Advanced Techniques for GA-based sequential ATPGs

F. Corno, P. Prinetto, M. Rebaudengo, M. Sonza Reorda
Politecnico di Torino
Torino, Italy

R. Mosca
CSP (Centro Supercalcolo Piemonte)
Torino, Italy

Abstract[*]

Genetic Algorithms have been recently investigated as an efficient approach to test generation for synchronous sequential circuits. In this paper we propose a set of techniques which significantly improves the performance of the GA-based ATPG algorithm proposed in [PRSR94]: in particular, the new techniques enhance the capability of the algorithm in terms of test length minimization and fault excitation. We report some experimental results gathered with a prototypical tool and show that a well-tuned GA-based ATPG is generally superior to both symbolic and topological ones in terms of achieved Fault Coverage and required CPU time.

1. Introduction

Different approaches have been proposed to solve the problem of Automatic Test Pattern Generation for Synchronous Sequential circuits.

The *topological approach* [NiPa91] is based on extending to sequential circuits the branch and bound techniques developed for combinational circuits by adopting the Huffman's Iterative Array Model. The method's effectiveness heavily relies on the heuristics adopted to guide the search; the approach uses a complete, but often fails when applied to large circuits, where the search space is excessively large to explore.

The *symbolic approach* [CHSo93] exploits techniques for Boolean function representation and manipulation which were initially developed for formal verification; this approach is based on a complete algorithm, too, and is very effective when small- and medium-sized circuits are considered. Unfortunately, it is completely unapplicable when circuits with more than some tens of Flip-Flops are dealt with. This greatly limits its usefulness in real practice.

Finally, the *simulation-based approach* [ACAg88] consists in generating random sequences, simulating them, and then modifying their characteristics in order to increase the obtained fault coverage. In the last few years, several methods [SSAb92] [RPGN94] [PRSR94] have been proposed, which combine this approach with the use of Genetic Algorithms (GAs) [Gold89]. Results demonstrated that the approach is very flexible and provides good results for large circuits, where other methods fail.

However, the analysis we performed on the behavior of GATTO (the tool described in [PRSR94]) shows that the algorithm has some weakness points:

- the cross-over operator is not as effective as in other problems GAs have been applied to;
- the method can hardly determine the length of the sequences; this results in an increase of the time required by the ATPG process, and of the number of generated vectors;
- the phase devoted to find sequences which excite faults is purely random; this obviously decreases the method effectiveness in terms of achieved fault coverage and required CPU time.

In this paper we introduce some new techniques to overcome the above problems. We devised a more effective cross-over operator, and added new techniques which provide the method with the capability of automatically determining the minimal length of the test sequences. Finally, we re-arranged the whole algorithm in order to increase the effectiveness of the fault excitation phase, which is no longer purely random, but exploits information from the already generated sequences. To experimentally prove the effectiveness of the proposed techniques we implemented an improved version of GATTO, named GATTO+. The results show substantial improvements in GATTO+: from one side, they are now comparable to the competing algorithms on the small- and medium-size circuits; from the other side, results are further enhanced on the largest ones.

The paper is organized as follow: in Section 2 we briefly summarize the GATTO algorithm; Section 3 describes the improvements introduced in GATTO+. Section 4 presents the experimental results we gathered and provides some comparisons with other ATPG algorithms. Section 5 draws some conclusions.

[*] This work has been partially supported by European Union through the ESPRIT PCI project #9452 94 204 70 PETREL. Contact address: Paolo Prinetto, Dipartimento di Automatica e Informatica, Politecnico di Torino, Corso Duca degli Abruzzi 24, I-10129 Torino (Italy), e-mail `Paolo.Prinetto@polito.it`

2. The GATTO algorithm

The GATTO algorithm, presented in [PRSR94], is organized in three phases:

- the first phase aims at selecting one fault (denoted as target fault); this phase consists of randomly generating sequences and fault simulating them w.r.t. the untested faults. As soon as one sequence is able to excite at least one fault, the fault is chosen as target fault;
- the second phase aims at generating a test sequence for the target fault; it is implemented as a Genetic Algorithm: each individual is a test sequence to be applied starting from the reset state; cross-over and mutation operators are defined to modify the population and generate new individuals; a fitness function evaluates how close each individual is to the final goal (i.e., detecting the target fault); this function is a weighted sum of the numbers of gates and Flip-Flops having a different value in the good and faulty circuit. After a maximum number of unsuccessful generations the target fault is aborted and the second phase is exited;
- the third phase is a fault simulation experiment which determines whether the test sequence possibly generated in phase 2 detects other faults (fault dropping).

The three phases are repeated till either all the faults have been tested or aborted, or a maximum number of iterations has been reached.

3. Improvements

Based on the results reported in [PRSR94], we realized that several points in the GATTO algorithm were still worth of improvements. We will describe them in details in the following subsections, together with the improved solutions we devised for each of them.

3.1. Cross-Over Operator

The cross-over operator adopted in GATTO belongs to the category denoted as *two-cuts* cross-over. The operator works in a *horizontal* manner: the new sequence is composed of some vectors coming from either parents (Fig. 1), according to the position of two randomly generated cut points. Unfortunately, there is no guarantee that the vectors coming from the second parent produce in the new sequence the same behavior they produce in the parent sequence, as the state from which they are applied is different. As a consequence, we observed in GATTO that the off-spring of two good individuals was often a bad individual; in general, the cross-over operator was not as effective for the ATPG problem as it usually is for other problems GAs have been applied to.

The cross-over operator defined for GATTO+ works in a *vertical* manner; the off-spring does not inherit whole vectors from parents: rather, the values for each input are taken either from one parent or from the other, depending on a random choice (Fig. 2), as the operator belongs to the category known as *uniform* cross-over. The length of the new sequence is that of the longest between the two parent sequences: inputs taken from the shortest parent are completed with random values (dark in Fig. 2).

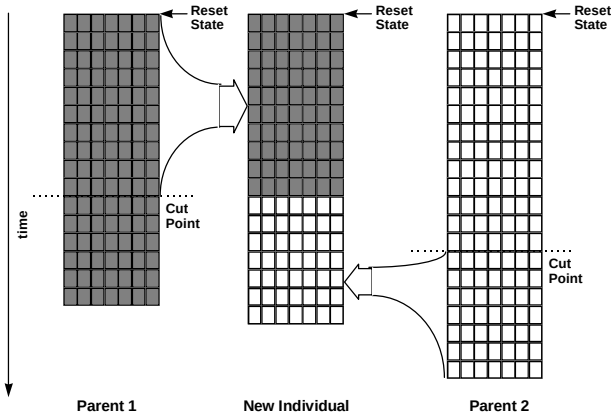

Fig. 1: Cross-over operator in GATTO.

3.2. Test Length

In GATTO it is up to the user to decide the initial test length, which is then automatically increased during the ATPG process. For some circuits, this results in a test length higher than the minimum one, while for other circuits the process spends many iterations for reaching the length required to test some faults. Moreover, the computational complexity of the whole process mainly depends on the cost of fault simulation; therefore, any unnecessary increase in the length of the sequences results in a corresponding waste in the required CPU time.

To face this problem we improved the GATTO algorithm in two ways: we first modified the evaluation function on which the fitness function is based, and then introduced new mutation operators.

3.2.1. New Evaluation Function

The evaluation function adopted in GATTO is based on the following expression

$$h(v^j_k, f_i) = c_1 * b_1(v^j_k, f_i) + c_2 * b_2(v^j_k, f_i) \quad (1)$$

which provides a measure of how close the k-th input vector v^j_k of a sequence s_j is to detect the fault f_i. In (1), c_1 and c_2 are constants, while b_1 and b_2 are functions, whose value is proportional to the number of gates and Flip-Flops (respectively) having a different value in the good and faulty circuit for fault f_i. Once the value of $h(v^j_k, f_i)$ is known for every vector in the sequence, the evaluation function H for the sequence s_j is computed as

$$H(s_j, f_i) = \max_k(h(v^j_k, f_i)) \quad \forall\, v^j_k \in s_j \quad (2)$$

In order to bias the evolution towards the shortest sequences, a modified version $H^*(s_j, f_i)$ of the evaluation function H has been introduced in GATTO+; the value of $h(v^j_k, f_i)$ for the k-th vector of the j-th sequence is weighted with a coefficient whose value decreases with k; the new evaluation function corresponds to the maximum value of the weighted function:

$$H^*(s_j, f_i) = \max_k(\mathrm{HANDICAP}^k \cdot h(v^j_k, f_i)) \quad \forall\, v^j_k \in s_j \quad (3)$$

where the value of the parameter HANDICAP ranges between 0 and 1; thanks to this coefficient, shorter sequences are preferred and the final test length is reduced.

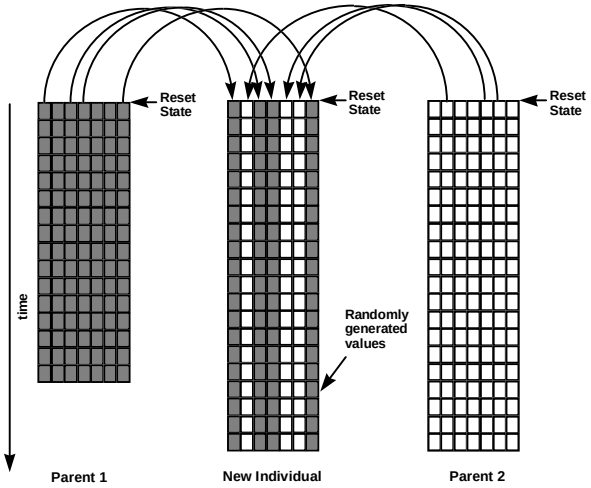

Fig. 2: Cross-over operator in GATTO+.

3.2.2. New Mutation Operators

In GATTO, any change in the length of the sequences during phase 2 stems from the cross-over operator: in fact, the length of any new sequence can randomly vary up to the sum of the lengths of the two parent sequences. The new cross-over operator presented in the previous Section behaves in a completely different way, and generates sequences as long as the longest parent. This means that the length of the sequences in a population can never be higher than that of the longest one in the previous generation. Unfortunately, there is thus no way to increase the sequence length.

To overcome this problem, and to force the algorithm to better explore all the search space, we introduce two new mutation operators (MO+ and MO-), which are activated on an existing sequence with a given activation probability:

- MO+ introduces a randomly generated vector in a random position within the existing sequence; thanks to this operator, longer sequences are generated and evaluated;
- MO- removes a randomly selected vector from the existing sequence: if the vector is not essential, the evaluation function of the sequence increases.

3.3. Fault Excitation

Fault excitation is one of the most critical problems when devising a GA-based ATPG. In fact, no way has been found, up to now, to evaluate how close a sequence is to excite a fault. Without such an evaluation function, a GA is not able to perform any search, and degenerates into a purely random method. All the GA-based ATPGs proposed in the literature resort to a purely random search for exciting faults.

In GATTO, the task of exciting faults is accomplished in phase 1 resorting to a completely random approach; in fact, the GA is only exploited for observing faults, i.e., in phase 2. Obviously, this results in some difficulty when trying to test hard-to-excite faults. To overcome this problem, we modified the algorithm in order to re-use as much as possible the work done in phase 2.

Once the GATTO's algorithm entered in phase 2, only the target fault is considered; if a test sequence is then generated, this is fault simulated w.r.t. all the untested faults (phase 3). However, it is likely that sequences generated in phase 2 to test one fault be also able to excite other faults, although they do not detect them. Therefore, in GATTO+, at the end of each phase 2 all the sequences belonging to the last population are stored, and then used in the following phase 1 instead of the randomly generated ones (as in GATTO).

If one of these sequences is able to excite at least one fault, this is selected as target fault, and a new phase 2 is activated. Otherwise, random sequences are generated trying to excite faults, like in GATTO. The pseudo-code of phase 1 is reported in Fig. 3.

4. Experimental Results

We implemented a prototypical version of GATTO+ containing all the techniques described above: the new cross-over operator substitutes the old one, and the operators MO+ and MO-, as well as the parameter HANDICAP, have been introduced. The values of all the parameters have been experimentally determined through a preliminary set of runs: the operators MO+ and MO- are activated with probability 0.05 and 0.1, the parameter HANDICAP holds the value 0.98, and C1 and C2 have been assigned the values 1 and 10, respectively. Tab. 1 reports the results in terms of Fault Coverage (FC), CPU time and test length for the whole set of ISCAS'89 circuits. Experiments have been performed on a workstation DEC Alpha 3000/500.

```
phase 1()
{  A =   {sequences belonging to the last
          population of the last phase 2};
   iteration_counter = 0;
   while (iteration_counter< MAX_ITER )
   {fault simulate all the sequences in A
              w.r.t. the  untested faults;
         if some fault fk is detected
              drop fk;
         if some fault fk is excited
         {  select fk as target fault;
            return ( fk );
         }
         A={randomly generated sequences};
         iteration_counter ++;
   }
   return ( NO_FAULT );
}
```

Fig. 3: Pseudo-code of phase 1.

4.1. GATTO+ vs. GATTO

To demonstrate the effectiveness of the described techniques we report in Tab. 2 a comparison with the results of GATTO published in [PRSR94], where only the largest ISCAS'89 circuits were considered. GATTO+ is able to increase the fault coverage in 11 cases out of 12, and in 6 cases the increase is greater than 4%. On the other side, the CPU time is decreased in 9 cases out of 12. For all the circuits, we were able either to increase the Fault Coverage by more than 4%, or to decrease the CPU time. GATTO+ achieves this result mainly thanks to the more effective technique adopted for phase 1, whose cost is now greatly decreased. Concerning the test length, the number of test vectors generated by GATTO+ is sometimes higher than those of GATTO, due to the new sequences added to detect other faults.

Circuit	# Faults		FC	CPU time	# Vectors
	Total	*Detected*	*%*	*[s]*	
S208	215	150	69,77	22	215
S298	308	273	88,64	23	240
S344	324	319	98,46	1	103
S349	332	325	97,89	1	104
S382	399	378	94,74	735	1850
S386	384	314	81,77	27	371
S400	424	396	93,40	212	1431
S420	430	204	47,44	55	197
S444	474	438	92,41	219	1733
S510	564	564	100,00	58	968
S526	555	462	83,24	214	2946
S526n	553	465	84,09	612	3042
S641	465	406	87,31	22	266
S713	581	480	82,62	26	281
S820	850	800	94,12	1287	1630
S832	869	586	67,43	80	401
S838	857	303	35,36	121	202
S953	1079	1069	99,07	28	852
S1196	1242	1236	99,52	118	1091
S1238	1355	1280	94,46	213	1905
S1423	1515	1465	96,70	362	2855
S1488	1486	1438	96,77	104	891
S1494	1506	1446	96,02	72	886
S5378	4603	3560	77,34	420	832
S9234	6927	405	5,85	19	8
S13207	9815	2007	20,45	280	272
S15850	11719	652	5,56	109	101
S35932	39094	35090	89,76	2958	1015
S38417	31180	5595	17,94	5724	593
S38584	36306	18030	49,66	6923	3249

Tab. 1: GATTO+ performance on ISCAS'89 circuits.

4.2. GATTO+ vs. other algorithms

W report in the following the data published for two other ATPG algorithms and concerning the ISCAS'89 benchmark circuits. In Tab. 3 we consider HITEC, a topological algorithm described in [NiPa91], and the GA-based ATPG proposed in [RPGN94]. The two algorithms were selected, as they are representative of the two categories we denoted above as *topological* and *simulation-based* ATPG algorithms; we did not consider any ATPG belonging to the category of *symbolic* ones, as they are not able to deal with large circuits, which are normally the most critical problem in the real world.

Two difficulties must be faced when performing such a comparison: the first one concerns the hardware platform, which is different for the three ATPGs (results for HITEC were gathered on a SPARCstation 1, those in [RPGN94] on a SPARCstation II, and those for GATTO+ on a DECstation 3000/500). The second difficulty comes from the fact that GATTO+ assumes that all the Flip-Flops in the circuits are resettable, and generates sequences starting from the all-0s state, while the two other algorithms do not make this assumption, and generate sequences starting from the all-Xs state.

Circuit	Fault Coverage		CPU time		# Vectors	
	%		[s]			
	GATTO	GATTO+	GATTO	GATTO+	GATTO	GATTO+
s1196	98,71	99,52	339	118	5202	1091
s1238	94,02	94,46	349	213	4672	1905
s1423	83,5	96,70	557	362	3394	2855
s1488	90,44	96,77	77	104	631	891
s1494	84,79	96,02	122	72	912	886
s5378	71,19	77,34	556	420	1132	832
s9234	5,85	5,85	72	19	220	8
s13207	20,12	20,45	299	280	166	272
s15850	5,5	5,56	111	109	18	101
s35932	84,27	89,76	1474	2958	563	1015
s38417	16,58	17,94	6004	5724	583	593
s38584	39,29	49,66	11957	6923	2478	3249

Tab. 2: comparison between GATTO and GATTO+.

Taking into account the two points above, the results in Tab. 3 show that:

- GATTO+ is able to reach higher Fault Coverage figures in all cases but 4 when HITEC is considered; the figures of GATTO+ are always better when the tool of [RPGN94] is analyzed;
- the CPU times required by GATTO+ are lower than the ones required by HITEC for all the circuits but S1196 and S1238; on the other side, GATTO+ is always faster than the method in [RPGN94]. For most circuits, we believe that the speed-up ratios are greater than any reasonable factor due to the different hardware platforms.

5. Conclusions

We described some advanced techniques to improve the effectiveness of a GA-based ATPG like GATTO [PRSR94]. They fully exploit the powerfulness of Evolutionary Computation by removing some weakness points concerning the cross-over operator, the ability to determine the optimal sequence length, and the fault excitation phase.

Experimental results demonstrate that the new techniques are able to significantly improve the performance of GATTO in terms of Fault Coverage and CPU times. We also compared them with the results of a state-of-the-art topological algorithm and with the ones of another GA-based ATPG algorithm.

As a main contribution, this paper experimentally demonstrates that a carefully tuned GA-based ATPG algorithm is able to provide better results than any other approach: in fact, symbolic techniques, although faster on the small circuits, do not work with the large ones, while topological techniques, although able to identify untestable faults, are generally slower.

Circuit	FC (%)			CPU Time [s]		
	HITEC	*[RPGN94]*	*GATTO+*	*HITEC*	*[RPGN94]*	*GATTO+*
S208	63,70		69,77	29		22
S298		85,94	88,64		363	23
S344	95,9	96,20	98,46	4785	351	1
S349	95,7	95,71	97,89	3135	350	1
S382		86,97	94,74		535	735
S386	81,7	76,82	81,77	82	207	27
S400		85,68	93,40		567	212
S420	41,6		47,44	2716		55
S444		85,44	92,41		630	219
S510	0		100,00	6		58
S526		74,95	83,24		858	214
S526n			84,09			612
S641	86,5	86,51	87,31	777	494	22
S713	81,9	81,93	82,62	124	565	26
S820	95,4	60,71	94,12	1430	804	1287
S832	93,6	61,95	67,43	2270	738	80
S838			35,36			121
S953	8,2		99,07	937		28
S1196	99,7	99,19	99,52	53	696	118
S1238	94,6	94,02	94,46	201	960	213
S1423		80,66	96,70		10188	362
S1488	97	93,67	96,77	13348	1512	104
S1494	96,4	94,02	96,02	6981	1392	72
S5378		68,98	77,34		21888	420
S9234	0,2		5,85	125		19
S13207			20,45			280
S15850	0,7		5,56	1704		109
S35932	88,8	89,55	89,76	59063	378720	2958
S38417			17,94			5724
S38585			49,66			6923

Tab. 3: results of HITEC, [RPGN94], and GATTO+.

6. References

[ACAg88] V.D. Agrawal, K.-T. Cheng, P. Agrawal, "CONTEST: A Concurrent Test Generator for Sequential Circuits," *Proc. 25th Design Automation Conf.*, 1988, pp. 84-89

[CHSo93] H. Cho, G.D. Hatchel, F. Somenzi, "Redundancy Identification/Removal and Test Generation for Sequential Circuits Using Implicit State Enumeration," *IEEE Trans. on CAD/ICAS*, Vol. CAD-12, No. 7, pp. 935-945, July 1993

[Gold89] D.E. Goldberg, "Genetic Algorithms in Search, Optimization, and Machine Learning," Addison-Wesley, 1989

[NiPa91] T. Niermann, J.H. Patel, "HITEC: A Test Generator Package for Sequential Circuits," *Proc. European. Design Aut. Conf.*, 1991, pp. 214-218

[PRSR94] P. Prinetto, M. Rebaudengo, M. Sonza Reorda, "An Automatic Test Pattern Generator for Large Sequential Circuits based on Genetic Algorithms," *Proc. Int. Test Conf.*, 1994, pp. 240-249

[RPGN94] E.M. Rudnick, J.H. Patel, G.S. Greenstein, T.M. Niermann, "Sequential Circuit Test Generation in a Genetic Algorithm Framework," *Proc. Design Automation Conf.*, 1994, pp. 698-704

[SSAb92] D.G. Saab, Y.G. Saab, J. Abraham, "CRIS: A Test Cultivation Program for Sequential VLSI Circuits," *Proc. Int. Conf. on Comp. Aided Design*, 1992, pp. 216-219