An Optimal Test Compression Procedure for Combinational circuits

Dorit S. Hochbaum*
Department of Industrial Engineering & Operations Research
Etcheverry Hall
University of California
Berkeley Ca. 94720-1777.
email: dorit@hochbaum.berkeley.edu

March 14, 1996

Abstract

The problem of optimal test compression is to derive, from a given set of test vectors, a smallest possible subset of test vectors that still test for the same collection of faults. This achieves optimal compression and largest reduction possible in test time relative to the original set of test vectors. We present a new approach based on the modeling of the problem as the Set Cover problem. Additionally, the approach implies an ordering of the faults according to the difficulty of covering them with the given set of test vectors. As such it can be used to facilitate the finding of a solution to the ultimate smallest test-set - the compression of the set of all possible test vectors. Our approach highlights the potential usefulness of integer programming techniques in testing and design.

*This research has been supported in part by ONR grant N00014-91-J-1241 and by the Competitive Semiconductor Manufacturing project funded by the Sloan Foundation.

1 Introduction

The problem of reducing test time (or compaction/compression) is a highly important and cost-effective quality control tool. Yet with the advent of circuits of large sizes (half a million gates is the current range for very large size), and with future trends indicating further growth, the complexity involved in generating "good" tests is beyond reach. Even for relatively small circuits, all existing approaches are of heuristic-type, i.e. they do not guarantee a set of tests that is optimal in some sense. The attempt is to generate test vectors to as many potential faults as possible while generating as few test vectors as possible. The problem of generating the smallest test set for all faults is referred to hereafter as the *ultimate test-set problem.*

The difficulty of generating a test for all faults with the fewest test vectors is primarily that of computational complexity: one can enumerate all possible test vectors (for n inputs there are 2^n such vectors) and thus test all testable faults. Since this approach is intractable, the alternative is to use effective heuristic rules to attempt and achieve this goal by generating the vectors so that each tests as many faults as possible, or generating the vectors per faults to be tested using various permutations on the order of the faults. In this paper we focus on the goal of compressing a previously generated *given* set of test vectors while maintaining the set of faults tested. However, we also propose an approach for addressing the ultimate test-set problem.

There are several approaches for generating a compressed set of test vectors. One approach is to break down the process into two phases: in the first phase a collection of test vectors is generated using *Automatic Test Pattern generation* (ATPG). In the second phase some heuristic procedure is applied in order to reduce the number of test vectors. Another type of approach is a "dynamic" compression where the choice of the generated test vectors is integrated with the test compression heuristic. These two approaches are labeled as *post-generation* versus *compaction during test generation* by Chang and Lin [5].

Our post-generation compaction procedure is applicable to *any* input set of generated test vectors. This approach provides the flexibility of generating the initial test pattern without prescribing how this generation is to be attempted. Moreover, with the integer phase it guarantees the smallest possible set of test vectors that tests the same faults as the input set while forming a subset of the input set. The running time of the procedure implemented is fast enough to justify its use as a post processor of any generated test set.

We model the problem of optimal test compression as the *Set Cover* problem (see e.g. Garey and Johnson [9]). The drawback of the Set Cover model is that the problem is NP-hard. We develop an implementation that is based on the linear programming relaxation of the problem, and is particularly suitable for dealing with very large test patterns. This implementation is emulating approaches used to solve the crew scheduling for the airlines. We discuss in Section 4 the relationship between the test compaction and the crew scheduling problem, and the features that are useful to adapt.

The linear programming relaxation is followed by a phase devised to find integer solutions using Branch and bound that is designed for the Set Cover problem. Although the compaction problem can be modeled as a Set Cover problem it also has some distinguishing features. Details on how these features can be exploited for more effective solutions are described in Section 5.

An important by-product of the linear programming solution is that it classifies faults according to the difficulty of testing them. This information is potentially useful in compaction during-test-generation procedures that repeatedly generate test vectors, guided by some ordering of the faults. For the benchmark circuits tested, it was deemed unnecessary to employ the full power of a Set Cover optimal algorithm in the integer phase. Instead, the linear programming solution gave in almost all cases the provably optimal solution, and in the other cases - solutions that are very close to the optimum.

The Set Cover for the compression problem is set up as follows: each column of the integer programming formulation corresponds to a test vector and each row corresponds to a circuit fault. If test vector i detects fault j, then the entry (i, j) is one; it is zero otherwise. The number of rows is polynomial in the size of the circuit. The number of columns is the number of test vectors in the given set.

For the more general ultimate test-set problem the objective is to find the smallest test set possible, not only with respect to a given set, but absolutely. To achieve such maximum compaction we should include all the possible input vectors (the number of which is exponential in number of inputs to the circuit) as columns. In a situation where the number of columns is exceedingly large, a technique called *column generation* is often useful [19]. Such a technique is useful in situations where the columns of a coefficient matrix are given implicitly and columns satisfying specific conditions can be efficiently generated on demand. The analogous situation in testing is that all potential test vectors are not listed in advance. Rather test vectors are generated for particular faults based on guideness on how difficult each fault is to test. This difficulty assessment derives from dual cost information, and will be discussed in Section 4. We point out similar technical structures that exist in the Set Cover problem corresponding to the airline crew scheduling problem, and propose how to exploit ideas developed in that context for more efficient compaction and for the ultimate test-set problem.

There are two sets of experiments reported in this paper, one is for the compaction of ATPG test vectors, and the other for the compaction of COMPACTEST vectors. For all the COMPACTEST experiments the LP relaxation solution was provably optimal. In most cases of the ATPG tests our compaction was either provably optimal or very close to the optimum compaction of the specific set of vectors. As a result we did not employ the full power of the Set Cover optimal algorithm. It will be interesting to test whether for substantially larger circuits, this pattern of having mostly integer values in the LP relaxation solution still applies.

The plan of the paper is as follows. Section 2 reviews highlights of the literature relevant to testing of combinational circuits. In Section 3, we describe the Set Cover problem and how solving it applies to the compaction problem. Section 4 discusses the type of algorithms applied to solve the crew scheduling problem, and the similar implementation for the test compression problem. We then discuss the distinguishing features of the compaction problem, and propose how to employ them in algorithms for solving the problem. Section 6 describes the empirical study on the benchmark circuits and finally we offer some concluding remarks.

2 Previous Work

The previous work on the problem of finding a compact set of test vectors for stuck-at faults in combinational logic circuits is described below.

The problem of finding a test for a given fault is NP-hard [15]. Yet as a result of extensive research done in this area, there are methods which have been shown to perform extremely well on the benchmark circuits. These methods include: D-Algorithm by Roth [23], PODEM by Goel [10], FAN by Fujiwara and Shimono [8], SOCRATES by Schulz et al. [24], TOPS by Kirkland and Mercer [14], Cheng and Aggarwal [6], Takamatsu and Kinoshita [25], FAST by Abramavoci et al. [1], ATWIG by Trischler [26], and Larrabee's [16]. Another important dimension of the problem is to find the smallest set of test vectors which tests for all the nonredundant faults in the circuit. The problem was left unaddressed for a long time because the problem of finding a test vector for a given fault was not well-solved. However, with more success in that area, many researchers in the last 10 years have moved on to tackling the test compression (or compaction) problem.

One of the first works in this field is by McDonald and Benmehrez [17]. The test patterns are generated based on the concept of multiple simultaneous path sensitization to reduce the number of patterns required. The strategy is used when coverage approaches 100%. First, separate observation paths are constructed for as many faults as possible. Simultaneous existence of other observation paths constrains lines of the circuit to assume certain values instead of don't cares as it would be the case when the fault is considered by itself. The faults whose observation paths are inconsistent with others are dropped. They use the notion of a control cube to represent variable signals (a notion identical to "don't cares"). The objective is to backtrace the variable signals without fixing them to 0 or 1.

Akers, Joseph and Krishnamurthy [2] take a more global approach. The role of independent fault sets in compact test sets is stressed. A fault set is called independent if no two faults in it can be detected by a single test vector. They provide an algorithm to generate maximal independent fault sets based on the ideas of fault dominance and fault equivalence. The independent sets form the basis for generating fault patterns. Faults are grouped/matched into sets which can be potentially tested by a single test pattern.

Schulz, Trischler and Sarfert [24] presented an efficient test pattern generator named SOCRATES. It was not their primary concern to have a compact set of test vectors. Nonetheless, they suggest the technique of *reverse order fault simulation* to compact the set of test vectors which forms the output of the main routine. Under this technique, the set of test vectors is applied one by one in reverse order to the fault list. Detected faults are removed from the list after every test vector. It is observed experimentally that often all the covered faults are detected using only a subset of the original set of test vectors. The intuitive reason for this phenomenon is simply that the test vectors which are further down the list detect faults which are most difficult to detect. Therefore, if we fault-simulate a test vector which is at the end of list first, it not only detects a "difficult" fault right away, it also detects many others by pure chance. This way difficult faults are out of the way early. It is a fairly effective technique for first attempt at compacting the set of test vectors. It is also indicative of the importance of the order of faults in the fault list.

Tromp [27] improved the approach of Akers et al. [2] in two ways. The independent fault set procedure is improved so that larger independent fault sets can be derived. A heuristic is used to find maximal independent fault sets. Next, Faults are matched into compatible fault sets using a bipartite matching algorithm one fault at a time. The implication procedure is also improved to enable more success in finding all compatible fault sets for a single test vector. The experimental results are provided for small benchmark circuits only.

Pomeranz, Reddy and Reddy [20] attack the problem in a three-pronged fashion. Their system is termed COMPACTEST. They first use a heuristic to order the faults in the fault list. It is based on the concept of a fanout free region (FFR) of a general combinational circuit. Maximal FFRs of the circuit are determined. For each FFR, a maximal independent fault set is found. The largest independent fault set thus found is placed on the top of fault list followed by the next largest set and so on. In the second phase, they use a dynamic compaction strategy termed *maximal compaction.* This forms the main heuristic underlying their system. Under this heuristic, a fault (*primary target*) is taken from the ordered fault list and a test vector is generated for it. Each specified input value of this test vector is considered separately. If the generated test vector is still a test for the fault when the input value is flipped, then the input value is unspecified. It is possible that the resulting test vector is no longer a test vector for the primary target. Secondary target faults are picked up from the fault list and removed from the list if a subset of the test vector tests for them. The process continues till either every fault is detected or declared redundant. They also enhance the test generation procedure by a technique termed *rotating backtrace.* For every gate, there is a flag that marks the next gate-input to be considered during backtrace. Whenever the backtrace procedure reaches the output of the gate with a value which can be set by setting one of the inputs of the gate to a controlling value, the input marked by the flag is selected, and the flag is incremented modulo the number of inputs to the gate, to mark the next input in the gate input list. The experimental results show that their heuristics outperform all pre-existing methods to compact test vectors.

Kajihara, Pomeranz, Kinoshita and Reddy [13] improve upon [20]. They add a procedure for computing independent fault sets which are used to select targets for pattern generation. They calculate lower bounds for the size of the smallest test set. Rotating backtrace is also enhanced. Experiment results are provided and for certain ISCAS'89 benchmark combinational circuits and fully-scanned sequential circuits.

Chang and Lin [5] introduce the concept of *forced -pair-merging* which attempts to modify vectors by adding X values (i.e. don't cares or unspecified values) if possible while maintaining the set of faults covered. As a result some pairs of vectors may become compatible and can be merged, resulting in a smaller set of test vectors. Their approach yields most of the best known tests to date. Yet, the running time required to achieve those tests is considerable - in the order of magnitude of thousands of secs.

Reddy, Pomeranz and Reddy [22] manipulate the don't cares in a vector so it is expanded with don't care values in order to achieve further compaction. While manipulated, some of the test vectors may become redundant, as all faults that they test are already tested in the "expanded" vectors. As a result the redundant vectors are removed and further compaction is achieved. This procedure is called ROTCO.

Ayari and Kaminska [3] present a new simple-to-implement approach for dynamic test vector compaction called COMPACT. After a test generation T_i, instead of selecting the next untested fault from the fault list and attempting test generation, COMPACT is used to compare other tests that have been generated already, and if T_i is compactable with any test T_j ($j < i$) (i.e., $T_i \cap T_j \neq \phi$), then T_j is replaced by $T_i \cap T_j$. If not, then T_i is not compactable. The intersection is performed bit-by-bit ($X \cap 1 = 1$, $X \cap 0 = 0$, $1 \cap 0 = \phi$, $0 \cap 0 = 0$). An efficient data-structure implementation is proposed. The experimental results are provided for benchmark circuits. The results are only a bit inferior to those in [13] but the procedure itself is considerably more efficient.

Unlike the approaches to date, ours can use the output of *any* procedure and reduce it to a minimum, while maintaining the same set of test faults. The output of our procedure can also be used by test generators in order to focus on a specific most difficult set of faults. We call the procedure used to generate the input - *the underlying test pattern generator.*

3 The Set Cover Model

The Set Cover approach is used to compact given test sets for combinational circuits. In the Set Cover problem, the goal is to identify the smallest collection of sets that cover a given (universal) set of elements. In the compression problem the elements are the faults to be covered by the given collection of test vectors. Suppose the circuit contains detected faults named $f_1, ..., f_m$ and the test set used as input for compaction consists of vectors $v_1, ..., v_n$. Each test vector v_j is associated with a subset of faults, S_j, $j = 1, \ldots, n$. The problem is to find the smallest subcollection that still covers faults $\{1, \ldots, m\}$.

A $m \times n$ detection matrix D is generated. The entry d_{ij} equals one if fault i is detected by test set j; otherwise it equals zero. The jth column of the matrix is then the 0-1 characteristic vector of the faults tested by v_j. A set cover problem is formulated as an integer programming problem:

$$\begin{array}{lllll} \min & \sum_j x_j & & & \\ & Dx & \geq & e & \\ & x_j & \in & \{0,1\} & j = 1, 2, ..., n \end{array}$$

There are a number of Integer Programming approaches that have been developed for solving the Set Cover problem. One important method is to solve the linear programming relaxation (LP-relaxation) of the problem. This amounts to relaxing (removing) the requirement that the variables are integer. We then get, in place of the constraints that state that the variables are binary, the constraints $1 \geq x_j \geq 0$, $j = 1, 2, .., n$. The calls for the LP relaxation are embedded in a branch and bound procedure for solving the problem.

The LP-relaxation of a minimization problem has the property that its optimal value is a lower bound on the value of the optimal integer solution to the problem. If the LP-relaxation solution happens to be integer, then it is also an optimal solution to the integer problem. For most of the benchmark circuits, it is observed experimentally that the LP-Relaxation gives us an optimal solution that is binary. Due to this observed feature, we do not need to resort to sophisticated (and potentially time consuming) alternative methods for solving the problem. We do elaborate however, in the next section, about these methods.

An easy heuristic for dealing with cases when the optimal LP-relaxation solution is not integer, is to use the rounding heuristic: If the optimal solution of resulting LP is x^*, then the output of the rounding heuristic is $\lceil x^* \rceil$ and thus obviously feasible. This heuristic was also proved to guarantee a bound on the worst case error ratio, [12]. Since there were no fractional valued variables, or very few of them, the introduction of this heuristic induced a very small deviation of the feasible solution derived from the optimal value.

The LP relaxation delivers, in addition to an optimal fractional solution, also a dual optimal solution. The interpretation of the dual costs corresponding to the constraints (i.e. individual faults) is the economic imputed cost of those faults which also corresponds to the difficulty of detecting the given faults with the current set of potential test vectors. The larger the dual cost, the more expensive is the fault to cover. As a result it would be particularly beneficial to generate alternative test vectors for the most expensive faults. Since most test generation procedures known are affected by the ordering of the faults, the ordering derived from the dual costs could serve as an excellent guide for generation of additional effective test vectors.

The integer phase for solving the Set Cover problem is based on a branch and bound procedure. The branching is determined by a heuristic approach to decide on which variables are most (or least) 'promising'. A variable is considered promising in a solution if either it covers the most rows, or if its reduced cost is smallest. there are either possible criteria as well, but as we see in the next section, the one based on reduced cost is likely to be most effective for the compaction problem.

4 Dealing with large test patterns -Similarities to the Crew Scheduling problem.

The airline crew scheduling leads to a number of large-scale optimization problems. One such problem has as inputs a planning period, crewbase locations, flight services with scheduled arrival/departure time and location. The objective is to assign crews to cover each flight to minimize the overall costs while meeting availability and work rule restrictions.

The problem is modeled as a minimum cost Set Cover problem with flight services as the set elements. A set is formed by a *pairing*. A pairing is a sequence of flights with the first flight beginning and the last flight ending at the same crewbase. Associated with each pairing are a variety of costs. The reader is referred to [4] for details.

The minimum cost Set Cover representing the crew scheduling problem is similar in structure to the ultimate test-set problem. There are numerous potential sets - pairings for the crew scheduling problem, and test vectors for the ultimate test-set problem. A flight leg can be covered by any of a large number of possible pairings. Analogously, a fault can be covered by any of a large number of test vectors. One difficulty is to determine for a given fault, whether additional test vectors should be generated, as not *all* test vectors could be enumerated due to the excessive time required to generate those. Similar structures imply that similar techniques might work well in both cases. On the other hand, there are some important differences too. In the case of crew scheduling, it is relatively straightforward to efficiently generate a column of required properties. It is not so in the case of test generation. The aspect of don't cares in test vectors also has no parallel in the context of crew scheduling. More details on making use of this feature are given in the next section.

The integer solution for the crew scheduling problem is typically determined on the basis of repeated calls to the LP-relaxation of the problem, embedded in a branch and bound procedure. The LP has a number of columns which is exponential in the number of flight legs, one for each possible pairing. For typical size problems, the number of all such pairings is too large to solve the LP directly. However, selective columns can be generated on demand using a specialized shortest path procedure which is relatively efficient. That is, for a given collection of flight legs the procedure generates a number of possible pairings that cover those legs and are potentially cost effective candidates. Various researchers at different airlines have successfully applied these ideas for crew scheduling (e.g., [18], [21], [7]).

A column generation procedure has been devised for solving linear programs with excessively large number of columns (or variables). Informally, the idea is to solve the problem first with only a small subset of the columns present. Then, one or more new columns are selected based on their "reduced cost". The reduced cost is a linear function of the original cost of the column adjusted for the value of the column's contribution to satisfying the constraints as is reflected by the dual variables.

In a column generation procedure the dual multipliers are used as the imputed costs of the individual flight legs. A collection of pairings that cover those flight legs is then computed. Among these the most attractive ones are added to the set of columns. The 'attractiveness' of a column (or pairing) is measured in terms of the cost of the pairing from the value of the covered flight legs is deducted. The more negative this figure, the more attractive the pairing. If all these are nonnegative, then the current LP solution is also optimal.

The analogous situation for ultimate test-set calls for solving the LP relaxation of the compaction problem, with an initial set of test vectors. New test vectors are then generated for the most expensive faults. The cost of each test vector is 1 and its net value is computed by subtracting the dual costs of the faults it covers. With those additional columns (their number is determined as a parameter) the compaction problem is resolved, and the procedure is repeated iteratively. This procedure is then embedded in a branch and bound procedure with level of sophistication that depends on the added value of the additional run-time required. One such branch and bound procedure is described in [11]. That procedure was proved to be most effective for problems of low density, where density is the number of 1s in the constraint matrix relative to the total number of entries in the constrain matrix. Since each test vector tends to detect a small fraction of the total number of faults, test compaction problems are in general of very low density.

The LP relaxation can be used as a guide for selective generation of test vectors: The LP is first solved exactly with a given set of columns. The optimal dual solution is next used to identify the fault which has highest dual multiplier. Such a fault can be intuitively thought of as being "expensive" to cover under the existing set of test vectors. For this fault, additional test patterns, distinct from the one already in the set, are determined. The problem is then solved repeatedly till either a set time limit has reached, or LP has become too large to solve or no more columns can be generated in reasonable time. We anticipate this strategy to work effectively for the test–set problem as it has been effective for the crew scheduling problems. Its effectiveness will increase if used in conjunction with a good test generation procedure.

5 Distinguishing features of the test compaction problem

We describe here how the features of the test compaction problem are used to get better solutions in the integer phase of solving the Set Cover problem. Once the fractional solution has been derived using the LP relaxation, the integer phase begins where a branch and bound procedure is employed.

While for the Set Cover problem, the sets considered are fully determined, the test vectors have don't care values (X values) which allows for added flexibility. Indeed those values are key in many compaction algorithms. For instance, the notion of compatibility of two test vectors implies that the two can be merged into one. Merging is possible when either the entries of the two vectors are identical or, if different, then at one of the two is a don't care value. More generally, when for a given subset of vectors, each input entry is either an X or a fixed value (either 0 or 1 but not both), then the subset can be merged into a single vector that has the fixed value in every input, or an X for inputs that are X for all the vectors.

It is straightforward to merge all possible pairs, until no pair is mergeable. When a set of vector has no mergeable pair it has been *maximally* merged and the vectors that were merged are a *maximal* merging. However, the order in which pairs are merged affects the size of the final set. We refer to the merging order that reduces the size of the set by most - *maximum merging* (maximum as opposed to maximal). The objective is to maximize the number of merged vectors, and therefore minimize the size of the remaining set. Finding the maximum merging is in itself an NP-complete problem. To see that, consider the following reduction from the 3-SAT problem.

Let the 3-SAT formula contain n variables and m clauses. For each clause $C_i = (x_{i_1}, x_{i_2}, x_{i_3})$ there is a fault f_i associated with it. That fault is detectable by the following 3 vectors, that assume the value X in all inputs other than i_1, i_2 and i_3. In those inputs (i_1, i_2, i_3) the vectors are $(1, X, X)$, $(X, 1, X)$ and $(X, X, 1)$ respectively. There are hence a total of $3m$ test vectors. There exists a satisfying assignment if and only if the maximum merging value is $3m - 1$ and the final set has only one vector in it (with values representing the satisfying assignment).

It is also possible to merge vectors that are incompatible while sacrificing some of the faults that they test for. Such procedure, *forced pair merging*, underlies the algorithms of Chang and Lin [5].

In the integer phase of the Set Cover problem certain variables are forced out of the current (incumbent) solution. When a variable is forced out, the procedure will look to merge it with a vector in the solution first. Assuming there are no compatible pairs, the merging will involve losing some of the faults that the two vectors jointly detect. The precise choice of the merged vectors is done by one of several heuristics. Further empirical study is necessary to determine which of the heuristics will perform the best, taking into account the trade-off between running time and effect on the speed of the branch and bound procedure.

6 Experimental results

A series of experiments were performed for the (post-generation) compaction approach using the formulation as Set Cover problem. As indicated earlier, it was unnecessary to employ the integer phase procedures beyond the LP relaxation.

The following tables contain results of the experiments run on a subset of ISCAS'91 benchmark combinational circuits. The results are divided into two parts. In first part, we use ATPG (Automatic Test Pattern Generator) program which is part of SIS (development version May 94), University of California at Berkeley, as the underlying test pattern generator. In the second part, we use COMPACTEST [20] program as the underlying test pattern generator.

For all the COMPACTEST compaction we achieved an optimal compaction, and a binary linear programming solution. This was the case also for all but two of the ATPG experiments.

The experiments reported were run on a SUN Sparc 1+ workstation.

6.1 SIS ATPG

Berkeley ATPG program uses boolean satisfiability subroutine to generate test sets. The program consists of three phases. In Phase I, a prescribed number (usually equal to the number of bits in integer representation which is equal to 32 in most computers) of random test vectors is generated repeatedly and fault simulation is performed in parallel for all of them. As long as a vector tests for a fault, it is included in the test set. The process continues till the random set fails to detect more than a threshold level of faults (by default equal to 10). In Phase II, one fault is taken at a time from the undetected fault list and boolean satisfiability subroutine is used to detect a test vector for it. If successful, the test vector is fault simulated and all the detected faults are removed from the list. The process continues till every fault is either detected, or determined to be redundant or in very few cases aborted. In Phase III, the final list of test vector is fault simulated with a reverse-order fault list. The Phase III is called ROFS (Reverse Order Fault Simulation) for short.

ATPG can be run with or without Phase I and Phase III. Only Phase II is mandatory for ensuring that a full attempt is made to find a test of every fault in the fault list. In the following table, we use ATPG in two different ways. For each circuit, first we use ATPG without Phase III (No ROFS) and then with Phase III (ROFS) to generate a set of test vectors. It should be pointed out here that because Phase I has a random element, ATPG runs are not entirely repeatable. The results provided here are not a minimum over many runs; instead they are just one instance.

The first column contains the name of the circuit. The second column contains two numbers separated by a slash("/"). The first number equals the size of the test set provided by ATPG run without Phase III. Call it Set 1. The second number equals the size of the test set provided by ATPG run with Phase III. Call it Set 2. The third column again contains two numbers separated by a slash("/"). Both numbers are product of the Set Cover (SCH) i.e. the application of the LP relaxation to the Set Cover formulation, applied to Set 1 and Set 2. The heuristic has been described above. The first number equals the size the the set of test vectors which form the output of SCH applied to Set 1. The second number equals the size the the set of test vectors which form the output of SCH applied to Set 2. The next column has the LP relaxation value which is a lower bound on the value of the optimal compaction. The next column has the running times that were required to run the LP relaxation, i.e. compaction times only. The "Best Known" column gives the size of the smallest test set known prior to this research. The paper reporting the result is cited. The missing entries indicate that this research, to the best of our knowledge, is the first one to report the size of the test set for those combinational circuits.

Note here that we achieved on average 40% reduction from the ATPG results. Note also that the best known results of [5] required substantial amount of running time (in thousands of CPU seconds) as compared to the procedure used here.

6.2 COMPACTEST

The program COMPACTEST [20] is described in the section on Previous Work. It consists of a pre-processing phase in which the faults are ordered for the next phase. We call it Phase I.

The following table presents the result of applying SCH on the sets of test vector which are the output of COMPACTEST. The table consists of three columns. The first column has the name of the circuit. The second column has two numbers separated by a dash. The first number equals the

Circuit Name	ATPG Test Set No ROFS/ROFS	SCH Test Set	LP Optimal Value	Time (sec)	Best Known
C17	8/6	5/5	5/5	0.1/0.4	5[3]
C432	90/83	49/49	49/49	1.3/1.1	29[5]
C499	82/57	52/52	52/52	4.0/3.7	53[5]
C880	136/89	69/70	69/70	3.7/2.8	21[5]
C1355	170/139	84/85	84/85	8.2/7.6	84[22][3]
C1908	242/211	115/118	115/118	16.0/14.2	106[5]
C2670	418/265	101/109	101/109	24.5/19.6	45[5]
C3540	326/304	169/164	163/159	29.4/28.0	91[5]
C5315	276/169	127/132	127/132	41.5/33.4	49[5]
C6288	59/41	38/33	27/33	23.4/16.7	14[5]
C7552	692/451	220/218	220/213	147.1/97.8	77[5]
b9	90/81	45/45	44/45	1.3/3.7	-
rot	209/176	95/87	95/87	12.1/15.4	-
k2	1371/934	643/640	643/640	78.0/68.3	-
dalu	456/344	205/183	194/175	53.1/50.0	-
t481	3245/2501	1221/1172	1220/1172	231.8/200.7	-
des	208/192	147/140	147/140	96.4/91.9	-

Table 1: Compression of ATPG test sets

Circuit Name	COMPACTEST With Phase I/ No Phase I	SCH Test Set	LP Optimal Value	Time (sec)
C432	42/44	39/39	39/39	1.4/1.1
C499	63/63	54/55	54/55	4.2/3.9
C880	32/33	32/31	32/31	2.6/1.4
C1355	87/88	84/85	84/85	5.7/6.0
C1908	142/145	120/116	120/116	13.4/13.9
C2670	63/62	62/61	62/61	9.8/7.8
C3540	114/119	110/112	110/112	15.6/16.3
C5315	64/59	64/59	64/59	14.4/13.3
C6288	25/22	20/20	20/20	9.9/9.7
C7552	90/94	90/93	90/93	29.7/29.6

Table 2: Compression of COMPACTEST test sets

size of the test set provided by COMPACTEST run with Phase I. The second number equals the size of the test set provided by COMPACTEST run without Phase I. The third column has two numbers again separated by a dash. They equal the size of test set after compaction using SCH. The input to SCH is the output of COMPACTEST run with and without Phase I.

Note that COMPACTEST is a set of vectors that has already gone through compaction procedures and as a result the number of vectors it is using is very close to the optimum (which is indicated by the values of SCH). So the additional savings delivered by the SCH approach is lesser than in the case of running ATPG test set.

7 Conclusions

In this paper we discuss a number of aspects of using optimization for test compression. We demonstrate that optimization procedures can be used effectively for compression of the set of test vectors required to test for faults in a combinational circuit. In particular, the Set Cover solution approach has the flexibility that it can use any set of test vectors as input, and generate a compact subset of those that still tests for the same faults. The approach devised here has also the advantage of pointing out specific faults that are expensive to test with the existing set of test vectors, thus guiding the generation of additional test vectors to those that are expected to be most effective in reducing the size of the test set.

The running time of the linear programming relaxation procedure is low enough to merit its incorporation as a final phase to follow any test generation and compression procedure.

Our approach here highlights the usefulness of integer programming optimization tools for testing and potentially for design problems as well.

8 Acknowledgements

The author wishes to thank Sushil Verma for work done on the software developments and other contributions to this write-up. We are also greatful to Irith Pomeranz and Sitaram Yadavalli for making the COMPACTSET vectors available. We thank the anonymous referees for their thoughtful comments, and to the referee that pointed out the important references [5] and [22].

References

[1] M. Abramovici, J.J. Kulikowski, P. R. Menon, and D. T. Miller. Smart and fast: Test generation for VLSI scan-design circuits. *IEEE Design and Test*, pages 43–54, Aug 1986.

[2] Sheldon B. Akers, Christie Joseph, and B. Krishnamurthy. On the role of independent fault sets in the generation of minimal test sets. *International Test Conference*, pages 1100–1107, 1987.

[3] Bechir Ayari and Bozena Kaminska. A new dynamic test vector compaction for automatic test pattern generation. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 13:353–358, 1994.

[4] C. Barnhart, E. L. Johnson, R. Anbil, and L. Hatay. A column generation technique for the long-haul crew assignment problem. Technical Report COC-91-01, Computational Optimization Center, School of Industrial and Systems Engineering, Georgia Institute of Technology, Atlanta, Georgia, 1991.

[5] J.-S. Chang and C.-S. Lin. Test set compaction for combinational circuits. *Asian Test Symposium*, 20–25, 1992.

[6] K. T. Cheng and V. D. Agarwal. A simulation-based directed-search method for test generation. *Proceedings of International Conference on Computers, Des.*, pages 48–51, Oct 1987.

[7] J. Desrosiers, Y. Dumas, M. Desrochers, F. Soumis, B. Sanso, and P. Trudeau. A breakthrough in airline crew scheduling. Technical Report G-91-11, GERAD, Montreal, Canada, 1991.

[8] H. Fujiwara and T. Shimono. On the acceleration of test generation algorithms. *IEEE Transactions on Computers*, C-32:1137–1144, Dec 1983.

[9] M. R. Garey and D. S. Johnson. *Computers and Intractability.* W.H. Freeman, San Francisco, 1979.

[10] Prabhakar Goel. An implicit enumeration algorithm to generate tests for combinational logic circuits. *IEEE Transactions on Computers*, C-30(3):215–222, 1981.

[11] N. Hall and D. S. Hochbaum. The Multicovering Problem. *European Journal on Operational Research*, 62, 323–339, 1992.

[12] D. S. Hochbaum. Approximation algorithms for the weighted set covering and vertex cover problems. *SIAM J. Comput.*, 11(3), 555–556, 1982. an extended version: W.P. #64-79-80, GSIA, Carnegie-Mellon University, April (1980).

[13] Seiji Kajihara, Irith Pomeranz, Kozo Kinoshita, and Sudhakar M. Reddy. Cost-effective generation of minimal test sets for stuck-at-faults in combinational logic circuits. *30th ACM/IEEE Design Automation Conference*, pages 102–106, 1993.

[14] T. Kirkland and M. R. Mercer. A topological search algorithm for atpg. *Proceedings of the 24th ACM/IEEE Design Automation Conference*, pages 502–508, Jun 1987.

[15] B. Krishnamurthy and Sheldon B. Akers. On the complexity of estimating the size of a test set. *IEEETC*, pages 750–753, Aug 1987.

[16] Tracy Larrabee. Test pattern generation using boolean satisfiability. *IEEE Transactions on Computer-Aided Design*, 2(1):625–636, Jan 1992.

[17] J. F. McDonald. Test set reduction using the subscripted d-algorithm. *International Test Conference*, pages 115–121, 1987.

[18] M. Minoux. Column generation techniques in combinatorial optimization: A new application to crew pairing problems. *AGIFORS Proceedings, France*, 1984.

[19] M. Minoux. *Mathematical Programming: Theory and Algorithms*. John Wiley and Sons, 1986.

[20] Irith Pomeranz, Lakshmi N. Reddy, and Sudhakar M. Reddy. COMPACTEST: A method to generate compact test sets for combinatorial circuits. *International Test Conference*, pages 194–203, 1991.

[21] B. Rannou. A new approach to crew pairing optimization. *26th AGIFORS Symposium, England*, 1986.

[22] Lakshmi N. Reddy, Irith Pomeranz and Sudhakar M. Reddy. ROTCO: A Reverse Order Test Compaction Technique *EURO-ASIC Conference*, pages 189–194, 1992.

[23] J.P. Roth. Diagnosis of automata failures: A calculus and a method. *IBM J. Research Development*, 10:278–291, Jul 1966.

[24] Michael H. Schulz, Erwin Trischler, and Thomas M. Sarfert. SOCRATES: A highly efficient automatic test pattern generation system. *IEEE Transactions on Computer-Aided Design*, 7(1):126–137, 1988.

[25] Y. Takamatsu and K. Kinoshita. Cont: A concurrent test generation algorithm. *FTCS-17*, pages 22–27, Jul 1987.

[26] E. Trischler. Atwig, an automatic test pattern generator with inherent guidance. *Proc. 1984 Int. Test Conf.*, pages 86–87, Oct 1984.

[27] Gert-Jan Tromp. Minimal test sets for combinational circuits. *International Test Conference*, pages 194–203, 1991.