On the Use of Clonal Selection Principle in Cartesian Genetic Programming for Designing Combinational Logic Circuits

José Eduardo H. da Silva[1], Luciana N. S. Prachedes[1], Heder S. Bernardino[1], José J. Camata[1], Itamar L. de Oliveira[1]

[1] Departamento de Ciência da Computação – Universidade Federal de Juiz de Fora
Minas Gerais, Brazil

{jehenriques,lucianaanascimento,heder,camata,itamar.leite}@ice.ufjf.br

Abstract. *Evolutionary techniques have been used in the design and optimization of combinational logic circuits. This procedure is called evolvable hardware and Cartesian Genetic Programming (CGP) is the evolutionary technique with the best performance in this context. Despite the good results obtained by CGP techniques, its search procedure usually evolves a single candidate solution by an evolution strategy and this approach tends to be trapped in local optima. On the other hand, clonal selection techniques in general, and CLONALG in particular, were designed to avoid converging to a low-quality local optimum. Thus, we propose here using the representation of CGP with the search procedure of a Clonal Selection Algorithm to minimize the number of transistors of combinational logic circuits. Furthermore, a parameter sensitivity analysis is performed. The results are assessed considering a benchmark from the literature and showed a reduction in the number of transistors when compared to the baseline ESPRESSO.*

1. Introduction

Digital circuits are present everywhere nowadays since the technology took place in all aspects of our lives. The miniaturization of electronic devices requires more compact circuits. Thus, optimization plays an important role in this regard.

The evolutionary computing community has been addressing the evolution of combinational logic circuits (CLCs) since the 1990s, being a part of Evolvable Hardware (EH) [Haddow and Tyrrell 2018]. EH refers to the employment of Evolutionary Algorithms (EAs) and Bio-Inspired Algorithms (BIAs) for creating, optimizing, and adapting physical hardware designs [Coello et al. 2000, Manfrini et al. 2016, Wang et al. 2008].

In general, the evolutionary design of circuits refers to the use of a randomly generated initial population, exploring possible solutions given a required behavior in a certain problem. The optimization refers to the minimization/maximization of some desired parameter, such as the number of logic gates, frequently motivated by practical needs [Hodan et al. 2021]. However, finding a fully working solution is a hard task, and there are scalability issues in which the computational complexity grows exponentially with the number of inputs.

Following the benchmark suite with several test-problems with different sizes [de Souza et al. 2020], in this work we aim the minimization of the number of transistors. The regular objective function for CLCs in the literature is to minimize the number of logic gates. The reason why de Souza et al. [de Souza et al. 2020] look to the number of transistors and not the number of gates is that transistors constitute the essential building block used to mount digital circuits in the context of integrated circuits. Therefore, when evaluating the effective cost of a given circuit, minimize the number of transistors enables it to be assembled into smaller and cheaper chips for the final product.

A common strategy when using EH [Stomeo et al. 2005] is the division of the problem into two, where first one desires to obtain a feasible solution and then optimize the solution [da Silva et al. 2018]. The interest in this work is the optimization of fully functional circuits, where the constraints represent the similarity between the outputs of the circuit being evaluated and those presented in a given truth table. Cartesian Genetic Programming (CGP) was adopted in [de Souza et al. 2020] and is pointed out as the best technique for evolving CLCs [Vasicek 2015]. Due to the CGP characteristic of evolving a single candidate solution by an evolutionary strategy, the method is inclined to be stuck in local optima. Here, we propose the modification of the standard CGP's evolutionary search procedure to a clonal selection algorithm called CLONALG [de Castro and von Zuben 2002], both in finding feasible solutions and minimizing the number of transistors of CLCs. This is motivated by the fact that CLONALG can escape from converging to low-quality local optima. The proposed combination of CGP with CLONALG is compared to a baseline CGP using the benchmark from [de Souza et al. 2020] and the proposal reached the best results in general.

The remainder of the paper is as follows: Section 2 describes the fundamentals of the techniques used in the development of this work. Section 3 goes into detail on how we combine CGP and CLONALG to improve its performance. In Section 4, the computational experiments are discussed and carried out with benchmark problems. Finally, Section 5 presents the conclusions and future works.

2. Methods

Artificial neural networks, ant colony algorithms, particle swarm optimization, artificial immune systems, and evolutionary techniques are examples of bio-inspired methods which have been widely applied in several areas, such as science, engineering and business management [Fan et al. 2020]. Artificial Immune Systems (AISs) are techniques which mimics the defence system of an organism against pathogens [Hatata et al. 2017] for solving computational intelligence problems. The immune system function in a live organism consists of innate and adaptive responses executed by specialized cells and molecules (antibodies) aimed at defending the organism against infection. The responses to invading microbes (antigens) can be of two fundamental types: innate (natural), the nonspecific immune system, in which the responses occur similarly to every pathogen; and adaptive, in which the responses improve on repeated exposure to a given infection [Delves and Roitt 2000].

Observing the behavior of this system and using its operating principles, different methods were implemented to solve problems: immune network theory [Zhang et al. 2013], negative selection [Ji and Dasgupta 2007], clonal selection [de Castro and von Zuben 2002], grammar-based immune programming [Bernardino and Barbosa 2011]. A clonal selection approach called CLONALG is used here as the search mechanism and is detailed in the following section.

2.1. Clonal Selection Algorithm

The clonal selection algorithm (CLONALG) [de Castro and von Zuben 2002] is an AIS approach which evolves the antibodies inspired by the concept of clonal expansion and selection. According to this method, each cell (candidate solution) is cloned, hypermutated, and those with higher antigenic affinity (quality) are selected.

An important step when using CLONALG is to determine the value of the mutation rate. Usually, it is taken inversely proportional to the affinity of the antibody with respect to the antigen. That is, it is proportional to the quality of the candidate solution when solving the optimization problem. Also, according to [Bernardino and Barbosa 2009a], AISs usually do not use recombination operators, such as crossover in traditional genetic programming techniques. Moreover, new candidate solutions can be randomly generated to improve the exploration capability of the algorithm.

Algorithm 1 shows CLONALG's pseudocode (adapted from [Bernardino and Barbosa 2009b]). In this algorithm, *affinities* contains the values of the objective function to be optimized (fitness), *antibodies* contains the population of candidate solutions. β is the number of clones that each antibody is allowed to generate, ρ is a parameter used to compute the mutation rate, *pRandom* is the percentage of new cells that are randomly generated, *clones* contains the population of clones generated by the function "clone", $nPop$ is the number of candidate solutions, and *cA* contains their corresponding affinities.

Algorithm 1: A CLONALG's pseudocode for optimization problems.

```
Data: $\beta$, $\rho$, $pRandom$, $nPop$
Result: $bestSolution$
1  begin
2      $antibodies \longleftarrow$ initializePopulation($nPop$);
3      $affinities \longleftarrow$ evaluate($antibodies$);
4      while stopping criteria is not met do
5          $clones \longleftarrow$ clone($antibodies$, $affinities$, $\beta$);
6          hypermutate($clones$, $affinities$, $\rho$);
7          $cA \longleftarrow$ evaluate($clones$);
8          select($antibodies$, $affinities$, $clones$, $cA$);
9          genNew($antibodies$, $affinities$, $pRandom$);
10     $bestSolution \longleftarrow$ getBest($antibodies$);
```

2.2. Cartesian Genetic Programming

Cartesian Genetic Programming (CGP) [Miller 2011] is a bio-inspired technique, in which programs are directed acyclic graphs (DAGs). CGP was firstly developed for evolving digital circuits. However, its representation is very flexible, and applications for evolving robot controllers [García and Coello 2002], neural networks [Goldman and Punch 2013] and image classifiers [Goldman and Punch 2015] can be found in the literature. CGP has three used defined parameters, namely number of

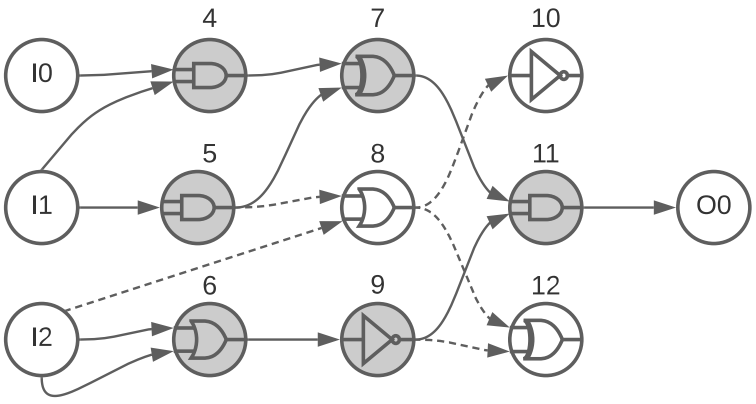

Figure 1. CGP individual's representation.

rows (n_r), number of columns (n_c), that defines the topology of the matrix, and levels-back (lb). The levels-back limits the number of columns at the left side where inputs can be selected to constrain the connectivity of the graph. Figure 1 presents a CGP individual with three primary inputs and one output. The functions are logical ones. The nodes that directly contribute to the outputs are called active nodes (grey nodes in Figure 1). The other ones are called inactive (white nodes in Figure 1). Continuous lines define the phenotype. Considering a situation which all nodes carry out some computational function, representing functions in the form of graphs is more concise than traditional GP trees, because they allow the reuse of previously calculated subgraphs [Miller 2011]

CGP commonly uses an evolutionary strategy (1+λ) as evolutionary mechanism, in which the best individual generates λ offspring. Usually λ = 4 [Miller 2011], presented in Algorithm 2.

Algorithm 2: Evolutionary strategy commonly used on CGP $(1 + \lambda)$.

```
1 begin
2     Initial population randomly generated;
3     Select the fittest individual to be the parent;
4     while stopping criteria is not met do
5         Mutate the parent to generate $\lambda$ new individuals;
6         Evaluate the $\lambda$ new individuals;
7         Select the fittest individual from the $(1 + \lambda)$ ones to be the next parent;
```

CGP mainly uses mutation as genetic variation operator. Miller [Miller 2011] highlights that there is no general-purpose crossover operator with good performance for CGP. However, for evolving digital circuits it is possible to find good-performing crossover operators [da Silva and Bernardino 2018, Walker et al. 2006, Husa and Kalkreuth 2018]. Mutation can vary depending on the CGP implementation and its function, for instance, point mutation (PM - the most common in CGP), single active mutation (SAM) [Goldman and Punch 2013] and multiple active mutation (MAM) [da Silva and Bernardino 2018]. In PM, an user defined parameter (μ_g) determines the number of nodes that will be mutated. The genes are randomly chosen and its value is changed to another valid random value. Furthermore, any node can be connected directly to the primary inputs, independently of the value of *lb*. However, PM can modify only inactive nodes, leading to wasted objective function evaluations since the offspring phenotype is the same as its parent. For this purpose, SAM was developed and it ensures that at least one active node is mutated for each individual of the offspring. While an active gene is not mutated, another gene is randomly selected and its value changed. This guarantees that (i) exactly one active node is mutated and (ii) zero or more inactive nodes can be mutated. MAM is based on SAM but mutates *n* (user defined) active nodes [da Silva and Bernardino 2018].

3. Proposed Approach

Here, we propose using CLONALG as the evolutionary mechanism of CGP instead of the the traditional evolutionary strategy $(1 + \lambda)$. Thus, the proposal adopts the original representation of CGP but the remaining of the search procedure is modified.In addition, to keep the diversity of the population, our proposal considers the following features in the search approach: (i) every antibody in the population is cloned, (ii) the number of clones is the same for every antibody, and (iii) each candidate solution is replaced if a better hypermutated clone is generated. Also, we defined *pRandom*$= 0$ and, thus, no random individual is generated during the evolutionary search. We intent to investigate this parameter in future works.

According to CLONALG, worst individuals are subject to more modification than the best ones, which need a finer tuning. The proposal here is to use the concept of somatic hypermutation borrowed from CLONALG in the mutation process that occurs in CGP. Thus, instead of just the fittest individual mutating to form the next generation, all individuals are cloned and modified to generate new candidate solutions. Also, these clones are modified according to the quality of the original candidate solutions. For this purpose, MAM is adopted here. The number of mutations is given by

$$nMutations(i) = round\left(\frac{nPop - 1 - i + (nMax \times i)}{nPop - 1}\right) \tag{1}$$

where ***nMutations*** is the number of active nodes modified in the i-th individual, ***nPop*** is the population size, $nMax$ is the maximum number of active nodes allowed to be modified and $round$ is a function that rounds its parameter to the nearest integer value. It is important to highlight that i represents the index of the ordered individuals according to the objective function, where $i = 0$ represents the best individual and $i = nPop - 1$ is the worst one.

The quality of an individual is given by its fitness. In the context of CGP applied to EH, fitness can be either the number of matches with respect to the truth table (for obtaining a feasible solution) and, once a feasible solution is found, the fitness is the combination of feasibility and the number of transistors of the solution.

Thus, according to the proposal, each individual of the population generates one child and the selection is performed by selecting the best individual between each parent and its corresponding offspring. Then, the population is ordered according to its fitness in order to apply the mutation scheme. We used the strategy from [Deb 2000] for ranking the candidate solutions, where: (i) any feasible solution is preferred when compared to an unfeasible one, (ii) when two feasible circuits are compared, that with the smallest number of transistors is better, and (iii) when two unfeasible solutions are compared, the one with smaller constraint violation (differences with respect to the truth table) is preferable.

Here we are interested in both (i) evolving and (ii) optimizing the circuits in terms of reducing the number of transistors. For this purpose, the objective function is changed from finding a feasible circuit to reduce the number of transistors, once a feasible circuit is found. Furthermore, according to the literature, CGP presents better results when considering a small population (5 individuals) [Miller 2011, da Silva et al. 2018, da Silva and Bernardino 2018]. However, a parameter sensitivity analysis is also performed here, in order to verify the behavior of the evolutionary search when using higher populations.

4. Computational Experiments

Computational experiments were performed to analyze the performance of the proposal when designing and optimizing CLCs in terms of reducing the number of transistors.The source code and supplementary material can be found at Github[1]. Also, a parameter sensitivity analysis is performed, where the proposed CGP with CLONALG is analyzed according to the benchmark proposed in [de Souza et al. 2020]. We analyzed the number of individuals in the population ($nPop$) and the maximum number of active nodes allowed to be modified during the hypermutation (nMax).

The results assessment considers the three mandatory metrics presented in [de Souza et al. 2020]: (i) The number of transistors of the solutions: this is the objective function of the optimization problem and reflects the characteristics of the circuit; (ii) the success rate (SR, in %): percentage of the independent runs in which a feasible solution is obtained. This performance measurement reflects the ability of search methods in obtaining fully functional circuits; and (iii) the relative reduction (RR, in %): represents the capacity of the search method in generating compact circuits. It is calculated as:

$$RR_{p,s} = \frac{|f_m(x_{p,s}) - f(x_{p,b})|}{f(x_{p,b})} \times 100, \quad (2)$$

where $f_m(x_{p,s})$ is the median of the number of transistors obtained by solver s when applied to problem p, and $f(x_{p,b})$ is the number of transistors of a baseline solution. ESPRESSO is a popular heuristic for optimization of CLCs and is adopted here as the baseline, as in [de Souza et al. 2020].

We also consider the use of performance profiles (PPs) [Dolan and Moré 2002, Barbosa et al. 2010] for analysing the relative performance of the algorithms. In PPs, the probability that the performance of a given method is within a factor $\tau > 1$ of the best one is $\rho_s(\tau)$, and one can extract: (i) the approach that obtained the best results for most problems (largest $\rho(1)$), (ii) the most reliable approach (smaller τ such that $\rho(\tau) = 1$), and (iii) the best overall performance (largest area under the PPs curves).

The problems are the same presented in [de Souza et al. 2020], categorized into three groups, considering the circuits' simplification, balancing, and the number of inputs and outputs.

[1]`https://github.com/ciml/cgp_clonalg`

The proposal was evaluated with respect to the following parameter settings: (i) $nPop = 10$ and $nMax = 3$ (labeled as 10_3), (ii) $nPop = 10$ and $nMax = 5$ (labeled as 10_5), and (iii) $nPop = 20$ and $nMax = 3$ (labeled as 20_3). Due to the stochastic nature of metaheuristics, 15 independent runs were performed for each scenario. For all experiments, we used the CGP parameters suggested in [de Souza et al. 2020], excepting lb, in which lb = nc, as recommended by the literature [da Silva et al. 2018, da Silva and Bernardino 2018, Goldman and Punch 2013, Miller 2011]. The comparison is performed considering the variation SAM-R, a implementation with CGP with SAM, starting with a randomly-generated initial population [de Souza et al. 2020].

Figure 2 presents the results of the performance profile for the median of the number of transistors considering all problems and methods. Based on PP, it is possible to conclude that (i) 20_3 obtained the best results for most problems (largest $\rho 1$), (ii) SAM-R can be considered the most reliable (obtained at least a feasible solution in more problems), and (iii) 20_3 has the best overall performance (largest area under the performance profiles curves).

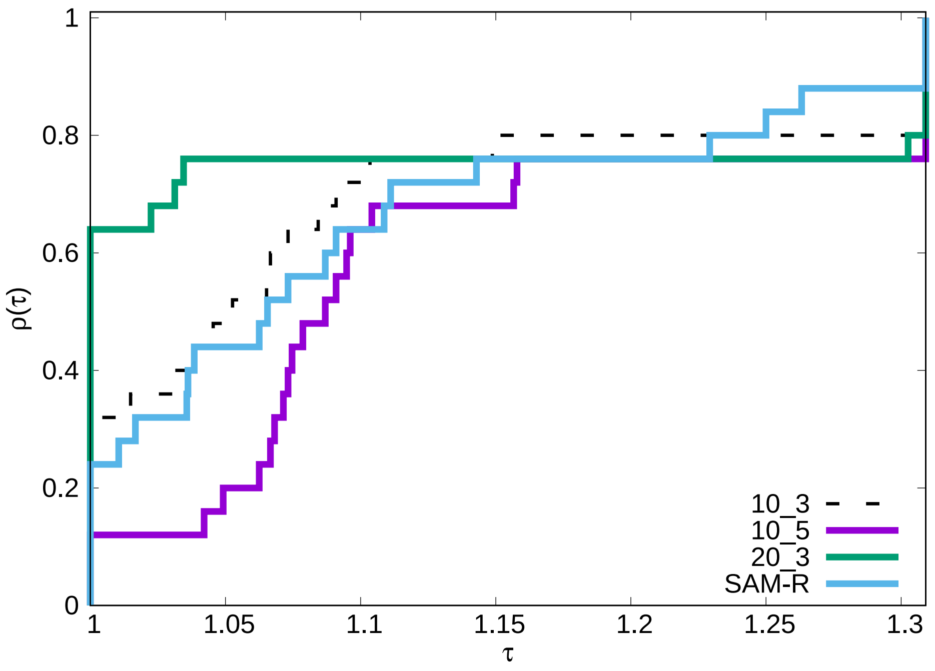

Figure 2. Performance Profiles for the median of the number of transistors considering all methods and problems. The normalized areas under the PP curves are: 10_3 - 0.9255, 10_5 - 0.7728, 20_3 - 1.00, and SAM-R - 0.8895.

Tables 1, 2, and 3 present the results for SAM-R and the proposed variants for groups 1, 2, and 3, respectively. The results of cmb, cordic, and vda are not presented since neither the benchmark nor the proposed approaches found feasible solutions. Both 10_3 and 20_3 obtained better results when compared to SAM-R. Furthermore, the relative reductions obtained by the proposal are higher than or equal to those obtained by SAM-R in most cases. However, 20_3 has lower success rates, in general, when compared to both 10_3 and SAM-R. Statistical significance tests results can be found on the supplementary material.

Table 1. Results for all problems of Group 1. Methods with asterisks (*) presents statistical difference. Best results are in boldface.

Method	Best	Q1	Median	Q3	Worst	Mean	std.	SR (%)	RR (%)
C17									
SAM-R	**9.00**	10.00	10.00	10.00	11.00	9.80	4.90e-01	**100.00**	61.54
10_3*	**9.00**	**9.00**	**9.00**	10.00	10.00	9.33	**4.88e-01**	**100.00**	**65.38**
10_5	**9.00**	**9.00**	**9.00**	10.00	10.00	9.47	5.16e-01	**100.00**	**65.38**
20_3*	**9.00**	**9.00**	**9.00**	**9.00**	**9.00**	**9.00**	0.000e+00	**100.00**	**65.38**
cm42a									
SAM-R	29.00	**31.00**	34.00	36.00	38.00	33.48	2.66e+00	**100.00**	78.21
10_3	30.00	32.00	**32.00**	**33.00**	**34.00**	**32.27**	**1.22e+00**	**100.00**	**79.49**
10_5	**28.00**	32.5	34.00	34.00	36.00	33.20	1.90e+00	**100.00**	78.21
20_3	30.00	**31.00**	33.00	**33.00**	35.00	32.46	1.39e+00	**100.00**	78.85
cm82a									
SAM-R	20.00	24.00	25.00	29.00	47.00	27.36	6.67e+00	**100.00**	84.28
10_3	**19.00**	**22.00**	**23.00**	23.5	25.00	22.87	1.60e+00	**100.00**	**85.53**
10_5	22.00	23.00	**23.00**	24.5	27.00	23.80	1.57e+00	**100.00**	**85.53**
20_3	**19.00**	**22.00**	**23.00**	**23.00**	**24.00**	**22.15**	**1.41e+00**	**100.00**	**85.53**
cm138a									
SAM-R	27.00	30.00	32.00	33.00	40.00	32.36	2.94e+00	**100.00**	78.38
10_3	**26.00**	27.50	**28.00**	**28.00**	**30.00**	27.87	1.006e+00	**100.00**	**81.008**
10_5	27.00	28.00	**28.00**	29.50	33.00	28.87	1.60e+00	**100.00**	**81.008**
20_3	**26.00**	**27.00**	**28.00**	**28.00**	28.00	**27.46**	**6.60e-01**	**100.00**	**81.008**
decod									
SAM-R	**41.00**	49.00	54.00	57.00	64.00	53.12	5.52e+00	**100.00**	59.09
10_3*	**41.00**	49.00	**52.00**	56.00	59.00	51.87	5.004e+00	**100.00**	**60.61**
10_5*	48.00	54.5	57.00	61.00	64.00	56.53	5.000e+00	**100.00**	56.82
20_3	42.00	**48.00**	**52.00**	**55.00**	**57.00**	**50.92**	**4.70e+00**	**100.00**	**60.61**
f51m									
SAM-R	74.00	**80.00**	**89.00**	**93.00**	128.00	89.60	1.22e+01	**100.00**	**86.05**
10_3	85.00	87.00	**89.00**	95.00	**97.00**	90.46	**4.35e+00**	86.67	**86.05**
10_5	**71.00**	85.00	96.00	105.00	119.00	95.15	1.50e+01	86.67	84.95
20_3	75.00	82.00	91.00	94.00	103.00	**88.77**	7.67e+00	**100.00**	85.74
majority									
SAM-R	**11.00**	12.00	12.00	13.00	23.00	12.92	2.56e+00	**100.00**	50.00
10_3	**11.00**	**11.00**	12.00	**12.00**	13.00	11.80	6.76e-01	**100.00**	50.00
10_5	**11.00**	12.00	12.00	**12.00**	13.00	12.13	5.16e-01	**100.00**	50.00
20_3	**11.00**	**11.00**	**11.00**	**12.00**	**12.00**	**11.38**	**5.06e-01**	**100.00**	**54.17**
z4ml									
SAM-R	35.00	42.00	48.00	51.00	76.00	48.08	8.86e+00	**100.00**	90.46
10_3	**34.00**	38.00	40.00	41.5	45.00	39.73	3.01e+00	**100.00**	92.05
10_5	36.00	38.5	41.00	45.00	47.00	41.47	3.80e+00	**100.00**	91.85
20_3	35.00	**36.00**	**38.00**	**39.00**	**42.00**	**37.92**	**2.25e+00**	**100.00**	**92.45**

Table 2. Results for all problems of Group 2. Methods with asterisks (*) presents statistical difference. Best results are in boldface.

Method	Best	Q1	Median	Q3	Worst	Mean	std.	SR	RR
9symml									
SAM-R	77.00	100.00	118.00	157.00	210.00	128.84	3.95e+01	**100.00**	88.64
10_3	89.00	95.5	99.00	117.00	126.00	106.00	**1.27e+01**	**100.00**	90.47
10_5	88.00	103.00	106.00	120.00	152.00	112.00	1.56e+01	**100.00**	89.8
20_3	**75.00**	**83.00**	**96.00**	**108.00**	**122.00**	**95.31**	1.56e+01	**100.00**	**90.76**
alu2									
SAM-R	**306.00**	**316.75**	**327.50**	**338.25**	**349.00**	**327.50**	**2.15E+01**	**8.00**	**81.70**
10_3	-	-	-	-	-	-	-	0.00	-
10_5	-	-	-	-	-	-	-	0.00	-
20_3	-	-	-	-	-	-	-	0.00	-
alu4									
SAM-R	**158.00**	**181.50**	**205.00**	**243.50**	**267.00**	**209.70**	**3.40E+01**	**92.00**	**98.02**
10_3	-	-	-	-	-	-	-	0.00	-
10_5	-	-	-	-	-	-	-	0.00	-
20_3	-	-	-	-	-	-	-	0.00	-
cm85a									
SAM-R	**42.00**	47.00	51.00	56.00	68.00	51.84	7.14e+00	**100.00**	91.64
10_3*	45.00	48.00	49.00	50.5	54.00	49.27	2.40e+00	**100.00**	91.97
10_5	43.00	49.00	50.00	51.00	53.00	49.47	2.53e+00	**100.00**	91.80
20_3*	44.00	**45.00**	**46.00**	**48.00**	**49.00**	**46.38**	**1.56e+00**	**100.00**	**92.46**
cm151a									
SAM-R	**36.00**	42.00	44.00	46.00	78.00	46.00	8.11e+00	**100.00**	71.43
10_3	38.00	41.00	44.00	45.00	**46.00**	42.93	2.58e+00	**100.00**	71.43
10_5	42.00	43.00	44.00	45.00	50.00	44.53	**2.07e+00**	**100.00**	71.43
20_3	39.00	**40.00**	**41.00**	**43.00**	**46.00**	**41.69**	2.18e+00	**100.00**	**73.38**
cm162a									
SAM-R	**54.00**	63.00	65.00	69.00	78.00	66.00	5.97e+00	**100.00**	67.50
10_3*	57.00	**59.00**	**61.00**	64.00	65.00	61.00	2.80e+00	**100.00**	**69.50**
10_5*	55.00	61.50	64.00	65.5	67.00	63.07	3.37e+00	**100.00**	68.00
20_3	57.00	**59.00**	**61.00**	**62.00**	**64.00**	**60.54**	**2.30e+00**	**100.00**	**69.50**
cu									
SAM-R	**58.00**	**62.00**	**67.00**	**68.00**	79.00	**66.28**	5.12e+00	**100.00**	**74.33**
10_3*	65.00	67.50	68.00	70.00	80.00	69.13	3.58e+00	**100.00**	73.95
10_5*	63.00	69.00	72.00	74.00	84.00	72.007	5.16e+00	**100.00**	72.41
20_3	64.00	66.00	**67.00**	**68.00**	**71.00**	67.23	**1.96e+00**	**100.00**	**74.33**
x2									
SAM-R	**51.00**	**56.00**	61.00	67.00	99.00	62.80	9.51e+00	**100.00**	64.94
10_3*	57.00	61.00	64.00	66.00	70.00	63.73	3.59e+00	**100.00**	63.22
10_5	57.00	63.00	64.00	68.75	72.00	65.07	4.20e+00	93.33	63.22
20_3*	56.00	59.00	**60.00**	**63.00**	**66.00**	**60.62**	**2.90e+00**	**100.00**	**65.52**

Table 3. Results for all problems of Group 3. Methods with asterisks (*) presents statistical difference. Best results are in boldface.

Method	Best	Q1	Median	Q3	Worst	Mean	std.	SR	RR
cc									
SAM-R	**74.00**	**81.00**	87.00	92.00	98.00	86.76	6.27e+00	**100.00**	66.02
10_3*	78.00	85.50	87.00	90.00	97.00	87.40	4.72e+00	**100.00**	66.02
10_5	84.00	87.50	90.00	94.50	101.00	91.07	5.00e+00	**100.00**	64.84
20_3*	79.00	**84.00**	**84.00**	**85.00**	**88.00**	**83.85**	**2.64e+00**	**100.00**	**67.19**
frg1									
SAM-R	84.00	89.00	96.00	100.00	114.00	96.20	7.27e+00	**100.00**	94.02
10_3	89.00	92.00	**95.00**	**97.00**	**102.00**	94.67	**3.27e+00**	**100.00**	**94.08**
10_5	**83.00**	94.50	99.00	100.75	104.00	97.07	5.53e+00	93.33	93.83
20_3	84.00	**91.00**	**95.00**	**97.00**	**102.00**	**94.15**	4.65e+00	**100.00**	**94.08**0
pm1									
SAM-R	**53.00**	**57.00**	**58.00**	**59.00**	**62.00**	**57.84**	**2.09e+00**	**100.00**	**97.22**
10_3*	58.00	60.50	64.00	69.00	72.00	64.73	4.85e+00	**100.00**	96.93
10_5	60.00	62.00	63.50	65.75	71.00	63.86	3.28e+00	93.33	96.95
20_3*	57.00	59.00	60.00	64.00	65.00	60.62	3.04e+00	**100.00**	97.12
sct									
SAM-R	**83.00**	**93.5**	**97.50**	**103.00**	**112.00**	**97.72**	6.93e+00	**72.00**	**79.08**
10_3	112.00	112.00	112.00	112.00	112.00	112.00	0.00	6.67	75.97
10_5	-	-	-	-	-	-	-	0.00	-
20_3	127.00	127.00	127.00	127.00	127.00	127.00	0.00	6.67	72.75
t481									
SAM-R	43.00	48.00	55.00	68.00	144.00	63.60	2.40e+01	**100.00**	99.42
10_3*	41.00	43.50	46.00	47.50	50.00	45.93	2.81e+00	**100.00**	99.52
10_5	41.00	45.00	47.00	49.50	56.00	47.47	3.60e+00	**100.00**	99.51
20_3*	**39.00**	**42.00**	**44.00**	**46.00**	**47.00**	**43.62**	**2.66e+00**	**100.00**	**99.54**
tcon									
SAM-R	**36.00**	**38.00**	43.00	49.00	57.00	43.80	6.31e+00	**100.00**	12.24
10_3*	**36.00**	40.00	**42.00**	**44.00**	**49.00**	**42.40**	3.68e+00	**100.00**	14.29
10_5*	38.00	47.50	48.00	49.00	52.00	47.60	**3.60e+00**	**100.00**	2.04
20_3	38.00	39.75	**41.50**	45.50	**49.00**	42.42	3.78e+00	93.33	**15.31**

5. Concluding Remarks and Future Work

We propose here improving CGP by using the search procedure of CLONALG when designing and optimizing CLCs. Computational experiments were carried out based on a benchmark with test-problems with different sizes (disposed into tree different groups) and functionalities. In addition, we performed a parameter evaluation for the number of individuals in the population and the maximum number of active nodes allowed to be modified.

When compared with SAM-R, the proposed 10_3 and 20_3 obtained the best results in general. Also, the relative reductions obtained by the proposed approaches are higher than or equal to those obtained by SAM-R in most cases. On the other hand, 20_3 has lower success rates in general when compared to SAM-R.

As a future work, we intend to merge CGP with others immune-inspired algorithms, such as those inspired by the immune network theory. Also, future work must consider to apply the proposed approach to solve larger problems. As the proposed approach increases the population size of CGP, high-performance computing techniques can be explored to deal with the computational cost of the evaluations.

References

Barbosa, H. J., Bernardino, H. S., and Barreto, A. M. (2010). Using performance profiles to analyze the results of the 2006 cec constrained optimization competition. In *Evolutionary Computation (CEC), 2010 IEEE Congress on*, pages 1–8. IEEE.

Bernardino, H. S. and Barbosa, H. J. (2009a). Artificial immune systems for optimization. In *Nature-Inspired Algorithms for Optimisation*, pages 389–411. Springer.

Bernardino, H. S. and Barbosa, H. J. C. (2009b). *Nature-Inspired Algorithms for Optimisation*, chapter Artificial Immune Systems for Optimization, pages 389–411. Springer Berlin / Heidelberg.

Bernardino, H. S. and Barbosa, H. J. C. (2011). Grammar-based immune programming. *Nat. Comput.*, 10(1):209–241.

Coello, C., Aguirre, A., and Buckles, B. (2000). Evolutionary multiobjective design of combinational logic circuits. In *Proceedings. The Second NASA/DoD Workshop on Evolvable Hardware*, pages 161–170. IEEE.

da Silva, J. E. and Bernardino, H. S. (2018). Cartesian genetic programming with crossover for designing combinational logic circuits. In *2018 7th Brazilian Conference on Intelligent Systems (BRACIS)*, pages 145–150. IEEE.

da Silva, J. E. H., Manfrini, F. A., Bernardino, H. S., and Barbosa, H. J. (2018). Biased mutation and tournament selection approaches for designing combinational logic circuits via cartesian genetic programming. In *Anais do XV Encontro Nacional de Inteligência Artificial e Computacional*, pages 835–846. SBC.

de Castro, L. N. and von Zuben, F. J. (2002). Learning and optimization using the clonal selection principle. *IEEE Trans. Evo. Comp.*, 6(3):239–251.

de Souza, L. A. M., da Silva, J. E. H., Chaves, L. J., and Bernardino, H. S. (2020). A benchmark suite for designing combinational logic circuits via metaheuristics. *Applied Soft Computing*, 91:106246.

Deb, K. (2000). An efficient constraint handling method for genetic algorithms. *Computer Methods in Applied Mechanics and Engineering*, 186(2):311–338.

Delves, P. J. and Roitt, I. M. (2000). The immune system. *New England journal of medicine*, 343(1):37–49.

Dolan, E. D. and Moré, J. J. (2002). Benchmarking optimization software with performance profiles. *Mathematical Program.*, 91(2):201–213.

Fan, X., Sayers, W., Zhang, S., Han, Z., Ren, L., and Chizari, H. (2020). Review and classification of bio-inspired algorithms and their applications. *Journal of Bionic Engineering*, 17(3):611–631.

García, B. M. and Coello, C. A. C. (2002). An approach based on the use of the ant system to design clcs. *Mathware and Soft Computing*, 9(2-3):235–250.

Goldman, B. W. and Punch, W. F. (2013). Reducing wasted evaluations in cartesian genetic programming. In *European Conference on GP*, pages 61–72. Springer.

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

Haddow, P. C. and Tyrrell, A. M. (2018). Evolvable hardware challenges: Past, present and the path to a promising future. *Inspired by Nature*, pages 3–37.

Hatata, A. Y., Osman, M. G., and Aladl, M. M. (2017). A review of the clonal selection algorithm as an optimization method. *Leonardo Journal of Sciences*, 16(30):1–14.

Hodan, D., Mrazek, V., and Vasicek, Z. (2021). Semantically-oriented mutation operator in cartesian genetic programming for evolutionary circuit design. *Genetic Programming and Evolvable Machines*, 22(4):539–572.

Husa, J. and Kalkreuth, R. (2018). A comparative study on crossover in cartesian genetic programming. In *European Conference on GP*, pages 203–219. Springer.

Ji, Z. and Dasgupta, D. (2007). Revisiting negative selection algorithms. *Evolutionary computation*, 15(2):223–251.

Manfrini, F. A., Bernardino, H. S., and Barbosa, H. J. (2016). A novel efficient mutation for evolutionary design of combinational logic circuits. In *International Conference on Parallel Problem Solving from Nature*, pages 665–674. Springer.

Miller, J. F. (2011). Cartesian genetic programming. In *Cartesian genetic programming*, pages 17–34. Springer.

Stomeo, E., Kalganova, T., Lambert, C., Lipnitsakya, N., and Yatskevich, Y. (2005). On evolution of relatively large combinational logic circuits. In *2005 NASA/DoD Conference on Evolvable Hardware (EH'05)*, pages 59–66. IEEE.

Vasicek, Z. (2015). Cartesian gp in optimization of combinational circuits with hundreds of inputs and thousands of gates. In *European Conference on Genetic Programming*, pages 139–150. Springer.

Walker, J. A., Miller, J. F., and Cavill, R. (2006). A multi-chromosome approach to standard and embedded cartesian genetic programming. In *Proceedings of the 8th annual conference on Genetic and evolutionary computation*, pages 903–910. ACM.

Wang, J., Chen, Q. S., and Lee, C. H. (2008). Design and implementation of a virtual reconfigurable architecture for different applications of intrinsic evolvable hardware. *IET Computers & Digital Techniques*, 2(5):386–400.

Zhang, W., Yen, G. G., and He, Z. (2013). Constrained optimization via artificial immune system. *IEEE Transactions on Cybernetics*, 44(2):185–198.