G. D.: The Viterbi algorithm. Proc. IEEE **61** (1973) 268–278 79
14. Bréhélin L.,Gascuel O., Caraux G: Hidden Markov Models with Patterns to Learn Boolean Vector Sequences; Application to the Built-in Self-Test for Integrated Circuits. TR 99006 (1999) 83

15. Bréhélin L., Gascuel O., Caraux G., Girard P., Landrault C.: Hidden Markov and Independence Models with Patterns for Sequential BIST. 18th IEEE VLSI Test Symposium, In Press (2000) 84, 86
16. Gascuel O. and the SYMENU group: Twelve numerical, symbolic and hybrid supervised classification methods. International Journal of Pattern Recognition and Artificial Intelligence (1998) 517–572 86