Optimizing CGP Hyperparameters for Efficient Search of Degenerate Multiplexer Designs

Yasmin Soares
Centro de Informática
Universidade Federal de Pernambuco
Recife, Brazil
ORCID 0009-0006-9814-0117

Humberto Tavora
Centro de Informática
Universidade Federal de Pernambuco
Recife, Brazil
ORCID 0009-0001-0713-6408

Stefan Blawid
Centro de Informática
Universidade Federal de Pernambuco
Recife, Brazil
ORCID 0000-0001-6982-4575

***Abstract*—With advancements in nanotechnology, digital systems are becoming increasingly complex and miniaturized. This miniaturization increases susceptibility to process defects, necessitating innovative approaches to ensure the reliability and robustness of digital circuits. Cartesian Genetic Programming (CGP) has emerged as a promising method for evolving defect-tolerant digital systems. CGP enables the search for degenerate digital circuit designs that can be used in redundancy strategies to mitigate accuracy reduction caused by process defects. This research aims to identify the optimal hyperparameters for the CGP algorithm. By systematically examining these parameters, we seek to understand their effects on the performance and efficiency of the evolutionary search for degenerate digital designs. We illustrate our approach by analyzing the defect-robustness of evolved multiplexer circuits, contributing to the field of defect-tolerant circuit design.**

***Index Terms*—Cartesian Genetic Programming, Multiplexers, Defect tolerance, Nanotechnology, Digital Systems**

I. Introduction

As nanotechnology progresses, precision in design and fabrication becomes paramount and must be reconsidered [1]. Electronic Design Automation (EDA) appears to be an essential ally in this endeavor. The convergence of nanotechnology and EDA is critical for the continued innovation and development of next-generation electronic systems [2]. Genetic programming, specifically Cartesian Genetic Programming (CGP), is a possible tool and well-suited for synthesizing nanoscale digital systems. Natural evolution-based strategies for handling defects reemerge in the evolution of digital circuits, especially when optimization toward minimal genotypes is avoided. This can be achieved by evolving circuits in the presence of defects, enhancing their robustness [3], [4]. CGP will not replace established and optimized EDA tools. However, CGP helps to explore neglected parts of the design space that may offer interesting solutions to the challenge of an increasing number of process defects.

This study focuses on the multiplexer, a versatile circuit widely used in electronics. Multiplexers are crucial in communication networks, where they combine data streams from multiple sources and transmit them over a single data channel [5]. Moreover, multiplexers can be used as logic function generators employed, e.g., in the programmable logic blocks of FPGAs. The flexibility and broad applicability of multiplexers make them an ideal subject for exploring the robustness of evolved circuits using CGP. Through a series of experiments with these circuits, we will collect and analyze hyperparameters, evaluating their impact on the search performance using specific indicators detailed throughout this study.

Our goal is to identify the optimal combination of hyperparameters for our evolutionary algorithm and provide insights for future research. The CGP algorithm searches for degenerate digital designs, i.e., circuits that implement the same logic function with a different schematic. Additionally, we will assess the circuits' resilience to defects, contributing to the understanding of defect-tolerant design in nanoscale digital systems. This comprehensive analysis aims to advance the field of evolutionary algorithms and enhance the robustness of digital circuits.

II. Multiplexers

In modern electronic systems, efficient data management is critical, and multiplexers (MUX) play a pivotal role in achieving this efficiency. For instance, in communication networks, multiplexers allow multiple data streams to be combined and transmitted over a single channel, optimizing bandwidth usage, increasing the amount of data that can be sent over the network within a certain amount of time [5]. A multiplexer is also called a data selector [6]. By using 2^N input lines controlled by N select lines, multiplexers enable the selection and routing of specific input signals to a single output line. This not only simplifies circuit design but also reduces complexity and wiring, enhancing overall system performance. A 4:1 multiplexer is depicted in Fig. 1, with its four inputs and one output controlled by two select line, exemplifying the MUX functionality [7].

III. Genetic Programming

High prototyping costs make modeling, simulation, and optimization techniques essential. Most microelectronics processes are standardized and can be simulated using existing EDA tools for process, device, and system simulation [8].

This work was supported by the Conselho Nacional de Desenvolvimento Científico e Tecnológico (CNPq) through grants from the Programa Institucional de Bolsas de Iniciação Científica (PIBIC).

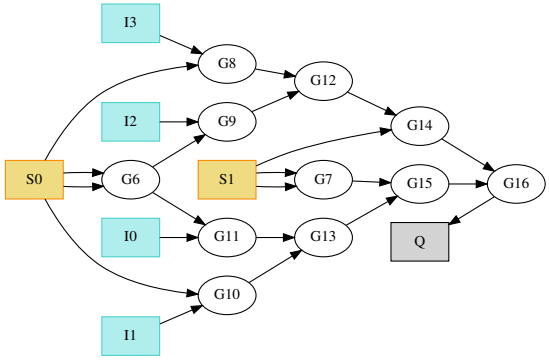

Fig. 1: Ideal 4x1 MUX circuit using only NANDs gates implemented as a direct acyclic graph structure.

Using a computer to model circuits is particularly advantageous as circuit complexity increases, allowing for efficient exploration of a vast design and optimization space. Moreover, the computer's ability to generate non-obvious connections, which might initially seem unpromising to a human, can lead to innovative and highly effective solutions. In our study, we tackle the modeling challenge using a form of Genetic Programming (GP), an intriguing technique that has proven effective in similar works [9], [10], [11].

Employing the concept of "survival of the fittest", Genetic Programming (GP) is an evolutionary computation technique in which computer programs evolve randomly, with mutations and selections occurring probabilistically, instead of following a fixed deterministic pattern. The fitness of a program is evaluated using a fitness function tailored to the specific problem [12]. In the case of MUXs, our fitness function was defined as the ratio of the number of correct outputs by the total possible outputs. For example, in a circuit with 2 inputs (4 possible outputs) and 3 correct outputs, the fitness would be 75%.

It is believed that the repeated process of mutation and selection leads to individuals increasingly adapted to their environment, i.e., it is assumed that they have increasing fitness [13]. The most critical and time-consuming part of the CGP cycle is fitness evaluation, which mainly limits the scalability of search-based design [14].

Our implementation of the CGP algorithm can be found in the GitHub Logic-Circuits-Evolution repository. Details will be reported elsewhere. Following a series of executions, a total of 9 distinct MUX designs were organized into a library (each MUX was randomly assigned a number from 1 to 9 as its identifier). A preview of this library is depicted in Fig. 2.

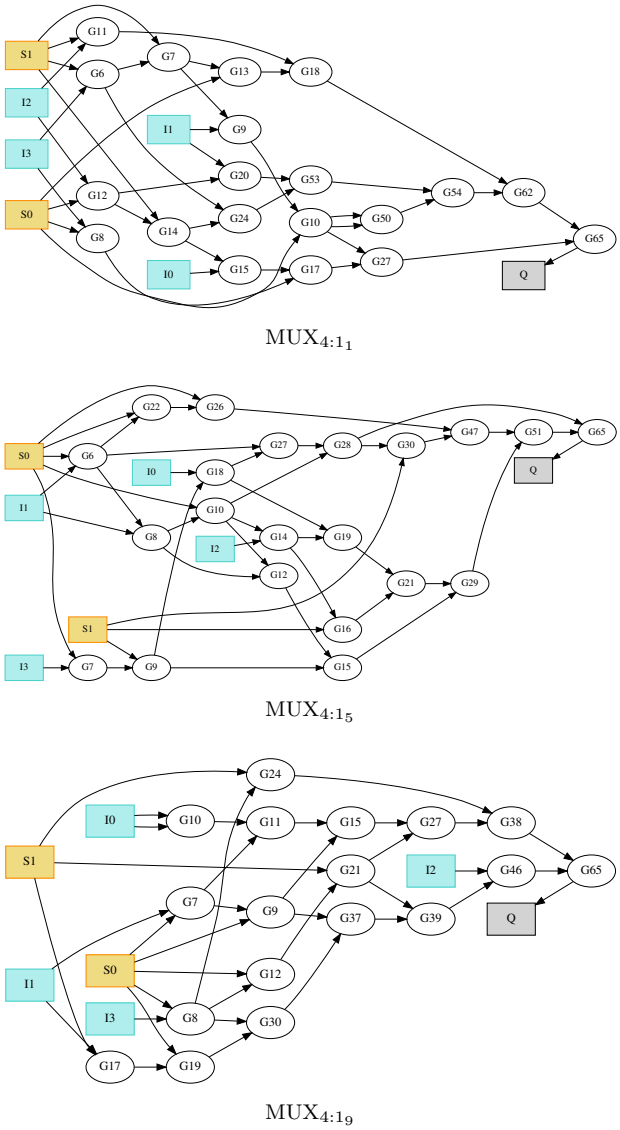

Fig. 2: Representation of MUX 4:1 designs of varying sizes constructed using only NAND gates according to the proposed method. Direct acyclic graphs where blue boxes denote inputs, 4 MUX inputs (I0, I1, I2, I3) in blue and 2 MUX selectors (S0, S1) in yellow, NAND gates are formed by ellipses, and 1 output depicted as a grey box. The full library is available at our GitHub repository, see the Sec. "Online resources".

IV. Optimization Strategy

The study of hyperparameters is a promising area and is generally associated with Machine Learning and Deep Learning. CGP, as a subset of Machine Learning, however, does not have as many discussions dedicated to the improvement of its multiple hyperparameters.

Given that, the experiments were conducted with a focus on extracting useful insights, aimed at optimizing CGP for the MUX's design automation. The goal was to enhance our understanding and performance of CGP through systematic hyperparameter tuning, different combinations were explored and investigated through grid-search. We analysed a total of six hyperparameters. Among these parameters are genotype size, mutation rate, noise variation applied to each fitness evaluation, and the number of offspring per generation, denoted as Lambda (λ). Each of these hyperparameters impacts the evolution process, affecting either the time required or the number of generations needed to achieve 100% fitness.

Increasing genotype size extends computation time for each generation due to more NAND operations during fitness calculation. Although genotype size is a genome parameter,

TABLE I: Hyperparameters optimization (For the platform details, please, check the online resources). The following hyperparameters were considered in the experiments: Genome length (G.Length), mutation rate (M.R), noise range (N.R) and number of offspring (λ). The optimization criteria included: Average execution time (AET), total time taken for the execution of 'N' different genotype lengths (TTT) and average generation count (AGC).

(a) G.Length = 40

M.R	N.R	λ	AET	TTT	AGC
0.05	0.003	6	30.94	0:05:09	4531.0
0.05	0.003	8	19.42	0:03:14	2275.5
0.05	0.003	10	18.14	0:03:01	1641.7
0.05	0	6	20.29	0:03:22	2778.5
0.05	0	8	29.87	0:04:58	3171.1
0.05	0	10	10.49	0:01:44	910.1
0.10	0.003	6	29.92	0:04:59	4053.8
0.10	0.003	8	27.96	0:04:39	2980.7
0.10	0.003	10	23.63	0:03:56	2069.3
0.10	0	6	21.55	0:03:35	3024.2
0.10	0	8	28.58	0:04:45	2937.8
0.10	0	10	18.67	0:03:06	1708.8
0.15	0.003	6	23.33	0:03:53	3300.0
0.15	0.003	8	25.92	0:04:19	3045.7
0.15	0.003	10	21.80	0:03:38	2179.5
0.15	0	6	23.31	0:03:53	3367.7
0.15	0	8	17.65	0:02:56	1914.6
0.15	0	10	33.10	0:05:30	3112.2

(b) G.Length = 50

M.R	N.R	λ	AET	TTT	AGC
0.05	0.003	6	10.78	0:01:47	1611.0
0.05	0.003	8	20.35	0:03:23	2324.6
0.05	0.003	10	22.04	0:03:40	2151.5
0.05	0	6	18.05	0:03:00	2573.7
0.05	0	8	9.84	0:01:38	1070.5
0.05	0	10	11.30	0:01:53	1010.3
0.10	0.003	6	14.53	0:02:25	1991.2
0.10	0.003	8	26.20	0:04:22	2855.5
0.10	0.003	10	11.30	0:01:53	1016.2
0.10	0	6	24.29	0:04:02	3338.8
0.10	0	8	29.59	0:04:55	3095.8
0.10	0	10	15.94	0:02:39	1518.4
0.15	0.003	6	12.49	0:02:04	1729.5
0.15	0.003	8	14.49	0:02:24	1628.4
0.15	0.003	10	14.12	0:02:21	1263.0
0.15	0	6	12.49	0:02:04	1813.2
0.15	0	8	21.26	0:03:32	2360.8
0.15	0	10	13.29	0:02:12	1178.4

(c) **G.Length = 60**

M.R	N.R	λ	AET	TTT	AGC
0.05	0.003	6	21.56	0:03:35	2772.7
0.05	0.003	8	11.77	0:01:57	1295.4
0.05	0.003	10	10.52	0:01:45	868.5
0.05	0	6	14.86	0:02:28	1914.2
0.05	0	8	11.14	0:01:51	1167.2
0.05	0	10	17.18	0:02:51	1452.3
0.10	**0.003**	**6**	**6.61**	**0:01:06**	**853.0**
0.10	0.003	8	17.18	0:02:51	1660.0
0.10	0.003	10	19.49	0:03:14	1635.2
0.10	0	6	7.47	0:01:14	985.4
0.10	0	8	9.39	0:01:33	987.3
0.10	**0**	**10**	**7.40**	**0:01:13**	**628.2**
0.15	0.003	6	7.71	0:01:17	1102.1
0.15	0.003	8	13.10	0:02:10	1332.8
0.15	0.003	10	21.79	0:03:37	1788.2
0.15	0	6	12.57	0:02:05	1637.9
0.15	0	8	24.12	0:04:01	2493.3
0.15	0	10	19.53	0:03:15	1625.8

it is included due to its relationship with mutation rate. For instance, a genotype with 40 genes and a 10% mutation rate mutates 4 genes. Raising the mutation rate to 12% has minimal impact since only integer parts are considered. Thus, the employed hyperparameter grid was chosen to explore diverse scenarios. Another critical aspect is noise variation. Each genome has two fitness values: one real and one with noise. A noise rate (N.R) of 0.003 means the noisy fitness fluctuates by ±0.003 from the real fitness, aiding in exploring different forms and reducing local maxima. Lastly, the number of offspring per generation (λ) is relevant. More offspring per generation increase the computational cost due to the larger number of genomes.

V. Results

The results are presented in Table I. We used a manual implementation of grid search, conducting 10 evolutions for each hyperparameter combination. It was ensured that the discovered MUX circuits had different sizes to avoid combinations that predominantly resulted in minimal solutions. In other words, 10 MUX circuits with different sizes and 100% fitness were found by the evolutionary search. Each combination is evaluated based on time metrics, where we can observe the time required to obtain each of the 10 different sizes of MUXs (TTT) and the average time each evolution took (AET). Additionally, we can also observe the average number of generations each evolution required to reach maximum fitness (AGC).

The Table I, reveals a general trend of performance improvement as genotype length (G.Length) increases. This aligns with our expectations, as more genes enhance the probability space for finding a combination that achieves 100% fitness. The impact of increasing the mutation rate (M.R) is more pronounced with 60 genes. Thus, if G.Length increases, the mutation rate should be reevaluated. The optimal configuration was achieved with a 10% mutation rate, mutating 6 genes per offspring. The majority of the best combinations have noise range (N.R) of 0, indicating no noise during evolution. While this result is expected, it is not universally optimal. In more complex circuits, noise can help the evolutionary process by temporarily reducing fitness to overcome local maxima. Lastly, the number of offspring per generation also showed consistent results, where increasing the number of offspring generally enhances performance indicators.

The combination that produced the best results included the largest genotype size in the search grid (60 genes), a 10% mutation rate. About the noise, and a λ, there is 2 good combinations that are related then selves. The combination without noise and with 10 offspring per generation results in a lower average number of generations required to reach the 10 different circuits. In contrast, the combination with noise and only 6 offspring per generation resulted in a higher average number of generations needed, but found the 10 different circuits in less time. This observation is logical, as a larger number of offspring leads to more fitness calculations being performed in each generation, thereby proportionally increasing the time required. Furthermore, this disparity in the number of generations is expected because a higher number of offspring facilitates a more comprehensive search within the probability space. Consequently, the number of generations is reduced at the expense of increased execution time due to the additional fitness evaluations needed for each generation. Analyzing each hyperparameter individually, these parameters yielded the bests two results. This combination provides a foundational basis for future experiments, offering a reference point for tackling more complex circuits.

The evolutionary search for several distinct MUX designs of varying sizes was motivated by the presence of defective NAND gates in modern nanoelectronics. Thus, we investigated

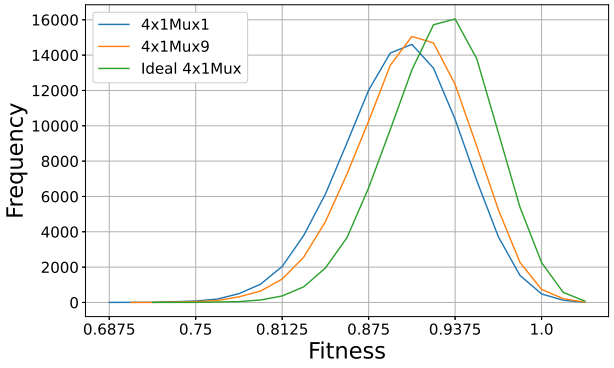

Fig. 3: Comparison of fitness distributions of different 4:1 multiplexers. The distributions were computed assuming a 5% defect probability for each NAND gate.

the fitness distribution of each design when a defect probability of 5% was attached to each NAND. A defect NAND gate implements an altered truth table and the same kind of defect was assumed for all NAND gates. The ideal NAND-based 4x1 MUX circuit can be represented by the direct acyclic graph shown in Fig. 1. To determine its fitness distribution, 100,000 fitness calculations were performed employing defective NAND gates. Employing the same approach, the fitness distributions of the evolved MUX library were also assessed. The resulting distributions are shown in Fig. 3.

It is perceived that the ideal 4x1 MUX still has a better fitness distribution compared to the ones of two selected designs from the evolved MUX library. However, as demonstrated in [4], a more detailed approach should directly search for designs with average high fitness and a narrow distribution width. In the present study, only an evolutionary search for degenerated designs with ideal fitness in the absence of defects was performed. Nevertheless, the result shown in Fig. 3 demonstrate that degenerated designs will lead to different fitness distributions. Thus, a corresponding CGP search for the best performing circuit can be conducted. Additionally, the experimental scenario considered a failure chance of 5%, which can be deemed low, given that the solution comprises only a few NAND gates. Future research should conduct a more detailed analysis, considering higher failure probabilities. Another interesting topic for evaluation is the application of redundancy through established techniques, such as von Neumann's NAND multiplexing. It would be valuable to assess how circuits evolved using these techniques perform relative to the ideal one.

VI. Conclusions

In this article, we made several contributions to the EDA of multiplexers. Firstly, through the use of the grid search technique, we gained important insights into the hyperparameters used in Cartesian Genetic Programming. Our experiments indicated better performance when the noise range is set to zero. Additionally, the number of offspring appears to be indirectly proportional to performance. We showed that CGP search can lead to degenerate MUX designs with different fitness distributions in the presence of NAND defects. Our analysis indicates that redundancy techniques and higher failure probabilities should be examined in future research to improve the robustness of CGP-designed circuits. This work provides a foundational basis for optimizing the tolerance against process defects of more complex circuits in subsequent studies.

Online Resources

The genetic program has been implemented as a Python module, facilitating its integration into diverse Python projects. A sample project is available in the GitHub Logic-Circuits-Evolution repository.

References

[1] R. Quhe, L. Xu, S. Liu, C. Yang, Y. Wang, H. Li, J. Yang, Q. Li, B. Shi, Y. Li, Y. Pan, X. Sun, J. Li, M. Weng, H. Zhang, Y. Guo, L. Xu, H. Tang, J. Dong, J. Yang, Z. Zhang, M. Lei, F. Pan, and J. Lu, "Sub-10 nm two-dimensional transistors: Theory and experiment," *Physics Reports*, vol. 938, pp. 1–72, 2021, sub-10 nm two-dimensional transistors: Theory and experiment. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0370157321003252

[2] O. Paranaiba, P. Oliveira, R. Marks, G. Novy, M. Vieira, L. O. Luz, P. A. Silva, R. Ferreira, J. Ramirez, J. A. Nacif *et al.*, "Design automation for emerging technologies," *Journal of Integrated Circuits and Systems*, vol. 17, no. 3, pp. 1–11, 2022.

[3] N. Milano, P. Pagliuca, and S. Nolfi, "Robustness, evolvability and phenotypic complexity: insights from evolving digital circuits," *Evolutionary Intelligence*, vol. 12, no. 1, pp. 83–95, 2019.

[4] N. Milano and S. Nolfi, "Robustness to Faults Promotes Evolvability: Insights from Evolving Digital Circuits," *PLOS ONE*, vol. 11, no. 7, pp. e0 158 627 – 17, 2016.

[5] L. Xingjun, S. Zhiwei, C. Hongping, and M. R. J. Haghighi, "A new design of qca-based nanoscale multiplexer and its usage in communications," *International Journal of Communication Systems*, vol. 33, no. 4, p. e4254, 2020.

[6] I. Varun and T. K. Gupta, "Ultra-Low Power NAND Based Multiplexer And Flip-Flop," *2013 Nirma University International Conference on Engineering (NUiCONE)*, pp. 1–5, 2013.

[7] V. Vaishali, S. Rajarajeshwari, and C. Saravanakumar, "A study on low power implementation of multiplexer," *Int. J. Emerg. Technol. Comput. Sci. Electron*, vol. 25, no. 5, 2018.

[8] B. F. Romanowicz, *Methodology for the Modeling and Simulation of Microsystems*. Springer Science & Business Media, 1998, vol. 2.

[9] L. Sekanina and Z. Vasicek, "Approximate circuit design by means of evolvable hardware," in *2013 IEEE International Conference on Evolvable Systems (ICES)*, 2013, pp. 21–28.

[10] Z. Vasicek and L. Sekanina, "Evolutionary approach to approximate digital circuits design," *IEEE Transactions on Evolutionary Computation*, vol. 19, no. 3, pp. 432–444, 2015.

[11] V. Mrazek, R. Hrbacek, Z. Vasicek, and L. Sekanina, "Evoapprox8b: library of approximate adders and multipliers for circuit design and benchmarking of approximation methods," in *Design, Automation & Test in Europe Conference & Exhibition (DATE), 2017*, 2017, pp. 258–261.

[12] A. Manazir and K. Raza, "Recent developments in cartesian genetic programming and its variants," *ACM Computing Surveys (CSUR)*, vol. 51, no. 6, pp. 1–29, 2019.

[13] S. Droste, T. Jansen, and I. Wegener, "On the analysis of the (1+ 1) evolutionary algorithm," *Theoretical Computer Science*, vol. 276, no. 1-2, pp. 51–81, 2002.

[14] M. Češka, J. Matyáš, V. Mrazek, L. Sekanina, Z. Vasicek, and T. Vojnar, "Approximating complex arithmetic circuits with formal error guarantees: 32-bit multipliers accomplished," in *2017 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*. IEEE, 2017, pp. 416–423.