Procedures for Static Compaction of Test Sequences for Synchronous Sequential Circuits

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

Abstract—We propose three static compaction techniques for test sequences of synchronous sequential circuits. We apply the proposed techniques to test sequences generated for benchmark circuits by various test generation procedures. The results show that the test sequences generated by all the test generation procedures considered can be significantly compacted. The compacted sequences thus have shorter test application times and smaller memory requirements. As a by-product, the fault coverage is sometimes increased as well. Additionally, the ability to significantly reduce the length of the test sequences indicates that it may be possible to reduce test generation time if superfluous input vectors are not generated.

Index Terms—Static test compaction, synchronous sequential circuits, test application time.

1 Introduction

Test compaction for synchronous sequential circuits is the process of reducing the length of a test sequence (or the total length of a set of test sequences) for the circuit. Test compaction is important for reducing the test application time and the volume of test data. Test compaction procedures can be classified into two categories. Dynamic compaction procedures incorporate into the test generation procedure heuristics aimed at reducing the test length. Thus, they attempt to avoid the generation of superfluous test vectors. Static compaction procedures perform test compaction as a postprocessing step, independent of the test generation process. Static compaction has two useful features. 1) Unlike dynamic compaction, static compaction does not require any modifications to the test generation procedure. 2) Since dynamic compaction is based on heuristics and is not guaranteed to achieve the minimum test length, static compaction is useful, even after dynamic compaction is applied during test generation, to further reduce the length of the test sequence.

Prior to the work reported here, compaction of test sequences for synchronous sequential circuits was considered in [1], [2], [3]. The static compaction procedures in [1] and [2] start from sets of test sequences produced by generating a separate test sequence for each fault or subset of faults. They use overlapping and reordering of the individual test sequences to produce a single test sequence of minimal length. The procedure of [3] is a dynamic compaction procedure. In this work, we present three static compaction procedures applicable to the case where a single test sequence is given. The test sequence can be generated directly by test generation procedures, such as [3], [4], [5], [6], or by using the procedures of [1] or [2] to combine individual test sequences into a single sequence.

The effectiveness of the proposed static compaction procedures on benchmark circuits is compared to identify the most effective static compaction procedure among the three procedures investigated in this work. The extent of test compaction possible for deterministic test sequences indicates that test pattern generators spend a significant amount of time generating test vectors that are not necessary. The compacted test sequences provide a target for more efficient deterministic test generators.

In trying to compact a given test sequence, we face the following problem that does not exist when performing static test compaction for combinational circuits: Consider a test sequence $T = (t_0 t_1 \dots t_{L-1})$, where t_i is the input vector applied at time unit u_i. If we remove or modify a vector t_i, then every fault detected by T at or after time unit u_i may potentially be left undetected. This is because fault detection requires a sequence of test vectors that may be disturbed when t_i is removed or modified. As a result, after changing the test sequence, we perform fault simulation to ensure that the change has not reduced the fault coverage. It is interesting to note that, by modifying the sequence, additional faults may be detected that were not detected by the original sequence. Thus, modification of a test sequence may serve not only to reduce its length, but also to increase its fault coverage. To capture these effects of reducing/increasing the fault coverage, fault simulation must be carried out. Thus, all three compaction procedures proposed here require large numbers of fault simulations. However, we believe that the gain in test length reduction and the potential increase in fault coverage justify the investment in additional fault simulation time. Furthermore, the compaction achieved in the work presented here motivated the development of more time-efficient static compaction procedures, presented in [7], [8], [9], [10], [11], [12], [13]. It could also lead to methods to develop more efficient test generation procedures for sequential circuits.

The paper is organized as follows: Section 2 contains definitions and notation used throughout this work. In

• *The authors are with the Electrical and Computer Engineering Department, University of Iowa, Iowa City, IA 52242.*
E-mail: {irith, reddy}@eng.uiowa.edu.

Manuscript received 3 Feb. 1997; revised 27 Apr. 1999; accepted 17 Apr. 2000.
For information on obtaining reprints of this article, please send e-mail to: tc@computer.org, and reference IEEECS Log Number 103741.

TABLE 1
Test Sequence 1 of $s27$

i	0	1	2	3	4	5	6	7	8	9
t_i	0110	1101	0111	0110	1001	0100	1001	1011	0010	1100
i	10	11	12	13	14	15	16	17	18	19
t_i	0110	0110	1110	0010	1101	1110	1001	1001	0111	1010

Section 3, we present a compaction procedure based on an insertion operation that duplicates subsequences of the test sequence and inserts them into the test sequence at specific positions. In Section 4, we present a compaction procedure based on omission of vectors. In Section 5, we present a compaction procedure based on selection of a minimal subset of subsequences sufficient to detect all the faults detected by the original sequence. Section 6 includes experimental results and a comparison among the three procedures. Section 7 concludes the paper.

2 Definitions and Notation

To describe the compaction procedures, we use the following definitions and notation.

A test sequence T is represented as $T = (t_0 t_1 \dots t_{L-1})$, where t_i is the input vector applied at time unit u_i.

The subsequence of T between time units u_j and u_k is denoted by $T[u_j, u_k]$. We have $T[u_j, u_k] = (t_j \dots t_k)$.

The state of the fault-free circuit at time unit u_i is denoted by S_i. The initial state S_0 is the all-unspecified (all-x) state in our experiments.

The output vector of the fault-free circuit at time unit u_i is denoted by z_i.

The set of target faults (collapsed single stuck-at faults) is denoted by F. The set of faults detected by a given test sequence T is denoted by F_{det}.

For every fault $f \in F$, we denote by S_i^f and z_i^f the state and output vector of the faulty circuit at time unit u_i, respectively. We also define the *combined fault-free/faulty state* S_i/S_i^f at time unit u_i.

The time unit where a fault $f \in F_{det}$ is detected for the first time is denoted by $u_{det}(f)$.

The *effective test length* L_{eff} of T is the minimum length of a subsequence of T that starts at time unit 0 and includes the detection time of every detected fault, or $L_{eff} = \max\{u_{det}(f) : f \in F_{det}\} + 1$.

3 Compaction Based on an Insertion Operation

In this section, we describe a test compaction method based on the following operation: Consider a fault $f \in F_{det}$ with detection time $u_{det}(f)$ (we describe the selection of the fault f later). Let u_j and u_k be two time units such that $u_j < u_k \leq u_{det}(f)$ and such that $S_j/S_j^f = S_k/S_k^f$ (i.e., $S_j = S_k$ and $S_j^f = S_k^f$). Since $S_j/S_j^f = S_k/S_k^f$, $T[u_j, u_{k-1}]$ only serves to take the fault-free/faulty circuits back to their states at time unit u_j and T detects f even if we omit $T[u_j, u_{k-1}]$ from T. The sequence obtained by omitting $T[u_j, u_k - 1]$ from T is $T[u_0, u_{j-1}] \circ T[u_k, u_{L-1}]$ ($\circ$ stands for concatenation of subsequences). Under the proposed operation, we define a new test sequence, where fault f is detected earlier, as follows: The subsequence $T[u_k, u_{det}(f)]$ is duplicated and inserted at time unit u_j. As a result, the detection time of f is reduced from $u_{det}(f)$ to $u_{det}(f) - (u_k - u_j)$. The remaining part of the sequence, $T[u_j, u_{L-1}]$, is pushed to the right. The new test sequence is

$$T' = T[u_0, u_{j-1}] \circ T[u_k, u_{det}(f)] \circ T[u_j, u_{L-1}].$$

We refer to this operation as the *insertion operation*. The insertion operation increases the total length of the test sequence; however, it allows us to reduce its effective length by reducing the highest detection times. The shorter sequence $T[u_0, u_{L_{eff}-1}]$ is then used instead of T. The following examples demonstrate the insertion operation.

Example. Consider the test sequence for ISCAS-89 benchmark circuit $s27$ shown in Table 1. The detected faults and their detection times are shown in Table 2, as follows: A fault v stuck-at α is given as v/α. In the row where $i = i_0$, we show all the faults for which $u_{det}(f) = u_{i_0}$. The total number of faults detected is 31. Simulating the fault 19/1, we find that the combined fault-free/faulty states are identical at time units 17 and 18. In addition, we know that the fault is detected at time unit 19. The insertion operation proceeds as follows: We extract $T[18, 19] = (0111, 1010)$ and insert it at time unit 17, pushing $T[17, 19]$ by two time units to the right. The resulting sequence is shown in Table 3. The only detection time that changes is that of the fault 19/1 (all other detection times are prior to time unit 17 and are therefore not affected by the change we made). The new detection time of the fault 19/1 is 18 (instead of 19). This shortens the effective sequence length by one time unit, from 20 to 19.

Example. Consider the test sequence of ISCAS-89 benchmark circuit $s27$ shown in Table 4. The detected faults and their detection times are shown in Table 5. The total

TABLE 2
Detection Information for Test Sequence 1 of $s27$

i	$\{f : u_{det}(f) = u_i\}$
1	2/0 9/1 14/1 18/1 20/0 21/1 26/0
3	6/1 24/1
4	3/0 4/0 8/0 9/0 11/0 12/0 15/1 21/0 25/1 26/1
5	14/0 16/0 17/0 22/0 24/0
6	7/0 8/1 13/1 15/0
16	5/0 25/0
19	19/1

TABLE 3
Test Sequence 1 of $s27$ after an Insertion Operation

i	0	1	2	3	4	5	6	7	8	9
t_i	0110	1101	0111	0110	1001	0100	1001	1011	0010	1100

i	10	11	12	13	14	15	16
t_i	0110	0110	1110	0010	1101	1110	1001

i	17	18	19	20	21
t_i	0111	1010	1001	0111	1010

TABLE 4
Test Sequence 2 of $s27$

i	0	1	2	3	4	5	6	7	8	9
t_i	0011	1101	0011	0011	1110	0011	1011	0001	0011	0110

i	10	11	12	13	14	15	16	17	18	19
t_i	0011	1011	0010	0100	0111	1110	0101	1000	0000	0110

number of faults detected by this sequence is 28. Simulating the fault 6/1, we find that the combined fault-free/faulty states are identical at time units 17 and 19. In addition, we know that the fault is detected at time unit 19. The insertion operation proceeds as follows: We extract $T[19] = (0110)$ and insert it at time unit 17, pushing $T[17, 19]$ by one time unit to the right. The resulting sequence is shown in Table 6. The change affects faults 6/1 and 24/1, with detection times 19 (other detection times are prior to the change we have made in the sequence and therefore are not affected by it). The detection times for the modified sequence are shown in Table 7. Both faults 6/1 and 24/1 that previously had detection time 19 are now detected at time unit 17. In addition, fault 19/1 that was not detected previously is detected at time unit 18 after the change. The result of the insertion operation is thus a reduction by one in the effective test length, and an increase by one in the number of detected faults.

After performing an insertion operation, additional insertion operations using the new sequence can further reduce the effective test length and increase the fault coverage. The proposed test compaction procedure applies the insertion operation iteratively until no additional improvements in effective test length and fault coverage can be obtained. A guaranteed reduction in effective test length can only be achieved if the highest detection times

TABLE 5
Detection Information for Test Sequence 2 of $s27$

i	$\{f : u_{det}(f) = u_i\}$
1	2/0 9/1 14/1 18/1 20/0 21/1 26/0
3	3/0 4/0 8/0 9/0 11/0 12/0 15/1 21/0 25/1 26/1
4	8/1 13/1
5	5/0 25/0
7	22/0
9	14/0 16/0 17/0 24/0
19	6/1 24/1

are reduced by performing the insertion operation for faults such that $u_{det}(f) = L_{eff} - 1$. Nevertheless, we consider all the faults since, by moving a lower detection time, it may become possible to reduce the highest detection times further than in the original sequence. In addition, a fault with a high detection time may be detected earlier or other faults, not detected by the test sequence, may be detected by applying the insertion operation to a fault with $u_{det}(f) = L_{eff} - 1$. We use the following considerations in designing the compaction procedure.

The previous examples demonstrate how the insertion operation can reduce the effective test length and increase the fault coverage of a given test sequence. The insertion operation ensures that the fault for which it is performed is still detected after the operation is performed and that its detection time is reduced. However, another fault detected by the sequence may not be detected after insertion is performed. For example, consider a fault f_1 with equal states at time units u_j and u_k and a fault f_2 with detection time $u_{det}(f_2) \geq u_j$. The insertion operation applied to f_1 changes the subsequence $T[u_j, u_{L-1}]$ and the new sequence may not detect f_2. To minimize this effect, we perform the insertion operation starting with faults that have the highest detection time and reduce the detection time considered only if no additional insertion operations are possible for faults with the currently considered detection time. We also select the time units u_j and u_k where the combined fault-free/faulty states are the same such that u_k is as high as possible and, if a choice exists, u_j is also as high as possible. To guarantee that the fault coverage is not reduced, we do not accept an insertion operation that reduces the fault coverage. If fault simulation after insertion reveals that the fault coverage is lower than before, we restore the test sequence before insertion and proceed to consider other faults.

From our experiments, we found that an insertion operation that does not reduce the effective test length or even increases it may be effective in allowing a later change to reduce the effective test length below what is otherwise possible. We thus allow insertion operations

TABLE 6
Test Sequence 2 of $s27$ after an Insertion Operation

i	0	1	2	3	4	5	6	7	8	9
t_i	0011	1101	0011	0011	1110	0011	1011	0001	0011	0110

i	10	11	12	13	14	15	16	17	18	19	20
t_i	0011	1011	0010	0100	0111	1110	0101	0110	1000	0000	0110

that (temporarily) increase the effective test length. We always store the best test sequence obtained so far to ensure that, at the end of the process, we can recover an earlier test sequence if it is shorter.

Several parameters are used to limit the run time of the procedure. An upper bound $L_{\max}$ is imposed on the total test length. Note that each insertion operation increases the length of the sequence, even if it reduces its effective length. For example, in Table 3, the total sequence length is increased from 20 to 22, although the effective length is reduced from 20 to 19. By setting a bound on the total test length, we limit the number of insertion operations that can be performed. Another bound, $N_{no\text{-}improve}$, ensures that at most $N_{no\text{-}improve}$ consecutive insertion operations are done that do not improve the fault coverage and/or reduce the effective test length. After $N_{no\text{-}improve}$ such operations, the procedure terminates.

Every time an insertion operation is accepted (i.e., it does not reduce the fault coverage), fault simulation is performed, detection times for all the faults are determined, and the faults are considered again, starting with the one that has the highest (new) detection time. An insertion operation that is not accepted is canceled and the next fault is considered. If all the faults are considered and no insertion operation is accepted, the procedure is terminated. The procedure is summarized next.

Procedure 1: Static compaction based on insertion operations

1. Simulate the test sequence T, find the set of detected faults F_{det} and find the detection time of every fault in F_{det}. Let $N_{det} = |F_{det}|$.
2. Set $T_{best} = T$. Set $N_{no\text{-}improve} = 0$.
3. Order the faults in F_{det} according to decreasing detection times.
4. If F_{det} is empty, go to Step 6.
5. Let the first fault in F_{det} be f. Perform the following steps for f:
   a. Remove f from F_{det}.
   b. Find time units u_j and u_k such that $u_j < u_k \leq u_{det}(f)$ and $S_j/S_j^f = S_k/S_k^f$.
   c. If u_j and u_k, as in Step b, do not exist, go to Step 4.
   d. Perform insertion according to u_j, u_k, and $u_{det}(f)$. If the test length exceeds $L_{\max}$, go to Step 6.
   e. Fault simulate the new test sequence, find the set of detected faults F_{det} and find the detection time of every fault in F_{det}.
   f. If $|F_{det}| < N_{det}$:
      i. Restore the previous sequence.
      ii. Set $n_{no\text{-}improve} = n_{no\text{-}improve} + 1$.
      iii. If $n_{no\text{-}improve} = N_{no\text{-}improve}$, go to Step 6.
      iv. Go to Step 4.
   g. If the number of faults detected is the same as before, but the effective test length is not reduced, set $n_{no\text{-}improve} = n_{no\text{-}improve} + 1$. If $n_{no\text{-}improve} = N_{no\text{-}improve}$, go to Step 6, otherwise go to Step 3.
   h. (This step is reached if either the number of faults detected is increased or the effective test length is reduced.) Set $N_{det} = |F_{det}|$ and go to Step 2.
6. Stop: T_{best} is the compacted sequence.

TABLE 7
Detection Information for the Modified Test Sequence 2 of $s27$

i	$\{f: u_{det}(f) = u_i\}$
1	2/0 9/1 14/1 18/1 20/0 21/1 26/0
3	3/0 4/0 8/0 9/0 11/0 12/0 15/1 21/0 25/1 26/1
4	8/1 13/1
5	5/0 25/0
7	22/0
9	14/0 16/0 17/0 24/0
17	6/1 24/1
18	19/1

It is important to note that the simulation procedure in Step 5.e of Procedure 1 does not have to consider faults that were not affected by the insertion operation. If insertion is performed based on u_j, u_k, and $u_{det}(f)$, then a fault g with $u_{det}(g) < u_j$ is not affected by the insertion operation and does not have to be simulated.

The worst-case complexity of Procedure 1 is as follows: Let us denote by N_{iter} the number of times Step 5 is performed by Procedure 1. Note that Procedure 1 terminates after Step 4 if F_{det} is empty. However, it may perform more than $|F_{det}|$ iterations, since the set F_{det} may be updated in Step 5.e. In Step 5.b, the procedure considers at most $O(L^2)$ pairs of time units, and compares values of N_{sv} lines, where N_{sv} is the number of state variables. In Step 5.e, fault simulation has complexity $O(|F_{det}|LN_{lines})$, where N_{lines} is the number of lines in the circuit. We obtain a total complexity of $O(N_{iter}[L^2N_{sv} + |F_{det}|LN_{lines}])$.

We show in Section 6 that compaction based on the insertion operation is effective in reducing the effective test length and increasing the fault coverage of test sequences generated by various test generation procedures. The main disadvantage of compaction based on the insertion operation is that the number of fault simulations it requires to achieve the minimum test length cannot be bounded. We

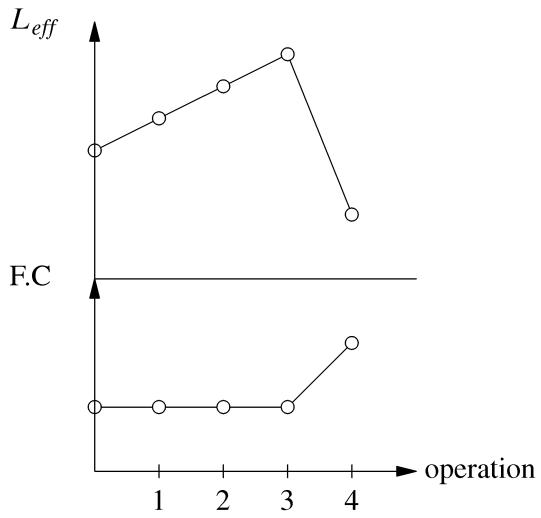


Fig. 1. A sequence of insertion operations.

found that, in many cases, a sequence of insertion operations is required that do not improve the fault coverage, and possibly increase the effective test length, before an additional insertion operation can reduce the effective test length below its original level and/or increase the fault coverage. Such a sequence of operations is shown graphically in Fig. 1, where Operations 1, 2, and 3 increase the effective test length, leaving the fault coverage unchanged, and Operation 4 reduces the effective test length below its original level (before Operation 1 is applied) and increases the fault coverage above its original level. In such cases, Operation 4 alone (not preceded by Operations 1-3) cannot achieve the same improvement. Due to the intricate relationships between consecutive insertion operations, heuristics for reducing the number of insertion operations and, hence, the number of fault simulations are difficult to derive. We therefore prefer the structure of Procedure 1, where the number of fault simulations is arbitrarily limited by setting a limit on the number of insertion operations that do not yield an improvement.

4 Compaction Based on Vector Omission

The compaction method described in this section is based on omission of test vectors from the given sequence. Omission of redundant vectors was considered before for combinational circuits under stuck-at faults and under path delay faults. Here, it is considered in the context of synchronous sequential circuits.

The omission of a vector t_i affects the detection of the faults $\{f\}$ for which $u_{det}(f) \geq u_i$. In addition, it may cause a fault which is undetected when t_i is included in the test sequence to be detected after t_i is omitted. These effects are taken into account by fault simulating the sequence after t_i is omitted.

We consider the test vectors for omission in the order in which they appear in the test sequence. For $i = 0, 1, \ldots, L-1$, we omit t_i and recompute the fault coverage by simulating only the faults with $u_{det}(f) \geq u_i$ and the undetected faults. If the fault coverage after omission is not lower than the fault coverage before omission, we accept the change. Otherwise, we restore t_i.

The omission of a test vector t_i requires that t_{j+1} would be copied into t_j for $j = i, i+1, \ldots, L-2$. Instead of copying parts of the test sequence every time a vector is omitted, we use the simulation process described by Procedure 2. We use a variable called $omitted[i]$. We set $omitted[i] = 1$ if vector t_i is omitted, otherwise, $omitted[i] = 0$. If $omitted[i] = 0$, conventional simulation is carried out. If $omitted[i] = 1$, simulation under t_i is not required and the present state at time unit $i+1$ is equal to the present state at time unit i. Procedure 2 is described for a single fault f.

Procedure 2: Fault simulation with omitted vectors

1. Set P and P^f to be the all-unspecified initial states.
2. Set $i = 0$.
3. If $omitted[i] = 0$:
   a. Apply to the combinational logic of the fault-free/faulty circuits the input value t_i/t_i and the combined present-state P/P^f.
   b. Obtain the combined output z_i/z_i^f and next state N/N^f.
   c. If z_i and z_i^f conflict on any output, set $u_{det}(f) = u_i$ and stop.
   d. Set $P/P^f = N/N^f$.
4. Set $i = i + 1$. If $i < L$, go to Step 3.

The test compaction procedure is summarized next. Note that, even if a vector t_i cannot be omitted from the original sequence, after omitting a vector t_j, where $j > i$, it may become possible to omit t_i. To take advantage of this observation, the test sequence after vector omission is considered again until no additional vectors can be omitted. We point out that t_i and t_j are not considered explicitly by the procedure. Instead, an additional iteration is made as long as vectors were omitted in the previous iteration.

Procedure 3: Static compaction based on vector omission

1. Set $omitted[i] = 0$ for every $0 \leq i \leq L-1$. Fault simulate the test sequence and store the fault coverage in FC.
2. Set $i = 0$.
3. Set $omitted[i] = 1$ and fault simulate the test sequence (only undetected faults and faults with $u_{det}(f) \geq u_i$ need to be simulated).
4. If the fault coverage is smaller than FC, set $omitted[i] = 0$ and restore the detection times prior to the omission of vector i. Otherwise, store the new fault coverage in FC.
5. Set $i = i + 1$. If $i < L$, go to Step 3.
6. If $omitted[i] = 1$ for any vector i, rearrange the sequence by omitting the vectors with $omitted[i] = 1$ and go to Step 1.

The complexity of Procedure 3 is as follows: We denote by N_{iter} the number of iterations of Steps 1-6 performed by Procedure 2. Since the procedure terminates when no reduction in test length is achieved in an iteration over Steps 1-5, and assuming the worst case where the test length is reduced by one vector at every iteration, we have

$N_{iter} \leq L$. In each iteration over Steps 1-5, Steps 3-5 are repeated L times. Fault simulation in Step 3 has complexity $O(L|F_{det}|N_{lines})$, where N_{lines} is the number of circuit lines. The total complexity is $O(N_{iter}L^2|F_{det}|N_{lines})$.

In Step 3 of Procedure 3, we simulate only faults detected at or after the omitted vector and undetected faults. Faults detected before the omitted test vector do not have to be resimulated. To allow this saving in fault simulation, we must have the values of $\{u_{det}(f)\}$ updated for the current test sequence. For this reason, in Step 4 of Procedure 3, if t_i cannot be omitted, then the detection times are restored. This ensures that the values of $\{u_{det}(f)\}$ are updated. For simplicity of presentation, this feature is omitted from the variation of Procedure 3 described below and all the faults are resimulated there.

We observed through experimental results that, when the sequence to be compacted is long, there is a large number of input vectors at the beginning of the sequence that can be omitted without reducing the fault coverage. In addition, there are long subsequences of consecutive vectors starting at arbitrary time units in the test sequence that can be omitted. To take advantage of the existence of such subsequences and reduce the number of simulations performed by Procedure 3, we use binary search. Binary search is initiated starting from a vector t_i that can be omitted. The binary search terminates with the last vector t_j such that $T[u_i, u_j]$ can be omitted. The advantage of binary search is that instead of performing $j - i + 1$ simulations to omit $t_i, t_{i+1}, \ldots, t_j$ sequentially in Procedure 3, the binary search procedure performs $\lceil \log_2(L - i) \rceil$ simulations. A procedure for omitting vectors that uses binary search is given next.

Procedure 4: Binary search for subsequences that can be omitted

1. Set $START = 0$ ($START$ indicates the first vector to be considered for omission).
2. Set $omitted[i] = 0$ for every $0 \leq i \leq L - 1$. Fault simulate the test sequence and store the fault coverage in FC.
3. Set $i = START$.
4. Set $omitted[i] = 1$ and fault simulate the test sequence.
5. If the fault coverage is smaller than FC:
   a. Set $omitted[i] = 0$.
   b. Set $i = i + 1$. If $i < L$, go to Step 4.
   c. If $START = 0$, stop: All the vectors in the sequence (from $i = START = 0$ to $i = L - 1$) have been tried and no additional vectors can be omitted.
   d. ($START \neq 0$ and there may be vectors before $START$ that can be omitted.) Set $START = 0$ and go to Step 3.
6. (This step is reached if vector i can be omitted. Binary search is used to find the longest subsequence starting from i that can be omitted). Set $start = i$ ($start$ is used to store the first vector that can be omitted). Set $LB = start$ and $UB = L - 1$ (LB and UB are the boundaries of the binary search, between which we will find the last vector that can be omitted).
7. Set $MID = (LB + UB)/2$. Set $omitted[i] = 1$ for $start \leq i \leq MID$. Fault simulate the test sequence.
8. If the fault coverage is smaller than FC, set $UB = MID - 1$. Otherwise set $LB = MID + 1$.
9. If $LB \leq UB$, go to Step 7.
10. Execute the following steps (this step is reached when $LB > UB$, indicating that the binary search is complete and that the vectors from t_{start} to t_{UB} can be omitted):
    a. Rearrange the sequence by omitting the vectors $t_{start}, \ldots, t_{UB}$.
    b. Set L to be the new length of the test sequence. Set $START = start + 1$.
    c. Go to Step 2.

Procedure 3 (and its extension Procedure 4) can be viewed as a reverse order fault simulation procedure that attempts to omit vectors that were included to detect certain faults, but are no longer necessary in order to detect those faults once the test sequence is extended to detect additional faults. A different view of reverse order fault simulation that performs simulation starting from the end of the sequence and keeps vectors that are required to detect yet-undetected faults, is given at the end of Section 5.

5 Compaction Based on Vector Selection

The procedure described in this section is based on selection of subsequences to detect the faults in F_{det}, and proceeds as follows: For every fault, we first collect all the subsequences of the given sequence that detect the fault. A subsequence $T[u_s, u_e]$ is said to detect a fault f if $T[u_s, u_e]$ detects f when the circuit is started from the combined all-unspecified fault-free/faulty initial state at time unit u_s. The subsequence $T[u_s, u_e]$ is represented by a pair (s, e). After collecting all the subsequences that detect every fault, we use a covering procedure to select a minimal subset of subsequences to detect all the faults. During the covering procedure, if two subsequences (s_1, e_1) and (s_2, e_2) such that $s_1 \leq s_2 \leq e_1 \leq e_2$ are selected, we merge the two subsequences into the subsequence (s_1, e_2) and mark as detected all the faults with subsequences contained in this range. The following example demonstrates this process.

Example. We consider $s27$ under the test sequence shown in Table 8. Fault simulating the sequence starting from time unit 0, we find that fault 2/0 is detected at time unit 3, fault 3/0 is detected at time unit 4, fault 4/0 is detected at time unit 4, fault 6/1 is detected at time unit 3, fault 7/0 is detected at time unit 9, and so on. The corresponding subsequences are (0, 3), (0, 4), (0, 4), (0, 3), and (0, 9).

Next, we start simulation from time unit 1, setting the combined fault-free/faulty state at time unit 1 to the all-unspecified state. We find that fault 2/0 is detected at time unit 9, fault 3/0 is detected at time unit 10, fault 4/0 is detected at time unit 4, fault 6/1 is detected at time unit 9, fault 7/0 is detected at time unit 9, and so on. The corresponding subsequences are (1, 9), (1, 10), (1, 4), (1, 9), and (1, 9). For the fault 4/0, we now have two

TABLE 8
Test Sequence 3 of $s27$

i	0	1	2	3	4	5	6	7	8	9
t_i	1101	1011	0100	0111	0001	0100	1100	1111	0101	0011

i	10	11	12	13	14
t_i	0011	0101	1101	1110	0100

subsequences defined by (0, 4) and (1, 4). Since the first subsequence contains the second one, we omit the first and keep only (1, 4). Similarly, for the fault 7/0, we only keep the range (1, 9).

After considering every time unit as a starting point and finding detection times for all the faults, we obtain the subsequences shown in Table 9. We now select a subset of subsequences to detect all the faults. The subsequence (7, 9) is necessary to detect the faults 7/0 and 15/0. The subsequence (3, 5) is necessary to detect the fault 16/0. Once these subsequences are selected, additional faults are covered, including 2/0, 6/1, 8/0, 9/1, and so on. The subsequences for the remaining faults are shown in Table 10.

Next, we consider each one of the subsequences of Table 10 and repeatedly select the best one. The best subsequence is the one that, together with the subsequences already selected, covers the largest number of remaining faults and requires the smallest number of additional input vectors. For example, selecting subsequence (0, 4) detects six additional faults (3/0, 4/0, 9/0, 11/0, 12/0, and 15/1) and requires three additional vectors (t_0, t_1, and t_2; t_3 and t_4 were already selected). Selecting subsequence (9, 12) detects all eight faults. For example, fault 3/0 is detected since (7, 9) has already been selected. By adding (9, 12), we obtain the subsequence (7, 12), containing the subsequence (7, 10) that detects 3/0. In this case, we select the subsequence (9, 12).

In summary, we selected the subsequences (3, 5), (7, 9), and (9, 12), to result in the new sequence $T[u_3, u_5] \circ T[u_7, u_{12}]$.

TABLE 9
Test Subsequences out of Sequence 3 of $s27$

fault	subsequences
2/0	(0,3) (7,9) (10,12)
3/0	(0,4) (7,10)
4/0	(1,4) (7,10)
6/1	(0,3) (7,9)
7/0	(7,9)
8/0	(3,4) (8,10) (9,11)
8/1	(3,6) (9,12)
9/0	(1,4) (7,10)
9/1	(0,3) (7,9) (11,12)
11/0	(1,4) (7,10)
12/0	(0,4) (7,10)
13/1	(3,6) (9,12)
14/0	(3,5) (9,11)
14/1	(0,3) (5,6) (7,9) (11,12)
15/0	(7,9)
15/1	(0,4) (7,10)
16/0	(3,5)
17/0	(3,5) (9,11)
18/1	(0,3) (7,9) (11,12)
20/0	(0,3) (5,6) (7,9) (11,12)
21/0	(3,4) (9,10)
21/1	(0,0) (6,6) (7,7) (12,12) (13,13)
24/0	(3,5) (9,11)
24/1	(0,3) (7,9)
25/1	(3,4) (9,10)
26/0	(0,0) (6,6) (7,7) (12,12) (13,13)
26/1	(3,4) (9,10)

In the example above, we selected the subsequences independently, without considering the faults detected when two selected subsequences (s_1, e_1) and (s_2, e_2) are placed next to each other. This saves the simulation effort required to identify such faults; however, it may result in sequences that are longer than necessary. In our implementation of the selection procedure, after selecting a subsequence, we create a new test sequence made up of the selected subsequences in the order by which they appear in the original sequence. We then simulate the new sequence to identify the faults that still need to be detected. For example, suppose that the subsequences (9, 11), (1, 4), (7, 9), (4, 5) are selected in this order. After selecting (9, 11) and (1, 4), we simulate the sequence $T' = (t_1t_2t_3t_4t_9t_{10}t_{11})$. After selecting (7, 9), we simulate the sequence $T'' = (t_1t_2t_3t_4t_7t_8t_9t_{10}t_{11})$. After selecting (4, 5), we simulate the sequence $T''' = (t_1t_2t_3t_4t_5t_7t_8t_9t_{10}t_{11})$. In every case, we drop the faults detected and select the next subsequence based on the remaining faults. Note that the faults detected by T'' are not necessarily a superset of the faults detected by T' since the addition of t_7 and t_8 may prevent certain faults that were accidentally detected by placing the subsequences (1, 4) and (9, 11) consecutively from being detected.

TABLE 10
Test Subsequences for Remaining Faults

fault	subsequences
3/0	(0,4) (7,10)
4/0	(1,4) (7,10)
8/1	(3,6) (9,12)
9/0	(1,4) (7,10)
11/0	(1,4) (7,10)
12/0	(0,4) (7,10)
13/1	(3,6) (9,12)
15/1	(0,4) (7,10)

However, by selecting additional subsequences as long as undetected faults remain, we ensure that all the faults are detected by the final sequence obtained. The procedure is summarized next.

Procedure 5: Static compaction based on selection

1. For every $f \in F$, set $S(f) = \phi$.
2. For every time unit u_s, $u_0 \leq u_s \leq u_{L-1}$:
   For every fault f:
   If f is detected by the test sequence $T[u_s, L-1]$ when the combined state at time unit u_s is the all-unspecified state and the detection time is u_e, add (s, e) to $S(f)$.
3. Set $F_{det} = \{f : S(f) \neq \phi\}$. Set $F_{left} = F_{det}$. Set $selected[u] = 0$ for every time unit $u_0 \leq u \leq u_{L-1}$.
4. For every fault $f \in F_{left}$ such that $|S(f)| = 1$:
   a. Let $S(f) = \{(s, e)\}$.
   b. Call procedure $select(s, e)$ (this procedure, given below, updates the faults detected and the vectors included in the test sequence when (s, e) is selected).
5. If $F_{left} = \phi$ stop: The new test sequence contains every vector t_i such that $selected[u_i] = 1$, in the order they appear in T.
6. For every $(s', e') \in \cup\{S(f) : f \in F_{left}\}$:
   a. Using the sets $\{S(f)\}$, compute the number of faults in F_{left} which are detected if (s', e') is added to the subsequences already selected. Let this number be n'_{det}.
   b. Compute the number of time units u such that $u_{s'} \leq u \leq u_{e'}$ and $selected[u] = 0$. Let this number be n'_{add}.
7. Select the subsequence (s', e') for which n'_{det} is maximum. If a choice exists, select the one for which n'_{add} is minimum. Call Procedure $select(s', e')$.
8. Go to Step 5.

Procedure $select(s, e)$:

1. Set $selected[u] = 1$ for every time unit $u_s \leq u \leq u_e$.
2. Construct a new test sequence that contains every vector t_i such that $selected[u_i] = 1$, in the order they appear in T.
3. Find the set of faults F'_{det} detected by the test sequence constructed in Step 2. Define $F_{left} = F_{det} - F'_{det}$.

The worst-case complexity of Procedure 5 is as follows: In Step 2, we simulate every fault starting from every time unit. The complexity of this step is $O(|F_{det}|L^2N_{lines})$. Steps 5-8 are repeated at most $|F_{det}|$ times until all the faults are detected. In Step 6, we consider at most $O(L^2)$ pairs of time units. In Step 6.a, computing the number of faults detected corresponding to (s', e') has complexity $O(|F_{det}|L^2)$. The overall complexity is $O(|F_{det}|^2L^4 + |F_{det}|L^2N_{lines})$.

Procedure 5 can be modified into a reverse order fault simulation procedure that omits test vectors similar to Procedure 3. The advantage of the modified procedure compared to Procedure 5 is a reduced number of fault simulations. The modified procedure proceeds as follows: Starting from time unit $u_s = u_{L-1}$ and reducing u_s, we find the last subsequence of T that detects every fault. During this simulation process, if a fault f is detected for the first time (corresponding to the highest value of u_s) by a subsequence (s, e), then f is not considered under smaller values of u_s. At the end of the simulation process, we have, for every fault f, a single subsequence (s, e), where u_s is the last time unit after which f can still be detected by a subsequence of T. We create a new test sequence by including only vectors t_i such that $u_s \leq u_i \leq u_e$ for some fault f and omitting the other test vectors. For example, in the case of $s27$ and the sequence shown in Table 8, we find, from Table 9, that the last subsequences to detect the detected faults are as shown in Table 11. We keep the subsequences of Table 11 and omit the test vectors not included in them. The resulting test sequence is $(t_3t_4t_5t_7 \ldots t_{13})$. This test sequence can be further compacted by repeating the same procedure. Similarly to Procedure 3, this procedure omits test vectors appearing earlier in the sequence if there exist vectors later in the sequence that allow the same faults to be detected. The difference from Procedure 3 is in the order of fault simulation. Procedure 3 starts from the beginning of the test sequence. In contrast, the modification of Procedure 5 starts from the end of the test sequence. The advantage of Procedure 3 over the modified Procedure 5 is that a decision to omit a vector can be made immediately when it is considered. In the modified Procedure 5, vectors can be omitted only after the last detecting subsequences are found for all the faults. Thus, it is impossible to take into account, in the modified Procedure 5, faults which are detected because two subsequences that were previously separated become adjacent after the modification.

TABLE 11
Selection of Latest Subsequences

subsequence	faults detected
(13,13)	21/1 26/0
(11,12)	9/1 14/1 18/1 20/0
(10,12)	2/0
(9,12)	8/1 13/1
(9,11)	8/0 14/0 17/0 24/0
(9,10)	21/0 25/1 26/1
(7,10)	3/0 4/0 9/0 11/0 12/0 15/1
(7,9)	6/1 7/0 15/0 24/1
(3,5)	16/0

6 Experimental Results

We applied Procedure 1 (based on insertion), Procedure 4 (based on omission), and Procedure 5 (based on selection) to test sequences produced by different test generation procedures [2], [3], [4], [5], [6]. In Procedure 1, we used $N_{no\text{-}improve} = 100$ and a maximum test length of 15,000. Three of the test generation procedures whose test sequences we consider [4], [5], [6] do not use any special test compaction techniques. The procedure of [2] uses static compaction, and the procedure of [3] uses aggressive dynamic compaction that results in very short test sequences. Test sequences of other procedures, such as [14], [15], [16], [17], are not available to us at this time.

TABLE 12
Results of the Three Compaction Procedures

	original		Proc.4 omission		Proc.1 insertion		Proc.5 selection	
circuit	e.len	det	e.len	det	e.len	det	e.len	det
s208	614	132	122	136	150	135	76	132
s298	1007	265	90	265	93	265	116	265
s344	3411	329	59	329	156	329	73	329
s382	5354	357	548	357	557	357	751	357
s386	6742	274	108	311	2853	300	104	276
s400	5354	372	492	372	557	372	3026	372
s420	406	174	121	177	101	177	97	174
s444	2922	416	1706	416	2922	416	2922	416
s641	623	403	93	404	163	403	174	403
total	26433	2722	3339	2767	7552	2754	7339	2724

(a)

	original		Proc.4 omission		Proc.1 insertion		Proc.5 selection	
circuit	e.len	det	e.len	det	e.len	det	e.len	det
s298	259	265	87	265	114	265	153	265
s344	108	329	53	329	106	329	55	329
s400	2069	350	381	372	815	380	860	350
s420	166	179	124	179	147	179	137	179
s641	211	404	96	404	183	404	133	404
s820	968	813	424	814	907	814	772	813
total	3781	2340	1165	2363	2272	2371	2110	2340

(b)

(a) Test sequences of LOCSTEP [6]. (b) Test seqences of HITEC [5].

The results of test compaction by Procedure 1 (based on insertion), Procedure 4 (based on omission), and Procedure 5 (based on selection) are reported in Tables 12 and 13. In each table, the effective test length and the number of detected faults by the original test sequence is followed by the same information for the modified sequences after test compaction. In the last row of each table, we show the total test length and the total number of detected faults obtained by the corresponding procedure. We applied all three static compaction procedures only to some of the test sequences and some of the circuits.

Table 12a contains the results of applying static compaction to test sequences produced by *LOCSTEP* [6]. *LOCSTEP* is a test generation procedure based on logic simulation and it generates very long test sequences. Table 12b contains the results of applying static compaction to test sequences produced by *HITEC* [5]. All three compaction techniques yield large reductions in the length of the test sequences produced by *HITEC* and *LOCSTEP*. In most cases, Procedure 4 is the most effective of the three procedures proposed. These claims are supported by the total test lengths under the columns corresponding to the original sequences, and to Procedures 1, 4, and 5. It can also be seen that the test compaction techniques sometimes increase the number of faults detected by the test sequence. This is evident from the total numbers of detected faults under the corresponding columns.

Since Procedure 4 proved to be the most effective static compaction procedure for two types of test generation procedures considered in Table 12, we apply only Procedure 4 to additional test sequences and circuits. Table 13a contains the results of applying Procedure 4 to additional test sequences produced by *LOCSTEP* [6]. Table 13b contains the results of applying Procedure 4 to additional test sequences produced by *HITEC* [5]. Table 13c contains the results of applying Procedure 4 to test sequences produced by *FASTEST* [4]. Table 13d contains the results of applying Procedure 4 to test sequences produced by *SEQCOM'* [3]. Table 13e contains the results of applying Procedure 4 to test sequences produced by the procedure of [2]. The results reported in Table 13 demonstrate that the proposed static compaction procedure significantly reduces the lengths of test sequences produced by a variety of test generation procedures. This includes the procedure of [3] that uses memory-intensive dynamic compaction to produce test sequences that are already very short. Furthermore, in many cases, an increase in fault coverage is also obtained. In most cases, Procedure 4 went through only one or two iterations before no additional vectors could be removed.

Next, we consider the run time of Procedure 4. In the experiments reported above, our goal was to show the level of compaction achievable by Procedure 4 and the run time was not taken into account in the implementation of the procedure. In the experiments reported below, we use the following techniques to limit the run time of the procedure. These techniques have no impact or, in the worst case, have marginal impact on the level of compaction.

TABLE 13
Results of Procedure 4

	original		omission	
circuit	e.len	det	e.len	det
s526	7743	441	919	444
s820	5788	698	206	758
s1238	9409	1268	241	1269
s1423	9616	1274	365	1336
s1488	2427	1400	434	1439
total	34983	5081	2165	5246

(a)

	original		omission	
circuit	e.len	det	e.len	det
s1238	478	1283	247	1283
s1488	1192	1444	607	1444
total	1670	2727	854	2727

(b)

	original		omission	
circuit	e.len	det	e.len	det
s298	132	259	81	261
s344	88	329	52	329
s382	50	196	38	227
s386	121	287	77	299
s444	59	265	48	265
s641	130	402	65	402
s820	142	486	81	534
s1488	132	1093	76	1172
s1423	489	1293	258	1314
s5378	904	3612	882	3612
total	2247	8222	1658	8415

(c)

	original		omission	
circuit	e.len	det	e.len	det
s208	114	137	105	137
s298	160	265	110	265
s386	135	314	121	314
s420	149	179	108	179
s641	80	404	63	404
s1196	238	1232	185	1232
s1488	358	1444	317	1444
total	1234	3975	1009	3975

(d)

	original		omission	
circuit	e.len	det	e.len	det
s298	165	264	104	264
s344	83	329	37	329
s386	251	314	138	314
s400	618	365	388	373
s641	178	404	97	404
s1196	486	1239	244	1239
s1488	965	1444	605	1444
total	2746	4359	1613	4367

(e)

(a) Test sequences of LOCSTEP [6]. (b) Test sequences of HITEC [5]. (c) Test sequences of FASTEST [4]. (d) Test sequences of $SEQCOM'$ [3]. (e) Test sequences of [2].

1. When the input vector at time unit u is omitted, only faults detected at time unit u or later are simulated. More generally, when the input vectors between time units u_1 and u_2 are omitted during binary search, only faults detected at time unit u_1 or later are simulated. This has no effect on the level of compaction achieved, but reduces the number of faults that need to be simulated, especially for large values of u or u_1.
2. In Procedure 4, all the target faults (or all the target faults detected at or after a given time unit u, as well as undetected faults) are simulated every time an input vector or subsequence of input vectors are omitted from the test sequence. To reduce the number of faults that need to be simulated, we simulate only the faults detected by the original test sequence T. Thus, if F_{det} includes all the faults detected by T, then only faults in F_{det} are simulated during the compaction procedure. This implies that a fault $f \notin F_{det}$ which is detected by a modified test sequence T', derived during the compaction procedure, is not recorded. Moreover, if a fault $f \in F_{det}$ is not detected by such a modified input sequence T', then the last step of vector omission leading to T' is not accepted even if T' detects a larger number of faults than T. This may have a marginal impact on the final test length obtained. The sequence may be shorter or longer than the sequence obtained without this technique. To capture faults that are detected by the final test sequence and not included in F_{det}, the final test sequence can be simulated after compaction terminates.

In our implementation of Procedure 4, we do not use any techniques such as parallel fault or parallel pattern simulation to speed up the fault simulation process itself. As a result, the run times of Procedure 4 are high. To report the run time in a meaningful way, we normalize it to the time it takes to fault simulate the given test sequence T. We denote by rt_{base} the time it takes to simulate all the target faults under T. During the application of Procedure 4, we record its run time after every application of binary search that yields a modified test sequence T'. We denote the time to obtain T' by rt'. We report the normalized run time $\frac{rt'}{rt_{base}}$ for every such modified sequence in the form of a graph where the test sequence length is plotted as a function of the normalized run time. The results using the test sequences generated by *HITEC* [5] for $s298$ and $s400$ are reported in Fig. 2. The vertical dashed lines in the figures indicate the end of an iteration of Procedure 4. An iteration starts when *START* is set to 0 either in Step 1 or in Step 5.d of Procedure 4 and the input vectors are considered for omission starting with the test vector at time unit 0. Additional results are reported in Table 14. For each iteration, we show in Table 14 the test length and the normalized run time.

The following conclusions can be drawn from Fig. 2 and Table 14.

1. Most of the compaction occurs in the first iteration of Procedure 4, although only a fraction of the total run time is spent in this iteration. Thus, it is possible to terminate Procedure 4 after its first iteration at a

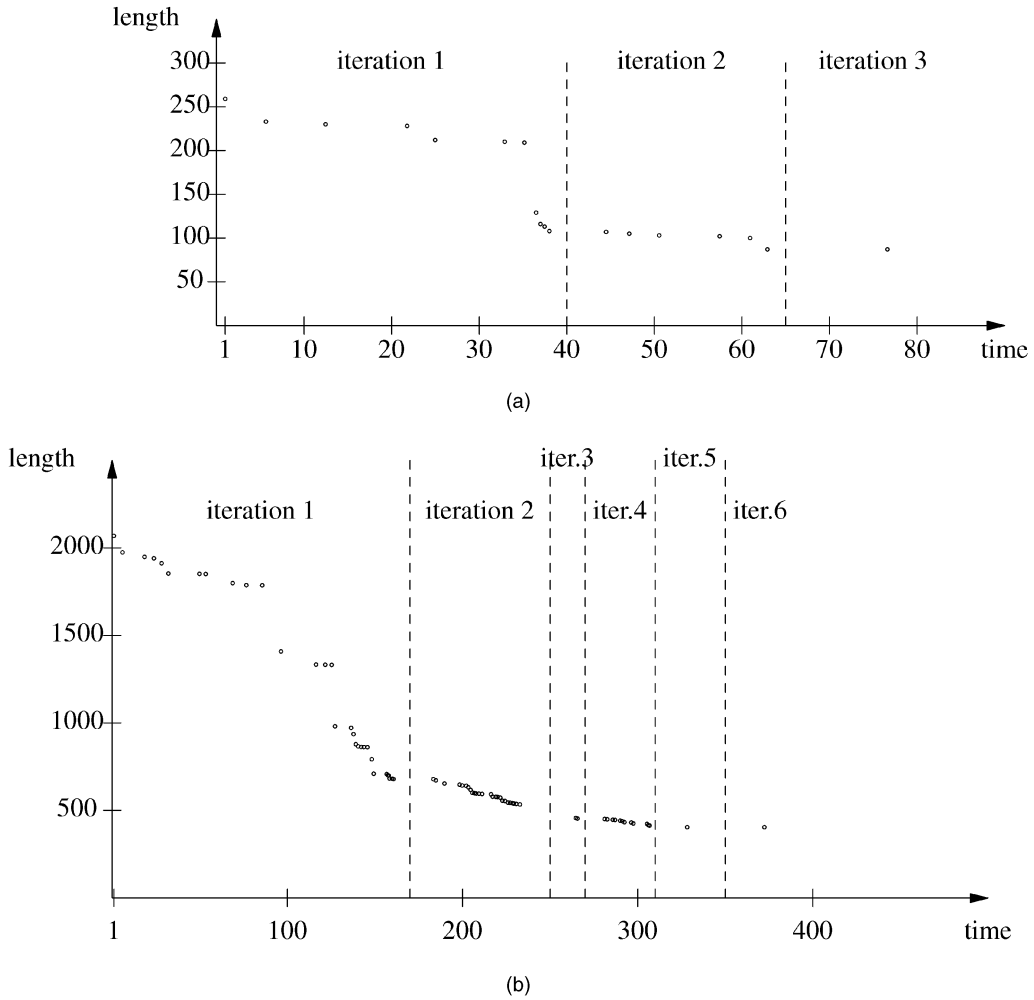


Fig. 2. Normalized run times. (a) $s298$. (b) $s400$.

significant reduction in run time but without a significant loss in compaction.

2. A large number of compaction steps result in relatively small reductions in test length. When only a small number of vectors are omitted, sequential search is faster than binary search. For example, consider a case where binary search is performed over 1,024 input vectors, requiring 10 passes of fault simulation. Suppose that only five vectors are omitted as a result. In this case, sequential search that omits the vectors one at a time may be faster than binary search. Careful selection between sequential search and binary search can be used to further reduce the run time of the procedure.

7 Concluding Remarks

We proposed three static compaction techniques for test sequences of synchronous sequential circuits. The first technique duplicated subsequences of the test sequence and inserted them at prior time units in an attempt to achieve earlier detection of faults. The second technique omitted superfluous input vectors. Binary search was used to identify subsequences that can be omitted. The third

TABLE 14
Normalized Run Times

	orig	iter. 1		iter. 2		iter. 3		iter. 4		iter. 5		iter. 6	
circuit	len	len	time	len	time	len	time	len	time	len	time	len	time
s298	259	108	38.02	87	62.92	87	76.62						
s344	108	53	31.74	53	43.24								
s400	2069	680	160.57	535	232.76	454	265.55	414	306.84	405	328.34	405	372.36
s420	166	128	10.08	124	15.52	124	24.10						
s641	211	106	66.03	96	74.00	96	95.66						
s820	968	447	437.35	424	524.00	424	611.66						

technique analyzed the coverage of subsequences of the test sequence and used a covering procedure to select a minimal subset. Comparison of the three techniques on test sequences generated for benchmark circuits by various test generation procedures showed that omission is the most effective as a static compaction technique. The results also show that test sequences generated by various test generation procedures can be significantly compacted. The compacted sequences thus have shorter test application times and smaller memory requirements. In addition, the ability to significantly reduce the length of the test sequences indicates that it may be possible to reduce test generation time if superfluous input vectors are not generated.

ACKNOWLEDGMENTS

The research reported in this paper was supported in part by U.S. National Science Foundation Grant No. MIP-9357581. This work is based on "On Static Compaction of Test Sequences for Synchronous Sequential Circuits," which appeared in the *Proceedings of the ACM/IEEE 33rd Design Automation Conference*, pp. 215-220, June 1996.

REFERENCES

[1] 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," *Proc. Int'l Conf. Computer-Aided Design,* pp. 382-385, Nov. 1988.

[2] B. So, "Time-Efficient Automatic Test Pattern Generation Systems," PhD thesis, Electrical Eng. Dept., Univ. of Wisconsin at Madison, 1994.

[3] I. Pomeranz and S.M. Reddy, "On Generating Compact Test Sequences for Synchronous Sequential Circuits," *Proc. European Design Automation Conf. '95,* pp. 105-110, Sept. 1995.

[4] T.P. Kelsey and K.K. Saluja, "Fast Test Generation for Sequential Circuits," *Proc. Int'l Conf. Computer-Aided Design,* pp. 354-357, Nov. 1989.

[5] T. Niermann and J.H. Patel, "HITEC: A Test Generation Package for Sequential Circuits," *Proc. European Design Automation Conf.,* pp. 214-218, 1991.

[6] I. Pomeranz and S.M. Reddy, "LOCSTEP: A Logic Simulation Based Test Generation Procedure," *Proc. 25th Fault-Tolerant Computing Symp.,* pp. 110-119, June 1995.

[7] I. Pomeranz and S.M. Reddy, "Vector Restoration Based Static Compaction of Test Sequences for Synchronous Sequential Circuits," *Proc. Int'l Conf. Computer Design,* pp. 360-365, Oct. 1997.

[8] M.S. Hsiao, E.M. Rudnick, and J.H. Patel, "Fast Algorithms for Static Compaction of Sequential Circuit Test Vectors," *Proc. VLSI Test Symp.,* pp. 188-195, Apr. 1997.

[9] M.S. Hsiao and S.T. Chakradhar, "State Relaxation Based Subsequence Removal for Fast Static Compaction in Sequential Circuits," *Proc. Conf. Design Automation and Test in Europe,* pp. 577-582, Feb. 1998.

[10] R. Guo, I. Pomeranz, and S.M. Reddy, "Procedures for Static Compaction of Test Sequences for Synchronous Sequential Circuits Based on Vector Restoration," *Proc. Conf. Design Automation and Test in Europe,* pp. 583-587, Feb. 1998.

[11] S.K. Bommu, S.T. Chakradhar, and K.B. Doreswamy, "Static Test Sequence Compaction Based on Segment Reordering and Accelerated Vector Restoration," *Proc. 1998 Int'l Test Conf.,* pp. 954-961, Oct. 1998.

[12] S.K. Bommu, S.T. Chakradhar, and K.B. Doreswamy, "Static Compaction Using Overlapped Restoration and Segment Pruning," *Proc. Int'l Conf. Computer-Aided Design,* pp. 140-146, Nov. 1998.

[13] R. Guo, I. Pomeranz, and S.M. Reddy, "On Speeding-Up Vector Restoration Based Static Compaction of Test Sequences for Sequential Circuits," *Proc. Seventh Asian Test Symp.,* pp. 467-471, Nov. 1998.

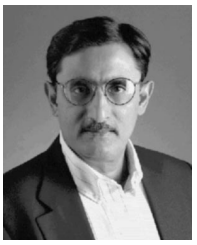
[14] D.G. Saab, Y.G. Saab, and J.A. Abraham, "CRIS: A Test Cultivation Program for Sequential VLSI Circuits," *Proc. Int'l Conf. Computer-Aided Design,* pp. 216-219, Nov. 1992.

[15] E.M. Rudnick, J.H. Patel, G.S. Greenstein, and T.M. Niermann, "Sequential Circuit Test Generation in a Genetic Algorithm Framework," *Proc. Design Automation Conf.,* pp. 698-704, June 1994.

[16] M. Abramovici, M.A. Breuer, and A.D. Friedman, *Digital Systems Testing and Testable Design.* Computer Science Press, 1990.

[17] W.-T. Cheng and T.J. Chakraborty, "Gentest: An Automatic Test Generation System for Sequential Circuits," *Computer,* pp. 43-49, Apr. 1989.

Irith Pomeranz received the BSc degree (summa cum laude) in computer engineering and the DSc degree from the Department of Electrical Engineering at the Technion-Israel Institute of Technology 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 at the University of Iowa, where she is currently a professor. Her research interests are testing of VLSI circuits, design for testability, synthesis, and design verification. Dr. Pomeranz is a recipient of the U.S. National Science Foundation Young Investigator Award in 1993 and of the University of Iowa Faculty Scholar Award in 1997. She serves as associate editor of the *ACM Transactions on Design Automation*. She served as guest editor of the *IEEE Transactions on Computers* January 1998 special issue on dependability of computing systems and as program cochair of the 1999 IEEE Fault-Tolerant Computing Symposium. Dr. Pomeranz is a fellow of the IEEE.



Sudhakar Reddy obtained his undergraduate degree in electrical and communication engineering from Osmania University, his MS degree from the Indian Institute of Science, and his PhD degree in electrical engineering from the University of Iowa, Iowa City. Dr. Reddy has been active in the areas of testable designs and test generation for logic circuits since 1972. He has been an associate editor and a guest editor twice of the *IEEE Transactions on Computers*. He is an associate editor of the *IEEE Transactions on Computer-Aided Design*. 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 a professor. In 1990, he was made a University of Iowa Foundation Distinguished Professor. Dr. Reddy is a fellow of the IEEE and a member of Tau Beta Pi, Eta Kappa Nu, and Sigma Xi.