# Independent Test Sequence Compaction through Integer Programming

Petros Drineas
Computer Science Department
Rensselaer Polytechnic Institute
Troy, NY 12180
drinep@cs.rpi.edu

Yiorgos Makris
Electrical Engineering Department
Yale University
New Haven, CT 06520
yiorgos.makris@yale.edu

## Abstract

*We discuss the compaction of independent test sequences for sequential circuits. Our first contribution is the formulation of this problem as an integer program, which we then solve through a well-known method employing linear programming relaxation and randomized rounding. The key contribution of this approach is that it yields the first polynomial time approximation algorithm for this problem. More specifically, it provides a provably good approximation guarantee while running in time polynomial with respect to the number of vectors in the original test sequences and the number of faults. Another virtue of our approach is that it provides a lower bound for the compacted set of test sequences and, therefore, a quality measure for the test compaction algorithm. Experimental results on benchmark circuits demonstrate that the proposed solution efficiently identifies nearly optimal sets of compacted test sequences.*

## 1. Introduction

Deterministic test generation methods typically target a primary fault and generate a test sequence for detecting it. Since the generated test sequence may also detect ancillary faults, fault simulation is subsequently employed and both the primary and the ancillary faults are eliminated from the fault list. The same fault dropping mechanism is also employed in simulation-based test generation methods, wherein random, pseudo-random, or algorithmically constructed test sequences are fault-simulated on the circuit. In either case, the primary objective is the derivation of a set of test sequences that detects all faults and fault dropping is an essential element in order to reduce test generation time. As a result, test generation methods typically produce a sub-optimal set of test sequences, i.e. a set wherein some test sequences (or portions thereof) may be redundant. Elimination or pruning of redundant test sequences is the objective of test compaction, which may be performed either during test generation (dynamic compaction), or after test generation (static compaction). Efficient test compaction methods are very important in order to reduce test storage, test application time, and by extension, test cost.

In this paper, we study a specific instance of the problem, namely the compaction of independent test sequences for sequential circuits. Such test sequences do not rely on any assumptions regarding the initial state of the circuit and are, thus, independent of it. It is also assumed that each test sequence is fault simulated only once, yet without fault dropping so that all detectable faults are obtained. Based on this information, it is possible that some test sequences may be eliminated or pruned without any reduction in fault coverage. Since each test sequence consists of a number of test vectors, the optimization objective of test compaction in this scenario is the minimization of the total number of test vectors in the compacted set of test sequences.

This instance of test compaction was first formulated in [1], where it is shown to be NP-hard and an approximate solution is computed through Genetic Algorithms. While significant levels of compaction within reasonable time are experimentally observed, no indication of proximity to the optimal solution is provided through this method. In an alternative method described in [2], an *exact* algorithm (i.e. not an approximation algorithm) that computes the optimal solution using a branch-and-bound method is proposed. Even though this algorithm works well for most benchmarks, is also lacks any provable, sub-exponential running time guarantee.

These deficiencies are formally addressed through the work presented herein; more specifically, we contribute a formulation of the problem as an Integer Program, which is subsequently approximated through Randomized Rounding [3] of the optimal solution to its Linear Programming relaxation. This approach has three major advantages:

1. The cost of the optimal solution to the Linear Programming relaxation is a *lower bound* for the cost of the optimal set of compacted test vectors. Such a lower bound not only establishes a mechanism for assessing the quality of test compaction, but may also provide an informed termination criterion for iterative approaches, such as the solution proposed in [1].

2. Unlike all previous approaches, the worst-case theoretical running time of our algorithm is $poly(N+k)$, where $N$ is the total number of vectors in the non-compacted test sequences and $k$ is the number of faults in the circuit.

Proceedings of the 21st International Conference on Computer Design (ICCD'03)
1063-6404/03 $ 17.00 © 2003 IEEE

3. We can theoretically prove that the cost of the solution identified by our algorithm is close to the cost of the optimal solution.

We should note here that previous approaches are quite accurate and fast in practice. Nevertheless, they are essentially heuristics that lack *any* theoretical analysis and provable bounds either with respect to the running time or the accuracy of the approximation (apart from the obvious exponential bounds). Hence, we provide the **first polynomial time approximation algorithm** for compaction of independent test sequences for sequential circuits. On the negative side, the error bound of our approximation algorithm is rather pessimistic, but experiments with alternative test sets for the ISCAS89 [4] benchmark circuits show that the proposed solution yields almost optimal solutions.

The rest of this paper is organized as follows: an extensive review of research efforts in test compaction is provided in Section 2. The problem of Compaction of Independent Test Sequences is formally defined and, for the purpose of completeness, the formulation of [1] is reviewed in Section 3. The proposed formulation as an Integer Program is described in Section 4 and the corresponding solution via Linear Programming relaxation and Randomized Rounding [3] is presented and analyzed in Section 5. Experimental results in support of both the proposed method and the method of [1] are provided in Section 6.

## 2. Related Work

The importance of test compaction is accentuated by the plethora of research efforts reported in the literature. Most of these approaches address different instances of the test compaction problem than the one targeted herein. A direct comparison to the proposed method can not be attempted, yet we provide references to an extensive list of solutions proposed for several instances of test compaction.

Several heuristics have been proposed for *static* test compaction in combinational circuits. Reverse order simulation [5, 6], compatibility analysis of partially specified vectors [7], forced pair merging along with essential fault pruning [8], and redundant vector elimination along with essential fault reduction [9] are the most notable ones. Similarly, many heuristics have been proposed for *dynamic* test compaction in combinational circuits [10]. These include compaction based on independent and compatible fault analysis which was introduced in [11] and was further improved in [12], maximal compaction and rotating backtrace [13], double detection, two-by-one, and three-by-two [14], as well as an attempt to formulate dynamic test generation of minimal test sets through Integer Programming [15].

Sequential circuits impose more stringent constraints on test compaction, since the effectiveness of test vectors relies on the particular order of application. Any reordering may require additional fault simulation to reassess fault coverage. *Static* test sequence compaction heuristics requiring only one fault simulation pass are proposed based on compatibility with skew or stretch in [16] and based on Genetic Algorithms in [1]. Heuristics that require two fault simulation passes include inert and recurrence subsequence removal [17], as well as state-relaxation based removal [18]. More expensive methods employing several passes of fault simulation have also been proposed based on vector restoration [19], segment reordering and accelerated vector restoration [20], fault restoration [21], vector insertion, omission, and selection [22], vector replacement [23] and forward-looking fault simulation [24]. Several *dynamic* test compaction methods have also been devised for sequential circuits. Interleaving of fault-oriented and fault-independent phases is employed in [25] and heuristics for selecting secondary target faults to complete a partially-specified sequence generated for a primary fault are introduced in [26]. Heuristics for improving the effect of random filling of partially-specified sequences are also proposed in [27]. The static compaction methods of [22] are employed for dynamic compaction in [28] and a symbolic BDD-based method is given in [29]. Test vector removal and random filling based on Genetic Algorithms is described in [30] and a property-based dynamic test compaction is devised in [31].

## 3. Compaction of Independent Test Sequences

We recall the problem formulation set forth in [1]. Assume that we are given a list of faults (say $f_1 \dots f_k$) and a list of independent test sequences (say $s_1 \dots s_m$) for a sequential circuit. Each test sequence consists of a number of vectors that need to be applied **in sequence**: we will denote by $n_i$ the number of vectors in sequence $s_i$ for all $i = 1 \dots m$. In general, we will denote by $v_{i1}, v_{i2}, \dots, v_{in_i}$ the vectors that comprise the sequence $s_i$; more compactly, $s_i = \{v_{i\ell}\}_{\ell=1}^{n_i}$.

In [1], the authors create an $m \times k$ matrix $F$ such that $F_{ij}$ is non-zero if and only if fault $f_j$ is detected by the test sequence $s_i$; in particular, if $s_i = \{v_{i\ell}\}_{\ell=1}^{n_i}$, $F_{ij}$ is set to $p$, where $p$ is the index of the **first** vector in the test sequence $s_i$ after which fault $f_j$ is detected. The total length of the original test sequence is $\sum_{i=1}^{m} \max(F_{(i)})$, where $\max(F_{(i)})$ denotes the maximum element of the $i$-th row of $F$.

Here, we can safely assume that $\max(F_{(i)})$ is equal to $n_i$ for all $i = 1 \dots m$. Otherwise, we can simply omit the last $n_i - \max(F_{(i)})$ vectors of the sequence $s_i$, since they do not detect any additional faults and thus they are redundant.

*Example:* Assume that we are given faults $f_1, f_2, f_3, f_4$ and 3 test sequences $s_1, s_2, s_3$. Test sequence $s_1$ is composed of 3 vectors; after its first vector is applied fault $f_1$ is detected, while after all three vectors are applied fault $f_3$ is also detected. Similarly, test sequence $s_2$ is composed of 5 vectors; after the first two vectors are applied fault $f_1$ is detected, while after all five vectors are applied fault $f_2$ is also detected. Finally, test sequence $s_3$ is composed of 4 vectors;


1063-6404/03 $ 17.00 © 2003 IEEE

after its first vector is applied fault $f_3$ is detected, after the first three vectors are applied fault $f_2$ is also detected, and after all of its four vectors are applied fault $f_4$ is also detected. Thus,

$$F = \begin{bmatrix} 1 & 0 & 3 & 0 \\ 2 & 5 & 0 & 0 \\ 0 & 3 & 1 & 4 \end{bmatrix}$$

Note that the total length of the original test set is 12 vectors. Also, $n_1 = 3$, $n_2 = 5$ and $n_3 = 4$; obviously, none of the 3 sequences contains any redundant vectors.

The objective of test compaction is to find the minimum number of vectors that detect all faults. We emphasize that the objective is two-fold: find and keep minimal subsequences of the test sequences $s_1, \ldots, s_m$ such that all faults are detected **and** the total length of these subsequences is minimal. To this end, some test sequences may be eliminated, while other test sequences may be pruned. In the above example, we could include all vectors of $s_3$ (a total of 4 vectors, detecting faults $f_2, f_3, f_4$) and the first vector of $s_1$ (a total of 1 vector, detecting fault $f_1$). Thus, test sequence $s_2$ would be eliminated, while test sequence $s_1$ would be pruned from 3 to 1 vectors, and all faults would be detected by a compacted test set of 5 vectors.

## 4. Integer Program Formulation

In this section we demonstrate how to model this problem as an Integer Program; namely, an optimization problem where the constraints and the optimization function are linear inequalities on a given set of *integer* variables.

It is easy to see that exactly solving this problem is NP-hard (reduction from Min-Cover); in [1] the authors employ a Genetic Algorithm to approximately solve it. The main disadvantage of this approach – as is often the case with genetic algorithms – is the lack of a provably good approximation guarantee; more specifically, there is no way to know how close the obtained solution is to the optimal one or provide any guarantees for the running time/approximation accuracy of the given algorithm. Our work bridges this gap by formulating the problem as an Integer Program and then showing that provably accurate solutions may be identified. Integer Programming is well known to be NP-hard, but trivial lower bounds on the optimization function exist. In particular, the optimal value of the Linear Programming relaxation of the Integer Program provides such a lower bound. We present a simple algorithm that *experimentally* almost always achieves the lower bound implied by the Linear Programming relaxation; we also present a theoretical bound for our approach, which is generally more pessimistic. The algorithm actually computes the solution to the Linear Programming relaxation of the Integer Program and then uses Randomized Rounding [3] to create an integer solution; the running time of our algorithm is polynomial in $\sum_{i=1}^{m} n_i$ and it is dominated by the time needed to solve a Linear Program of the same dimensions as the Integer Program.

Our approach starts by creating a new matrix $A$ from $F$. Given $F$ (a matrix of **sequences vs. faults**), we create $A$ (a matrix of **subsequences vs. faults**). Every sequence $s_i$ of length $n_i$ gives rise to $n_i$ subsequences, denoted by $\{A_\ell^{s_i}\}_{\ell=1}^{n_i}$; we will assume that $n_i = \max(F_{(i)})$ for all $i = 1 \ldots m$. Each subsequence $A_\ell^{s_i}$ contains the first $\ell$ vectors of the sequence $s_i$. Each row of $A$ corresponds to one of the $A_\ell^{s_i}$; more specifically, the $j$-th element of that row ($j = 1 \ldots k$, where $k$ is the total number of faults) is set to 1 if and only if the $j$-th fault is detected by the subsequence $A_\ell^{s_i}$. Let $N = (\sum_{i=1}^{m} n_i)$; $A$ is an $N \times k$ matrix. We should note here that $A$ is a rather sparse matrix, thus it may be stored efficiently. The following example explains how $A$ is created from $F$:

$$A = \left[\begin{array}{l|cccc l}
 & F_1 & F_2 & F_3 & F_4 & \\
\text{After } v_1 & 1 & 0 & 0 & 0 & \\
\text{After } v_2 & 1 & 0 & 0 & 0 & \text{Sequence } S_1 \text{ (3 vectors)} \\
\text{After } v_3 & 1 & 0 & 1 & 0 & \\
\text{After } v_1 & 0 & 0 & 0 & 0 & \\
\text{After } v_2 & 1 & 0 & 0 & 0 & \\
\text{After } v_3 & 1 & 0 & 0 & 0 & \text{Sequence } S_2 \text{ (5 vectors)} \\
\text{After } v_4 & 1 & 0 & 0 & 0 & \\
\text{After } v_5 & 1 & 1 & 0 & 0 & \\
\text{After } v_1 & 0 & 0 & 1 & 0 & \\
\text{After } v_2 & 0 & 0 & 1 & 0 & \text{Sequence } S_3 \text{ (4 vectors)} \\
\text{After } v_3 & 0 & 1 & 1 & 0 & \\
\text{After } v_4 & 0 & 1 & 1 & 1 &
\end{array}\right]$$

Note that the first 3 rows correspond to the first test sequence, the next 5 rows correspond to the second test sequence and the last 4 rows correspond to the third test sequence.

We are now ready to express our problem as an Integer Program; associate a 0-1 integer variable $x_\ell^{s_i}$ (for all $\ell$ and $i$) to every row of the matrix $A$. Here $x_\ell^{s_i}$ denotes whether the subsequence corresponding to that particular row of $A$ will be kept in the compacted test set. There are two restrictions: first, we must detect all faults that were detected by the original, non-compacted sequences. Note that even though we are given information on faults $f_1 \ldots f_k$ it may be the case that only a subset of these faults is detected by the sequences $s_1 \ldots s_m$. We will denote by $b$ a $k$-vector of 0s and 1s; 1 denotes that the corresponding fault is detected by one of the sequences $s_1 \ldots s_m$. We can create $b$ in one pass through the matrix $A$ by examining which faults are detected by the given test sequences.

Let $x$ denote the $N$-vector of all the variables $x_\ell^{s_i}$. In order to guarantee that our compacted test set has the same fault coverage as the original, non-compacted one, the following constraint must be satisfied:

$$A^T x \geq b \tag{1}$$

Proceedings of the 21st International Conference on Computer Design (ICCD'03)
1063-6404/03 $ 17.00 © 2003 IEEE
IEEE COMPUTER SOCIETY

A moment's thought reveals another set of more subtle constraints: for each $i = 1 \ldots m$, at most one of the $\{x_\ell^{s_i}\}_{\ell=1}^{n_i}$ will be set to 1! This essentially means that there is no reason to keep two different subsequences of the same sequence; we could only keep the longer one without any loss in fault coverage. Thus,

$$\sum_{\ell=1}^{n_i} x_\ell^{s_i} \leq 1, \; \forall i = 1 \ldots m \tag{2}$$

Given the above restriction, the optimization function is now straightforward: for each subsequence that we decide to keep, we will be penalized by the number of vectors in the subsequence. Let $c$ be a $(\sum_{i=1}^{m} n_i)$-vector, denoting the number of vectors in the corresponding row of $A$. Thus, we seek to minimize

$$c^T x \tag{3}$$

where $c$ is the vector of the costs that are associated with each subsequence.

*Example:* In our example, $b = (1\ 1\ 1\ 1)^T$ since all faults are detected by the original sequences. If we denote our integer variables (12 in this case) by $x_1^{s_1} \ldots x_3^{s_1}, x_1^{s_2} \ldots x_5^{s_2}, x_1^{s_3} \ldots x_4^{s_3}$, the restrictions implied by equation 2 are:

$$\begin{aligned} x_1^{s_1} + x_2^{s_1} + x_3^{s_1} &\leq 1 \\ x_1^{s_2} + x_2^{s_2} + x_3^{s_2} + x_4^{s_2} + x_5^{s_2} &\leq 1 \\ x_1^{s_3} + x_2^{s_3} + x_3^{s_3} + x_4^{s_3} &\leq 1 \end{aligned}$$

Finally the cost vector is $c = (1\,2\,3\,1\,2\,3\,4\,1\,2\,3\,4\,5)$.

More generally, given the above definitions, our Integer Program is:

$$\min\ c^T x \tag{4}$$
$$A^T x \geq b \tag{5}$$
$$\sum_{\ell=1}^{n_i} x_\ell^{s_i} \leq 1, \; \forall\, i = 1 \ldots m \tag{6}$$
$$0 \leq x_\ell^{s_i} \leq 1 \quad , \quad \forall\, \ell = 1 \ldots n_i, i = 1 \ldots m \tag{7}$$
$$x_\ell^{s_i} \text{ integers} \quad , \quad \forall\, \ell = 1 \ldots n_i, i = 1 \ldots m \tag{8}$$

A final note: in order to diminish the size of the integer program, one might remove rows of $A$ that are identical and only keep the one with the smallest cost. Such rows essentially correspond to subsequences of different lengths that detect the same faults; thus only the shortest of these subsequences should be kept.

## 5. Proposed Solution

We now employ a two-step approach in order to approximately solve the above Integer program. The main tool is a technique called *Randomized Rounding* [3]. The idea of Randomized Rounding is simple: solve the Linear Programming relaxation of the Integer Program and round the resulting *real* values *probabilistically*, thus forcing them to integers.

More specifically, the Linear Programming relaxation of the Integer Program of the previous section is

$$\min\ c^T x \tag{9}$$
$$A^T x \geq b \tag{10}$$
$$\sum_{\ell=1}^{n_i} x_\ell^{s_i} \leq 1, \; \forall\, i = 1 \ldots m \tag{11}$$
$$0 \leq x_\ell^{s_i} \leq 1 \quad , \quad \forall\, \ell = 1 \ldots n_i, i = 1 \ldots m \tag{12}$$

which is simply the Integer Program after removing the constraints of equation (8).

**Step 1:** Let $\tilde{x}$ denote the solution of the Linear Programming relaxation; set

$$x_\ell^{s_i} = \begin{cases} 1 & \text{, with probability } \tilde{x}_\ell^{s_i} + \delta \\ 0 & \text{, otherwise} \end{cases} \tag{13}$$

where $\delta = \sqrt{\ln(40N)}/\sqrt{2N}$. The vector $x$ is our – approximate – solution to the integer program; obviously $x$ is a 0-1 integer vector. In Theorem 1 we will argue that, with high probability, $x$ satisfies the constraints of equation (5) and achieves a bounded penalty in the cost function.

**Step 2:** In order to satisfy the constraints of equation (6), we employ the following simple algorithm for all $i = 1 \ldots m$: if more than one $x_\ell^{s_i}$, $\ell = 1 \ldots n_i$ are set to one, only keep the one with the highest cost and set the rest to zero. Thus, we satisfy the constraints without compromising the fault coverage of the compacted test set while, at the same time, we decrease the cost of the final solution.

Prior to stating Theorem 1, we note that the cost of the optimal solution to the Linear Program is necessarily less than or equal to the cost of any feasible solution to the integer program; equivalently,

$$c^T \tilde{x} \leq c^T x$$

for all possible 0-1 vectors $x$. As we shall see in our experiments, the randomized rounding technique identifies solutions that are essentially optimal (their cost is almost equal to $c^T \tilde{x}$). The proof of our main theorem follows the lines of [3] and argues that our 2-step algorithm identifies accurate solutions with high probability. We skip the rather folklore proof of the theorem for the sake of brevity; the proof makes heavy use of the well-known Chernoff-Hoeffding bounds [32].

**Theorem 1** *If $\tilde{x}$ is the solution to the Linear Programming relaxation and $x$ is the integer solution that we obtain using randomized rounding and elimination of redundant sequences, with probability at least* $0.95$, *1.* $c^T x \leq c^T \tilde{x} + 2^{-1/2}(M+1)\sqrt{N \ln(40N)}$

*2. The constraints of equation (5) are satisfied, thus $x$ is a feasible solution for our Integer Program.*

*In the above, $M = \max_i(n_i)$ (notice that $M = O(1)$). Also, by construction, the constraints of equations (6), (7) and (8) are satisfied.*

Finally, we briefly comment on the running time of our approach, which is dominated by the time required to solve the linear programming relaxation. The linear program may be solved in $poly(N+k)$ time using the Ellipsoid algorithm [33] or Interior Point methods [34]. In practice, there are many software packages (usually commercial) that efficiently solve Linear Programs with a very large number of constraints and variables. The randomized rounding step is easy to implement in $O(N)$ time.

## 6. Experimental Results

In order to evaluate the proposed methodology we repeat the experiment described in [1], wherein the authors generated sets of independent test sequences for the ISCAS89 [4] benchmark circuits using three different ATPG tools, GATTO, HITEC, and SYMBAT. Details and the resulting fault detection matrices are available at [35]. These matrices are the starting point for our experiments. Test sequences are extended into subsequences, the proposed method is applied and results are reported in Figures (1)-(3)[1].

The number of test sequences and total vectors in the original test set before compaction are reported in columns 2 and 3. The number of test sequences and total vectors in the compacted test set yielded by the proposed method are reported in columns 4 and 5. The difference between the number of vectors in the identified solution and the theoretical lower bound given by the Linear Program solution is reported in column 6. Column 7 indicates the size of the compacted test set as a percentage of the size of the original test set. Finally, column 8 indicates the test compaction efficiency of the Genetic Algorithms method proposed in [1].

The most important observation is that our approach almost always identifies the optimal solution, despite the rather pessimistic prediction of Theorem 1. As shown in the tables, the distance from the theoretical lower bound is 0 for most circuits. The same observation applies for the results of the Genetic Algorithm described in [1]. One can also observe that, for some circuits, out method achieves better compaction ratio over [1] (i.e. GATTO test set for S3271, HITEC test sets for S1269 and S3271).

The actual running times of our approach (for the HITEC and GATTO test sets) are reported in Figure (4). We caution the reader, however, that comparing these times to those reported in previous work [1, 2] would be quite misleading: our algorithm is implemented in MatLab – a slow, interpreted language – while previous algorithms are implemented in C; additionally, the various approaches are implemented and examined on different platforms. More importantly, we emphasize that the objective of this work was not to develop a worst-case exponential time heuristic that yields fast running times for some benchmark instances of the problem at hand. Rather, the goal of our paper is to provide a **polynomial time approximation algorithm** with provable guarantees regarding loss of optimality. In this respect, we feel that the proposed approach, combining an integer programming formulation and a randomized rounding solution, may prove fruitful in tackling various other problems in the area of test.

| Circuit | Original Test Set | | Compacted Test Set | | Distance From Lower Bound | Proposed Method | GA [1] Method |
|---|---|---|---|---|---|---|---|
| | # Seq | # Vec | # Seq | # Vec | | % Red | % Red |
| **S208** | 36 | 1096 | 6 | 347 | 0 | 31.66 | 31.66 |
| **S298** | 24 | 302 | 11 | 141 | 0 | 46.69 | 46.69 |
| **S344** | 19 | 141 | 10 | 66 | 0 | 46.81 | 46.81 |
| **S349** | 19 | 144 | 11 | 84 | 0 | 58.33 | 58.33 |
| **S382** | 17 | 840 | 7 | 485 | 0 | 57.74 | 57.74 |
| **S386** | 38 | 418 | 15 | 221 | 0 | 52.87 | 52.87 |
| **S400** | 16 | 916 | 7 | 502 | 0 | 54.08 | 54.08 |
| **S420** | 33 | 797 | 8 | 333 | 1 | 41.78 | 41.78 |
| **S444** | 22 | 1434 | 9 | 788 | 0 | 54.95 | 54.95 |
| **S499** | 29 | 465 | 9 | 192 | 0 | 41.29 | 41.29 |
| **S510** | 37 | 989 | 7 | 237 | 0 | 23.96 | 23.96 |
| **S526** | 18 | 1050 | 9 | 769 | 0 | 73.24 | 73.24 |
| **S526n** | 16 | 862 | 6 | 523 | 0 | 60.67 | 60.67 |
| **S641** | 48 | 395 | 24 | 221 | 0 | 55.95 | 55.95 |
| **S713** | 55 | 557 | 23 | 250 | 0 | 44.88 | 44.88 |
| **S820** | 38 | 669 | 14 | 347 | 0 | 51.87 | 51.87 |
| **S832** | 33 | 425 | 10 | 196 | 0 | 46.12 | 46.12 |
| **S838** | 37 | 1323 | 12 | 476 | 3 | 35.98 | 35.75 |
| **S938** | 37 | 1323 | 11 | 473 | 0 | 35.75 | 35.75 |
| **S953** | 75 | 1099 | 32 | 539 | 0 | 49.04 | 49.04 |
| **S967** | 72 | 1223 | 31 | 660 | 1 | 53.96 | 54.70 |
| **S991** | 20 | 448 | 9 | 365 | 0 | 81.47 | 81.47 |
| **S1196** | 133 | 1805 | 74 | 1124 | 0 | 62.27 | 62.66 |
| **S1238** | 123 | 1554 | 74 | 1004 | 0 | 64.61 | 64.80 |
| **S1269** | 52 | 450 | 29 | 245 | 0 | 54.44 | 54.44 |
| **S1423** | 107 | 2691 | 28 | 1279 | 0 | 47.53 | 47.71 |
| **S1488** | 65 | 1824 | 19 | 946 | 0 | 51.86 | 51.86 |
| **S1494** | 62 | 1244 | 19 | 652 | 0 | 52.41 | 52.41 |
| **S1512** | 52 | 772 | 14 | 289 | 0 | 37.44 | 37.44 |
| **S3271** | 132 | 2529 | 50 | 1178 | 0 | 46.58 | 60.58 |
| **S3384** | 58 | 888 | 22 | 410 | 0 | 46.17 | 46.17 |
| **S4863** | 112 | 1533 | 42 | 790 | 8 | 50.88 | 48.66 |
| **S5378** | 71 | 919 | 42 | 493 | 0 | 53.65 | 53.65 |
| **S6669** | 64 | 592 | 36 | 301 | 0 | 50.84 | 51.18 |
| **S13207** | 34 | 544 | 9 | 187 | 0 | 34.38 | 34.38 |
| **S15850** | 10 | 153 | 3 | 91 | 0 | 59.48 | 59.48 |

**Figure 1. Results for GATTO Test Sets**

[1]A "*" in the table of Figure (2) indicates a minor discrepancy between the numbers reported in [1] and the size of the tables available from [35].

| Circuit | Original Test Set | | Compacted Test Set | | Distance From Lower Bound | Proposed Method | GA [1] Method |
|---|---|---|---|---|---|---|---|
| | # Seq | # Vec | # Seq | # Vec | | % Red | % Red |
| **S208** | 44 | 739 | 10 | 292 | 1 | 39.51 | 39.27 |
| **S298** | 20 | 188 | 7 | 116 | 0 | 61.70 | 54.38* |
| **S344** | 11 | 61 | 6 | 45 | 0 | 73.77 | 75.41 |
| **S349** | 16 | 84 | 9 | 63 | 0 | 75.00 | 76.19 |
| **S382** | 16 | 358 | 2 | 155 | 0 | 43.30 | 43.45 |
| **S386** | 58 | 258 | 31 | 162 | 0 | 62.79 | 62.79 |
| **S400** | 16 | 354 | 2 | 155 | 0 | 43.75 | 43.70 |
| **S420** | 52 | 786 | 10 | 274 | 0 | 34.86 | 34.90 |
| **S444** | 18 | 305 | 2 | 204 | 0 | 66.89 | 66.56* |
| **S510** | 38 | 845 | 27 | 623 | 0 | 73.73 | 73.67* |
| **S526** | 18 | 260 | 2 | 172 | 0 | 66.15 | 66.54 |
| **S526n** | 17 | 257 | 2 | 169 | 0 | 65.76 | 65.23* |
| **S713** | 74 | 270 | 35 | 169 | 1 | 62.59 | 63.33 |
| **S820** | 121 | 1170 | 62 | 671 | 0 | 57.35 | 57.44 |
| **S832** | 112 | 1058 | 60 | 617 | 0 | 58.32 | 58.51 |
| **S838** | 52 | 671 | 12 | 310 | 1 | 46.20 | 45.93 |
| **S938** | 52 | 671 | 12 | 310 | 1 | 46.20 | 45.93 |
| **S953** | 111 | 825 | 38 | 404 | 0 | 48.97 | 48.97 |
| **S967** | 120 | 831 | 38 | 407 | 0 | 48.98 | 48.98 |
| **S991** | 50 | 83 | 25 | 46 | 0 | 55.42 | 55.42 |
| **S1196** | 189 | 509 | 110 | 339 | 2 | 66.60 | 66.60 |
| **S1238** | 191 | 513 | 110 | 334 | 2 | 65.11 | 64.72 |
| **S1269** | 67 | 255 | 26 | 136 | 0 | 53.33 | 61.18 |
| **S1423** | 50 | 282 | 16 | 187 | 0 | 66.31 | 66.43 |
| **S1488** | 24 | 69 | 16 | 56 | 0 | 81.16 | 81.16 |
| **S1494** | 60 | 523 | 43 | 418 | 0 | 79.92 | 81.02 |
| **S1512** | 60 | 282 | 14 | 117 | 0 | 41.49 | 41.70 |
| **S3271** | 61 | 1158 | 19 | 489 | 0 | 42.23 | 49.70 |
| **S3384** | 18 | 212 | 8 | 164 | 0 | 77.36 | 77.83 |
| **S4863** | 106 | 373 | 57 | 256 | 0 | 68.63 | 68.35* |
| **S5378** | 95 | 250 | 50 | 153 | 1 | 61.20 | 60.80 |
| **S6669** | 68 | 466 | 23 | 259 | 0 | 55.58 | 55.58 |
| **S9234** | 7 | 19 | 2 | 9 | 0 | 47.37 | 52.63 |
| **S13207** | 15 | 97 | 6 | 57 | 0 | 58.76 | 59.79 |
| **S15850** | 15 | 39 | 4 | 14 | 0 | 35.90 | 38.46 |
| **S38417** | 281 | 806 | 14 | 131 | 0 | 16.25 | 16.38 |
| **S38584** | 48 | 509 | 30 | 435 | 0 | 85.46 | 85.46 |

**Figure 2. Results for HITEC Test Sets**

## 7. Conclusion

In this paper we demonstrate the formulation of static compaction of independent test sequences as an Integer Program. Solving the Linear Program relaxation of this Integer Program provides a lower bound for the optimal solution, i.e. the minimal set of test sequences. Subsequently, Randomized Rounding of the optimal point of the Linear Program is employed to obtain a solution for the Integer Program. The key advantage of this approach is that it provides a polynomial time approximation algorithm as well as an indication of proximity to the optimal solution and, thus, a measure for evaluating compaction efficiency. As indicated by experimental results, the proposed method is efficiently identifying almost optimal solutions.

| Circuit | Original Test Set | | Compacted Test Set | | Distance From Lower Bound | Proposed Method | GA [1] Method |
|---|---|---|---|---|---|---|---|
| | # Seq | # Vec | # Seq | # Vec | | % Red | % Red |
| **S208** | 64 | 2049 | 35 | 1356 | 2 | 66.18 | 66.08 |
| **S298** | 34 | 344 | 17 | 193 | 0 | 56.10 | 56.10 |
| **S344** | 47 | 187 | 23 | 111 | 0 | 59.36 | 59.36 |
| **S349** | 46 | 184 | 24 | 113 | 0 | 61.41 | 61.41 |
| **S382** | 59 | 2580 | 25 | 1318 | 0 | 51.09 | 51.09 |
| **S386** | 76 | 340 | 44 | 206 | 2 | 60.59 | 60.00 |
| **S400** | 59 | 2538 | 26 | 1352 | 0 | 53.27 | 53.27 |
| **S420** | 49 | 9377 | 26 | 8750 | 2 | 93.31 | 93.29 |
| **S444** | 39 | 2034 | 24 | 1262 | 0 | 62.05 | 62.05 |
| **S499** | 33 | 418 | 23 | 298 | 0 | 71.29 | 71.29 |
| **S510** | 59 | 1066 | 41 | 810 | 0 | 75.98 | 75.98 |
| **S526** | 74 | 3607 | 31 | 1679 | 0 | 46.55 | 46.55 |
| **S526n** | 73 | 3573 | 31 | 1679 | 0 | 46.99 | 46.99 |
| **S713** | 164 | 538 | 100 | 356 | 0 | 66.17 | 66.17 |
| **S820** | 202 | 1425 | 108 | 777 | 0 | 54.53 | 54.74 |
| **S832** | 195 | 1370 | 107 | 768 | 0 | 56.06 | 56.06 |
| **S953** | 155 | 1261 | 86 | 763 | 2 | 60.51 | 60.35 |
| **S967** | 162 | 1322 | 88 | 795 | 0 | 60.14 | 60.14 |
| **S1196** | 297 | 613 | 199 | 378 | 3 | 61.66 | 61.34 |
| **S1238** | 300 | 619 | 204 | 386 | 3 | 62.36 | 62.20 |
| **S1488** | 157 | 1709 | 101 | 1118 | 8 | 65.42 | 64.95 |
| **S1494** | 160 | 1787 | 99 | 1140 | 0 | 63.79 | 63.79 |

**Figure 3. Results for SYMBAT Test Sets**

## References


[1] F. Corno, P. Prinetto, M. Rebaudegno, and M. Sonza Reorda, "New static compacion techniques of test sequences for sequential circuits," in *European Design and Test Conference*, 1997, pp. 37–43.

[2] J. Raik, A. Jutman, and R. Ubar, "Fast static compaction of tests composed of independent sequences: Basic properties and comparison of methods," *International Conference on Electronics, Circuits, and Systems*, pp. 445–448, 2002.

[3] P. Raghavan and C. Thompson, "Randomized rounding: A technique for provably good algorithms and algorithmic proofs," *Combinatorica*, vol. 7, no. 4, pp. 365–374, 1987.

[4] "ISCAS'89 benchmark circuits information," Available from `http://www.cbl.ncsu.edu`.

[5] M. H. Schulz, E. Trischler, and T. M. Sarfert, "SOCRATES: a highly efficient automaitc test pattern generation system," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 7, no. 1, pp. 126–137, 1988.

[6] J. A. Waicukauski, P.A. Shupe, D. J. Giramma, and A. Matin, "ATPG for ultra-large structured designs," in *International Test Conference*, 1990, pp. 44–51.

[7] M. Abramovici, M. A. Breuer, and A. D. Friedman, *Digital Systems Testing and Testable Design*, IEEE Press, 1990.

[8] J-S. Chang and C-S. Lin, "Test set compaction for combinational circuits," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 14, no. 11, pp. 1370–1378, 1995.

[9] Ilker Hamzaoglu and Janak H. Patel, "Test set compaction algorithms for combinational circuits," in *International Conference on Computer-Aided Design*, 1998, pp. 283–289.



Proceedings of the 21st International Conference on Computer Design (ICCD'03)
1063-6404/03 $ 17.00 © 2003 IEEE



IEEE COMPUTER SOCIETY

| Circuit | HITEC (in secs) | GATTO (in secs) | Circuit | HITEC (in secs) | GATTO (in secs) |
|---|---|---|---|---|---|
| **S208** | 5.16 | 6.7 | **S991** | 14.99 | 1054.53 |
| **S298** | 9.28 | 11.3 | **S1196** | 371.32 | 197.6 |
| **S344** | 10.69 | 14.6 | **S1238** | 326.18 | 1830.3 |
| **S349** | 11.04 | 14.4 | **S1269** | 153.69 | 106.2 |
| **S382** | 6.80 | 17.7 | **S1423** | 35.52 | 1063.91 |
| **S386** | 7.86 | 20.5 | **S1488** | 11.47 | 1246.7 |
| **S400** | 7.62 | 10.7 | **S1494** | 150.33 | 489.7 |
| **S420** | 9.47 | 40.9 | **S1512** | 18.09 | 90.3 |
| **S444** | 8.82 | 41.32 | **S3271** | 1024.40 | 2881.18 |
| **S510** | 137.28 | 30.6 | **S3384** | 119.00 | 741.3 |
| **S526** | 8.59 | 24.1 | **S4863** | 220.36 | 51.4 |
| **S526n** | 7.79 | 945.4 | **S5378** | 146.51 | 36.8 |
| **S713** | 26.48 | 1830.3 | **S6669** | 948.44 | 43.7 |
| **S820** | 182.09 | 106.2 | **S9234** | 15.78 | 397.7 |
| **S832** | 157.27 | 40.1 | **S13207** | 29.71 | 256.1 |
| **S838** | 20.44 | 32.2 | **S15850** | 5.29 | 10.1 |
| **S938** | 20.10 | 8.5 | **S38417** | 1601.44 | n/a |
| **S953** | 188.08 | 32.2 | **S38584** | 1779.58 | n/a |
| **S967** | 177.15 | 1206.67 | | | |

**Figure 4. Running times for HITEC/GATTO Test Sets**


[10] P. Goel and B. C. Rosales, "Test generation and dynamic compaction of tests," in *Digest of Papers, Test Conference*, 1979, pp. 189–192.

[11] S. B. Akers, C. Joseph, and B. Krishnamurthy, "On the role of independent fault sets in the generation of minimal test sets," in *International Test Conference*, 1987, pp. 1100–1107.

[12] G-J. Tromp, "Minimal test sets for combinational logic," in *International Test Conference*, 1991, pp. 204–209.

[13] I. Pomeranz, L. N. Reddy, and S. M. Reddy, "COMPACTEST: a method to generate compact test sets for combinational circuits," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 12, no. 7, pp. 1040–1049, 1993.

[14] 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 Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 14, no. 12, pp. 1496–1504, 1992.

[15] J. P. Marques Silva, "Integer programming models for optimization problems in test generation," in *Asia South Pacific Design Automation Conference*, 1998, pp. 481–487.

[16] T. M. Niermann, R. K. Roy, J. H. Patel, and J. A. Abraham, "Test compaction for sequential circuits," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 11, no. 2, pp. 260–267, 1992.

[17] M. H. Hsiao, E. M. Rudnick, and J. H. Patel, "Fast static compaction algorithms for sequential circuit test vectors," *IEEE Transactions on Computers*, vol. 48, no. 3, pp. 311–322, 1999.

[18] M. Hsiao and S. T. Chakradhar, "State relaxation based subsequence removal for fast static test compaction," in *Design Automation and Test in Europe Conference*, 1998, pp. 577–582.

[19] I. Pomeranz, S. M. Reddy, and R. Guo, "Static test compaction for synchronous sequential circuits based on vector restoration," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 18, no. 7, pp. 1040–1049, 1999.

[20] S. K. Bommu, S. T. Chakradhar, and K. B. Doreswamy, "Static test sequence compaction based on segment reordering and accelerated vector restoration," in *International Test Conference*, 1998, pp. 954–961.

[21] X. Lin, W-T. Cheng, I. Pomeranz, and S. M. Reddy, "SIFAR: static test compaction for synchronous sequential circuits based on single fault restoration," in *VLSI Test Symposium*, 2000, pp. 205–212.

[22] I. Pomeranz and S. M. Reddy, "Procedures for static compaction of test sequences for synchronous sequential circuits," *IEEE Transactions on Computers*, vol. 49, no. 6, pp. 596–607, 2000.

[23] I. Pomeranz, S. M. Reddy, and R. Guo, "Vector replacement to improve static-test compaction for synchronous sequential circuits," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 20, no. 2, pp. 336–342, 2001.

[24] I. Pomeranz and S. M. Reddy, "Forward-looking fault simulation for improved static compaction," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 20, no. 10, pp. 1262–1265, 2001.

[25] I. Pomeranz and S. M. Reddy, "On generating compact test sequences for synchronous sequential circuits," in *European Design Automation Conference*, 1995, pp. 105–110.

[26] A. Ragunathan and S. T. Chakradhar, "Acceleration techniques for dynamic vector compaction," in *International Conference on Computer-Aided Design*, 1995, pp. 310–317.

[27] T. J. Lambert and K. K. Saluja, "Methods for dynamic vector compaction in sequential test generation," in *International Conference on VLSI Design*, 1996, pp. 166–169.

[28] I. Pomeranz and S. M. Reddy, "Dynamic test compaction for synchronous sequential circuits using static compaction techniques," in *Fault Tolerant Computing Symposium*, 1996, pp. 53–61.

[29] R. Bevacqua, L. Guerazzi, F. Ferrandi, and F. Fummi, "Implicit test sequences compaction for decreasing test application cost," in *International Conference on Computer Design*, 1996, pp. 384–389.

[30] E. M. Rudnick and J. H. Patel, "Efficient techniques for dynamic test sequence compaction," *IEEE Transactions on Computers*, vol. 48, no. 3, pp. 323–330, 1999.

[31] R. Guo, S. M. Reddy, and I. Pomeranz, "PROPTEST: a property based test pattern generator for sequential circuits using test compaction," in *Design Automation Conference*, 1999, pp. 653–659.

[32] W. Hoeffding, "Probability inequalities for sums of bounded random variables," *American Statistical Association Journal*, pp. 13–30, March 1962.

[33] L.G. Khachiyan, "A polynomial-time algorithm for linear programming," *Soviet Math. Dokl.*, vol. 20, no. 1, pp. 191–194, 1979.

[34] N. Karmarkar, "A new polynomial-time algorithm for linear programming," *Proceedings of the Annual Symposium on Theory of Computing*, pp. 302–311, 1984.

[35] "Test sequence tables used in [1]," Available from `http://www.cad.polito.it/tools`.




1063-6404/03 $ 17.00 © 2003 IEEE