Evolutionary Algorithms for State Justification in Sequential Automatic Test Pattern Generation

Aiman H. El-Maleh
Department of Computer Engineering, King Fahd University of Petroleum and Minerals, Dhahran, Saudi Arabia

Sadiq M. Sait
Department of Computer Engineering, King Fahd University of Petroleum and Minerals, Dhahran, Saudi Arabia

Syed Z. Shazli
Department of Computer Engineering, King Fahd University of Petroleum and Minerals, Dhahran, Saudi Arabia

Jun 18, 2003

Abstract. Sequential circuit test generation using deterministic, fault-oriented algorithms is highly complex and time consuming. New approaches are needed to enhance the existing techniques, both to reduce execution time and improve fault coverage. Evolutionary algorithms have been effective in solving many search and optimization problems. A common search operation in sequential Automatic Test Pattern Generation is to justify a desired state assignment on the sequential elements. State justification using deterministic algorithms is a difficult problem and is prone to many backtracks, which can lead to high execution times. In this work, a hybrid approach which uses a combination of evolutionary and deterministic algorithms for state justification is proposed. A Genetic Algorithm is employed, to engineer state justification sequences vector by vector. This is in contrast to previous approaches where GA is applied to the whole sequence. The proposed method is compared with previous GA-based approaches. Significant improvements have been obtained for ISCAS benchmark circuits in terms of state coverage and CPU time. Furthermore, it is demonstrated that the state-justification sequence generated, helps the ATPG in detecting a large number of hard-to-detect faults.

Keywords: Sequential ATPG, Genetic Algorithms, Evolutionary strategies, State Justification

1. Introduction

Testing of integrated circuits is an important area which nowadays, accounts for a significant percentage of the total design and production costs of ICs. In order to obtain acceptably high quality tests, design for testability (DFT) techniques are in use [12]. One such technique is *Full-scan design*, which reduces the sequential test generation problem to a less difficult combinational test generation problem. In large circuits however, this technique adversely affects the test application time as all

the test vectors have to be scanned in and out of the flip-flops. *Partial-scan design* involves scanning a selected set of memory elements. A sequential test generator is necessary in case of a partial scan or no-scan design [13]. In this work, we assume either no-scan or partial scan designs. Generating test sequences for synchronous sequential circuits is a more challenging problem than that of combinational circuits for several reasons. They may be itemized as follows:

- Each fault must be first excited by presenting a given value not only on the primary inputs but also on the flip-flop outputs.
- The assignments on the flip-flops have to be justified backward in time, requiring a state justification sequence.
- The difference existing on the fault source between the values of the fault-free and the faulty circuit must be then propagated to the primary outputs. This is accomplished normally in the next time frames, requiring a fault propagation sequence.
- The lengths of justification and propagation sequences are not known before hand as they depend on the starting state and the Finite State Machine (FSM).
- The presence of invalid states and signal dependencies across multiple time-frames, results in a large number of backtracks.
- Untestable faults require a large amount of time to be identified due to the large search space.

For sequential circuit ATPG, the worst-case search space is 9^m, where m is the number of flip-flops. This exponential search space makes exhaustive ATPG search computationally impractical for large sequential circuits [13]. In recent years, one of the main goals of researchers was to develop effective algorithms for sequential circuit test pattern generation [7].

State Justification: A common operation in sequential ATPG, is to justify a desired state assignment on the memory elements. Both deterministic and simulation-based algorithms have been used for state justification. In simulation-based approaches, the search proceeds in the forward direction only. Hence, there are no backtracks and state justification is easier as compared to deterministic ATPGs. The main drawback of simulation-based approaches lies in their inability to detect untestable faults [21].

Genetic Algorithms (GAs) have been effective in solving many search and optimization problems [26]. Several approaches to test generation

using genetic algorithms have been proposed in the past [2] [5] [20] [21] [10] [7] [24] [22] [3] [6] [25] [23] [8] [14] [4] [9] [11]. A major difference in various GA-based approaches lies in the way the fitness (closeness to the optimal solution) is computed. Some techniques use logic simulation for evaluation of candidate vectors or sequences, while other techniques use fault simulation. In addition, there are certain other techniques which target different objectives in various phases of test generation. These techniques typically, use both logic and fault simulation in evaluating candidate sequences. The main advantage of GA-based ATPGs, as compared to other approaches, is their ability to cover a larger search space in lower CPU time. Traversing a larger number of states, improves the fault coverage of sequential ATPGs as mentioned in [16].

In [10], Genetic Algorithms have been used for state justification. The state justification sequences were genetically engineered. The length of the sequence was predetermined and fixed. It was a function of the structural sequential depth of the circuit, where sequential depth is defined as the minimum number of flip-flops in a path between the primary inputs and the farthest gate. In case of feed-back loops however, the structural sequential depth may not give a correct estimate of the number of vectors required for justifying a given state. Thus, if a state requires longer justification sequence, it will not be justified by this approach. The approach also does not take into account the quality of intermediate states reached and evaluates a sequence only on the basis of the final state reached.

In this work, a hybrid framework for state justification is proposed. Both deterministic and Genetic-based approaches are used. We use an incremental approach in our Genetic-based state justification. State justification sequences are genetically engineered vector by vector. Thus, the length of the sequences is not restricted or fixed.

The remainder of this paper is organized as follows: The proposed genetic-based state justification technique is presented in Section 2. Experimental results are given in Section 3. Section 4 concludes the paper.

2. Genetic-based State Justification

State justification is the most difficult task in sequential ATPG. Storing the complete state information for large circuits is impractical. Similarly, keeping a list of sequences capable of reaching each reachable state is also infeasible. In [10] and [21], deterministic algorithms were used for fault excitation and propagation, and a GA was used for state justification. Sequences were evolved over several generations. The fitness of each individual was a measure of how closely the final state reached matched the desired state. A chromosome was represented by a

sequence of vectors. Candidate sequences were simulated starting from the last state reached at the end of the previous test sequence. The objective was to engineer a test sequence that justified the required state. If a sequence was found which justified the required state, the sequence was added to the test set.

In this work, we genetically engineer state justification sequences vector by vector. Individual vectors are represented by chromosomes in the population and genetic operators are applied at individual bit positions. Deterministic ATPG is run for every target fault. First, the fault is activated and propagated to a primary output. Next, state justification is attempted. If the required state is justified by the deterministic ATPG, then the derived sequence is fault simulated and all detected faults are dropped from the fault list. Otherwise, our GA-based algorithm attempts to justify the required state. The best sequence obtained in a given number of generations is viewed as a partial solution. The state reached by this sequence could be close to the required state, which could help the ATPG in justifying it in the next attempts. The generated sequence is then fault simulated and the detected faults are dropped. A block diagram of the proposed state-justification methodology is shown in Figure 1.

Encoding of the chromosome: In this work, a binary encoding is used. A chromosome is represented by a single vector. Each bit of a vector corresponds to the value at a primary input.

Fitness Function: In Genetic Algorithms, a solution is considered to be better than another if its fitness is higher. We logic simulate each vector to get the state reached. This state is compared with all the flip-flop assignment values of the target state. The fitness $f(v_i)$ of a vector v_i is computed as follows:

$$f(v_i) = \frac{m(s_i, s_j)}{B(s_j)}$$

where s_i is the state reached by vector v_i, s_j is the target state and $m(s_i, s_j)$ are the number of matching specified bits in s_i and s_j. $B(s_j)$ gives the number of specified bits in s_j (i.e., those which are not don't cares).

Crossover and Mutation: In this work, one-point uniform crossover is used. In one-point uniform crossover, a random cut-point is selected. Each of the two parents are divided into two parts at this random cut point. We generate an offspring by catenating the segment of one parent to the left of the cut point with the segment of the second parent to

the right of the cut point. *Mutation* introduces new characteristics in the offspring by randomly changing values of some genes. In this work, mutation corresponds to flipping a randomly selected bit.

Selection for the next generation: A generation corresponds to an iteration of GA where parents are selected for crossover and offsprings are created. A constant number of individuals are selected from the offsprings for the new generation. The new population thus consists of both members from the current generation and the offsprings created. In this work, three replacement strategies have been experimented with.

$(n+1)$ *selection strategy:* In this strategy, one chromosome is changed in every generation. A crossover is performed on two selected parents. A child replaces the worst member of the previous generation, if its fitness is higher. Hence, the best $n-1$ members are selected for the new generation from a population of n.

Random Elitist strategy: We produce n offsprings, by performing $n/2$ crossovers on a population of n chromosomes. The best $n/2$ members of both the offsprings and the original population are transferred to the next generation. The remaining $n/2$ members of the new generation are selected randomly from the leftover chromosomes.

Roulette Elitist strategy: This strategy is the same as Random Elitist strategy except that the second half of the members of the new generation are selected based on a roulette wheel mechanism. This gives an advantage to the relatively more fit members of the population to be transferred to the next generation.

A flowchart of the proposed GA-based state justification algorithm is shown in Figure 4. The algorithm works as follows.

A target state is initially selected from the list of desired states. Genetic Algorithm is run for a fixed number of iterations. If the target state is reached by any of the chromosomes, that chromosome is appended to the final state justification sequence and the next target state is picked. If however, the target state is not reached, the fittest chromosome found is picked, and the state reached by the chromosome is added in a *tabu list.* Tabu List [26], is used to prevent the algorithm from visiting recently visited states. The chromosome is appended to the final sequence. The next fit chromosome is chosen, if the state reached by the fittest one is already in the Tabu List. If the states reached by all the chromosomes in the population are present in the tabu list, the algorithm backtracks to a previously visited state and the last vector is removed from the final state justification sequence. The algorithm

stops further searching for a target state after a specified number of backtracks. Another stopping criteria is also used. If the fitness of the currently visited state, $fit(s)$, is less than the average fitness of the last $Nlimit$ states, $fit(Nlimit)$, the algorithm stops further searching of the desired state; otherwise the search is continued. Hence, at least $Nlimit$ number of states are traversed before the algorithm gives up the search for a desired state. When a sequence is generated by the algorithm for a target state, the states reached by the sequence are compared with the list of desired states. All the desired states reached by the sequence are removed from the list of target states. This is done to prevent searching again for those states which have been already reached while searching for some other target state.

3. Experimental Results and Discussion

In this work, experiments have been conducted on ISCAS89 benchmark circuits [1]. The circuits used are those for which the deterministic ATPG HITEC, was unable to detect the faults after exhausting a back-track limit of $1,000,000$. In addition, four re-timed circuits for which HITEC required a large amount of CPU time [17], are considered. Table I lists the number of primary inputs, primary outputs, and D flip-flops of the circuits used in this work.

In this section, the proposed genetic-based state justification procedure is evaluated.

A list of target states is obtained for each circuit as follows:

- A deterministic test pattern generator HITEC [19] is stretched to a backtrack limit of 10^6 to identify the redundant faults.
- The aborted faults are taken and are converted to their full-scan equivalents
- HITEC then produces a test for each of these faults.
- The required state is then relaxed to produce a partially specified state using PROOFS [18].
- The desired states are merged if they are compatible.

Table II lists the target states obtained for each of the circuits.

The logic simulator HOPE [15], is used to compute the fitness of chromosomes. Three replacement policies have been experimented with. The results of the simulations carried out using these three replacement policies are shown in Table III.

In Table III, the number of states reached (SR) and the time taken to reach those states are given for each replacement strategy described above. It was observed that the $(n+1)$ replacement strategy was the best in terms of execution time. It also reached a comparable number of states for most of the circuits. This strategy changes only one member of the previous generation and hence the number of operations in one generation of $(n+1)$ replacement strategy requires less time as compared to other strategies. Moreover, changes in the characteristics of the population do not occur as abruptly as in the other two schemes. Figure 3 shows the average and best fitness of the population against the number of generations for one of the target states that is justified by the algorithm using the $(n+1)$ replacement strategy. It can be seen that the average fitness increases monotonically with the number of generations. This is due to the fact that we are always preserving the best chromosome in each generation. One-point crossover was used with a probability of 1 and mutation rate was kept at 0.01. A roulette-wheel selection scheme, as given in [20], was used as it gave the best results.

State traversal for one of the target states of the circuit s1423 that has been reached by the algorithm is shown in Figure 4. It can be seen that better states are reached in terms of the Hamming distance as the algorithm runs for more iterations. Less fit states are reached if a better state cannot be reached due to the Tabu restriction. Moreover, the best state among all alternatives is chosen, even if that state is worse than the current state. This helps in avoiding the local minima.

In Figure 5, state traversal for one of the unreached states is shown. In case the desired state is not reached, the sequence leading to the best reached state is generated. It can be observed from the figure, that the quality of states reached is better in terms of Hamming distance as the algorithm runs for more iterations.

In order to determine the effect of the parameters on the results, extensive experiments were run to perform a sensitivity analysis of each of the parameters. Based on the experiments performed and the observed results, certain parameters were found to perform better than others for most of the circuits.

A population size of 32 gave good quality results in comparable time. 400 generations of the algorithm gave solutions of good quality. A backtrack limit of 10 and Tabu List size of 15 gave good results. $Nlimit$ is suggested as 1.5 times the number of flip-flops in the circuit. The algorithm was run with the parameters suggested above for all circuits. Results are shown in Table IV. In Table V, the best results obtained for each of the circuits are presented. The parameters are listed for every circuit. It can be seen from Table IV and Table V, that

the suggested parameters gave good quality results when compared with the best results obtained for each circuit.

The proposed state justification technique, which uses a GA for traversing from a state to a state, has been compared with the one proposed in [10][21]. In [10], GA has been used for state justification and sequences are genetically engineered. GA has been applied on a sequence of vectors as opposed to individual vectors in this case.

The parameters proposed in [10] were 32 chromosomes and 8 generations. The number of vectors in each chromosome was 4 times the sequential depth of the circuit.

The experiments were run on SUN ULTRA 10 stations and the results of comparing the two state justification techniques are shown in Table VI.

The first column in the table shows the circuit name. The states reached and CPU time obtained by the proposed algorithm are given in the next two columns. For comparison purposes, the algorithm proposed in [10] was run for several number of generations and the results are shown in the next columns.

It can be observed from the results that the number of desired states reached by the proposed technique is more than those reached by the technique used in [10] for all the circuits. Many of the target states may correspond to redundant faults as the ATPG aborted the search of a sequence for detecting the target fault after a given number of backtracks. Hence, those states are not reached by the algorithm. Furthermore, the proposed technique reached a higher number of states than [10] in all the circuits even when the latter was run for a greater amount of CPU time.

In order to verify the effectiveness of the generated state justification sequences in detecting hard-to-detect faults, the sequences were seeded to the deterministic test pattern generator HITEC [19]. HITEC makes use of previously visited states while doing state justification. The faults detected by an initial run of HITEC with 1000000 backtracks were removed from the fault list. The results are shown in Table VII.

It can be observed that a large number of hard-to-detect faults are detected when HITEC is seeded with the state justification sequence obtained by the proposed strategy. The number of faults detected are significantly higher than the faults detected when the ATPG is seeded with the state justification sequences generated by the technique proposed in [10]. Apart from justifying more states, the porposed technique takes advantage of the partial justification sequences generated.

4. Conclusion

In this work, a hybrid approach is proposed which uses a combination of evolutionary and deterministic algorithms for state justification. Genetic Algorithms (GAs) are used for generating sequences that will help the Automatic Test Pattern Generator (ATPG) in detecting more faults by reaching specific states. A new state justification technique based on GA is proposed which engineers the sequence vector by vector. In previous approaches, GA has been applied to the whole sequence. The previous approaches fail to justify many hard-to-reach states because of fixed-length sequences. Moreover, they evaluate a chromosome only on the basis of the final state reached. In this work, dynamic length sequences are used and the fitness measure takes into account all the states reached by the sequence. The approach has been compared with previous approaches and improvements in reached states and fault coverage have been demonstrated.

Acknowledgment: The authors thank King Fahd University of Petroleum and Minerals, Dhahran, Saudi Arabia, for support.

References

1. Brglez, F., D. Bryan, and K. Kozminski: 1989, 'Combinational profiles of sequential benchmark circuits'. *International Symposium on Circuits and Systems* pp. 1929–19347.
2. Corno, F., P. Prinetto, M. Rebaudengo, and S. Reorda: 1996a, 'A Parallel Genetic Algorithm for aoutomatic generation of test sequences for digital circuits'. In: *International Conf. on High Performance Computing and Networking, Belgium.*
3. Corno, F., P. Prinetto, M. Rebaudengo, and S. Reorda: 1996b, 'GATTO: A genetic algorithm for automatic test pattern generation for large synchronous sequential circuits'. *IEEE Transactions on CAD of Integrated circuits and systems* **15**, 991–1000.
4. Corno, F., P. Prinetto, M. Rebaudengo, S. Reorda, and E. Veiluva, 'A portable ATPG tool for parallel and distributed systems'. In: *VLSI Test Symposium.* pp. 29–34.
5. Corno, F., M. Rebaudengo, and S. Reorda: 1998, 'Experiences in the use of evolutionary techniques for testing digital circuits'. In: *Application and Science of Neural Networks, Fuzzy Systems and Evolutionary computation, SPIE.*
6. F.Corno, P.Prinetto, M.Rebaudengo, S. Reorda, and M.Violante: 1997, 'Exploiting logic simulation to improve simulation-based sequential ATPG'. In: *Sixth IEEE Asian Test Symposium, Arta, Japan.*
7. Hsiao, M. S.: 1997, 'Use of Genetic Algorithms for testing sequential circuits'. *Ph.D. Dissertation, UIUC.*
8. Hsiao, M. S., E. M. Rudnick, and J. H. Patel: 1996, 'Alternating strategies for sequential circuit ATPG'. In: *European Design and Test Conference.* pp. 368–374.

9. Hsiao, M. S., E. M. Rudnick, and J. H. Patel: 1997, 'Sequential circuit test generation using dynamic state traversal'. In: *European Design and Test Conference*. pp. 22–28.
10. Hsiao, M. S., E. M. Rudnick, and J. H. Patel: 1998, 'Application of genetically engineered finite-state-machine sequences to sequential circuit ATPG'. *IEEE Transactions on CAD of Integrated circuits and systems* **17**, 239–254.
11. Hsiao, M. S., E. M. Rudnick, and J. H. Patel: 2000, 'Dynamic state traversal for sequential circuit test generation'. *ACM Transactions on Design Automation of Electronic Systems* **5**.
12. Kim, Y. C. and K. K. Saluja: 1998, 'Sequential test generators: past, present and future'. *INTEGRATION, the VLSI journal* **26**, 41–54.
13. Konijnenburg, M. H., J. T. van der Linden, and A. J. van de Goor: 1997, 'Sequential test generation with advanced illegal state search'. In: *International Test Conference*.
14. Krishnaswamy, D., M. S. Hsiao, V. Saxena, E. M. Rudnick, and J. H. Patel: 1997, 'Parallel Genetic Algorithms for simulation-based sequential circuit test generation'. In: *IEEE VLSI Design Conference*. pp. 475–481.
15. Lee, H. K. and D. S. Ha: 1996, 'HOPE: An Efficient Parallel Fault Simulator for Synchronous Sequential Circuits'. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* **15**, 1048–1058.
16. Marchok, T., A. El-Maleh, W. Maly, and J. Rajski: 1995, 'Complexity of Sequential ATPG'. In: *European Design and Test Conference*. pp. 252–261.
17. Marchok, T., A. El-Maleh, W. Maly, and J. Rajski: 1998, 'A complexity analysis of sequential ATPG'. *IEEE Transactions on CAD of Integrated circuits and systems* **15**, 1409–1423.
18. Niermann, T., W. T. Cheng, and J. H. Patel: 1990, 'PROOFS: A fast memory efficient sequential circuit fault simulator'. In: *Design Automation Conf.* pp. 535–540.
19. Niermann, T. and J. H. Patel: 1991, 'HITEC: a test generation package for sequential circuits'. In: *European Test Conf.* pp. 214–218.
20. Rudnick, E. M., J. G. Holm, D. G. Saab, and J. H. Patel: 1994a, 'Application of Simple Genetic Algorithm to sequential circuit test generation'. In: *European Design and Test Conference*. pp. 40–45.
21. Rudnick, E. M. and J. H. Patel: 1996, 'State justification using Genetic Algorithms in sequential circuit test generation'. *A survey report from CRHC Univ. of Illinois, Urbana.*
22. Rudnick, E. M., J. H. Patel, G. S. Greenstein, and N. M. Niermann: 1994b, 'Sequential circuit test generation in a genetic algorithm framework'. In: *Design Automation Conference*. pp. 698–704.
23. Rudnick, E. M., J. H. Patel, G. S. Greenstein, and N. M. Niermann: 1997, 'A Genetic algorithm framework for test generation'. *IEEE Transactions on CAD of Integrated circuits and systems* **16**, 1034–1044.
24. Saab, D. G., Y. G. Saab, and J. A. Abraham: 1992, 'CRIS: A test cultivation program for sequential VLSI circuits'. In: *International Conf. on Computer-aided Design*. pp. 216–219.
25. Saab, D. G., Y. G. Saab, and J. A. Abraham: 1996, 'Automatic test vector cultivation for sequential VLSI circuits using genetic algorithms'. *IEEE Transactions on CAD* **15**, 1278–1285.
26. Sait, S. M. and H. Youssef: 1999, *Iterative Computer Algorithms with applications in Engineering: Solving combinatorial optimization problems*. IEEE Computer Society.

Table I. The benchmark circuits used.

circuit	# of PI	# of PO	# of DFF
s1423	17	5	74
s3271	26	14	116
s3384	43	26	183
s5378	35	49	179
s6669	83	55	239
scfRjisdre	27	54	20
s832jcsrre	18	19	31
s510Rjcsrre	20	7	30
s510Rjosrre	20	7	32

Table II. The number of target states obtained.

circuit	# of target states
s1423	135
s3271	45
s3384	102
s5378	524
s6669	32
scfRjisdre	267
s832jcsrre	57
s510Rjcsrre	114
s510Rjosrre	114

Table III. Comparison of the selection schemes

circuit	CHR	GEN	BT	NLimit	TLS	(n+1) SR	(n+1) Time	Random Elitist SR	Random Elitist Time	Roulette Elitist SR	Roulette Elitist Time
s1423	16	100	10	120	150	58	126	19	508	32	748
	32	100	10	120	150	64	365	31	778	49	3586
	64	100	10	120	150	64	572	49	11300	68	13704
s3271	16	800	20	225	150	20	4592	11	5023	15	11214
	32	100	20	225	150	21	6244	18	11805	20	18113
	256	100	20	225	150	21	10625	19	12976	21	109612
s3384	16	800	10	375	150	65	11849	23	14912	34	17445
	64	800	10	375	150	66	23115	51	24905	41	30023
	256	800	10	375	150	66	41225	65	68428	50	100615
s5378	16	400	10	275	150	64	25294	22	84225	45	112610
	32	400	10	275	150	113	29274	53	100324	61	141251
	64	400	10	275	150	115	34893	55	117520	61	161225
s6669	16	10	10	375	150	19	130	19	871	22	914
	16	100	10	375	150	27	503	19	5151	22	8681
	16	400	10	375	150	30	1664	22	17905	22	24668
scfRjisdre	16	100	10	40	150	18	25	17	285	26	836
	64	100	10	40	150	19	42	34	832	43	6700
	256	100	10	40	150	20	114	46	5055	50	48820
s832jcsrre	16	400	100	100	150	7	79	6	77	6	82
	256	400	100	100	150	7	190	7	1946	7	2126
	1024	400	100	100	150	9	360	8	3441	9	4956
s510Rjcsrre	16	400	10	45	150	12	14	8	120	6	140
	256	400	10	45	150	16	132	23	523	23	1208
	512	400	10	45	150	23	260	31	2340	31	5038
s510Rjosrre	16	800	10	45	150	12	92	5	233	4	305
	64	800	10	45	150	19	661	13	1171	11	2841
	256	800	10	45	150	19	2740	17	9870	19	19342

Table IV. Results obtained from the suggested parameters

circuit	Reached	Time(sec)
s1423	74	3119
s3271	21	6015
s3384	67	18314
s5378	115	31281
s6669	30	1764
scfRjisdre	48	803
s832jcsrre	8	139
s510Rjcsrre	16	163
s510Rjosrre	16	181

Table V. Best results obtained for each circuit

circuit	Chromes	Gen	BT	NLimit	TLS	Reached	Time(sec)
s1423	32	400	100	120	15	75	4212
s3271	16	400	10	225	150	21	2455
s3384	16	800	100	375	15	68	17703
s5378	32	400	10	275	15	115	31281
s6669	16	400	10	375	15	30	1466
scfRjisdre	16	800	100	40	15	48	735
s832jcsrre	1024	400	100	100	150	9	360
s510Rjcsrre	512	400	10	45	150	23	260
s510Rjosrre	64	800	10	45	150	19	661

Table VI. Comparison of the two state-justification techniques

circuit	our approach states reached	 time(sec)	approach in Hsiao98 gens	 states reached	 time(sec)	approach in Hsiao98 gens	 states reached	 time(sec)
s1423	74	3119	8	50	2743	50	61	3953
s3271	21	6015	8	15	1664	200	18	6319
s3384	67	18314	8	31	3794	250	45	21161
s5378	115	31281	8	45	3133	100	48	225160
s6669	30	1764	8	23	1701	50	24	2289
scfRjisdre	48	803	8	25	501	100	31	5196
s832jcsrre	8	139	8	7	120	100	7	2170
s510Rjcsrre	16	163	8	12	61	100	13	504
s510Rjosrre	16	181	8	9	62	100	13	583

Table VII. Faults detected by the two state-justification techniques

		faults detected			states reached	
circuit	TF	approach in Hsiao98	our approach	TS	approach in Hsiao98	our approach
s1423	926	312	578	135	61	74
s3271	61	34	41	45	18	21
s3384	376	91	116	102	45	67
s5378	1221	103	285	524	48	115
s6669	40	29	31	32	24	30
scfRjisdre	1920	1397	1802	267	31	48
s832jcsrre	293	38	147	57	7	8
s510Rjcsrre	374	45	85	114	13	16
s510Rjosrre	459	232	431	114	13	16

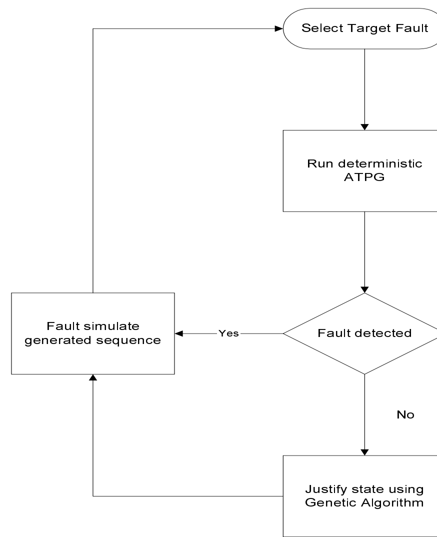

Figure 1. A block diagram of the hybrid state-justification methodology.

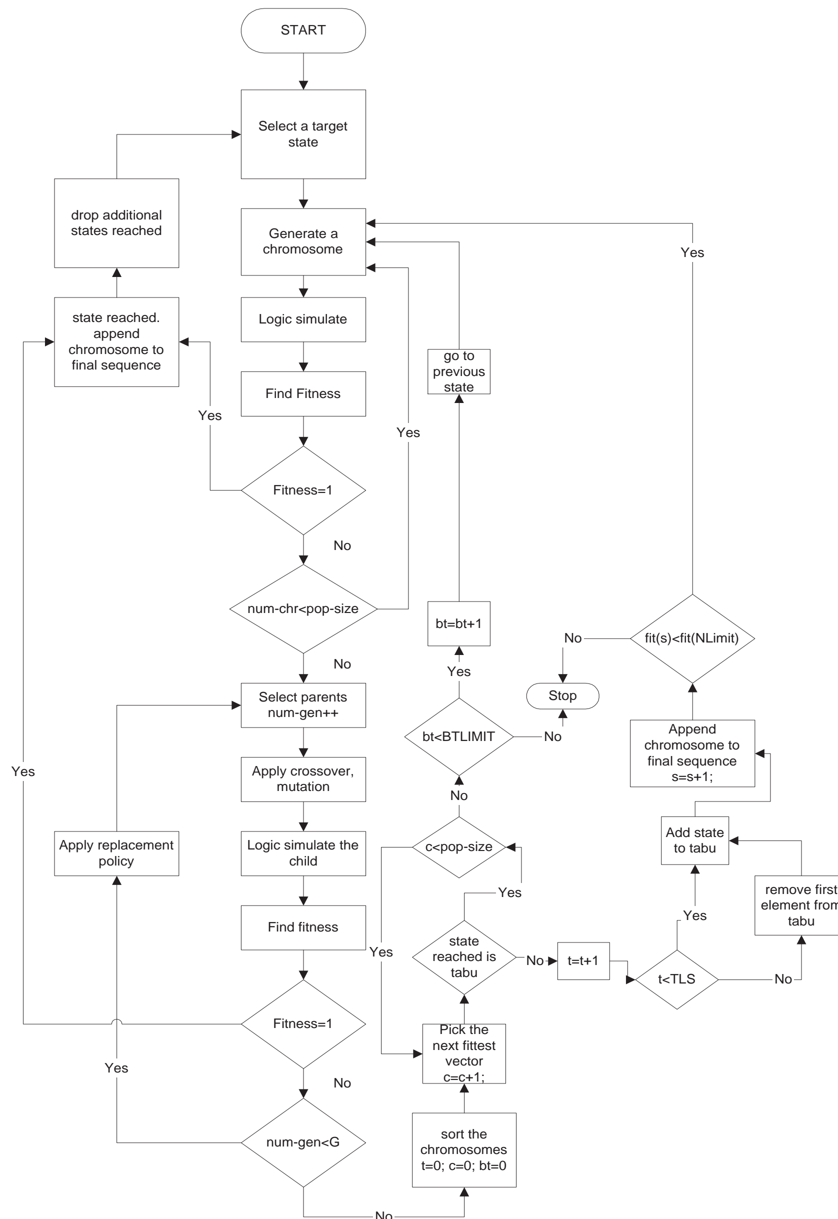

Figure 2. A flowchart of the meta-heuristic used.

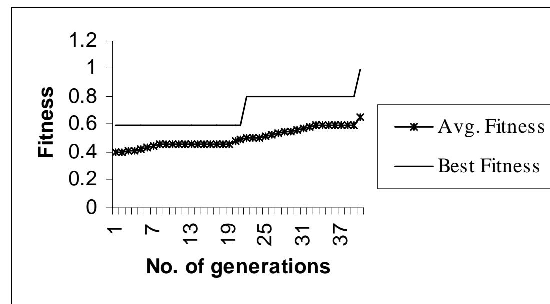

Figure 3. Average and best fitness vs. number of generations.

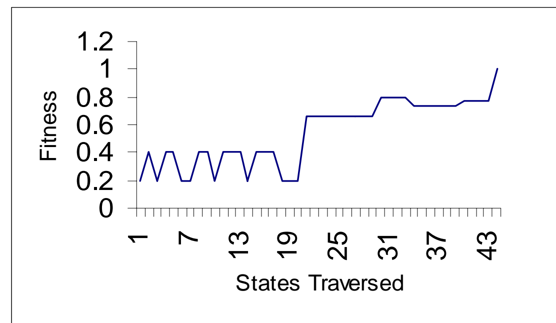

Figure 4. States traversed versus the fitness of reached states for a target state of s1423 circuit.

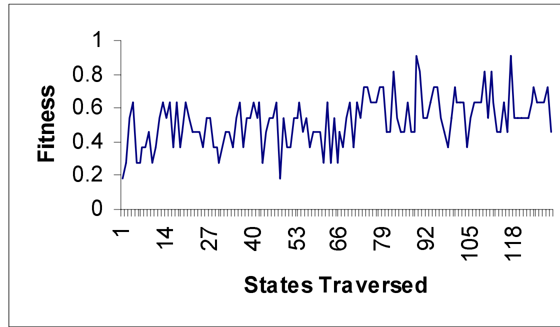

Figure 5. States traversed vs the fitness of reached states for one of the unreached state of s1423 circuit.