Simulation-Based Techniques for Dynamic Test Sequence Compaction

Elizabeth M. Rudnick **Janak H. Patel**
Center for Reliable & High-Performance Computing
University of Illinois, Urbana, IL

Abstract

Simulation-based techniques for dynamic compaction of test sequences are proposed. The first technique uses a fault simulator to remove test vectors from the partially-specified test sequence generated by a deterministic test generator if the vectors are not needed to detect the target fault, considering that the circuit state may be known. The second technique uses genetic algorithms to fill the unspecified bits in the partially-specified test sequence in order to increase the number of faults detected by the sequence. Significant reductions in test set sizes were observed for all benchmark circuits studied. Fault coverages improved for many of the circuits, and execution times often dropped as well, since fewer faults had to be targeted by the computation-intensive deterministic test generator.

1 Introduction

Deterministic test generators for single stuck-at faults in sequential circuits typically target individual faults, and once a test is generated, the test is fault simulated to find all other faults detected. The number of faults that must be specifically targeted by the test generator is thus reduced, but the tests do not necessarily detect a significant number of additional faults. Shorter tests sets are desirable in reducing test application time, which is an important consideration, since it directly impacts the testing cost. If shorter test sets can be used that still obtain a given fault coverage, more chips can be tested in a given time period, and fewer testers are needed. Furthermore, testers have a limited amount of memory for storing tests, so test sets should be smaller than the test vector limit to avoid an expensive memory reloading operation.

Heuristics have been proposed for static compaction of test sets through merging of test sequences, and large reductions in test set lengths were reported [1]. However, some faults incidentally detected by the original test sets were no longer detected by the compacted test sets, and additional passes of test generation and compaction were required to obtain the original fault coverage, increasing the overall execution time. Another static compaction approach was reported in [2] in which a given test set is modified through vector insertion, vector omission, and vector selection. Omission of unnecessary vectors was found to provide very compact test sets, but execution times were high due to the large number of fault simulations performed.

In contrast to static compaction, dynamic compaction is performed concurrently with the test generation process. Various different dynamic compaction approaches for sequential circuits have been proposed. In the first approach [3], test generation alternates between fault-independent and fault-oriented phases, and vectors are added to the test set one-at-a-time to minimize test set size. Very compact test sets were obtained for the few circuits reported, but execution times were not given. In the second approach [4, 5, 6], heuristics are used for selection of *secondary* target faults after a partially-specified test sequence is obtained for a *primary* target fault. Attempts are made to extend the test sequence generated for the primary fault to cover secondary faults using assignments to unspecified primary inputs (PIs) only. In the third approach [7], three heuristics were used to obtain compact test sets. Selecting the best of 64 random fillings for unspecified bits in a partially-specified sequence was found to reduce test set sizes significantly. In the fourth approach [8], the static compaction techniques proposed in [2] are used for dynamic compaction; test vectors are omitted from the existing test set or inserted into the existing test set so that the test set will be able to detect the current fault being targeted. Multiple passes are necessary, since faults previously covered by the test set may not be covered after the modifications. In the last approach [9], a symbolic algorithm is used to find all test sequences for each fault and to combine test sequences into a compact test set that covers the testable faults. This approach is suitable for control-dominant circuits that can be managed using binary decisions diagrams.

We propose a different approach to dynamic test sequence compaction that uses fault simulation and genetic algorithms (GAs) to reduce the number of test vectors in a generated test sequence and increase the number of faults detected. Some of the test vectors at the beginning and end of a partially-specified test sequence generated by a deterministic test generator may not be needed to detect a target fault if the circuit state is known. The test generator may assume that the circuit starts from an unknown state, but a fault simulator interfaced to the test generator can be used to remove any unnecessary vectors. Furthermore, each

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

ICCAD '96
1063-6757/96 $5.00 © 1996 IEEE

unspecified bit in a partially-specified test sequence is typically filled randomly with a one or zero. Instead, we propose to use a GA to find a better filling of unspecified bits that allows more faults to be detected. The genetic approach has the advantage of being a simulation-based approach in which processing occurs in the forward direction only. Thus, constraints on the test sequences generated are easily handled. The GA is able to make use of the previous good and faulty circuit states reached after all previous test vectors in the test set have been applied, often improving the fault coverage. Increasing the number of faults covered by a given test sequence reduces the number of faults that must be specifically targeted by the deterministic test generator. Therefore, reductions in the overall execution time can be expected as well. The approach is much simpler than many of the previous approaches, more scalable to larger circuits, and can easily be added to an existing deterministic test generator. This work is different from our previous work on test application time reduction for full scan and partial scan circuits in which test application time was reduced by limiting the scan operations, i.e., the number of test vectors for which flip-flop values must be scanned in and out [10]. The proposed technique is intended to be a simple addition to any existing deterministic test generator.

2 Genetic Algorithms

The simple GA, as described by Goldberg [11], contains a population of *strings*, or individuals. Each string is an encoding of a solution to the problem at hand and has an associated *fitness*, which depends on the application. The population is initialized with random strings, and the evolutionary processes of *selection*, *crossover*, and *mutation* are used to generate an entirely new population from the existing population. This process is repeated for several generations. To generate a new population from the existing one, two individuals are selected, with selection biased toward more highly fit individuals. The two individuals are crossed to create two entirely new individuals, and each character in a new string is mutated with some small mutation probability. The two new individuals are then placed in the new population, and this process continues until the new generation is entirely filled. In our work, we use tournament selection without replacement and uniform crossover. In *tournament selection without replacement*, two individuals are randomly chosen and removed from the population, and the best is selected; the two individuals are not replaced into the original population until all other individuals have also been removed. In *uniform crossover*, characters from the two parents are swapped with probability 1/2 at each string position in generating the two offspring. The goal of the evolutionary process is to improve the fitness of the best individual in each successive generation by combining the good portions of fit individuals from the preceding generation. However the best individual may appear in any generation, so we save the best individual found.

3 Overview

Deterministic sequential circuit test generators use backtracing operations to excite a fault, propagate its effects to a primary output, and justify the required state. In the process, values are assigned to a subset of the bit positions in the test vectors that make up the test sequence generated. The remaining bits are unspecified. Often several additional faults are detected by the same partially-specified sequence, but for other faults, particular values must be assigned to the unspecified bits if the faults are to be detected. In the HITEC sequential circuit test generator [12], the unspecified bits are filled randomly with ones and zeros in an attempt to cover many of these faults. However, more faults can often be detected if several alternative random fillings are fault simulated and only the best one is added to the test set. Even a greater number of faults can sometimes be detected if a GA is used to evolve the best filling of ones and zeros.

Genetic filling of unassigned values is illustrated in Figure 1. A partially-specified test sequence that detects the primary target fault is generated by a deterministic test generator. The test sequence is then sent to the dynamic compactor, which is a constrained GA-based test generator. The objective of the dynamic compactor is to find a filling of ones and zeros in the unspecified bit positions that detects the greatest number of secondary faults. All individuals in the GA population are initialized with random values, and unspecified values in the test sequence are filled using values in successive bit positions from an individual in the GA population. In the example shown in the figure, two unspecified bits in the first vector of the test sequence are filled using the first two values in the GA individual, one unspecified bit in the second vector is filled using the third value in the GA individual, and so on, until all unspecified bits are filled. The fully-specified test sequence is then fault simulated to obtain its fitness value; the fitness value measures the quality of the corresponding solution, primarily in terms of fault coverage. The same fitness evaluation procedure is used for all individuals in the population, and then a new generation is evolved. Processing continues for a set number

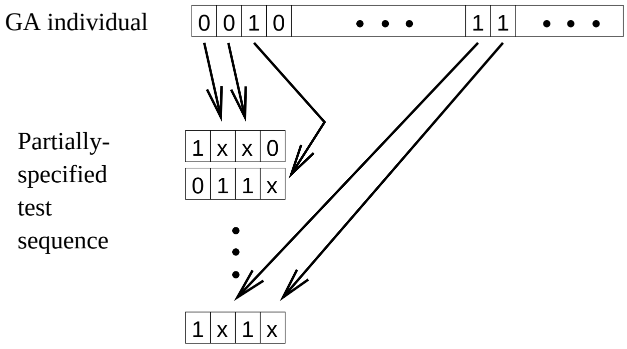

Figure 1: Genetic filling of unspecified bits.

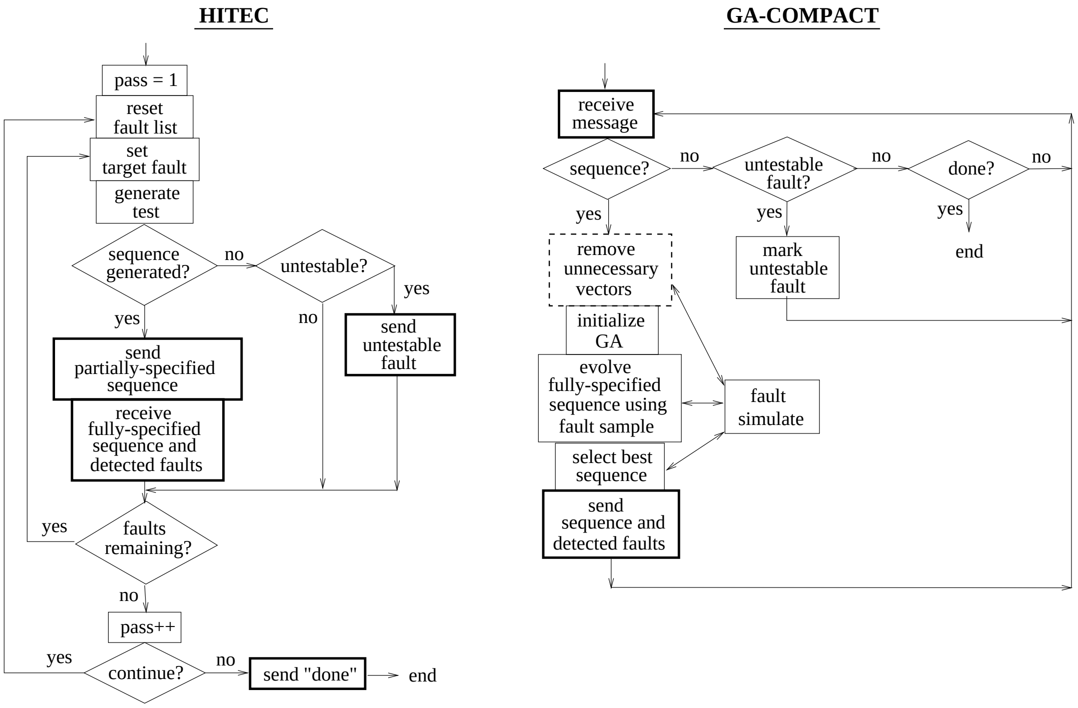

Figure 2: Test generation with dynamic compaction.

of generations, and the best test sequence found in any generation is added to the test set.

Before the GA is used, unnecessary vectors at the beginning and end of the partially-specified test sequence may be identified and removed using a fault simulator. These extra vectors are generated if the test generator assumes that the initial circuit state is unknown. However, the fault simulator may have information about the circuit state if any test vectors have been previously added to the test set. Thus, for a test sequence of length N, fault simulations are performed for the target fault in which the first k vectors are removed from the test sequence, where k ranges between 1 and N minus one. The shortest subsequence that detects the target fault is selected. Then any vectors at the end of the sequence that do not contribute to detecting the fault are removed. Note that this technique is only useful for test generators such as HITEC [12] which assume that the circuit starts in an unknown state. Test generators such as FASTEST [13] that make use of the circuit state would not benefit from this procedure, although they would benefit from the genetic filling of unspecified bits.

The overall algorithm for test generation with dynamic compaction is illustrated in Figure 2. The deterministic test generator, HITEC [12], and the dynamic compactor, GA-COMPACT, run independently, communicating through UNIX sockets. HITEC acts as a client, and GA-COMPACT acts as a server. HITEC makes several passes through the fault list, with increasing time limits per fault for successive passes. The next fault in the fault list is selected as the target fault, and test generation is attempted. If a test is successfully generated, it is sent to GA-COMPACT, which returns a fully-specified test sequence and a list of detected faults. The detected faults are removed from the fault list, and the process continues for the next fault in the fault list. If the targeted fault is identified as untestable, this information is sent to GA-COMPACT. After each pass through the fault list, the user is prompted about whether to continue with the next pass. Processing terminates when the user responds negatively.

GA-COMPACT receives messages from HITEC until it receives a message indicating that processing is finished. When a test sequence is received, unnecessary vectors at the beginning and end of the sequence are first removed in an optional step if desired. The GA is then initialized, and a fully-specified test sequence is evolved by the GA using a small fault sample. The PROOFS sequential circuit fault simulator [14] is used to identify the unnecessary vectors, to evaluate the fitness of each candidate test sequence, and again to update the state of the circuit after the best test sequence is selected. Finally the fully-specified test sequence is sent back to HITEC, along with the list of detected faults obtained through fault simulation using the full fault list. If the message sent by HITEC is an untestable fault identification, the corresponding fault is marked as untestable to avoid targeting it in future GA operations.

4 GAs for Dynamic Compaction

In applying GAs to dynamic compaction, we use each string in the GA population to represent a candidate filling of unspecified bits in a partially-specified test sequence. A binary coding is used, and binary values to be substituted for successive unspecified bits in the test sequence are placed in adjacent positions along the string. Strings are evolved over several generations, with the fitness of each individual being a measure of the number of faults detected by the corresponding test sequence. Thus, fault simulation is required. Since the original partially-specified test sequence detects the primary target fault, the fully-specified test sequence evolved is guaranteed to detect at least one fault.

The number of faults detected is the primary metric in the fitness function, since the objective of the GA is to maximize the number of faults detected by a given test sequence, To differentiate test sequences that detect the same number of faults, we include the number of fault effects propagated to flip-flops in the fitness function, since fault effects at the flip-flops may be propagated to the primary outputs in the next time frame. However, the number of fault effects propagated is offset by the number of faults simulated and the number of flip-flops to ensure that the number of faults detected is the dominant factor in the fitness function:

$$\begin{aligned} fitness \;=\; & \#\ faults\ detected \\ & + \frac{\#\ faults\ propagated\ to\ flip\ flops}{(\#\ faults\ simulated)(\#\ flip\ flops)} \end{aligned}$$

While an accurate fitness function is essential in achieving a good solution, the high computational cost of fault simulation may be prohibitive, especially for large circuits. To avoid excessive computations, we can approximate the fitness of a candidate test by using a small random sample of faults. In this work, we use a sample size of about 100 faults if the number of faults remaining in the fault list is greater than 100. Targeting untestable faults is a waste of time and can prevent the evolution of good test sequences in circuits having a large number of untestable faults. Therefore, the deterministic test generator sends information about untestable faults once they are identified. GA-COMPACT marks these faults as untestable and does not include them in the fault samples, but untestable faults are still included in the full fault list simulated after the fully-specified test sequence is evolved.

Several GA parameters are important in achieving good results. Small population sizes ranging between 8 and 64 are used, and the number of generations is limited to eight to minimize execution time. A binary coding is used in which each character in a string represents the value to be applied to a PI at a particular time frame. Thus, mutation is done by flipping a bit. Nonoverlapping generations and crossover and mutation probabilities of 1 and 1/64 are used.

5 Results

A simulation-based dynamic compactor, GA-COMPACT, was implemented using the existing PROOFS [14] source code and 1650 additional lines of C++ code. The sockets interface in the HITEC deterministic test generator [12] was also updated with 140 lines of C++ code for communicating with GA-COMPACT. Tests were generated for several of the ISCAS89 sequential benchmark circuits [15] and several synthesized circuits on an HP 9000 J200 with 256 MB memory. In the first two sets of experiments, results of which are shown in Tables 1 and 2, the effects of the genetic filling of unspecified bits in the partially-specified sequences were studied. Thus, unnecessary vectors at the beginnings and ends of the test sequences were not removed before the GA was invoked. Instead, the fully-specified test sequences evolved by the GA were fault simulated with the entire fault list, and noncontributing vectors at the ends of the sequences were removed before the sequences were sent back to the deterministic test generator. Results of GA-COMPACT using HITEC are shown in Table 1 for a GA having a population size of 8 and 8 generations (**HITEC + GA-COMPACT [8x8]**). Results for HITEC alone and for HITEC using the best filling from a random sample of 64 (**HITEC + RANDOM [64x1]**) are shown for comparison. GA-COMPACT was used to find the best of 64 random fillings by setting the population size to 64 and the number of generations to one. Three passes through the fault list were made by HITEC for all circuits except pcont2, for which only two passes were used to reduce the execution time. Time limits for the three passes were 0.5, 5, and 50 seconds per fault. For each circuit, the number of faults detected (**Det**), the number of test vectors generated (**Vec**), the execution time, and the number of untestable faults identified (**Unt**) using HITEC with and without GA-COMPACT are shown. Execution times for HITEC and for GA-COMPACT are separated in columns 4 and 12.

The combined approach, HITEC + GA-COMPACT, was successful in reducing the test set size for all circuits except s526, Am2910, and pcont2, but for these circuits the fault coverages increased. Fault coverages can drop if HITEC is unsuccessful in generating tests for hard-to-test faults, eg., s344 and mult16. However, in most cases, the fault coverage increases in the first two passes through the fault list, and a higher fault coverage is sometimes observed after the third pass, eg., s1423, s35932, div16, and pcont2. Comparing results for the genetic and random fillings of unspecified bits, the genetic approach outperforms the random approach for a majority of the circuits, even though the same number of fillings are simulated for each partially-specified test sequence. These results demonstrate the power of genetic algorithms. However, the random approach sometimes provides more compact test sets or higher fault coverages. Either approach is a significant improvement over the current practice of using a single random filling. When GA-COMPACT is used, more faults are typically covered by the test sequences generated by the computation-intensive deterministic test generator, and fewer faults have to be targeted. Consequently, the execution time for HITEC often drops when GA-COMPACT is used in conjunction. Because small fault samples are used by GA-COMPACT, execution times for the GA are minimal. Therefore, execution times for GA-COMPACT are usually significantly

Table 1: Genetic Compaction Results

	HITEC + GA-COMPACT [8x8]				**HITEC**				**HITEC + RANDOM [64x1]**			
Circuit	**Det**	**Vec**	**Time**	**Unt**	**Det**	**Vec**	**Time**	**Unt**	**Det**	**Vec**	**Time**	**Unt**
s298	265	207	16.5m+14.2s	26	265	322	16.2m	26	265	225	16.5m+12.7s	26
s344	321	80	10.7m+6.70s	11	324	115	8.07m	11	324	77	7.56m+6.16s	11
s349	332	**51**	5.37m+4.56s	13	332	128	7.73m	13	331	*56*	6.19m+4.95s	13
s382	301	*1193*	1.51h+1.95m	9	301	1463	1.52h	9	301	*1193*	1.51h+1.91m	9
s386	314	237	7.86s+9.06s	70	314	286	7.25s	70	314	**218**	6.57s+6.76s	70
s400	341	*1367*	1.20h+2.06m	17	341	1845	1.21h	17	341	1594	1.19h+2.00m	17
s444	373	1532	1.70h+2.23m	24	373	1761	1.49h	25	373	1733	1.49h+2.40m	25
s526	329	*678*	5.55h+51.9s	23	316	436	5.79h	23	337	**678**	5.44h+51.7s	23
s641	404	103	4.61s+11.4s	63	404	209	4.84s	63	404	116	4.70s+12.4s	63
s713	476	*86*	6.99s+9.99s	105	476	173	6.71s	105	476	109	7.15s+12.4s	105
s820	814	*842*	3.68m+1.29m	36	813	1115	3.50m	37	814	905	3.71m+1.36m	36
s832	818	860	5.33m+1.13m	51	817	1137	5.75m	53	818	*814*	5.39m+61.8s	51
s1196	1239	268	3.73s+34.2s	3	1239	435	5.50s	3	1239	280	3.82s+34.3s	3
s1238	1283	293	6.50s+36.9s	72	1283	475	8.23s	72	1283	475	6.73s+38.2s	72
s1423	931	140	12.1h+30.1s	14	723	150	13.9h	14	1023	**153**	11.4h+32.5s	14
s1488	1444	831	17.6m+2.79m	41	1444	1170	16.5m	41	1444	821	15.2m+2.54m	41
s1494	1453	812	10.2m+2.73m	52	1453	1245	9.59m	52	1453	845	9.07m+3.01m	52
s3271	3237	582	42.3m+2.66m	4	3227	641	39.4m	5	3236	542	36.5m+2.52m	4
s3330	2108	386	10.5h+1.67m	121	2097	551	10.6h	124	2110	483	10.5h+1.99m	121
s3384	3028	113	5.52h+48.6s	1	2996	161	6.06h	1	3026	101	5.54h+43.8s	1
s4863	4629	293	2.25h+2.27m	18	4621	477	2.38h	25	4629	302	2.28h+2.21m	19
s5378	3240	*568*	18.2h+3.54s	218	3231	912	18.4h	217	3238	589	18.3h+2.93m	224
s6669	6664	150	25.1m+1.81m	0	6655	319	33.3m	0	6670	183	19.3m+2.21m	0
s35932	34925	341	4.28h+20.3m	3984	34901	496	4.73h	3984	34910	402	4.42h+21.3m	3984
Am2910	2175	1006	58.8m+4.58m	173	2171	871	56.4m	173	2190	**594**	37.0m+2.69m	173
div16	1683	**183**	4.98h+43.0s	136	1667	228	5.35h	136	1679	*177*	5.04h+42.0s	136
mult16	1560	73	2.10h+25.3s	22	1582	**111**	1.90h	23	1571	*85*	1.91h+25.1s	23
pcont2	4343	*7*	8.22h+19.3s	2773	3514	7	9.58h	2773	4270	7	8.31h+20.3s	2773

8x8: GA population size = 8, number of generations = 8 64x1: GA population size = 64, number of generations = 1

Table 2: Genetic Compaction Results for Various GA Population Sizes

		HITEC + GA-COMPACT								
	Total	**Population Size = 16**			**Population Size = 32**			**Population Size = 64**		
Circuit	**Faults**	**Det**	**Vec**	**Time**	**Det**	**Vec**	**Time**	**Det**	**Vec**	**Time**
s298	308	265	*207*	16.5m+27.3s	265	**190**	16.1m+41.0s	265	243	16.4m+1.72m
s344	342	328	**52**	3.49m+9.64s	324	*56*	7.82m+17.5s	324	80	7.70m+44.6s
s349	350	332	73	5.41m+12.1s	332	*59*	5.40m+21.0s	332	67	5.37m+34.8s
s382	399	301	*1195*	1.51h+3.76m	301	**1192**	1.51h+7.51m	301	1577	1.49h+15.8m
s386	384	314	*226*	7.64s+16.5s	314	*228*	7.67s+32.5s	314	*227*	6.56s+52.1s
s400	426	341	**1366**	1.20h+4.00m	341	*1367*	1.20h+7.63m	341	**1366**	1.20h+15.8m
s444	474	373	*1370*	1.68h+4.28m	373	*1370*	1.68h+8.95m	373	**1367**	1.68h+15.6m
s526	555	316	434	5.76h+1.21m	323	678	5.64h+3.09m	323	434	5.66h+5.85m
s641	467	404	*87*	4.13s+19.1s	404	*88*	4.17s+38.0s	404	**81**	3.92s+1.13m
s713	581	476	**78**	6.85s+17.0s	476	*85*	6.92s+36.8s	476	**78**	6.89s+1.11m
s820	850	814	927	3.62m+2.60m	814	*857*	3.59m+5.09m	814	**820**	3.65m+9.13m
s832	870	817	*792*	6.60m+2.10m	818	*819*	5.15m+4.13m	818	**791**	5.94m+8.33m
s1196	1242	1239	*254*	3.43s+57.9s	1239	*245*	3.33s+1.80m	1239	**226**	3.21s+3.39m
s1238	1355	1283	*269*	6.08s+1.06m	1283	*262*	6.04s+2.00m	1283	**246**	6.02s+3.74m
s1423	1515	977	141	11.1h+58.4s	1018	*150*	10.2h+1.99m	900	125	11.9h+2.89m
s1488	1486	1444	804	17.6m+5.31m	1444	*788*	17.2m+9.76m	1444	**764**	17.2m+18.9m
s1494	1506	1453	845	10.0m+5.82m	1453	**746**	10.1m+9.55m	1453	802	10.1m+21.1m
s3271	3270	3244	617	34.5m+4.83m	3249	*562*	25.4m+10.0m	3251	**545**	23.9m+20.0m
s3330	2870	2114	*403*	10.3h+2.98m	2112	*396*	10.4h+6.16m	2115	**422**	10.4h+13.5m
s3384	3380	3045	97	5.24h+1.22m	3069	**111**	4.85h+3.14m	3043	99	5.26h+5.72m
s4863	4764	4634	*269*	2.06h+3.72m	4632	*263*	2.15h+7.29m	4636	**285**	2.04h+15.7m
s5378	4603	3239	*549*	18.2h+5.69m	3241	**583**	18.2h+11.0m	3236	*510*	18.3h+21.8m
s6669	6684	6672	**144**	20.2m+3.08m	6668	*146*	18.8m+6.53m	6669	*148*	18.3m+12.8m
s35932	39094	34923	386	4.12h+37.3m	34929	**328**	3.95h+1.43h	34925	*319*	4.11h+1.85h
Am2910	2391	2181	859	45.2m+6.74m	2183	852	52.4m+15.0m	2187	983	40.7m+31.2m
div16	2147	1681	*169*	5.01h+1.20m	1681	*151*	5.00h+2.31m	1672	149	5.28h+4.13m
mult16	1708	1567	63	1.98h+38.0s	1558	67	2.13h+1.57m	1565	77	2.00h+3.30m
pcont2	11300	4425	**7**	8.19h+25.1s	4178	7	8.59h+43.0s	4306	*7*	8.25h+59.3s

smaller than the HITEC test generation times, and the overall time for HITEC + GA-COMPACT is smaller than the execution time for HITEC alone for many of the circuits.

Genetic compaction results are shown in Table 2 for GA population sizes of 16, 32, and 64; the number of generations was set to 8 for all experiments. The total number of collapsed faults targeted is also given. The number of faults found to be untestable was about the same as that for the GA population size of 8 shown in Table 1. The most compact test set sizes are shown in bold in Tables 1 and 2 for experiments in which the fault coverage is maximal. Results that are nearly as good are italicized. A larger population can be expected to provide a better filling for a given partially-specified test sequence, which may lead to higher fault coverages and more compact test sets. For many of the circuits, the highest population size did in fact provide the most compact test sets at the maximum fault coverage achieved in any of the experiments. For many of the circuits, the smaller population sizes gave better results, and for four circuits – s386, s526, s1423, and Am2910 – the best of 64 random fillings provided the best results. The original HITEC algorithm with a single random filling gave the best results for one circuit only, mult16. These results highlight the fault-order dependency of an algorithm such as HITEC. Many of the faults are incidentally detected by the test sequences generated, and HITEC is not able to generate sequences for some of these faults. A population size of 32 gave good results for the largest number of circuits, but population sizes of 16 and 8 provided good results as well.

Experiments were carried out to determine if removing unnecessary vectors at the beginning and end of a partially-specified test sequence before the GA is invoked is effective in reducing the test set size. Results are shown in Table 3 for a GA having a population size of 32 and 8 generations. Test set sizes are shown in bold if they are smaller than those for any of the previous experiments and the fault coverage is at or near the maximum value obtained. Results are italicized if they are nearly as good as the best. Significant reductions in test set size occurred for many of the circuits, especially for the highly-sequential circuits such as s382. Test sequences of 60 vectors or more were often reduced to less than 5 vectors. In addition, fault coverages for four of the circuits were highest when this procedure was used: s400, s444, s35932, and mult16. For most of the remaining circuits, test set sizes were very close to the minimal obtained by GA-COMPACT.

Table 3: HITEC + GA-COMPACT with Removal of Unnecessary Vectors

Circuit	Det	Vec	Time	Unt
s298	265	**145**	16.3m+28.2s	26
s344	326	*54*	5.31m+15.2s	11
s349	332	*61*	5.53m+17.3s	13
s382	323	**272**	1.17h+1.65m	9
s386	314	**182**	7.25s+23.0s	70
s400	364	**389**	51.3m+2.35m	17
s444	374	**228**	1.59h+1.51m	24
s526	275	76	6.18h+35.3s	23
s641	404	*83*	4.18s+33.8s	63
s713	476	**60**	6.09s+27.5s	105
s820	814	**567**	2.75m+2.75m	36
s832	817	**561**	4.42m+2.50m	53
s1196	1239	*232*	4.62s+1.85m	3
s1238	1283	*248*	7.25s+2.10m	72
s1423	963	119	11.6h+1.64m	14
s1488	1444	**475**	14.3m+5.76m	41
s1494	1453	**502**	8.52m+6.51m	52
s3271	3242	593	35.4m+8.35m	5
s3330	2104	246	10.5h+4.08m	124
s3384	3047	*85*	5.21h+2.30m	1
s4863	4626	263	2.18h+7.48m	21
s5378	3239	**189**	18.2h+5.82m	223
s6669	6666	*129*	21.4m+5.60m	0
s35932	35051	**160**	2.04h+34.3m	3984
Am2910	2184	*531*	49.5m+9.96m	173
div16	1678	*151*	5.17h+2.61m	136
mult16	1594	**82**	1.77h+1.79m	22
pcont2	2847	3	10.5h+44.0s	2773

For a few circuits, including s526, s1423, and pcont2, the results are not as good as in some of the previous experiments, but this is due to a drop in fault coverage rather than a failure in compaction. HITEC is unable to generate tests for many of the faults in these circuits when they are specifically targeted. A simulation-based test generator such as GATEST [16] is more capable of generating tests for them. Thus, to handle these circuits, we added an extra phase at the end of each pass through the fault list in which completely-unspecified sequences are passed to GA-COMPACT, and sequences are generated until no more faults are detected. Unnecessary vectors at the ends of the sequences are removed using fault simulation with the full fault list before the fully-specified sequences are returned to HITEC. Test sequence lengths of x, $2x$, and $4x$ are used in the first, second, and third passes through the fault list, respectively, where x is equal to half the structural sequential depth of the circuit (or the next larger integer). Results are shown in Table 4. Again, a GA population size of 32 was used. Fault coverages increased to their highest values for four of the five circuits (shown in bold), and the fault coverage for the fifth circuit, s526 (shown italicized), was almost as high as the best of

Table 4: HITEC + GA-COMPACT with GATEST Phase

Circuit	Det	Vec	Time	Unt
s526	*325*	92	3.44h+47.5s	23
s1423	**1050**	201	7.85h+3.10m	14
s3271	**3259**	714	29.9m+11.9m	4
s6669	**6675**	197	10.2m+9.71m	0
pcont2	**6813**	71	4.76h+6.07m	2770

the previous experiments. Furthermore, test generation times dropped significantly, since faults were detected much earlier in the test generation process, and test sets remained reasonably compact, since the GATEST approach tends to produce compact test sets [16].

Test sets generated by HITEC/GA-COMPACT were more compact than those generated in [3] for two of five circuits, and for one of the two circuits, the HITEC/GA-COMPACT fault coverage was higher. Results of the approach described in [3] were not reported for the larger circuits. The HITEC/GA-COMPACT test sets were significantly smaller than any reported in [4, 5, 6] for comparable fault coverages. The best of 64 random fillings implemented using the FASTEST [13] test generator gave similar test set sizes for four circuits – s641, s713, s1196, and s1238 – although execution times were significant for the FASTEST/RANDOM combination since the full fault lists were used in evaluating the random fillings [7]. Finally, the HITEC/GA-COMPACT test sets were smaller than those reported in [8] for four of the six circuits having comparable fault coverages. Further compaction may be possible by performing the static compaction techniques of [2] in a postprocessing step.

6 Conclusions

High test application time can be a problem for sequential circuits, and therefore, compact test sets are desirable. However, the compaction techniques developed for combinational circuits are not directly applicable to sequential circuits. In our approach to dynamic compaction for sequential circuits, we use fault simulation and GAs to compact the sequences generated by a deterministic test generator and increase the number of faults detected by the sequences. Significant reductions in test set sizes were observed for all of the benchmark circuits studied, corresponding to reductions in test application time. Fault coverages sometimes improved, and execution times dropped for many of the circuits. Further improvements in fault coverage were obtained for a few of the circuits by adding an extra phase of simulation-based test generation after each pass through the fault list. Test sets generated using these techniques were about as compact as the most compact test sets reported, yet the execution times are considerably smaller. The proposed techniques can easily be added to existing commercial test generators with little or no detrimental effect on performance and are expected to provide a significant improvement in the quality of the test sets generated.

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] I. Pomeranz and S. M. Reddy, "On static compaction of test sequences for synchronous sequential circuits," *Proc. Design Automation Conf.*, pp. 215–220, 1996.

[3] I. Pomeranz and S. M. Reddy, "On generating compact test sequences for synchronous sequential circuits," *Proc. European Design Automation Conf. (EURO-DAC)*, pp. 105–110, 1995.

[4] S. T. Chakradhar and A. Raghunathan, "Bottleneck removal algorithm for dynamic compaction and test cycles reduction," *Proc. European Design Automation Conf. (EURO-DAC)*, 1995.

[5] A. Raghunathan and S. T. Chakradhar, "Acceleration techniques for dynamic vector compaction," *Proc. Int. Conf. Computer-Aided Design*, pp. 310–317, 1995.

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

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

[8] I. Pomeranz and S. M. Reddy, "Dynamic test compaction for synchronous sequential circuits using static compaction techniques," *Proc. Int. Symp. Fault-Tolerant Computing*, pp. 53–61, 1996.

[9] R. Bevacqua, L. Guerrazzi, F. Ferrandi, and F. Fummi, "Implicit test sequences compaction for decreasing test application cost," *Proc. Int. Conf. Computer Design*, 1996.

[10] E. M. Rudnick and J. H. Patel, "A genetic approach to test application time reduction for full scan and partial scan circuits," *Proc. Int. Conf. VLSI Design*, pp. 288–293, 1995.

[11] D. E. Goldberg, *Genetic Algorithms in Search, Optimization, and Machine Learning,* Reading, MA: Addison-Wesley, 1989.

[12] 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.

[13] T. P. Kelsey, K. K. Saluja, and S. Y. Lee, "An efficient algorithm for sequential circuit test generation," *IEEE Trans. Computers*, vol. 42, no. 11, pp. 1361–1371, Nov. 1993.

[14] T. M. Niermann, W. -T. Cheng, and J. H. Patel, "PROOFS: A fast, memory-efficient sequential circuit fault simulator," *IEEE Trans. Computer-Aided Design*, vol. 11, no. 2, pp. 198–207, Feb. 1992.

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

[16] 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.