Search Space Pruning Techniques in ATPG for VLSI Circuits

MICHAEL DIMOPOULOS, PANAGIOTIS LINARDIS
Department of Informatics
Aristotle University of Thessaloniki
GR-54006 Thessaloniki
GREECE
http://www.csd.auth.gr

Abstract: - This paper presents a, common, unified approach to solve either the test sequence compaction problem or the power minimization problem during circuit testing. This approach is based on an exact Branch and Bound algorithm that exploits information from the respective problems. In particular decision making during the Branch and Bound method follows some rules devised so as to avoid unnecessary choices and thus reducing the search space. Experimental results that are presented, comparing the proposed algorithm with other solvers from literature, show the effectiveness of the proposed method.

Key-Words: - Sequential Digital Circuits, Sequence Compaction, Test Generation, Set Cover methods.

1 Introduction

In several areas in Electronic Design Automation [1] (circuit synthesis, test generation, layout, etc) the computation of the optimum solution of a problem may be of great importance. Most of these problems are NP-complete so in order for one to be able to find an optimal solution of large-size instances, methods to effectively prune the search space need to be devised.

Here we'll concentrate on two problems from the field of Electronic Design Automation: the compaction of Test Sequences for sequential circuits [2, 5, 7] and the power minimization problem during testing for CMOS sequential circuits [3, 4]. For these two problems there have been developed many methods some of which rely on genetic algorithms [2, 7], others on various heuristics [4] or on exact methods using Branch and Bound (B&B) techniques [5, 6, 9].

In this paper we propose an exact algorithm as a common algorithmic framework for solving both problems. Our method is based on a specialized B&B algorithm that exploits the special structure of sequential test sets in order to further prune the search space, by carefully avoiding non-solution areas. The algorithm converges to the desired solution faster than the standard B&B algorithms, thus enabling the effective solution of larger instances of these problems.

The paper is organized as follows: In section 2 are presented the two problems. In section 3 the main lines of our method are analyzed. In section 4 the proposed overall method is presented. In section 5 experimental results are given, supporting the potential of the proposed method.

2 Problem Formulation

Given a VLSI sequential circuit let the *test set* $T=[S_1, S_2,\ldots, S_n]$, consisting of the *n* test sequences S_i, detect the *m* possible faults f_i of the circuit denoted by the set $F=[f_1, f_2,\ldots,f_m]$ (Fig. 1). Our problem is to properly select sequences or subsequence parts from T so as to cover all faults from F with minimum cost.

In our analysis the cost may represent either the sum of *test vectors* in the final solution (when the purpose is to minimize the *testing time* and the problem is known as *test sequence compaction*), or the number of *circuit transitions* after applying a selected set of subsequences during circuit testing (when the problem is to select those test vectors that minimize the *power dissipated* during circuit testing). Both problems belong to the category of Set Covering problems, for which various algorithms [3, 5, 8, 9] have been devised.

In our case, a common algorithmic framework is proposed here to solve either of the above problems. From the sets T and F a matrix D_{mn} (see example Fig. 2) is built where the m faults f_i form the rows, the n sequences S_j form the columns and the entries d_{ij} are the cost for covering fault f_i by selecting the corresponding subsequence of S_j. This matrix is known in the literature either as *Covering Matrix* [5] (test sequence compaction) or as *Transition Covering Matrix* [4] (power minimization).

Example

Let the test set T=[S_1, S_2, S_3] of Fig. 1 cover the set of faults F={f_1, f_2, f_3} of a given sequential circuit. Let us consider the case where the costs d_{ij} represent the number of vectors in each subsequence participating in the final solution (i.e. we have a test sequence compaction problem).

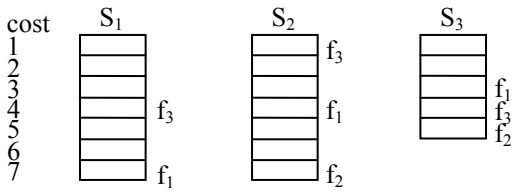

Fig.1. A set of test sequences

From the test set of Fig.1 we build the *Covering Matrix* [5] of Fig.2.

	S_1	S_2	S_3
f_1	d_{11}	d_{12}	d_{13}
f_2	d_{∞}	d_{22}	d_{23}
f_3	d_{31}	d_{32}	d_{33}

S_1	S_2	S_3
7	4	3
∞	7	5
4	1	4

Fig. 2. Covering Matrix D_{33}

The problem, now, is to select from D subsequences S_{ik} containing the first k vectors so that all faults f_i are detected (covered) with the minimum total number of test vectors. Equally well, in a power minimization problem d_{ij} represent the number of transitions.

3 Proposed Methodology

Let, at a stage of the algorithm, the following subsequences have been selected (decisions) from matrix D_{mn} (see example Fig. 2): (1) d_{ij} part from sequence S_j in order to cover fault f_i, (2) d_{pk} part from sequence S_k in order to cover fault f_p, (3) d_{qm} part from sequence S_m in order to cover fault f_q and (4) neither of the selected subsequences contains more than one from the *boundary* (top) faults f_i, f_p, f_q. Some sequences may not be selected at all, like S_r. This selection may be depicted by Fig 3. It must be noted that this selection covers not only faults f_i, f_p and f_q but, also, all faults that are contained within these subsequences.

The above selection imposes an ordering (decending) on the subsequences (Fig 3) and so an ordering on the decisions. This ordering should be respected in order to avoid examining multiple times the same sequence of selections. As seen from Fig 3, this selection divides the search space into two subspaces: subspace A containing the selected subsequences and subspace B containing the unselected or the remaining upper parts (higher cost) of the selected subsequences. Let us assume that the selection of Fig. 3 covers all faults from F, with a total cost *upper_cost*= $d_{ij} + d_{pk} + d_{qm}$. Since all faults have been covered by this selection the next step to proceed in order to search for the solution with the minimum *upper_cost* is to try to exchange a selected subsequence (from subspace A) that covers a *boundary* fault by another subsequence that covers the same fault. Since we need to cover all faults in F, after the removal of an already selected subsequence we have, by solution construction (requirement 4), that there is not in subspace A a subsequence that covers the *boundary* fault after the removal of it's respective subsequence. Therefore we must select (in order to cover at least this *boundary* fault) a subsequence from subspace B. Furthermore, the new subsequence from subspace B should not cover more than one boundary fault.

Our method starts by exchanging subsequence S_m and then proceeds to the left of Fig. 3 exchanging subsequences S_k and S_j.

For example in Fig. 3 let us try to exchange the selected part d_{pk} of sequence S_k which covers the boundary fault f_p. There are 4 cases to consider:

I) There is a sequence S_r (unselected sequence from subspace B) which contains f_p with part $d_{pr} < d_{\infty}$ and does not cover another *boundary* fault to the left of fault f_p (i.e. $d_{pr} < d_{ir} \leq d_{\infty}$).

II) There is a sequence S_r (unselected sequence in subspace B) which contains f_p with $d_{pr} < d_{\infty}$ and covers another *boundary* fault (i.e. $d_{ir} < d_{pr} < d_{\infty}$).

III) There is a sequence S_j (selected sequence which contains f_p with subsequence part $d_{pj} < d_{ij} < d_{\infty}$) and covers another *boundary* fault f_i.

IV) There is a sequence S_j (selected sequence which contains f_p with subsequence part $d_{ij} < d_{pj} < d_{\infty}$) and covers another *boundary* fault f_i.

Case III) contradicts to the way the subsequences in subspace A were selected and is disregarded. Case II) also contradicts to the way the subsequences in subspace A were selected because we would exchange d_{pk} with a subsequence d_{pr} that covers a boundary fault (f_i) of a different subsequence (S_j).

Case (IV) selects a larger part from a selected subsequence leading to a new subsequence part d_{pj} that merges the older boundary fault f_i.

Therefore, only in cases (I) and (IV) we go and make the exchange.

In order to ensure search space completeness for the above subsequence exchanges, a B&B algorithm is employed. The above subsequence exchanging process is formulated with the following two Rules which are used within the B&B algorithmic environment:

Rule I: Given an initial decision (f_i, d_{ij}) every other decision (f_p, d_{pk}) with $k \neq j$ (subspace B Fig. 3) is *valid* wrt the initial decision if $d_\infty \geq d_{ik} > d_{pk}$.

Rule II: Given an initial decision (f_i, d_{ij}) every other decision (f_p, d_{pk}) with $k=j$ is *valid* wrt the initial decision if $d_{ik} < d_{pk} < d_\infty$ (i.e. lies in subspace B Fig. 3).

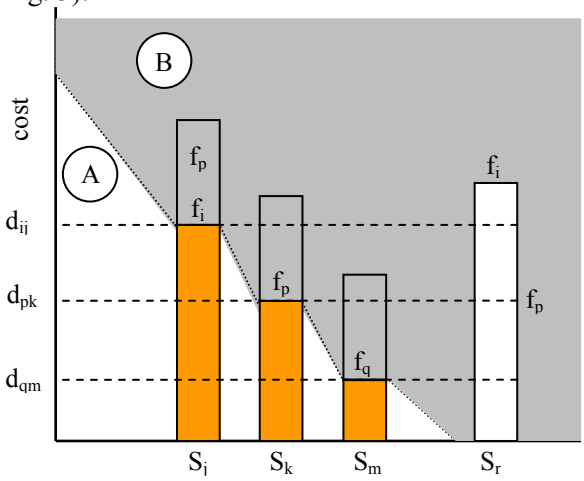

Fig. 3 Subsequence selection

The Rules I and II constitute the validity analysis (section 3.1) which is used in our Branch and Bound algorithm. If either Rule I or II is invalidated (i.e. a selection is attempted from within subspace A) then we denote this case as invalid and we say that a *conflict* has occurred. In case of a conflict our selection method backtracks immediately and searches for another choice. In this way, our searching method adds to the standard bound conflicts the new bound conflicts due to the validity analysis (section 3.1)

Let us apply the above subsequence exchange process to the example of Fig. 1:

For this example we have six possible decision orderings for the 3 faults f_1, f_2, f_3: (I) f_1, f_2, f_3, (II) f_1, f_3, f_2, (III) f_2, f_3, f_1, (IV) f_2, f_1, f_3, (V) f_3, f_2, f_1, (VI) f_3, f_1, f_2. For example the ordering (I) (Fig. 5) gives the search tree of Fig. 4. A standard B&B algorithm would make an initial subsequence selection with cost *upper_cost* and then would try to bound the searching by using the *upper_cost*. By adding the Rules (I, II) we have developed to the standard B&B algorithm we will need to consider fewer test cases. For example let us suppose that, initially we selected d_{11} to cover f_1 (Fig. 1). According to Rule I the choices d_{22} for f_2 and d_{23} for f_2 are immediately crossed out because it is: $d_\infty > d_{22} > d_{12}$ and $d_\infty > d_{23} > d_{13}$. Therefore our B&B algorithm would backtrack and select other subsequence (d_{12}, d_{13}) for f_1. With similar to the above analysis our algorithm drops of all selections marked by dashed lines. Therefore from the initial six different choices (each leaf boxed-node denotes a possible solution by considering the encountered decisions along the path from the tree root to the respective leaf node) we only have to examine (search) three (i.e. we have a 50% reduction in the number of search nodes).

In Fig. 5 are given several orderings of the faults f_1, f_2, f_3, and in table form, the decision tree and for every selection a result (column 'Validity Analysis') after applying Rules (I, II). From Fig. 5 we see that the decision orders (III) [f_2, f_3, f_1] and (IV) [f_2, f_1, f_3] give the smallest decision trees of size 2 (only 2 decisions need to be searched-analyzed) while order (VI) in Fig. 5 [f_3, f_1, f_2] gives the largest decision tree having a total of 14 search nodes (choices).

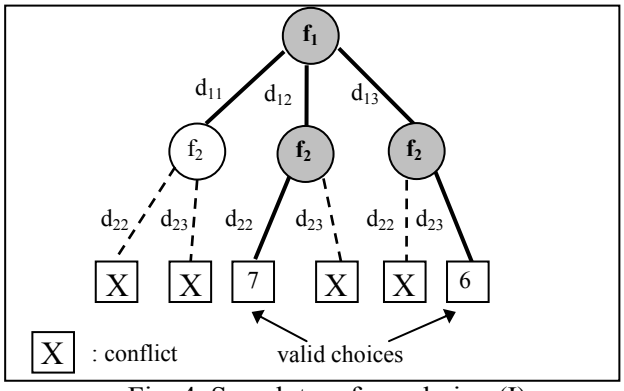

Fig. 4. Search tree for ordering (I)

A/A	Order I		Validity Analysis
1	f_1,d_{11}	f_2,d_{22}	conflict
2	f_1,d_{11}	f_2,d_{23}	conflict
3	f_1,d_{12}	f_2,d_{22}	valid
4	f_1,d_{12}	f_2,d_{23}	conflict
5	f_1,d_{13}	f_2,d_{22}	conflict
6	f_1,d_{13}	f_2,d_{23}	valid

A/A	Orders III, IV	Validity Analysis
1	f_2,d_{22}	valid
2	f_2,d_{23}	valid

A/A	Order VI			Validity Analysis
1	f_3,d_{31}	f_1,d_{11}	f_2,d_{22}	conflict
2	f_3,d_{31}	f_1,d_{11}	f_2,d_{23}	conflict
3	f_3,d_{31}	f_1,d_{12}	f_2,d_{22}	conflict
4	f_3,d_{31}	f_1,d_{12}	f_2,d_{23}	conflict
5	f_3,d_{31}	f_1,d_{13}	f_2,d_{22}	conflict
6	f_3,d_{31}	f_1,d_{13}	f_2,d_{23}	conflict
7	f_3,d_{32}	f_2,d_{22}		valid
8	f_3,d_{32}	f_2,d_{23}		conflict
9	f_3,d_{33}	f_1,d_{11}	f_2,d_{22}	conflict
10	f_3,d_{33}	f_1,d_{11}	f_2,d_{23}	conflict
11	f_3,d_{33}	f_1,d_{12}	f_2,d_{22}	conflict
12	f_3,d_{33}	f_1,d_{12}	f_2,d_{23}	conflict
13	f_3,d_{33}	f_1,d_{13}	f_2,d_{22}	conflict
14	f_3,d_{33}	f_1,d_{13}	f_2,d_{23}	valid

Fig. 5. Validity Analysis for the example

3.1 Validity Analysis

The chain of decisions may be represented by a *decision tree*, with each node leading to a smaller problem that needs to be solved. Any path from the tree root (initial problem) towards the leaves represents a solution i.e. all faults in F are covered (section 2). A decision node contains the pair (f_i, d_{ij}) denoting that in order to cover fault f_i (i-row) a subsequence of j is selected with cost (vectors or transitions) d_{ij} (subspace A in Fig. 3).

During this validity analysis step which is incorporated into the Branch and Bound process, it is checked that the next to try (new) decision (f_p, d_{pk}) is in accordance with Rules I and II (section 3). This is accomplished by testing the new decision with all previously-made decisions. In case of a *conflict* (section 3) one of the *valid* choices (for fault f_p) should be considered.

A schematic application of this validity analysis was presented in Fig. 5, were we were able to observe the prunning of the search space. As we will see, also, such reductions are confirmed by the experimental results (section 5).

In the case where all possible choices for a fault f_p are not valid then we say that this fault is *blocked* by previously-made decisions. For example let us assume that we have (Fig 6) the ordered sequence of decisions: (f_i, d_{ij}), (f_p, d_{pr}), (f_q, d_{qk}). Also, let the decision (f_p, d_{pr}) producing a conflict to (f_q, d_{qk}) and the decision (f_i, d_{ij}) producing a conflict to (f_q, d_{qm}). The fault f_q is *blocked* and some of the previously made decisions don't belong to the minimum solution (see Lemma 2).

In the event of a *blocked* fault our B&B algorithm backtracks to the nearest (**most recent**) decision (f_p, d_{pr}) (chronological backtrack) that is a source of conflict, to some choices for fault f_p, and makes an alternative selection. In the case where this decision is not the immediately previous one we will have a non-chronological backtrack [6] step.

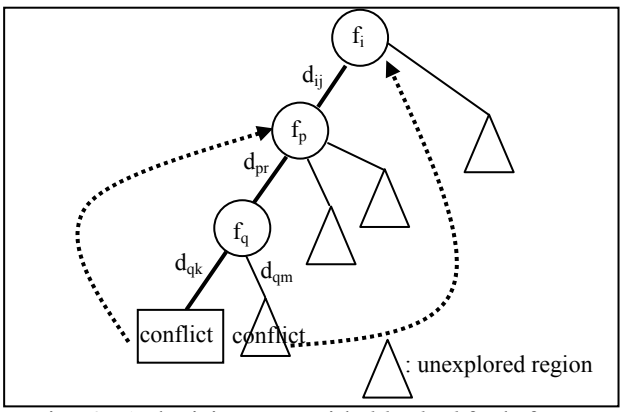

Fig. 6. A decision tree with *blocked* fault f_q

The above steps may be coded into the following rule: If a fault f_p is *blocked*, revert to the **most recent** decision that contributes to the blocking of fault f_p and select another candidate.

In our algorithm we make use of the following remarks:

Lemma 1: The minimum solution is not affected by the validity conditions.

Proof: Let's assume that the minimum solution [(f_i, d_{ij})...(f_p, d_{pk})] doesn't respect the validity conditions. We have two cases to consider:

I) k=j, $d_{ik} < d_{pk} \leq d_\infty$, i.e. fault f_i is covered by a subsequence of k sequence with length (cost) d_{pk} and therefore this makes redundant the previous selected subsequence of sequence j with length (cost) d_{ij}. (contradiction due to the minimum solution).

II) $k \neq j$, $d_{pj} < d_{ij} < d_\infty$, i.e. fault f_p is covered (without extra cost) by the subsequence of sequence j with length (cost) d_{ij}.

Lemma 2: If a fault is *blocked* then at least one of the previously-made decisions involved in the blocking of the fault does not belong to the minimum solution.

Proof: It follows from Lemma 1.

Lemma 3: If all available choices for a fault f_i cause the blocking of another fault f_q then at least one of the previously-made decisions (before fault f_i) does not belong to the minimum solution and a backtrack step is necessary in order to change some previously (before fault f_i) made decision.

Proof: It follows from Lemma 2.

Lemma 4: A Branch and Bound algorithm applied to the problem of compacting a set of Test Sequences enhanced with the *validity* analysis is a *complete* algorithm.

Proof: From Lemma 1 and the validity analysis only decision nodes that violate the validity conditions are ignored and therefore the optimal solution is retained.

4 The Overall algorithm

The proposed algorithm consists of the following: procedures:

1) Formulation of the Covering Matrix (section 2) from the given test sequences by fault simulating each sequence.

2) Heuristic computation of an initial solution (i.e. computation of an initial upper bound) using the MinMax algorithm [4]. This initial solution is used to seed the following B&B algorithm.

3) A B&B (*ImprBB*) algorithm equipped with the validity analysis step (section 3.1) tries to select a subset of subsequences, which cover all faults with the smallest possible cost (number of vectors,

number of transitions). In our Branch and Bound algorithm for every decision the minimum cost choice is explored first. The reason will be explained with the help of Fig. 7.

In Fig.7 we have a sequence of decisions (decision tree) leading to an initial solution: { (f_i,d_{ij}), (f_p, d_{pr}), (f_q,d_{qk}) }. As we proceed from f_q and upwards H_q=d_{qk} is an upper bound for the set of faults F_q covered by the choice of (f_q, d_{qk}). If d_{qk} is the minimum (least) cost choice to cover fault f_q then H_q will be also a lower bound for the set F_q and the remaining choices (to the right of d_{qk} in Fig. 7) may be skipped. Therefore, by selecting for every decision the minimum cost choice we may prune earlier the search space. Also certain branches are pruned due to the validity analysis (section 3.1)

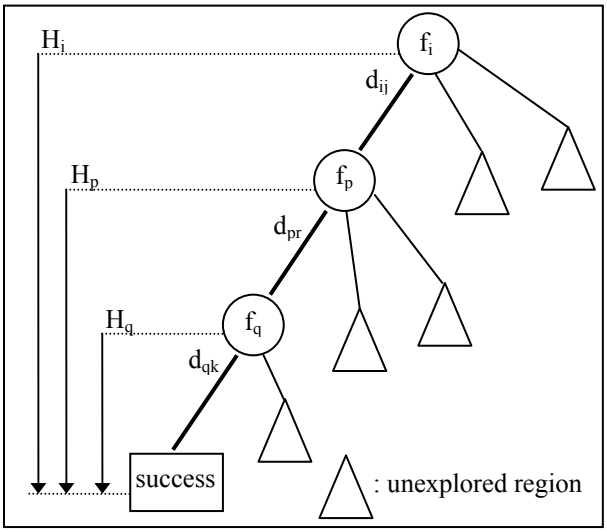

Fig. 7. Example of a decision tree

In *ImprBB* algorithm the following bound conflicts can be identified: a) bound conflicts as encountered in a standard B&B method and b) bound conflicts (identified without search) due to validity analysis (section 3). The bound conflicts occur when the lower bound (*lower_bound*) is higher than or equal to the upper bound (*upper_cost*). This condition may be written as:

$$path\_cost + lower\_bound \geq upper\_cost,$$

where: *path_cost* : is the cost of already-made decisions, *lower_bound* : is an estimate on the cost of covering the rows (faults) not yet covered.

Since the problems to be solved (test sequence compaction, power minimization during circuit testing) belong to the NP-complete class, certain limits are set to *ImprBB* and a flag is raised whenever *ImprBB* exceeds these limits: (a) a time limit of 10 min, and (b) a limit on the total number of backtracks of 10^6 nodes (i.e. number of different search decisions that are analyzed).

5 Experimental Results

Our algorithm *ImprBB* has been implemented in C. The efficiency of the algorithm was measured by running the ISCAS'89 benchmark circuits [10] on a Pentium III PC with 256 Mb.

In Table 1 are presented the Original Problem (#seq. is the number of sequences, #faults are the detected faults and #vectors is the total amount of vectors) that were obtained from [7]. Also for every example the Minimum Test Set is computed by a B&B method [5].

As a first experiment, a standard B&B algorithm from literature [11] is applied on the test sets of Table 1. The Time limit is set to 20 min per instance: Unfortunately, and for all the examples, this algorithm failed to produce the minimum solution in the given time limit.

On top of this standard B&B algorithm we build our proposed B&B algorithm *ImprBB* (section 4). This was done, in order to evaluate the actual performance of our method with respect to the standard B&B which is not attributed to implementation issues.

The results obtained by *ImprBB* are presented in Table 1. Under column 'Time' we have the time required to solve the respective problem and under column '#Backtracks' we have the total number of unsuccessful searches before proving the optimality of the solution. As we see, our method manages to solve to optimality all of the given examples and at reasonable times.

As a next experiment we applied on the same set of examples two other efficient algorithms from literature: algorithm *MINCOV* which is used in the logic synthesis package of ESPRESSO [9] and a method that solves ILP problems, *glpsol-4.8* (GLPK package [13]). In the literature there are several attempts to combine the strength of general purpose ILP methods with problem specific knowledge from Electronic Design Automation [12]. Having these methods as a guide we applied the ILP solver *glpsol-4.8* to our own problem instances.

From Table 1 we have that *MINCOV* solves all but two of the larger examples (s35932, s38584), while *glpsol-4.8* fails to solve four examples (s3330, s6669, s35932, s38584).

Comparing the results of our method *ImprBB* and those of *MINCOV* and *glpsol-4.8*, we observe that: a) *ImprBB* succeeds in solving all examples, 2) in smaller time (actually *ImprBB* is faster than *MINCOV* and *glpsol-4.8* by 2 to 3 orders of magnitude) and 3) by searching fewer decisions (column #Backtracks).

Table 1. Experimental results for GATTO [7] Test Sets

circuit	Original Problem		Minimum		ImprBB		MINCOV		glpsol-4.8	
	#faultsx#seq	#vectors	#seq	#vectors	Time(s)	#Backtracks	Time(s)	#Backtracks	Time(s)	#Backtracks
s510	551 x 37	989	7	239	0,02	26	1,12	55	6	571
s953	1044 x 75	1099	32	541	0,14	307	1,69	173	13,8	1145
s967	1019 x 72	1223	31	671	0,07	153	1,65	237	15	1150
s991	857 x 20	448	9	367	0,02	10	0,76	21	6	866
s1196	1200 x 133	1805	74	1126	0,32	713	2,79	1808	21	1511
s1238	1227 x 133	1554	74	1006	0,23	694	6,04	3815	16,8	1471
s1269	1306 x 52	450	29	247	0,12	255	2,73	1181	15,6	1484
s1423	1418 x 107	2691	28	1281	0,12	35	2,04	62	19	1521
s1488	1422 x 65	1824	19	948	0,1	50	4,39	73	46	1502
s1494	1412 x 62	1244	19	654	0,12	20	2,5	41	24,2	1481
s3271	3188 x 132	2529	50	1180	1,22	1068	35,21	1879	230,8	3455
s3330	2336 x 108	2028	43	1069	0,47	398	16,02	1716	-	-
s3384	3096 x 58	888	22	412	0,31	61	12,61	209	114	3164
s4863	4482 x 112	1533	41	747	1,5	968	157,18	4389	337	4686
s5378	3271 x 71	919	42	495	0,5	191	15,31	453	138,6	3434
s6669	6507 x 64	592	36	303	1,42	865	32,85	540	-	>13200
s13207	1994 x 34	544	9	189	0,23	23	3,7	23	31,8	2018
s15850	659 x 10	153	3	93	0,12	17	0,3	23	1,8	661
s35932	34302 x 59	903	8	310	13,8	23	-	-	-	-
s38417	6516 x 95	1617	30	686	6,1	2520	224,28	1515	630	7125
s38584	18390 x 271	8065	105	3808	48	5861	-	-	-	-

6 Conclusion

The development of methods that exploit problem specific knowledge gives leverage over more general purpose algorithms. In this paper, a common algorithmic framework is presented that handles the problems of test sequence compaction and power minimization during circuit testing. In this framework an exact method consisting of a specially designed Branch and Bound algorithm to handle these specific problem instances is shown to be more effective than general purpose solvers.

References:

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

[2] M. Dimopoulos, P. Linardis, "Improving a GA-based ATPG for Sequential Circuits by Exploiting Dynamically Generated Essential Sequences", in "Advances in Scientific Computing, Computational Intelligence and Applications", *Ed. N. Mastorakis etc, WSES Press, 2001*, pp 373-377.

[3] F. Corno, P. Prinetto, M. Rebaudengo, M. Sonza Reorda, "A Test Pattern Generation Methodology for Low Power Consumption", in 16th IEEE VTS, April 1998, pp. 453-457.

[4] M. Dimopoulos, P. Linardis, "Improved Selection of Test SubSequences in Sequential Circuits for Reduced Power Consumption", IMACS/IEEE-CSCC'2003, July 2003, Greece.

[5] M. Dimopoulos, P. Linardis, "Accelerating the Compaction of Test Sequences in Sequential Circuits through Problem Size Reduction", IEEE Trans. on CAD, vol. 22, no. 10, October 2003, pp. 1443-1449.

[6] V. Manquinho, J. Silva, "Conditions for Non-Chronological Backtracking in Boolean Optimization", AAAI Workshop on the Integeration of AI and OR Techiques, July 2000.

[7] F. Corno, P. Prinetto, M. Rebaudengo, M. Sonza Reorda, "New Static Compaction Techniques of Test Sequences for Synchronous Sequential Circuits", ED&TC, 1997, pp. 37-43.

[8] O. Coudert, "Two-level logic minimization: An overview", VLSI journal, Oct. 1994, pp. 97-140.

[9] E. Goldberg, L. Carloni, T. Villa, R. Brayton, A. Sangiovanni-Vincentelli, "Negative Thinking in Branch-and-Bound: the Case of Unate Covering", TCAD, vol. 19, no. 3, March 2000.

[10] F. Brglez, D. Bryan and K. Kozminski, "Combinational profiles of sequential benchmark circuits", Int. Symp. on Circuits and Systems, 1989, pp. 1929-1934. (ISCAS'89 Benchmarks from: www.cbl.ncsu.edu/pub/Benchmark_dirs/.).

[11] T. Cormen, C. Leiserson, R. Rivest, C. Stein, "Introduction to Algorithms", MIT Press.

[12] S. Liao, S. Devadas, "Solving Covering Problems Using LPR-Based Lower Bounds", DAC, 1997.

[13] http://www.gnu.org/software/glpk/glpk.html