Fast Static Compaction of Tests Composed of Independent Sequences: Basic Properties and Comparison of Methods

Jaan Raik, Artur Jutman, Raimund Ubar

Tallinn Technical University, Estonia

ABSTRACT

Current paper presents an overview of recently proposed fast methods for static compaction of sequential circuit tests divided into independent test sequences. The methods include genetic algorithm based, greedy and deterministic approaches. We explain the algorithms and discuss the underlying complexity issues. The different methods were tested on a large number of publicly available benchmark test sets in order to assess their efficiency. In addition to the comparison of approaches, our experiments reveal some remarkable properties of realistic sequential circuit test sets.

1. INTRODUCTION

Majority of the earlier works in the field of static compaction consider the case, where there is a single test sequence that we are trying to minimize by removing some patterns from it. This requires iterative fault simulation during the compaction process in order to check that the fault coverage has not decreased. Thus the run times are very long.

Faster approach has been proposed in [1, 2 and 3]. The technique in [2] requires keeping track of the internal state of the circuit. In [1] the whole test set is divided into independent test sequences separated by global reset and fault simulation is performed only once, prior to compaction. In addition, in [1] a set of benchmarks [4] consisting of 103 fault matrices of ISCAS'89 circuits tested by three different ATPG tools [5, 6, 7] were made publicly available. In [3] a method was proposed that is in average two orders of magnitude faster than the one presented in [1]. It is based on detecting the set of essential patterns and subsequent application of a greedy algorithm. The average compaction of this method was better than of any previously published method belonging to this particular class.

In this paper we consider the above-mentioned case of static compaction, where the test set is divided into test sequences. We compare the approaches proposed in [1] and in [3] to a new method based on branch-and-bound search. We explain the algorithms and discuss the corresponding complexity issues. In addition, we consider some relevant properties of realistic fault matrixes. These properties include fault matrix sizes before and after applying implications and the ratio of essential patterns in the test sets.

2. STATIC COMPACTION OF FAULT MATRICES

A *test set* T consists of *test sequences* $t_i \in T$, i = 1, ... , n. Each sequence t_i contains in turn L_i *test vectors*. We refer to L_i as the *test length* of sequence t_i. The set of faults f_j, j = 1, ..., m detected by T is denoted by F. Total test length of test set T can be viewed as a sum

$$\sum_{i=1}^{n} L_i$$

In the problem of static compaction of test sequences, our task is to find values for the L_i such that the above sum would be minimal while the number of faults that the test set T detects would still be |F|.

Consider the test set example shown in Fig. 1 that consists of three test sequences s_1, s_2 and s_3, respectively. Sequence s_1 consists of four test vectors covering fault f_2 at the third vector and f_1 at the fourth vector. Sequence s_2 consists of three test vectors covering f_1 at the first vector and f_3 at the third vector. Finally, sequence s_3 consists of four test vectors covering f_2 at the first vector, f_3 at the second vector and f_4 at the fourth vector.

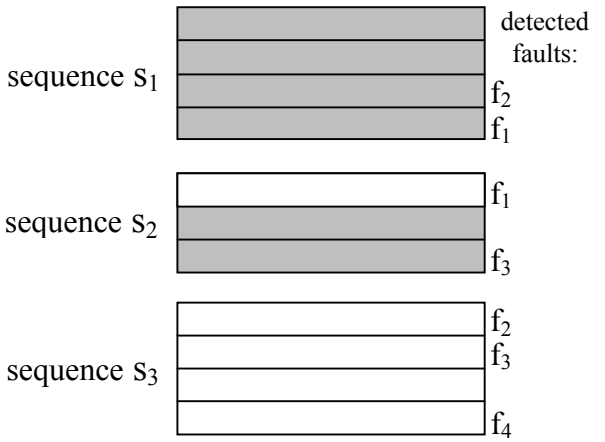

Fig.1. Test set compaction example

Initial test length of this test set is 11 vectors. It can be found that the optimal solution for the static compaction problem is selecting sequence s3 and the first vector from sequence s2. Hence, the length of the optimal compacted test set will be 5 vectors.

The fault matrix representation for the test set in Fig. 1 is shown below.

Table 1. Fault matrix for the test set of Fig. 1

	f_1	f_2	f_3	f_4
s_1	4	3	0	0
s_2	1	0	3	0
s_3	0	1	2	4

In this type of descriptions test set T consisting of n faults and m test sequences can be viewed as a matrix

$$T = \begin{pmatrix} t_{f_1,s_1} & t_{f_2,s_1} & \dots & t_{f_n,s_1} \\ t_{f_1,s_2} & t_{f_2,s_2} & \dots & t_{f_n,s_2} \\ \dots & \dots & \dots & \dots \\ t_{f_1,s_m} & t_{f_2,s_m} & \dots & t_{f_n,s_m} \end{pmatrix}$$

where $t_{si,fj}$ is equal to k if sequence s_i covers fault f_j at the k-th vector and zero if sequence s_i does not cover fault f_j.

If we select k vectors from sequence s_i then all the faults $\{f_j : k \geq t_{si,fj} > 0\}$ are said to be covered by these vectors. In our algorithm we remove the columns corresponding to the covered faults from matrix T. In addition, we must subtract k from all the non-zero elements $t_{si,fj}$ of the row corresponding to the sequence s_i. Our task is to cover all the faults (i.e. columns of matrix T) by selecting the minimal number of vectors. As it was shown in [1], this task belongs to the class of NP-complete problems.

3. COMPACTION ALGORITHMS

3.1. Genetic algorithm for static compaction

Genetic algorithms are derived from observations in nature, where generations of living beings are improved by evolutionary process. In this type of algorithms each solution (called individual) is first represented by *encoding*. A set of individuals is referred to as *population*. Some solutions are better than others. Fitness value is assigned to individuals according to a quality (*fitness*) *function*. Individuals with higher fitness are preferred to be chosen for reproduction. The individuals in a population can reproduce themselves by combining the *genes* of two selected parent individuals during *crossover*.

In [1] a straightforward encoding is adopted in order to represent the test sequence ordering: each individual is a string composed of n genes (n being the number of sequences), and directly corresponds to an ordering. As an example, let us consider the test set composed of three sequences introduced in Fig. 1. Possible individuals are sequence orderings (s_1 s_2 s_3) and (s_2 s_3 s_1).

Evaluation for every individual is done according to the number of vectors eliminated in respect to the original test set provided by the ATPG. Each individual is assigned a fitness value, which represents the position of the individual in the ranking based to their evaluation value. This reduces the risk for one individual with evaluation value much higher than others to prevail in every selection procedure.

In the method, uniform cross-over is adopted: one half of the genes (randomly chosen) of the child individual are taken from one parent, the others are taken from the other parent, in the same order they appear in them.

Exact parameters of the genetic algorithm for static compaction are presented in [1]. The main shortcoming of the above mentioned approach is that it always considers the entire search space, never taking advantage of any implications. For example, according to our experiments, essential patterns form a vast majority of the patterns in the final optimized test set. In the following subsection we consider these implications, which allow us to remarkably prune the search space.

3.2. Greedy optimization method

In the following two methods, three relationships are implemented to prune the search space for the compaction algorithm.

3.2.1. Selecting essential vectors

If fault f_j is detected by the k-th vector of test sequence s_i and is not detected by any other sequence belonging to the test set then k first vectors of sequence s_i are called essential. After selecting the essential vectors we remove them from the test sequences. In addition we remove the columns corresponding to faults covered by these vectors from matrix T. This simple pre-processing step allows to significantly reduce the search space for the static compaction algorithm.

3.2.2. Removing dominated faults

Column f_a is said to *be dominated by column* f_b and will be removed from matrix T if

$$\forall_{i=1}^{m} t_{s_i,f_b} \neq 0 \Rightarrow t_{s_i,f_a} \neq 0,\ t_{s_i,f_b} \geq t_{s_i,f_a}.$$

The motivation for removing dominated faults is the following. Let us assume that a fault f_b dominates fault f_a. The above relationship shows that no matter by which sequence selection we cover fault f_b we will also cover fault f_a. Since in order to achieve full matrix coverage we will have to cover f_b in any case, column f_a represents redundant information for the optimization problem.

3.2.3. Removing dominating sequences

A row corresponding to sequence s_b is said to be a *dominating sequence* of s_a and will be removed from matrix T if

$$\forall_{j=1}^{n} t_{s_b,f_j} \neq 0 \Rightarrow t_{s_a,f_j} \neq 0,\ t_{s_a,f_j} \leq t_{s_b,f_j}.$$

A dominating sequence can be removed because the sequence dominated by it covers all the same faults in a shorter or equal vector range.

The method presented in [3] applies above described implications as far as possible. Whenever it encounters a selection between alternative solutions, it switches to a greedy algorithm. The greedy selection function implemented in [3] is described in the following. Let us denote by *Minrange*(f_j) the minimal number of vectors that has to be selected from any test sequence in order to detect a fault f_j. Let *Maxrange* be the maximum *Minrange*(f_j) of all the faults.

The selection function selects *Maxrange* vectors from the corresponding test sequence. If there exist multiple maximal *Minrange*(f_j) values then the algorithm prefers the sequences that detect more faults in *Maxrange* first vectors.

3.3. Branch-and-bound search

In this paper we present a branch-and-bound algorithm for static compaction of independent test sequences. Unlike the above-described genetic algorithms and greedy optimization, branch-and-bound is capable of finding the globally optimal solution for this type of compaction problem. In fact, as our experiments show, this algorithm is capable of finding globally optimal results for all the 103 benchmark test sets of [4] in a relatively short time.

The algorithm uses depth-first approach for the decision tree traversal. While the breadth-first approach would limit the search space further it was not implemented due to its excessive memory requirements. (In the breadth-first search we would have to trace all the decisions in parallel, while in the depth-first approach only one decision is considered at a time). Differently from the greedy approach of [3], this algorithm does not stop after finding the first solution but it checks all the consistent solutions until the optimal is found.

The search problem for the branch-and-bound approach can be further simplified by discarding decision combinations equivalent to previously traversed ones, i.e. the ones which contain exactly the same set of decisions that were tried before but in a different order. Equivalent search state identification requires that the decisions are ordered (ranked) in the systematic search process. In current implementation the primary decision ordering criterion is the number of vectors to be selected, the secondary criterion is index of the test sequence. During decision making we never consider a decision whose rank is less than the one of the previously made decision in the decision tree. The general idea is not new and it has been successfully implemented in automatic test pattern generation.

If we do not consider equivalent search states the total number of decisions to be considered will be:

$n!(1/0! + 1/1! + 1/2! + \ldots + 1/(n-1)!)$, which approaches asymptotically $e \bullet n!$.

However, by identifying equivalent search states we have to consider 2^n-1 decisions in the worst case. While this is still a problem of high complexity, it is considerably less so than the traditional full search considered above.

4. EXPERIMENTAL RESULTS

In current experiments we used the test set benchmarks, which contain fault matrixes for test sets generated for ISCAS'89 benchmarks by three different ATPG tools: GATTO [5], HITEC [6] and SYMBAT [7]. The fault matrixes were obtained by fault simulating the test sets and are available at [4].

Table 3 presents the compaction results for the GATTO test sets only. The reason for that is the fact that the SYMBAT and HITEC test sets appeared to be very easy to compact and all the three methods yielded fairly identical results. The table presents the number of faults, the size of initial test set, test set size contributed by essential patterns, test set size after applying search state pruning (described in subsection 3.2) and comparative results for the three methods for each benchmark circuit, respectively. The overall best results are marked by bold numbers in the table. The experiments for [1] and [3] were run on SUN SPARC 5 workstations and experiments for the branch-and-bound algorithm were run on a 366 MHz SUN UltraSPARC 60 workstation with 512 MB RAM under Solaris 2.5.1 operating system. As it can be seen by comparing the run times of the greedy and branch-and-bound approaches, the performance difference between these two types of computers is roughly 5 times.

As Table 3 shows, the new technique allows the best compaction results being capable of finding globally optimal results for all the test sets in the given benchmark family. Furthermore, the compaction times are short. However, the fastest compaction is achieved by applying greedy optimization, which allows to reach optimal or nearly optimal results for the benchmark test sets. Table 2 presents important statistics about the efficiency of the three algorithms.

Table 2. Comparison of static compaction methods

	Genetic [1]	Greedy [3]	Branch-and-bound
average compaction	49.86 %	50.06 %	**50.27 %**
best compaction results	32	30	**40**
proved optimal results	N/A	30	**40**

In addition to the fact that the above experiments allow us to compare different methods for static compaction of test sets, they reveal some interesting properties of realistic fault matrixes. Probably the most significant of these are the ratio of essential patterns in the compacted test set and the ratio of patterns selected by search state pruning (see subsection 3.2) in the compacted result. As our experiments showed, in average 89 % (62 % - 100 %) of the patterns in the compacted test sets were essential. Furthermore, in average 99 % (87 % - 100 %) of the patterns in the compacted test sets were selected during search space pruning. This surprising result explains the relative efficiency of the simple greedy compaction method presented in [3] on given benchmark test sets.

5. CONCLUSIONS

In this paper, the case of static compaction, where the test set is divided into test sequences was considered. Three different recently proposed methods were compared, including genetic algorithm, greedy optimization and branch-and-bound based approaches. Experiments showed that the branch-and-bound technique allows the best compaction results being capable of finding globally optimal results for all the test sets in the given benchmark family. Furthermore, the compaction times were short, never exceeding 400 seconds on a 366 MHz SUN UltraSPARC server. However, the fastest compaction was achieved by applying greedy optimization, which allowed to reach optimal or nearly optimal results for the benchmark test sets. The relative efficiency of the greedy approach was mainly due to the fact that according to our study, vast majority of patterns in the compacted real world sequential circuit test sets were essential. In addition, the search space could be efficiently pruned by using dominance relationships of faults and sequences.

6. REFERENCES

[1] F. Corno et al. "New static compaction techniques of test sequences for sequential circuits". *ED&TC*, 1997, pp.37-43.

[2] M. S. Hiao et al., "Sequential circuit test generation using dynamic state traversal", *Proc. ED&TC*, pp. 22-28, 1997.

[3] J. Raik et al., "Fast static compaction of test sequences using implications and greedy search", *Proc. ETW 2001*, pp. 207-209.

[4] URL: http://www.cad.polito.it

[5] F. Corno et al., "GATTO: A genetic algorithm for automatic test pattern generation … ", *IEEE Trans. CAD*, Aug. 1996.

[6] T.M. Niermann, J.H. Patel, "HITEC: A test generation package for sequential circuits", *Proc. of EDAC*, 1991.

[7] G. Cabodi, F. Corno, P. Prinetto, M. Sonza Reorda, "Symbat's user guide", Politecnico di Torino, Sept. 1993.

Table 3. Static compaction results for the GATTO test set

circuit	# faults	Initial test set		Essential test		After pruning	GA result* [1]		greedy result* [3]		Branch-and-bound**	
		# seq.	# vec.	# seq.	# vec.	# vectors	# vec.	time, s	# vec.	time, s	# vec.	time, s
s208	216	36	1096	4	290	347	**347**	8,0	**347**	0,08	**347**	0,01
s298	309	24	302	10	130	141	**141**	8,0	**141**	0,09	**141**	0,01
s344	322	19	141	6	49	66	**66**	7,0	**66**	0,08	**66**	0,01
s349	330	19	144	9	75	84	**84**	7,0	**84**	0,09	**84**	0,01
s382	399	17	840	7	485	485	**485**	8,0	**485**	0,08	**485**	0,01
s386	373	38	418	11	199	221	**221**	17,0	**221**	0,11	**221**	0,03
s400	425	16	916	6	443	502	**502**	9,0	**502**	0,06	**502**	0,01
s420	453	33	797	5	325	333	**333**	21,0	**333**	0,10	**333**	0,01
s444	475	22	1434	9	788	788	**788**	12,0	**788**	0,11	**788**	0,02
s499	538	29	465	8	182	192	**192**	19,0	**192**	0,19	**192**	0,04
s510	550	37	989	4	146	237	**237**	18,0	263	0,32	**237**	0,05
s526	556	18	1050	8	696	769	**769**	14,0	**769**	0,09	**769**	0,02
s526n	554	16	862	6	523	523	**523**	12,0	**523**	0,08	**523**	0,01
s641	466	48	395	23	219	221	**221**	25,0	**221**	0,24	**221**	0,04
s713	582	55	557	19	226	250	**250**	10,0	**250**	0,37	**250**	0,06
s820	816	38	669	13	335	347	**347**	14,0	**347**	0,19	**347**	0,03
s832	837	10	425	10	184	196	**196**	12,0	**196**	0,16	**196**	0,03
s838	929	37	1323	5	323	412	**473**	16,0	482	0,18	**473**	0,05
s938	929	37	1323	5	323	412	**473**	16,0	482	0,18	**473**	0,05
s953	1053	75	1099	26	447	539	**539**	20,0	**539**	0,91	**539**	0,15
s967	1038	72	1223	27	606	669	**669**	19,0	679	0,82	**669**	0,14
s991	877	20	448	9	365	365	**365**	5,0	**365**	0,30	**365**	0,05
s1196	1215	133	1805	67	1086	1124	1131	43,0	**1124**	0,93	**1124**	0,18
s1238	1327	123	1554	62	956	976	1007	50,0	**1004**	0,91	**1004**	16,4
s1269	1309	52	450	23	198	245	**245**	17,0	**245**	0,70	**245**	3,36
s1423	1516	107	2691	25	1237	1279	1284	46,0	**1279**	0,99	**1279**	0,20
s1488	1471	65	1824	16	867	946	**946**	24,0	**946**	1,07	**946**	0,20
s1494	1491	62	1244	19	652	652	**652**	25,0	**652**	0,89	**652**	0,15
s1512	1281	52	772	12	261	289	**289**	27,0	294	0,55	**289**	0,09
s3271	3206	132	2529	43	1047	1087	1532	111,0	1210	5,40	**1178**	27,6
s3330	2866	108	2028	39	1018	1067	**1067**	110,0	1070	2,86	**1067**	0,51
s3384	3360	58	888	17	327	410	**410**	60,0	**410**	2,85	**410**	0,49
s4863	4666	112	1533	31	666	738	746	154,0	749	5,64	**745**	2,41
s5378	4603	71	919	37	464	493	**493**	139,0	**493**	3,24	**493**	0,58
s6669	6506	64	592	29	240	301	303	114,0	**301**	6,62	**301**	1,12
s13207	9810	34	544	6	159	187	**187**	207,0	**187**	1,70	**187**	0,32
s15850	11724	10	153	2	79	91	**91**	96,0	**91**	0,44	**91**	0,09
s35932	38449	59	903	6	247	308	**308**	706,0	**308**	25,0	**308**	4,39
s38417	27733	95	1617	22	588	672	697	1601,0	698	11,78	**684**	2,75
s38584	36303	271	8065	95	3416	3705	N/A	4918,0	3812	60,0	**3806**	341,9

* - SUN SPARC 5 workstation
** - SUN UltraSPARC 60 server
bold numbers – best compaction result
underlined numbers – after pruning, all faults are covered