Implementation of a combinatorial digital circuit with Genetic Algorithms

Seema Sharma[#1], Dr Ajay Kumar[#2]
#1 AP, ICE Dept. , DAV University, Jalandhar,
#2 HOD, ECE Dept., Beant College of Engineering, Gurdaspur

ABSTRACT

Genetic Algorithms (GAs) are adaptive heuristic search algorithms premised on the evolutionary ideas of natural selection and genetics. Many of the real world problems involved finding optimal parameters, which might prove difficult for traditional methods but ideal for GAs. However, because of their outstanding performance in optimization, GAs have been wrongly regarded as a function optimizer alone. Our aim in this paper is to present a simple method to synthesize a simple circuit with some fitness function required for any VLSI circuit and reduce the gate complexity of the circuit.

Key words: GA, selection, cell crossover, mutation, schemata

Corresponding Author: Seema Sharma

INTRODUCTION

The basic concept of GAs is designed to simulate processes in natural system necessary for evolution, specifically those that follow the principles first laid down by Charles Darwin of survival of the fittest [1]. As such they represent an intelligent exploitation of a random search within a defined area. First pioneered by John Holland in the 60s, Genetic Algorithms have been widely studied, experimented and applied in many fields in engineering worlds. Not only do GAs provide alternative methods to solving problems, they consistently outperform other traditional methods in most of the problem domains.

STRENGTHS OF GENETIC ALGORITHMS

The first and most important point is that genetic algorithms are intrinsically parallel. Most other algorithms are serial and can only explore the solution space to a problem in one direction at a time, and if the solution they discover turns out to be suboptimal, there is nothing to do but abandon all work previously completed and start over. However, since GAs have multiple offspring, they can explore the solution space in multiple directions at once. If one path turns out to be a dead end, they can easily eliminate it and continue work on more promising avenues, giving them a greater chance each run of finding the optimal solution [1].

Schema Theorem

However, the advantage of parallelism goes beyond this. Consider the following: All the 8-digit binary strings (strings of 0's and 1's) form a search space, which can be represented as ******** (where the * stands for "either 0 or 1"). The string 01101010 is one member of this space. However, it is also a member of the space 0*******, the space 01******, the space 0******0, the space 0*1*1*1*, the space 01*01**0, and so on. By evaluating the fitness of this one particular string, a genetic algorithm would be sampling each of these many spaces to which it belongs. Over many such evaluations, it would build up an increasingly accurate value for the average fitness of each of these spaces, each of which has many members. Therefore, a GA that explicitly evaluates a small number of individuals is implicitly evaluating a much larger group of individuals – just as a pollster who asks questions of a certain member of an ethnic, religious or social group hopes to learn something about the opinions of all members of that group, and therefore can reliably predict national opinion while sampling only a small percentage of population. In the same way, the GA can "home in" on the space with the highest fitness individuals and find the overall best one from that group. In the context of evolutionary algorithms, this is known as Schema Theorem, and is the "central advantage" of a GA over other problem solving methods [2,3].

Due to the parallelism that allows them to implicitly evaluate many schema at once, genetic algorithms are particularly well-suited to solving problems where the space of all potential solutions is truly huge – too vast to search exhaustively in any reasonable amount of time. Most problems that fall into this category are known as "nonlinear". In a linear problem, the fitness of each component is independent, so any improvement to any one part will result in an improvement of the system as a whole. Needless to say, few real world problems are like this. Nonlinearity is the norm, where changing one component may have ripple effects on entire system, and where multiple changes that individually is detrimental may lead too much greater improvements in fitness when combined. Nonlinearity results in a combinatorial explosion: the space of 1,000-digit binary strings can be exhaustively searched by evaluating only 2,000 possibilities if the problem is linear, whereas if it is nonlinear, an exhaustive search requires evaluating 21000 possibilities – a number that would takeover 300 digits to write in full. Fortunately, the implicit parallelism of a GA allows it to surmount even this enormous number of possibilities, successfully finding optimal or very good results in a short period of time after directly sampling only small regions of the vast fitness landscape [4].

Processing of Schemata

Each individual in the population is an instance of 2^m schemata, where m is the length of each individual string. For instance, the length-2 bit string '11' is an instance of **, 1*, *1, and 11. In a population of n-strings, it is easy to see that there will be anything between 2^m and $n.2^m$ different schemata. Thus, the population represents a very large number of schemata, even for relatively small population sizes. It can, therefore, be said that while the GA explicitly evaluates n strings during any given generation, it is implicitly evaluating a much larger number of schemata. This effect is known as intrinsic parallelism in GAs [1,4].

After a given number of generations have been run, a rough estimate of the average fitness of all its schemata or sub-placements can be obtained. The estimates of these averages are not stored explicitly since the schemata themselves are not explicitly evaluated. However, it is possible, as in the next section, to look at the increase or decrease of a given schema in a

population - because it can be described as though the GA were storing these implicitly calculated values.

With each new generation examined, the number of each of its 2^m to $n.2^m$ different schemata present in the population is adjusted according to its fitness. As more generations are tried out, the relative proportions of the various schemata in the population reflect their fitness values more and more accurately. When a fitter schema is introduced in the population through an offspring, it is inherited by others in the succeeding generation, and thus, its proportion in the population increases. It starts driving out the less fit schemata, and the average fitness of the population improves [1,4]

From the previous related research [1,6], the number of schemata is correlated to the total number of similarity or diversity bits. However, a difficulty arises when the total similarity bits count for two different populations is same but the breakup of similarity bits count is different. The unique schemata count is correlated with the breakup of similarity bits count.

Unique schemata count = n.2m – (n–1)2m–1

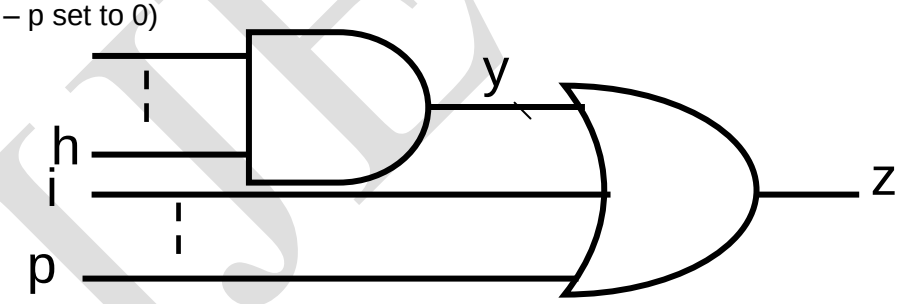
This relationship has been developed for a restricted case of maximum similarity bits count being just one less than the length of the string (i.e., s(b) = m–1) which means that the strings are non-repeated and further that the count of such maximum similarity bit counts is (n-1).

PROBLEM FORMULATION

Many of the optimization problems encountered in VLSI design, layout, and test automation can be solved with high-quality results using genetic algorithms. Consider the problem of test generation for a very simple circuit. A simple GA was implemented with the following fitness function to solve this problem:

FITNESS FUNCTION

Fitness = (1 if fault is excited or 0 otherwise) +(fraction of inputs a – h set to 1) +(fraction of inputs i – p set to 0)



GA – based test generation

To excite the fault node Y must be driven to logic 1 in the fault free circuit, so that there will be a difference between the fault free circuit and faulty circuit. Nodes a through h must therefore be set to 1. To propagate the fault effects to output node Z, nodes i through p must be set to 0. The required test vector is nearly impossible to find using a random approach. Specific values are required on 16 circuit inputs, and it is very unlikely that correct combination will be generated randomly. If the GA fitness function simply indicates whether or not the fault is detected, the test is very unlikely to be found. However if the fitness function favors setting nodes a through h to 1 and setting nodes i through p to 0, the test has a good chance of being found [11].

EXPERIMENTAL RESULTS

FITNESS OF POPULATION IS

Population	Fitness
1001110110011101	1
1001110011101111	0.625
1111110011101101	1
1011110011101111	0.75
1101001110111101	0.875
1001001110101101	0.875
1001001110100101	1
1111001110110000	1.375
1001111001001101	1.125
1001110010011101	0.875
1001111110011101	1.125
1010010011011101	0.625

Max fitness = 1.375 Avg fitness = 0.9375

Finding next Generation
Selecting parents....
Selected parents

9
10
First parent::1001110010011101
Second parent::1001111110011101

For Crossover

Press 'e' for entering crossover probability else 'd' for default d

Cut point for cross-over is 10
1st child is1001110010011101
2nd child is1001111110011101
Fitness after Crossover
0.875
1.125
For Mutation

Enter e for entering probability of mutation,press d for default value d
No mutation
Fitness after Mutation

_____

0.875
1.125

4th Generation
POPULATION ALONG WITH FITNESS

_____

Population	Fitness
1001001110101101	0.875
1001111001010000	1.375
1001101110101101	1
1001111001001101	1.125
1111110110010000	1.625
1111001101001101	1.25
1111001110010000	1.5
1111110110111101	1.125
1001110010111101	0.75
1111001101001101	1.25
1111110001001101	1.25
1001011110010000	1.375

Max fitness = 1.625 Avg fitness = 1.20833

CONCLUSION

Genetic Algorithm was conceived by Holland in 1975, and since then, it has emerged as a possible solution for many search and optimization problems. In the area of VLSI test, GA has been successfully used in stuck - at fault test [11]. The GA may be viewed as an evolutionary process where in the population of feasible solutions to the optimization problem evolves over a sequence of generations. Each solution is represented as a string of elements and is assigned a fitness value based on the value given by an evaluation function. The fitness value measures how close the individual is to the optimal solution. A set of individuals constitutes a population that evolves from one generation to the next through the creation of new individuals and the deletion of some old ones. The process usually starts with an initial population created randomly. The integrated Circuit fabrication process is prone to random defects, which may affect the

functionality of a device and cause erroneous outputs. Testing of finished circuits is essential in weeding out defective parts to ensure that electronic systems build using the ICs function correctly. Sets of test vector applied to circuits by a tester must have high defect coverages if they are to be effective in identifying defective chips. Furthermore, since the cost of testing VLSI chips is a significant fraction of the overall manufacturing cost, the time required to test a chip should be minimized. Effective tools for automatic test generation (ATG) are needed to obtain compact test sets with high defect coverage. Here, we have discussed how genetic algorithms can be used for automatic test generation. Genetic algorithms have been very effective for sequential circuit test generation, especially when combined with deterministic algorithms [11,17]. We begin with an over view of test generation problem and then discuss how tests can be generated in a GA framework. GA parameters and fitness function and an implementation is described. A Genetic Algorithm Frame work is developed for use in sequential circuit test generation. Populations of candidate tests are evolved by the GA starting from a random initial population.

FUTURE SCOPE

Genetic algorithms are little understood because the genetic transformations and iterations through generations are difficult to model by known analytical techniques. Although some probabilistic analyses were made by John Holland and others and schema theorems were proposed to explain the genetic optimization process, no clear and insightful analytical techniques yet exist that can explain rigorously the convergence of genetic algorithms or how many iterations are required to obtain the globally optimal solution or a solution within some predefined infinitesimally close range of globally solution [11]. Genetic algorithms have been used successfully to solve numerous different problems in VLSI design, layout and test automation. We sincerely hope that in the future, GAs will prove to be a general – purpose powerful heuristic method for solving a wider class of engineering and scientific problems.

REFERENCE

[1] M. Mitchell, "An Introduction to Genetic Algorithms," *New Delhi: PHI Pvt. Ltd.*, 1998.
[2] D.E. Goldberg, "Genetic Algorithms in Search, Optimization and Machine Learning," *Reading, MA: Addison-Wesley*, 1989
[3]J.H. Holland, "Outline for a logical theory of adaptive systems," *J. Assoc. Comput. Mach.*, vol.3, pp.297-314, 1962.
[4] J.S. Saini, et.al., "An alternative to schemata Count for Genetic Algorithms," *All-India Seminar on Power & Energy for Sustainable Growth (PESG-2003) organized by IE(I) Hr. Centre & CRSCE, Murthal,* Pp. 41-48, Feb. 20-21, 2003.
[5] Khushro Shahookar and Pinaki Mazumder, "A Genetic Approach to standard Cell Placement Using Meta – Genetic Parameter Optimization," *IEEE Transactions on Computer Aided Design*, vol. 9, No. 5, May 1990.
[6] Pinaki Mazumder and Elizabeth M. Rudnick, "Genetic Algorithms for VLSI Design, Layout & Test Automation," *Delhi: Pearson Education (Singapore) Pte. Ltd., Indian Branch*, 2003.

[7] N. Sherwani, "Algorithms for VLSI Physical Design Automation", *Kluwer Academic Publishers*,1993
[8] V. Schnecke, O. Vornberger, "Genetic Design of VLSI-Layouts", *Procs. First IEE/IEEE Int. Conf. on GAs in Engineering Systems: Innovations and Applications, GALESIA'95, IEE Conference Publication No*. 414, 1995, 430-435
[9] S. M. Sait, H. Youssef, "VLSI Physical Design Automation: Theory and Practice", *McGraw-Hill* (1995)
[10] K. Shahookar and P. Mazumder ,"Genetic multiway partitioning," Proc. *IEEE Int. conf. VLSI Design*, New Delhi, India, pp.365-369, Jan. 1995
[11] E.J. McCluskey and S.B. Nesbat, "Design for autonomous test," *IEEE Trans. Computers*, vol.C-30, pp.866-874, Nov. 1981.
[12] E.M. Roudnick and J.H Patel," A Genetic Algorithm framework for test generation," *IEEE Trans. Computer-Aided Design*, vol. 16, no. 9, pp. 1034-1044, Sept.1997
[13] 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*, pp.216-219, Nov.1992.
[14] T. M. Niermann, "Techniques for sequential circuit automatic test generation," *Technical Report, coordinated science laboratory*, March 1991.
[15] D. Schlierkamp-Voosen, H. Mühlenbein," Strategy Adaptation by Competing Subpopulations," *3rd Conf. on Parallel Problem Solving from Nature, Springer Lecture Notes in Computer Science* 866, 1994, 199-208
[16] Hideyuki Takagi and Kimiko Ohya," Discrete Fitness Values for Improving the Human Interface in an Interactive GA," *Proceedings of 1996 IEEE International Conference on Evolutionary Computation (ICEC'96)*, 109-12, 1996.
[17] Ackley, D. H," A connectionist algorithm for genetic search, " *proceedings of an international conference on Genetic Algorithms and their applications*, pp.121-135
[18] H.Chan and P. Mazumder, " Genetic Algorithm and Graph Partitioning," *Proc. AAAI Conf., Sydney, Australia*, Nov. 1994.
[19] V. Chickermane and J. H. Patel, "An optimization based approach to the partial scan design problem," *Proc. Int. Test Conf.*, pp.377-386, 1990.
[20] K.T. Cheng and V. D. Agrawal, "An economical scan design for sequential logic test generation," *Proc. 19th Int. Symp. Fault-Tolarant Computing*, pp. 28-35, 1989.