Fast Static Compaction Algorithms for Sequential Circuit Test Vectors

Michael S. Hsiao, *Member*, *IEEE*, Elizabeth M. Rudnick, *Member*, *IEEE*, and Janak H. Patel, *Fellow*, *IEEE*

Abstract—Two fast algorithms for static test sequence compaction are proposed for sequential circuits. The algorithms are based on the observation that test sequences traverse through a small set of states and some states are frequently revisited throughout the application of a test set. Subsequences that start and end on the same states may be removed if necessary and if sufficient conditions are met for them. Contrary to the previously proposed methods, where multitudes of fault simulations are required, the techniques described in this paper require only two fault simulation passes and are applied to test sequences generated by various test generators, resulting in significant compactions very quickly for circuits that have many revisited states.

Index Terms—Static test set compaction, test generation, recurrence subsequence, fault simulation.

1 Introduction

Test sequence compaction produces test sequences of reduced lengths, which can greatly reduce the test application time. Test application time directly impacts the cost of testing and, thus, shorter test sequences are desired. Two types of compaction techniques exist: dynamic and static compaction. Dynamic test sequence compaction performs compaction concurrently with the test generation process and often requires modification of the test generator. Static test sequence compaction is done in a postprocessing step to test generation and is independent of the test generation algorithm and process. If dynamic compaction is used within the test generator, static compaction can further reduce the test set size after the test generation process is finished.

Several static compaction approaches for sequential circuits have been proposed in the past [1], [2], [3]. The static compaction techniques proposed in [1], [2] use overlapping and reordering of test sequences obtained from targeting individual faults to produce a minimal-length test set. Three static compaction techniques have been proposed by Pomeranz and Reddy [3] (which use multiple fault simulation passes) and they have shown that subsequences can often be removed from a test without reducing the original fault coverage. The three compaction techniques [3] are based on vector insertion, omission, or selection. When a vector is to be omitted or swapped, the fault simulator is invoked to make sure the fault coverage is not affected by the alteration in the test sequence. These techniques produce very compact test sets at the expense of long execution times; however, they may not be practical for very large circuits and/or large original test sets because of the large number of fault simulations required.

Our approach to test set compaction is based on the observation that test sequences traverse through a small set of states, and some states are frequently revisited. Table 1 shows the number of vectors and states traversed by the HITEC [4], [5] test sets for some ISCAS89 [6] benchmark circuits. Since the numbers of states visited is small relative to the total number of vectors for most circuits, it can be concluded that many subsequences that start and end on the same states exist within these test sets. Test sets generated by other test generators also exhibit similar phenomena. The subsequences that start and end on the same state may be removed from a test set if necessary and sufficient conditions on them are met. For circuits that have few or no repeated states, such as s1423 and s5378 in this table, this technique will not be applicable. The proposed technique is fast because only *two* fault simulation passes through the test set are necessary for compaction. Our static compaction is applied to test sets generated by various test generators; significant reductions in test set lengths have been obtained. In addition, long sequences for large circuits can be handled.

The remainder of the paper is organized as follows: Section 2 gives some definitions of terms for this work. Sections 3 and 4 describe two compaction algorithms in detail. The compaction framework for a given test set is explained in Section 5, experimental results are given in Section 6, and Section 7 concludes the paper.

2 Definitions

Several definitions are given below in regard to subsequences in a test set T. The notation for a subsequence T_{sub} is $T[v_i, v_{i+1}, \ldots, v_j]$, where v_i and v_j are the ith and jth vectors in the test set T, respectively.

- *M.S. Hsiao is with the Department of Electrical and Computer Engineering, Rutgers University, Piscataway, NJ 08855. E-mail: mhsiao@ece.rutgers.edu.*
- *E.M. Rudnick and J.H. Patel are with the Center for Reliable and High-Performance Computing, Department of Electrical and Computer Engineering, University of Illinois, 1308 W. Main Street, Urbana, IL 61801.*

Manuscript received 5 Aug. 1997.
For information on obtaining reprints of this article, please send e-mail to: tc@computer.org, and reference IEEECS Log Number 105462.

TABLE 1
Number of States Traversed by HITEC Test Sets

Circuit	Vec	States	Circuit	Vec	States
s298	292	137	s832	1136	24
s344	127	113	s1196	435	294
s382	2074	646	s1238	475	332
s400	2214	690	s1423	150	150
s444	2240	592	s1488	1170	47
s526	2258	625	s1494	1245	47
s641	209	103	s5378	912	912
s713	173	85	s35932	496	381
s820	1114	24			

Vec: Test Set Size *States: # States Traversed*

Definition 1. *A* propagation subsequence T^f_{prop} *for a particular fault* f *is a subsequence* $T[v_i, v_{i+1}, \ldots, v_j]$ *such that the fault effects of* f*, stored in the starting state at vector* v_i*, are propagated through all time frames within the subsequence.*

Definition 2. *A* detection subsequence T^f_{det} *for a particular fault* f *is a subsequence* $T[v_i, v_{i+1}, \ldots, v_{j-1}, v_j]$ *such that* f *is activated in time frame* i*,* $T[v_{i+1}, \ldots, v_{j-1}]$ *is a propagation subsequence for* f *and the fault* f *is detected in time frame* j*.*

Both the propagation and detection subsequences for a particular fault are identified approximately and pessimistically in this work; some of the initial vectors of the subsequence may not contribute to the actual propagation of the fault, as shown in Fig. 1. In order to reduce the complexity of obtaining propagation and detection subsequences, backtracing of fault effects through the circuit is avoided, resulting in a continuous sequence of time frames in which fault effects are propagated to the flip-flops. These continuous sequences are then taken as the propagation or detection subsequence, even though these are pessimistic measures.

Definition 3. *A* state-recurrence subsequence T_{rec} *is a subsequence of vectors* $T[v_i, v_{i+1}, \ldots, v_j]$ *such that the fault-free states reached at the end of vectors* v_{i-1} *and* v_j *are identical.*

Definition 4. *An* inert recurrence subsequence *or, simply,* inert subsequence, T_{inert} *is a state-recurrence subsequence* $T_{rec}[v_i, v_{i+1}, \ldots, v_j]$ *such that no additional faults are detected within the subsequence* T_{rec}*.*

Definition 5. *Given a fault-free state* S*, the* error vector E_f *for a particular fault* f *is equal to* $S \oplus S_f$*, where* S_f *is the corresponding faulty state for the same time frame.*

The error vector indicates which flip-flops carry fault effects.

Definition 6. *Given two identical fault-free states* S*, the error vector* E_f *for a fault* f covers *another error vector* E'_f *for the same fault and state if* $E_f \bigcup E'_f = E_f$*.*

For example, if the fault-free state S is 10110, S_f is 11011 and S'_f is 10011, error vectors E_f and E'_f would be 01101 and 00101, respectively, and E_f covers E'_f. However, E'_f does not cover E_f because a fault effect has propagated to the second flip-flop only in S_f. In the case where S_f equals S'_f, the two error vectors are said to cover one another.

3 Inert Subsequence Removal

Many inert subsequences may exist within a test set and many of them may be removable. Consider the test sequence described in Table 2. An inert subsequence is present: vectors v_4 to v_6. Given necessary boundary conditions for the propagation and detection subsequences of faults f_4 and f_8, which are detected by vector v_7, this subsequence $T_{inert}[v_4 \ldots v_6]$ may be removed from the test set without affecting the fault coverage. These necessary conditions will be discussed later in this section, but,

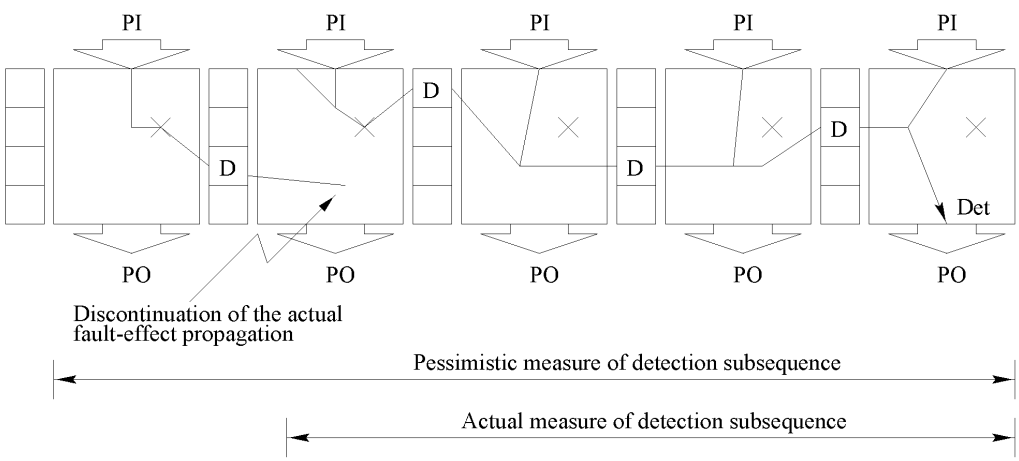

Fig. 1. Pessimistic measure of detection subsequence.

TABLE 2
Example Test Sequence

Vector	Next State	Detected Faults
v_1	A	f_1, f_6, f_7
v_2	B	f_2, f_9, f_{11}
v_3	C	f_3
v_4	D	
v_5	E	
v_6	C	
v_7	F	f_4, f_8

intuitively, the conditions check whether the detections of faults f_4 and f_8 depend on the inert subsequence.

The algorithm for removing inert subsequences is very efficient. Theoretically, it can be implemented in a *single* fault simulation pass at a high cost of storing faulty state information. Instead of a single fault simulation pass, we use a two-pass algorithm. A normal fault simulation pass through the test set is made to collect the inert subsequences and detection sequences for all faults within the test set. Then, a second pass is made to obtain faulty state information for the detected faults at the boundaries of the inert subsequences only. Much storage can be saved by skipping the faulty states that do not fall on the boundaries of inert subsequences. Once the data is collected, each inert subsequence is analyzed for possible removal. An inert subsequence may be removed if any one of the following four criteria are met.

Criterion 1. *For an inert subsequence $T_{inert}[v_i, \ldots, v_j]$, if faulty state S_f^{i-1} at the end of time frame $i-1$ and faulty state S_f^j at the end of time frame j are identical for every undetected fault f which is activated at time frames $i-1$ and j, T_{inert} can be removed.*

This is the trivial case, since the combined fault-free/faulty states at time frames $i-1$ and j are identical. With no faults detected within the subsequence T_{inert}, T_{inert} can be removed from the test set without affecting the fault coverage.

Criterion 2. *For an inert subsequence $T_{inert}[v_i, \ldots, v_j]$, if error vector E_f^j at the end of time frame j covers E_f^{i-1} at the end of time frame $i-1$ for every activated fault f and the additional fault effects propagated at time frame j do not lead to detection, T_{inert} can be removed. Fig. 2 illustrates this case.*

Detection subsequences for the additional fault effects at the end of the inert subsequence are used to determine whether these fault effects lead to a detection. When they do

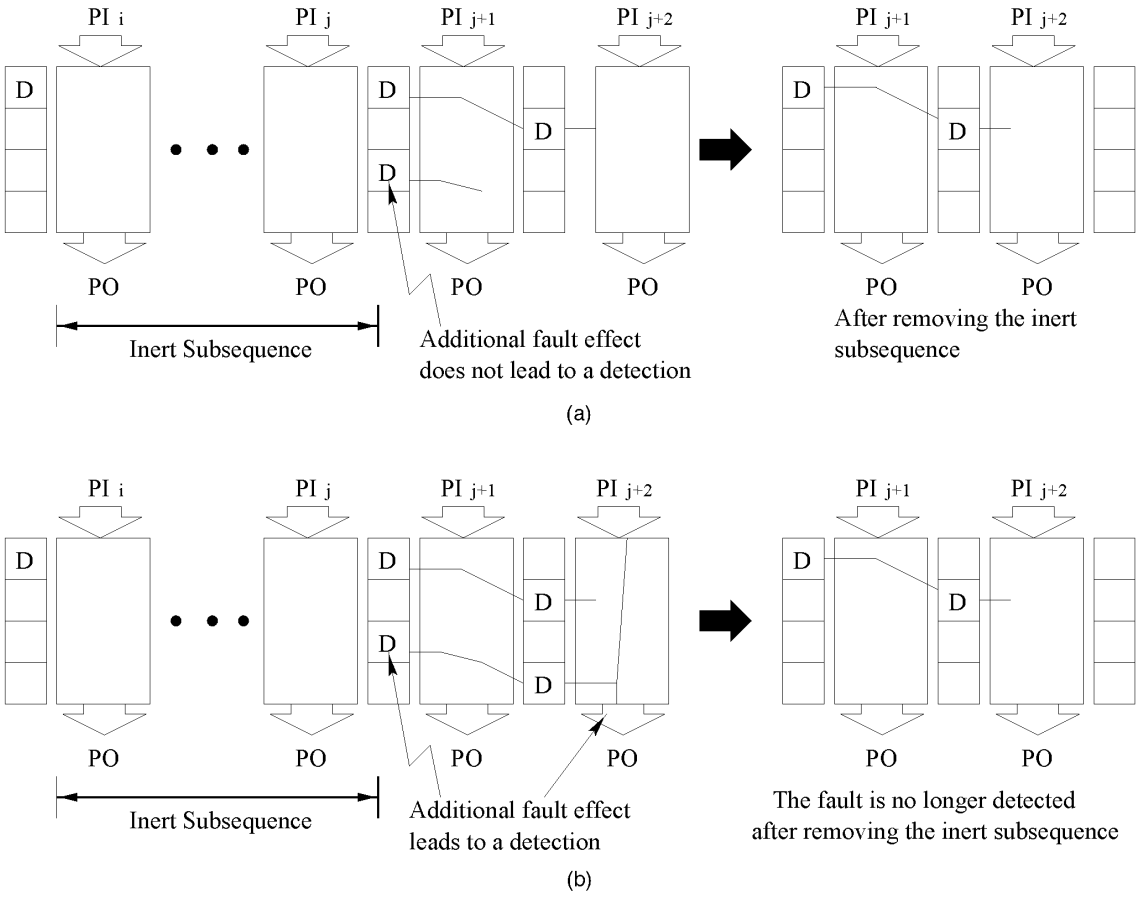

Fig. 2. Cases of Criterion 2. (a) Additional fault-effects do not lead to detection. (b) Additional fault-effects lead to detections.

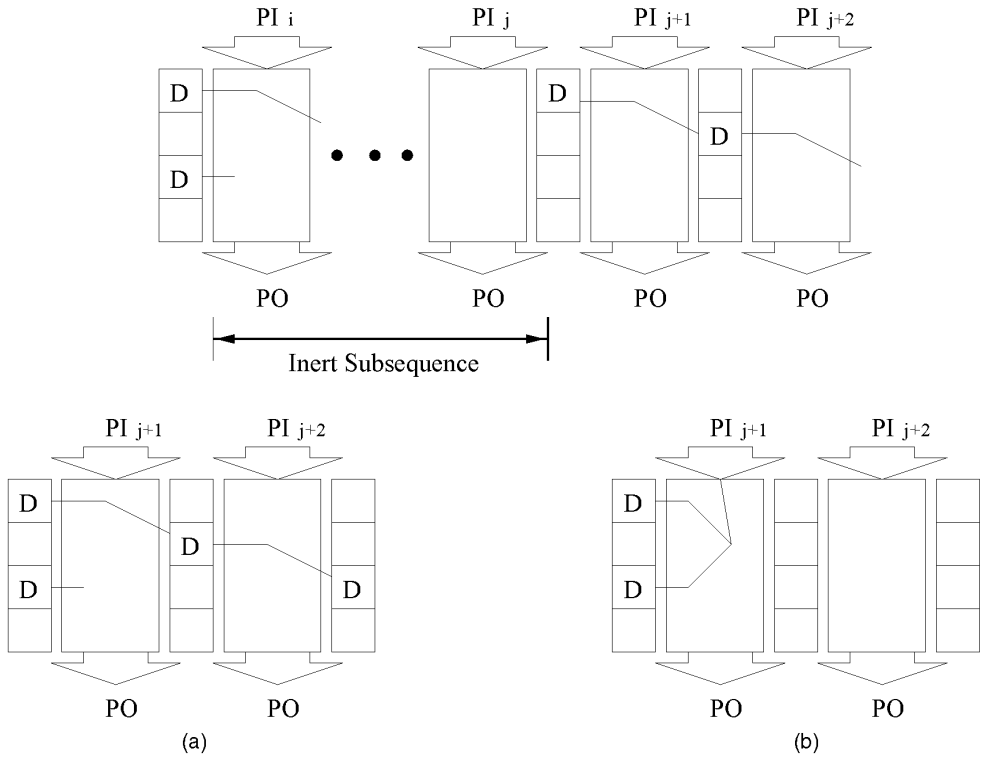

Fig. 3. Cases of Criterion 3. (a) No masking when fault-effects applied in frame j + 1. (b) Fault masking occures when applied in time frame j + 1.

not lead to a detection (as shown in Fig. 2a), eliminating the inert subsequence will not cause adverse effects. However, if the additional fault effects do lead to a detection, the fault would no longer be detected if the inert subsequence was removed, as depicted in Fig. 2b. Thus, an inert subsequence is not removed if the additional fault effects at time frame j are part of the detection subsequences.

Criterion 3. *For an inert subsequence $T_{inert}[v_i, \ldots, v_j]$, if error vector E_f^{i-1} at the end of time frame $i-1$ covers E_f^j at the end of time frame j for every activated fault f, T_{inert} can be removed if the additional fault effects propagated at time frame $i-1$ do not cause fault-masking in time frames starting at frame $j+1$, as shown in Fig. 3.*

Because the additional fault effects at the beginning of time frame i are not present at the end of time frame j, it is intuitive that these fault effects cease to propagate at some point within the inert subsequence. If the inert subsequence is removed, the additional fault effects at the end of time frame $i-1$ will appear at the beginning of time frame $j+1$. Fig. 3 shows the scenario where the additional fault effect may cause fault masking once the subsequence is removed. From the figure, once the inert subsequence is removed, the additional fault effects at the end of time frame $i-1$ will appear at the beginning of time frame $j+1$. Since the primary input vector PI_{j+1} may very well differ from PI_i, the differences in the primary inputs may cause fault masking to occur, as depicted in Fig. 3b. As a result, multiple fault effects may cancel each other.

Criterion 4. *For an inert subsequence $T_{inert}[v_i, \ldots, v_j]$, if neither error vectors E_f^{i-1} nor E_f^j cover the other, conditions imposed on activated faults in both Criteria 2 and 3 need to be satisfied in order for the inert subsequence T_{inert} to be removed.*

With the four criteria given, the compaction algorithm simply gathers all of the inert subsequences and checks for any match with the removal criteria. The pseudocode for this algorithm is described in Fig. 4. This is a fast inert-subsequence removal algorithm because only the inert subsequences are considered for compaction. A simple extension of this algorithm is possible based on the following observation: There may exist many state-recurrence subsequences that are unnecessary since the faults detected within these subsequences can be detected outside the subsequences. The next algorithm removes unnecessary state-recurrence subsequences that are not known to be inert using fault simulation with fault dropping.

```
inert_subsequence_removal()
    Collect detection and inert subsequences via fault
        simulation for all faults, dropping detected faults

    For each inert subsequence T_inert_i collected
        If any of 4 removal criteria satisfied
            Remove T_inert_i from the test set
```

Fig. 4. Inert subsequence removal algorithm.

TABLE 3
Test Sequence with State-Recurrence Subsequence

Vector	Next State	Detected Faults
v_1	A	f_1, f_6, f_7
v_2	B	f_9, f_{11}
v_3	C	f_3
v_4	D	f_2, f_8
v_5	E	f_5
v_6	C	
v_7	F	f_4, f_8
v_8	G	f_2, f_5
v_9	B	f_8

$T[v_3, \ldots, v_9]$ *and* $T[v_4, \ldots, v_6]$ *are two state-recurrence subsequences.*

4 Recurrence Subsequence Removal

Many state-recurrence subsequences exist within the test sets generated by both deterministic and simulation-based test generators. Deterministic test generators backtrace until all flip-flops have don't care (**X**) values for each target fault. Thus, the initial vectors of each test sequence derived act as synchronizing sequences for the circuit. Consequently, many of these *synchronized* states are visited repeatedly in the test set. In simulation-based test generators, states are repeatedly visited as well because only forward processing is involved and easy-to-reach states are often revisited.

In terms of fault detection properties, typically, an easy fault in the circuit is detected multiple times by the test set. An easy fault is defined as one which requires only a few constraints on the primary inputs and flip-flop state in order for detection; thus, a large number of combinations of possible vector sequences may detect an easy fault. This observation, together with the fact that many state-recurrence subsequences reside within the test set, allows the possibility of reducing the test set size by removing state-recurrence subsequences that only detect easy faults. In order to identify multiple detections by the test set, *fault simulation without fault dropping* is necessary, starting from the occurrence of the first state-recurrence subsequence.

Table 3 gives an example of a test set that has state-recurrence sequences $T_{rec}[v_3 v_9]$ and $T_{rec}[v_4 \ldots v_6]$. Some faults are detected within each subsequence; thus, neither subsequence is inert. However, the three faults that are detected by the state-recurrence subsequence $T_{rec}[v_4 \ldots v_6]$, namely faults f_2, f_8, and f_5, are easy faults and are detected also by vectors v_7 and v_8. If the *detection subsequences* for these three faults do not overlap with the state-recurrence subsequence $T_{rec}[v_4 \ldots v_6]$, and if the inert subsequence removal criteria described in the previous section are met at the boundaries, the state-recurrence subsequence may be safely removed from the test set. The criteria for removing state-recurrence subsequences can then be formally stated as follows:

Criterion 5. *For a recurrence subsequence $T_{rec}[v_i, \ldots, v_j]$, T_{rec} can be removed if the following four conditions are met:*

1. *All faults detected within T_{rec} have detection subsequences that do* not *overlap with T_{rec}.*
2. *For each fault active at the end of time frame j, if error vector E_f^j at the end of time frame j covers error vector E_f^{i-1} at the end of time frame $i-1$, the additional fault effects propagated at time frame j do not lead to detection.*
3. *For each fault active at the end of time frame $i-1$, if error vector E_f^{i-1} at the end of time frame $i-1$ covers error vector E_f^j at the end of time frame j, the additional fault effects propagated at time frame $i-1$*

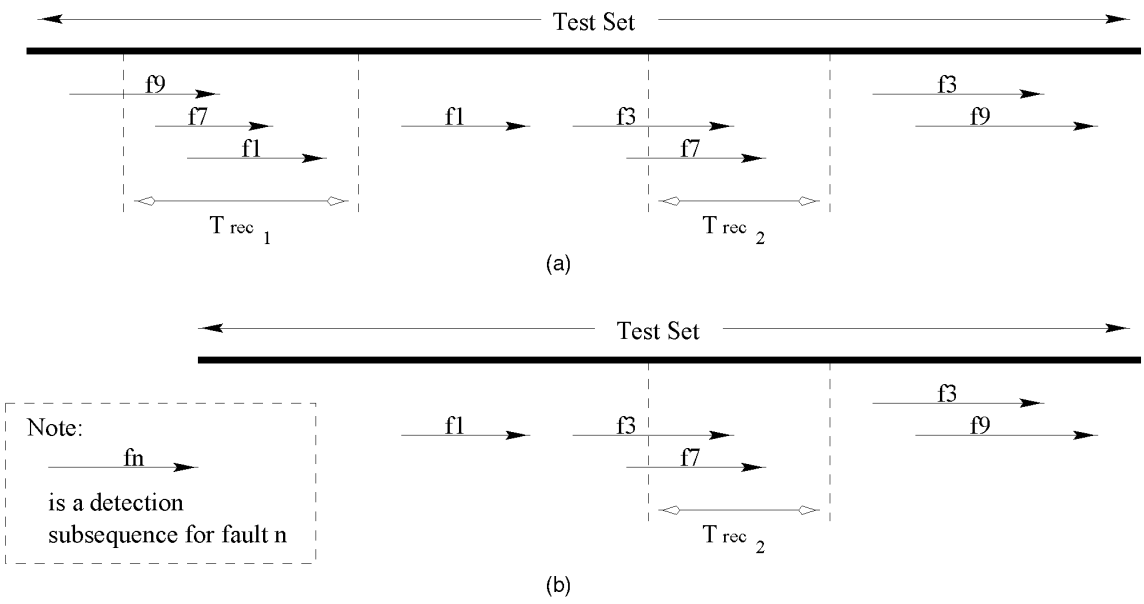

Fig. 5. Recurrence subsequence removal. (a) Test sequence before removal of any state-recurrence subsequence. (b) Test sequence after removal of first state-recurrence subsequence.

```
recurrence_subsequence_removal()
  Collect detection & state-recurrence subsequences via
    fault simulation for all faults without fault-dropping

  For each state-recurrence subsequence T_rec_i collected
    If boundary conditions & removal criteria satisfied
      Remove T_rec_i from the test set
```

Fig. 6. Recurrence subsequence removal algorithm.

do not cause fault-masking in time frames starting at frame $j+1$.

4. *For each fault active at the end of time frames $i-1$ and j, if neither error vector E_f^{i-1} nor error vector E_f^j covers the other, conditions imposed on activated faults in items 2 and 3 above are satisfied.*

Conditions 2, 3, and 4 are not strict necessary conditions, since faults which violate these conditions may be detected multiple times.

An example of removal of state-recurrence subsequences is illustrated in Fig. 5. In Fig. 5a, all faults detected within the state-recurrence subsequence T_{rec_1}, faults f_1, f_7, and f_9, have additional detection subsequences that do not overlap with T_{rec_1} itself, so T_{rec_1} can safely be removed from the test set if the criteria for faults active at the beginning and end of the sequence are met. After the removal of T_{rec_1}, all three faults f_1, f_7, and f_9 are still detected in the compacted test set, shown in Fig. 5b. Now, if T_{rec_2} were to be removed as well, both f_3 and f_7 also need to have detection subsequences outside of T_{rec_2}. However, in this case, f_3 is the only fault within T_{rec_2} which has a detection subsequence outside of T_{rec_2}, since the other detection subsequence of f_7 has already been removed. Therefore, T_{rec_2} may not be removed from the test set.

Again, a *single* pass of fault simulation is sufficient to carry out the algorithm. However, two passes are used here, again in order to save storage costs. The pseudocode for recurrence-subsequence removal is shown in Fig. 6.

Compaction using the inert-subsequence removal of Section 3 requires inert subsequences to be present in the test set for the algorithm to be effective. Recurrence-subsequence removal, on the other hand, attempts to compact the test sets across several inert subsequences in a state-recurrence subsequence if the faults inside the state-recurrence subsequence can be detected elsewhere. Other differences exist between the two algorithms. Inert-subsequence removal compacts the test set in a **local** fashion: the compaction proceeds from the *locally inert* subsequences. Recurrence-subsequence removal, on the contrary, compacts the test set in a **global** manner: The compaction starts with the *globally* longest *state-recurrence* subsequence.

The cost of fault simulation without fault dropping may be large in larger circuits. To handle such circuits, many faults may be dropped by the initial vectors in the test set; thus, fault simulation without fault dropping is only performed for the remainder of the faults, making the process less expensive.

5 Compaction Framework

Application of the inert-subsequence and recurrence-subsequence removal algorithms involves the selection of a set of inert and state-recurrence subsequences. Optimal selection is similar to the set covering problem, which is NP-complete in complexity. Let us consider the test set shown in Fig. 7, with the inert subsequences illustrated. Determining the optimal selection of subsequences to remove is nontrivial; a greedy algorithm is chosen to accomplish this task since it has been shown to give a near-optimal solution to the set covering problem in polynomial time [7]. Although our greedy algorithm only achieves near-optimal selection of subsequences for removal, it will be shown in the experimental results section that significant compactions are still achieved.

Each inert subsequence T_{inert_i} in Fig. 7 has to be examined for the boundary criteria for removal. In the case of T_{inert_1} and T_{inert_2}, they both start from the same vector but end at different vectors. If both subsequences satisfy the criteria for inert-subsequence removal, T_{inert_2} is preferable for removal, unless a combination of T_{inert_1} and other inert subsequence(s) results in a more compact test set. Removal of T_{inert_1} eliminates the possibility of removing T_{inert_3} since T_{inert_3} would no longer exist.

To facilitate the greedy algorithm, a single fault simulation pass is used in the original test set to identify each inert subsequence. If two inert subsequences begin at the same time frame, the longer subsequence is examined first for possible removal. This process is continued throughout the test set. In this example, if the subsequences T_{inert_1}, T_{inert_3}, T_{inert_4}, T_{inert_6}, and T_{inert_8} satisfy the removal criteria, after the greedy algorithm, T_{inert_1}, T_{inert_4}, and T_{inert_8} will be removed by the selection process.

Removal of recurrence subsequences is performed in a similar manner, except that the examination of removal eligibility for recurrence subsequences starts from the longest recurrence subsequence in the test set.

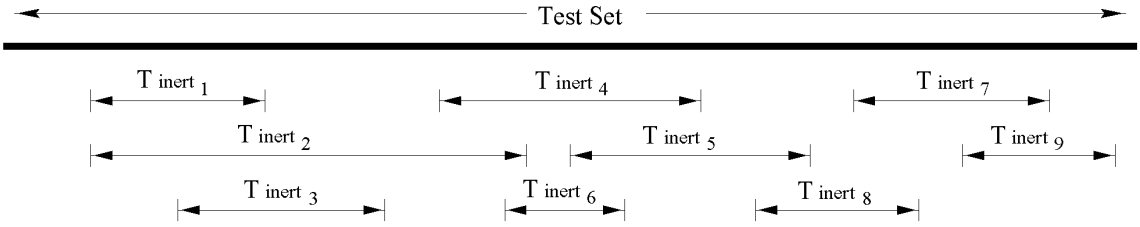

Fig. 7. Selection of susequences for removal.

TABLE 4
Compation Results for HITEC Test Sets

Ckt	Total	Original		ISR			RSR			CSR		
	Faults	FC	Vec	FC	% R	Time	FC	% R	Time	FC	% R	Time
s298	308	86.0	292	86.0	**11.0**	0.1	86.0	7.19	0.6	86.0	**11.0**	0.7
s344	342	95.9	127	95.9	3.94	0.2	95.9	**9.45**	0.3	95.9	4.72	0.3
s382	399	78.2	2074	78.2	3.09	0.8	78.2	**61.9**	2.7	78.2	61.5	7.4
s400	426	82.6	2214	82.6	2.80	0.9	82.6	50.2	7.4	82.6	**52.9**	8.3
s444	474	82.1	2240	82.1	4.15	1.3	82.1	**55.3**	9.8	82.1	55.2	10.0
s526	555	65.1	2258	65.1	**20.4**	1.9	65.1	10.1	9.7	65.1	**20.4**	8.8
s641	467	86.5	209	86.5	1.91	0.2	86.5	**27.3**	0.7	86.5	**27.3**	0.9
s713	581	81.9	173	81.9	1.16	0.2	81.9	**17.9**	0.7	81.9	**17.9**	0.8
s820	850	95.7	1114	95.5	24.4	0.8	95.5	45.6	3.6	95.5	**45.9**	3.8
s832	870	93.9	1136	94.0	23.7	0.8	94.0	46.6	3.9	94.0	**46.8**	3.9
s1196	1242	99.8	435	99.8	0.00	0.7	99.8	**1.15**	2.3	99.8	**1.15**	2.6
s1238	1355	94.7	475	94.7	0.00	0.8	94.7	**2.32**	2.5	94.7	**2.32**	0.8
s1423	1515	47.7	150	-	-	-	-	-	-	-	-	-
s1488	1486	97.2	1170	97.0	7.95	1.4	97.0	**34.0**	17.3	97.0	7.95	1.4
s1494	1506	96.5	1245	96.3	8.67	1.4	96.3	**40.5**	15.9	96.3	8.67	1.4
s5378	4603	70.2	912	-	-	-	-	-	-	-	-	-
s35932	39094	89.3	496	89.3	4.44	28.8	89.3	**49.8**	4165	89.3	4.44	4318
am2910	2391	91.6	1973	91.6	0.05	3.5	91.6	**0.15**	84.1	91.6	0.05	3.5
mult16	1708	92.6	111	92.6	0.00	0.9	92.6	0.00	3.0	92.6	0.00	0.9
div16	2147	78.0	238	78.0	5.46	0.6	78.0	**7.98**	4.2	78.0	5.46	0.6
pcont2	11300	31.1	7	-	-	-	-	-	-	-	-	-
piir8o	19920	45.1	21	-	-	-	-	-	-	-	-	-

FC: Fault cover in % Vec: Test set length % R: Percentage of test set length reduced
*Time: Execution time in seconds on HP 9000 J200 Greatest reductions highlighted in **bold***

6 Experimental Results

The static test set compaction algorithms were implemented in C++; various test sets generated for ISCAS89 sequential benchmark circuits [6] and several synthesized circuits [10] were used to evaluate the effectiveness of the algorithms. No scan is inserted in any of the benchmark circuits. All compactions were evaluated on an HP 9000 J200 with 256 MB RAM. Test sets generated by four test generators were used for evaluating the effectiveness of the compaction algorithms: HITEC [4], [5], GATEST [8], [9], DIGATE [10], and STRATEGATE [11]. HITEC is a deterministic test generator for sequential circuits, while GATEST, DIGATE, and STRATEGATE are all based on genetic algorithms. Inert-subsequence, recurrence-subsequence, and combined inert/recurrence subsequence removal algorithms were applied to the test sets. Since inert and recurrence-subsequence removal algorithms compact the test sets from different perspectives (local versus global, as described in Section 4), the corresponding results will differ. It would be interesting, therefore, to put the two algorithms together. Inert-subsequence removal followed by recurrence-subsequence removal is the combined approach performed for all the test sets.

In the actual implementation, the tests for criteria 3, 4, and 5 are computationally costly to implement in their entirety. In particular, parts of criteria 3, 4, and 5 require checking for fault masking. In our implementation of these two criteria, it is assumed that fault masking will not occur, since it has an extremely low probability. As a result, some faults detected by the original test sets may not be detected by the compacted test sets. Compaction results are shown in Tables 4, 5, 6, and 7 for test sets generated by each of the four test generators. Results for inert, recurrence, and combined inert/recurrence subsequence removal (**ISR**, **RSR**, **CSR**, respectively) are shown in all tables. The total number of collapsed faults for each circuit is shown only in Table 4. Fault coverage is defined as the percentage of faults detected. The fault coverages and lengths of the original test sets are shown, followed by the fault coverages, test set lengths, percent reduction in test set length after compaction, and execution times of compaction for each of the four test generators. A dash (-) in an entry indicates that no state-recurrence subsequences are present within the original test sets; thus, the compaction algorithms will not be applicable.

The results for HITEC test sets are first discussed. Little or no reduction in test set size was achieved for am2910 and mult16 because very few state-recurrence subsequences exist in the original test sets. For s1423, s5378, pcont2, and

TABLE 5
Compaction Results for GATEST Test Sets

Ckt	Original		ISR			RSR			CSR		
	FC	Vec	FC	% R	Time	FC	% R	Time	FC	% R	Time
s298	86.0	143	86.0	**2.10**	0.1	86.0	**2.10**	0.3	86.0	**2.10**	0.3
s344	96.2	84	96.2	0.00	0.1	96.2	0.00	0.3	96.2	0.00	0.3
s382	87.5	322	87.5	0.00	0.2	87.5	0.00	1.5	87.5	0.00	1.7
s400	86.2	316	86.2	0.00	0.2	86.2	0.00	1.7	86.2	0.00	1.8
s444	86.1	246	86.1	0.00	0.2	86.1	0.00	1.5	86.1	0.00	1.6
s526	76.2	371	76.2	0.00	0.3	76.2	0.00	2.5	76.2	0.00	3.2
s641	86.5	116	86.5	**1.72**	0.1	86.5	0.86	0.6	86.5	**1.72**	0.7
s713	81.9	125	81.9	20.8	0.1	81.9	**22.4**	0.6	81.9	**22.4**	0.6
s820	54.5	168	54.3	22.0	0.2	54.3	**35.7**	0.6	54.3	26.8	0.7
s832	55.9	151	55.8	40.4	0.2	55.8	43.7	0.5	55.8	**47.7**	0.5
s1196	99.2	447	99.2	12.3	0.4	99.2	**13.2**	2.7	99.2	**13.2**	2.8
s1238	94.3	381	94.3	3.94	0.4	94.3	5.77	2.4	94.3	**6.04**	2.4
s1423	85.3	696	85.3	0.00	3.7	85.3	0.00	19.0	85.3	0.00	23.2
s1488	83.1	244	82.9	33.2	0.5	82.9	36.9	2.4	82.9	**38.1**	2.5
s1494	84.3	358	84.1	38.0	0.7	84.1	**56.7**	4.2	84.1	53.6	3.2
s5378	69.3	560	-	-	-	-	-	-	-	-	-
s35932	89.3	246	89.3	0.81	906	89.3	**6.10**	9513	89.3	**6.10**	9509
am2910	90.5	728	90.5	3.43	1.5	90.5	3.98	32.6	90.5	**4.12**	34.6
mult16	96.8	204	96.8	3.92	0.6	96.8	**4.41**	9.0	96.8	**4.41**	8.4
div16	79.5	704	79.5	27.1	1.0	79.5	29.4	25.8	79.5	**29.7**	13.9
pcont2	60.4	256	60.4	41.0	3.4	60.4	50.4	90.2	60.4	**52.0**	76.2
piir8o	75.3	530	75.3	30.9	13.6	75.3	**51.1**	1061	75.3	50.6	866

FC: Fault Coverage in % Vec: test set length % R: Percentage of test set length reduced
*Time: Execution time in seconds on HP 9000 J200 Greatest reductions highlighted in **bold***

piir8o, no state-recurrence subsequences are present and, therefore, the compaction algorithms are not applicable. For the remaining circuits, significant reductions in test set sizes were obtained. The reductions were greater for the recurrence-subsequence removal algorithm than for the inert-subsequence removal algorithm in most circuits. For s382, s400, s444, s1488, s1494, and s35932, compaction using inert-subsequence removal achieved only 3 percent to 9 percent reductions, while 30 percent to 60 percent reductions were obtained using the recurrence-subsequence removal algorithm. For s298 and s526, recurrence-subsequence removal obtained less reduction than inert-subsequence removal, with reductions dropping from 11 percent to 7 percent and 20 percent to 10 percent, respectively. The execution times for recurrence-subsequence removal are generally longer than those for inert-subsequence removal due to fault simulation without fault dropping. For most HITEC test sets, either inert or recurrence subsequence removal dominates the compaction and the combined approach usually results in a test set reduction equal to the greater of the two obtained individually. For some circuits, including s400, s820, and s832, the combined approach results in better compaction. For many circuits, the execution times for the combined approach are smaller when compared with the time required for recurrence-subsequence removal alone because inert-subsequence removal in the first stage of the combined approach reduces the original test set significantly before the more costly recurrence-subsequence removal algorithm is applied, thus saving much computation. For several circuits, recurrence-subsequence removal provides better results than the combined approach. This is due to the fact that after removal of some inert subsequences in the first stage, the recurrence-subsequence removal algorithm may become ineffective in the combined approach. In other words, the crucial vectors that can aid recurrence-subsequence removal are eliminated in the first stage. Notice that, in a few cases, a very slight drop or increase in fault coverages (one or two faults) result after compaction. This is due to fault masking after removal of the inert subsequences, which is assumed not to occur in our implementation of Criterion 3.

The compaction results for GATEST test sets shown in Table 5 display trends similar to the HITEC compaction results. However, GATEST was targeted at generating compact test sets and, because GATEST test sets are much more compact than HITEC test sets to begin with, less compaction is obtained. Again, either inert or recurrence-subsequence removal dominates the compaction for most test sets. The circuits for which the combined approach

TABLE 6
Compaction Results for DIGATE Test Sets

Ckt	Original		ISR			RSR			CSR		
	FC	Vec	FC	% R	Time	FC	% R	Time	FC	% R	Time
s298	85.7	201	85.7	**1.99**	0.2	85.7	1.49	0.6	85.7	**1.99**	0.6
s344	96.2	93	96.2	16.1	0.1	96.2	**18.3**	0.3	96.2	**18.3**	0.3
s382	91.0	582	91.0	0.00	0.3	91.0	0.00	2.9	91.0	0.00	5.2
s400	89.7	3370	89.7	40.9	1.2	89.7	**53.0**	21.2	89.7	**53.0**	12.5
s444	88.6	1394	88.6	58.1	0.5	88.6	**68.7**	9.5	88.6	**68.7**	4.0
s526	80.4	2868	80.0	31.8	1.8	80.0	40.6	7.3	80.0	**45.0**	7.1
s641	86.5	213	86.5	12.2	0.2	86.5	**31.5**	1.1	86.5	21.1	1.1
s713	81.9	166	81.9	7.83	0.2	81.9	15.1	1.1	81.9	**15.7**	1.2
s820	59.8	245	59.7	15.5	0.3	59.7	**33.1**	0.9	59.7	31.4	1.0
s832	57.9	183	57.8	17.5	0.2	57.8	36.1	0.8	57.8	**39.9**	0.9
s1196	99.5	469	99.5	2.13	0.6	99.5	**5.76**	2.9	99.5	**5.76**	3.2
s1238	94.5	605	94.5	9.92	0.8	94.5	7.27	4.2	94.5	**11.2**	3.9
s1423	92.0	4045	92.0	0.20	10.3	92.0	**6.58**	140	92.0	**6.58**	154
s1488	92.7	543	92.6	45.1	1.0	92.6	52.7	7.4	92.6	**54.7**	6.6
s1494	87.8	719	87.7	50.3	1.0	87.7	**66.8**	8.1	87.7	66.3	6.1
s5378	74.8	11648	74.8	0.00	31.0	74.8	0.00	1022	74.8	0.00	985
s35932	89.8	589	89.8	48.2	57.1	89.8	**64.3**	4824	89.8	51.1	4385
am2910	91.8	2195	91.8	9.02	3.1	91.8	**9.57**	125.9	91.8	**9.57**	124
mult16	97.4	916	97.4	**23.7**	1.5	97.4	22.5	32.3	97.4	**23.7**	27.9
div16	83.9	4482	83.9	25.3	4.9	83.9	26.1	212.0	83.9	**26.4**	99.6
pcont2	60.5	3353	60.5	96.0	15.4	60.5	89.3	2544	60.5	**96.9**	89.8
piir8o	75.6	370	75.5	6.22	14.1	75.5	**8.38**	1424	75.5	**8.38**	1426

FC: Fault Coverage in % Vec: test set length % R: Percentage of test set length reduced
*Time: Execution time in seconds on HP 9000 J200 Greatest reductions highlighted in **bold***

produces the greatest compaction are s832, s1238, s1488, am2910, div16, and pcont2.

DIGATE aims to produce test sets that achieve high fault coverages. The DIGATE test sets are thus longer than GATEST test sets but are comparable with HITEC test sets when similar fault coverages are obtained. Significant reductions in DIGATE test sets are obtained using all three algorithms for most circuits, as shown in Table 6. The reason for greater reductions in DIGATE test sets is the existence of more state-recurrence and inert subsequences within the original test sets, leaving more room in the original test set for compaction. The reductions in test set sizes after combined inert/recurrence subsequence removal are greater than either algorithm alone for many of the circuits. Fig. 8 illustrates the percentage reduction in DIGATE test set size by the three algorithms for 10 benchmark circuits. For s1238, inert-subsequence removal achieves more compaction than recurrence-subsequence removal; however, the combined inert/recurrence subsequence removal does the best. For s35932, compaction by recurrence-subsequence removal achieves the greatest reduction. For the remaining circuits, recurrence-subsequence removal is more effective than inert-subsequence removal, and the combined approach achieves even better results. It should be noted that the execution time for the combined approach is lower than the execution time for the recurrence-subsequence removal alone.

STRATEGATE obtains more compact test sets than DIGATE and HITEC for most circuits. The results are shown in Table 7. Significant reductions in test set sizes are obtained for most circuits, even when the original test set sizes are already quite compact to start with. One plausible reason for this effect could be that STRATEGATE relies on dynamic state traversal, which will inevitably introduce more recurrence subsequences than other test generators. More than 40 percent reductions were observed for many circuits.

When comparing the results of the four test sets, one has to be aware that the fault coverages obtained are quite different. For example, for the test sets of s526, the fault coverages obtained by HITEC, GATEST, DIGATE, and STRATEGATE are 65.1 percent, 76.2 percent, 80.4 percent, and 81.8 percent, respectively; therefore, the absolute numbers of original and compacted test vectors will differ, making it hard to compare. For the test sets for which comparable fault coverages are obtained, such as the test sets of s298, s344, s641, and s713, the compaction results depend on the nature of the original test sets (i.e., the length

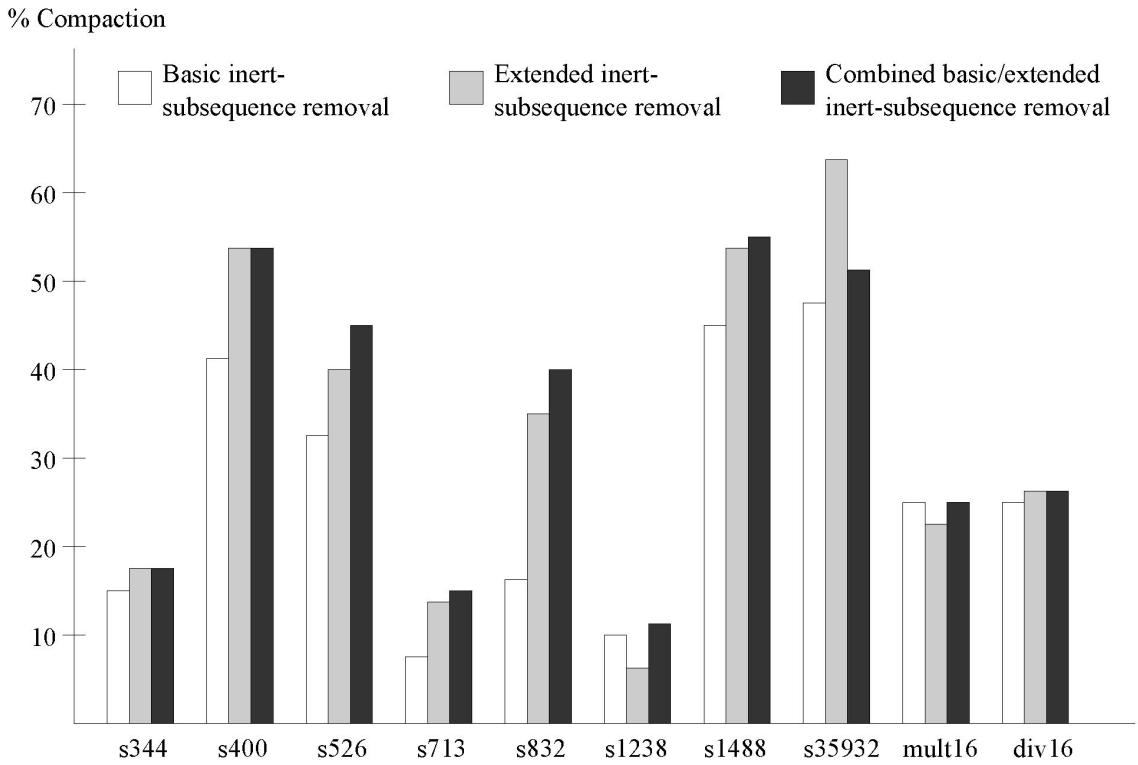

Fig. 8. Percentage reduction for DIGATE test sets.

of the original test sets and the number of states repeated, etc.). For instance, the HITEC test set for s713 was reduced from 173 vectors to 142 vectors (17.9 percent reduction), the GATEST test set was reduced from 125 vectors to 97 vectors (22.4 percent reduction), the DIGATE test set was reduced from 166 vectors to 140 vectors (15.7 percent reduction), and the STRATEGATE test set was reduced from 176 vectors to 141 vectors (19.9 percent reduction). This original test set dependency phenomenon was also observed in [3].

How do our compaction results compare with other static and dynamic compaction algorithms? Two issues need to be addressed here. First, the fault coverages obtained by static and dynamic compaction techniques are different and, second, the execution times needed are also different. Table 8 compares our results with the static compaction technique proposed in [3] and the dynamic compaction technique proposed in [12] for the HITEC test sets. Fault coverage, test set size, and time for compaction are shown for each technique after completion of the compaction. Execution times were not reported in [3]. Notice that the fault coverages resulting from compaction differ among the three techniques. The static compaction technique proposed in [3] produces very compact test sets for a few circuits, however, at the expense of long execution times performing multiple fault simulations; due to long execution times, results for large circuits were not reported. The execution times for our approach take a few seconds for the same circuits, although less compact test sets may result. When compared with the results obtained by dynamic compaction proposed in [12], the test sets obtained by [12] are more compact for all circuits. The execution times, on the other hand, are orders of magnitude higher for most circuits, although the compaction often results in an equivalent reduction in the time required for test generation. For example, in s526, the time taken by [12] is 35.3 seconds, while only 1.9 seconds are needed in our approach. Similar results were observed in the other circuits. However, in the circuit s35932, the dynamic compaction time is shorter. The extra computation costs in our approach are due to fault simulation without fault dropping. This cost can be reduced by dropping many faults early in the test set during RSR, thus making fault simulation without fault dropping less expensive. Finally, our algorithms often reduced test set sizes by more than 40 percent and they can be applied to functional test sets not generated by automatic test generators.

Our techniques can also be applied to partial-scanned circuits. Specifically, addition of scan will increase the effectiveness of the proposed techniques, since the state space for the scanned circuit is dramatically decreased due to fewer flip-flops and the likelihood of repeated states will then be increased.

7 Conclusions

Very fast static compaction algorithms have been presented. Our technique is deterministic, testing for specific conditions. Previously proposed trial and retrial-based approaches could take many hours for static compaction for even small test sets and small circuits; only a few seconds or minutes are needed by our compaction techniques and large test sets and circuits can be practically handled. Inert and recurrence subsequence removal techniques are based on the observation that test sets traverse through many

TABLE 7
Compaction Results for STRATEGATE Test Sets

Ckt	Original		ISR			RSR			CSR		
	FC	Vec	FC	% R	Time	FC	% R	Time	FC	% R	Time
s298	85.7	306	85.7	0.00	0.0	85.7	0.00	0.7	85.7	0.00	0.6
s344	96.2	86	96.2	1.2	0.5	96.2	**8.1**	0.5	96.2	**8.1**	0.6
s382	91.2	1486	91.2	55.3	3.5	91.2	61.4	2.5	91.2	**62.2**	4.1
s400	90.1	2424	90.1	49.4	5.2	90.1	**67.7**	8.3	90.1	63.7	7.1
s444	89.5	1945	89.5	49.0	4.5	89.5	59.4	7.1	89.5	**60.1**	5.9
s526	81.8	2642	81.8	30.4	6.0	81.8	35.8	9.2	81.8	**37.0**	8.9
s641	86.5	166	86.5	10.2	0.7	86.5	**19.3**	0.8	86.5	**19.3**	0.9
s713	81.9	176	81.9	15.3	0.7	81.9	**19.9**	0.8	81.9	18.2	0.9
s820	95.8	590	95.8	3.7	1.0	95.8	21.7	1.9	95.8	**23.2**	1.0
s832	94.0	701	94.0	10.6	1.8	94.0	29.0	2.4	94.0	**30.0**	2.9
s1196	99.8	574	99.8	2.61	1.5	99.5	**3.66**	2.0	99.5	**3.66**	2.6
s1238	94.6	625	94.5	**8.64**	0.7	94.5	7.68	0.9	94.5	**8.64**	1.0
s1423	93.3	3943	93.3	37.9	56.8	93.3	**39.6**	140	93.3	38.1	72.1
s1488	97.2	593	97.1	7.08	2.9	97.1	20.9	7.4	97.1	**24.1**	7.0
s1494	96.5	540	96.4	4.26	2.7	96.4	10.0	6.0	96.4	**13.3**	6.7
s5378	79.1	11,481	79.1	**9.09**	243	79.1	**9.09**	447	79.1	**9.09**	453
s35932	89.8	257	89.8	**15.6**	150	89.8	**15.6**	819	89.8	**15.6**	788
am2910	91.9	2509	91.9	12.4	25.7	91.9	13.0	125.9	91.9	**13.1**	44.4
mult16	97.5	1530	97.5	48.4	14.2	97.4	66.5	25.9	97.4	**68.1**	20.2
div16	84.7	3476	84.7	46.4	20.3	84.7	36.8	36.4	84.7	**46.9**	28.9
pcont2	60.5	194	60.5	21.6	99.4	60.5	39.2	123	60.5	**39.4**	122
piir8o	75.7	1398	75.7	76.5	131	75.7	78.3	499	75.7	**78.5**	332

FC: Fault Cover in % Vec: test set length % R: Percentage of test set length reduced
*T: Execution time in seconds on HP 9000 J200 Greatest reductions highlighted in **bold***

similar states, and compaction is based on eliminating candidate *state-recurrence* and *inert* subsequences inside the test sets. Significant reductions in test set sizes have been obtained for HITEC [4], [5], DIGATE [10], and STRATEGATE [11] test sets in small execution times; moderate reductions have been obtained for GATEST [8], [9] test sets, which are more compact than the HITEC and DIGATE test sets.

ACKNOWLEDGMENTS

This research was conducted at the University of Illinois and was supported in part by the Semiconductor Research Coporation under contract SRC 96-DP-109, in part by DARPA under contract DABT63-95-C-0069, and by Hewlett-Packard under an equipment grant.

REFERENCES

[1] T.M. Niermann, R.K. Roy, J.H. Patel, and J.A. Abraham, "Test Compaction for Sequential Circuits," *IEEE Trans. Computer-Aided Design*, vol. 11, no. 2, pp. 260-267, Feb. 1992.

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

[3] I. Pomeranz and S.M. Reddy, "On Static Compaction of Test Sequences for Synchronous Sequential Circuits," *Proc. Design Automation Conf.*, pp. 215-220, June 1996.

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

[5] T.M. Niermann and J.H. Patel, "Method for Automatically Generating Test Vectors for Digital Integrated Circuits," U.S. Patent No. 5,377,197, Dec. 1994.

[6] F. Brglez, D. Bryan, and K. Kozminski, "Combinational Profiles of Sequential Benchmark Circuits," *Proc. Int'l Symp. Circuits and Systems*, pp. 1,929-1,934, 1989

[7] T.H. Cormen, C.E. Leiserson, and R.L. Rivest, *Introduction to Algorithms*. Cambridge, Mass.: The MIT Press, 1990.

[8] E.M. Rudnick, J.H. Patel, G.S. Greenstein, and T.M. Niermann, "A Genetic Algorithm Framework for Test Generation," *IEEE Trans. Computer-Aided Design*, vol. 16, no. 9, pp. 1,034-1,044, Sept. 1997.

[9] E.M. Rudnick, "Simulation-Based Techniques for Sequential Circuit Testing," PhD dissertation, Dept. of Electrical and Computer Eng., Technical Report CRHC-94-14/UILU-ENG-94-2229, Univ. of Illinois, Aug. 1994.

[10] M.S. Hsiao, E.M. Rudnick, and J.H. Patel, "Automatic Test Generation Using Genetically-Engineered Distinguishing Sequences," *Proc. VLSI Test Symp.*, pp. 216-223, 1996.

[11] M.S. Hsiao, E.M. Rudnick, and J.H. Patel, "Sequential Circuit Test Generation Using Dynamic State Traversal," *Proc. European Design and Test Conf.*, pp. 22-28, 1997.

[12] E.M. Rudnick and J.H. Patel, "Simulation-Based Techniques for Dynamic Test Sequence Compaction," *Proc. Int'l Conf. Computer-Aided Design*, pp. 67-73, 1996.

TABLE 8
Comparing Various Compaction Techniques on HITEC Test Sets

Circuit	Dynamic compaction			Static compaction								
	[12]			Orig.	[3]				Our Approach			
	FC	Vec	Time	Vec	FC	Vec	% R	Time	FC	Vec	% R	Time
s298	86.0	145	28.2	292	86.0	87	**70.2**	-	86.0	260	11.0	0.1
s344	95.3	54	15.2	127	95.9	53	**58.3**	-	95.9	115	9.45	0.3
s382	81.0	272	99	2074	-	-	-	-	78.2	791	**61.9**	2.7
s400	85.4	389	141	2214	87.3	381	**82.8**	-	82.6	1042	**52.9**	8.3
s444	78.9	228	90.6	2240	-	-	-	-	82.1	1002	**55.3**	9.8
s526	49.5	76	35.3	2258	-	-	-	-	65.1	1797	20.4	1.9
s641	86.5	83	33.8	209	86.5	96	**54.1**	-	86.5	152	27.3	0.7
s713	81.9	60	27.5	173	-	-	-	-	81.9	142	17.9	0.7
s820	95.8	567	165	1114	95.7	424	**61.9**	-	95.5	603	**45.9**	3.8
s832	93.9	561	150	1136	-	-	-	-	94.0	604	**46.8**	3.9
s1196	99.8	232	111	435	-	-	-	-	99.8	430	1.15	2.3
s1238	94.7	248	126	475	94.7	247	**48.0**	-	94.7	464	2.32	0.8
s1488	97.2	475	346	1170	97.2	607	**48.1**	-	97.0	772	34.0	17.3
s1494	96.5	502	391	1245	-	-	-	-	96.3	741	**40.5**	15.9
s35932	89.7	160	2058	496	-	-	-	-	89.3	249	**49.8**	4165
div16	78.2	151	157	238	-	-	-	-	78.0	219	7.98	4.2

FC: *Fault coverage after compaction* *%* ***R:*** *Percent reduction from original test set size*
Time: *Execution time during compaction, measured in seconds on HP 9000 J200*
More than 40 percent reductions in test set sizes are highlighted in ***bold***

Michael S. Hsiao received the BS degree in computer engineering from the University of Illinois at Urbana-Champaign in 1992, and the MS and PhD degrees in electrical engineering in 1993 and 1997, respectively, from the same university. During the academic year 1992, he was a teaching assistant in the Department of Electrical and Computer Engineering at the University of Illinois. Between 1993 and 1997, he was a research assistant at the Center for Reliable and High-Performance Computing, University of Illinois. He was a visiting scientist at NEC USA in Princeton, New Jersey, during the summer of 1997. He is currently with Rutgers, the State University of New Jersey, where he is an assistant professor in the Department of Electrical and Computer Engineering. His current research focuses on VLSI testing, design for testability, design verification and diagnosis, power estimation, low power design, and computer architecture. He is a member of the IEEE.

Elizabeth M. Rudnick received the BS degree in chemical engineering and the MS and PhD degrees in electrical engineering from the University of Illinois at Urbana-Champaign in 1983, 1990, and 1994, respectively. She has worked at Motorola, Sunrise Test Systems, and Advanced Micro Devices in the areas of design verification, test generation, electronic design automation, and yield enhancement. She is currently an assistant professor in the Department of Electrical and Computer Engineering and a research assistant professor in the Coordinated Science Laboratory at the University of Illinois. Her research interests include test generation, defect diagnosis, design verification, and design for testability. She is a member of the IEEE.

Janak H. Patel received the BSc degree in physics from Gujarat University, India. He also received the BTech degree from the Indian Institute of Technology, Madras, India, and MS and PhD degrees from Stanford University, Stanford, California, all in electrical engineering. He is with the University of Illinois at Urbana-Champaign, where he is currently a co-director of the Center for Reliable and High-Performance Computing and a professor of electrical and computer engineering and computer science and a research professor with the Coordinated Science Laboratory. He is a co-founder of Sunrise Test Systems, an ATG and testability software company. He has also provided consulting and tutorial services to the industry on VLSI design and test. He is currently engaged in teaching, research, and consulting in the areas of automatic test generation, design for testability, fault simulation, and diagnosis. He is a fellow of the IEEE.