Journal of Engineering Science and Technology Review 9 (5) (2016) 88 - 92

Research Article

JOURNAL OF
Engineering Science and Technology Review

www.jestr.org

A Cartesian Genetic Programming Approach for evolving Optimal Digital Circuits

S. A. Kazarlis*, J. Kalomiros, and V. Kalaitzis

Department of Informatics Engineering, Technological Educational Institute of Central Macedonia, Serres, 62124, Greece

Received 30 June 2015; Accepted 25 January 2016

Abstract

In this work, a Cartesian Genetic Programming (CGP) method is presented for the evolutionary synthesis of optimal digital circuits. In this evolutionary method, circuits are represented as directed graphs in the form of an MxN Cartesian grid. Evolved circuits are qualified using a fast custom emulator of digital circuits that was built to serve as a fitness function for the algorithm. The performance of the CGP algorithm is tested on six different digital circuits, used as benchmarks. The evolution results prove the ability of the CGP scheme to find optimal solutions with significant probabilities. Moreover, the CGP algorithm is able to produce unconventional solutions for known circuits.

Keywords: Cartesian Genetic Programming, Digital Circuits, Evolvable Hardware, Digital Circuit Emulation.

1. Introduction

The field of Evolutionary Algorithms (EAs) [1], [2], is quite young and describes a large set of stochastic optimization methods inspired from biological evolution and natural systems. Numerous times in the literature, EAs have proved their merits as powerful optimizers of difficult real world problems. Genetic Programming (GP) [3], [4], is such a method that was primarily used for evolving software that was encoded using a special tree encoding scheme. But soon researchers extended GP's applications by applying them to other real world problems with tree-encoded solutions, like the optimal design of analog and digital circuits [5], [6]. The evolution of circuits has moved ahead by evaluating potential solutions on FPGA-like platforms and thus another EA was born called Evolvable Hardware (EH) [7], [8].

A variation of GP, named Cartesian Genetic Programming (CGP), was proposed by Miller and Thomson [9]. In CGP, circuits are not encoded as trees, but as directed graphs that usually have the form of MxN Cartesian grids. In cases where such grids are used for encoding digital circuits [17], [18], the grids contain MxN nodes, each representing a digital gate. Moreover, these nodes can interconnect with each other, forming arbitrary circuits of combinatorial or sequential nature.

For the application of any EA on any optimisation problem one has to build a Fitness Function [10], in order to be able to evaluate every evolved solution. This Fitness Function can be seen as a mapping function between the set of possible solutions and the set of real numbers, assigning a quality value to each possible solution. In our work the Fitness Function should be able to qualify proposed digital circuits. Thus a special Digital Circuit Emulator [11] was built in order to cover this need for the CGP algorithm. The Digital Circuit Emulator was built as a function that receives the circuit under evaluation as its input in a string-encoded form. It is capable of simulating both combinatorial and sequential circuits of up to 500 gates, and can be extended even more by altering its input encoding.

The proposed CGP implementation that uses the Digital Circuit Simulator is applied on six (6) digital circuit synthesis problems. Each problem corresponds to a well-known digital circuit whose truth table is given as a specification to the CGP algorithm. The required task is to find an optimal digital circuit which a) exactly matches the specified truth table, and b) minimizes the necessary number of gates. The digital circuit synthesis problem defined above can be characterized as a Multi-Objective optimization problem [12]. Usually Multi-Objective optimisation problems need the construction of a consolidated fitness function that will combine all individual objective functions into one.

In order to extract useful information about the performance of the CGP scheme, a large number of simulation runs have been scheduled, for different sizes of the grid structure. The results of this effort are presented and discussed in this article. Moreover, the CGP scheme has revealed another merit: the ability to evolve unconventional designs of a particularly small number of gates. These solutions have to be further examined and studied against the literature.

The organization of the paper is as follows: in Section 2, the Cartesian Genetic Programming implementation is presented. The digital circuit simulator is described in Section 3. In Section 4, the test set used for testing the CGP scheme is described. The simulation results and conclusions are presented in Section 5.

2. The Cartesian Genetic Programming Implementation

2.1. The Cartesian Grid Structure

The Cartesian Genetic Programming (CGP) approach used in this work represents a possible circuit as an MxN grid. Each node on the grid represents a digital gate that is defined by the evolutionary algorithms from a set of available Boolean functions. Also, the evolutionary algorithm defines

* E-mail address: kazarlis@teicm.gr

the interconnections between the grid nodes, in order to form a potential circuit. In this work, only forward connections are allowed between gates of adjacent columns. However, circuit inputs can be connected not only to the gates of the first column but to any gate in the grid. Moreover, the outputs of the circuit can be drawn not only from the gates of the last column, but from any gate. A grid of this form with a 3x3 dimension can be seen in Fig. 1.

Each gate in the grid can be chosen among the following values: 0. Non-existent, 1. AND, 2. OR, 3. NOT, 4. NAND, 5. NOR, 6, XOR, 7, XNOR. The "non-existent" value allows a circuit to have less gates than MxN. The minimization of the number of gates is also an optimization goal.

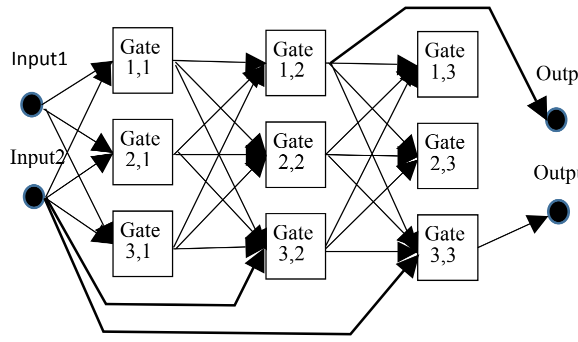

Fig. 1. A 3x3 Cartesian Grid for digital circuit evolution

All gates are considered as two-input, one-output gates except for the NOT gate that has a single input. A gate's input can connect: a) to a circuit's input, b) to the output of a gate of the previous column, and c) to logic 0 or 1.

2.2. The genetic representation of solutions

Evolutionary Algorithms usually operate on solution spaces formed by encoding solutions in symbol strings of a usually binary alphabet. Thus, the following representation scheme has been adopted: each grid node corresponds to a binary chromosome that encodes the gate type and the interconnections of this gate. Each chromosome comprises 4 subparts:

a) gate-type subpart: a 3-bit subpart enough to encode all 8 different node configurations.
b) gate-input-1 subpart: a subpart encoding the connection of the first input of the gate. The number of bits needed for this subpart is:

$$ceil(log2\ (\ NoOfGridRows+NoOfCircuitInputs+2\ )\ ) \qquad (1)$$

where ceil() is a function that returns the smallest integer greater or equal than its argument and the "+2" term is for including the cases of logical 0 and 1.
c) gate-iput-2 subpart: a subpart encoding the connection of the second input of the gate, similar to the previous one.
d) gate-output subpart: this subpart encodes whether the gate produces a circuit output or not and needs a number of bits equal to:

$$ceil\ (\ log2\ (\ NoOfOutputs+1\ )\ ) \qquad (2)$$

For example, for a circuit with 3 inputs and 2 outputs (like the full-adder) which is optimised using a 4x4 grid, 13 bits are needed for each gate (3 gate-type bits, 4 gate-input-1 bits, 4 gate-input-2 bits, and 2 gate-output bits) with a total of 117 bits for the whole genotype.

The genotype encoding scheme can be seen in Fig. 2.

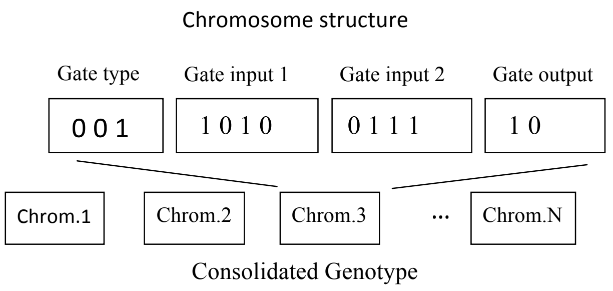

Fig. 2. The binary representation for circuit encoding

2.3. The Evolutionary Algorithm

In this work, a Genetic Algorithm (GA) [13], [14] was used for evolving optimal digital circuits. The GA featured a population of 500 genotypes randomly initialized at the beginning, roulette wheel parent selection [13], uniform crossover [15], binary mutation with a low per-bit probability, replacement of all parents with offspring at each generation, the elitism mechanism [13], and a generation limit of 10,000 generations. Moreover, an automatic scheme for adaptive operator probabilities is used, described in [16].

3. The Digital Circuit Simulator

3.1. The Simulator

In order to evaluate each genetically produced solution, a fitness function must be built to provide a metric for the quality of the solution. In this work, we have used a fast digital circuit simulator, which is described in [11]. Each circuit under evaluation must be encoded in a string form and passed to the simulator as a parameter. Also the complete array of all input vectors to be tested is also passed.

The simulator works as follows: it first analyses and parses the input string describing the circuit and creates an internal array structure with the gates' parameters and circuit topology. Then it presents all input vectors to the circuit, one by one, and calculates the circuit's response for each input. The simulation is performed using a second internal array that keeps the state of binary signals for each gate and interconnection of the circuit.

The digital circuit simulator uses a discrete-time simulation technique inspired by the principles of the propagation of digital signals through the logic gates, and is described in [11]. The propagation of digital signals through gates is simulated step by step, using discrete time quantities that are effectively implemented as loop iterations. These time steps play the role of consequent periods of an informal internal state clock. The period of this ideal clock coincides with the delay of a single gate, whilst the signal propagation through the interconnections is considered to be instantaneous.

When all input vectors are presented to the circuit and all corresponding output vectors have been calculated and registered, the simulator creates a consolidated array of output vectors and returns this array to the Genetic Algorithm.

3.2. The GA Fitness Function

When GA receives the complete output array from the simulator, it compares this array to the array of desired outputs for the specific circuit and calculates the total hamming distance for all bits. The minimization of this hamming distance is the primary objective of the GA. However, another optimization goal exists, and that is to minimize the number of gates of the circuit. This definition makes the problem a multi-objective optimization problem [12]. For handling these two optimization objectives, usually a consolidated objective function is formed as a weighted sum of the individual objective values.

In this work we used the following consolidated function:

$$\begin{array}{ll} \textit{If}\ (F1(S) > 0\,) & Fitness(S) = 100 \times F1(S) \\ \textit{Else} & Fitness(S) = 100 \times F1(S) + 1 \times F2(S) \end{array} \quad (3)$$

where:

$$F1(S) = HammingDistance(\,Os\,,\,D\,) \quad (4)$$

$$F2(S) = NoOfGates(S) \quad (5)$$

where *S* is a solution (circuit) under evaluation, *F1(S)* and *F2(S)* are the two individual optimization functions, *Os* is the output array for solution *S*, and *D* is the array of desired outputs.

This makes the GA to work in two stages: in the first stage the GA tries to find a digital circuit that will satisfy the complete truth table, ignoring the number of gates needed for the circuit, and in the second stage, when it has already satisfied the truth table, it tries to minimize the circuit size. This fitness function has proved to give much better optimization results over all test cases.

4. The Simulation Test Set

For testing the CGP scheme proposed in this work, a set of six (6) elementary and well known digital circuits of increasing complexity and of increasing number of gates has been employed. The test set is shown in Table 1.

Table 1. Parameters of the digital circuits included in the test set

Circuit	No of 2-input gates	No of inputs	No of input combinations	No of outputs	No of output bits
Half Adder	2	2	4	2	8
Decoder 2 to 4	6	2	4	4	16
Full Adder	5	3	8	2	16
2-bit Multiplier	8	4	16	4	64
Decoder 3 to 8	19	3	8	8	64
2-bit Comparator	15	4	16	3	48

In Table 1, the "No of Output Bits" column contains the product of the "No of Input Combinations" and the "No of Outputs", thus expressing the size of the output vector that has to match the desired one from the circuit's truth table.

The typical synthesis of such circuits as well as their complete truth tables is well described in the literature [19]. For example, the picture of a typical "2-bit Comparator" circuit is shown in Fig. 3.

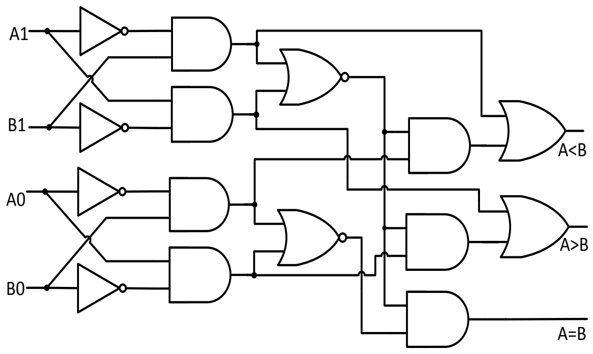

Fig. 3. The 2-bit comparator used in the test set

5. Simulation Results

Simulation results were performed using the same set of GA parameters for all test cases. The complete set of GA parameters is shown in Table 2. For each circuit case, a number of different simulation experiments have been conducted for different grid sizes. For the simpler circuits four different grid sizes have been considered, while for the larger ones three grid sizes have been considered.

Since Evolutionary Algorithms are stochastic algorithms, and in order to avoid statistical errors, ten (10) runs have been made for each circuit case and each grid size. Thus, for each test case several statistical figures have been calculated in order to judge the performance of the proposed implementation.

Table 2. Parameters of the Genetic Algorithm

GA Parameter	Value	GA Parameter	Value
Population	500	Crossover Probability	0.4 to 0.9
Selection	Roulette Wheel	Mutation Probability	0.001 to 0.1
Crossover	Uniform	Elitism	Yes
Mutation	Binary Mutation	Population Replacement	Whole Population
Operator Probabilities	Automatically adapted	Termination	10,000 generations

The simulation results are shown in Table 3. A test case was considered successful if it could find a solution that completely satisfied the circuit's truth table. This was achieved when the hamming distance between the output vector of the genetically produced solution and the desired one was equal to zero (0).

The results were obtained on an Intel Core-i7 workstation with 8GB RAM, running Windows 8.1, and the software was developed using native C++.

As can be seen from Table 3, the CGP scheme manages to find optimal solutions that completely justify the desired truth table, in all test cases.

Table 3. Simulation Results

Circuit	CGP Grid	Success Rate	Avg gates on success	Min gates on success	Max gates on success	Avg gener. to find optimum	Avg exec. time per task (minutes)
Half Adder	2x2	**100%**	2	2	2	< 50	2,0
Half Adder	3x3	**100%**	2	2	2	< 50	2,9
Half Adder	4x4	**100%**	2,3	2	4	275	3,8
Half Adder	5x5	**100%**	2,6	2	3	1150	5,0
Decoder 2 to 4	3x3	**100%**	4,6	4	5	345	3,6
Decoder 2 to 4	4x4	**100%**	4,8	4	6	2850	4,8
Decoder 2 to 4	5x5	**100%**	5,5	4	7	1815	6,0
Decoder 2 to 4	6x6	**100%**	6,1	4	9	3720	8,9
Full Adder	3x3	**80%**	5,4	5	7	1705	4,7
Full Adder	4x4	**90%**	5,6	5	6	880	6,5
Full Adder	5x5	**100%**	7	5	10	1660	8,8
Full Adder	6x6	**100%**	7,3	6	9	2120	17,2
2bit multiplier	4x4	**30%**	8,7	8	9	4750	18,6
2bit multiplier	5x5	**30%**	11	9	13	4950	44,4
2bit multiplier	6x6	**50%**	10,6	8	12	5920	114,6
Decoder 3 to 8	5x5	**20%**	17	17	17	1900	25,5
Decoder 3 to 8	6x6	**10%**	20	20	20	7250	90,8
Decoder 3 to 8	7x7	**40%**	22,5	18	26	6350	155,2
2bit comparator	5x5	**40%**	12	11	13	4875	40,6
2bit comparator	6x6	**60%**	12,2	9	17	7175	113,1
2bit comparator	7x7	**20%**	22	22	22	5925	286,9

For the problems of the Half Adder and the Decoder 2-to-4 the CGP scheme finds the optimum consistently with a 100% success rate, for all grid sizes. However, for the rest of the cases a general decrease of the success rate is observed, as the circuits become more complex and with a larger number of gates.

Also, it is obvious from the results in Table 3, that for each specific test case the grid size on which the circuit is evolved plays a significant role in the overall performance. In the easier cases of the Half Adder and the Decoder 2-to-4, smaller grids lead to smaller search spaces that make the search process easier and faster. But in more complex circuits, the small grids seem to make the search more difficult.

For example, in the case of the Full Adder, using a 4x4 grid seems a better idea than using a 3x3 grid, as the success rate, the solution quality and the speed are better. The 4x4 grid leads to fast discovery of optimal solutions (880 generations needed on average) but there is a slight possibility of missing the optimum (90% success rate). However, when using a 5x5 grid the CGP consistency seems to increase (100% success rate) but with a sacrifice in convergence speed, possibly due to the larger search space.

Similar conclusions can be drawn for the other circuit cases as well. For the 2-bit Multiplier, the 6x6 grid gives better success rate but lower convergence speed. For the Decoder 3-to-8 the best grid choice is 7x7. And for the 2-bit Comparator the 6x6 grid outperforms the other two.

It is also worth mentioning that in most cases the CGP algorithm has produced many unconventional solutions with the same minimal number of gates as the conventional ones, and in some cases even less. The most profound example is that of the 2-bit Comparator. Typical circuit designs as the one shown in Fig. 3 use a total of 15 two-input gates. By searching the web, designs with as few as 11 gates can be found. The CGP scheme has produced a solution that completely satisfies the desired truth table and uses only 9 two-input gates, as can be seen in the case of the 6x6 grid. The schematic diagram of this genetically produced 2-bit comparator circuit is depicted in Fig. 4. Moreover, it can be also noted that in the 5x5 grid case, all solutions use less gates than the 15 of the typical design, as the best one uses only 11 and the worst uses 13.

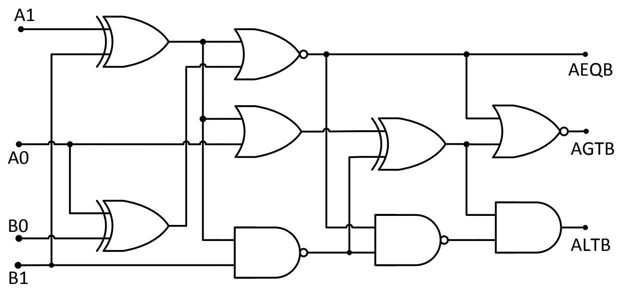

Fig. 4. The 2-bit comparator implementation with 9 gates proposed by the CGP scheme.

A similar case is that of the Decoder 3-to-8 where typical designs use a number of 19 two-input gates, while the CGP scheme has produced solutions with only 17 gates.

These unconventional solutions produced by the CGP scheme with less than typical number of gates, will be further studied in our future work. The ability of the CGP scheme to discover unconventional solutions with the same or even lower number of gates than the typical designs, is justifying its characterization as an "invention machine".

This paper was presented at Pan-Hellenic Conference on Electronics and Telecommunications - PACET, that took place May 8-9 2015, at Ioannina Greece.

Acknowledgement
The authors wish to acknowledge financial support provided by the Research Committee of the Technological Educational Institute of Central Macedonia, Greece, under grant 80/9/20-5-2015.

References

1. T. Back, D. Fogel, and Z. Michalewicz, Handbook of Evolutionary Computation, Oxford Univ. Press, 1997.
2. J. H. Holland, Adaptation in Natural and Artificial Systems, The University of Michigan Press, Ann Arbor, 1975.

3. J.R. Koza, Genetic Programming: On the Programming of Computers by Means of Natural Selection. MIT Press, Cambridge, 1992.
4. W. Banzhaf, P. Nordin, R. E. Keller, and F. D. Francone, Genetic programming: an introduction, (Vol. 1), San Francisco: Morgan Kaufmann, 1998.
5. J. R. Koza, F. H. Bennett III, D. Andre, M. A. Keane, and F. Dunlap, “Automated synthesis of analog electrical circuits by means of genetic programming”, IEEE Transactions on Evolutionary Computation, 1.2, pp. 109-128, 1997.
6. J.F. Miller, D. Job, V.K. Vassilev, “Principles in the Evolutionary Design of Digital Circuits – Part I”, Genetic Programming and Evolvable Machines 1(1), pp. 8-35, 2000, Kluwer Academic Publishers.
7. T. Higuchi, et al. “Evolvable hardware with genetic learning.” IEEE International Symposium on Circuits and Systems, 1996, ISCAS'96. Connecting the World., Vol. 4. IEEE, 1996.
8. T. Higuchi, and X. Yao, Evolvable hardware. Vol. 11. Springer Science & Business Media, 2006.
9. J.F. Miller, and P. Thomson, “Cartesian Genetic Programming,” in: R. Poli, W. Banzhaf, W.B. Langdon, J. Miller, P. Nordin, T.C. Fogarty, (eds.), EuroGP 2000, LNCS, vol. 1802, pp. 121–132. Springer, Heidelberg, 2000.
10. S. Kazarlis and V. Petridis, “Varying Fitness Functions in Genetic Algorithms: Studying the Rate of Increase of the Dynamic Penalty Terms,” Proceedings of the 5th International Conference on Parallel Problem Solving from Nature (PPSN-V), Amsterdam, pp. 211-220, 27-30 September 1998.
11. S. Kazarlis, J. Kalomiros, P. Mastorocostas, V. Petridis, A. Balouktsis, V. Kalaitzis, A. Valais, “A Method for Simulating Digital Circuits for Evolutionary Optimization,” Proceedings of the 10th Annual International Joint Conferences on Computer, Information, and Systems Sciences, and Engineering (CISSE 2014), December 12-14, 2014
12. K. Deb, Multi-objective optimization using evolutionary algorithms. Vol. 16. John Wiley & Sons, 2001.
13. L. Davis, ed. Handbook of genetic algorithms. Vol. 115. New York: Van Nostrand Reinhold, 1991.
14. D. E. Golberg, “Genetic algorithms in search, optimization, and machine learning”, Addion Wesley, 1989.
15. G. Sywerda, “Uniform crossover in genetic algorithms”, Proceedings of the third international conference on Genetic algorithms, J. D. Schaffer, ed., Morgan Kaufmann Publishers Inc., pp. 2-9, 1989.
16. V. Petridis and S. Kazarlis, “Varying Quality Function in Genetic Algorithms and the Cutting Problem,” Proceedings of the First IEEE Conference on Evolutionary Computation, IEEE Service Center, Vol. 1, pp. 166-169, 1994.
17. J. F. Miller, Cartesian genetic programming, Springer Berlin Heidelberg, 2011.
18. J. F. Miller, and S. L. Smith, “Redundancy and computational efficiency in Cartesian genetic programming”, IEEE Transactions on Evolutionary Computation, 10.2, pp. 167-174, 2006.
19. M. Morris Mano, M. D. Ciletti, Digital Design, 5th Edition, Prentice Hall, 2013.