# Exploring Curriculum Learning on a CGP Logic Optimization Flow

Naiara Sachetti, Bryan M. Lima, Augusto Berndt, Cristina Meinhardt and Jonata Tyska Carvalho
*Department of Informatics and Statistics, Federal University of Santa Catarina - UFSC*
Florianópolis, Brazil
naiara.sachetti@grad.ufsc.br, bryan.l@grad.ufsc.br, augusto.berndt@posgrad.ufsc.br,
cristina.meinhardt@ufsc.br, jonata.tyska@ufsc.br

***Abstract*—This work proposes an improvement to a Cartesian Genetic Programming implementation for learning and optimizing logic circuits during logic synthesis. This step is an essential part of the design flow for manufacturing circuits. In this context, we work with evolutionary computing as an alternative to the growing complexity of logic circuits. Our proposed improvement to Cartesian Genetic Programming consists of a technique known as Curriculum Learning, which classifies the input given during learning, indicating if each input is easy or hard to learn. Our results demonstrate that Curriculum Learning may provide a better convergence during learning by distributing the difficulty of inputs or prioritizing the hard inputs, since some experimented exemplars achieve higher accuracy with the implemented technique.**

***Index Terms*—Logic Optimization, Cartesian Genetic Programming, Curriculum Learning**

## I. Introduction

The continuous technology evolution allows the implementation of more complex logic functions directly in integrated circuits. Nowadays, there is a crescent demand for hardware implementations of complex functions that requires new approaches for logic optimization tools, considering the large number of inputs on these problems, for example, neural networks. A derived approach is to explore Logic Learning strategies instead of considering the complete logic description of these functions. In this case, the Logic Synthesis problem can be implemented with incompletely specified functions attempting to generalize a reduced set of logic variables.

The logic learning problem has been investigated in the previous two years on the IWLS contest [1], exploring the generalization capability of Machine Learning methods to optimize incomplete logic functions. The generalization capability of a logic learning method is measured by its accuracy, calculated with a portion of the truth table not used for training the models. Moreover, logic learning is also appropriated to synthesize approximate circuits, since most of the approximate strategies works with an incomplete Truth Table for the original problem.

Most of the recent works about logic learning implement the representation of a logic function with Boolean networks, such as an AIG (And-Inverter-Graph). The AIG is the state-of-the-art data structure for technology-independent optimizations during logic synthesis [2]. Also, the approaches usually solve the logic learning problem by mapping a Machine Learning model to the Boolean network. The Machine Learning techniques explored by previous work include Decision Trees [3] [4], [5], Random Forests [5], Look-Up-Table (LUT) network [6], [7], Neural Networks (NN) [7] and Cartesian Genetic Programming (CGP) [8], [9].

The CGP technique is an evolutionary computing approach introduced by [10] and described as so in [11]. Originally, it uses directed acyclic graphs represented as a two-dimensional grid of computational nodes to represent programs [12] (or, in logic learning, circuits). Since its definition, other works have also explored the implementation of Genetic Programming as a logic learning technique, with an active research community. Previous work on CGP usually attempt to improve its main drawback: runtime [13]. In [8], [9], a CGP-based flow is proposed seeking to improve the accuracy and size of approximate circuits representing a logic function. The logic minimization by CGP shows to be effective when considering simultaneously accuracy and the number of nodes as objective functions.

Despite the results presented on [8], [9] are promising for many logic functions, the CGP-based solution does not provide good results for many of the logic functions presented. Therefore, the method would benefit from new mechanisms and adjustments that could increase its evolvability, which means the capacity of the evolutionary process to keep finding better solutions. In this sense, a possible direction is trying to seek among the techniques capable of improving the accuracy of Machine Learning-based solutions, and investigate how they can be explored in the proposed CGP logic optimization flow to improve accuracy or reduce the time necessary to learn a function. Following that direction, we decided to investigate a technique called Curriculum Learning (CL) [14].

As first formalized by [15], a curriculum can be seen, at an abstract level, as a sequence of training criteria, with each training criterion being associated with a different set of weights on the training examples. The authors of [16] bring this concept to the context of evolutionary training of embodied agents, using CL to manipulate the environmental conditions in which the evolving agents are evaluated. The goal is to select environmental conditions that challenge the weakness of the current evolving agents, maintaining the proper level of difficulty.

In this work, we apply the Curriculum Learning approach,

similar to the one in [16], to the CGP Logic Optimization flow described in [8] and [9]. CL is used to select examples in the training set accordingly to the difficulty of each example in the training set and a difficulty function. The difficulty of each example is defined as the number of times each of the candidate solutions manage to produce the correct output during the evolutionary process. Furthermore, we present results of initial experiments that have as main goal to search for evidences that the CL might be an effective strategy to improve accuracy of evolving circuits.

## II. Base CGP-based flow

Our implementation of Curriculum Learning uses as foundation the CGP-based optimization flow proposed in [9], described in the Figure 1. Such flow is designed to perform optimization of Boolean functions described as complete or incomplete truth tables, being capable of accomplish that on random initialized circuits (pure CGP flow) as well as on previously optimized circuits (fine-tuning flow).

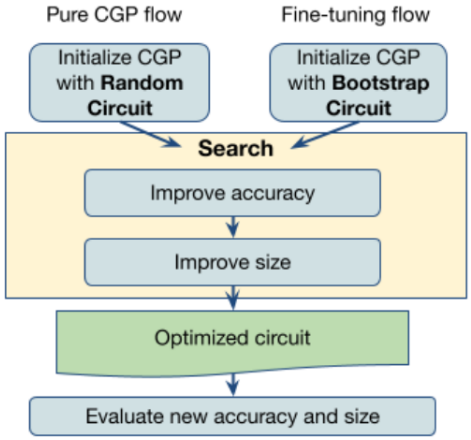



Fig. 1: CGP-based optimization flow used as foundation. Source: [9].

After the establishment of an initial population of circuits, each of them is evaluated and the ones with the best accuracy and larger functional circuits (i.e. circuits with larger quantities of functional nodes) are selected; This is known to lead to the improvement of accuracy and faster convergence [14]. In a second phase, smaller functional circuits are selected, aiming to the shrinkage of circuit area. Finally, the new and optimized circuit is generated and its accuracy and size are evaluated [9].

The node functions used in the flow are AND, Inverters and XORs, this means that AIG (AND-Inverter Graph) and XAIG (XOR-AIG) structures are used to represent circuits [9].

Also, it is possible to control the maximum number of generations the CGP-based flow can use to evolve circuits. In each generation the individuals are evaluated with mini-batches, composed by a portion of combinations (i.e. lines in the truth table) randomly selected from the truth table used as training set. The number of combinations in each mini-batch is given by the *batch size* hyperparameter. Mini-batches stay unchanged by a certain number of generations, controlled by the *change each* hyperparameter [9].

Keeping in mind that the main goal of our initial experiments is to find evidences that the Curriculum Learning might be an effective strategy to improve accuracy of evolving circuits, some features of the CGP-based flow were established to compose the comparison (and baseline) version to our implementation of CL, named here CGP-noCL. They are:

- The circuits are represented only as AIGs.
- It uses only random generated initial populations.
- The "Improve size" step was skipped.

In the following section, we describe the features of our implementation of Curriculum Learning, made on top of CGP-noCL.

## III. Methodology

Our implementation of CL substitutes the random selection of combinations for the mini-batches by a selection based on the difficulty of each combination in the training set and in some *difficulty function*.

The difficulty of a combination is given by Equation 1, where $H_i$ is the number of times a truth table line $i$ output was correctly predicted by a circuit being evaluated and $E_i$ is the number of times that line $i$ participated of evaluations of circuits.

$$D_i = \frac{H_i}{E_i} \tag{1}$$

In other words, the more the output of a line is correctly predicted by the evaluated circuits (i.e. the more $D_i$ is close to 1), the easier this combination is considered.

Initially, the difficulty of each line in the training set is defined by a proceeding that evaluated each line by a number of random generated circuits. These circuits are part of the population to be evolved.

Similarly to what is presented by [16], the mini-batch composition of lines, i.e. the curriculum, is selected with respect to some difficulty interval. Such interval is defined by the difficulty function.

Equation 2 determines each interval superior limit of difficulty, where $b$ is the mini-batch position index and $m$ is the value of *batch size*.

$$L_b = \frac{f(b)}{f(m)} \tag{2}$$

So, ideally, each position in the batch will receive a line classified in the interval for its own index. If no combination is classified in a given interval, one from the nearest non-empty neighbor interval (superior or inferior) is selected.

Figure 2 presents the range limits to a linear (in blue) and a quadratic (in red) difficulty function for a *batch size* of 64. Note that while the former is capable of creating balanced mini-batches, the latter can form harder mini-batches.

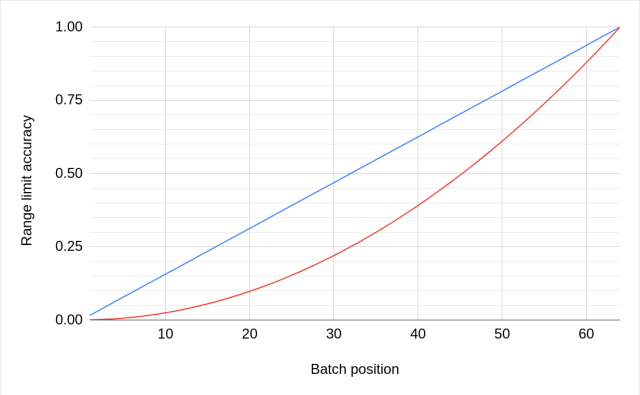



Fig. 2: CL interval limit accuracy.

## IV. Evaluation and Results

The evaluation of our implementation was made with a portion of the benchmarks of IWLS2020 contest (available at [17]). Specifically, we used the functions with a maximum of 19 inputs, in order to make our analysis feasible with respect to the computational resources at hand. The benchmarks used are presented in Table I.

TABLE I: Benchmark information. Adapted from: [1] and [5].

| Func. | Logic type | # Inputs | | Logic |
|---|---|---|---|---|
| | | Min | Max | Domain |
| 20,21 | Multipliers | 16 | 16 | Arithmetic |
| 30 | Comparators | 20 | 20 | Arithmetic |
| 40,41,43 | Square-root | 10 | 18 | Arithmetic |
| 50 | PicoJava design | 19 | 19 | Random |
| 65,69,73,74 | MCNC benchmarks | 16 | 19 | Random |
| 75-79 | Symmetric functions | 16 | 16 | Random |

The experiments used the set of hyperparameters listed in Table II. *Initial population* defines the number of individuals used to establish the initial difficulty of the combinations in the training set. The rest of hyperparameters presented define the number of nodes in the circuits being selected, the maximum number of generations, the size of the mini-batches and the frequency with which the mini-batches will be changed, respectively.

In this work, we used a set of default parameters for all experiments, without exploring the hyperparameters, to define a similar environment on all the evaluation and exclusively observe the influence of the curriculum learning algorithm on the accuracy of the CGP Logic Optimization flow. Thus. we are interested on the difference between the investigated methods, keeping the investigation on hyperparameter optimization as an intended future work.

TABLE II: Hyperparameters used

| Parameter | Value |
|---|---|
| Initial population | 5 |
| Number of nodes | 250 |
| Generations | 50000 |
| Batch size | 64 |
| Change each | 250 |

The results of the experiments are summarized in Figures 3, 4b and 4a. For the box plots in Figure 3, the accuracies obtained in each run for each benchmark were summarized by making the mean of them. This was made separately for the three situations: CGP-noCL and CGP with linear and quadratic difficulty functions. The box plots present the absolute differences between the means of the versions with CL and the one without CL.

Observing the chart, it is possible to visualize that for the major part of the benchmarks runned with CL, gains in average accuracy were observed.

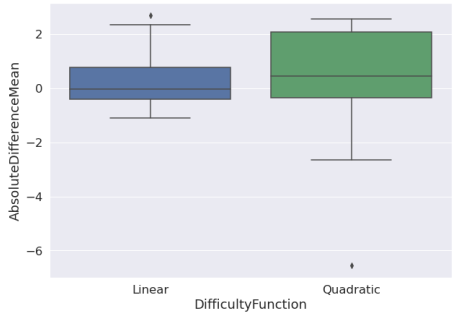



Fig. 3: Absolute differences with respect to mean of accuracies in CGP-noCL.

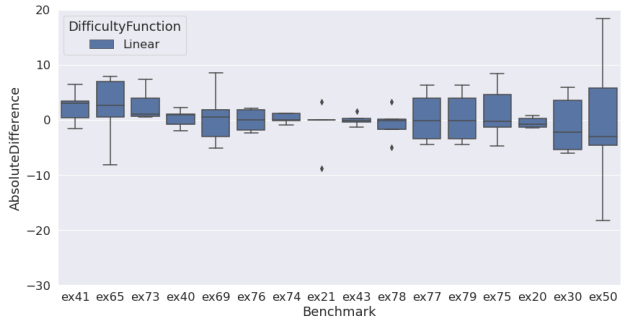
To observe the results with more details, the Figure 4a presents box plots for the absolute differences of each benchmark with respect to accuracies of CGP-noCL for the linear difficulty function. Note that there are benchmarks where the major part of the circuits evolved with CL had an overall superior accuracy than the ones evolved without CL. Furthermore, ex73 only presented gains in accuracy and for ex50 gains of nearly 20% were registered. On the other hand, also note that at ex50 losses of nearly 20% were registered.

Figure 4b presents the box plots for the absolute differences of each logic function with respect to accuracies of CGP-noCL for the quadratic difficulty function. We can note a slight improvement on the accuracy when it is adopted the quadratic function in the Curriculum Learning algorithm, compared to the CGP-noCL version. A particular behavior is observed to the ex75, that only reports gains in accuracy, and the extreme of gains and losses were of nearly 15% and 25%, respectively.

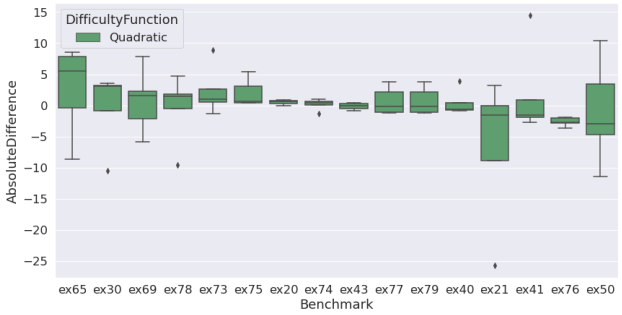
Additionally, Table III presents the maximum and minimum number of coding nodes (i.e., in CGP, the number of nodes that contributes to the calculation of the output data) in each version of the flow. The circuits generated with the versions of the flow with Curriculum Learning presented a small increase in size.

TABLE III: Circuit sizes in number of functional nodes

| Flow version | Coding nodes | |
|---|---|---|
| | Min | Max |
| CGP-noCL | 0 | 102 |
| CGP-CLv1 linear | 1 | 107 |
| CGP-CLv1 quadratic | 5 | 108 |




(a) For a linear difficulty function.




(b) For a quadratic difficulty function.

Fig. 4: Absolute differences with respect to accuracies in CGP-noCL.

The preliminary charts presented make clear that Curriculum Learning might be an effective technique to improve accuracy in the CGP-based optimization flow. We intend to investigate further to understand what are the main features that distinguish the benchmarks that benefit from this technique and the ones that do not, as well as to comprehend the hyperparameters that have direct impact on results.

## V. Conclusion

Through this work, we proposed a technique to be combined with a CGP-based flow for logic optimization, the Curriculum Learning, and evaluated the possibility of it to be effective in improving accuracy of evolved circuits by distributing the difficulty of inputs or prioritizing the hard ones.

The results of our evaluation show that there are several situations in which gains in accuracy were observed, with a small increase in size, meaning that further investigation is valuable and needed to understand better what influences the gains and losses observed. Furthermore, the quadratic approach seems to provide slightly higher accuracy results.

In future works, we intend to evaluate the hyperparameters that affect directly the accuracy of circuits evolved and optimize them. Also, we intend to execute another set of experiments using other difficulty functions (e.g. root function).

## Acknowledgment

This work was financed in part by National Council for Scientific and Technological Development – CNPq and the Propesq/UFSC.

## References

[1] Shubham Rai and et al. Logic synthesis meets machine learning: Trading exactness for generalization. In *2021 Design, Automation & Test in Europe Conference & Exhibition (DATE)*. IEEE, 2021.

[2] H. Riener, W. Haaswijk, A. Mishchenko, G. De Micheli, and M. Soeken. On-the-fly and dag-aware: Rewriting boolean networks with exact synthesis. In *2019 Design, Automation Test in Europe Conference Exhibition (DATE)*, pages 1649–1654, 2019.

[3] Valerio Tenace and Andrea Calimera. Inferential Logic: a Machine Learning Inspired Paradigm for Combinational Circuits. In *2018 IFIP/IEEE International Conference on Very Large Scale Integration (VLSI-SoC)*, pages 149–154, 2018.

[4] Brunno A. de Abreu, Augusto Berndt, Isac S. Campos, Cristina Meinhardt, Jonata T. Carvalho, Mateus Grellert, and Sergio Bampi. Fast logic optimization using decision trees. In *2021 IEEE International Symposium on Circuits and Systems (ISCAS)*, pages 1–5, 2021.

[5] Wei Zeng, Azadeh Davoodi, and Rasit Onur Topaloglu. Lorax: Machine learning-based oracle reconstruction with minimal i/o patterns. In *2021 IEEE Computer Society Annual Symposium on VLSI (ISVLSI)*, pages 126–131, 2021.

[6] Satrajit Chatterjee. Learning and memorization. In Jennifer Dy and Andreas Krause, editors, *Proceedings of the 35th International Conference on Machine Learning*, volume 80 of *Proceedings of Machine Learning Research*, pages 755–763. PMLR, 10–15 Jul 2018.

[7] Yukio Miyasaka, Xinpei Zhang, Mingfei Yu, Qingyang Yi, and Masahiro Fujita. Logic Synthesis for Generalization and Learning Addition. In *2021 Design, Automation Test in Europe Conference Exhibition (DATE)*, pages 1032–1037, 2021.

[8] Augusto Berndt, Brunno A. de Abreu, Isac S. Campos, Bryan Lima, Mateus Grellert, Jonata T. Carvalho, and Cristina Meinhardt. Accuracy and size trade-off of a cartesian genetic programming flow for logic optimization. In *Proceedings of the 34th Symposium on Integrated Circuits and Systems Design*, SBCCI '21, 2021.

[9] Augusto Berndt, Brunno A. de Abreu, Isac S. Campos, Bryan Lima, Mateus Grellert, Jonata T. Carvalho, and Cristina Meinhardt. A cgp-based logic flow: Optimizing accuracy and size. *Journal of Integrated Circuits and Systems (JICS)*, 2022.

[10] Julian Miller, P. Thomson, T. Fogarty, and I Ntroduction. Designing electronic circuits using evolutionary algorithms. arithmetic circuits: A case study. *Genetic Algorithms and Evolution Strategies in Engineering and Computer Science*, 10 1999.

[11] Julian F. Miller. An empirical study of the efficiency of learning boolean functions using a cartesian genetic programming approach. In *Proceedings of the 1st Annual Conference on Genetic and Evolutionary Computation - Volume 2*, GECCO'99, page 1135–1142, San Francisco, CA, USA, 1999. Morgan Kaufmann Publishers Inc.

[12] Julian F. Miller. *Cartesian Genetic Programming*, pages 17–34. Springer Berlin Heidelberg, Berlin, Heidelberg, 2011.

[13] Julian Francis Miller. Cartesian genetic programming: its status and future. *Genetic Programming and Evolvable Machines*, pages 1–40, 2019.

[14] Nicola Milano, Paolo Pagliuca, and Stefano Nolfi. Robustness, evolvability and phenotypic complexity: insights from evolving digital circuits. *Evolutionary Intelligence*, 12(1):83–95, 2019.

[15] Yoshua Bengio, Jérôme Louradour, Ronan Collobert, and Jason Weston. Curriculum learning. In *Proceedings of the 26th Annual International Conference on Machine Learning*, ICML '09, page 41–48, New York, NY, USA, 2009. Association for Computing Machinery.

[16] Nicola Milano and Stefano Nolfi. Automated curriculum learning for embodied agents a neuroevolutionary approach. *Scientific Reports*, 11(8985):1–14, April 2021.

[17] IWLS 2020 programming contest. Iwls 2020 benchmarks. https://github.com/iwls2020-lsml-contest/iwls2020-lsml-contest, 2020.