On Heuristics for Seeding the Initial Population of Cartesian Genetic Programming Applied to Combinational Logic Circuits

[Extended Abstract]

Francisco A. L. Manfrini
UFJF, Juiz de Fora, MG, Brazil
IFET, Juiz de Fora, MG, Brazil
francisco.manfrini@ifsudestemg.edu.br

Heder S. Bernardino
UFJF, Juiz de Fora, MG, Brazil
heder@ice.ufjf.br

Helio J. C. Barbosa
LNCC, Petrópolis, RJ, Brazil
UFJF, Juiz de Fora, MG, Brazil
hcbm@lncc.br

ABSTRACT

The design of circuits is an important research field and the corresponding optimization problems are complex and computationally expensive. Here, a Cartesian Genetic Programming (CGP) technique was used to design combinational logic circuits. Several configurations were tested for seeding the initial population. First, the number of rows, columns, and levels-back were varied. In addition, the initial population was generated using only NAND gates. These configurations were compared with results from the literature in four benchmark circuits, where in all instances it was possible to find that some seeding configurations contributed beneficially to the evolutionary process, allowing CGP to find a solution employing a lower number of fitness evaluations. Finally, the variation of the number of nodes of the individuals during the search was also analyzed and the results showed that there is a correlation between the topology of the initial population and the region of the search space which is explored.

CCS Concepts

•**Computing methodologies** → *Search methodologies;*

Keywords

Cartesian Genetic Programming, population seeding, combinational logic circuits

1. INTRODUCTION

The design of combinational logic circuits (CLC) is a discrete optimization problem which requires knowledge and creativity [1]. Particularly, a large set of possibilities is investigated in evolutionary design, as it is not limited by the conventional knowledge of the designer [7].

GECCO'16 Companion July 20-24, 2016, Denver, CO, USA

ACM ISBN 978-1-4503-4323-7/16/07.

DOI: http://dx.doi.org/10.1145/2908961.2909031

CGP [5] is a genetic programming technique in which the programs are modeled as a graph and, thus, a large number of computational structures can be easily represented, such as CLCs [8]. Nowadays, CGP is one of the most efficient methods for evolutionary design and optimization of digital combinational circuits [6].

Here, we propose and test changes in the number of columns and rows, as well as in the parameter *levels-back* and types of gates used in the initial population generation of CGP. The effectiveness of our approach is demonstrated when applied to four benchmark problems.

2. CARTESIAN GP

CGP provides a great generality enabling the representation of neural networks, circuits, and other computational structures [8]. The most common form of CGP uses a $(\mu+\lambda)$ reproduction strategy.

Goldman [3] proposed a method in which a single active gene is modified every time an offspring is generated. The Single's iterative process generates an offspring by mutating randomly selected genes until an active gene is changed. When this mutation is used, one can see that: (i) one active gene is mutated, (ii) inactive genes may be changed, and (iii) no mutation rate needs to be specified by the user. As Single Active Mutation achieved the best results on the hardest test-problem from [3], here we adopted this mutation operator for all the computational experiments.

3. DESCRIPTION OF OUR APPROACH

3.1 Heuristic Population Seeding Procedures

One can see that *levels-back* controls the connectivity and affects the number the inactive nodes. Thus, we propose and compare several scenarios for seeding the initial population. Initially, by varying the three parameters in the representation of CGP: the number of columns, rows, and *levels-back*. It is important to notice that this constraint is applied only during the creation of the initial population.

Then another constraint was included: the initial population was generated using only NAND gates. NAND gates possesses a special property: they are universal, in the sense that they can be used to create the three basic logic expressions (OR, AND, and INVERT). Thus it is possible to implement any logic expression using only NAND gates and no other type of gate. This characteristic provides flexibility

and is very useful in logic circuit design [9]. Since the internal circuitry of the NAND gates are simpler, the resulting circuits usually have better characteristics [2, 9].

Thus, the use of only NAND gates in the initial population will be tested. However, along the evolutionary process NAND gates can be exchanged by any other via mutation.

3.2 Analysis of the Evolution

Comparing the number of evaluations each algorithm requires in order to solve a problem does not provide much understanding concerning their relative performance [4].

The proposed approach is based on the idea of analyzing the behavior of the topology when a beneficial mutation occurs. Every time an improvement in fitness occurs, the number of active nodes of the child is stored.

In all the following examples the median of this vector was chosen as a parameter to analyse the variation of the number of active nodes of the individuals during the search, thus providing information on topology behavior.

4. EXPERIMENTS

For the comparative study reported in this paper, four benchmark examples studied by Goldman [4] and widely used in the electronics literature [9, 2] were chosen to verify the effectiveness of our approach. For each of the examples reported, we used $\mu = 1$ and $\lambda = 4$ [4], and performed 51 independent runs. The best offspring replaces the parent if its fitness is not worse than that of the parent. Ties among offspring are broken by random selection and ties between the offspring and the parent are awarded to the offspring: in this manner inactive genes are allowed to drift, which has been shown [6, 10] to significantly improve performance. The algorithm is terminated when the maximum number of evaluations is exhausted or a feasible solution is obtained. The maximum number of objective function evaluations allowed are 5000, 150000, 50000, and 1100000, respectively, for the examples 1, 2, 3, and 4. For all problems, we used the function set AND, OR, NAND, NOR.

The proposed technique were analysed considering the following four test-problems: 3-Bit Parity (Example 1), 16-4 bit encoder (Example 2), 16-4 bit decoder (Example 3), and 3-bit multiplier (Example 4). This last problem is very difficult by comparison to the other problems.

The following observations summarize our analysis of the obtained results: (i) Varying only the *levels-back* was beneficial to the evolutionary process in the Example 4; (ii) Varying only the topology led to no improvement in all scenarios; (iii) Varying the topology and *levels-back* was beneficial to the evolutionary process in the Examples 3 and 4; (iv) Initializing using only NAND gates was beneficial to the evolutionary process in the Example 4; (v) Initializing using only NAND gates and varying *levels-back* was beneficial to the evolutionary process in the Examples 1, 2, and 4; (vi) Initializing using only NAND gates and varying topology was beneficial to the evolutionary process in the Examples 3 and 4; (vii) Initializing using only NAND gates, and varying both topology and *levels-back*, was beneficial to the evolutionary process in the examples 1, 2, and 4; (viii) When *levels-back* was at 25% the population was able to decrease the amount of active nodes along the search; In most cases, this behavior was beneficial to the evolutionary process. (ix) The population that started with a high degree of connectivity was able to decrease the number of nodes and thus explore regions of the search space with different degrees of connectivity; (x) When *levels-back* was set to 100% or 50%, the number of active nodes in the solution is roughly the same as in the initial population; and (xi) The *levels-back* parameter determines not only the level of connectivity of the initial population, but also defines the behavior of the topology throughout evolution.

5. CONCLUSIONS

Experimental results confirmed the superiority of the proposed heuristics over random seeding in reducing the number of fitness evaluations to reach a feasible solution. Differently from what has been pointed out in the literature, we have shown that the parameter *levels-back* used to generate the initial population should be constrained, instead of allowing connections to be formed between any pair of nodes. Additionally, using only NAND gates in the initialization procedure led to increased performance in the majority of the cases studied.

Acknowledgments

The authors would like to thank the support provided by CNPq (grant 310778/2013-1), FAPEMIG, and PGMC/UFJF.

6. REFERENCES

[1] C. A. C. Coello, A. D. Christiansen, and A. H. Aguirre. Use of evolutionary techniques to automate the design of combinational circuits. *Intl. Journal of Smart Engineering System Design*, 2:299–314, 2000.

[2] M. D. Ercegovac, J. H. Moreno, and T. Lang. *Introduction to digital systems*. John Wiley & Sons, Inc., 1998.

[3] B. W. Goldman and W. F. Punch. *Reducing wasted evaluations in cartesian genetic programming*. Springer, 2013.

[4] B. W. Goldman and W. F. Punch. Analysis of cartesian genetic programming's evolutionary mechanisms. *IEEE Transactions on Evolutionary Computation*, 19(3):359–373, 2015.

[5] J. F. Miller. An empirical study of the efficiency of learning boolean functions using a cartesian genetic programming approach. In *Proc. of the Genetic and Evolutionary Computation Conference*, volume 2, pages 1135–1142, 1999.

[6] J. F. Miller. *Cartesian genetic programming*. Springer, 2011.

[7] J. F. Miller, D. Job, and V. K. Vassilev. Principles in the evolutionary design of digital circuits Part I. *Genetic Programming and Evolvable Machines*, 1(1-2):7–35, 2000.

[8] J. F. Miller and S. L. Smith. Redundancy and computational efficiency in cartesian genetic programming. *IEEE Transactions on Evolutionary Computation*, 10(2):167–174, 2006.

[9] R. J. Tocci, N. S. Widmer, and G. L. Moss. *Digital systems*. Pearson, 2011.

[10] A. J. Turner and J. F. Miller. Neutral genetic drift: an investigation using cartesian genetic programming. *Genetic Programming and Evolvable Machines*, pages 1–28, 2015.