Test Set Compaction Algorithms for Combinational Circuits [*]

Ilker Hamzaoglu and Janak H. Patel
Center for Reliable & High-Performance Computing
University of Illinois, Urbana, IL 61801

Abstract

This paper presents two new algorithms, Redundant Vector Elimination (RVE) and Essential Fault Reduction (EFR), for generating compact test sets for combinational circuits under the single stuck at fault model, and a new heuristic for estimating the minimum single stuck at fault test set size. These algorithms together with the dynamic compaction algorithm are incorporated into an advanced ATPG system for combinational circuits, called MinTest. MinTest found better lower bounds and generated smaller test sets than the previously published results for the ISCAS85 and full scan version of the ISCAS89 benchmark circuits.

1 Introduction

Compact test sets are very important for reducing the cost of testing the VLSI circuits by reducing the test application time. This is especially important for the scan-based circuits as the test application time for these circuits is directly proportional to product of the test set size and the number of storage elements used in the scan chain. Small test sets also reduce the test storage requirements.

Since even the problem of estimating the size of a minimum single stuck at fault test set for a given irredundant combinational circuit is proven to be NP-hard [10], several test set compaction algorithms based on different heuristics are proposed in the literature, e.g. static compaction [6], dynamic compaction [6], independent and compatible fault sets based test generation [1, 9, 12, 14], reverse order fault simulation [13], maximal compaction [12], rotating backtrace [12], double detection [8, 9], Two_by_one [9], Three_by_two [9], forced pair merging [4] and essential fault pruning [4].

Although these algorithms are successful in producing small test sets, the resulting test sets are still larger than the known lower bounds. This is because of the following two reasons; the previously published test set compaction algorithms are unable to compact the test sets any further, and the known lower bounds are not tight. In order to close this gap further, this paper addresses both of these problems. We present two new test set compaction algorithms, Redundant Vector Elimination (RVE) and Essential Fault Reduction (EFR), and a new heuristic for estimating the minimum single stuck at fault test set size. These algorithms and the dynamic compaction algorithm proposed in [6] are incorporated into an advanced ATPG system for combinational circuits [7], called MinTest. MinTest found better lower bounds and generated smaller test sets than the previously published results for the ISCAS85 and full scan version of the ISCAS89 benchmark circuits [2, 3].

The rest of the paper is organized as follows. Section 2 presents the definitions that will be used in this paper. Section 3 presents the RVE algorithm. The EFR algorithm is presented in Section 4. Minimum Test Set Size estimation heuristic is described in Section 5. The experimental results are given in Section 6. Finally Section 7 presents the conclusions.

2 Preliminaries

In this section, we present the definitions that will be used in this paper. A test vector in a given test set is called an *essential vector*, if it detects at least one fault that is not detected by any other test vector in this test set. A fault is defined to be an *essential fault* of a test vector, if it is detected only by this test vector in a given test set [4, 9]. In other words an essential vector detects at least one essential fault. A test vector is *redundant* with respect to a given test set, if it does not detect any essential faults, i.e. all the faults detected by it are also detected by the other test vectors in this test set [9].

An essential fault ef_i of a test vector t_i is said to be *pruned*, if a test vector $t_j \neq t_i$ in the test set is replaced by a new test vector t'_j which detects ef_i, the essential faults of t_j and the faults detected only by t_i and t_j [4].

If two faults can be detected by a single test vector, they are called *compatible*. Similarly two faults are called *incompatible*, if they cannot be detected by a single test vector. An *incompatibility graph* for a given set of faults, FS = $\{ f_i \mid 1 \leq i \leq n \}$, is defined as IG(FS) = (V, E) where V = $\{ v_i = f_i \mid 1 \leq i \leq n \}$ and E = $\{ e_j = (v_k, v_l) \mid v_k$ and v_l are incompatible, $1 \leq k \leq n$ and $1 \leq l \leq n \}$ [1, 4, 9, 14]. A fault set is called an *independent fault set*, if all the faults in this

[*]This research was supported in part by the Semiconductor Research Corporation under contract SRC 96-DP-109 and in part by DARPA under contract DABT63-95-C-0069.

```
REDUNDANT VECTOR ELIMINATION (t : test vector)
{
  Fault simulate test vector t for each fault that is not proven to be redundant
  For each fault $f_i$ that is detected by vector t
  {
    detection_count[$f_i$]++
    if (detection_count[$f_i$] == 1) then
      number_of_essential_faults[t]++
    else if (detection_count[$f_i$] == 2) then
    {
      Identify the vector $t_j \neq$ t that detects $f_i$
      number_essential_faults[$t_j$]−−
      if (number_of_essential_faults[$t_j$] == 0) then
      {
        drop vector $t_j$ from the test set
        For each fault $f_k$ that is detected by $t_j$
        {
          detection_count[$f_k$]−−
          if (detection_count[$f_k$] == 1) then
          {
            Identify the vector $t_k \neq t_j$ that detects $f_k$
            number_of_essential_faults[$t_k$]++
          }
        }
      }
    }
  }
}
```

Figure 1: RVE algorithm

set are pairwise incompatible [1]. For a given combinational circuit an independent fault set of maximum size is called a *maximum independent fault set*. Since the problem of finding a maximum independent fault set is NP-hard [10], *maximal independent fault sets* are used in practice.

Minimum test set size of a given combinational circuit under the single stuck at fault model is defined to be the minimum number of test vectors required to detect all the testable single stuck at faults in this circuit.

3 Redundant Vector Elimination

During automatic test pattern generation, some of the faults detected by the earlier test vectors may also be accidentally detected by the test vectors generated later. As a result as more vectors are generated during the ATPG process, a test vector generated earlier may become redundant. Redundant Vector Elimination (RVE) algorithm identifies these redundant vectors during test generation and dynamically drops them from the test set. As it is shown in Figure 1, RVE fault simulates all the faults in the fault list except the ones that are proven to be untestable, and it keeps track of the faults detected by each vector, the number of essential faults of each vector and the number of times a fault is detected. During test generation if the number of essential faults of a vector reduces to zero, i.e. the vector becomes redundant, it is dropped from the test set.

As illustrated in the example below, RVE algorithm can reduce the size of a test set more than Reverse Order Fault Simulation (ROFS) [13]. This is because ROFS cannot identify a redundant test vector if some of the faults detected by it are only detected by the test vectors generated earlier. It can only identify a redundant vector, if all the faults detected by it are also detected by the test vectors generated later.

```
ESSENTIAL FAULT REDUCTION (T : test set, NumIteration: int,
MFL : int, MEFL : int)
{
  For NumIteration times
  {
    For each test vector $t_i$ in T with less than MEFL essential faults
    {
      all_ef_pruned = true
      failure_limit = 0
      For each essential fault $f_j$ of $t_i$
      {
        pruned = false
        For each test vector $t_k \neq t_i$
        {
          pruned = Multiple_Target_Test_Generation($t_k$, $t_i$, $f_j$)
          if (pruned == true) then break
        }
        if (pruned == true) then
        {
          Update T by replacing $t_k$ with the new test vector
          Fault simulate the new test vector
        }
        else
        {
          failure_limit++
          all_ef_pruned = false
          if (failure_limit == MFL) then break
        }
      }
      if (all_ef_pruned == true) then drop vector $t_i$ from T
    }
  }
}
```

Figure 2: EFR algorithm

Example: Consider the fault set $\{f_1, f_2, f_3, f_4\}$. Suppose that for this fault set the ATPG system generated the test set $\{ t_1, t_2, t_3 \}$ in the given order, and t_1 detects the faults f_1 and f_2, t_2 detects the faults f_3 and f_1, and t_3 detects the faults f_4 and f_3. In this example, after t_3 is generated, RVE algorithm detects that t_2 becomes redundant and drops it from the test set. Thus it reduces the test set to $\{ t_1, t_3 \}$. However, ROFS cannot reduce the size of this test set.

The performance of the RVE algorithm is similar to the Double Detection (DD) algorithm introduced in [8, 9], even though slightly different results may be produced because of the order of dropping redundant vectors. However, we are not proposing RVE as a standalone test set compaction algorithm, rather as the first step of a two-step compaction framework that includes both RVE and Essential Fault Reduction (EFR) algorithms. In addition to the number of essential faults for each test vector, which is also obtained by DD, EFR needs the additional information that is produced by RVE; faults detected by each test vector and the exact number of times each fault is detected by the current test set. If DD is used instead of RVE, then EFR itself should obtain this information. Since RVE spends most of its execution time for computing this information, the execution time of the RVE and EFR algorithms combined is smaller than the DD and EFR algorithms combined.

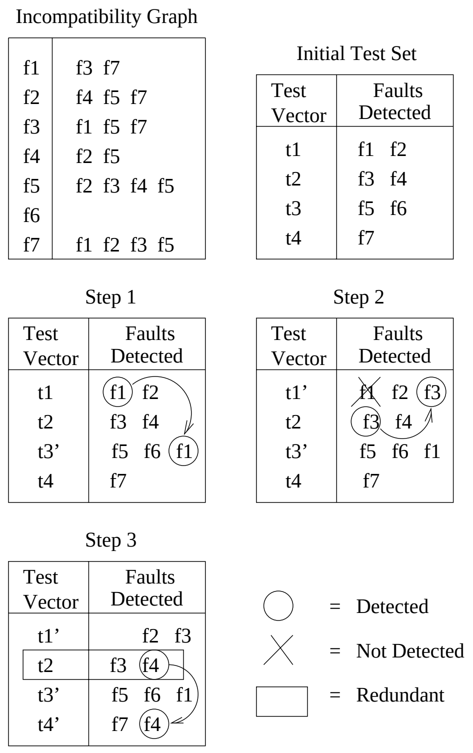
4 Essential Fault Reduction

Since pruning an essential fault of a test vector decreases the number of its essential faults by one, if all the essential faults of a test vector is pruned then it becomes redundant, and it can be dropped from the test set. As it is shown in Figure 2 after the initial test set is generated, Essential Fault Reduction (EFR) algorithm is used iteratively to further compact the test set by pruning the essential faults of each vector as much as possible. EFR uses the Multiple Target Test Generation (MTTG) procedure [4, 9] to generate a test vector that will detect a given set of faults. EFR algorithm improves the Two_by_One (TBO) [9] and the Essential Fault Pruning (EFP) algorithms [4].

Given an initial test set, TBO tries to reduce the test set size by replacing two test vectors with a new one. This is achieved by finding a test vector that detects the essential faults of the both vectors as well as the faults detected only by these two vectors. However, even if it is not possible to find such a test vector, it may still be possible to eliminate these two test vectors from the test set. This may be achieved by a three_by_two algorithm (TBT) which tries to replace three test vectors with two new ones. In general, the algorithm can be extended to an N_by_M (M $<$ N) algorithm. However, in the worst case, TBO needs to check O(V^2) vector pairs for possible compaction, where V is the number of test vectors in the initial test set, TBT needs to check O(V^3) vector triplets, and in general N_by_M algorithm needs to check O(V^N) vector sets. Thus, the N_by_M algorithm is computationally too expensive for N $>$ 2, and implementation of an N_by_M algorithm where N $>$ 2 is not reported.

EFP, on the other hand, tries to reduce the test set size by trying to prune the essential faults of each test vector. If all the essential faults of a test vector is pruned, then this vector becomes redundant and it can be dropped from the test set. TBO can be seen as a special case of EFP in which a test vector is allowed to prune its essential faults by replacing only one vector. EFP achieves better performance than TBO by relaxing this restriction and allowing a test vector to prune its essential faults by replacing more than one vector in the test set. In the worst case, EFP will try to generate a test vector for O(E x V) fault sets, where E is the number of essential faults and V is the number of test vectors in the initial test set. Since in almost all cases E is larger than V, EFP is computationally more expensive than TBO. However, for N $>$ 2 in most cases N_by_M algorithm is computationally more expensive.

The problem of compacting a given test set can be viewed as distributing the essential faults of this test set to the given test vectors such that the number of redundant vectors is maximized. Therefore, the search space that should be explored is all possible distributions of the essential faults to the given test vectors. Since neither TBO nor EFP algorithms have this global view of the search space, they carry out a localized greedy search by concentrating only on removing one test vector at a time from the test set by pruning its essential faults. They prune an essential fault of a test vector only if this causes this vector to be redundant, otherwise they do not prune the essential fault. Because of this restriction, they only explore part of the search space.



Incompatibility Graph

f1	f3 f7
f2	f4 f5 f7
f3	f1 f5 f7
f4	f2 f5
f5	f2 f3 f4 f5
f6	
f7	f1 f2 f3 f5

Initial Test Set

Test Vector	Faults Detected
t1	f1 f2
t2	f3 f4
t3	f5 f6
t4	f7

Step 1

Test Vector	Faults Detected
t1	f1 f2
t2	f3 f4
t3'	f5 f6 f1
t4	f7

Step 2

Test Vector	Faults Detected
t1'	~~f1~~ f2 f3
t2	f3 f4
t3'	f5 f6 f1
t4	f7

Step 3

Test Vector	Faults Detected
t1'	f2 f3
t2	f3 f4
t3'	f5 f6 f1
t4'	f7 f4

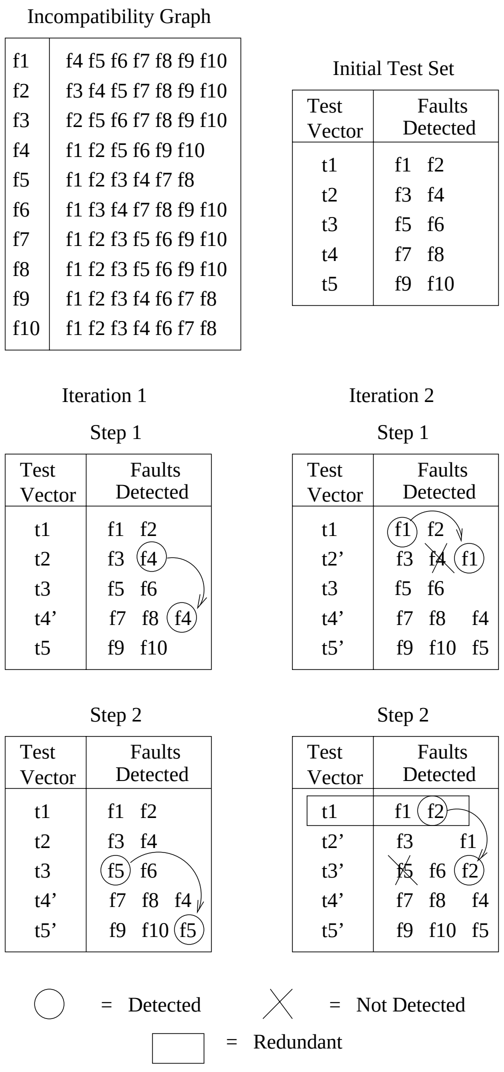
○	= Detected
✕	= Not Detected
▭	= Redundant

Figure 3: EFR Example

EFR algorithm, on the other hand, has a global view of the search space. It overcomes the limitation of the TBO and EFP algorithms by carrying out a non-greedy global search by trying to distribute the essential faults to the given test vectors such that the number of redundant vectors is maximized. Therefore, even if a vector does not become redundant, EFR tries to reduce the number of its essential faults as much as possible by trying to prune as many of its essential faults as possible. Even if it fails to prune one of the essential faults of a test vector, it still tries to prune its other essential faults. This way EFR explores a larger portion of the search space than both TBO and EFP. As illustrated in the example below, using this new search technique EFR can produce smaller test sets than the ones produced by TBO, TBT, and EFP algorithms.

Example: Consider the test set $\{t_1, t_2, t_3, t_4\}$. Suppose that t_1 detects the faults $\{f_1, f_2\}$, t_2 detects $\{f_3, f_4\}$, t_3 detects $\{f_5, f_6\}$ and t_4 detects $\{f_7\}$, and the adjacency list representation of the incompatibility graph is as given in Figure 3. EFR can reduce the size of this test set by one in the first iteration. As it is illustrated in Figure 3, this can be achieved by replacing the test vectors t_1 with t'_1 that detects f_2 and f_3, t_3 with t'_3 that detects f_1, f_5, and f_6, and t_4 with t'_4 that detects f_4 and f_7. After these replacements t_2 becomes redundant, thus it can be dropped from the test set. None of the TBO, TBT, and EFP algorithms can reduce the size of this initial test set.

As illustrated in the example below, by means of the new search technique, EFR can further compact a given test set when it is used iteratively. This is not possible with TBO



Incompatibility Graph

f1	f4 f5 f6 f7 f8 f9 f10
f2	f3 f4 f5 f7 f8 f9 f10
f3	f2 f5 f6 f7 f8 f9 f10
f4	f1 f2 f5 f6 f9 f10
f5	f1 f2 f3 f4 f7 f8
f6	f1 f3 f4 f7 f8 f9 f10
f7	f1 f2 f3 f5 f6 f9 f10
f8	f1 f2 f3 f5 f6 f9 f10
f9	f1 f2 f3 f4 f6 f7 f8
f10	f1 f2 f3 f4 f6 f7 f8

Initial Test Set

Test Vector	Faults Detected
t1	f1 f2
t2	f3 f4
t3	f5 f6
t4	f7 f8
t5	f9 f10

Iteration 1

Step 1

Test Vector	Faults Detected
t1	f1 f2
t2	f3 f4
t3	f5 f6
t4’	f7 f8 f4
t5	f9 f10

Step 2

Test Vector	Faults Detected
t1	f1 f2
t2	f3 f4
t3	f5 f6
t4’	f7 f8 f4
t5’	f9 f10 f5

Iteration 2

Step 1

Test Vector	Faults Detected
t1	f1 f2
t2’	f3 ~~f4~~ f1
t3	f5 f6
t4’	f7 f8 f4
t5’	f9 f10 f5

Step 2

Test Vector	Faults Detected
t1	f1 f2
t2’	f3 f1
t3’	~~f5~~ f6 f2
t4’	f7 f8 f4
t5’	f9 f10 f5

○ = Detected ✗ = Not Detected

□ = Redundant

Figure 4: EFR Iteration Example

and TBT. Although it is possible that EFP may further compact a given test set when it is used iteratively, this is very unlikely. Because this can only happen if one of the new test vectors “accidentally” detects one or more essential faults of the other test vectors that it is not intended to detect. This may make it possible to prune the essential faults of a test vector in the second iteration, even though it was not possible in the first iteration.

Example: Consider the test set $\{t_1, t_2, t_3, t_4, t_5\}$. Suppose that t_1 detects the faults $\{f_1,\ f_2\}$, t_2 detects $\{f_3,\ f_4\}$, t_3 detects $\{f_5,\ f_6\}$, t_4 detects $\{f_7,\ f_8\}$, and t_5 detects $\{f_9,\ f_{10}\}$, and the adjacency list representation of the incompatibility graph is as given in Figure 4. As it is illustrated in Figure 4, in the first iteration of the EFR algorithm only f_4 will be pruned by replacing t_4 with t_4', and f_5 will be pruned by replacing t_5 with t_5'. Thus, in the first iteration EFR will not be able to reduce the test set size. However, since after the first iteration f_4 is not an essential fault of t_2 and f_5 is not an essential fault of t_3 anymore, in the second iteration f_1 will be pruned by replacing t_2 with t_2' and f_2 will pruned be replacing t_3 with t_3'. Because of these t_1 will become redundant, and it will be dropped from the test set. On the other hand, none of the TBO, TBT, and EFP algorithms can reduce the size of this initial test set.

EFR has the same worst-case computational complexity as EFP, i.e. it will try to generate a test vector for O(E x V) fault sets, where E is the number of essential faults and V is the number of test vectors in the initial test set. If it is used iteratively then the worst-case complexity becomes O(I x E x V), where I is the number of iterations.

An incompatibility graph is used for reducing the average case execution time of TBO and EFP algorithms [4, 9]. However, the incompatibility graph definition used in [1, 4, 9, 14] cannot represent the following incompatibility relation between stuck at faults. It is possible that even though a fault is pairwise compatible with all the faults in a given fault set, it may be incompatible with these faults when they are targeted together. Therefore, we extended the definition of the incompatibility graph by allowing a graph node to represent a set of faults. This new incompatibility graph is constructed in a demand-driven way during compaction. We have observed that using this incompatibility graph reduced the execution time of EFR algorithm.

5 Minimum Test Set Size Estimation

To be able to assess the effectiveness of test set compaction algorithms for a combinational circuit, it is necessary to know the Minimum Test Set Size (MTSS) for this circuit. In addition if the MTSS is known, test set compaction time can be reduced by stopping the iteration of EFR algorithm whenever the minimum test set size is reached rather than iterating a predetermined number of times. Since the problem of computing the size of a minimum single stuck at fault test set for a given irredundant combinational circuit is proven to be NP-hard [10], heuristic techniques are used for finding a lower bound for MTSS.

One of the most commonly used heuristics for finding a lower bound is finding the size of the maximal independent fault set. The size of the maximal independent fault set is less than or equal to the minimum test set size. The maximal independent fault set can be computed by finding the maximal clique in the incompatibility graph of the given single stuck at fault set [1, 4, 8, 9, 11, 14].

Since the problem of finding the maximal clique in a given graph is proven to be NP-complete [5], several heuristics are proposed for finding a maximal clique in a given incompatibility graph. Since the essential faults of the test vectors in a given test set are highly incompatible, in [4] it is suggested to compute the maximal clique by first considering only the essential faults and then enlarging this clique by considering the other faults. In [9], it is reported that for some circuits computing the maximal clique in the incompatibility graph that is constructed only by using the essential faults found larger cliques than computing the maximal clique in the incompatibility graph that is constructed by considering all the faults. Based on the following theorem and the corollary, we also compute the maximal clique by first considering only the essential faults and then enlarging this clique by considering the other faults in the given fault list.

Theorem 1: Given a complete single stuck at fault test set TS of size N for a combinational circuit, if there exists a maximal independent fault set MIFS of size K ($N/2 \leq$ K $\leq$ N) for this circuit, then that MIFS contains at least 2K-N essential faults each from a different test vector.

Proof: If there exists a maximal independent fault set MIFS of size K, then by definition there are K pairwise incompatible faults in MIFS. Since the faults in MIFS are pairwise incompatible, no test vector can detect more than one fault in MIFS. Therefore each fault in MIFS should be detected by a different test vector in TS. Since the test set is complete, at least K test vectors in TS detect these K faults. If none of these K faults are detected by any of the remaining N-K test vectors in TS, then each one of them is an essential fault of a different test vector in TS. In the worst case, each one of the remaining N-K test vectors detects a different fault in MIFS. Since these N-K faults are detected by two vectors, they are not essential faults. Therefore, in the worst case, K-(N-K) = 2K-N of the faults in MIFS is an essential fault of a different test vector in TS. In other words, MIFS contains at least 2K-N essential faults each from a different test vector. $\square$

Corollary: Given a complete single stuck at fault test set TS of size N for a combinational circuit, if there exists a maximal independent fault set MIFS of size N for this circuit, then that MIFS contains one essential fault from each test vector in TS.

Proof: According to Theorem 1, MIFS should contain one essential fault from at least 2K-N test vectors in TS where K is the size of the maximal independent fault set. For K=N the expression 2K-N is equal to N. Since TS has N test vectors, this means that MIFS should contain one essential fault from each test vector in TS. $\square$

Theorem 1 shows that the maximal clique in the incompatibility graph of a small test set is likely to contain many essential faults, and the corollary indicates that for a minimum size test set if there exists a clique of this size in the incompatibility graph then the clique contains only essential faults. Since EFR algorithm produces test sets that are either minimum size or very close to it, based on this theorem, we compute the lower bound for MTSS by searching for the maximal clique that includes one essential fault from as many test vectors as possible in the test set produced by EFR algorithm and then enlarging this clique by considering the other faults in the given fault list.

Since the size of the search space for computing the maximal clique by choosing one essential fault from as many test vectors as possible is very large, it is computationally too expensive to search it exhaustively. If there are n vectors $\{t_1, t_2, \ldots, t_n\}$ in the test set, and if the sizes of their essential fault sets are $efs_1, efs_2, \ldots, efs_n$ respectively, then the size of the search space is $O(\prod_{i=1}^{n} efs_i)$. Therefore, we propose the following new heuristic to guide the branch and bound search algorithm. When trying to choose an essential fault from each test vector, consider the vectors in ascending order of the number of essential faults that they have, and explore more branches for the initial test vectors.

This heuristic increases the probability of computing the maximal clique in a short amount of time because of the following reason. Once an essential fault is included in a clique, this reduces the number of essential faults of the remaining test vectors that can be included in this clique. If a test vector t_i has efs_i essential faults and if each one of these essential faults is equally likely to be in the maximal clique, then when trying to select an essential fault of this test vector the probability of choosing the essential fault that is in the maximal clique is $1/efs_i$. If the number of essential faults of t_i that can be included in the maximal clique decreases, the probability of selecting the essential fault that is in the maximal clique increases. Since for the test vectors with small number of essential faults the probability of selecting the essential fault that is in the maximal clique is already high, and after selecting these essential faults, for the test vectors with larger number of essential faults the probability of selecting the essential fault that is in the maximal clique increases, considering the vectors with smaller number of essential faults first increases the overall probability of computing the maximal clique in a short amount of time.

6 Experimental Results

We incorporated the minimum test set size estimation, RVE and EFR algorithms that we propose and the dynamic compaction algorithm proposed in [6] into our advanced ATPG system for combinational circuits [7], called MinTest. MinTest is designed in an object-oriented style and implemented in C++. We tested MinTest on the ISCAS85 and full scan version of the ISCAS89 benchmark circuits [2, 3]. The performance results for MinTest are obtained on a 200 MHz Pentium Pro PC with 128MB RAM running Linux 2.0.0 using GNU CC version 2.8.0.

We compared the performance of MinTest on minimum test set size estimation with the previously published results and presented the comparision of the performance results in Table 1. The "–" sign in the table indicates that the lower bound for this circuit is not reported. The results show that our algorithm computed better lower bounds than the previously published ones. For 38 out of 40 circuits, the lower bounds computed by our algorithm are greater than or equal to the best published lower bounds. For 14 of these 38 circuits our algorithm computed as much as 33% larger lower bounds than the previously published results, e.g. 25% larger for c7552, 33% larger for s1423 and 11% larger for s9234. These 14 new lower bounds are indicated by an asterisk (*) in the table.

Our minimum test set size estimation algorithm is applicable to large circuits, and its execution time is similar to the

		[8]				MinTest	
Circuit	[4]	Loc	Glb	[9]	[11]	LB	Time
c432	24	20	20	–	27	27	15.0
c499	52	50	50	–	52	52	0.1
c880	12	9	10	–	–	13*	21.9
c1355	84	82	82	–	–	84	0.9
c1908	94	91	68	99	–	106*	88.1
c2670	40	38	39	42	–	44*	47.1
c3540	80	67	65	–	–	78	174.5
c5315	37	22	36	–	–	37	748.6
c6288	5	6	6	–	–	6	347.7
c7552	49	26	28	52	–	65*	663.8
TOTAL	477	411	404	–	–	512	2107.7
s208	–	26	27	–	27	27	0.1
s298	–	19	20	–	23	23	0.1
s344	–	13	13	–	13	13	0.3
s349	–	13	12	–	13	13	1.1
s382	–	25	25	–	25	25	0.1
s386	–	62	62	63	63	63	0.1
s400	–	24	23	–	24	24	0.1
s420	–	42	43	–	43	43	0.3
s444	–	24	23	–	24	24	0.1
s510	–	53	53	–	54	54	1.9
s526	–	34	35	–	49	49	2.5
s526n	–	34	35	–	49	49	2.2
s641	–	19	19	–	21	21	2.0
s713	–	19	19	–	21	21	2.7
s820	–	90	90	–	93	93	86.3
s832	–	91	91	–	–	94*	21.6
s838	–	74	75	–	–	75	430.7
s953	–	65	68	73	–	76*	84.7
s1196	–	59	99	105	–	112*	161.0
s1238	–	61	107	115	–	121*	920.7
s1423	–	14	15	–	–	20*	9.9
s1488	–	98	98	100	–	101*	28.7
s1494	–	97	97	100	–	100	62.3
s5378	–	85	92	–	–	97*	198.4
s9234	–	84	88	90	–	100*	6319.4
s13207	–	233	233	–	–	233	722.7
s15850	–	89	85	90	–	91*	938.9
s35932	–	–	–	9	–	9	5687.4
s38417	–	50	60	–	–	62*	12287.9
s38584	–	–	–	93	–	89	10134.5
TOTAL	–	1597	1707	–	–	1922	38108.7

Table 1: Lower Bounds on Minimum Test Set Size

algorithms presented in [8, 9]. The algorithm presented in [11] can only be applied to small circuits, and for these circuits our algorithm computed the same lower bounds with this algorithm. The algorithm presented in [4] is a computationally expensive algorithm, and neither its execution time for ISCAS85 circuits nor its performance for ISCAS89 circuits is reported.

The performance of MinTest on test set compaction is compared against the two best test set compaction algorithms published in the literature, CompacTest (CT) [8, 9, 12] and TSC [4]. The comparison of the performance results is presented in Table 2. In the table, the smallest known test size for each circuit is marked by an asterisk (*). The largest known lower bound on the minimum test set size of each circuit is presented in the LB column. Some of these lower bounds are computed by our minimum test set size estimation algorithm and the rest is taken from [9]. In all the experiments, a backtrack limit of 6 is used in MinTest. The execution times of MinTest include fault simulation and initial test set generation times, and all the test sets generated by MinTest have 100% fault coverage. The performance results for CT and TSC are taken from [9] and [4] respectively. The performance of TSC for ISCAS89 circuits is not reported.

The following observations can be made from the experimental results. For all the circuits, sizes of the test sets generated by MinTest are smaller than or equal to the best published results. For 31 out of 40 circuits, sizes of the test sets generated by MinTest are equal to the known lower bounds for these circuits. Even by executing only one iteration of EFR algorithm, MinTest generated smaller test sets than both CT and TSC for both ISCAS85 and ISCAS89 circuits. Moreover, for some circuits MinTest produced even smaller test sets by executing EFR algorithm iteratively. The test sets generated by MinTest are as much as 23% smaller than the previously published results, e.g. 16% smaller for c5315, 23% smaller for s1423, and 20% smaller for s38417.

In order to measure the performance of EFR algorithm when it is used iteratively, we iterated it 3 times for the circuits for which the lower bound is not achieved after the first iteration. As it can be seen in the column headed “3 its”, for some of these circuits MinTest produced even smaller test sets when EFR is used iteratively. When EFR algorithm is iterated more than 3 times, MinTest produced even smaller test sets for nine circuits. These results are presented in the column headed “$>$ 3 its”. Next to the test set sizes presented in the columns headed “3 its” and “$>$ 3 its”, we indicated the iteration number that this test set size is reached in parenthesis. The times presented in the column headed “$>$ 3 its” are the execution times of MinTest only for this many iterations.

The CT and TSC execution times presented in Table 2 are obtained on a SUN SPARC 2 workstation. The compaction times presented for CT and MinTest include the initial test generation time as well. However, the compaction times presented for TSC only show the execution time of TSC starting from a given initial test set. Since MinTest is exploring a larger search space, its execution time is larger than that of CT. In [4], it is reported that to be within a reasonable running time, currently, TSC is only applicable to the medium size circuits with the largest being c7552. However, the experimental results show that MinTest is applicable to large circuits.

7 Conclusions

This paper presented two new algorithms for generating compact test sets for combinational circuits under the single stuck at fault model, and a new heuristic for estimating the minimum single stuck at fault test set size. These algorithms together with the dynamic compaction algorithm are incorporated into an advanced ATPG system for combinational circuits, called MinTest. MinTest found better lower bounds and generated smaller test sets than the previously published results for the ISCAS85 and full scan version of the ISCAS89

Circuit	Test Set Size						Time (secs)				
				MinTest					MinTest		
	LB	CT	TSC	1 it	3 its	> 3 its	CT	TSC	1 its	3 its	> 3 its
c432	27	29	29	27*	–	–	7	13.6	6.2	–	–
c499	52	52*	53	52*	–	–	5	13.2	17.4	–	–
c880	13	21	18	20	18(2)	16*(10)	12	36.4	10.4	20.5	50.9
c1355	84	84*	86	84*	–	–	16	58.5	29.4	–	–
c1908	106	106*	106*	106*	–	–	55	257.8	78.9	–	–
c2670	44	45	44*	44*	–	–	130	246.1	73.3	–	–
c3540	80	91	90	87	87	84*(15)	262	423.1	178.1	305.1	1372.9
c5315	37	44	46	41	40(3)	37*(12)	362	1126.3	265.4	573.8	1983.5
c6288	6	14	14	13	13	12*(8)	398	419.4	65.6	134.5	306.9
c7552	65	80	76	73*	73	–	1311	1931.8	794.7	1733.8	–
TOTAL	514	566	562	547	544	535	2558	4526.2	1519.4	–	–
s208	27	27*	–	27*	–	–	0.8	–	0.4	–	–
s298	23	24	–	23*	–	–	1.5	–	0.7	–	–
s344	13	15	–	13*	–	–	1.5	–	0.7	–	–
s349	13	14	–	13*	–	–	1.7	–	0.7	–	–
s382	25	25*	–	25*	–	–	1.7	–	0.8	–	–
s386	63	63*	–	63*	–	–	3.8	–	3.1	–	–
s400	24	24*	–	24*	–	–	1.8	–	0.8	–	–
s420	43	43*	–	44	43*(2)	–	3.2	–	2.1	2.9	–
s444	24	24*	–	24*	–	–	2.3	–	0.9	–	–
s510	54	54*	–	54*	–	–	6.0	–	3.6	–	–
s526	49	50	–	49*	–	–	4.9	–	3.0	–	–
s526n	49	50	–	49*	–	–	4.9	–	3.3	–	–
s641	21	22	–	21*	–	–	3.1	–	2.1	–	–
s713	21	22	–	21*	–	–	4.6	–	2.8	–	–
s820	93	94	–	94	93*(2)	–	19	–	27.7	34.1	–
s832	94	94*	–	95	95	94*(12)	20	–	23.9	28.7	80.1
s838	75	75*	–	76	75*(2)	–	13	–	11.9	15.3	–
s953	76	76*	–	76*	–	–	25	–	30.3	–	–
s1196	113	118	–	113*	–	–	48	–	43.6	–	–
s1238	121	124	–	122	122	121*(4)	102	–	68.7	90.8	127.4
s1423	20	26	–	22	22	20*(25)	32	–	14.6	31.0	205.3
s1488	101	101*	–	101*	–	–	40	–	75.1	–	–
s1494	100	100*	–	100*	–	–	43	–	80.4	–	–
s5378	97	103	–	97*	–	–	216	–	131.5	–	–
s9234	100	108	–	106	106	105*(5)	1085	–	1103.6	2020.7	3157.1
s13207	233	235	–	233*	–	–	1096	–	1178.4	–	–
s15850	91	95*	–	96	96	95*(17)	1375	–	1406.1	2189.3	9252.2
s35932	9	13	–	12*	12	–	8388	–	10075.7	11334.5	–
s38417	62	85	–	70	68*	–	13210	–	11773.6	28955.8	–
s38584	93	115	–	111	110*	–	14446	–	17219.7	38538.9	–
TOTAL	1927	2019	–	1974	1968	1962	40199.8	–	43289.8	–	–

Table 2: Compaction Results

benchmark circuits.

References

[1] S. B. Akers, C. Joseph, and B. Krishnamurthy, "On the Role of Independent Fault Sets in the Generation of Minimal Test Sets", in *Proc. of the Int. Test Conf.*, pp. 1100-1107, August 1987.

[2] F. Brglez and H. Fujiwara, "A Neutral Netlist of 10 Combinational Benchmark Designs and a Special Translator in Fortran", in *Proc. of the Int. Symp. on Circuits and Systems*, June 1985.

[3] F. Brglez, D. Bryan, and K. Kozminski, "Combinational Profiles of Sequential Benchmark Circuits", in *Proc. of the Int. Symp. on Circuits and Systems*, pp. 1929-1934, May 1989.

[4] J.-S. Chang and C.-S. Lin, "Test Set Compaction for Combinational Circuits", *IEEE Trans. on Computer-Aided Design*, pp. 1370-1378, November 1995.

[5] M. R. Garey and D. S. Johnson, *Computers and Intractability: A Guide to the Theory of NP-Completeness*, W. H. FreeMan, 1979.

[6] P. Goel and B.C. Rosales, "Test Generation and Dynamic Compaction of Tests", in *Digest of Papers 1979 Test Conf.*, pp. 189-192, October 1979.

[7] I. Hamzaoglu and J. H. Patel, "New Techniques for Deterministic Test Pattern Generation", in *Proc. of the IEEE VLSI Test Symp.*, pp. 446-452, April 1998.

[8] S. Kajihara, I. Pomeranz, K. Kinoshita and S. M. Reddy, "Cost Effective Generation of Minimal Test Sets for Stuck at Faults in Combinational Logic Circuits", in *Proc. of the Design Automation Conf.*, pp. 102-106, June 1993.

[9] S. Kajihara, I. Pomeranz, K. Kinoshita and S. M. Reddy, "Cost Effective Generation of Minimal Test Sets for Stuck at Faults in Combinational Logic Circuits", *IEEE Trans. on Computer-Aided Design*, pp. 1496-1504, December 1995.

[10] B. Krishnamurthy and S. B. Akers, "On the Complexity of Estimating the Size of a Test Set", *IEEE Trans. on Computers*, pp. 750-753, August 1984.

[11] Y. Matsunaga, "MINT - An Exact Algorithm for Finding Minimum Test Sets", *IEICE Trans. Fundamentals*, pp. 1652-1658, October 1993.

[12] I. Pomeranz, L. Reddy, and S. M. Reddy, "Compactest: A Method To Generate Compact Test Sets for Combinational Circuits", in *Proc. of the Int. Test Conf.*, pp. 194-203, October 1991.

[13] M. H. Schulz, E. Trischler, and T. M. Sarfert, "SOCRATES: A highly efficient automatic test pattern generation system", *IEEE Trans. on Computer-Aided Design*, pp. 126-137, January 1988.

[14] G.-J. Tromp, "Minimal Test Sets for Combinational Circuits", in *Proc. of the Int. Test Conf.*, pp. 204-209, October 1991.