Sequential Circuit Test Generation in a Genetic Algorithm Framework

Elizabeth M. Rudnick Janak H. Patel Gary S. Greenstein Thomas M. Niermann

Center for Reliable & High-Performance Computing
University of Illinois, Urbana, IL

Sunrise Test Systems, Inc.
Santa Clara, CA

Abstract—Test generation using deterministic fault-oriented algorithms is highly complex and time-consuming. New approaches are needed to augment the existing techniques, both to reduce execution time and to improve fault coverage. In this work, we describe a genetic algorithm (GA) framework for sequential circuit test generation. The GA evolves candidate test vectors and sequences, using a fault simulator to compute the fitness of each candidate test. Various GA parameters are studied, including alphabet size, fitness function, generation gap, population size, and mutation rate, as well as selection and crossover schemes. High fault coverages were obtained for most of the ISCAS89 sequential benchmark circuits, and execution times were significantly lower than in a deterministic test generator in most cases.

I Introduction

Simulation-based test generation has been used to avoid the long execution times of deterministic algorithms and to reduce the complexity of the test generator. In particular, in a simulation-based approach, processing occurs in the forward direction only; i.e., no backtracing is required. Therefore, complex component types are more easily handled. As a result, the development time is greatly reduced.

Seshu and Freeman [1] first proposed simulation-based test generation, and several simulation-based test generators have since been developed [2, 3, 4, 5, 6, 7]. Breuer [2] used a fault simulator to evaluate sets of random vectors and to select the best vector to apply in each time frame. Weighted random pattern generators were interfaced with fault simulators in [3, 4, 5], and high fault coverages were obtained for combinational circuits. The test generators in [6, 7] were also built around fault simulators, but only candidate vectors of Hamming distance one from the previous vector were considered. Specific faults were targeted in [6], with a backtrace step used to select the bit to be flipped. Cost functions calculated during concurrent fault simulation were used to evaluate candidate vectors in [7]. While development of these random and mutation-based test generators was simplified and test generation time was reduced, the test sets generated were typically much longer than those generated by deterministic test generators.

Genetic algorithms (GAs) were first used as a framework for simulation-based test generation in [8, 9], but only combinational circuits were handled in [9]. The CRIS test generator [8] used a logic simulator to evaluate candidate test sequences; consequently the test sets generated often had lower fault coverages than those generated by a deterministic test generator. Furthermore, a heuristic crossover scheme was used to exploit problem-specific knowledge, making it difficult to separate the effects of the GA from the application-specific heuristics used. The simple GA described by Goldberg [10] was applied to the generation of individual test vectors for combinational and sequential circuits in [11]. Compact test sets with high fault coverages were obtained for most combinational circuits and many of the sequential circuits. However, fault coverages were lower for highly sequential circuits.

In this work we extend the GA-based test generator to evolve test sequences in addition to individual test vectors. The GA generates candidate test vectors and sequences, and the fitness of each candidate test is computed by a sequential circuit fault simulator. One issue in evolving test sequences is the alphabet size, i.e., whether a binary or nonbinary coding should be used. In a binary coding, the individual vectors in a sequence are packed into a single string, and the GA operates on that string. In a nonbinary coding, each possible vector is a separate character in the alphabet, and the GA operates on the test sequence as a string of characters in the alphabet, with special operators developed for the nonbinary alphabet. Another concern is to achieve compact test sets in a reasonable execution time. We have chosen to use fault simulation of candidate tests rather than the less accurate logic simulation used in CRIS [8] to provide a better quality test set. We use a small sample of faults in the fitness computation to speed up the execution. Another technique to reduce execution time is to use overlapping populations in the GA in which only a fraction of candidate tests are replaced in each generation.

We begin with a brief description of GAs. An overview of test generation in a GA framework is given next, and alphabet size, fitness functions, overlapping populations, and GA parameters are discussed. Modifications made to the fault simulator are then described, and results for the ISCAS89 sequential benchmark circuits [12] are presented.

*This research was supported in part by the Semiconductor Research Corporation under Contract SRC 93-DP-109.

II Genetic Algorithms

GAs are composed of populations of strings, or *chromosomes*, and three evolutionary operators: *selection, crossover,* and *mutation* [10]. The chromosomes may be binary-coded or they may contain characters from a larger alphabet [13, 14]. The initial population is typically generated randomly, but it may also be supplied by the user. A highly fit population is evolved through several generations by *selecting* two individuals, *crossing* the two individuals, and *mutating* characters in the resulting individuals with a given mutation probability. In a simple GA, distinct generations are evolved, and the processes of selection, crossover, and mutation are repeated until all entries in a new generation are filled. Then the old generation is discarded. In a GA having overlapping generations, only a fraction of the individuals are replaced in each generation [15, 16]. The fitness of each individual depends on the application, and selection is biased towards more highly fit individuals. Hence the fitness of the overall population is expected to increase in successive generations.

Various selection schemes have been used, but we will focus on *roulette wheel selection, stochastic universal selection,* and binary *tournament selection* with and without replacement [17, 18]. Roulette wheel selection is a proportionate selection scheme in which the slots of a roulette wheel are sized according to the fitness of each individual in the population, and an individual is selected by spinning the roulette wheel. Stochastic universal selection is a less noisy version of roulette wheel selection in which N equidistant markers are placed around the roulette wheel, where N is the number of individuals in the population. N individuals are selected in a single spin of the roulette wheel, and the number of copies of each individual selected is equal to the number of markers inside the corresponding slot. In binary tournament selection, two individuals are taken at random, and the better individual is selected from the two. The two parents may or may not be replaced into the original population for the next selection.

Once two chromosomes are selected, the *crossover* operator is used to generate two offspring. In *one-* and *two-point crossover,* one or two chromosome positions are randomly selected between 1 and (L - 1), where L is the chromosome length, and the two parents are crossed at those points. For example, in one-point crossover, the first child is identical to the first parent up to the crossing point and identical to the second parent after the crossing point. In *uniform crossover,* each chromosome position is crossed with some probability, typically 1/2. As the new individuals are generated, each character is mutated with a given probability. In a binary-coded GA, mutation is done by flipping a bit, while in a nonbinary-coded GA, mutation involves randomly generating a new character in a specified position.

III Test Generation in a GA Framework

Genetic algorithms can be used to generate populations of candidate test vectors and sequences and to select the best test to apply in a given time frame. This process is illustrated in Figure 1. The test generator begins by generating individual test vectors. Then test sequences are generated until no more progress is made, at which point test generation terminates. A

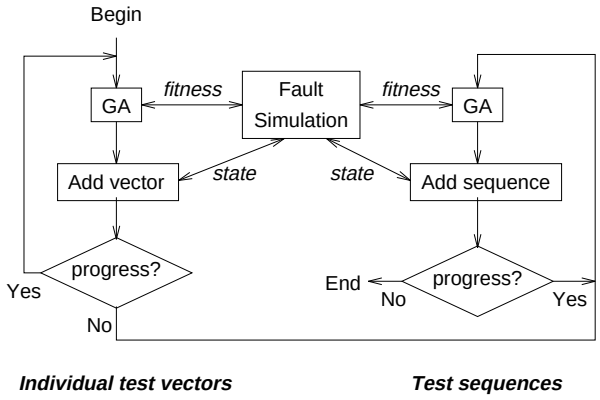

Figure 1: GA-Based Sequential Circuit Test Generation

GA having a random initial population is used to generate each test vector or sequence, and a sequential circuit fault simulator is used to evaluate the fitness of each candidate test. The best test evolved in any generation is selected and added to the test set. Then the fault simulator is used to update the state of the circuit and to drop detected faults.

Generation of individual test vectors is repeated until a given number of vectors are successively generated which do not improve the fault coverage. Several vectors may be required to change the state of a sequential circuit so that additional faults may be detected. On the other hand, the circuit is not guaranteed to go into a desirable state, and a large vector limit may increase the execution time and test set size. In our test generator, we use a small multiple of the sequential depth as the test vector *progress limit.*

Even when test sequences are being generated, a sequence may not be found to improve the fault coverage if the initial population does not contain the right combination of vectors. Therefore, the GA is reinitialized with a new random population for each attempt at generating a useful test sequence. Test generation for a given sequence length is terminated if four consecutive attempts fail to improve the fault coverage. In addition, various sequence lengths are used in an attempt to achieve a high fault coverage in a reasonable time. In this work we use one, two, and four times the sequential depth for most circuits, beginning with the smallest sequence length. Many of the faults are detected by the individual test vectors and shorter sequences, and only the most difficult faults remain for the longest sequences. Execution time is thus reduced.

A Alphabet Size

During generation of individual test vectors, each character of a chromosome in the population is mapped to a primary input. A binary coding is used, so the chromosome represents a test vector. In contrast, either a binary or nonbinary coding can be used during test sequence generation, and the genetic operators used depend on the alphabet size. If a binary coding is used, the individual vectors in a sequence are placed in adjacent positions on a single chromosome. Then the GA processes that chromosome using the same selection, bitwise crossover, and bitwise mutation operators that are used in generating individual test vectors. In a nonbinary coding, all 2^L possible vectors are separate characters in the alphabet, where

L is the vector length, and the individual characters of a chromosome represent separate test vectors in a sequence. In our implementation, the characters are mapped to their binary equivalents to enable simulation of candidate tests, but test vector boundaries are maintained to separate characters. The increased alphabet size has no effect on the selection operator, but the crossover and mutation operators must be modified. Crossover can occur at test vector boundaries only, and mutation involves replacing a given vector in a sequence with a randomly-generated vector.

B Fitness Function

An accurate fitness function is needed to achieve a high quality test set. Several factors may be important in generating a test vector or sequence, depending on the phase of test generation. In the initial phase of test vector generation, test vectors are generated to initialize the flip-flops. Therefore the fitness of a candidate vector is a measure of the number of flip-flops set to a known (0 or 1) state. To differentiate vectors which cause the same number of flip-flops to be set, we also include the fraction of flip-flops changing values since the previous time frame. Only a good circuit simulation is required to obtain the flip-flop state information. When all flip-flops are set, the test generator switches to *phase 2* in which test vectors are generated to maximize the number of faults detected. In this phase, the fitness of a candidate vector indicates the number of faults it detects. To differentiate vectors which detect the same number of faults, we include the number of fault effects propagated to flip-flops in the fitness function, since fault effects at the flip-flops may be propagated to the primary outputs in the next time frame. However, the number of fault effects propagated is offset by the number of faults simulated and the number of flip-flops to ensure that the number of faults detected is the dominant factor in the fitness function. When a test vector is generated which detects no additional faults, the test generator enters *phase 3* and begins counting the number of noncontributing test vectors. In order to encourage the evolution of useful vectors, we add the good and faulty circuit activity levels to the other two measures used in phase 2. Vectors which activate more faults and propagate more fault effects will then have higher fitness values, and the GA will be more likely to evolve a vector which can propagate the effects of some fault to a primary output. If a test vector is found which detects any faults before the number of noncontributing vectors generated reaches the progress limit, the test generator switches back to phase 2 and the noncontributing vector count is reset to zero. The procedure for generation of individual test vectors is summarized in Figure 2.

When the number of successive noncontributing vectors generated exceeds the progress limit, the test generator proceeds with test sequence generation. In this phase, the fitness function used is the same as that for the second phase of test vector generation except that the test sequence length is included in the metric for the number of fault effects propagated to flip-flops. In summary, the fitness of a candidate vector is calculated as follows:

$$Phase\ 1\text{:}\ fitness = total\ flip\ flops\ set + fraction\ of\ flip\ flops\ changed$$

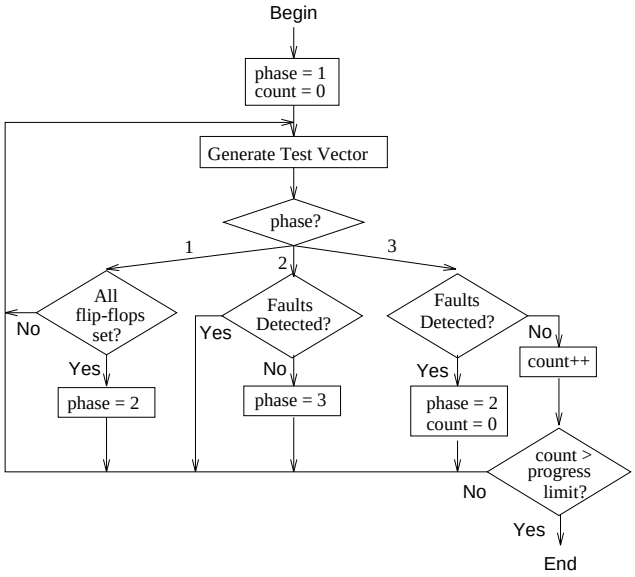

Figure 2: Generation of Individual Test Vectors

$$Phase\ 2\text{:}\ fitness = \#\ faults\ detected + \frac{\#\ faults\ propagated\ to\ flip\ flops}{(\#\ faults)(\#\ flip\ flops)}$$

$$Phase\ 3\text{:}\ fitness = \#\ faults\ detected + \frac{\#\ faults\ propagated\ to\ flip\ flops}{(\#\ faults)(\#\ flip\ flops)} + \frac{2(\#\ good\ and\ faulty\ circuit\ events)}{(\#\ circuit\ nodes)(\#\ faults)}$$

$$Phase\ 4\ (Test\ Sequence\ Generation)\text{:}\ fitness = \#\ faults\ detected + \frac{\#\ faults\ propagated\ to\ flip\ flops}{(\#\ faults)(\#\ flip\ flops)(sequence\ length)}$$

While an accurate fitness function is essential in achieving a good solution, the high computational cost of fault simulation may be prohibitive, especially for large circuits. To avoid excessive computations, we can approximate the fitness of a candidate test by using a small sample of faults. In particular, we can use a small fraction of the remaining faults chosen at random, e.g., 1%–10%, or a set sample size, e.g., 100–300 faults.

C Overlapping Populations

Since computation of the fitness function is very expensive in this application, overlapping populations may be effective in speeding up execution by reducing the number of fitness computations. With this approach, only a fraction of candidate tests are replaced in each generation. Let the population size be N. In evolving the next generation, the GA generates only g offspring, where $1 \leq g \leq N$. $G = g/N$ is referred to as the *generation gap.* The g worst individuals in the previous generation are replaced by the new individuals. In this way, $(N-g)$ fewer fitness computations are required per generation. However, a larger population size may be needed to provide diversity, and more generations may be required.

D GA Parameters

Several GA parameters are important in achieving good results. A sufficient population size is needed to ensure adequate diversity. All characters in the alphabet should be present at every chromosome position, and a sufficient population size is needed to provide good combinations of characters. At the same time, a reasonable limit on the population size is needed to reduce computations. A second key parameter is the number of generations. A sufficient number of generations is needed to allow useful combinations of characters from several chromosomes to mix. In this work we limit the number of generations to 8 to reduce the run time.

The remaining parameters of interest are the crossover and mutation probabilities. We use a crossover probability of 1; i.e., two individuals are always crossed in generating two new individuals. Mutation is used to prevent the loss of key characters at the various chromosome positions. However, mutation also destroys good combinations of characters, so a balance must be found. The population sizes and mutation probabilities used in this work are shown in Table 1 for the generation of individual test vectors. During test sequence generation, a population size of 32 and a mutation rate of 1/64 are used for all circuits.

Table 1: GA Parameter Values

Vector Length (L)	Population Size	Mutation Probability
< 4	8	1/8
4–16	16	1/16
> 16	16	1/L

IV Modifications to the Fault Simulator

The sequential circuit fault simulator PROOFS [19] is used to evaluate the fitness of each candidate test. During fault simulation, the good and faulty circuit states are updated after each test vector is simulated. In addition, the status of each detected fault is changed, and detected faults are removed from the fault list. To accommodate the simulation of candidate tests, the fault simulator must be modified to store and restore the good and faulty circuit states and the fault detection status before and after each test is applied. Also, the faulty circuit event count must be kept, as well as a count of the number of faults whose effects propagate to flip-flops.

V Results

The GA-based test generator was implemented around the PROOFS sequential circuit fault simulator [19] in 3000 additional lines of C++ code. Tests were generated for the ISCAS89 sequential benchmark circuits [12] using the GA-based test generator on a SUN SPARCstation II with 64 MB memory. Circuit descriptions and test generation results are shown in Table 2.

The structural sequential depth is taken from [20] and is the minimum number of flip-flops in a path between the primary inputs and the furthest gate. The numbers of faults detected, the numbers of test vectors generated, and the execution times are shown for tournament selection without replacement and uniform crossover. A binary coding was used. The progress limit for test vector generation was equal to the sequential depth for s5378 and s35932 and four times the sequential depth for all other circuits. Test sequence lengths were 1/4, 1/2, and 1 times the sequential depth for s5378 and s35932 and 1, 2, and 4 times the sequential depth for all other circuits. Each result is the average from ten runs (except for circuit s35932 which is averaged over eight runs), and a new random seed was used for each run; standard deviations are given in parentheses. The numbers of faults detected, the numbers of vectors generated, and the execution times on a Sun SPARCstation SLC are shown for the HITEC deterministic, fault-oriented test generator [20] for comparison. The number of faults detected was greater than or equal to that of HITEC for seven of the 17 circuits for which fault coverages were available. The fault detection count was within 10 faults for another 3 circuits and within 20 faults for an additional 3 circuits. Fault coverages were higher than those reported for CRIS [8] for 17 of 18 circuits, although the execution time was between 6 and 40 times as long, depending on the circuit. Test set length was one-third that of CRIS and 42% that of HITEC on average.

In most cases, test generation time for the GA-based test generator is a small fraction of the time required by HITEC. Thus, the GA-based test generator can be used as a first pass in test generation to screen out many of the faults before applying a deterministic test generator. Note that untestable faults cannot be identified by a simulation-based test generator, so the deterministic fault-oriented test generator is still needed for this purpose. Results for various selection and crossover schemes are shown in Table 3. Again results are shown averaged over ten runs. Circuits s344, s349, s382, s400, s444, s641, and s713 are omitted from the table since these circuits had about the same fault coverages for all selection and crossover schemes. Significant differences in fault coverage were found for many of the larger circuits. The best selection scheme was tournament selection without replacement; both tournament selection schemes gave better results than either of the proportionate selection schemes. Uniform crossover gave consistently better results than 1-point or 2-point crossover.

The effects of mutation rate on fault coverage were also investigated. Results are shown in Table 4 averaged over ten runs for various mutation rates used during test sequence generation; mutation rates given in Table 1 were used while individual test vectors were being generated. Tournament selection without replacement and uniform crossover were used. Circuits having about the same fault coverage for all mutation rates are not included. The mutation rate had a much smaller effect on fault coverage than the selection and crossover schemes. Significant differences in fault coverage were obtained for only 2 of the larger circuits, s1423 and s5378, but for those circuits, the lowest mutation rates (1/128 and 1/256) tended to give the highest fault coverage.

Binary and nonbinary codings of test sequences are compared in Table 5. Results are averaged over ten runs. Population sizes of 16, 32, and 64 were used during test sequence generation; the population sizes given in Table 1 were used while individual test vectors were being generated. Circuits having about the same fault coverage for all experiments are

Table 2: Sequential Circuit Results

Circuit	PIs	Seq Depth	Total Faults	HITEC			GA		
				Det	Vec	Time	Det	Vec	Time
s298	3	8	308	265	306	4.44h	264.7(0.5)	161(28)	6.05m
s344	9	6	342	328	142	1.33h	329.0(0.0)	95(14)	5.85m
s349	9	6	350	335	137	52.2m	335.0(0.0)	95(14)	5.83m
s382	3	11	399	363	4931	12.0h	347.0(1.2)	281(27)	8.91m
s386	7	5	384	314	311	1.03m	295.2(2.2)	154(24)	3.45m
s400	3	11	426	383	4309	12.1h	365.1(2.7)	280(26)	9.45m
s444	3	11	474	414	2240	16.1h	405.7(1.7)	275(21)	10.5m
s526	3	11	555	365	2232	46.8h	416.7(4.8)	281(42)	14.3m
s641	35	6	467	404	216	18.0m	404.0(0.0)	139(31)	8.24m
s713	35	6	581	476	194	1.52m	476.0(0.0)	128(7)	9.41m
s820	18	4	850	813	984	1.61h	516.5(29.2)	146(17)	13.4m
s832	18	4	870	817	981	1.76h	539.0(32.1)	150(17)	12.3m
s1196	14	4	1242	1239	453	1.53m	1232(3)	347(45)	11.6m
s1238	14	4	1355	1283	478	2.20m	1274(3)	383(40)	16.0m
s1423	17	10	1515	-	-	-	1222(51)	663(103)	2.83h
s1488	8	5	1486	1444	1294	3.60h	1392(32)	243(26)	25.2m
s1494	8	5	1506	1453	1407	1.91h	1416(20)	245(39)	23.2m
s5378	35	36	4603	-	-	-	3175(53)	511(54)	6.08h
s35932	35	35	39094	34902	240	3.80h	35009(51)	197(43)	105.2h

Table 3: Selection and Crossover Scheme Comparison: Detected Faults

Circuit	Roulette Wheel Selection			Stochastic Universal Selection			Tournament Selection					
							No Replacement			Replacement		
	1-pt	2-pt	Unif	1-pt	2-pt	Unif	1-pt	2-pt	Unif	1-pt	2-pt	Unif
s298	264.1	264.1	264.0	264.8	264.8	264.1	264.2	264.3	264.7	264.3	264.8	264.9
s386	294.2	293.0	295.5	296.6	296.1	297.8	294.6	296.7	295.2	297.3	296.2	295.9
s526	419.7	419.7	417.8	422.0	414.7	417.9	415.6	417.2	416.7	416.7	418.3	419.5
s820	501.2	478.4	514.3	502.9	497.4	524.1	520.4	519.6	516.5	527.9	527.5	504.5
s832	512.0	503.7	506.6	500.6	515.9	512.5	522.2	516.4	539.0	516.4	502.1	514.7
s1196	1228	1228	1232	1229	1228	1231	1227	1229	1232	1227	1225	1230
s1238	1270	1272	1274	1273	1271	1275	1269	1272	1274	1268	1272	1275
s1423	1243	1229	1257	1210	1243	1223	1242	1219	1222	1250	1227	1212
s1488	1363	1381	1352	1378	1360	1367	1392	1390	1392	1380	1388	1395
s1494	1357	1362	1361	1352	1401	1394	1412	1388	1416	1384	1391	1408
s5378	3169	3160	3216	3124	3183	3167	3175	3165	3175	3168	3150	3180

1-pt: one-point crossover **2-pt**: two-point crossover **Unif**: uniform crossover

not shown. Tournament selection without replacement and uniform crossover were used. Fault coverages tended to improve with increasing population size, as expected. The best results were obtained using the largest population size, but good results were also obtained with population sizes of 16 and 32. Therefore we recommend using a population size of 16 or 32 to reduce the execution time. At the largest population size, a nonbinary coding gave better results, but significant differences were obtained for two circuits only. At the smaller population sizes, a binary coding usually gave better results.

Execution times may be reduced by using only a small fraction of the fault list in the fitness evaluation. Results of using this approach are shown in Table 6 averaged over several runs for sample sizes of 100, 200, and 300 faults. Tournament selection without replacement, uniform crossover, and a binary coding were used. If the number of faults remaining in the fault list dropped below the fault sample size, then all remaining faults were simulated. For the smaller circuits, the differences in fault coverage were due to the nondeterminism of the algorithm used, since the undetected fault list size eventually dropped below the fault sample sizes. The highest fault coverages were typically obtained when the entire fault list was used. In general, the execution times were lower for the smaller fault sample sizes, particularly for the larger circuits. The minimum size of the fault sample needed to obtain good results tends to increase with increasing circuit size.

Overlapping populations were also investigated as a means of reducing execution time. Generation gaps of 2/N, 1/4, 1/2, and 3/4 were tried, where N is the population size. Corresponding population sizes used were 3-, 2-, 1.5-, and 1-times the population size used for nonoverlapping populations. The number of generations was also adjusted for generation gaps of 2/N and 1/4 to provide approximately the same number of evaluations for all experiments. Since the number of evalua-

Table 4: Mutation Rate Comparison: Detected Faults

Circuit	Mutation Rate				
	1/16	1/32	1/64	1/128	1/256
s298	264.4	264.8	264.7	264.8	264.3
s386	296.1	296.8	295.2	296.1	295.5
s820	510.7	509.0	516.5	510.4	510.3
s832	533.5	533.6	539.0	533.5	533.1
s1196	1231	1230	1232	1231	1230
s1238	1274	1275	1274	1276	1274
s1423	1216	1226	1222	1244	1258
s1488	1394	1394	1392	1393	1391
s1494	1416	1415	1416	1418	1417
s5378	3204	3159	3175	3175	3192

Table 5: Binary and Nonbinary Coding Comparison: Detected Faults

Circuit	Pop 16		Pop 32		Pop 64	
	Bin	Non	Bin	Non	Bin	Non
s298	264.6	263.6	264.7	264.4	264.8	264.9
s386	294.4	294.0	295.2	294.8	296.5	295.8
s526	416.1	416.1	416.7	416.7	417.4	417.0
s820	507.4	508.3	516.5	508.4	509.0	510.0
s832	533.0	534.6	539.0	533.5	533.4	534.2
s1196	1228	1223	1232	1228	1233	1229
s1238	1273	1262	1274	1267	1277	1273
s1423	1196	1202	1222	1219	1246	1266
s1488	1389	1386	1392	1387	1396	1395
s1494	1416	1413	1416	1416	1417	1415
s5378	3162	3165	3175	3190	3179	3205

Bin: binary coding **Non**: nonbinary coding

tions was about 81% of the number used for nonoverlapping populations, execution times were correspondingly smaller. Results for overlapping populations are given in Table 7 averaged over ten runs. Fault coverages for a generation gap of 3/4 were only 0.4% lower on average than for nonoverlapping populations. Generation gaps of 1/2, 1/4, and 2/N resulted in successively lower fault coverages overall.

VI Conclusions

A genetic algorithm framework was developed for use in sequential circuit test generation. Populations of candidate tests are evolved by the GA starting from a random initial population, and the best test evolved is added to the test set in a given time frame. A highly accurate fitness function is used to evaluate candidate tests in order to achieve good quality test sets. Results for the ISCAS89 sequential benchmark circuits indicate that the selection and crossover schemes used have a significant impact on fault coverage. The best results were obtained for tournament selection without replacement and uniform crossover. Variations in mutation rate had a much smaller effect on fault coverage, and binary codings tended to give higher fault coverages when small population sizes of 16 or 32 were used. Nonoverlapping populations gave the highest fault coverages, but average speedups of 1.3 were obtained by using overlapping populations, with only a 0.4% drop in fault coverage. More significant reductions in execution time were obtained by using small fault samples in the fitness evaluation. Genetic algorithms are particularly amenable to parallel implementations, so very good speedups are expected for a parallel GA-based test generator. In addition, the GA-based test generator is not limited to the single stuck-at fault model, and other fault models can easily be accommodated with appropriate fitness functions.

Acknowledgment

The authors would like to thank Prof. David Goldberg for providing several useful suggestions.

References

[1] S. Seshu and D. N. Freeman, "The diagnosis of asynchronous sequential switching systems," *IRE Trans. Electronic Computing,* vol. 11, August 1962, pp. 459-465.

[2] M. A. Breuer, "A random and an algorithmic technique for fault detection test generation for sequential circuits," *IEEE Trans. Computers,* vol. 20, no. 11, November 1971, pp. 1364-1370.

[3] H. D. Schnurmann, E. Lindbloom, and R. G. Carpenter, "The weighted random test-pattern generator," *IEEE Trans. Computers,* vol. 24, no. 7, July 1975, pp. 695-700.

[4] R. Lisanke, F. Brglez, A. J. Degeus, and D. Gregory, "Testability-driven random test-pattern generation," *IEEE Trans. Computer-Aided Design,* vol. 6, no. 6, November 1987, pp. 1082-1087.

[5] H.-J. Wunderlich, "Multiple distributions for biased random test patterns," *IEEE Trans. Computer-Aided Design,* vol. 9, no. 6, June 1990, pp. 584-593.

[6] T. J. Snethen, "Simulator-oriented fault test generator," *Proc. Design Automation Conf.,* 1977, pp. 88-93.

[7] V. D. Agrawal, K. T. Cheng, and P. Agrawal, "A directed search method for test generation using a concurrent simulator," *IEEE Trans. Computer-Aided Design*, vol. 8, no. 2, February 1989, pp. 131-138.

[8] D. G. Saab, Y. G. Saab, and J. A. Abraham, "CRIS: A test cultivation program for sequential VLSI circuits," *Proc. Int. Conf. Computer-Aided Design*, 1992, pp. 216-219.

[9] M. Srinivas and L. M. Patnaik, "A simulation-based test generation scheme using genetic algorithms," *Proc. Int. Conf. VLSI Design*, 1993, pp. 132-135.

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

[11] E. M. Rudnick, J. G. Holm, D. G. Saab, and J. H. Patel, "Application of simple genetic algorithms to sequential circuit test generation," *Proc. European Design and Test Conf.*, 1994, pp. 40-45.

[12] F. Brglez, D. Bryan, and K. Kozminski, "Combinational profiles of sequential benchmark circuits," *Int. Symposium on Circuits and Systems*, May 1989, pp. 1929-1934.

Table 6: Fault Sampling

Circuit	Sample Size								
	100 Faults			200 Faults			300 Faults		
	Det	Vec	Spdup	Det	Vec	Spdup	Det	Vec	Spdup
s298	264.5	161	1.05	264.7	168	0.99	265.0	179	0.95
s382	348.1	295	1.06	347.2	277	1.03	347.3	274	1.01
s386	286.8	128	1.16	297.3	133	1.11	295.3	143	1.07
s526	417.0	293	1.79	417.4	314	1.04	418.8	295	1.04
s820	494.7	144	2.75	536.8	157	1.77	532.2	155	1.45
s832	476.4	137	2.51	526.3	158	1.70	546.2	156	1.40
s1196	1230	373	1.55	1231	384	1.08	1230	348	1.12
s1238	1269	389	1.26	1274	375	1.19	1274	381	1.18
s1423	1245	619	3.28	1255	587	2.32	1287	778	1.11
s1488	1153	211	2.14	1394	272	1.03	1378	233	1.12
s1494	1303	267	1.65	1370	235	1.17	1400	242	1.10
s5378	3048	394	6.31	3095	409	5.24	3130	450	4.25
s35932	34839	234	4.53	34854	185	4.74	34926	203	4.35

$$\textbf{Spdup}: \frac{Execution\ time\ using\ full\ fault\ list}{Execution\ time\ using\ fault\ sample}$$

Table 7: Overlapping Populations

Circuit	Generation Gap											
	2/N			1/4			1/2			3/4		
	Det	Vec	Spdup	Det	Vec	Spdup	Det	Vec	Spdup	Det	Vec	Spdup
s298	263.9	205	1.03	264.4	183	1.14	264.7	173	1.12	265.0	167	1.27
s382	348.1	270	1.24	347.8	277	1.23	346.7	283	1.17	347.0	270	1.28
s386	294.4	137	1.28	294.9	134	1.34	295.5	142	1.26	296.8	144	1.30
s526	416.7	306	1.20	420.4	299	1.21	417.2	298	1.13	418.1	301	1.25
s820	520.2	155	1.28	522.4	144	1.37	519.5	141	1.34	500.1	138	1.38
s832	512.2	140	1.22	508.0	154	1.14	521.9	151	1.14	500.7	142	1.21
s1196	1231	341	1.30	1231	374	1.20	1231	356	1.22	1230	385	1.20
s1238	1271	388	1.30	1274	393	1.31	1274	378	1.27	1273	394	1.36
s1423	1213	666	1.23	1216	677	1.20	1247	657	1.14	1239	669	1.16
s1488	1381	220	1.38	1410	252	1.33	1393	231	1.28	1404	247	1.35
s1494	1410	256	1.21	1402	236	1.28	1402	250	1.15	1408	239	1.32
s5378	3164	522	1.12	3170	560	1.09	3156	490	1.23	3193	500	1.33

$$\textbf{Spdup}: \frac{Execution\ time\ using\ nonoverlapping\ populations}{Execution\ time\ using\ overlapping\ populations}$$

[13] D. E. Goldberg, "Real-coded genetic algorithms, virtual alphabets, and blocking," IlliGAL Report 90001, Illinois Genetic Algorithms Laboratory, Dept. of General Engineering, University of Illinois, Urbana, IL, 1990.

[14] L. J. Eshelman and J. D. Schaffer, "Real-coded genetic algorithms and interval-schemata," in *Foundations of Genetic Algorithms,* L. D. Whitley (ed.), Morgan Kaufmann, San Mateo, CA, 1993, pp. 187-202.

[15] D. Whitley, "The GENITOR algorithm and selection pressure: Why rank-based allocation of reproductive trials is best," *Proc. Third Int. Conf. Genetic Algorithms*, 1989, pp. 116-121.

[16] K. A. De Jong and J. Sarma, "Generation gaps revisited," in *Foundations of Genetic Algorithms,* L. D. Whitley (ed.), Morgan Kaufmann, San Mateo, CA, 1993, pp. 19-28.

[17] D. E. Goldberg, and K. Deb, "A comparative analysis of selection schemes used in genetic algorithms," in *Foundations of Genetic Algorithms,* G. Rawlins (ed.), Morgan Kaufmann, San Mateo, CA, 1991, pp. 69-93.

[18] J. E. Baker, "Reducing bias and inefficiency in the selection algorithm," *Proc. Second Int. Conf. Genetic Algorithms,* 1987, pp. 14-21.

[19] T. M. Niermann, W. -T. Cheng, and J. H. Patel, "PROOFS: A fast, memory-efficient sequential circuit fault simulator," *IEEE Trans. Computer-Aided Design*, February 1992, pp. 198-207.

[20] T. M. Niermann, "Techniques for sequential circuit automatic test generation," Technical Report CRHC-91-8, Coordinated Science Laboratory, University of Illinois at Urbana-Champaign, March 1991.