LOCSTEP: A Logic-Simulation-Based Test Generation Procedure

Irith Pomeranz, *Senior Member*, IEEE, and Sudhakar M. Reddy, *Fellow, IEEE*

***Abstract*—We present a method to generate test sequences that detect large numbers of faults (close to or higher than the number of faults that can be detected by deterministic methods) at a cost which is significantly lower than any existing test generation procedure. The generated sequences can be used alone or as prefixes of deterministic test sequences. To generate the sequences, we study the test sequences generated by several deterministic test generation procedures. We show that when deterministic test sequences are applied, the fault-free circuits go through sequences of state transitions that have distinct characteristics which are independent of the specific circuit considered. Test sequences with the same characteristics are generated in this work by using logic simulation only on the fault-free circuit, and by considering several random patterns as candidates for inclusion in the test sequence at every time unit. By fault simulating these sequences, we find that the fault coverage achieved is very close to the fault coverage achieved by deterministic sequences, and sometimes is even higher.**

I. Introduction

THE problem of test generation for synchronous sequential circuits has been studied extensively, resulting in several test generation procedures capable of achieving complete or close to complete fault coverage [1]–[15]. However, test generation time is typically on the order of several hours or more for the larger ISCAS-89 benchmark circuits. The long run time may prevent the application of these test generation procedures to large circuits or may result in low fault coverage. A sequence of random patterns can be used as a prefix of a deterministic test sequence to achieve low-cost detection of some faults, and thus reduce test generation time. However, in many cases, long random sequences are required to detect significant numbers of faults. In addition, if a random sequence detects only faults which are easy to detect deterministically, then the saving in test generation time may not be significant.

Our goal in this work is to generate test sequences that detect large numbers of faults (as close as possible to the number of faults that can be detected by deterministic methods and even higher) at a cost which is significantly lower than the cost of deterministic test generation. Such a test generation procedure would help alleviate the problems related to the long run times of these procedures. To achieve this goal, we study the test sequences generated by the test generation procedures HITEC [9] and FASTEST [7]. We show that under these test sequences, the fault-free circuits go through sequences of state transitions that have distinct characteristics which are independent of the specific circuit considered. By using logic simulation on the fault-free circuit and considering a number of random patterns as candidates for inclusion in the test sequence at every time unit, we are able to generate a test sequence that takes the fault free circuit through a sequence of state transitions that has the same characteristics as sequences of state transitions under deterministic test sequences. By fault simulating the test sequences generated in this way, we find that the fault coverage achieved is very close to the fault coverage achieved by deterministic sequences and sometimes even higher, and typically higher than the fault coverage achieved by other test generation procedures [11], [15]. Since logic simulation has significantly lower complexity than test generation or even fault simulation, the test sequences are generated at a significantly reduced cost. The result is an effective and time-efficient test generation procedure. We stress that the characteristics used are general, i.e., they are applicable to any circuit, although they are learned on a limited set of circuits, and that the test sequences generated based on the characteristics learned are effective for unstudied circuits.

The proposed procedure can be extended to consider random sequences instead of single patterns as candidates for inclusion in the test sequence. The consideration of sequences may allow us to generate test sequences whose characteristics better match those of deterministic test sequences. However, large numbers of random sequences may have to be considered to realize this advantage. We therefore prefer to consider single input patterns.

We apply the proposed method to circuits without a reset mechanism. The test sequences we study [7], [9] are generated under the single observation time approach [16] and using three-value simulation. Therefore, the test sequences generated in this work are also applied under the same conditions. A similar study under the multiple observation time approach [16] is also possible.

In [19], the effect of retiming on test generation complexity was studied, and it was shown experimentally that the fault coverage increases with the number of states traversed by the test sequence. Our studies reinforce this observation, and identify additional properties of test sequences that achieve high fault coverage. The conclusion of [19] is that circuits with larger fractions of reachable states are easier to test. Here, we use the properties derived to generate an input sequence that achieves high fault coverage for a given circuit.

Manuscript received October 20, 1995; revised July 22, 1996 and February 27, 1997. This work was supported in part by NSF Grant MIP-9220549 and NSF Grant MIP-9357581. This work is based on a paper presented at the 25th IEEE Fault-Tolerant Computing Symposium, June 1995. This paper was recommended by Associate Editor K.-T. Cheng.

The authors are with the Department of Electrical and Computer Engineering, University of Iowa, Iowa City, IA 52242-1595 USA.

Publisher Item Identifier S 0278-0070(97)06198-8.

The paper is organized as follows. In Section II, we demonstrate the existence of distinct characteristics of deterministic test sequences that hold for a variety of circuits and are independent of the specific circuit considered. In Section III, we present four procedures to generate test sequences with the same characteristics as the deterministic test sequences. In Section IV, we present experimental results, and compare the fault coverage obtained based on the proposed method to the fault coverage of several other test generation procedures. In Section V, we consider a parameterized test generation procedure, where the distance between consecutive states and state weights are used as guidelines for selecting input patterns for inclusion in the test sequence. Section IV concludes the paper.

II. Characteristics of Deterministic Test Sequences

Our goal in this section is to identify characteristics of deterministic test sequences generated by test generation procedures that achieve complete or close to complete fault coverage. The information collected is then used to establish test generation procedures that generate test sequences having similar characteristics, without performing deterministic test generation. Our conjecture is that test sequences that imitate the behavior of deterministic test sequences achieve similar fault coverage for the same circuits, as well as for other circuits that are not studied in this section. To distinguish between test sequences generated by a deterministic test generation procedure and test sequences generated by the proposed method, we refer to the latter as *imitation test sequences* and to the former as *test sequences* or as *deterministic test sequences*. To keep the process of generating imitation test sequences simple, we restrict this study to characteristics of deterministic test sequences that can be learned through logic simulation of the fault-free circuit.

A. Experimental Results

To collect the required information, we performed the following experiment. We performed logic simulation of fault-free circuits under test sequences generated by the deterministic test generation procedures of [7] and [9]. The circuits were started from the fully unspecified initial state. For every test sequence, we recorded the sequence of states that the fault-free circuit visits during the application of the test sequence. A state is identified by the values stored in the flip-flops. Each state is also given an integer identifier to simplify the storage and representation of states. The integer identifiers are given according to the order in which the states are reached. In addition, we record the number of different total states that the fault-free circuit goes through along the test sequence. A *total state* is a combination of a present state and a primary input pattern (the term is borrowed from asynchronous circuit theory). Two total states are said to be different if they differ in the present state or in the primary input pattern. A detailed example of how the results are collected is given next for $s298$ under the test sequence generated by HITEC [9], followed by short descriptions of the results for several other circuits. We point out that the test sequences generated by HITEC [9] were made available to us by the authors, and do not always correspond to the test sequences reported in the literature. Thus, the test lengths and fault coverages of these sequences reported in this section may be different from the ones reported in Table III below, which is based on [9].

Starting from the all-unspecified initial state, the first few states visited by $s298$ with the application of the test sequence of length 259 produced by HITEC are the following: 000000$xxxxxx$00, 10000001100011, 01001001100000, 11001001100010, 00101001100010, and 10101001100010. We number these states 0, 1, 2, 3, 4, and 5, respectively. Thus, the sequence of states that $s298$ goes through under the HITEC test sequence starts with 0 1 2 3 4 5. The complete sequence of states visited is the following:

0 1 2 3 4 5 6 7 8 9 10 11 12 12 13 12 1 14 15 4
16 17 18 19 20 21 12 22 12 20 23 24 25 26 27
28 29 30 31 32 33 34 35 36 37 38 39 40 41 42
43 44 45 46 47 48 49 50 51 12 12 20 52 12 22
2 53 54 55 6 7 56 57 58 59 14 12 13 60 24 61
62 63 64 65 66 67 68 69 3 4 5 70 71 56 57 72
73 74 53 54 55 6 7 75 76 77 78 11 12 12 13 52
79 80 81 82 18 83 84 85 1 86 87 88 89 90 91
92 93 94 59 14 15 95 96 97 98 99 100 67 68
69 53 4 55 6 7 75 101 12 12 20 12 20 23 102
25 26 27 28 29 103 104 105 43 44 45 46 47
106 107 108 109 110 111 34 35 36 112 38 113
40 41 42 114 115 116 117 118 48 119 30 31
32 33 120 121 122 37 123 39 124 125 126 127
128 129 130 131 132 133 134 135 136 137 138
139 140 141 142 49 50 135 136 137 138 129
130 131 123 143 103 144 42 51 13 12 20 21
3 54 5 90 91 19 12 12 12 12 12 13 23 24 80
16 63 145 146

The number of times each state is visited varies between one and 14. Most of the states are visited only once.

For $s344$, HITEC generates a test sequence of length 108. The fault-free circuit goes through the following sequence of states:

0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18
19 20 21 22 23 24 25 25 25 26 27 28 29 30 31
32 33 34 35 36 37 38 39 40 41 42 43 44 45 46
47 2 48 49 50 51 52 53 54 55 56 57 58 59 60
61 62 63 64 65 66 67 68 69 70 70 70 71 72 73
74 75 76 77 78 79 80 81 82 82 82 83 84 85 86
87 88 89 90 91 92 93 94 95 96 97 98 99 100

The number of times each state is visited varies between one and three, with most states being visited only once.

For $s420$, HITEC generates a test sequence of length 166. The fault-free circuit goes through the following sequence of

states:

0 1 0 1 0 0 0 0 0 1 0 0 0 0 1 2 3 4 5 6 7 8 9 10 11
12 13 14 15 0 1 0 0 1 0 1 2 3 4 5 6 7 8 0 1 0 1 2
3 4 5 6 7 8 9 10 11 12 13 14 15 16 1 0 1 2 3 0 0
1 2 3 4 0 0 0 0 0 1 2 3 4 5 6 7 0 1 0 1 2 3 4 5 6 7
8 9 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 0 0
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 0 1 0 1 2
3 4 5 6 7 8 9 10 11 12 13 14 15 16 0 0 1 2 3 4 5
6 7 8 9 10 11 12 13

The number of times each state is visited varies between four and 34.

B. Characteristics

From the results reported in the previous subsection and results collected for other circuits, we conclude that deterministic test sequences have the following characteristics. In Section III, we consider the incorporation of these characteristic (some or all of them) into test generation procedures based on logic simulation.

1) New States: The test sequences considered in the previous subsection can be divided into two classes according to the way in which new states are reached. In the first class, the test sequence takes the fault-free circuit into a different state in almost every time unit. The number of different states is smaller than the test sequence length only when transfer sequences are required from one new state to another, that go through states already visited. For example, consider the following prefix of the sequence of states produced by HITEC for $s298$, given above:

0 1 2 3 4 5 6 7 8 9 10 11 12 12* 13 12* 1* 14 15
4* 16 17 18 19 20 21 12* 22 12* 20* 23 24 25
26 27 28 29 30 31 32

The states corresponding to transfer sequences (states visited before along the test sequnece) are marked with asterisks. An example with even fewer transfer sequences is $s344$, where only seven states are repeated as part of transfer sequences. We conclude that effective deterministic test sequences cause the fault-free circuit to visit a new state whenever possible. This results in the following guideline.

Guideline 1: Generate an imitation test sequence that brings the fault-free circuit into as many different states as possible.

For the second class of circuits, deterministic test sequences take the circuit through repeated sequences of consecutive states. For example, in the state sequence obtained from the test sequence produced by HITEC for $s420$, the following subsequence is repeated five times:

0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15

One of the reasons for repeating the same subsequence is that no new states can be reached. Experiments indicate that this may be the case for both $s208$ and $s420$, where at most 17 different states could be reached starting from the all-unspecified initial state. For such circuits, Guideline 2 is useful in generating imitation test sequences.

TABLE I
DIFFERENT TOTAL STATES IN HITEC TEST SEQUENCES

circuit	length	total-states
s298	259	205
s344	108	108
s400	2069	830
s420	430	166
s526	192	191
s641	211	211

Guideline 2: If no new states can be reached, generate the imitation test sequence such that it repeats the states already reached before in the order they were reached for the first time.

2) Repeated States: The circuits considered can be partitioned into two classes according to the states they transverse during transfer sequences. In this context, a transfer sequence takes the circuit from one new state to the next, or into a previously traversed sequence of states. In the first class, low-index states are repeated during the transfer sequences. For illustration, part of the HITEC state sequence for $s400$ is shown below. The transfer sequences, marked with asterisks, are comprised mainly of state 0

0 1 2 3 4 5 0* 1* 0* 6 7 8 9 10 11 12 13 14 15
16 2* 17 18 19 20 21 22 23 24 25 26 27 28
29 30 31 32 33 343 35 0* 0* 36 7* 37 9* 38 11*
39 40 41 42 43 44 45 46 47 48 49 50 51 52
53 54 55 56 57 58 59 60 61 62 63 64 65 66 67
68 69 70 71 72 73 74 75 76 77 78 79 80 81
82 0* 83 0* 6* 84

This leads to the following guideline.

Guideline 3: When it is impossible to reach a new state or repeat a preceding sequence, select the state with the lowest index that can be reached.

Guideline 3 results in test sequences that visit the lower index states more often than the higher index states. The second class of circuits includes, for example, $s298$, where all of the states are visited approximately the same number of times. To generate such imitation test sequences, we use the following guideline.

Guideline 4: When it is impossible to reach a new state or repeat a preceding sequence, select a state that has been visited a minimum number of times. If a choice exists, select the state with the lowest index.

3) New Total States: For all of the circuits considered, new total states are used whenever possible. The test length and the number of different total states traversed during HITEC test sequences are given in Table I. This leads to the following guideline.

Guideline 5: Generate an imitation test sequence using as many different total states as possible.

4) Summary: The five guidelines derived above are repeated next for ease of reference.

Guideline 1: Generate an imitation test sequence that brings the fault-free circuit into as many different states as possible.

Guideline 2: If no new states can be reached, generate the imitation test sequench such that it repeats the states already reached before in the order they were reached for the first time.

Guideline 3: When it is impossible to reach a new state or repeat a preceding sequence, select the lowest index state that can be reached.

Guideline 4: When it is impossible to reach a new state or repeat a preceding sequence, select a state that has been visited a minimum number of times. if a choice exists, select the lowest index state.

Guideline 5: Generate an imitation test sequence using as many different total states as possible.

III. Generating Imitation Test Sequences

Based on the guidelines presented in the previous section, we experimented with several procedures that give different importance to the various guidelines. We report on four procedures for generating imitation test sequence that were most effective in achieving high fault coverage. The procedures construct an imitation test sequence by concatenating single input patterns. The input pattern at a given time unit is selected out of a collection of up to K randomly generated input patterns, where K is a predetermined constant ($K = 100$ in our experiments). Each random pattern is simulated on the fault-free circuit until a suitable one is found and selected, or until all are simulated. In the latter case, the pattern most appropriate for satisfying the guidelines above is selected. Next, we describe each one of the procedures.

Procedure 1 attempts to traverse as many new states as possible, according to Guideline 1. To satisfy Guideline 4, when a state has to be repeated, the state that has been visited a minimum number of times and has the smallest index is selected. In this way, after tracing new states $S_0, S_1, \cdots, S_k$, every one of $\{S_0, S_1, \cdots, S_k\}$ has been visited once. If no new state can be reached and S_0 can be reached, S_0 would be selected. At this point, S_0 would be visited twice. If no new state can be reached from S_0, then the states that have been visited a minimum number of times are $\{S_1, S_2, \cdots, S_k\}$. If S_1 can be reached, then it would be selected. Next, S_2 would be the best candidate, and so on. Thus, Guideline 2 would be followed as closely as possible. In addition, Procedure 1 satisfies Guideline 5 by selecting new total states whenever possible. The procedure is given next.

Procedure 1: Imitation test sequence generation (Heuristic 1).

1) Set the state P of the circuit to the all-unspecified state. Set $S = \phi$ (S is the set of states already reached along the imitation test sequence). Set $T = \phi$ (T is the imitation test sequence). Set $u = 0$ (u is the time unit).
2) Set $i = 0$.
3) Generate a random primary input pattern r_i. Simulate r_i on the fault-free circuit starting from state P. Let the next state be Q_i.
   a) If $Q_i \notin S$ (Q_i is a new state), add Q_i to S, concatenate r_i to the end of T, set $P = Q_i$, and go to step 5.
   b) If the total state (r_i, P) is not included in T, set the flag $new\_tot\_state_i$ to 1. Set $visited_i$ equal to the number of times state Q_i is reached under T.
   c) Set $i = i + 1$. If $i < K$, where K is a predetermined constant, go to the beginning of Step 3.
4) Select an input pattern r_{i_0} such that $visited_{i_0}$ is minimum. In case of ties, select an input pattern r_{i_0} such that Q_{i_0} has the smallest index. If several input patterns exist for which $visited_i$ is minimum and Q_i has the smallest index, select r_{i_0} such that $new\_tot\_state_{i_0}$ is 1. Add Q_{i_0} to S, concatenate r_{i_0} to the end of T, and set $P = Q_{i_0}$.
5) Set $u = u + 1$. If $u < L$, where L, is a predetermined bound on test sequence length, go to Step 2.

Procedure 2 is similar to Procedure 1, except that it tries to satisfy Guideline 3 as follows. After a new state is reached and if no additional new states can be reached, the sequence returns to the smallest index state it can find (in accordance with Guideline 3). Otherwise, if the current state is not a new one, the selection criteria used in Procedure 1 (based on Guidelines 2, 4, and 5) are used. Procedure 2 is given below.

Procedure 2: Imitation test sequence generation (Heuristic 2).

1) Set the state P of the circuit to the all-unspecified state. Set $S = \phi$ (S is the set of states already reached along the imitation test sequence). Set $T = \phi$ (T is the imitation test sequence). Set $u = 0$ (u is the time unit). Set $minimal\_index = 0$.
2) Set $i = 0$.
3) Generate a random primary input pattern r_i. Simulate r_i on the fault-free circuit starting from state P. Let the next state be Q_i.
   a) If $Q_i \notin S$ (Q_i is a new state), add Q_i to S, concatenate r_i to the end of T, set $P = Q_i$, set $minimal\_index = 1$, and go to Step 5.
   b) If the total state (r_i, P) is not included in T, set the flag $new\_tot\_state_i$ to 1. Set $visited_i$ equal to the number of times state Q_i is reached under T.
   c) Set $i = i + 1$. If $i < K$, where K is a predetermined constant, go to the beginning of Step 3.
4) If $minimal\_index = 0$:
   Select an input pattern r_{i_0} such that $visited_{i_0}$ is minimum. In case of ties, select an input pattern r_{i_0} such that Q_{i_0} has the smallest index. If several input patterns exist for which $visited_i$ is minimum and Q_i has the smallest index, select r_{i_0} such that $new\_tot\_state_{i_0} = 1$.
   If $minimal\_index = 1$:
   Select an input pattern r_{i_0} such that Q_i has the smallest index. In case of ties, select one such that $new\_tot\_state_{i_0} = 1$.
   Add Q_{i_0} to S, concatenate r_{i_0} to the end of T, and set $P = Q_{i_0}$. Set $minimal\_index = 0$.
5) Set $u = u + 1$. If $u < L$, where L is a predetermined bound on test sequence length, go to Step 2.

TABLE II
FAULT COVERAGE BY IMITATION TEST SEQUENCES

	LOCSTEP											
	Proc. 1			Proc. 2			Proc. 3			Proc. 4		
circuit	det	len		det	len		det	len		det	len	
s208	132	614		132	614		135	2002	*	135	2002	*
s298	265	522		265	1007		265	522		265	1243	
s344	329	1583	*	329	3411	*	328	962		325	6002	
s382	350	7100		357	5354	*	354	4002		354	4002	
s386	263	8042		274	6742		275	9003		303	6946	*
s400	367	7100		374	5354	*	371	4003		371	4003	
s420	174	406		174	406		177	2004	*	177	2004	*
s444	411	6178		416	2922	*	413	9002		413	9002	
s526	438	7315		441	7743	*	439	9042		439	6004	
s641	403	1059		403	623		403	1828		404	7792	*
s713	475	1059		475	623		475	1828		476	7792	*
s820	700	8454		698	5788		721	8655		742	9034	*
s1238	1267	8349		1268	9409	*	1263	9406		1263	9406	
s1423	1274	9616	*	1274	9616	*	920	9894		920	9894	
s1488	1403	2091		1400	2427		1425	7002	*	1412	8026	
s5378	3045	9284		3045	9284		3045	9284		3045	9284	
				3059	7545							
s35932	33372	102		6221	56		34812	2408	*	34812	2408	*

Due to the use of random patterns, several applications of Procedures 1 or 2 may result in several different test sequences that detect different faults. In addition, the fault coverage of a long test sequence tends to saturate such that the last input patterns are less effective in detecting new faults than the first ones. To take advantage of these observations, procedures 3 and 4 generate test sequences of length L by repeating Procedures 1 and 2, respectively, several times, generating individual sequences of length Δ. They concatenate L/Δ such test sequences. The procedures are given next.

Procedure 3 (4): Imitation test sequence generation [Heuristic 3(4)].

1) Set $T = \phi$.
2) Repeat L/Δ times:
   a) Generate an imitation test sequence T' of length Δ using Procedure 1 (2).
   b) Concatenate T' to T.

The length of the test sequence L is determined arbitrarily in our experiments. It is also possible to use a circuit-dependent test length by running a fault simulation procedure in parallel with the sequence generation procedure. Sequence generation can then stop when the fault coverage saturates.

IV. EXPERIMENTAL RESULTS

We applied Procedures 1–4 to generate imitation test sequences for ISCAS-89 benchmark circuits. We used the fault simulator PROOFS [17] to obtain the fault coverage of these imitation test sequences. We then compared the fault coverage obtained to the fault coverage of deterministic test pattern generators and of test pattern generators based on genetic optimization. The imitation test sequences were generated using $K = 100$ random patterns at ever time unit. All of the sequences were of length $L = 10\,000$. In Procedures 3 and 4, we initially used $\Delta = 1000$ as the subsequence length. If the results were improved compared to Procedures 1 or 2, respectively, we considered the fault coverage curve. If it turned out that most of the faults were detected close to the beginning of the subsequences, we reduced Δ to 100. Consequently, Δ was reduced to 100 for $s386$ under Procedure 4, and for $s35932$ under Procedures 3 and 4. The results are given in Table II as follows. For each procedure, we give the number of faults detected and the last test pattern that was effective in detecting any fault. For $s5378$, when we applied Procedure 2 using a different seed for random pattern generation, the number of faults detected was 3059, using 7545 tests. We include this result since there is a significant difference between the number of faults detected by the two applications of Procedure 2. The best fault coverage for each circuit is marked with an asterisk (unless all four procedures gave the same fault coverage).

For comparison, we give in Table III the number of detected faults and the last effective pattern by a pure-random sequence of length 10 000. Four different random sequences were generated, and the best result is reported. We also give the number of detected faults and the test length for HITEC [9] and for the procedures from [11] and [15], which are based on genetic optimization. Under column LOCSTEP, we repeat the best result obtained by the proposed procedures. Using the best result corresponds to applying all four procedures to each circuit and selecting the best test sequence obtained. It can be seen that the fault coverage of LOCSTEP is always better than that of pure random patterns. It is close to and sometimes even higher than the fault coverage of HITEC. It is higher than the fault coverage of [11] in most cases, and higher than [15] in many of the circuits. It should be pointed out, however, that the results in [15] were obtained by taking the average over ten runs, and that many of the runs gave better results than the average. As can be expected, the effective test lengths of the LOCSTEP sequences are relatively high. The sequences can be significantly compacted by using static compaction procedures [20].

We also considered other circuits, not included in the ISCAS-89 suite. We applied Procedure 2, which is one of the

TABLE III
COMPARISON WITH TEST GENERATION PROCEDURES

	pure-rand.		HITEC[9]		CRIS[11]		[15]		LOCSTEP	
circuit	det	len	det	len	det	len	det	len	det	len
s208	94	9580	137	184	131	715	NA	NA	135	2002
s298	247	9984	265	306	253	476	265	161	265	522
s344	329	2173	329	108	328	115	329	95	329	1583
s382	53	593	363	4931	274	246	347	281	357	5354
s386	260	7830	314	311	292	1230	295	154	303	6946
s400	57	3853	383	4309	357	758	365	280	374	5354
s420	131	7161	179	218	145	857	NA	NA	177	2004
s444	60	4337	414	2240	397	519	406	275	416	2922
s526	52	116	365	2232	428	692	417	281	441	7743
s641	403	7667	404	216	398	628	404	139	404	7792
s713	475	7667	476	194	475	1124	476	128	475	1059
s820	398	8006	813	984	451	1381	517	146	742	9034
s1238	1249	9269	1283	478	1229	4313	1274	383	1268	9409
s1423	797	9806	816	NA	1167	2696	1222	663	1274	9616
s1488	1071	9550	1444	1294	1355	1960	1392	243	1425	7002
s5378	3046	9825	3150	900	3029	1255	3175	511	3059	7545
s35932	33300	9291	34902	240	34481	1525	35009	197	34812	2408

most effective procedures according to Table II. The results of Procedure 2 are reported next.

For circuit $fir8$ considered in [11], the number of detected faults reported in [11] is 2488. *LOCSTEP* detected 2596 faults.

For circuit $am2910$ considered in [18], the number of detected faults by HITEC is 2164, and the number of detected faults by the procedure reported in [18] (that uses a combination of deterministic test generation and genetic optimization) is 2188. LOCSTEP detected 2221 faults.

For circuit *pcont2* considered in [18], the number of detected faults by HITEC is 3354, and the number of detected faults by the procedure reported in [18] is 6767. LOCSTEP detected 6806 faults.

Next, we consider the run time of Procedures 1–4. We use Procedure 2 to illustrate our observations. Similar analysis applies to the other procedures as well.

Procedures 1–4 can be implemented efficiently by using techniques for parallel logic simulation, and techniques for storing visited states and total states that allow fast insertion and search. For example, storing visited states and state transitions in hash tables of size N can reduce the time for search and storage of new states and state transitions approximately N times on the average. An implementation that uses these techniques is beyond the scope of this paper. However, to demonstrate the importance of these techniques in the context of the proposed procedures, we include in Table IV information about the run time of Procedure 2 for several of the circuits considered above. The run time is divided into three components. Under column "*l.sim*," we show the fraction of the total run time spent in logic simulation; under column "*new*," we show the fraction of run time spent in identifying new states and new state transitions; and under column "*other*," we show the time spent in the remaining procedures. Run time was measured on an HP 730 workstation. It can be seen that for most of the circuits, most of the time is spent in searching and storing states and state transitions. Logic simulation takes the next significant part of the total time. These data support our claim that well-known techniques for speeding up these procedures will yield an efficient implementation of the LOCSTEP procedures. The only exceptions to the observations above are $s1423$ and $s5378$. In $s5378$, approximately equal time is spent on logic simulation, identifying new states and new transitions, and the remaining procedures. The reason for this is that in $s5378$, almost every input pattern takes the circuit into a new state. Thus, only one input pattern (or a small number of input patterns) needs to be simulated before the input pattern at every time unit can be selected. As a result, the total run time for $s5378$ is also relatively low, 11 min. Similar observations apply to $s1423$. For $s1423$, the total run time is also relatively low, 15 min.

TABLE IV
RELATIVE RUN TIMES

circuit	l.sim	new	other
s208	0.066	0.924	0.010
s298	0.115	0.870	0.015
s344	0.079	0.908	0.013
s382	0.074	0.910	0.017
s386	0.137	0.855	0.008
s400	0.080	0.903	0.016
s420	0.110	0.870	0.021
s444	0.091	0.892	0.017
s526	0.111	0.872	0.017
s641	0.138	0.831	0.031
s713	0.154	0.815	0.031
s820	0.209	0.784	0.007
s1238	0.222	0.763	0.015
s1423	0.097	0.620	0.282
s1488	0.391	0.603	0.006
s5378	0.298	0.348	0.353

V. DISTANCE AND WEIGHT INFORMATION

In this section, we study additional characteristics of deterministic test sequences that may be useful as guidelines for generating effective imitation test sequences. They are considered here separately since they require that the imitation test generation procedures be parameterized, using circuit-dependent parameters. Before addressing the problem of parameterizing Procedures 1–4, we describe the characterstics

TABLE V
PRIMARY INPUT WEIGHTS AND DISTANCES

circuit	inp	HITEC[9]		pure-random	
		$\overline{PI\_D}$	$\overline{PI\_W}$	$\overline{PI\_D}$	$\overline{PI\_W}$
s298	3	0.73	0.73	1.50	1.50
s344	9	4.48	4.53	4.50	4.50
s400	3	0.45	0.34	1.50	1.50
s420	19	8.71	9.17	9.50	9.50
s526	3	0.76	0.71	1.50	1.50
s641	35	17.94	17.97	17.50	17.50
s820	18	7.94	8.57	9.00	9.00
s1238	14	6.94	7.20	7.00	7.00
s1423	17	7.85	8.73	8.50	8.50
s1488	8	3.64	4.43	4.00	4.00
s5378	35	18.14	17.36	17.50	17.50
s35932	35	20.17	17.95	17.50	17.50

TABLE VI
STATE WEIGHTS AND DISTANCES

circuit	s.v	HITEC[9]		pure-random	
		$\overline{ST\_D}$	$\overline{ST\_W}$	$\overline{ST\_D}$	$\overline{ST\_W}$
s298	14	3.74	4.67	1.93	3.19
s344	15	4.58	8.35	4.20	8.56
s400	21	2.66	8.57	1.98	5.26
s420	16	1.57	1.47	0.50	0.26
s526	21	2.72	5.86	1.96	3.24
s641	19	2.38	4.82	1.98	4.04
s820	5	1.49	1.64	0.79	0.55
s1238	18	5.46	9.28	5.29	9.32
s1423	74	14.69	15.75	10.06	11.61
s1488	6	3.06	2.51	0.81	0.48
s5378	179	38.07	67.42	38.97	69.31
s35932	1728	493.76	424.49	291.94	173.83

and present experimental data regarding their values and the benefit of using them. The characterstics are the following.

- The distance between consecutive states and consecutive input patterns (the *distance* between two vectors v_1 and v_2 is the number of bits where v_1 and v_2 are specified to opposite $\{0,1\}$ values).
- The weights of the states and input patterns (the *weight* of a vector v is the number of bits specified to 1 in v).

The purpose of this study is to check the following points.

1) In Sections III and IV, we used pure random input patterns as candidates for inclusion in imitation test sequences. For a circuit with N_{PI} primary inputs, pure random patters are characterized by an average weight of $N_{PI}/2$ and an average distance of $N_{PI}/2$. If deterministic test sequences have different weights and distances, then incorporating this information into Procedures 1–4 may increase their effectiveness.

2) In Sections III and IV, we selected the first input pattern that resulted in a new state. If it turns out that deterministic test sequences reach states with specific weights or create specific distances between consecutive states, then this information can be used to improve Procedures 1–4.

The distance and weight information for benchmark circuits under deterministic test sequences are presented next. We considered the test sequences generated by HITEC [9]. For every circuit, we simulated the sequence on the fault-free circuit and collected the following information at every time unit u.

- $PI\_D(u)$ the distance between the primary input pattern applied at time unit u and the primary input pattern applied at time unit $u+1$.
- $PI\_W(u)$ the weight of the primary input pattern applied at time unit u.
- $ST\_D(u)$ the distance between the state reached at time unit u and the state reached at time unit $u+1$.
- $ST\_W(u)$ the weight of the state reached at time unit u.

We then computed the following parameters.

$\overline{PI\_D}, \overline{PI\_W}, \overline{ST\_D}, \overline{ST\_W}$—the average over all u of $PI\_D(u), PI\_W(u), ST\_D(u), ST\_W(u)$, respectively.

$N\_PI\_D(n)$, $N\_PI\_W(n)$, $N\_ST\_D(n)$, $N\_ST\_W(n)$—the number of time units u where $PI\_D(u) = n, PI\_W(u) = n, ST\_D(u) = n, ST\_W(u) = n$, respectively.

The values of $\overline{PI\_D}$ and $\overline{PI\_W}$ under HITEC test sequences are shown in Table V for several circuits. We also include in the table the number of primary inputs and the values of $\overline{PI\_D}$ and $\overline{PI\_W}$ when pure random patterns are used. For most circuits, the values of $\overline{PI\_D}$ and $\overline{PI\_W}$ are similar for deterministic test generation and for pure random patterns. Consider, for example, $s820$. An average weight of 8.57 implies that the probability of a 1 in the test sequence is approximately 8.57/18 = 0.48. The same probability for a pure random sequence is 0.5. The difference between the two numbers is small, and we believe that it does not justify the added complexity of selecting appropriate weights for generation of weighted random sequences. The only exceptions are the circuits having three primary inputs; however, since the fault coverages achieved for these circuits in Section IV are high, we did not attempt to generate weighted random patterns for them. The distributions of the various values of $PI\_D(u)$ and $PI\_W(u)$, given by $N\_PI\_D(n)$ and $N\_PI\_W(n)$, respectively, are also similar for deterministic and for pure random test sequences. We conclude that the use of pure random patterns to construct imitation test sequences is justified.

Next, we considered the weights and distances for states reached under the test sequences produced by HITEC. The values of $\overline{ST\_D}$ and $\overline{ST\_W}$ for several circuits are shown in Table VI. Under the "s.v" column, we give the number of state variables. We also include in Table VI the values of $\overline{ST\_D}$ and $\overline{ST\_W}$ when pure random patterns are used. It can be see that large relative differences sometimes exist between the state distances and weights created by deterministic and random sequences. For example, for $s1423$, the average distance for HITEC is 14.69, and for pure random patterns, it is 10.06. Consequently, it may be important to incorporate these parameters into the process of generating imitation test sequences.

Studying the distributions of $ST\_D(u)$ and $ST\_W(u)$ given by $N\_ST\_D(n)$ and $N\_ST\_W(n)$ under the test sequences produced by HITEC, we found that there are two types of circuits. The first type is represented by $s5378$ in Fig. 1(a). For this type of circuit, there is a single peak in the graph of $N\_ST\_W(n)$ versus n. The second type of circuits is represented by $s1423$ in Fig. 1(b). For this type of circuit,

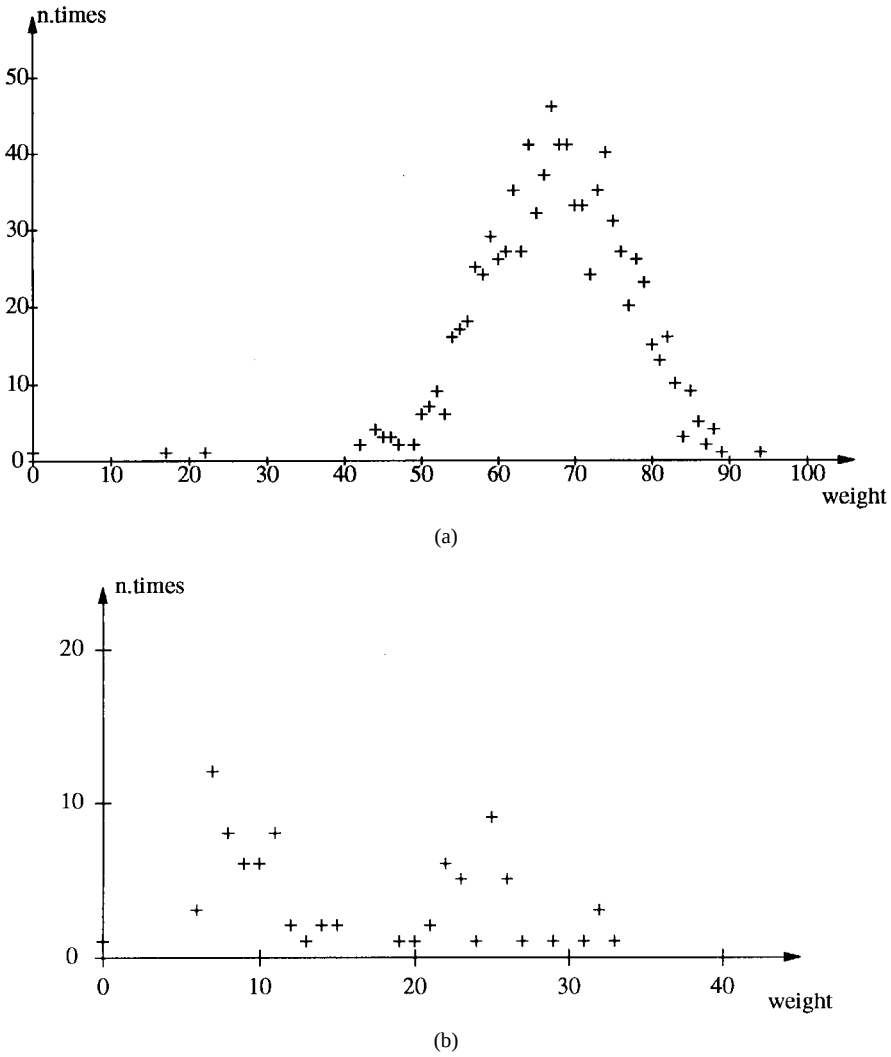

Fig. 1. State weight distributions.

there are several peaks in the graph of $N\_ST\_W(n)$ versus n. Considering pure random sequences, we noted that the forms of the graphs are similar (however, the peaks are located at different weights and distances, and have different heights). We conclude that we need only consider the average weight and distance when incorporating weight and distance information into Procedures 1–4.

Next, we consider the modification of Procedures 1–4 to accommodate distance and weight information. We concentrate on Procedure 2 since it is one of the more effective procedures according to Table II. First, we wanted to check whether incorporating distance and weight information would improve the fault coverage of imitation test sequences. Based on Table VI, we selected a range of distances and weights for each circuit, as illustrated next. For $s400$, the average distance is 2.66 and the average weight is 8.57. When selecting an input pattern that leads into a new state, we impose the following restrictions.

The distance from the current state should be in the range [2, 3], i.e.,

- either between 2 and 3, or
- if it is smaller than 2, it should be as close to 2 as possible, or
- if it is larger than 3, it should be as close to 3 as possible.

Similarly, the weight of a new state should be in the range [8, 9], i.e.,

- either between 8 and 9, or
- if it is smaller than 8, it should be as close to 8 as possible, or
- if it is larger than 9, it should be as close to 9 as possible.

These additional restrictions are added to Step 3 of Procedure 2, as follows. In Step 3a), if an input pattern c can be found that leads into a new state within the specified distance and weight ranges, then c is selected and no additional input patterns are considered. Otherwise, if c leads into a new state but the new state is not within the specified ranges, then c is stored and additional input patterns are considered in Step 3. In Step 4, if an input pattern that leads into a new state was recorded in Step 3, then the one that has a next state which is the closest to the specified ranges of distance and weight is selected. The weight range is given higher priority than the distance range in this case. We do not add similar restrictions on the selection of an input pattern when only a state visited before can be reached since the other heuristics dominate the selection of an input pattern in this case.

TABLE VII
PROCEDURE 2 USING DISTANCES AND WEIGHTS

	w/o d/w		with d/w				
circuit	det	len	det.tg	dist	weight	det	len
s386	274	6742	FASTEST	1-2	1-2	280	263
s400	374	5354	HITEC	2-3	8-9	378	4093
s526	441	7743	HITEC	2-3	6-6	447	9639
s1423	1274	9616	HITEC	14-15	16-16	1362	9597

Several results of the application of Procedure 2 when weight and distance information is added are given in Table VII. We compare the results of the original Procedure 2 (column "w/o d/w") to Procedure 2 with weight and distance information (column "with d/w"). Under the column "with d/w," we give the deterministic test generation procedure whose weights are used, followed by the distance and weight ranges used for imitation test sequence generation, the number of faults detected by imitation test sequences, and the effective test length. It can be seen that weight and distance information can sometimes improve the number of faults detected. For example, six additional faults are detected in $s386$ and $s526$, and 88 additional faults are detected in $s1423$. Note that in the case of $s1423$, the number of faults detected is higher than the number of faults detected by HITEC.

Selection of appropriate weight and distance ranges can be done in several ways. They can be selected based on a deterministic test sequence for a subset of the faults. Alternatively, several ranges of weights and distances can be tried, and the best one can be selected. From Table VI, it appears that the weight is generally larger than the distance, and thus the search space can be restricted accordingly. Next, we describe a procedure similar to binary search for selecting appropriate ranges of weight and distance for a given circuit. A combination of distance in the range $[d_1, d_2]$ and weight in the range $[w_1, w_2]$ is denoted by $\{[d_1, d_2][w_1, w_2]\}$.

Consider a circuit with N state variables. The values for weights and distances for the circuit are in the range $[0, N]$. In the first iteration of the procedure, we divide this range in half, and consider the ranges $[0, N/2]$ and $[N/2+1, N]$ in three combinations, as follows.

$\{[0, N/2][0, N/2]\}$ (i.e., distance in the range $[0, N/2]$ and weight in the range $[0, N/2]$), $\{[0, N/2][N/2+1, N]\}$, and $\{[N/2+1, N][N/2+1, N]\}$.

The combination $\{[N/2+1, N][0, N/2]\}$ is omitted since we restrict ourselves to weights which are not smaller than distances. A test sequence is generated based on each one of the combinations, and its fault coverage is recorded. Each combination of ranges is then partitioned again into three or four combinations. In general, a combination of ranges $\{[d_1, d_2][w_1, w_2]\}$ is divided into the following combinations.

$$\{[d_1, d_1 + (d_1 + d_2)/2][w_1, w_1 + (w_1 + w_2)/2]\},$$
$$\{[d_1, d_1 + (d_1 + d_2)/2][w_1 + (w_1 + w_2)/2 + 1, w_2]\},$$
$$\{[d_1 + (d_1 + d_2)/2 + 1, d_2][w_1, w_1 + (w_1 + w_2/2]\}$$

if

$$w_1 \geq d_1 + (d_1 + d_2)/2 + 1$$

and

$$w_1 + (w_1 + w_2)/2 \geq d_2,$$

and

$$\{[d_1 + (d_1 + d_2)/2 + 1, d_2][w_1 + (w_1 + w_2)/2 + 1, w_2]\}.$$

The process of partitioning the ranges can be represented in the form of a tree, as follows. In the root of the tree, we have the combination of ranges $\{[0, N][0, N]\}$. The successors of the root form level 1 of the tree. They contain the combinations of ranges where the ranges are of length approximately $N/2$, i.e., $\{[0, N/2], [0, N/2]\}$, $\{[0, N/2], [N/2+1, N]\}$, and $\{[N/2+1, N], [N/2+1, N]\}$. In level i of the tree, we have the combinations of ranges where the ranges are of length approximately $N/2^i$. Clearly, if the complete tree is explored, i.e., every combination of ranges is used to define new combinations, then the number of combinations considered grows exponentially. The following criteria are used to select the combinations to be further partitioned.

1) All of the combinations in level 1 of the tree are partitioned. There are three such combinations, and they create 10 combinations in level 2 of the tree.

2) A combination C in level i of the tree, where $i \geq 2$, is partitioned if the following condition is satisfied. Let C be obtained from a combination C' in level $i-1$ of the tree. Let the number of faults detected by an imitation test sequence that uses C be $N_{\rm det}$, and let the number of faults detected by an imitation test sequence that uses C' be $N'_{\rm det}$. Then C is partitioned if $N_{\rm det} > N'_{\rm det}$.

The results obtained by this procedure for $s5378$ are shown in Fig. 2 and Table VIII. $s5378$ has 179 state variables. We use ranges starting from 1 instead of 0; however, this does not affect the results obtained. Fig. 2 shows the first three levels of the tree. Table VIII shows the complete set of results we obtained. In Fig. 2, each combination of ranges has a vertex in the tree. Below the ranges, we show the number of faults detected and the test length. Next to the number of faults detected, we include the symbol $<, >$ or $=$, depending on whether the number of faults detected is smaller than, larger than, or equal to the number of faults detected by the predecessor combination. Only combinations with the symbol $>$ are further partitioned. In Table VIII, two levels of the tree are separated by a double line. Combinations derived from the same predecessor combination are separated by single lines. In the first column of Table VIII, we show the predecessor combination from which the combinations in the second column are obtained. Below the combination in the first column, we show the number of faults detected when it is used for generation of an imitation test sequence. In

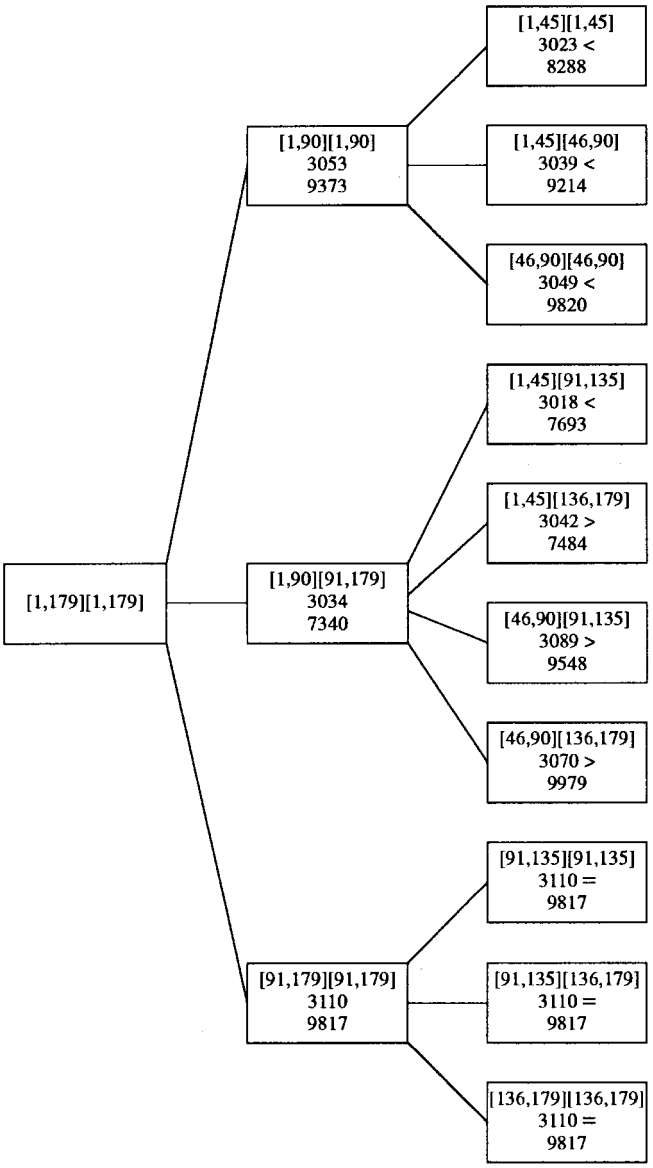

Fig. 2. Part of the tree for $s5378$.

TABLE VIII
RESULTS OF BINARY SEARCH FOR $s5378$ WITH WEIGHTS AND DISTANCES

from	dist	weight	det	len	
1-179,1-179	1-90	1-90	3053	9373	
	1-90	91-179	3034	7340	
	91-179	91-179	3110	9817	
1-90,1-90	1-45	1-45	3023	8288	<
3053	1-45	46-90	3039	9214	<
	46-90	46-90	3049	9820	<
1-90,91-179	1-45	91-135	3018	7693	<
3034	1-45	136-179	3042	7484	>
	46-90	91-135	3089	9548	>
	46-90	136-179	3070	9979	>
91-179,91-179	91-135	91-135	3110	9817	=
3110	91-135	136-179	3110	9817	=
	136-179	136-179	3110	9817	=
1-45,136-179	1-23	136-157	2852	9542	<
3042	1-23	158-179	2852	9542	<
	24-45	136-157	3042	7484	=
	24-45	158-179	3042	7484	=
46-90,91-135	46-68	91-113	3188	9540	>
3089	46-68	114-135	3070	9979	<
	69-90	91-113	3109	8362	>
	69-90	114-135	3190	9903	>
46-90,136-179	46-68	136-157	3070	9979	=
3070	46-68	158-179	3070	9979	=
	69-90	136-157	3190	9903	>
	69-90	158-179	3190	9903	>
46-68,91-113	46-57	91-102	3123	9941	<
3188	46-57	103-113	3045	9970	<
	58-68	91-102	3082	7945	<
	58-68	103-113	3076	9606	<
69-90,91-113	69-79	91-102	3109	8362	=
3109	69-79	103-113	3190	9903	>
	80-90	91-102	3109	9817	=
	80-90	103-113	3109	9817	=
69-90,114-135	69-79	114-124	3190	9903	=
3190	69-79	125-135	3190	9903	=
	80-90	114-124	3109	9817	<
	80-90	125-135	3109	9817	<
69-90,136-157	69-79	136-146	3190	9903	=
3190	69-79	147-157	3190	9903	=
	80-90	136-146	3109	9817	<
	80-90	147-157	3109	9817	<
69-90,158-179	69-79	158-168	3190	9903	=
3190	69-79	169-179	3190	9903	=
	80-90	158-168	3109	9817	<
	80-90	169-179	3109	9817	<
69-79,103-113	69-74	103-108	3189	9817	<
3190	69-74	109-113	3189	9817	<
	75-79	103-108	3111	9817	<
	75-79	109-113	3111	9817	<

each row, we include a combination of ranges, the number of faults detected when it is used, and the test length. In the last column, we use the symbol $<, >$ or $=$, depending on whether the number of faults detected is smaller than, larger than, or equal to the number of faults detected by the predecessor combination (shown in the first column of the table). For example, consider the first row in the table, corresponding to the combination $\{[1,90][1,90]\}$. This range is obtained by partitioning the range $\{[1,179][1,179]\}$. The number of faults detected when using the combination $\{[1,90][1,90]\}$ is 3053 and the test length is 9373. This range appears again in the first column under the first double line. It is divided there into the ranges $\{[1,45][1,45]\}, \{[1,45][46,90]\}$ and $\{[46,90][46,90]\}$. For the combination $\{[1,45][1,45]\}$, the number of faults detected is 3023. This is smaller than the number of faults detected by the combination $\{[1,90][1,90]\}$, which is 3053. The combination $\{[1,45][136,179]\}$ is the first one where the number of faults detected exceeds the number of faults detected by its predecessor, and it is the first one that is partitioned under the second double line in Table VIII. We terminate the search when the fault coverage does not improve for any combination. In Table VIII, the fault coverage increases from 3059, which we obtained before without using distance and weight information, to 3190, obtained, e.g., for the combination $\{[69,90][114,135]\}$. This number of faults detected is higher than that of any of the procedures reported in Table III. It is possible to explore additional combinations that were not partitioned in Table VIII; however, we did not consider this option.

VI. CONCLUDING REMARKS

We presented a method to generate test sequences that detect large numbers of faults using logic simulation. The generated sequences can be used alone or as prefixes of deterministic

test sequences. To generate the sequences, we studied the test sequences generated by several deterministic test generation procedures. We showed that under these test sequences, the fault-free circuits go through sequences of state transitions that have distinct characteristics. By using logic simulation on the fault-free circuit and considering a number of random patterns as candidates for inclusion in the test sequence at every time unit, we were able to make the fault-free circuit go through a sequence of state transitions that has characteristics very similar to the sequence of state transitions it goes through under deterministic test sequences. Fault simulation of these input sequences showed that the fault coverage achieved is very close to the fault coverage achieved by deterministic sequences and higher than the fault coverage achieved by other methods.

The approach described above is a special case of the following more general paradigm. Consider two problems P_1 and P_2 with algorithms A_1 and A_2 such that the complexity of A_1 is larger than the complexity of A_2, and such that P_2 accepts as input the results of P_1. In this work, P_1 was the test generation problem, and P_2 was the logic simulation problem. Let A_1 be used to produce solutions $s_1, s_2, \cdots, s_k$ to k different instances of P_1. Let A_2 be applied to the same instances and their solutions with respect to $P_1, s_1, s_2, \cdots, s_k$. In this case, A_2 produces new (typically compressed) information regarding $s_1, s_2, \cdots, s_k$. In this work, by applying logic simulation to input sequences generated by deterministic test generation procedures, we found the sequences of states of the fault-free circuit. An algorithm A_3 of complexity similar to A_2 can be devised that generates solutions having similar characteristics to $s_1, s_2, \cdots, s_k$ when A_2 is applied to them. In this work, A_3 was the proposed imitation test generation procedure. Our results in this work show that such an algorithm A_3 can be effective in generating solutions for unstudied instances of P_1.

ACKNOWLEDGMENT

The authors would like to thank Prof. J. Patel for providing test sequences generated by HITEC and benchmark circuits studied in [18], and Prof. K. Saluja for providing test sequences generated by FASTEST.

REFERENCES

[1] S. Mallela and S. Wu, "A sequential test generation system," in *Proc. Int. Test Conf.*, Sept. 1985, pp. 57–61.
[2] R. Marlett, "An effective test generation system for sequential circuits," in *Proc. Design Automation Conf.*, June 1986, pp. 250–256.
[3] H.-K. T. ma, S. Devadas, A. R. Newton, and A. S-Vincentelli, "Test generation for sequential circuits," *IEEE Trans. Computer-Aided Design*, pp. 1081–1093, Oct. 1988.
[4] W-T. Cheng, "The back algorithm for sequential test generation," in *Proc. Int. Conf. Comput. Design*, Oct. 1988, pp. 66–69.
[5] V. D. Agrawal, K. T. Cheng, and P. Agrawal, "A directed search method for test generation using concurrent simualto," *IEEE Trans. Computer-Aided Design*, pp. 131–138, Feb. 1989.
[6] W-T. Cheng and T. J. Chakraborty, "Gentest: An automatic test generation system for sequential circuits," *IEEE Computer*, pp. 43–49, Apr. 1989.
[7] T. P. Kelsey and K. K. Saluja, "Fast test generation for sequential circuits," in *Proc. Int. Conf. Computer-Aided Design*, Nov. 1989, pp. 354–357.
[8] A. Ghosh, S. Devadas, and A. R. Newton, "Test generation and verification for highly sequential circuits," *IEEE Trans. Computer-Aided Design*, pp. 952–667, May 1991.
[9] T. Niermann and J. H. Patel, "HITEC: A test generation package for sequential circuits," in *Proc. European Design Automation Conf.*, 1991, pp. 214–218.
[10] D.-H. Lee and S. M. Reddy, "A new test generation method for sequential circuits," in *Proc. 1991 Int. Conf. Computer-Aided Design*, Nov. 1991, pp. 446–449.
[11] D. G. Saab, Y. G. Saab, and J. A. Abraham, "CRIS: A test cultivation program for sequential VLSI circuits," in *Proc. Int. Conf. Computer-Aided Design*, Nov. 1992, pp. 216–219.
[12] H. Cho, S.-W. Jeong, F. Somenzi, and C. Pixley, "Synchronizing sequences and symbolic traversal techniques in test generation," *JETTA*, pp. 19–31, Feb. 1993.
[13] H. Cho, G. D. Hachtel, and F. Somenzi, "Redundancy identification/removal and test generation for sequential circuits using implicit state enumeration," *IEEE Trans. Computer-Aided Design*, pp. 935–945, July 1993.
[14] I. Pomeranz and S. M. Reddy, "Application of homing sequences to synchronous sequential circuit testing," *IEEE Trans. Comput.*, pp. 569–580, May 1994.
[15] E. M. Rudnick, J. H. Patel, G. S. Greenstein, and T. M. Niermann, "Sequential circuit test generation in a genetic algorithm framework," in *Proc. Design Automation Conf.*, June 1994, pp. 698–704.
[16] I. Pomeranz and S. M. Reddy, "The multiple observation time test strategy," *IEEE Trans. Comput.*, pp. 627–637, May 1992.
[17] T. M. Niermann, W.-T. Cheng, and J. H. Patel, "PROOFS: A fast, memory-efficient sequential circuit fault simulator," *IEEE Trans. Computer-Aided Design*, pp. 198–207, Feb. 1992.
[18] E. M. Rudnick, "Simulation based techniques for sequential circuits," Ph.D. dissertation; also, Tech. Rep. CRHC-94-14, Center for Reliable and High Performance Computing, Univ. Illinois, Aug. 1994.
[19] T. E. Marchok, A. El-Maleh, W. Maly, and J. Rajski, "Complexity of sequential ATPG," in *Proc. European Design and Test Conf.*, Mar. 1995, pp. 252–261.
[20] I. Pomeranz and S. M. Reddy, "On static compaction of test sequences for synchronous sequential circuits," in *Proc. 33rd Design Automation Conf.*, June 1996, pp. 215–220.

Irith Pomeranz (M'89–SM'96) received the B.Sc. degree (summa cum laude) in computer engineering and the D.Sc degree from the Department of Electrical Engineering, Technion–Israel Institute of Technology, Haifa, in 1985 and 1989, respectively.

From 1989 to 1990, she was a Lecturer in the Department of Computer Science at the Technion. In 1990, she joined the Department of Electrical and Computer Engineering, University of Iowa, where she is currently an Associate Professor. She is an Associate Editor of the *ACM Transactions on Design Automation*. Her research interests are testing of VLSI circuits, design for testability, synthesis, and design verification.

Dr. Pomeranz is a recipient of the NSF Young Investigator Award, and is a Guest Editor of the IEEE TRANSACTIONS ON COMPUTERS Special Issue on Dependability of Computing Systems, 1997.

Sudhakar M. Reddy (S'68–M'68–SM'84–F'87) received the undergraduate degree in electrical and communication engineering from Osmania University, the M.S. degree from the Indian Institute of Science, and the Ph.D. degree in electrical engineering from the University of Iowa, Iowa City.

He has been active in the areas of testable designs and test generation for logic circuits since 1972. He has been an Associate Editor and twice a Guest Editor of the IEEE TRANSACTIONS ON COMPUTERS. Since 1968, he has been a member of the Faculty of the Department of Electrical and Computer Engineering, University of Iowa, where he is currently the Department Chairman. In 1990, he was made a University of Iowa Foundation Distinguished Professor.

Dr. Reddy is an Associate Editor of the IEEE TRANSACTIONS ON COMPUTER-AIDED DESIGN OF INTEGRATED CIRCUITS AND SYSTEMS, and is a member of Tau Beta Pi, Eta Kappa Nu, and Sigma Xi.