Restoration-Based Procedures With Set Covering Heuristics for Static Test Compaction of Functional Test Sequences

Irith Pomeranz, *Fellow, IEEE*

***Abstract*— The goal of static test compaction is to reduce the number or tests, or the lengths of test sequences, without reducing the fault coverage. Static test compaction that reduces the number of tests was formulated as a set covering problem in order to benefit from the heuristics that exist for solving this problem. This paper applies set covering concepts and heuristics to static test compaction that reduces the length of a functional test sequence. Although set covering is not applicable directly to a single test sequence, it provides a theoretical framework and justification for a particular set of heuristics. The procedure uses a parameter denoted by n to determine the computational effort for computing the sets that are used for making compaction decisions. With $n = 1$, the procedure is equivalent to a static test compaction procedure that does not use set covering. Experimental results demonstrate that shorter test sequences are obtained for $n > 1$ than for $n = 1$. A variation of the static test compaction procedure that produces a monotonic decrease in test sequence length with n is also described.**

***Index Terms*— Functional test sequence, set covering, single stuck-at faults, static test compaction, transition faults.**

I. Introduction

FUNCTIONAL test sequences are applied to a circuit in its functional mode of operation. They have several advantages over structural (scan-based) tests. Functional test sequences avoid overtesting due to nonfunctional operation conditions [1]–[3]. They can also detect defects that are not detected by scan-based tests [4]. Functional test sequences are also useful for speed-binning and for simulation-based design verification. To facilitate at-speed application of functional test sequences, the sequences can be generated on-chip [5]–[8], or compacted and compressed sequences can be stored on-chip.

Compression of functional test sequences can be accomplished by encoding of their test vectors [9]. Test compaction for functional test sequences was considered in [10] to [25]. These procedures include dynamic test compaction procedures, which incorporate test compaction heuristics into the test generation process, and static test compaction procedures, which are applied after test generation is complete. Static test compaction procedures include ones that attempt to compact a set of sequences, as well as ones that attempt to compact a single sequence. This paper focuses on static test compaction of a single test sequence. Even when the goal is to produce a set of sequences, it is advantageous to compact each sequence individually. In addition, merging of test sequences in a set into a smaller number of sequences, and compacting each merged sequence, can reduce the overall length of the test sequences in the set. Static test compaction of a single test sequence also has a special role in the generation of a functional test sequence for the following reason.

The generation of a functional test sequence by a branch and bound process is computationally expensive. Therefore, simulation-based test generation procedures are used for addressing the cost of test generation. These procedures typically produce test sequences that are longer than necessary. Therefore, static test compaction can reduce the test sequence length significantly. Static test compaction can be applied after the sequence has been generated in full. It can also be applied after every test generation step where the test sequence is extended to detect additional faults [26]. In this application, the static test compaction procedures based on [14] and [17] have the added advantage that they can increase the fault coverage accidentally.

For scan-based tests, the static test compaction procedures that can produce the smallest test sets, without modifying any test, are based on set covering [27]–[30]. Given a test set T for a full-scan circuit, such that T detects a set of target faults F, the goal of set covering is to select the smallest subset $C \subseteq T$ such that C detects all the faults from F. A typical set covering procedure performs fault simulation without fault dropping of F under T in order to associate, with every fault $f_j \in F$, a set of tests $T_j \subseteq T$ that detect f_j. The procedure starts from $C = \emptyset$, and F that includes all the target faults. For every fault $f_j \in F$, if T_j includes a single test, the procedure adds the test to C and removes from F the faults that it detects. In general, in an arbitrary step, the procedure selects the fault $f_j \in F$ with the smallest set T_j. For every $t_i \in T_j$, the procedure finds the number of additional faults that will be detected if t_i is added to C. The procedure selects the test t_i for which this number is the largest. It adds t_i to T, and removes from F the faults that it detects. This continues until all the faults in F are detected.

Set covering is also applicable to a set of functional test sequences, where it selects a subset of sequences that detects the same faults as the complete set [24].

The goal of this paper is to apply the concepts and heuristics of set covering to the static test compaction of a single functional test sequence. Although set covering is not applicable

Manuscript received August 13, 2012; revised January 10, 2013; accepted March 14, 2013. Date of publication April 18, 2013; date of current version March 18, 2014.

The author is with the School of Electrical and Computer Engineering, Purdue University, West Lafayette, IN 47907 USA (e-mail: pomeranz@ecn.purdue.edu).

Digital Object Identifier 10.1109/TVLSI.2013.2253499

directly to a single test sequence, the motivation for this paper is the following.

Static test compaction procedures for single test sequences are based on heuristics that are expected to lead to short test sequences. Set covering provides a theoretical framework and justification for a particular set of heuristics. These heuristics were shown to be effective in the context of full-scan circuits, and for sets of test sequences. Experimental results presented in this paper demonstrate that the use of set covering heuristics to enhance a static test compaction procedure for a single test sequence leads to shorter test sequences than those generated by the same procedure when it is not enhanced by set covering heuristics.

A set covering procedure is built in this paper on top of the static test compaction procedure referred to as restoration-based [17]. This procedure is the basis for other static test compaction procedures. In the implementation described in [21], static test compaction has the same computational cost as fault simulation. The restoration-based procedure that includes set covering heuristics is referred to as a restoration and set covering (R&SC) based procedure.

The R&SC-based procedure uses a parameter denoted by n to determine the computational effort for computing the sets that it uses for making compaction decisions. With $n = 1$, the procedure is equivalent to a restoration-based procedure that does not use set covering. Experimental results demonstrate that shorter test sequences are obtained for $n > 1$. However, the reduction in test sequence length is not guaranteed to be monotonic with n. To address this issue, this paper also describes a variation of the R&SC-based procedure that produces a monotonic decrease in the test sequence length as n is increased. This variation is based on the restoration-based procedure from [25]. It makes random decisions regarding the initialization of the restoration process in order to allow it to reduce the test sequence length over a larger number of iterations. During these iterations, the procedure increases the value of n gradually in order to benefit from more accurate compaction decisions. The first procedure described in this paper is referred to as deterministic (or nonrandom) since it does not make any random decisions. The second procedure is characterized by the random initialization it uses.

The rest of this paper is organized as follows. Section II describes the sets that the R&SC-based procedure uses to make test compaction decisions. Section III describes the deterministic R&SC-based procedure. Section IV presents experimental results of this procedure. Section V describes the procedure with random initialization decisions. Section VI presents experimental results of this procedure. Section VII discusses other options related to restoration-based procedures that can be enhanced with set covering heuristics, and presents additional experimental results.

II. Sets for Set Covering

This section describes the sets that the R&SC-based procedure uses. The description considers a functional test sequence T of length L for a set of target faults F. Let $T = T(0)T(1)\dots T(L-1)$, where $T(u)$ is the test vector at time unit u of T, for $0 \leq u < L$.

TABLE I
Example Sequence

u	T	omit	T_1	T_2
0	1111	1	1110	1110
1	1110	1	0100	0100
2	1100	1	0100	0100
3	0100	1	1011	1011
4	0100	0	1110	1001
5	1011	0	1001	0000
6	1110	0	0000	1001
7	1001	0	1001	1000
8	1000	1	1000	0110
9	1101	1	0110	0001
10	0000	1	0001	0000
11	0010	1	0000	0111
12	0110	1	0111	1011
13	0110	1	0000	0011
14	0000	1	1011	
15	1001	1	0011	
16	1000	1		
17	1010	1		
18	0110	1		
19	0110	0		
20	0001	0		
21	0101	1		
22	0000	0		
23	1101	1		
24	0001	1		
25	0110	1		
26	0111	0		
27	0000	1		
28	1011	0		
29	0011	0		

For every fault $f_j \in F$, the procedure used in this paper finds a set of time units where f_j is detected by T. The sets can be obtained by fault simulation of F under T without fault-dropping. This implies that a fault $f_j \in F$ is simulated under the complete test sequence T even after it is already detected.

However, the computational complexity of fault simulation without fault-dropping is high. This is especially true here since sequential fault simulation is required. Therefore, in this paper, fault simulation without fault-dropping is replaced with n-detection fault simulation. This fault simulation procedure drops a fault from consideration after it is detected at n different time units of T. The set of time units, where a fault $f_j \in F$ is detected before it is dropped by the n-detection fault simulation procedure, is denoted by $U_n(f_j)$. With $n = 1$, the n-detection fault simulation procedure is identical to a fault simulation procedure with fault-dropping. In this case, the R&SC-based procedure will be identical to a restoration-based procedure that does not use set covering. With $n > 1$, the procedure will benefit from more information about the options for detecting the faults in F. This will allow it to make better test compaction decisions.

For illustration, a functional test sequence of length $L = 30$ for ISCAS-89 benchmark $s27$ is shown in Table I under column T. For every time unit u, column T shows the test

TABLE II
DETECTION TIME UNITS

	T		T_1	T_2
j	$U_1(f_j)$	$U_4(f_j)$	$U_4(f_j)$	$U_4(f_j)$
0	5	5 23 24 25	3	3 4 5
1	15	15 20 21 22	7 10 11 12	6 9 10 11
2	15	15 20 21 22	7 10 11 12	6 9 10 11
3	7	7	5 6	4 5
4	4	4 11 12 13	2	2
5	5	5	3	3 4 5
6	15	15 20 21 22	7 10 11 12	6 9 10 11
7	7	7 8 9 10	5 6 8 9	4 5 7 8
8	15	15 20 21 22	7 10 11 12	6 9 10 11
9	5	5 23 24 25	3	3 4 5
10	15	15 28 29	7 14 15	6
11	15	15 20 21 22	7 10 11 12	6 9 10 11
12	15	15 20 21 22	7 10 11 12	6 9 10 11
13	16	16 17 18 19	8 9	7 8
14	21	21 22	11 12 13	10 11
15	4	4 5 6 7	2 3 4 5	2 3 4 5
16	5	5	3	3 4 5
17	15	15 20 21 22	7 10 11 12	6 9 10 11
18	22	22	11 12 13	10 11
19	21	21 22	12 13	11
20	5	5 23 24 25	3	3 4 5
21	14	14 16 17 18	8 9	7 8
22	4	4 5 6 7	2 3 4 5	2 3 4 5
23	15	15 20 21 22	7 10 11 12	6 9 10 11
24	0	0 1 2 3	0 1 2 3	0 1 2 3
25	29	29	15	13
26	21	21 22	11 12 13	10 11
27	4	4 11 12 13	2	2
28	7	7	5 6	4 5
29	15	15 20 21 22	7 10 11 12	6 9 10 11
30	0	0 1 2 3	0 1 2 3	0 1 2 3
31	15	15 20 21 22	7 10 11 12	6 9 10 11

vector $T(u)$. The other columns of Table I will be used later. Column T of Table II shows the results of n-detection fault simulation of T for $n = 1$ and 4. For every fault $f_j \in F$, Table II shows the sets $U_1(f_j)$ and $U_4(f_j)$ of time units where f_j is detected by T. For example, using $n = 1$ shows that f_0 is detected at $u = 5$, and the fault is dropped after it is detected. Using $n = 4$, f_0 is simulated until it is detected at four time units, $u = 5$, 23, 24, and 25. The fault f_3 is detected only once by T, at time unit $u = 7$.

III. DETERMINISTIC (NONRANDOM) RESTORATION AND SET COVERING-BASED PROCEDURE

This section describes the deterministic R&SC-based procedure for an arbitrary value of n.

Given a test sequence $T = T(0)T(1)\dots T(L-1)$, a restoration-based procedure starts by omitting some or all of the test vectors from T. It then restores test vectors only as necessary for restoring the detection of target faults. All the faults are considered repeatedly in the same order until all of them are detected. Repetition is needed for the following reason.

Suppose that, before considering a fault f_{j1}, the subsequence $T(u_0)\dots T(u_1)$ is restored into T. Suppose that f_{j1} is detected after restoring the subsequence $T(u_4)\dots T(u_5)$, where $u_1 < u_4$. This implies that the subsequence $T(u_0)\dots T(u_1)T(u_4),\dots T(u_5)$ detects f_{j1}. Next, suppose that, in order to restore the detection of a fault f_{j2}, the subsequence $T(u_2)\dots T(u_3)$ is restored into T, and $u_1 < u_2 < u_3 < u_4$. This breaks the subsequence $T(u_0)\dots T(u_1)T(u_4)\dots T(u_5)$ that detects f_{j1}, and results in $T(u_0)\dots T(u_1)T(u_2)\dots T(u_3)T(u_4)\dots T(u_5)$. If this subsequence does not detect f_{j1}, f_{j1} needs to be considered again, and additional vectors need to be restored in order to detect it. Repetition of the restoration process ensures that all the faults will eventually be detected.

After all the faults are detected, test vectors that were not restored are omitted permanently from the sequence. This process is repeated until an iteration in which all the test vectors must be restored in order to detect all the target faults.

Different restoration-based procedures differ in the vectors they initially omit, in the order by which they consider the faults for restoration, and in the way they form the compacted test sequence from the restored vectors. The procedure described in this section initially omits all the test vectors from T. It keeps track of the omitted vectors by associating a variable denoted by omit(u) with every time unit u, for $0 \le u < L$. The procedure sets omit(u) = 1 for $0 \le u < L$ initially to indicate that all the test vectors are omitted. To restore the test vector at time unit u, it sets omit(u) = 0.

The procedure considers the faults for restoration one at a time. The order by which the faults are considered is based on two parameters.

1) The set of detection time units $U_n(f_j)$ is analogous to the set T_j of tests that detect f_j in a test set that consists of several different tests. Set covering heuristics indicate that it is preferred to consider earlier a fault with a smaller set. With a smaller set $U_n(f_j)$, there are fewer options for restoring test vectors in order to detect f_j. Therefore, f_j is less likely to be detected accidentally when other faults are considered. As a result, it is likely to require test vectors to be restored when it is eventually considered. In contrast, a fault with a higher number of detection time units has more options for detection. As a result, it is more likely to be accidentally detected when other faults are considered. It is thus advantageous to consider the faults with the smaller sets $U_n(f_j)$ earlier, in order to restore test vectors for them earlier, and allow faults with larger sets $U_n(f_j)$ to be detected accidentally without restoring additional vectors.
2) One of the restoration-based procedures from [17] considers the faults by decreasing order of their first detection time unit. This detection time unit is obtained by fault simulation with fault-dropping in [17]. Here, it is the first entry in $U_n(f_j)$. This order is based on the expectation that a fault with a higher first detection time unit is harder to detect accidentally. With the same rationale as in 1), it is preferred to target such a fault earlier.

Using these parameters, the R&SC-based procedure considers the faults by increasing number of detection time units. For faults with the same number, the procedure considers the faults by decreasing order of the first detection time unit.

For example, using the sets $U_1(f_j)$ from Table II, all the faults have the same number of detection time units, equal to 1. Using the first detection time unit, the faults are considered in the following order: Faults whose first detection time unit is $u = 29$, f_{25}; faults whose first detection time unit is $u = 22$, f_{18}; faults whose first detection time unit is $u = 21$, f_{14}, f_{19}, and f_{26}; and so on.

Using the sets $U_4(f_j)$ from Table II, the faults are considered in the following order: Faults with a single detection time unit, in the order f_{25}, f_{18}, f_3, f_{28}, f_5, and f_{16}; faults with two detection time units, in the order f_{14} f_{19}, and f_{26}; and so on.

The order for $n = 4$ is different from that for $n = 1$. It is expected to lead to a shorter test sequence since it is based on more accurate information. Specifically, it will allow more difficult to detect faults to be considered earlier.

When a fault f_j is considered for restoration, the procedure checks whether f_j is already detected by T with the current omitted vectors. If the fault is detected, it is not considered further. Otherwise, the procedure has the option of restoring test vectors around every one of the time units in $U_n(f_j)$. The R&SC-based procedure considers every one of these options. It then selects the one that yields the largest number of omitted vectors. This is again analogous to the case of set covering for a test set that consists of several tests, where the procedure selects the test in T_j that detects the largest number of faults from F.

When $u_k \in U_n(f_j)$ is considered, the goal is to restore some or all of the vectors preceding it (including u_k itself) in order to restore the detection of f_j. Restoration of test vectors in order to detect f_j proceeds as follows: The procedure considers time units $u = u_k, u_{k-1}, \ldots$, until f_j is detected. When time unit u is considered, if $\text{omit}(u) = 1$, the procedure sets $\text{omit}(u) = 0$, thus restoring $T(u)$. It then simulates f_j to check if it is detected. In the worst case, f_j will be detected after all the test vectors at time units $u_k, u_{k-1}, \ldots, 0$ are restored. In practice, it is typically sufficient to restore fewer test vectors in order to detect f_j. If the test vector at time unit u is restored, the procedure includes time unit u in a set denoted by R.

Next, the procedure checks if any of the test vectors at the time units included in R can be omitted without losing the detection of f_j. For every $u \in R$, the procedure sets $\text{omit}(u) = 1$ and simulates f_j. If f_j is not detected, the procedure restores the vector by setting $\text{omit}(u) = 0$. Otherwise, the vector remains omitted.

For u_k, $O(u_k)$ denotes the number of omitted vectors in T after the detection of f_j is restored. Of all the time units in $U_n(f_j)$, the procedure selects the one for which $O(u_k)$ is the highest. It uses this solution to restore the detection of f_j.

The following example illustrates the restoration process for $s27$ with the test sequence shown in Table I under column T, and $n = 4$. Initially, all the test vectors are omitted, and the number of omitted vectors is 30. The procedure considers the faults as follows.

TABLE III
STUCK-AT FAULT COVERAGE

Circuit	Gates	Flts	T_{ex}	T_{rs}	T_{rs+ex}
s208	96	215	63.72	63.72	63.72
s298	119	308	86.04	86.04	86.04
s344	169	342	96.20	96.20	96.20
s382	158	399	91.23	89.47	91.23
s386	159	384	81.77	81.77	81.77
s420	196	430	41.63	41.63	41.63
s526	193	555	81.80	80.18	81.80
s641	380	467	86.51	86.51	86.51
s820	289	850	95.76	60.12	95.76
s1196	529	1242	99.76	97.10	99.76
s1423	657	1515	93.33	81.85	93.33
s5378	2779	4603	79.06	73.04	79.06
s35932	16385	39094	89.78	89.78	89.78
b03	145	452	73.89	73.89	73.89
b04	468	1346	86.78	86.78	86.78
b05	575	1816	59.25	59.25	59.25
b07	380	1183	70.75	70.75	70.75
b08	158	489	94.68	94.68	94.68
b09	129	420	81.19	81.19	81.19
b10	160	512	91.21	90.43	91.21
b11	358	1089	92.19	89.07	92.19
b14	3426	9981	88.13	78.68	88.13
des_area	3943	9933	-	99.87	99.87
i2c	881	2337	-	63.59	63.59
simple_spi	808	2101	-	56.45	56.45
usb_phy	529	1300	-	62.92	62.92
wb_dma	3467	9097	-	67.60	67.60

For f_{25} with $U_4(f_{25}) = \{29\}$, the procedure restores the test vectors at time units 29, 28, 27, and 26, and then omits the vector at time unit 27. The number of omitted vectors is 27.

For f_{18} with $U_4(f_{18}) = \{22\}$, the procedure restores the test vectors at time units 22, 21, 20, and 19, and then omits the vector at time unit 21. The number of omitted vectors is 24.

For f_3 with $U_4(f_3) = \{7\}$, the procedure restores the test vectors at time units 7 and 6, and does not omit any test vector. The number of omitted vectors is 22.

The fault f_{28} is considered next. The fault is already detected, and no restoration is performed for it.

For f_5 with $U_4(f_5) = \{5\}$, the procedure restores the test vectors at time units 5 and 4, and does not omit any test vector. The number of omitted vectors is 20.

The faults f_{16}, f_{14}, f_{19}, and f_{26} are considered next. These faults are already detected, and no restoration is performed for them.

When f_{10} is considered, the omitted vectors are as shown in Table I under column omit. With $U_4(f_{10}) = \{15, 28, 29\}$, the procedure considers three options.

With $u_0 = 15$, it restores the test vectors at time units 15 and 14, and does not omit any test vector. The number of omitted vectors is $O(15) = 18$.

With $u_1 = 28$, it restores the test vector at time unit 27, and does not omit any test vector (the test vector at time unit 28 was already restored earlier). The number of omitted vectors is $O(28) = 19$.

With $u_2 = 29$, it restores the test vector at time unit 27, and does not omit any test vector (the test vectors at time units 29

TABLE IV
STUCK-AT FAULTS, TEST SEQUENCE LENGTH

Circuit	T_{ex}	T_{rs}	T_{rs+ex}	$T_{drc.1}$	$T_{drc.2}$	$T_{drc.4}$	$T_{drc.8}$	$T_{drc.16}$	$T_{drc.32}$
s208	105	4095	4095	144	140	140	134	134	129
s298	117	2099	2099	110	106	99	93	109	80
s344	57	1873	1873	60	60	60	56	54	56
s382	516	3269	3457	731	676	703	634	640	640
s386	121	3940	3940	156	128	131	113	116	108
s420	108	2813	2813	122	120	113	103	102	96
s526	1006	7179	8163	1245	1285	1095	1060	1114	1060
s641	101	2790	2790	105	99	92	94	85	85
s820	491	8263	8749	405	388	382	393	387	380
s1196	238	8323	8561	266	238	229	214	213	213
s1423	1024	8511	9535	845	643	647	742	649	585
s5378	646	9127	9773	967	883	632	853	519	836
s35932	150	3123	3123	156	133	118	113	126	116
b03	130	2458	2458	113	92	93	85	93	91
b04	168	4843	4843	213	159	146	137	142	122
b05	321	2034	2034	213	213	211	211	212	213
b07	380	2734	2734	226	220	186	181	178	187
b08	415	5475	5475	355	354	353	352	334	298
b09	265	5214	5214	452	302	378	310	315	315
b10	190	4825	4996	184	171	164	144	118	118
b11	554	7309	7677	371	325	327	326	323	342
b14	5611	8727	14338	5358	5183	5197	5165	5161	5165
des_area	-	920	920	236	267	245	265	267	236
i2c	-	8705	8705	249	177	178	168	175	172
simple_spi	-	8635	8635	783	820	762	699	782	778
usb_phy	-	8170	8170	1040	848	833	838	845	875
wb_dma	-	8371	8371	572	575	603	579	564	582

and 28 were already restored earlier). The number of omitted vectors is $O(29) = 19$.

The procedure selects to restore test vectors for f_{10} based on $u_1 = 28$, and obtains 19 omitted test vectors.

The procedure continues to restore test vectors based on f_{13}, f_0, and f_4. A second iteration over all the faults shows that all the faults are detected. The procedure omits the test vectors with $\text{omit}(u) = 1$ to obtain the test sequence shown in Table I under column T_1.

The procedure recomputes the sets $U_4(f_j)$ based on $T_1(u)$. The sets are shown in Table II under column T_1. It then applies the same process to T_1. In this case, it restores test vectors based on f_{25}, f_0, f_5, f_4, f_{19}, f_{13}, f_3, and f_{18} before all the faults are detected. The resulting test sequence is shown in Table I under column T_2.

Repeating the process a third time does not reduce the test sequence length any further, and the procedure terminates. The sets $U_4(f_j)$ obtained for T_2 are shown under the corresponding column of Table II.

The deterministic R&SC-based procedure is summarized next.

Procedure 1: Deterministic restoration and set covering.

1) Let T be a given test sequence of length L. Let F be the set of target faults.
2) Assign $\text{omit}(u) = 0$ for $0 \le u < L$.
3) Perform n-detection fault simulation of F under T. For every $f_j \in F$, let $U_n(f_j)$ be the set of time units where f_j is detected by T.
4) Define an ordered set of faults F_{ord} where faults appear by increasing size of $U_n(f_j)$. For the same size of $U_n(f_j)$, the faults appear by decreasing first entry of $U_n(f_j)$.
5) Assign $\text{omit}(u) = 1$ for $0 \le u < L$.
6) Set restore $= 0$. For every fault $f_j \in F_{\text{ord}}$, considering the faults by the order they appear in F_{ord}, if f_j is not detected by T.
   a) Set restore $= 1$.
   b) For every $u_k \in U_n(f_j)$.
      i) Assign $\hat{\text{omit}}(u) = \text{omit}(u)$ for $0 \le u < L$.
      ii) Call Procedure restore(f_j, u_k).
      iii) Assign to $O(u_k)$ the number of time units u such that $\text{omit}(u) = 1$.
      iv) Assign $\text{omit}(u) = \hat{\text{omit}}(u)$ for $0 \le u < L$.
   c) Select the time unit $u_k \in U_n(f_j)$ for which $O(u_k)$ is the highest. Call Procedure restore(f_j, u_k).
7) If restore $= 1$, go to Step 6.
8) Remove from T every test vector $T(u)$ such that $\text{omit}(u) = 1$. If any test vector is removed, go to Step 2.

Procedure restore(f_j, u_k)*:*

1) Assign $R = \emptyset$.
2) For $u = u_k, u_{k-1}, \ldots$, as long as f_j is not detected by T, if $\text{omit}(u) = 1$, set $\text{omit}(u) = 0$ and add u to R.
3) For every $u \in R$, if f_j is detected after assigning $\text{omit}(u) = 1$, assign $\text{omit}(u) = 1$.

TABLE V
STUCK-AT FAULTS, RELATIVE LENGTH

Circuit	$T_{drc.1}$	$T_{drc.2}$	$T_{drc.4}$	$T_{drc.8}$	$T_{drc.16}$	$T_{drc.32}$
s208	1.00	0.97	0.97	0.93	0.93	0.90
s298	1.00	0.96	0.90	0.85	0.99	0.73
s344	1.00	1.00	1.00	0.93	0.90	0.93
s382	1.00	0.92	0.96	0.87	0.88	0.88
s386	1.00	0.82	0.84	0.72	0.74	0.69
s420	1.00	0.98	0.93	0.84	0.84	0.79
s526	1.00	1.03	0.88	0.85	0.89	0.85
s641	1.00	0.94	0.88	0.90	0.81	0.81
s820	1.00	0.96	0.94	0.97	0.96	0.94
s1196	1.00	0.89	0.86	0.80	0.80	0.80
s1423	1.00	0.76	0.77	0.88	0.77	0.69
s5378	1.00	0.91	0.65	0.88	0.54	0.86
s35932	1.00	0.85	0.76	0.72	0.81	0.74
b03	1.00	0.81	0.82	0.75	0.82	0.81
b04	1.00	0.75	0.69	0.64	0.67	0.57
b05	1.00	1.00	0.99	0.99	1.00	1.00
b07	1.00	0.97	0.82	0.80	0.79	0.83
b08	1.00	1.00	0.99	0.99	0.94	0.84
b09	1.00	0.67	0.84	0.69	0.70	0.70
b10	1.00	0.93	0.89	0.78	0.64	0.64
b11	1.00	0.88	0.88	0.88	0.87	0.92
b14	1.00	0.97	0.97	0.96	0.96	0.96
des_area	1.00	1.13	1.04	1.12	1.13	1.00
i2c	1.00	0.71	0.71	0.67	0.70	0.69
simple_spi	1.00	1.05	0.97	0.89	1.00	0.99
usb_phy	1.00	0.82	0.80	0.81	0.81	0.84
wb_dma	1.00	1.01	1.05	1.01	0.99	1.02

TABLE VI
STUCK-AT FAULTS, RELATIVE RUNTIME

Circuit	$T_{drc.1}$	$T_{drc.2}$	$T_{drc.4}$	$T_{drc.8}$	$T_{drc.16}$	$T_{drc.32}$
s208	1.00	2.93	4.32	3.94	8.47	21.38
s298	1.00	1.38	1.43	2.49	3.97	3.71
s344	1.00	1.33	1.79	3.85	4.60	5.73
s382	1.00	1.12	1.62	1.88	2.39	3.49
s386	1.00	1.33	1.71	2.77	4.06	5.66
s420	1.00	2.57	2.36	3.76	5.63	5.76
s526	1.00	2.08	3.04	8.15	11.18	30.02
s641	1.00	1.30	1.39	1.59	2.03	2.38
s820	1.00	0.92	1.08	1.28	1.51	1.85
s1196	1.00	1.70	1.83	2.24	2.97	3.85
s1423	1.00	0.67	0.79	0.68	0.85	1.74
s5378	1.00	2.89	3.59	6.02	7.51	12.02
s35932	1.00	1.34	1.48	1.58	1.32	2.24
b03	1.00	1.55	3.09	2.79	4.80	9.37
b04	1.00	1.41	2.55	2.59	5.34	20.01
b05	1.00	1.11	1.63	1.81	1.79	1.96
b07	1.00	1.16	1.22	1.68	2.71	5.93
b08	1.00	1.22	1.56	2.38	3.88	9.73
b09	1.00	2.19	3.33	2.22	3.18	3.36
b10	1.00	1.78	2.31	2.61	3.75	6.39
b11	1.00	1.36	1.47	2.16	3.20	5.68
b14	1.00	1.66	1.86	1.88	2.32	3.24
des_area	1.00	1.00	1.15	2.36	2.47	4.32
i2c	1.00	1.72	2.08	3.25	4.93	10.56
simple_spi	1.00	1.34	2.50	3.82	5.13	9.91
usb_phy	1.00	1.04	1.67	2.35	2.62	4.45
wb_dma	1.00	2.24	3.47	4.04	6.04	11.13

TABLE VII
TRANSITION FAULT COVERAGE

Circuit	Flts	T_{ex}	T_{rs}	T_{rs+ex}
s208	416	51.68	54.57	54.57
s298	596	69.97	72.65	72.65
s344	688	85.47	90.70	90.70
s382	764	73.43	75.52	76.31
s386	772	67.49	75.91	75.91
s420	840	27.86	28.69	28.69
s526	1052	59.03	62.17	62.26
s641	1280	75.16	80.78	80.78
s820	1640	74.33	45.79	77.13
s1196	2392	81.44	92.10	96.45
s1423	2846	81.66	70.31	83.80
s5378	10590	71.67	65.66	72.74
s35932	71864	86.50	87.21	87.21
b03	768	54.17	56.77	56.77
b04	2284	71.94	79.90	79.95
b05	2976	37.23	38.21	38.21
b07	1936	45.56	47.11	47.11
b08	844	73.70	76.54	76.54
b09	678	65.93	70.35	70.50
b10	870	70.00	78.05	78.51
b11	1830	78.36	73.50	79.67
b14	17180	70.35	65.79	72.81

IV. EXPERIMENTAL RESULTS OF THE DETERMINISTIC PROCEDURE

This section reports the results of the deterministic R&SC-based procedure when applied to single stuck-at faults, and to transition faults, in ISCAS-89 and ITC-99 benchmark circuits. The procedure is also applied to single stuck-at faults in several IWLS-05 benchmark circuits.

A. Sequences

For comparison, this section uses an existing test sequence that was generated for single stuck-at faults by the test generation procedure from [11], [26], or [31], and compacted using a restoration-based procedure without set covering [17]. The sequence is denoted by T_{ex} (ex stands for existing). It achieves the highest single stuck-at fault coverage that is known to be possible for the benchmark circuits considered.

The main part of the test sequence T, to which the R&SC-based procedure is applied, is generated by a low-complexity simulation-based procedure that attempts to avoid what is called repeated synchronization [32]. This part of T is denoted by T_{rs} (rs stands for repeated synchronization). It is constructed from subsequences of length 1024 that avoid repeated synchronization to different extents. The only test compaction method incorporated into the generation of T_{rs} is to remove test vectors that do not detect any faults from the end of the sequence after every subsequence of length 1024 is concatenated. The length of T_{rs} is limited to 8192.

TABLE VIII
TRANSITION FAULTS, TEST SEQUENCE LENGTH

Circuit	T_{ex}	T_{rs}	T_{rs+ex}	$T_{drc.1}$	$T_{drc.2}$	$T_{drc.4}$	$T_{drc.8}$	$T_{drc.16}$	$T_{drc.32}$
s208	105	6856	6856	131	132	124	107	116	129
s298	117	1913	1913	141	130	136	132	134	126
s344	57	3753	3753	137	113	109	103	103	84
s382	516	4364	4633	903	831	820	800	812	833
s386	121	6447	6447	211	197	190	202	185	206
s420	108	3435	3435	123	131	124	123	121	118
s526	1006	8201	8836	1382	1370	1251	1182	1147	1279
s641	101	8333	8333	216	178	178	173	178	182
s820	491	8291	8774	421	424	413	410	405	393
s1196	238	9016	9254	443	397	373	367	352	365
s1423	1024	8969	9993	1049	967	947	845	829	939
s5378	646	8396	9042	1008	856	1278	677	929	786
s35932	150	4982	4982	442	413	465	410	381	415
b03	130	2655	2655	106	130	118	121	130	106
b04	168	8838	8889	319	267	267	272	262	238
b05	321	3419	3419	307	292	291	355	360	360
b07	380	4336	4336	291	293	344	304	304	304
b08	415	8289	8289	517	423	406	426	426	442
b09	265	8809	9020	419	413	395	310	330	312
b10	190	8231	8402	215	207	194	193	182	196
b11	554	8400	8814	473	428	426	398	388	350
b14	5611	8603	14214	6174	5705	5441	5491	4545	4345

TABLE IX
TRANSITION FAULTS, RELATIVE LENGTH

Circuit	$T_{drc.1}$	$T_{drc.2}$	$T_{drc.4}$	$T_{drc.8}$	$T_{drc.16}$	$T_{drc.32}$
s208	1.00	1.01	0.95	0.82	0.89	0.98
s298	1.00	0.92	0.96	0.94	0.95	0.89
s344	1.00	0.82	0.80	0.75	0.75	0.61
s382	1.00	0.92	0.91	0.89	0.90	0.92
s386	1.00	0.93	0.90	0.96	0.88	0.98
s420	1.00	1.07	1.01	1.00	0.98	0.96
s526	1.00	0.99	0.91	0.86	0.83	0.93
s641	1.00	0.82	0.82	0.80	0.82	0.84
s820	1.00	1.01	0.98	0.97	0.96	0.93
s1196	1.00	0.90	0.84	0.83	0.79	0.82
s1423	1.00	0.92	0.90	0.81	0.79	0.90
s5378	1.00	0.85	1.27	0.67	0.92	0.78
s35932	1.00	0.93	1.05	0.93	0.86	0.94
b03	1.00	1.23	1.11	1.14	1.23	1.00
b04	1.00	0.84	0.84	0.85	0.82	0.75
b05	1.00	0.95	0.95	1.16	1.17	1.17
b07	1.00	1.01	1.18	1.04	1.04	1.04
b08	1.00	0.82	0.79	0.82	0.82	0.85
b09	1.00	0.99	0.94	0.74	0.79	0.74
b10	1.00	0.96	0.90	0.90	0.85	0.91
b11	1.00	0.90	0.90	0.84	0.82	0.74
b14	1.00	0.92	0.88	0.89	0.74	0.70

TABLE X
TRANSITION FAULTS, RELATIVE RUNTIME

Circuit	$T_{drc.1}$	$T_{drc.2}$	$T_{drc.4}$	$T_{drc.8}$	$T_{drc.16}$	$T_{drc.32}$
s208	1.00	1.29	0.99	1.35	1.75	3.30
s298	1.00	1.23	1.68	2.18	3.33	4.39
s344	1.00	1.52	1.87	2.97	3.32	5.34
s382	1.00	0.94	1.74	1.61	2.64	5.62
s386	1.00	1.01	1.46	2.21	3.69	5.32
s420	1.00	1.36	1.45	1.67	2.28	3.14
s526	1.00	2.91	4.51	6.11	9.89	22.09
s641	1.00	1.21	1.34	1.88	2.62	2.86
s820	1.00	1.41	1.85	1.81	2.24	3.66
s1196	1.00	1.56	1.77	2.46	2.59	3.08
s1423	1.00	1.78	1.53	1.58	1.69	2.65
s5378	1.00	2.11	2.09	1.55	2.91	3.72
b03	1.00	2.01	2.06	2.60	3.28	5.11
b04	1.00	1.12	1.28	1.43	2.42	4.02
b05	1.00	1.39	1.71	3.61	3.09	3.59
b07	1.00	1.45	1.58	1.27	1.70	2.97
b08	1.00	1.45	2.05	3.41	3.93	7.44
b09	1.00	2.64	2.35	2.59	3.95	6.78
b10	1.00	1.28	1.24	1.74	2.23	4.20
b11	1.00	1.16	1.23	1.91	2.55	2.31
b14	1.00	1.17	2.51	8.04	1.44	10.69

T_{rs} represents the results of a low-complexity simulation-based test generation procedure, which is likely to be used for the generation of functional test sequences, and which needs to be followed by static test compaction.

T_{rs} is generated for single stuck-at faults when these faults are targeted by the static test compaction procedure, and for transition faults when these faults are targeted. T_{ex} is always the same sequence generated for single stuck-at faults.

TABLE XI
STUCK-AT FAULTS, RANDOM INITIAL OMISSION

Circuit	T_{ex}	T_{rs}	T_{rs+ex}	$T_{rrc.1}$	$T_{rrc.2}$	$T_{rrc.4}$	$T_{rrc.8}$	$T_{rrc.16}$	$T_{rrc.32}$	$T_{drc.32}$
s208	105	4095	4095	2065	556	129	106	103	103	129
s298	117	2099	2099	1051	296	113	95	93	93	80
s344	57	1873	1873	943	251	73	50	46	45	56
s382	516	3269	3457	1877	886	719	656	648	648	640
s386	121	3940	3940	1971	520	148	116	113	104	108
s420	108	2813	2813	1411	381	143	107	107	107	96
s526	1006	7179	8163	4380	1539	1116	969	961	957	1060
s641	101	2790	2790	1416	403	129	99	90	89	85
s820	491	8263	8749	4552	1375	465	354	345	343	380
s1196	238	8323	8561	4311	1181	332	246	239	239	213
s1423	1024	8511	9535	5133	1858	785	667	640	640	585
s5378	646	9127	9773	5064	1694	831	710	623	617	836
s35 932	150	3123	3123	1580	467	170	158	150	140	116
b03	130	2458	2458	1230	327	114	90	88	88	91
b04	168	4843	4843	2429	649	184	147	142	142	122
b05	321	2034	2034	1028	371	229	222	221	221	213
b07	380	2734	2734	1429	481	208	180	179	179	187
b08	415	5475	5475	2768	812	320	282	277	277	298
b09	265	5214	5214	2694	888	362	298	288	287	315
b10	190	4825	4996	2517	662	155	120	115	115	118
b11	554	7309	7677	3901	1116	397	350	311	302	342
b14	5611	8727	14338	9001	4931	3612	3342	3313	3144	5165
aes_core	-	2775	2775	1425	881	796	782	778	778	-
des_area	-	920	920	469	318	275	268	257	253	236
i2c	-	8705	8705	4393	1184	282	181	171	168	172
simple_spi	-	8635	8635	4642	1667	859	743	717	717	778
tv80	-	8740	8740	4600	1643	848	731	705	699	-
usb_phy	-	8170	8170	4261	1491	796	681	657	641	875
wb_dma	-	8371	8371	4297	1527	660	585	554	527	582

In general, it is expected that only T_{rs} (or a test sequence that was generated by a low-complexity test generation procedure) will be available, and will need to be compacted. However, considering single stuck-at faults, T_{rs} does not always achieve the same fault coverage as T_{ex}. To allow a comparison for the same fault coverage, T_{ex} is concatenated to T_{rs}, and the last test vectors are removed if they do not detect any faults. T_{ex} is removed in its entirety if T_{rs} already detects all the detectable faults. The resulting test sequence is denoted by T_{rs+ex}. T_{rs+ex} is also computed for transition faults in case T_{ex} detects faults that are not detected by T_{rs}.

For IWLS-05 benchmark circuits, the sequence T_{ex} is not available, and $T = T_{rs+ex} = T_{rs}$.

The deterministic R&SC-based procedure is applied to T_{rs+ex} using $n = 1$, 2, 4, 8, 16, and 32. The compacted test sequence obtained for a given value of n is denoted by $T_{drc.n}$ (drc stands for deterministic restoration covering).

B. Single Stuck-At Faults

Test sequences for single stuck-at faults are simulated starting from the all-unspecified initial state. The results of static test compaction are reported in Tables III–VI as follows.

Table III shows the number of gates and the number of single stuck-at faults. It then shows the single stuck-at fault coverage of T_{ex}, T_{rs}, and T_{rs+ex}. For T_{ex} and T_{rs+ex}, this is the highest fault coverage that is known to be achievable. In some cases, T_{rs} achieves a lower fault coverage since it does not target faults directly. All the compacted test sequences produced by the R&SC-based procedure achieve the same fault coverage as T_{rs+ex}.

Table IV shows the lengths of T_{ex}, T_{rs}, T_{rs+ex}, and $T_{drc.n}$ for $n = 1, 2, 4, \ldots, 32$.

Table V shows the lengths of $T_{drc.n}$ for $n = 1, 2, 4, \ldots, 32$ as a fraction of the length of $T_{drc.1}$. This shows the reduction in test length as n is increased compared to $n = 1$.

Table VI shows the runtime for the R&SC-based procedure to produce $T_{drc.n}$, for $n = 1, 2, 4, \ldots, 32$, as a fraction of the runtime for producing $T_{drc.1}$. This shows the increase in runtime due to the increase in n. With $n = 1$, the procedure is similar to the restoration-based procedure from [17], and has similar runtimes.

The following points can be seen from Tables III–VI.

Although T_{ex} is compacted by a restoration-based procedure, Table IV shows that the R&SC-based procedure

TABLE XII
TRANSITION FAULTS, RANDOM INITIAL OMISSION

Circuit	T_{ex}	T_{rs}	T_{rs+ex}	$T_{rrc.1}$	$T_{rrc.2}$	$T_{rrc.4}$	$T_{rrc.8}$	$T_{rrc.16}$	$T_{rrc.32}$	$T_{drc.32}$
s208	105	6856	6856	3437	894	204	155	151	151	129
s298	117	1913	1913	976	314	130	116	116	116	126
s344	57	3753	3753	1881	495	117	89	89	89	84
s382	516	4364	4633	2614	1427	898	787	776	768	833
s386	121	6447	6447	3230	840	244	173	163	160	206
s420	108	3435	3435	1738	492	151	127	127	127	118
s526	1006	8201	8836	4646	2047	1509	1257	1211	1073	1279
s641	101	8333	8333	4193	1163	314	199	169	150	182
s820	491	8291	8774	4542	1388	498	399	385	375	393
s1196	238	9016	9254	4710	1395	513	413	395	383	365
s1423	1024	8969	9993	5487	2218	1302	1154	1139	1092	939
s5378	646	8396	9042	4711	1743	1399	1191	1110	803	786
s35932	150	4982	4982	2536	857	461	409	390	385	415
b03	130	2655	2655	1333	361	140	107	105	105	106
b04	168	8838	8889	4475	1200	333	250	235	230	238
b05	321	3419	3419	1765	682	416	357	354	354	360
b07	380	4336	4336	2235	706	378	330	328	328	304
b08	415	8289	8289	4187	1276	621	470	464	388	442
b09	265	8809	9020	4637	1512	588	431	404	392	312
b10	190	8231	8402	4231	1135	313	196	191	186	196
b11	554	8400	8814	4501	1356	514	400	388	369	350
b14	5611	8603	14214	9749	6870	5690	5443	5294	4475	4345

sometimes produces test sequences that are shorter than T_{ex}. This happens for $n > 1$ even if the test sequence produced with $n = 1$ is longer than T_{ex}. For example, for $s386$, the length of T_{ex} is 121, the length of $T_{drc.1}$ is higher at 156, and the length of $T_{drc.32}$ is lower at 108. These results point to the effectiveness of the set covering heuristics as part of the restoration-based procedure.

A more direct comparison is between the test sequences produced by the R&SC-based procedure for different values of n. Tables IV and V show that, as n is increased, the procedure is able to produce shorter test sequences. The reduction is not monotonic because of the heuristic nature of the procedure. The reduction is significant given that the comparison is with a test sequence that is compacted using the same procedure with different values of n. In particular, it is interesting to compare the case where $n = 1$ with the cases where $n > 1$. With $n = 1$, the procedure does not benefit from the set covering heuristics. In most of the cases, $n > 1$ produces a shorter test sequence than $n = 1$.

As expected, the normalized runtime increases with n. However, Table VI shows that the increase is slower than linear with n, and allows $n > 1$ to be considered for improved reductions in the length of the compacted test sequence.

C. Transition Faults

This subsection reports on the application of the R&SC-based procedure to transition faults.

For the simulation of transition faults, a transition fault is associated with an extra delay of a single clock cycle [33]. It is possible to consider transition faults with higher extra delays. However, such faults tend to be easier to detect. Test sequences for transition faults are simulated starting from the all-zero initial state.

The test sequence T_{rs} is generated targeting transition faults. T_{ex} is the sequence generated for single stuck-at faults. The results for transition faults considering several of the benchmark circuits are shown in Tables VII–X in the same format as Tables III–VI.

For most of the circuits in Tables VII–X, the test sequence T_{rs} detects more transition faults than T_{ex}. In addition, for several of the circuits, when T_{ex} is concatenated to T_{rs}, it does not detect any additional transition faults.

As for single stuck-at faults, the procedure with $n > 1$ reduces the test sequence length significantly in several cases compared with the case where $n = 1$. As with single stuck-at faults, the reduction is not monotonic with n because of the heuristic nature of the procedure. It is significant since the comparison is among different compacted sequences that are produced by the same procedure with different values of n.

V. R&SC-BASED PROCEDURE WITH RANDOM INITIAL OMISSION

The deterministic R&SC-based procedure yields a non-monotonic reduction in test sequence length as n is increased. This section addresses this issue. It also addresses the increase in computational effort as n is increased. It uses for this purpose the restoration-based procedure described in [25].

The restoration-based procedure from [25] improves the ability to compact a sequence by making random decisions

TABLE XIII
RUNTIME COMPARISON

	Stuck-at Faults		Transition Faults	
Circuit	$T_{drc.32}$	$T_{rrc.32}$	$T_{drc.32}$	$T_{rrc.32}$
s208	23.02	3.91	3.53	2.38
s298	8.67	9.95	8.13	7.24
s344	12.96	15.16	10.67	8.12
s382	39.25	64.12	59.22	35.14
s386	11.92	7.25	8.89	5.80
s420	4.02	1.84	1.09	1.37
s526	507.30	257.45	221.93	161.90
s641	4.55	8.72	4.04	5.92
s820	7.74	16.98	6.65	10.11
s1196	14.67	24.72	11.44	21.96
s1423	26.03	20.23	17.20	26.75
s5378	29.04	9.04	18.70	27.71
s35932	4.55	10.94	-	-
b03	11.87	5.49	5.42	3.73
b04	29.66	7.72	8.26	6.65
b05	2.23	5.32	2.13	3.09
b07	9.31	6.91	3.20	3.65
b08	26.15	23.87	34.03	29.74
b09	9.92	15.04	15.12	13.15
b10	16.68	12.33	9.60	8.25
b11	28.91	17.19	9.61	15.04
b14	31.40	33.70	465.64	264.89
des_area	-	113.21	-	-
i2c	9.24	4.77	-	-
simple_spi	20.26	8.34	-	-
usb_phi	14.06	7.09	-	-
wb_dma	-	15.28	-	-

regarding the test vectors that are omitted from the test sequence initially. Randomly keeping certain vectors in the sequence before starting the restoration process slows down the convergence of the procedure and causes it to make more iterations. Each iteration has a substantially different starting point from other iterations because of the random initial omission. As a result, the procedure yields shorter test sequences.

The R&SC-based procedure with random initial omission can take advantage of the increased number of iterations to increase n gradually. The procedure described in this section starts with $n = 1$ for the first iteration, and doubles the value of n with every additional iteration, until n reaches its final value, which is denoted by n_{max}. In this way, the increased computational effort of n-detection fault simulation with increasing values of n is offset by the reduction in test sequence length. Moreover, considering $n > 1$, the test sequence for n is obtained from the test sequence for $n/2$. This guarantees a monotonic reduction in test sequence length as n is increased.

Random initial omission is performed in this procedure using a probability denoted by p. At the beginning of an iteration, p of the test vectors are selected randomly, and the remaining vectors are omitted from the sequence. Restoration is then performed starting from this initial set of omitted vectors.

The probability p is also changed from one iteration to the next to match the fact that n has a lower value in earlier iterations. With lower values of n, the procedure makes less accurate compaction decisions. Higher values of p allow it to make these decisions with respect to fewer omitted vectors. In later iterations, p is decreased, and the procedure is able to make decisions on more omitted vectors based on more accurate information.

Specifically, let T be a test sequence of length L to which the procedure is applied. Iteration 0 is applied with $n = 1$ and $p = 1/2$. The initial omission reduces the length of T to (approximately) $1/2L$. Restoration produces a test sequence $T_{rrc.1}$ of length $L_{rrc.1} \geq 1/2L$ ($L_{rrc.1} \approx 1/2L$ if all the faults are detected by the initial sequence and restoration of test vectors is not needed for any fault).

Iteration 1 is applied with $n = 2$ and $p = 1/4$. The initial omission reduces the length of T to (approximately) $1/4L_{rrc.1}$. Restoration produces a test sequence $T_{rrc.2}$ of length $L_{rrc.2} \geq 1/4L_{rrc.1} \geq 1/(2 \cdot 4)L$.

As p is decreased, the lower bound on the test sequence length is decreased. Eventually, the vectors that are initially retained are not sufficient for detecting most of the faults, and the sequence length after restoration is determined by the need to detect target faults. Experimental results (not presented here) indicate that it is important to reach such low values of p in order to allow the sequence to be compacted.

When p reaches its final value, the procedure keeps it at this value. When n reaches its final value of n_{max}, the procedure continues to iterate with the same value of n until no further reductions in test sequence length are obtained.

For the same number of iterations, the same final value n_{max} of n, and the same sequence lengths, the deterministic R&SC-based procedure has a higher computational cost since it performs all the iterations using $n = n_{max}$, while the procedure with random initial omission increases n gradually. However, in effect, the numbers of iterations are not the same, and the compacted sequence lengths are not the same. The procedure with random initial omission obtains a more gradual reduction in test sequence length, and performs more iterations. As a result, it may have a higher runtime, but also produce shorter compacted test sequences.

VI. EXPERIMENTAL RESULTS OF THE PROCEDURE WITH RANDOM INITIAL OMISSION

The R&SC-based procedure with random initial omission was applied to single stuck-at faults and to transition faults in ISCAS-89, ITC-99, and IWLS-05 benchmark circuits using the sequences $T = T_{rs+ex}$ as in Section IV. The procedure was applied using the following parameter values.

The values considered for n are $n = 1, 2, 4, \ldots, 32$. The corresponding values for p are $p = 1/2, 1/4, 1/8, \ldots, 1/64$.

The test lengths produced by the procedure are shown in Tables XI and XII. The tables show the lengths of T_{ex}, T_{rs}, T_{rs+ex}, and $T_{rrc.n}$ for $n = 1, 2, 4, \ldots, 32$. In this case, $T_{rrc.n}$, for $1 \leq n < 32$, is an intermediate compacted sequence that is obtained during a process that increases n gradually. The final compacted sequence is $T_{rrc.32}$. For comparison, the last

TABLE XIV
STUCK-AT AND TRANSITION FAULTS, RANDOM INITIAL OMISSION

Circuit	Flts	T_{ex}	T_{rs}	T_{rs+ex}	$T_{rrc.1}$	$T_{rrc.2}$	$T_{rrc.4}$	$T_{rrc.8}$	$T_{rrc.16}$	$T_{rrc.32}$	$T_{rrc.64}$	$T_{rrc.128}$	n.time
s208	631	105	6967	6967	3485	910	254	195	194	178	=	=	2.53
s298	904	117	2102	2102	1070	329	141	126	121	=	=	=	7.75
s344	1030	57	3753	3753	1881	495	117	89	=	=	=	=	7.97
s382	1163	516	4538	4807	2720	1420	978	938	783	782	=	=	25.80
s420	1270	108	3620	3620	1832	518	169	145	137	=	=	=	1.31
s526	1607	1006	8449	9433	5019	2257	1527	1358	1340	1263	1260	=	79.75
s641	1747	101	8333	8333	4196	1157	310	219	187	181	179	=	4.34
s820	2490	491	8312	8803	4569	1425	530	447	420	416	=	=	10.14
s1196	3634	238	9016	9254	4713	1385	504	419	405	402	=	=	17.75
s1423	4361	1024	8969	9993	5621	2258	1216	1103	1077	1027	1004	923	25.88
s5378	15193	646	8725	9371	4907	2657	1636	1074	991	973	902	727	19.85
s35932	110958	150	4958	4958	2494	926	541	479	440	440	440	440	-
b03	1220	130	2655	2655	1333	375	145	116	108	=	=	=	3.64
b04	3630	168	8486	8537	4295	1154	323	237	229	227	226	219	5.27
b05	4792	321	3419	3419	1754	642	383	365	361	356	=	=	4.70
b07	3119	380	5352	5352	2708	903	403	348	347	=	=	=	6.11
b08	1333	415	8394	8394	4244	1249	590	471	467	446	445	429	29.02
b09	1098	265	8717	8928	4588	1479	613	398	369	=	=	=	10.86
b10	1382	190	8355	8526	4294	1150	280	205	189	185	184	=	9.06
b11	2919	554	8728	9142	4666	1384	540	435	421	410	409	=	13.76
b14	27161	5611	8650	14261	10342	7604	6589	6225	6100	6054	5916	5522	-

column shows the length of $T_{drc.32}$ that was obtained by the deterministic procedure.

It is interesting to note that the final test sequence length is sometimes obtained in Tables XI and XII before the final value of n, $n_{max} = 32$, is considered.

Comparing the final test sequence length in Tables XI and XII to the corresponding test sequence lengths of the deterministic procedure, it can be seen that, in most cases, random initial omission allows shorter test sequences to be obtained. Overall, the compacted sequences are shorter with random initial omission.

Tables XI and XII also show that the procedure with random initial omission produces monotonic reductions in the test sequence length as n is increased.

The effect of using random initial omission on the runtime of the R&SC-based procedure is captured as follows: The runtime for producing $T_{rrc.32}$ by the procedure with random initial omission is normalized to the runtime for simulating T_{rs+ex}. For comparison, the runtime for producing $T_{drc.32}$ from T_{rs+ex} by the deterministic procedure is also normalized to the runtime for simulating T_{rs+ex}. The two normalized runtimes for single stuck-at faults and for transition faults are shown in Table XIII.

Table XIII shows that there are circuits where the deterministic procedure is faster than the procedure with random initial omission. However, overall, the procedure with random initial omission is faster than the deterministic procedure.

Finally, the procedure with random initial omission was run on a set of target faults that consists of both single stuck-at faults and transition faults. This case is interesting since, with functional test sequences, the detection of transition faults does not guarantee the detection of single stuck-at faults and vice versa. Therefore, test sequences that detect both types of faults are, in general, longer. This case is also used for investigating the need to use higher values of n_{max}.

TABLE XV
RETAINING A SYNCHRONIZING PREFIX

Circuit	T_{ex}	T_{rs}	T_{rs+ex}	$T_{drc.1}$	$T_{drc.32}$	ntime
s208	105	4095	4095	132	127	22.71
s298	117	2099	2099	98	80	11.69
s344	57	1873	1873	59	58	14.17
s382	516	3269	3457	692	655	74.51
s386	121	3940	3940	137	110	14.62
s420	108	2813	2813	131	93	3.18
s526	1006	7179	8163	1141	967	426.20
s641	101	2790	2790	105	92	8.31
s820	491	8263	8749	390	361	7.67
s1196	238	8323	8561	258	215	15.29
s1423	1024	8511	9535	841	642	15.41
b03	130	2458	2458	88	75	9.45
b04	168	4843	4843	162	121	18.09
b05	321	2034	2034	215	213	4.74
b07	380	2734	2734	188	147	8.25
b08	415	5475	5475	374	333	77.50
b09	265	5214	5214	301	284	9.69
b10	190	4825	4996	130	118	19.69
b11	554	7309	7677	304	272	24.93

For this experiment, the sequence T_{rs} is generated targeting both fault models.

The test lengths and the normalized run time for several circuits are shown in Table XIV. For Table XIV, the procedure is run using $n_{max} = 128$. If the test length reaches its final

TABLE XVI
EFFECT OF p

Circuit	T_{ex}	T_{rs}	T_{rs+ex}	$T_{rrc.1}$	$T_{rrc.2}$	$T_{rrc.4}$	$T_{rrc.8}$	$T_{rrc.16}$	$T_{rrc.32}$	ntime
s208	105	4095	4095	1050	230	109	108	108	108	3.40
s298	117	2099	2099	545	147	99	98	98	97	3.61
s344	57	1873	1873	479	82	45	45	45	45	11.94
s382	516	3269	3457	1240	710	666	664	662	660	96.26
s386	121	3940	3940	996	194	114	111	111	111	5.41
s420	108	2813	2813	717	204	137	99	99	99	1.80
s526	1006	7179	8163	2556	1244	1056	1030	999	999	132.06
s641	101	2790	2790	733	185	120	105	99	99	6.85
s820	491	8263	8749	2455	603	361	336	324	322	10.52
s1196	238	8323	8561	2204	443	246	232	232	232	16.65
s1423	1024	8511	9535	2986	1108	833	669	662	662	14.55
b03	130	2458	2458	634	140	99	95	89	84	4.66
b04	168	4843	4843	1228	227	136	115	113	113	5.35
b05	321	2034	2034	622	241	222	219	219	219	4.40
b07	380	2734	2734	809	220	183	182	182	182	4.31
b08	415	5475	5475	1492	495	332	283	279	278	28.68
b09	265	5214	5214	1516	455	326	307	293	291	14.12
b10	190	4825	4996	1277	214	126	111	108	103	8.72
b11	554	7309	7677	2054	463	303	279	278	278	10.93

value for $n_{fin} < n_{max}$, the entries for $n_{fin} < n \leq n_{max}$ are an equal sign (=).

From Table XIV, it can be seen that T_{rs} is longer when both single stuck-at and transition faults are targeted compared with the case where only one of the fault models is considered. The difference is lower for the compacted test sequence.

In addition, values of n that are higher than 32 are sometimes effective in reducing the test sequence length. However, overall, $n_{max} = 32$ is typically sufficient for obtaining most of the reduction in test sequence length.

VII. OTHER OPTIONS FOR R&SC-BASED PROCEDURES

As mentioned in Section III, there are various options related to the implementation of a restoration-based procedure. These options affect the sequence lengths it produces and its runtime. In every case, the restoration-based procedure can be enhanced using set covering heuristics as described in Sections III and V. This section discusses additional options and presents experimental results for single stuck-at faults in several circuits to demonstrate their effects.

An option that is relevant to the deterministic procedure is related to the vectors that are initially omitted from the sequence. In Section III, all the vectors were initially omitted. A different option, which was considered in [17], is to retain a synchronizing prefix of the sequence. The synchronizing prefix brings the fault-free circuit from the all-unspecified state to a fully specified state, and it is important for the detection of all the faults. Retaining it allows all the faults to use the same vectors for synchronization, thus producing shorter test sequences. However, this also increases the possibility that restoring vectors for one fault will cause another fault to lose its detection, as discussed in Section III. Consequently, the procedure may need to perform more iterations, and its runtime may increase.

The results of the deterministic procedure with the option of retaining a synchronizing prefix are shown in Table XV, which shows the lengths of the sequences obtained with $n = 1$ and with $n = 32$. These lengths can be compared with the ones in Table IV. The normalized runtime is given in the last column of Table XV. It can be compared with the normalized run time given in the second column of Table XIII.

For the procedure with random initial omission, there is an option of using different values for p. As discussed earlier, the final values of p must be sufficiently low to allow the procedure to reduce the length of the sequence. Table XVI shows the results obtained when the values for p are $p = 1/4$, $1/8, \ldots, 1/128$. The test lengths in Table XVI can be compared with those in Table XI. The normalized run time can be compared with the one given in the third column of Table XIII.

From Tables XV and XVI, it can be seen that, with these options also, the addition of set covering heuristics reduces the lengths of the compacted sequences that are produced by the restoration-based procedures. In general, one may expect that, regardless of the options selected, the use of set covering heuristics will reduce the lengths of the compacted test sequences.

VIII. CONCLUSION

This paper described the enhancement of a restoration-based static test compaction procedure by set covering heuristics. Restoration-based static test compaction is applied to a single functional test sequence. It reduces its length by omitting vectors that are not necessary for detecting target faults. Sets for set covering were computed using n-detection fault simulation. This process associated a set of up to n detection time units with every fault. The set covering heuristics affected the order by which faults were considered during the restoration process. They also guided the computation of several alternatives

for every fault, from which the best one was selected. The paper described two variations of the procedure. The first was deterministic, and used a constant value of n. Experimental results using this procedure showed that, overall, the R&SC-based procedure produces shorter test sequences for $n > 1$ than for $n = 1$. This paper also described a variation of the R&SC-based procedure that uses random initial omission to ensure that test length reduction occurs over a larger number of iterations. During these iterations, the procedure increased the value of n gradually in order to benefit from increasingly accurate compaction decisions as the test sequence length was reduced. The result was a monotonic reduction in test sequence length with n and, in some cases, shorter test sequence lengths. The runtime was reduced in many cases.

REFERENCES

[1] J. Rearick, "Too much delay fault coverage is a bad thing," in *Proc. Int. Test Conf.*, Nov. 2001, pp. 624–633.

[2] J. Saxena, K. M. Butler, V. B. Jayaram, S. Kundu, N. V. Arvind, P. Sreeprakash, and M. Hachinger, "A case study of IR-drop in structured at-speed testing," in *Proc. Int. Test Conf.*, Oct. 2003, pp. 1098–1104.

[3] S. Sde-Paz and E. Salomon, "Frequency and power correlation between at-speed scan and functional tests," in *Proc. IEEE Int. Test Conf.*, Oct. 2008, pp. 1–9.

[4] P. C. Maxwell, R. C. Aitken, K. R. Kollitz, and A. C. Brown, "IDDQ and AC scan: The war against unmodelled defects," in *Proc. Int. Test Conf.*, Oct. 1996, pp. 250–258.

[5] L. Chen and S. Dey, "Software-based self-testing methodology for processor cores," *IEEE Trans. Comput. Aided Design Integr. Circuits Syst.*, vol. 20, no. 3, pp. 369–380, Mar. 2001.

[6] P. Parvathala, K. Maneparambil, and W. Lindsay, "FRITS - A microprocessor functional BIST method," in *Proc. Intl. Test Conf.*, 2002, pp. 590–598.

[7] N. Kranitis, A. Paschalis, D. Gizopoulos, and G. Xenoulis, "Software-based self-testing of embedded processors," *IEEE Trans. Comput.*, vol. 54, no. 4, pp. 461–475, Apr. 2005.

[8] M. Nakazato, S. Ohtake, M. Inoue, and H. Fujiwara, "Design for testability of software-based self-test for processors," in *Proc. Asian Test Symp.*, Nov. 2006, pp. 375–380.

[9] I. Koren and C. M. Krishna, *Fault-Tolerant Systems*. San Mateo, CA, USA: Morgan Kaufman Publishers, 2007.

[10] R. K. Roy, T. M. Niermann, J. H. Patel, J. A. Abraham, and R. A. Saleh, "Compaction of ATPG-generated test sequences for sequential circuits," in *Proc. Int. Conf. Comput. Aided Design*, Nov. 1988, pp. 382–385.

[11] I. Pomeranz and S. M. Reddy, "On generating compact test sequences for synchronous sequential circuits," in *Proc. Eur. Design Autom. Conf.*, Sep. 1995, pp. 105–110.

[12] T. J. Lambert and K. K. Saluja, "Methods for dynamic test vector compaction in sequential test generation," in *Proc. 19th Int. Conf. VLSI Design*, Jan. 1996, pp. 166–169.

[13] A. Raghunathan and S. T. Chakradhar, "Dynamic test sequence compaction for sequential circuits," in *Proc. 19th Int. Conf. VLSI Design*, Jan. 1996, pp. 170–173.

[14] I. Pomeranz and S. M. Reddy, "On static compaction of test sequences for synchronous sequential circuits," in *Proc. 33rd Design Autom. Conf.*, Jun. 1996, pp. 215–220.

[15] E. M. Rudnick and J. H. Patel, "Simulation-based techniques for dynamic test sequence compaction," in *Proc. IEEE/ACM Int. Conf. Comput. Aided Design*, Nov. 1996, pp. 67–73.

[16] F. Corno, P. Prinetto, M. Rebaudengo, and M. Sonza Reorda, "New static compaction techniques of test sequences for sequential circuits," in *Proc. Eur. Design Test Conf.*, Mar. 1997, pp. 37–43.

[17] I. Pomeranz and S. M. Reddy, "Vector restoration based static compaction of test sequences for synchronous sequential circuits," in *Proc. IEEE Int. Conf. Comput. Design*, Oct. 1997, pp. 360–365.

[18] E. M. Rudnick and J. H. Patel, "Putting the squeeze on test sequences," in *Proc. Int. Test Conf.*, Nov. 1997, pp. 723–732.

[19] M. S. Hsiao and S. T. Chakradhar, "State relaxation based subsequence removal for fast static compaction in sequential circuits," in *Proc. Design Autom. Test Eur.*, Feb. 1998, pp. 577–582.

[20] S. K. Bommu, S. T. Chakradhar, and K. B. Doreswamy, "Static compaction using overlapped restoration and segment pruning," in *Proc. IEEE/ACM Int. Conf. Comput. Aided Design*, Nov. 1998, pp. 140–146.

[21] X. Lin, W.-T. Cheng, I. Pomeranz, and S. M. Reddy, "SIFAR: Static test compaction for synchronous sequential circuits based on single fault restoration," in *Proc. 18th IEEE VLSI Test Symp.*, May 2000, pp. 205–212.

[22] I. Pomeranz and S. M. Reddy, "Vector replacement to improve static test compaction for synchronous sequential circuits," *IEEE Trans. Comput. Aided Design Integr. Circuits Syst.*, vol. 20, no. 2, pp. 336–342, Feb. 2001.

[23] I. Pomeranz and S. M. Reddy, "Improving the efficiency of static compaction based on chronological order enumeration of test sequences," in *Proc. 11th Asian Test Symp.*, Nov. 2002, pp. 61–66.

[24] M. Dimopoulos and P. Linardis, "Efficient static compaction of test sequence sets through the application of set covering techniques," in *Proc. Design, Autom. Test Eur. Conf. Exhibit.*, Feb. 2004, pp. 194–199.

[25] I. Pomeranz and S. M. Reddy, "Vector restoration based static compaction using random initial omission," *IEEE Trans. Comput. Aided Design Integr. Circuits Syst.*, vol. 23, no. 11, pp. 1587–1592, Nov. 2004.

[26] R. Guo, S. M. Reddy, and I. Pomeranz, "PROPTEST: A property-based test generator for synchronous sequential circuits," *IEEE Trans. Comput. Aided Design Integr. Circuits Syst.*, vol. 22, no. 8, pp. 1080–1091, Aug. 2003.

[27] M. Abramovici, M. A. Breuer, and A. D. Friedman, *Digital Systems Testing and Testable Design*. Piscataway, NJ, USA: IEEE Press, 1995.

[28] D. S. Hochbaum, "An optimal test compression procedure for combinational circuits," *IEEE Trans. Comput. Aided Design Integr. Circuits Syst.*, vol. 15, no. 10, pp. 1294–1299, Oct. 1996.

[29] P. F. Flores, H. C. Neto, and J. P. Marques-Silva, "On applying set covering models to test set compaction," in *Proc. 19th Great Lakes Symp. VLSI*, Mar. 1999, pp. 8–11.

[30] K. O. Boateng, H. Konishi, and T. M. Nakata, "A method of static compaction of test stimuli," in *Proc. 10th Asian Test Symp.*, Nov. 2001, pp. 137–142.

[31] M. S. Hsiao, E. M. Rudnick, and J. H. Patel, "Sequential circuit test generation using dynamic state traversal," in *Proc. Eur. Design and Test Conf.*, Mar. 1997, pp. 22–28.

[32] I. Pomeranz and S. M. Reddy, "Primary input vectors to avoid in random test sequences for synchronous sequential circuits," *IEEE Trans. Comput. Aided Design*, vol. 27, no. 1, pp. 193–197, Jan. 2008.

[33] K.-T. Cheng, "Transition fault testing for sequential circuits," *IEEE Trans. Comput. Aided Design Integr. Circuits Syst.*, vol. 12, no. 12, pp. 1971–1983, Dec. 1993.

Irith Pomeranz (M'89–SM'96–F'99) 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, in 1985 and 1989, respectively.

She was a Lecturer with the Department of Computer Science, Technion, from 1989 to 1990. From 1990 to 2000, she was a Faculty Member with the Department of Electrical and Computer Engineering, University of Iowa, Iowa City, IO, USA. In 2000, she joined the School of Electrical and Computer Engineering, Purdue University, West Lafayette, IN, USA. Her current research interests include testing of VLSI circuits, design for testability, and synthesis and design verification.

Dr. Pomeranz was a recipient of the NSF Young Investigator Award in 1993 and the University of Iowa Faculty Scholar Award in 1997. She served as an Associate Editor of the *ACM Transactions on Design Automation*, the IEEE TRANSACTIONS ON COMPUTERS, and the IEEE TRANSACTIONS ON VERY LARGE SCALE INTEGRATION (VLSI) SYSTEMS. She served as a Guest Editor of the IEEE TRANSACTIONS ON COMPUTERS January 1998 special issue on Dependability of Computing Systems, and as a Program Co-Chair of the 1999 Fault-Tolerant Computing Symposium. She served as a Program Chair of the 2004 and the 2005 VLSI Test Symposium, and as a General Chair of the 2006 VLSI Test Symposium. She is a Golden Core Member of the IEEE Computer Society.