IEEE VLSI Test Symposium 1997, pp. 188-195

Fast Algorithms For Static Compaction of Sequential Circuit Test Vectors

Michael S. Hsiao, Elizabeth M. Rudnick, and Janak H. Patel
Center for Reliable and High-Performance Computing
University of Illinois, Urbana, IL 61801

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 re-visited throughout the application of a test set. Subsequences that start and end on the same states may be removed if necessary and sufficient conditions are met for them. The techniques 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.

I Introduction

Test sequence compaction produces test sequences of reduced lengths, which can greatly reduce the test application time. Test application time is important because it directly impacts the cost of testing. 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 post-processing 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[1-3]. 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 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 because of the large number of fault simulations required.

Our approach to test compaction is based on the observation that test sequences traverse through a small set of states, and some states are frequently re-visited. Table 1 shows the number of vectors and states traversed by the HITEC [4, 5] test sets for some ISCAS89 [6] benchmark circuits. It is clear that many subsequences that start and end on the same states exist within most 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 the test set if necessary and sufficient conditions are met. For circuits that have few or no repeated states, such as s1423 and s5378, 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.

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

The remainder of the paper is organized as follows. Section II gives some definitions of terms for this work. Sections III and IV describe two compaction algorithms in detail. The compaction framework for a given test set is explained in Section V, experimental results are given in Section VI, and Section VII concludes the paper.

*This research was supported in part by the Semiconductor Research Corporation under contract SRC 96-DP-109, in part by DARPA under contract DABT63-95-C-0069, and by Hewlett-Packard under an equipment grant.

II Definitions

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

Definition 1: A *propagation subsequence* T^f_{prop} for a particular fault f is a subsequence $T[v_i, v_{i+1}, ..., v_j]$ 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 $T[v_i, v_{i+1}, ..., v_{j-1}, v_j]$ such that f is activated in time-frame i, $[v_{i+1}, ..., v_{j-1}]$ 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 Figure 1. In order to simplify the compaction algorithms and avoid backtracing of fault-effects through the circuit, a continuous sequence of time-frames in which fault-effects are propagated to the flip-flops is taken as the propagation subsequence, even though this is a pessimistic measure.

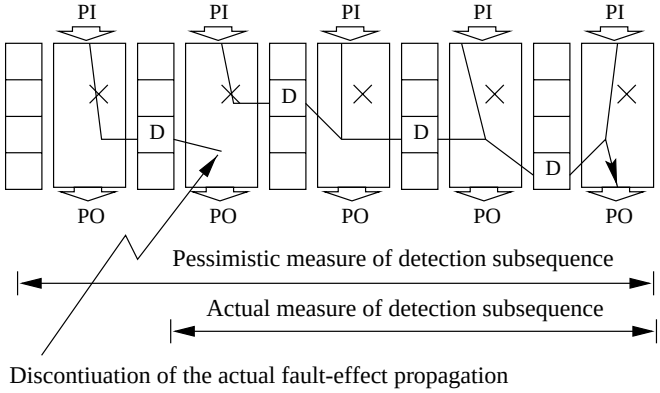


Figure 1: Pessimistic Measure of Detection Subsequence.

Definition 3: A *state-recurrence subsequence* T_{rec} is a subsequence of vectors $T[v_i, v_{i+1}, ..., v_j]$ 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 $T_{rec}[v_i, v_{i+1}, ..., v_j]$ 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. If S_f equals S'_f, the two error vectors cover one another.

III Inert Subsequence Removal

Many inert subsequences may exist within the 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 faults f_4 and f_8 (detected by vector v_7), this subsequence $T_{inert}[v_4 \ ... \ v_6]$ may be removed from the test set without affecting the fault coverage. These necessary conditions will be discussed later in this section, but intuitively, the conditions check if the detection of faults f_4 and f_8 depend on the inert 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

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, 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. 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 $T_{inert}[v_i, ..., v_j]$, if faulty state S^{i-1}_f at the end of time-frame $i-1$ and faulty state S^j_f 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 $T_{inert}[v_i, ..., v_j]$, 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.*

Detection subsequences for the additional fault-effects at the end of each inert subsequence are used to determine whether these fault-effects lead to a detection. When they do not lead to a detection, 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. Thus, an inert subsequence is not removed if the additional fault-effects at time-frame j are within their detection subsequences. Because of the pessimistic lengths of the detection subsequences, some inert subsequences that could have been removed may remain in the test set, resulting in a less compact test set.

Criterion 3: *For an inert subsequence $T_{inert}[v_i, ..., v_j]$, 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-frame $j+1$.*

Because the additional fault-effects at the beginning of time-frame $j+1$ 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 $j+1$. Since PI_{j+1} may very well differ from PI_i, the differences in the primary inputs may cause fault masking. As a result, multiple fault-effects may cancel each other.

Criterion 4: *For an inert subsequence $T_{inert}[v_i, ..., v_j]$, if neither error vectors E_f^{i-1} nor E_f^j covers 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 detects all of the inert subsequences and checks for any match with the removal criteria. The pseudo-code for this algorithm is described below:

```
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
```

The algorithm described above is a fast inert-subsequence removal algorithm because only the inert subsequences are considered for compaction. There may exist many state-recurrence subsequences, however, that can become inert if the faults detected within the state-recurrence subsequences can be detected outside the subsequence. The next algorithm removes state-recurrence subsequences that are not known to be inert using fault simulation with fault dropping.

IV 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.

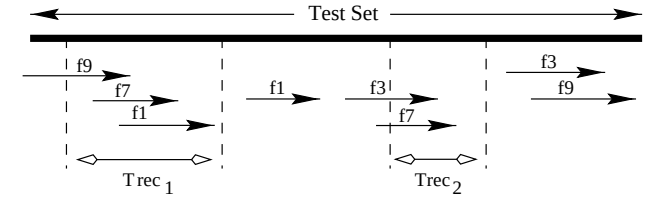
In terms of fault detection properties, typically an easy fault in the circuit is detected multiple times by the test set. This is because an easy fault requires only a few constraints on the primary inputs and flip-flop state in order for detection. This observation, together with the fact that many state-recurrence subsequences reside within the test set, allow the possibility of reducing the test set size even further 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 $T_{rec}[v_4 \ ... \ v_6]$. Some faults are detected within each subsequence. However, the three faults that are detected by the state-recurrence subsequence $T_{rec}[v_4 \ ... \ v_6]$, 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 $T_{rec}[v_4 \ ... \ v_6]$, and if the inert subsequence

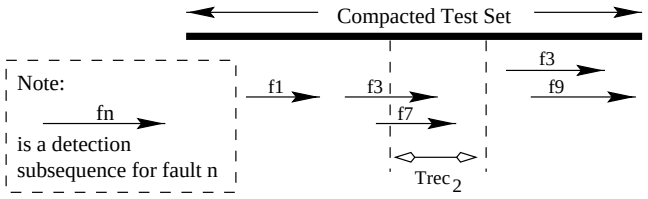
Table 3: Test Seq. With State-Recurrence Subseq.

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

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



(a) Sequence before removal of any state-recurrence subsequence



(b) Sequence after removal of first state-recurrence subsequence

Figure 2: Recurrence Subsequence Removal.

removal criteria described in the previous section are met at the boundaries, the state-recurrence subsequence may be safely removed from the test set.

Some boundary conditions apply to the removal of state-recurrence subsequences as well. Removal of a state-recurrence subsequence is illustrated in Figure 2. In part (a) of the figure, 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 be safely removed from the test set if the fault masking criterion at the boundary of T_{rec_1} for f_9 described in the previous section is 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 part (b) of the figure. Now, if T_{rec_2} were to be removed as well, both f_3 and f_7 need to have detection subsequences outside of T_{rec_2} as well. However, in this case, f_3 is the only fault within T_{rec_2} which has a detection subsequences outside of T_{rec_2} (i.e., the detection subsequence of f_7 originally in T_{rec_1} has already been removed), so T_{rec_2} may not be removed from the test set.

Again, a *single* pass of fault simulation is necessary to carry out the algorithm. In order to save storage cost, two passes are used instead. The pseudo-code for recurrence-subsequence removal is shown below:

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*

Compaction using the inert-subsequence removal of Section III requires inert subsequences to be present in the test set for the algorithm to be effective. Recurrence-subsequence removal, however, will attempt 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 somewhere outside. Other differences exist between the two algorithms. Inert-subsequence removal compacts the test set in a **bottom-up** fashion: the compaction proceeds from the *local inert* subsequences. Recurrence-subsequence removal, on the other hand, compacts the test set in a **top-down** manner: the compaction starts with the *globally* longest *state-recurrence* subsequence.

V 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. Consider the test set shown in Figure 3, with the inert subsequences illustrated. Determining the optimal selection of subsequences to remove is non-trivial; 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].

Each inert sequence T_{inert_i} 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 state but end at different states. 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 Figure 3, 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.

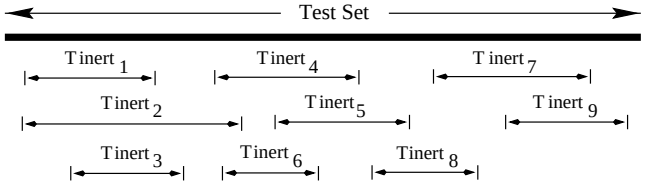


Figure 3: Selection of Subsequences for Removal.

VI Experimental Results

The static test set compaction algorithms were implemented in C++; various test sets generated for both ISCAS89 sequential benchmark circuits [6] and several

Table 4: Compaction Results for HITEC Test Sets

Ckt	Flts	Original		ISR				RSR				CSR			
		FC	Vec	FC	Vec	% R	T	FC	Vec	% R	T	FC	Vec	% R	T
s298	308	86.0	292	86.0	260	**11.0**	0.1	86.0	271	7.19	0.6	86.0	260	**11.0**	0.7
s344	342	95.9	127	95.9	122	3.94	0.2	95.9	115	**9.45**	0.3	95.9	121	4.72	0.3
s382	399	78.2	2074	78.2	2010	3.09	0.8	78.2	791	**61.9**	2.7	78.2	798	61.5	7.4
s400	426	82.6	2214	82.6	2152	2.80	0.9	82.6	1102	50.2	7.4	82.6	1042	**52.9**	8.3
s444	474	82.1	2240	82.1	2147	4.15	1.3	82.1	1002	**55.3**	9.8	82.1	1003	55.2	10.0
s526	555	65.1	2258	65.1	1797	**20.4**	1.9	65.1	2030	10.1	9.7	65.1	1797	**20.4**	8.8
s641	467	86.5	209	86.5	205	1.91	0.2	86.5	152	**27.3**	0.7	86.5	152	**27.3**	0.9
s713	581	81.9	173	81.9	171	1.16	0.2	81.9	142	**17.9**	0.7	81.9	142	**17.9**	0.8
s820	850	95.7	1114	95.5	842	24.4	0.8	95.5	606	45.6	3.6	95.5	603	**45.9**	3.8
s832	870	93.9	1136	94.0	867	23.7	0.8	94.0	607	46.6	3.9	94.0	604	**46.8**	3.9
s1196	1242	99.8	435	99.8	435	0.00	0.7	99.8	430	**1.15**	2.3	99.8	430	**1.15**	2.6
s1238	1355	94.7	475	94.7	475	0.00	0.8	94.7	464	**2.32**	2.5	94.7	464	**2.32**	0.8
s1423	1515	47.7	150	-	-	-	-	-	-	-	-	-	-	-	-
s1488	1486	97.2	1170	97.0	1077	7.95	1.4	97.0	772	**34.0**	17.3	97.0	1077	7.95	1.4
s1494	1506	96.5	1245	96.3	1137	8.67	1.4	96.3	741	**40.5**	15.9	96.3	1137	8.67	1.4
s5378	4603	70.2	912	-	-	-	-	-	-	-	-	-	-	-	-
s35932	39094	89.3	496	89.3	474	4.44	28.8	89.3	249	**49.8**	4165	89.3	474	4.44	4318
am2910	2391	91.6	1973	91.6	1972	0.05	3.5	91.6	1970	**0.15**	84.1	91.6	1972	0.05	3.5
mult16	1708	92.6	111	92.6	111	0.00	0.9	92.6	111	0.00	3.0	92.6	111	0.00	0.9
div16	2147	78.0	238	78.0	225	5.46	0.6	78.0	219	**7.98**	4.2	78.0	225	5.46	0.6
pcont2	11300	31.1	7	-	-	-	-	-	-	-	-	-	-	-	-
piir8o	19920	45.1	21	-	-	-	-	-	-	-	-	-	-	-	-

FC: Fault coverage 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**

synthesized circuits [10] were used to evaluate the effectiveness of the algorithms. All compactions were evaluated on an HP 9000 J200 with 256 MB RAM. Test sets generated by three test generators were used for evaluating the effectiveness of the compaction algorithms: HITEC [4, 5], GATEST [8, 9], and DIGATE [10]. HITEC is a deterministic ATPG for sequential circuits, while GATEST and DIGATE are both 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 (bottom-up versus top-down as described in Section VI), 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 and 4 are computationally costly to implement in their entirety. In particular, part of criteria 3 and 4 require the checking for fault masking. In our implementation of these 2 criteria, it is assumed that fault masking will not occur since it has an extremely low probability. Compaction results are shown in Tables 4, 5, and 6 for HITEC, GATEST, and DIGATE test sets, respectively. 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 three 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 the remaining circuits, significant reductions in test set sizes were obtained. The reductions were greater for the recurrence-subsequence removal algorithm. For s382, s400, s444, s1488, s1494, and s35932, compaction using inert-subsequence removal achieved only 3% to 9% reductions, while 30% to 60% 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% to 7% and 20% to 10%, respectively. The execution times for recurrence-subsequence removal are generally longer than those for inert-subsequence removal due to fault simulation

Table 5: Compaction Results for GATEST Test Sets

Ckt	Original		ISR				RSR				CSR			
	FC	Vec	FC	Vec	% R	T	FC	Vec	% R	T	FC	Vec	% R	T
s298	86.0	143	86.0	140	**2.10**	0.1	86.0	143	0.00	0.3	86.0	140	**2.10**	0.3
s344	96.2	84	96.2	84	0.00	0.1	96.2	84	0.00	0.3	96.2	84	0.00	0.3
s382	87.5	322	87.5	322	0.00	0.2	87.5	322	0.00	1.5	87.5	322	0.00	1.7
s400	86.2	316	86.2	316	0.00	0.2	86.2	316	0.00	1.7	86.2	316	0.00	1.8
s444	86.1	246	86.1	246	0.00	0.2	86.1	246	0.00	1.5	86.1	246	0.00	1.6
s526	76.2	371	76.2	371	0.00	0.3	76.2	371	0.00	2.5	76.2	371	0.00	3.2
s641	86.5	116	86.5	114	**1.72**	0.1	86.5	115	0.86	0.6	86.5	114	**1.72**	0.7
s713	81.9	125	81.9	99	20.8	0.1	81.9	97	**22.4**	0.6	81.9	97	**22.4**	0.6
s820	54.5	168	54.3	131	22.0	0.2	54.3	108	**35.7**	0.6	54.3	123	26.8	0.7
s832	55.9	151	55.8	90	40.4	0.2	55.8	85	43.7	0.5	55.8	79	**47.7**	0.5
s1196	99.2	447	99.2	392	12.3	0.4	99.2	388	**13.2**	2.7	99.2	388	**13.2**	2.8
s1238	94.3	381	94.3	366	3.94	0.4	94.3	359	5.77	2.4	94.3	358	**6.04**	2.4
s1423	85.3	696	85.3	696	0.00	3.7	85.3	696	0.00	19.0	85.3	696	0.00	23.2
s1488	83.1	244	82.9	163	33.2	0.5	82.9	154	36.9	2.4	82.9	151	**38.1**	2.5
s1494	84.3	358	84.1	222	38.0	0.7	84.1	155	**56.7**	4.2	84.1	166	53.6	3.2
s5378	69.3	560	-	-	-	-	-	-	-	-	-	-	-	-
s35932	89.3	246	89.3	244	0.81	906	89.3	231	**6.10**	9513	89.3	231	**6.10**	9509
am2910	90.5	728	90.5	703	3.43	1.5	90.5	699	3.98	32.6	90.5	698	**4.12**	34.6
mult16	96.8	204	96.8	196	3.92	0.6	96.8	195	**4.41**	9.0	96.8	195	**4.41**	8.4
div16	79.5	704	79.5	513	27.1	1.0	79.5	497	29.4	25.8	79.5	495	**29.7**	13.9
pcont2	60.4	256	60.4	151	41.0	3.4	60.4	127	50.4	90.2	60.4	123	**52.0**	76.2
piir8o	75.3	530	75.3	366	30.9	13.6	75.3	259	**51.1**	1061	75.3	262	50.6	866

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 length equal to the smaller 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 of the combined approach, the recurrence-subsequence removal algorithm may become ineffective. 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 (1 or 2 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 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. However, since the fault coverages achieved by DIGATE are generally greater than those achieved by either HITEC or GATEST, the test sets are a little longer. For the DIGATE test sets, significant reductions 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. Figure 4 illustrates the percentage reduction in DIGATE test set size by the three algorithms for ten 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

Table 6: Compaction Results for DIGATE Test Sets

Ckt	Original		ISR				RSR				CSR			
	FC	Vec	FC	Vec	% R	T	FC	Vec	% R	T	FC	Vec	% R	T
s298	85.7	201	85.7	197	**1.99**	0.2	85.7	198	1.49	0.6	85.7	197	**1.99**	0.6
s344	96.2	93	96.2	78	16.1	0.1	96.2	76	**18.3**	0.3	96.2	76	**18.3**	0.3
s382	91.0	582	91.0	582	0.00	0.3	91.0	582	0.00	2.9	91.0	582	0.00	5.2
s400	89.7	3370	89.7	1990	40.9	1.2	89.7	1586	**53.0**	21.2	89.7	1586	**53.0**	12.5
s444	88.6	1394	88.6	584	58.1	0.5	88.6	436	**68.7**	9.5	88.6	436	**68.7**	4.0
s526	80.4	2868	80.0	1957	31.8	1.8	80.0	1705	40.6	7.3	80.0	1578	**45.0**	7.1
s641	86.5	213	86.5	187	12.2	0.2	86.5	146	**31.5**	1.1	86.5	168	21.1	1.1
s713	81.9	166	81.9	153	7.83	0.2	81.9	141	15.1	1.1	81.9	140	**15.7**	1.2
s820	59.8	245	59.7	207	15.5	0.3	59.7	164	**33.1**	0.9	59.7	168	31.4	1.0
s832	57.9	183	57.8	151	17.5	0.2	57.8	117	36.1	0.8	57.8	110	**39.9**	0.9
s1196	99.5	469	99.5	459	2.13	0.6	99.5	442	**5.76**	2.9	99.5	442	**5.76**	3.2
s1238	94.5	605	94.5	545	9.92	0.8	94.5	561	7.27	4.2	94.5	537	**11.2**	3.9
s1423	92.0	4045	92.0	4037	0.20	10.3	92.0	3779	**6.58**	140	92.0	3779	**6.58**	154
s1488	92.7	543	92.6	298	45.1	1.0	92.6	257	52.7	7.4	92.6	246	**54.7**	6.6
s1494	87.8	719	87.7	354	50.3	1.0	87.7	239	**66.8**	8.1	87.7	242	66.3	6.1
s5378	74.8	11648	74.8	11648	0.00	31.0	74.8	11648	0.00	1022	74.8	11648	0.00	985
s35932	89.8	589	89.8	305	48.2	57.1	89.8	210	**64.3**	4824	89.8	288	51.1	4385
am2910	91.8	2195	91.8	1997	9.02	3.1	91.8	1985	**9.57**	125.9	91.8	1985	**9.57**	124
mult16	97.4	916	97.4	701	**23.7**	1.5	97.4	710	22.5	32.3	97.4	701	**23.7**	27.9
div16	83.9	4482	83.9	3349	25.3	4.9	83.9	3314	26.1	212.0	83.9	3298	**26.4**	99.6
pcont2	60.5	3353	60.5	135	96.0	15.4	60.5	360	89.3	2544	60.5	104	**96.9**	89.8
piir8o	75.6	370	75.5	347	6.22	14.1	75.5	339	**8.38**	1424	75.5	339	**8.38**	1426

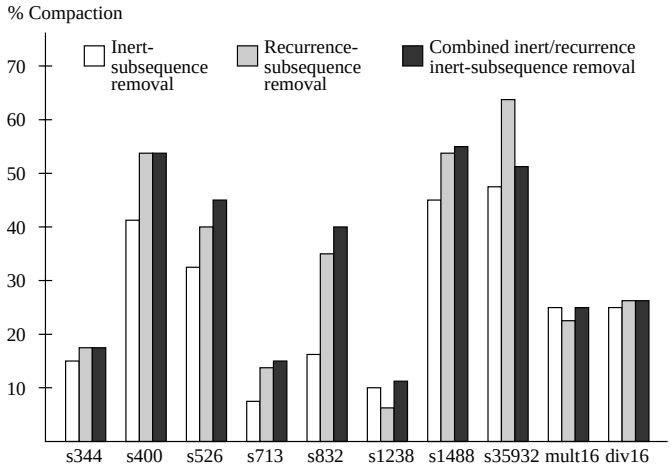


Figure 4: Results for DIGATE Test Sets.

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.

When comparing the results of the three 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, and DIGATE are 65.1%, 76.2%, and 80.4%, respectively; therefore, the absolute number 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. For instance, the HITEC test set for s713 was reduced from 173 vectors to 142 vectors (17.9% reduction), the GATEST test set was reduced from 125 vectors to 97 vectors (22.4% reduction), and the DIGATE test set was reduced from 166 vectors to 140 vectors (15.7% 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 are different, and second, the execution times needed are also different. Table 7 compares our results with the static compaction technique proposed in [3] and the dynamic compaction proposed in [11] 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; 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 result. When compared with the results obtained by dynamic compaction proposed in [11], the test sets obtained by [11] are more compact for all circuits. The execution times, on the other hand, are orders of magnitude higher for all

Table 7: Comparing Various Compaction Techniques on HITEC Test Sets

Circuit	Dynamic compaction			Static compaction								
	[11]			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% reductions in test set sizes are highlighted in **bold**

circuits except s35932, although the dynamic compaction often reduced the test generation time by an equivalent amount. Finally, our algorithms often reduced test set sizes by more than 40%.

VII Conclusions

Very fast static compaction algorithms have been presented. Our technique is deterministic, testing for specific conditions. Trial and re-trial 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 the proposed 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 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] and DIGATE [10] 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.

Acknowledgment

We would like to thank Prof. Irith Pomeranz of the University of Iowa for many helpful discussions.

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," Ph.D. Thesis, EE 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, December 1994.

[6] F. Brglez, D. Bryan, and K. Kozminski, "Combinational profiles of sequential benchmark circuits," *Int. Symposium on Circuits and Systems*, pp. 1929-1934, 1989.

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

[8] 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, 1994.

[9] E. M. Rudnick, "Simulation-based techniques for sequential circuit testing," Ph.D. dissertation, Dept. Electrical & Computer Engr, Tech. Report CRHC-94-14/UILU-ENG-94-2229, University 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] E. M. Rudnick and Janak H. Patel "Simulation-based techniques for dynamic test sequence compaction," *Proc. Intl. Conf. Computer-Aided Design,* pp. 67-73, 1996.