Gate - A Genetic Algorithm for Compacting Randomly Generated Test Sets

J. H. Alyor, J. P. Cohoon
E. L. Feldhousen, B. W. Johnson

Computer Science Report No. TR-90-27
September 3, 1990

GATE — A GENETIC ALGORITHM FOR COMPACTING RANDOMLY GENERATED TEST SETS[†]

J. H. Aylor, J. P. Cohoon, E. L. Feldhousen,[‡] B. W. Johnson

Center for Semicustom Integrated Systems
University of Virginia
Charlottesville, VA 22903

ABSTRACT

A new technique, named GATE, is presented for the generation of compact test sets. GATE combines a previously proven method for random test pattern generation with the adaptive searching capabilities of genetic algorithms to produce very high quality test sets. A series of experiments demonstrated that our technique performed consistently better than the traditional method with respect to both fault coverage and test set size.

† This research was supported in part by the Virginia Center for Innovative Technology through grant 5-30971 and by the Defense Advanced Projects Research Agency (DARPA) through contract N00014-89-J1699.

‡ This author's current address is Cardiology Business Unit, Hewlett-Packard Company, 1700 S. Baker Street, McMinnville, OR 97128.

1. INTRODUCTION

The generation of test vectors for combinational VLSI circuits continues to be an important research area. The current situation, which has significant costs associated with test generation and application, is due largely both to the NP-completeness of deterministic test pattern generation [FUJI82, IBAR75] and to the incompleteness of random techniques. As a result, great effort has been spent developing efficient test generation techniques that use a variety of methods to produce compact test sets with high fault coverage. These methods range from the deterministic ones that attempt to generate at least one test for each single stuck-at fault in a circuit, to random and pseudo-random test generation that are used in conjunction with built-in self test, and to exhaustive methods that apply all possible input combinations [CART85, FUJI83, GOEL81, MCCL81, ROTH66]. While all these methods are appropriate under the proper circumstances, they can be further improved. For example, deterministic algorithmic methods attempt to generate at least one test for each single stuck-at fault in a circuit. And while they are complete in the sense that given enough time, they will determine a test for all detectable faults in a circuit [FUJI85], the required computation time for the fastest algorithm grows exponentially with circuit size [BRAH87]. Similar concerns can be expressed about other methods.

The work presented here uses genetic algorithms to offer an improved test generation technique, where a genetic algorithm [GOLD89, HOLL75] is an optimization method that resembles the mechanics of natural evolution that seems particularly well-suited to VLSI problems [COHO91]. Specifically, this research focuses on new ways of creating compact test sets from test sets that were generated with random techniques. Methods that improve the coverage of such test sets and approaches to minimizing the computational cost of this new technique are also presented. This is achieved in part by extending and exploiting the successful *SOFE* method of random pattern generation and fault simulation developed by Carter, Dennis, Iyengar, and Rosen [CART85].

The remainder of this paper is organized in the following fashion. Section 2 provides background information on genetic algorithms, as well as the fault simulation method *SOFE* upon which GATE is based. Sections 3 and 4 detail the GATE method, and Section 5 discusses experimental results. The computation of the GATE method is analyzed in Section 6 and Section 7 presents conclusions and areas for continued investigation.

2. PRELIMINARIES

The *genetic algorithm* paradigm [HOLL75] has been previously proposed to generate solutions to a wide range of problems [GOLD89]. In particular, it has been applied successfully to the VLSI problem domain [COHO87, COHO91, SMIT85]

In a genetic algorithm (GA), a *population* of solutions is maintained and successive *generations* are produced by manipulating the solutions in the current population. Each solution has a *fitness* that measures its *competence.* New solutions are formed typically by merging two previous ones via a *crossover* operator. Other new solutions are simply modifications of previous ones, using a *mutation* operator. Successive generations are produced with new solutions probabilistically replacing older ones based on relative fitness. An ad hoc termination condition is often used and the best seen solution is normally reported.

The following helps to illustrate more concretely the operation of genetic algorithms. Assume that the population of a GA contains n solutions, each of which is encoded using a string of finite length. During the crossover stage, a percentage C of the total population is chosen through *probabilistic selection by fitness* to participate in producing offspring solutions (i.e. solutions with the highest fitness are weighted most

```
for G iterations do
    while $n \times C \leq$ number of offspring created do
        Select two solutions
        Crossover the two solutions to create offspring
    end
    Add offspring to population
    Calculate fitnesses
    Select a population of $n$ elements
    Generate $n \times M$ random mutations
end
```

Figure 1. — High-level code for genetic algorithm [COHO91].

favorably to be parents and solutions with the lowest fitness are minimally weighted). After crossover and a probabilistic selection of the most fit solutions for survival, a fixed percentage of the surviving population, M, is then mutated. The algorithm repeats this process for G generations. The high level code in Figure 1 shows the general operation of a GA [COHO91].

By selecting the most fit solutions to participate in crossover, and by coding the solutions in an intuitively meaningful way [SUH87], the best parts of the best solutions may be combined to form better solutions. Thus, genetic algorithms can take advantage of the high quality building blocks in the best solutions found so far. In addition, genetic algorithms benefit from the simplicity of crossover and mutation operations, which are simple string copying and modification tasks, as opposed to algorithms which require a detailed knowledge of the problem being solved [GOLD89].

Besides using the adaptive searching capabilities of genetics algorithm, GATE also exploits and extends the successful *SOFE* method of random pattern generation and fault simulation developed by Carter *et al.* [CART85] Their *SOFE* method first constructs an initial test set using random pattern generation. This is followed by the application of heuristics to permute the subsequent test sets coupled with multiple fault simulations to reduce the size of the test sets. The fault simulation procedure ceases simulating a particular fault as soon as the first test in a set detects that fault. Thus, the method is called Stop On First Error, or *SOFE*. In their scheme, "two or more trials" (*SOFE*/ 2+) are performed. Each trial involves fault simulation of the current test set. The first trial randomly generates a set of tests and then fault simulates them using the *SOFE* principle, keeping only tests that detect a new fault. The second trial reverses the order of test set and re-invokes fault simulation using the *SOFE* principle. The heuristic reduction in the new test set size arises from the expectation that tests generated late in the previous trial found both hard to detect faults and many of the easier, earlier detected faults. Further trials with other test set orderings are performed to realize additional important reductions. To make this procedure more efficient, a covering heuristic is applied in tandem. The benefits of the covering heuristic stem from the fact that the fault simulator processes multiple tests in parallel. After each group of tests is simulated in parallel, the heuristic can be used to remove tests that are covered by other tests in that group. For the *ISCAS-85* Automatic Test Pattern Generation (ATPG) Benchmarks [BRGL85], *SOFE*/6 when used in conjunction with the covering heuristic removed on average half of the tests obtained in the first trial. In addition, almost all further improvement was found in the second trial. However, Carter *et al.* note that since the great majority of the fault simulation computational effort occurs during the first trial, the cost of trials 3-6 is insignificant. They also speculate that an important avenue of future *SOFE* research is the determination of intelligent permutations for further *SOFE* trials.

3. GENETIC ALGORITHM APPROACH

Our technique first randomly generates a population of *n* test sets where each set has reasonable fault coverage. GATE then uses this population and its adaptive search ability to effectively explore the solution space. To speed-up the convergence of GATE, the initial sets are pre-processed in a *SOFE*-like manner. An additional benefit of this pre-processing is an early identification of some of the crossover building blocks. Fault simulation or a covering heuristic determines the amount of fault coverage and test set length for each test set in the population, thus allowing fitness to be calculated. Fitness-based probabilistic selection is then used to determine which test sets should be used in crossover. The crossover operator exchanges subsets of the current population test sets to produce offspring. In addition, some test sets are mutated by either changing the order of tests in the test set or by adding new tests to the set. This adding of new tests to the population has the favorable side-effect of often increasing coverage. The relative fitnesses of all the test sets are then determined, and the most fit solutions are favored probabilistically to survive into the next generation. The algorithm runs normally until a particular coverage or compactness figure-of-merit is reached, or until some predetermined number of generations have passed. The test set solution with the highest fitness is returned as the solution of the problem.

In the remainder of this section we discuss in more detail several of the above components that comprise GATE.

Initial Population Generation

Although the initial population is generated randomly, several options are still available. The test sets that form the population may be generated using a pure random (uniform) distribution, using one or more weighted distributions [LISA86], or using a combination of uniform and weighted random distributions. The results presented here resulted from an initial population generated using a uniform distribution. In the future, we plan to consider the other options. One possible scenario is to use a separate GA to generate the initial population for GATE.

Crossover

Once a population has been generated, probabilistic selection by fitness is used to select members of the population to participate in crossover. The crossover operators that we considered involved either one-way or two-way transfers of portions of test sets from members of the populations; other methods are also possible.

Two sample members u and v of a population are given in Figure 2 to illustrate these several possible operations. The top and bottom sections of these members are

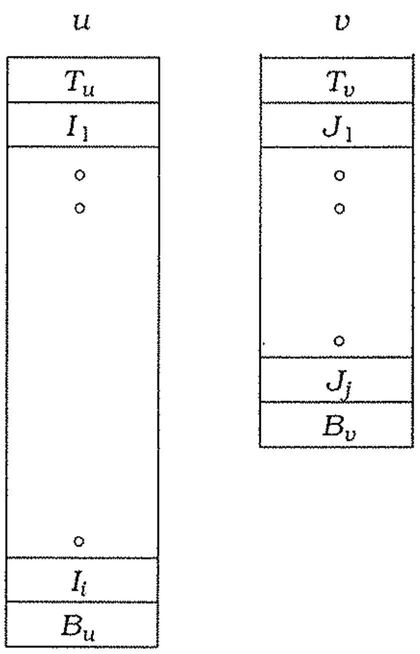

Figure 2. — Two sample population members.

identified respectively as T_u (T_v) and B_u (B_v). While we may not expect the interior region to be densely packed with tests for difficult to detect faults, some such tests may be present due to the random nature of the pattern generation. Also, as the genetic algorithm continues, the test set orderings are permuted and the "valuable" tests become more evenly distributed. Therefore, we divide the interiors of u and v into sections $I_1, \ldots, I_i$ and $J_1, \ldots, J_j$ respectively. Several crossover operators are considered. For example, given two tests sets u and v as in Figure 2, one possible crossover is to combine into an offspring either the top (bottom) section T_u (B_u) with the interior and bottom (top) sections J_v and B_v (T_v). Some other crossover operators that we considered, combined T_u and B_u with all of v. Other simple combinational crossover operators are readily conceived.

A second type of crossover is the two-way crossover that produces two offspring for each crossover. For example, T_u might be combined with all of the parent v to form one offspring, while at the same time B_v might be combined with u to form another offspring. Other combinations of top, bottom, and interior section exchanges constitute the various possible crossover operations of this type. An important crossover issue is whether the transferred section or sections should be placed at the top, in the interior or at the end of the resulting offspring. Another important issue is whether to simply add or replace the patterns of the transferred sections. Simply adding the patterns is attractive since it prevents a reduction in fault coverage, while replacement is attractive as it maintains compactness.

Mutation

Mutation is performed during each generation on randomly selected members of the population. It serves two purposes: it prevents premature convergence by perturbing the elements that comprise the population and it introduces additional information into the system. Several mutation operators were considered. They were selected to aid in one or both purposes. For example, one mutation operator simply randomly permuted the sections of a given test, and another randomly permuted the tests within a given section. Such permutations are quite simple to perform and relatively inexpensive since the cost of fault simulation is already incurred due to the need to evaluate new offspring.

Another mutation operator that we considered added a new test to one or more members of the population. Like the generation of the initial population, the new test may be generated using either a random or weighted scheme. If a weighted scheme is used, the distribution could be determined from the existing tests in the given set. Another considered mutation operator was one that randomly permuted or complemented the bit pattern of a given test.

Fitness Evaluation

Individual test sets in the population must be evaluated to determine their fitnesses. Fitness is used both in determining which individuals are selected for crossover, and which individuals are allowed to survive across generations. The fitness function $f(x)$ is a weighted function with two terms, $f_s(x)$ and $f_c(x)$, that are related respectively to size and coverage. Both $f_s(x)$ and $f_c(x)$ have the following form

$$\frac{\mu - score(x) + \alpha\sigma}{2\alpha\sigma}$$

where μ and σ are respectively the mean and standard deviation of the population scores (i.e., size or coverage) and α is a normalization constant.

We investigated two approaches for fitness evaluation that are related to the approach of Carter *et al.* [CART85]. One approach uses fault simulation to gather statistics regarding test set size and coverage; the other approach uses a covering heuristic. Due to the nature of our fault simulator, it was not possible to combine effectively these two approaches.

When pure fault simulation is used for fitness evaluation, it is very desirable to minimize the total amount of such simulation. This follows from the computational complexity of fault simulation which is at least $O(n^2)$ [DUBA85], where n is a measure of circuit complexity. Thus, for algorithmic feasibility it is necessary that the genetic operators and parameters be chosen in a manner that guarantees rapid convergence of the algorithm to a satisfactory solution. Even if heuristics are used to reduce the amount of fault simulation, the cost of fault simulation for offspring evaluation remains the dominating factor in the GA's cost.

Our alternative to fault simulation for fitness evaluation, was a covering heuristic. If a reasonably efficient heuristic is used, the overall algorithmic efficiency may improve significantly. The appeal of the covering heuristic stems from the repeated simulation of the same test patterns as the patterns disseminate throughout the population. With the covering heuristic approach, a single simulation is performed for each test pattern and the result is recorded in a fault table. Fitness evaluation is calculated using the fault table. The principle drawback of this approach is the size of the table, which is a function of the number of test patterns and faults.

The "wallclock" cost of fitness evaluation may be improved with a parallel or distributed implementation. Since the evaluation of each offspring is largely independent, acceleration of fitness evaluation step is possible through the use of a parallel or distributed processor architecture. In a related research project, we are evaluating the effectiveness of a new parallel genetic algorithm [COHO91].

4. IMPLEMENTATION DETAILS

In the experiments summarized in the next section, we used a population size of ten test sets. Each test set was derived from a different random collection of 256 tests that had a *SOFE*/ 1-like procedure applied to them before they were incorporated into the initial population. Our preliminary experiments indicated that both sufficient diversity was present and that the computation costs associated with GATE's genetic algorithm operators remained feasible. If a circuit of far different complexity is to be processed, the number of tests and the number of test sets may need to be modified.

We found in our preliminary experiments that the preferred crossover operator was a simple one-way crossover that divided a test set u into approximately four equally-sized sections, T_u, I_1, I_2, and B_u. A section from one parent was combined at the beginning of a second parent. The section selected to combine was chosen probabilistically using a crossover weight vector.

Since the available fault simulator was relatively primitive, a covering heuristic became the method of choice to determine fitness. The covering heuristic that we used was a straight-forward greedy one that repeatedly selected from a given test set, the test pattern that covered the greatest number of remaining faults. The only real efficiency that was introduced was a pre-processing step that automatically included all test patterns that uniquely covered a fault.

The preferred mutation operator was one that produced a significantly different test set, but not one so different that it was nearly a completely new, randomly generated test set. The mutation operator that worked best, inserted a small number of new patterns at or near the beginning of the test set to be mutated.

5. EXPERIMENTAL RESULTS

As a result of a preliminary investigation using two simple circuits and three circuits from *ISCAS-85* ATPG benchmark set [BRGL85], a baseline implementation of GATE was constructed. The parameters settings for the baseline GATE are defined in Table 1.

Population size: 10
Crossover rate: 100%
Crossover type: one way
Sections: 4
Crossover weight vector: (0.3, 0.125, 0.125, 0.45)
Self-crossover: not permitted
Mutation rate: 10%
Mutation range: 25%
Fitness relative weights: (0.5, 0.5)
Maximum initial test set size: 256
Number of generations: 18

Table 1. — Baseline GATE parameter settings.

The table indicates that for each of the 18 generations of the GA, GATE creates 10 offspring (i.e., 100% crossover rate). In creating a test set, 256 test patterns are generated one at-a-time. If the current pattern covers a new fault with respect to the patterns in the test set so far, it is added to the test set. The fitness function evenly weights the contributions of test size and coverage. The table also indicates that the one-way crossover operation probabilistically favors the bottom section the most, the top section next, and the two interior sections the least. Mutation insertions are done in the first 25% of a selected test set.

Results associated with C432 from the ATPG benchmark using fault simulation for fitness calculation are typical of the five circuit instances, and are summarized in Table 2. The results demonstrate that GATE is a very effective tool with respect to both figures of merit (i.e., coverage and size). On average it improved the initial population test sets with respect to fault coverage by 2.7% and with respect to test set compaction by 18.7%. Note, in this and subsequent tables that labels C and S denote respectively coverage and size.

Statistic	Average Initial *SOFE* Results		Baseline GATE Results		GATE Population Improvement	
	C (%)	*S*	*C* (%)	*S*	*C* (%)	*S* (%)
Worst	95.15	53.3	98.50	43.0	2.2	17.6
Best	96.26	51.0	98.50	42.0	3.4	19.5
Average	95.80	52.2	98.50	42.4	2.7	18.7

Table 2. — Preliminary results for C432.

Since the cost of using the fault simulator to determine fitness was deemed too great for its use subsequently, the remaining experiments used the covering heuristic to evaluate fitness.

Tables 3 and 4 summarize 20 trials of both our baseline GATE and a GATE-variant with a single interior section on the ATPG benchmarks C432 and C3540. For both circuits, the GATE implementations were quite effective with respect to the two figures of merit. We also note that the use of covering heuristic improved GATE's performance with respect to the preliminary experiment where GATE used fault simulation to determine fitness. The labels "average", "max", and "min", indicate the standard associated statistic with respect to the population in question. The label "top" refers to the test set with the best fitness with respect to the population in question.

Statistic	Initial Population		Baseline GATE		Variant GATE	
	C (%)	*S*	*C* (%)	*S*	*C* (%)	*S*
Average	96.35	51.7	99.24	39.3	99.24	39.4
Max	97.56	59.0	99.25	44.0	99.25	43.0
Min	93.05	46.0	99.06	37.0	99.06	37.0
Top	97.18	49.0	99.25	37.0	99.25	37.0

Table 3. — Results for C432 with covering heuristic.

Statistic	Initial Population		Baseline GATE		Variant GATE	
	C (%)	*S*	*C* (%)	*S*	*C* (%)	*S*
Average	94.29	153.4	95.90	128.3	95.97	128.0
Max	94.77	158.0	95.97	133.0	95.97	128.0
Min	93.01	146.0	95.83	122.0	95.95	128.0
Top	94.77	152.0	95.83	122.0	95.97	128.0

Table 4. — Results for C3540 with covering heuristic.

Note in Table 3, the maximum coverage shown for C432 is 99.25%, which is greater than the maximum value of 99.23% given by Carter *et al.* [CART85]. This small discrepancy exists because in the TEGAS system, gates with more than four inputs are expanded into multiple gates, thus slightly altering the total number of faults and the maximum possible coverage. This situation also applies to circuit C3540.

Table 5 compares the performances of GATE and *SOFE*/ 2. This table (as with the previous tables) indicates statistically that GATE is a very effective compactor. For ATPG circuits C432 and C3540, GATE produces a typical improvement of approximately 24% and 16% respectively with respect to *SOFE*/ 2 (regardless of the statistic in question). Although not indicated in the table, GATE's coverage was higher by approximately 3% and 2% respectively for the two circuits.

Circuit	*SOFE*/ 2 Size		GATE Size		Improvement (%)	
	Average	Best	Average	Best	Average	Best
C432	51.7	49.0	39.4	37.0	23.8	24.5
C3540	153.4	152.0	128.00	128.0	16.5	15.8

Table 5. — Performance comparison with *SOFE*/ 2.

Table 6 compares the performance of GATE with *SOFE*/ 6 as the previous table did with *SOFE*/ 2. Once again GATE proves to an effective compactor—it produced solutions for the two circuits in question that were on average approximately 12% and 14% better respectively than the complete *SOFE* scheme. Similarly, with respect to the best performance statistic, GATE was able to produce a solution almost 18% better. Although not indicated in this table, GATE's coverage was marginally higher on average than the complete *SOFE* scheme by approximately 0.01%. However, we do not believe that GATE's coverage improvement is statistically significant for this circuit.

Circuit	*SOFE*/ 6 Size		GATE Size		Improvement (%)	
	Average	Best	Average	Best	Average	Best
C432	45.0	45.0	39.4	37.0	12.4	17.8
C3540	149.0	149.0	128.00	128.0	14.1	14.1

Table 6. — Performance comparison with *SOFE*/ 6.

From these experiments, we conclude that the GATE approach is a very effective test tool. In particular, the GATE implementation using a covering heuristic produces superior test set solutions—its solutions are experimentally on average 13% better than those produced by the traditional *SOFE* techniques.

6. COMPUTATION COSTS

Besides test set size and coverage, the other important characteristic of a test set generator is computation cost. The standard figure of merit for the cost of fault simulation is *gate-fault-patterns*. This is a fair estimate of the fault simulation effort, if the number of faults being simulated at every point in the fault simulation procedure is used. In estimating the costs, we assumed that all gates are simulated for the remaining undetected faults. Using this approach it is possible to estimate and compare the costs of *SOFE*/ 6, the GATE implementation using fault simulation, and the cost of the initial population generation for the GATE implementation using the covering heuristic.

It is difficult to estimate the total cost of GATE implementation using the covering heuristic, as the *gate–fault–patterns* metric is not applicable. Thus, to relate the cost of a specific amount of fault simulation effort to a specific amount of covering heuristic effort, a proportionality constant β is used. Constant β relates the cost of a basic unit of fault simulation work with a basic unit of covering heuristic work, where the basic unit of work is the evaluation of one offspring from a crossover operation. In our preliminary GATE implementation, it was observed that the computation time for evaluating one offspring by a covering heuristic is approximately 10% of that of the fault simulation cost. Accordingly, in the computation cost results presented below, the ratio is assumed to be in the range 0.05 to 0.25.

Figures 3 and 4 show the range of differences between the costs of the approaches. Both graphs show the ratio of the two GATE approaches for the C3540

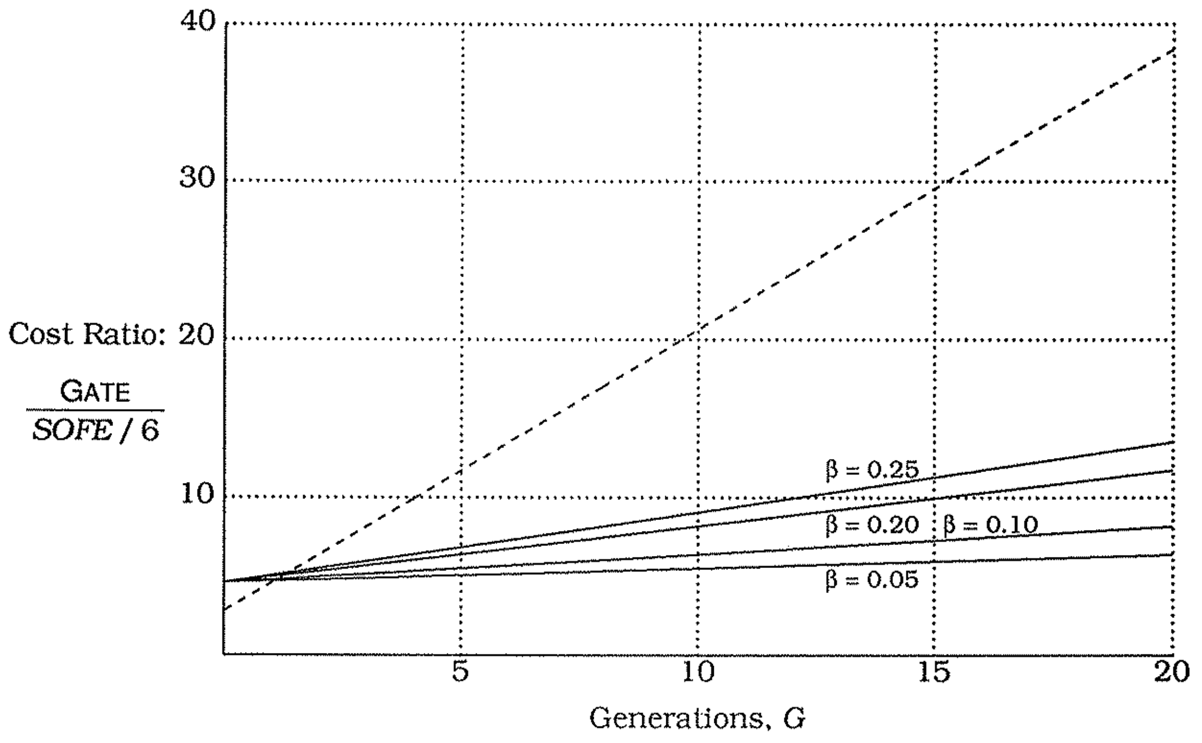

Figure 3. — Relative costs of GATE and *SOFE*/ 6 for C3540.

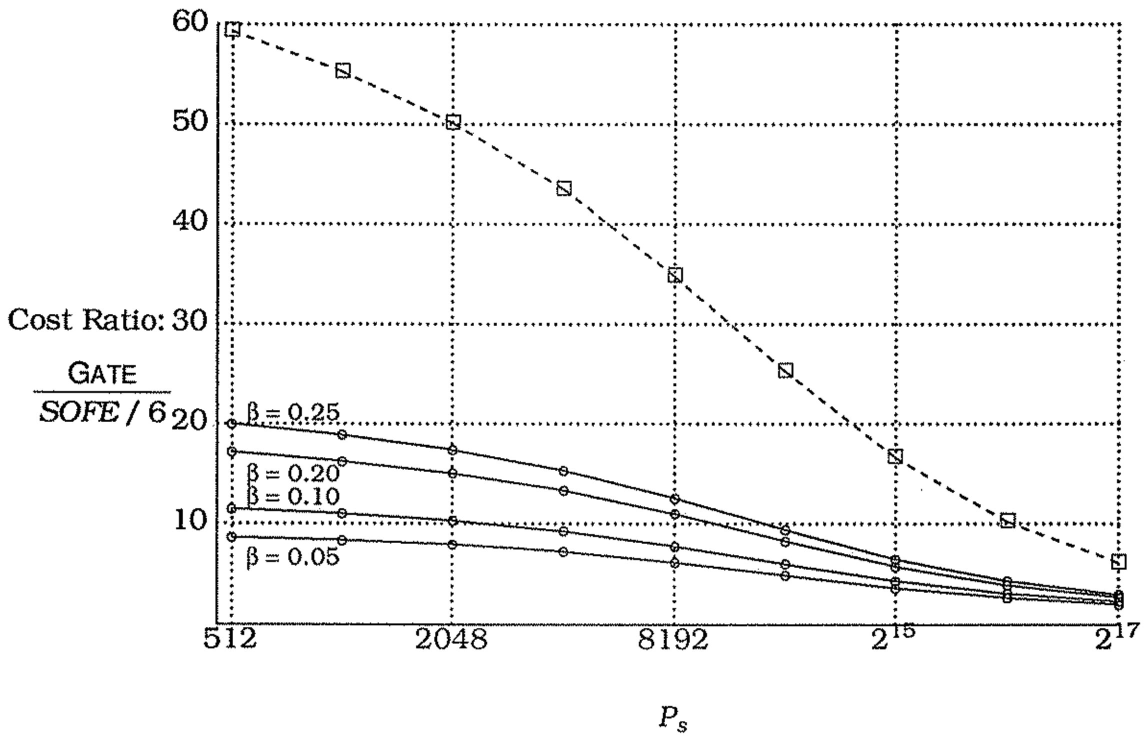

Figure 4. — Relative cost of GATE and *SOFE* / 6 for C3540.

benchmark to *SOFE* / 6. The solid curves are for the covering heuristic implementation, and the dashed curve is for the fault simulation implementation. The parameter β in the figures is the proportionality constant discussed above. In the computations for both graphs, the number of patterns, P_g, simulated in GATE's initial population generation is one eighth of those simulated in the *SOFE* / 1 portion of *SOFE* / 6 (P_s). This is a reasonable assumption for two reasons. Firstly, each method begins with approximately the same coverage potential; and secondly, the population in all GATE instances is comprised of 10 test sets. Figure 3 shows the relationship between β, the number of generations, *G*, and the cost ratios. The number of patterns used here for initial population generation is the same as those used for the baseline results of the previous section. Figure 4 shows the relationship between the number of patterns initially generated and the ratios. In this figure, the number of generations (18) is the number used typically by GATE. Note that the cost ratio improves as the number of initial patterns grows larger.

While the graphs demonstrate the marked superiority of the GATE implementation using a covering heuristic over the GATE implementation using fault simulation, they also demonstrate that GATE's solution quality over *SOFE* comes at a price. Its computation cost is approximately six to nine times greater than the computation cost of *SOFE* / 6.

These costs imply that GATE should be used in an environment where test set length has significance. The most obvious example is for integrated circuits manufactured in very large quantities, where any tester time savings would be multiplied because of the large quantities. The additional computation cost required by GATE would be justified because of the multiplied tester time savings. A more specific and possibly more appropriate area is in the testing of circuits using one of the scan techniques such as LSSD [EICH78] or Scan Path [FUNA75]. Such scan techniques require each bit of each test pattern to be shifted into scan strings, the savings associated with eliminating a single pattern will consist of the test pattern application and output comparison time, *and* as many shift operations as the longest scan string in the circuit. These savings could be multiplied even further if the integrated circuit is manufactured in high volumes. Another potential area is in the testing of very large combinational circuits. Because of the exponential growth of the computation requirements of purely deterministic test generation, test generation might be feasible for such circuits only using methods based upon, at least partially, random pattern generation and fault simulation, as fault simulation costs are bounded by a polynomial. If test set length is a concern, then the GATE test generation approach or the *SOFE* / 6 approach are good candidates for creating the compact test sets. The preference of GATE over *SOFE* depends upon the relative importance of test set length.

7. SUMMARY AND CONCLUSIONS

This paper describes GATE, a new approach to test generation and compaction. GATE uses the adaptive searching capabilities of the genetic algorithm paradigm to find compact test sets from a population of randomly generated test sets.

Two implementations of GATE were performed. Based on experimental analysis, we conclude that the implementation using a covering heuristic for fitness evaluation provides the best results, both in terms of test set quality and computation cost. In particular, for two well-known benchmark circuits GATE produced test sets that are approximately 20% smaller than a random starting configuration. To place this in perspective, GATE's solutions are on average 13% more compact than results obtained by the traditional random test generation and compaction technique *SOFE* /6 with no loss in fault coverage. In fact, GATE produces on average a solution with a slight coverage increase. We estimate the computation cost of a non-parallel GATE approach to be about seven times more than the cost of the *SOFE* /6. However, a parallel GATE implementation is estimated to be able to reduce the wallclock computation cost by a factor in the range of 2 to 7.

8. ACKNOWLEDGEMENTS

The authors' work has been supported in part by the Virginia Center for Innovative Technology through grant 5-30971 and by the Defense Advanced Projects Research Agency (DARPA) through contract N00014-89-J1699. Their support is greatly appreciated.

9. REFERENCES

[BRAH87] D. S. Brahme and J. A. Abraham, Knowledge Based Test Generation for VLSI Circuits, *IEEE International Conference on Computer-Aided Design*, Santa Clara, CA, November 1987, 292-295.

[BRGL85] F. Brglez, R. Pownall and R. Hum, Accelerated ATPG and Fault Grading via Testability Analysis, *Proceedings of the 1985 IEEE International Symposium on Circuits and Systems*, June 1985, 695-698.

[CART85] J. L. Carter, S. F. Dennis, V. S. Iyengar and B. K. Rosen, ATPG via Random Pattern Simulation, *Proceedings of the 1985 International Symposium on Circuits and Systems*, June 1985, 683-686.

[COHO87] J. P. Cohoon and W. D. Paris, Genetic Placement, *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems CAD-6*,6 (November 1987), 956-964.

[COHO91] J. P. Cohoon, S. U. Hegde, W. N. Martin and D. S. Richards, Floorplan Design Using Distributed Genetic Algorithms, *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems CAD-9*(1991).

[DUBA85] P. A. Dubam, R. K. Roy, J. A. Abraham and W. A. Rogers, Fault Simulation in a Distributed Environment, *22nd Design Automation Conference Proceedings*, Las Vegas, NV, 1985, 686-691.

[EICH78] E. B. Eichelberger and T. W. Williams, A Logic Design Structure for LSI Testability, *Journal of Design Automation and Fault Tolerant Computing 2*(May 1978), 165-178.

[FUJI82] H. Fujiwara and S. Toida, The Complexity of Fault Detection Problems for Combinational Logic Circuits, *IEEE Transactions on Computers C-31*(June 1982), 555-560.

[FUJI83] H. Fujiwara and T. Shimono, On the Acceleration of Test-Generation Algorithms, *IEEE Transactions on Computers C-32*(December 1983), 1137-1144.

[FUJI85] H. Fujiwara, FAN: A Fanout Oriented Test Pattern Generation Algorithm, *Proceedings of the 1985 IEEE International Symposium on Circuits and Systems*, 1985, 671-674.

[FUNA75] S. Funatsu, N. Wakatsuki and T. Arima, Test Generation Systems in Japan, *Proceedings of the 12th Design Automation Conference 2*(June 1975), 114-122.

[GOEL81] P. Goel, An Implicit Enumeration Algorithm to Generate Tests for Combinational Logic Circuits, *IEEE Transactions on Computers C-30*(March 1981), 215-222.

[GOLD89] D. E. Goldberg, *Genetic Algorithms in Search, Optimization, and Machine Learning*, Addison-Wesley, Reading, MA, 1989.

[HOLL75] J. H. Holland, *Adaptation in Natural and Artificial Systems*, University of Michigan Press, Ann Arbor, MI, 1975.

[IBAR75] O. H. Ibarra and S. K. Sahni, Polynomially Complete Fault Detection Problems, *IEEE Transactions on Computers C-24*(March 1975), 242-249.

[LISA86] R. Lisanke, F. Brglez, A. de Geus and D. Gregory, Testability-Driven Random Pattern Generation, *IEEE International Conference on Computer-Aided Design Proceedings*, Santa Clara, CA, November 1986, 144-147.

[MCCL81] E. J. McCluskey and S. Bozorgui-Nesbat, Design for Autonomous Test, *IEEE Transactions on Computers C-30*(November 1981), 866-875.

[ROTH66] J. P. Roth, Diagnosis of Automata Failures: A Calculus and a Method, *IBM Journal of Research and Development Vol. 10*(July 1966), 278-291.

[SMIT85] D. Smith, Bin Packing with Adaptive Search, *Proceedings of an International Conference on Genetic Algorithms and Their Applications*, Pittsburgh, PA, 1985, 202-206.

[SUH87] J. Y. Suh and D. V. Gucht, Incorporating Heuristic Information into Genetic Search, *Proceedings of the Second International Conference on Genetic Algorithms and Their Applications*, Boston, MA, 1987, 100-107.