Enhancing GA-based Sequential ATPG through Guided Crossover

MICHAEL DIMOPOULOS, PANAGIOTIS LINARDIS
Department of Informatics
Aristotle University of Thessaliniiki
GR-54006 Thessaloniki
GREECE

Abstract: Test Generation for digital circuits using deterministic methods is an NP-complete problem and for that reason Genetic Algorithms have been recently investigated as an alternative to test generation. In this paper a Genetic Algorithm “GATPG” is presented for generating test sequences for sequential circuits. The aim is to produce compact test sequences that attain high fault coverage. In order to fulfill these requirements a non-uniform selection probability for crossover is employed combined with an aging factor of variable-length sequences and a two-phase fitness function. Candidate test sequence evaluation is accomplished with a 3-valued fault simulator, allowing the circuit to start from an arbitrary (unknown) state. Experimental results with respect to the ISCAS’89 benchmarks are presented to show the viability of the proposed approach.

Key-Words: Genetic Algorithms, Test Generation, Sequential Digital Circuits, ATPG.

1 Introduction

Test generation of combinational digital circuits is a highly complex problem, typically it is NP-complete [3]. For sequential digital circuits the testing problem, which is the subject of this paper, is even more complex.

Sequential circuits, for testing purposes, are usually modeled as iterative arrays of combinational circuits and test generation techniques developed for combinational circuits are extended to handle sequential circuits. Mainly two approaches to testing are followed:

- Deterministic methods [3, 4, 6] that use branch and bound techniques with the aid of heuristics to prune the search space. Due to the vast search space these techniques are often unable to handle large sequential circuits[3, 12].
- Simulation-based methods [3] which are trial-and-error methods. They generate random test vectors which are evaluated by fault simulation according to a “cost” function. Best trial vector is selected and added to the test sequence.

In the “simulation” class of methods belong the Genetic Algorithm (GA) simulation based techniques [5, 9, 10, 11, 12]. In GA, initial random test sequences guided by genetic operators [1, 2] evolve to highly fit solutions.

Recently, deterministic methods combined with GAs were devised that achieve very good results [7, 8].

In this paper we propose a GA simulation-based method, called GATPG, which contains a two phase fitness function and puts emphasis on shorter, more compact, test sequences by introducing a non-uniform crossover operator.

The paper is organized as follows: In section 2 is presented the testing problem for sequential circuits. In section 3 the structure of the GATPG algorithm is analyzed. In section 4 experimental results are given, supporting the potential of the proposed method.

2 Problem Formulation

A synchronous sequential circuit can be considered to be a Finite State Machine M defined as a quintuple \( M = (I, O, S, \delta, \lambda) \) where \( I \) is the set of input vectors, \( O \) is the output set, \( S \) is the set of states, \( \delta \) is the next state function and \( \lambda \) is the output function. A stuck-at fault \( f \) transforms the machine \( M \) into a machine \( M_f = (I, O, S, \delta_f, \lambda_f) \). For a given list of stuck-at faults \( F = \{ f_1, f_2, \ldots, f_n \} \) the test generation problem is to find a sequence of input vectors \( V \), called Test Sequence, that detects the faults in \( F \), that is when \( V \) is applied to each \( M_f \) will produce different responses from \( M \).

It is assumed, here, that the initial state of \( M \) and \( M_f \) is unknown, as is usually the case.

In this work the test sequences \( V \) are generated with the help of genetic algorithms.
3 GATPG ALGORITHM

The algorithm is shown in Fig.1. Starting with an initial population of randomly produced sequences of test vectors (Create_random_population) each sequence (individual) is evaluated (Evaluate_fsm) by performing fault simulation to find the faults which are detected, propagated to flip-flops (activated), and state and output differences between M and M_f. These results are used to determine the fitness value of each sequence, as will be explained in section 3.2. The crossover operation (cross_over) applied here follows a non-uniform probability of candidate selection.

3.1 Individuals
An individual is a binary-coded 2-dimensional bit string corresponding to a sequence of input vectors. The length of this bit sequence is L=ninputs*no_of_test_vectors. A characteristic of the GATPG algorithm is that it uses variable-length sequences with respect to the number of test vectors. Starting with an initial sequence length of 5 vectors, sequence length progressively increases every 3 generations by adding at the end one randomly generated vector. This approach has two advantages: lower simulation cost and more efficient test sequences. The simulator, developed by the authors, is a PROOFs-based [4] 3-valued fault simulator. The third value X emulates the unknown initial state of the circuit.

3.2 Fitness Function
The results from the simulation are used to rank the individuals according to certain evaluation rules, which form the so called fitness function. Our fitness function is complex and has the form:

\[
\text{fitness} = \begin{cases} 
  f_1 & \text{iff } (\text{ngen}/\text{MAX_GENERATIONS}) \leq 0.25 \\
  f_2 & \text{else} 
\end{cases}
\]

where:

\[
f_1 = 20 \cdot R_1 + R_3 \cdot R_2
\]

\[
f_2 = 20 \cdot R_1 + R_3 + R_4 \cdot R_5 \cdot R_2
\]

where: \( R_i \) is a value denoting how “close” the individual is to satisfying rule \( i \).

Create_random_population.
For each individual
Evaluate_fsm(individual);
end for
Sort_Population(); /* with fit. value descending*/
ngen=0; /* num of generation */
do {
  ******* crossover *******
  for (j=0,i=0; i < ncross; j +=2, i++)
  { 
cross_over(Individual[], Individual[j+1],
           child1, child2);
  Evaluate_fsm(child1);
  Evaluate_fsm(child2);
  update_age_of(child1);
  update_age_of(child2);
  ******* mutations *******
  for (i=0; i < nmut; i ++) 
  { 
    mutation(Individual[], child);
    mutation(Individual[], child1);
    Evaluate_fsm(child);
    Evaluate_fsm(child1);
    update_age_of(child1);
    update_age_of(child2);
  }
  Sort_Population();
  if ((ngen % 3) == 0)
  { 
    Expand_sequence(EXPAND_STEP);
    Evaluate_fsm(Individual[0]);/* check best*/
  }
  ngen++;
} while (ngen < MAX_GENERATIONS);

Fig.1 The GATPG algorithm
where:

\[ C_{FF} = \sum_{k=1}^{\text{seq len}} \sum_{j=1}^{\text{Total Faults}} \sum_{i=1}^{\text{num FF}} G_{ij} \]

\[ C_{OUT} = \sum_{k=1}^{\text{seq len}} \sum_{j=1}^{\text{Total Faults}} \sum_{i=1}^{\text{num gates}} G_{ij} \]

\[ G_{ij} = \begin{cases} 1 \text{ iff } c_{ft, i}^{\text{good}} 
eq c_{ft, j}^{\text{good}} \\
0 \text{ else } \end{cases} \]

In this fitness function emphasis is given in the maximization of detected faults while favouring smaller test sequences. A two-phase function is used. Because in practice we have "easy" and "difficult" to test faults [5] we start with \( f_1 \) and after a number of generations switch to a different function \( f_j \). As was mentioned, test sequences are extended every three generations by appending a new randomly generated vector. In order to escape stagnation an aging factor is incorporated so that offsprings having the same fitness value with their parents are given higher precedence in the next generation.

3.3 Genetic Operators

The creation of an offspring is accomplished with the help of a crossover operator that interchanges the bits of two individuals. The crossover operation applied here is a one-point crossover [1, 2].

Two crossover operators are used:

- Standard cut-point selection is used with uniform probability.
- A square root probability distribution function is used to direct the cut-point selection towards the end of the test sequence, thus giving emphasis on optimizing the tail of the sequence as new vectors are appended to it.

To ensure diversity, mutation is applied to the best 2 individuals in the population. Two different mutation operations are used:

- Single-bit mutation: it randomly selects a bit and complements it.
- Multi-bit-mutation: it randomly selects a vector and for every bit within it a choice is made with a probability of \( \frac{1}{2} \) whether to keep its value or to complement it.

4 Experimental Results

The efficiency of the GATPG algorithm, implemented in C, was measured by using some of the ISCAS’89 benchmark circuits [13]. The main characteristics of these benchmark circuits are given in Table 1, where \( i, o, ff, gates \) denote the number of inputs, outputs, flip-flops, gates. Total detected are the number of faults that can be detected and against which the results are judged.

<table>
<thead>
<tr>
<th>a/a</th>
<th>circuit</th>
<th>i/o/ff/gates</th>
<th>Faults</th>
<th>Total Detected</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>s298</td>
<td>3/6/14/119</td>
<td>308</td>
<td>265</td>
</tr>
<tr>
<td>2</td>
<td>s344</td>
<td>9/11/15/160</td>
<td>342</td>
<td>329</td>
</tr>
<tr>
<td>3</td>
<td>s349</td>
<td>9/11/15/161</td>
<td>350</td>
<td>335</td>
</tr>
<tr>
<td>4</td>
<td>s382</td>
<td>3/6/21/158</td>
<td>399</td>
<td>364</td>
</tr>
<tr>
<td>5</td>
<td>s386</td>
<td>7/7/6/159</td>
<td>384</td>
<td>314</td>
</tr>
<tr>
<td>6</td>
<td>s400</td>
<td>3/6/21/164</td>
<td>426</td>
<td>384</td>
</tr>
<tr>
<td>7</td>
<td>s444</td>
<td>3/6/21/181</td>
<td>474</td>
<td>424</td>
</tr>
</tbody>
</table>

Table 1. ISCAS’89 circuits

For the GATPG we used the following parameter values.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>POPULATION</td>
<td>16</td>
</tr>
<tr>
<td>MAX_GENERATIONS</td>
<td>300</td>
</tr>
<tr>
<td>PCROSSOVER</td>
<td>0.6</td>
</tr>
<tr>
<td>PMUTATION</td>
<td>0.2</td>
</tr>
<tr>
<td>EXPAND_STEP</td>
<td>1</td>
</tr>
</tbody>
</table>

In Tables 2 and 3 we present results regarding uniform and square-root (sqrt) probability of cut-point selection. Det., Vec. and Time represent the number of detected faults, of test sequence length and of the generations required to achieve these sequences.

<table>
<thead>
<tr>
<th>Circuit</th>
<th>Det.</th>
<th>Vec.</th>
<th>Time (gen)</th>
</tr>
</thead>
<tbody>
<tr>
<td>s298</td>
<td>264</td>
<td>79</td>
<td>242</td>
</tr>
<tr>
<td>s344</td>
<td>327</td>
<td>56</td>
<td>266</td>
</tr>
<tr>
<td>s349</td>
<td>332</td>
<td>51</td>
<td>295</td>
</tr>
<tr>
<td>s382</td>
<td>316</td>
<td>88</td>
<td>284</td>
</tr>
<tr>
<td>s386</td>
<td>254</td>
<td>39</td>
<td>284</td>
</tr>
<tr>
<td>s400</td>
<td>329</td>
<td>86</td>
<td>293</td>
</tr>
<tr>
<td>s444</td>
<td>360</td>
<td>91</td>
<td>258</td>
</tr>
</tbody>
</table>

Table 2. Uniform selection probability

| Sum     | 2182  | 490  | 1922       |

Table 3. Total selection probability
As we see from the above results sqrt selection probability (Table 3) is better than uniform selection probability (Table 2) because, in order of importance, on the average: (a) it detects more faults, (b) in relatively small sequences and (c) in shorter time.

In Table 4 we compare our results with results from [5, 6]. were \( f_c \) and \( \text{Vec.} \) are the fault coverage (detected faults to total detectable faults) and the test sequence length.

As we see in Table 4 our results (sqrt, uniform) regarding fault coverage for the first three circuits are nearly the same with those of the others. For the remaining circuits although our fault coverage is lower (average fault cov. 0.932 (sqrt) compared to 0.972 [5]) the size of our test sequences is 2.5 times smaller than [5] and 22.8 times smaller than HITEC.

We must note that HITEC is a state-of-the-art deterministic test pattern generator that achieves high fault coverage but requires long CPU time to achieve satisfactory results. The method of [5] belongs to the same category with our method. There is no comparison with methods from [10, 12] because they assume that the circuit starts from a given initial state instead of the more general case of an unknown (arbitrary) one.

<table>
<thead>
<tr>
<th>Circuit</th>
<th>Det.</th>
<th>Vec.</th>
<th>Time (gen)</th>
</tr>
</thead>
<tbody>
<tr>
<td>s298</td>
<td>265</td>
<td>93</td>
<td>292</td>
</tr>
<tr>
<td>s344</td>
<td>329</td>
<td>64</td>
<td>232</td>
</tr>
<tr>
<td>s349</td>
<td>335</td>
<td>65</td>
<td>295</td>
</tr>
<tr>
<td>s382</td>
<td>323</td>
<td>94</td>
<td>276</td>
</tr>
<tr>
<td>s386</td>
<td>275</td>
<td>57</td>
<td>238</td>
</tr>
<tr>
<td>s400</td>
<td>337</td>
<td>85</td>
<td>232</td>
</tr>
<tr>
<td>s444</td>
<td>375</td>
<td>85</td>
<td>278</td>
</tr>
</tbody>
</table>

| Sum     | 2239 | 543  | 1843       |

Table 3. Sqrt selection probability

<table>
<thead>
<tr>
<th>Circuit</th>
<th>sqrt</th>
<th>uniform</th>
<th>HITEC [5, 6]</th>
<th>[5]</th>
</tr>
</thead>
<tbody>
<tr>
<td>s298</td>
<td>1.00</td>
<td>0.93</td>
<td>0.96</td>
<td>79</td>
</tr>
<tr>
<td>s344</td>
<td>1.00</td>
<td>0.94</td>
<td>0.97</td>
<td>36</td>
</tr>
<tr>
<td>s349</td>
<td>1.00</td>
<td>0.91</td>
<td>0.97</td>
<td>46</td>
</tr>
<tr>
<td>s382</td>
<td>0.88</td>
<td>0.68</td>
<td>0.97</td>
<td>439</td>
</tr>
<tr>
<td>s386</td>
<td>0.76</td>
<td>0.57</td>
<td>0.97</td>
<td>31</td>
</tr>
<tr>
<td>s400</td>
<td>0.86</td>
<td>0.63</td>
<td>0.97</td>
<td>4240</td>
</tr>
<tr>
<td>s444</td>
<td>0.86</td>
<td>0.63</td>
<td>0.97</td>
<td>2240</td>
</tr>
</tbody>
</table>

| fault cov. | 0.932 | 0.909 | 0.995 | 0.972 |
| test seq.  | 543   | 490   | 12376 | 1341  |

Table 4. Comparison with results from [5, 6].

4 Conclusion

A GA-based test generation algorithm is presented which has some unique features. Apart from the fitness function used here and the aging of individuals, crossover is enhanced with a non-uniform crossover-selection probability. This "directed" cut-point selection in crossover performs better than the classical one with uniform probability cut-point selection as is evident from experimental results presented here. Although the preliminary results that were presented are quite competitive with those of others the GATPG algorithm may be further improved by adding more circuit specific knowledge in the fitness function and elaborating on GA-operators.

References:


