On the Parameterization of Cartesian Genetic Programming

Paul Kaufmann
Gutenberg School of Management & Economics
Johannes Gutenberg University, Mainz, Germany
paul.kaufmann@uni-mainz.com

Roman Kalkreuth
Department of Computer Science
Dortmund Technical University, Germany
roman.kalkreuth@tu-dortmund.de

***Abstract*—In this work, we present a detailed analysis of Cartesian Genetic Programming (CGP) parametrization of the selection scheme ($\mu + \lambda$), and the levels back parameter l. We also investigate CGP's mutation operator by decomposing it into a self-recombination, node function mutation, and inactive gene randomization operators. We perform experiments in the Boolean and symbolic regression domains with which we contribute to the knowledge about efficient parametrization of two essential parameters of CGP and the mutation operator.**

***Index Terms*—Cartesian Genetic Programming**

I. Introduction

Genetic Programming (GP) can be considered as nature-inspired search heuristic, which allows for the automatic derivation of programs for problem-solving. First work on GP has been done by [1], [2] and [3]. Later work by [4] significantly popularized the field of GP. GP is traditionally used with trees as program representation. Over two decades ago Miller, Thompson, Kalganova, and Fogarty presented first publications on Cartesian Genetic Programming (CGP) — an encoding model inspired by the two-dimensional array of functional nodes connected by feed-forward wires of an Field Programmable Gate Array (FPGA) device [5] CGP offers a graph-based representation that, in addition to standard GP problem domains, makes it easy to be applied to many graph-based applications such as electronic circuits [6], [7], image processing [8], and neural networks [9]. The geometric and algorithmic parameters of CGP are usually configured according to the findings of Miller [10]. The experiments of Goldman and Punch [11], [12] and Kaufmann and Kalkreuth [13], [14] showed, however, that CGP's standard parameterization can be improved significantly in different ways. In this work, we argue that by taking a more in-depth look into the interplay of CGP's inner mechanisms, the convergence can be improved even further. For instance, CGP relies solely on the mutation operator. The mutation operator decomposes functionally into the self-recombination, node function mutation, and inactive gene randomization operators. The first two operators operate only on genes that contribute to the coding of a CGP phenotype. The inactive gene randomization operator exclusively touches genes that are not used for the *genotype-to-phenotype mapping*. The self-recombination and inactive gene randomization operators are linked together by CGP's optimization algorithm. Traditionally, CGP uses a variant of the $(1 + 4)$ Evolutionary Strategy (ES) that prefers off-spring individuals over parents of the same fitness. The consequence of the selection preference is that inactive genes that have been touched by the randomization operator do not contribute to a change of the fitness and therefore propagate always to the next generation. This circumstance allows a steady influx of random genetic material, and we assume that it supports the self-recombination operator to explore close and distant parts of the search space, i.e., increase its search horizon.

The search horizon of the self-recombination operator can also be improved using another inner mechanism of CGP. Goldman and Punch showed that active genes are distributed unevenly in a single-line CGP model [15]. The probability to be used for the encoding of a phenotype rises for genes located close to the beginning of a genotype. However, the effectiveness of the self-recombination operator depends on the amount of genetic and random genetic material accessible to it. Distributing the probability of active gene locations more evenly throughout the genotype by setting the levels back parameter to some fraction of the genotype length improves self-recombination's effectiveness.

The role of node function mutation and its relation to the self-recombination operator has not been investigated yet. This also has been the major motivation for our work. Our experiments show that node function mutation contributes to the successful evolution of goal functions but that the self-recombination contributes to a greater extent. Switching off node function mutation can, therefore, improve the convergence rates of CGP.

This paper is divided into six Sections. After the Introduction, Section II surveys related work for our experiments. Section III introduces Cartesian Genetic Programming. We discuss the motivation and methodology for our experiments in Section IV. The experiments are presented and discussed in Section V. Section VI concludes our work.

II. Related Work

Many works and contributions in the field of CGP primarily head for extending and improving CGP by novel ideas such as biogeography-based optimization [16], introduction of recurrent wires [17], and the evolution of differentiation systems [18]. Recently, Miller [19] published a survey on the status of CGP and reflected the many extensions that have been introduced to CGP in recent years. However, his article surveys only a few works, which address the understanding of the inner mechanisms of standard CGP and its mutation operator. The work of Turner and Miller [20] takes a deeper look into the various "neutrality" effects in CGP. The authors investigate the effects of "algorithmic neutrality", i.e., preferring off-spring individuals over parents of same fitness and "neutral genetic drift" in inactive genotype regions. They also take a look into scaling up CGP's genotype sizes in search of the right balance between search space sizes and convergence speeds.

Closer to our work are the results of Kaufmann and Kalkreuth [14], [13]. Using an automatic parameter tuner [21], the authors tested and evaluated different grid and $(\mu + \lambda)$ configurations of CGP and made comparisons to Simulated Annealing (SA). The outcome of their study is that for regression benchmarks, large μ's and huge λ's as well as for combinational benchmarks SA perform best. The use of an automatic parameter tuner limits, however, the scope of Kaufmann and Kalkreuth's study. It becomes more challenging to derive general dependencies of CGP's parameters from these experiments. This work follows a different approach – the experiments test as many relevant parameter combinations as possible to find relationships between them.

The contribution which can bee seen as closest to our work are the findings of Goldman and Punch [15], [11], [12]. In their work, the authors have defined a parameter-free mutation procedure, called "single-mutation", that we utilize for our experiments. The mutation procedure implements a loop that picks a gene randomly and mutates it by flipping the respective value in the legal range. Setting the maximal loop count to one would correspond to the conventional mutation operator of CGP. Single-mutation's loop, however, stops after hitting an active gene for the first time. The rationale behind this mutation procedure is that the functional quality needs an evaluation only if the encoded phenotype changes.

Another result of Goldman and Punch is the analysis of the distribution of active genes within the genotype of a CGP individual. When not restricting CGP's wire length, the probability for genes being active is higher if located at the beginning of the genotype. A cluster of active genes at the beginning of a genotype results in a representational bias towards smaller phenotypes. Evolution of functions with larger topologies is hampered by this property [15]. Goldman and Punch also conjectured that the steady injection of random genes into the inactive genotype regions may be beneficial for CGP. In this work, we validate this hypothesis experimentally.

Compared to the number of works that introduce manifold novel ideas for CGP, we feel that only a minority aims to achieve a more fundamental understanding of CGP. This has been the major motivation for our work. The contributions of our work to the understanding of CGP are, on the one hand, more clarity about the parametrization of two significant parameters of CGP. This is in line with the work of Kaufmann and Kalkreuth, however, in a more systematic manner. We also follow up the work of Goldman and Punch on the *single-mutation* operator with further experiments on the decomposition of CGP's mutation operator.

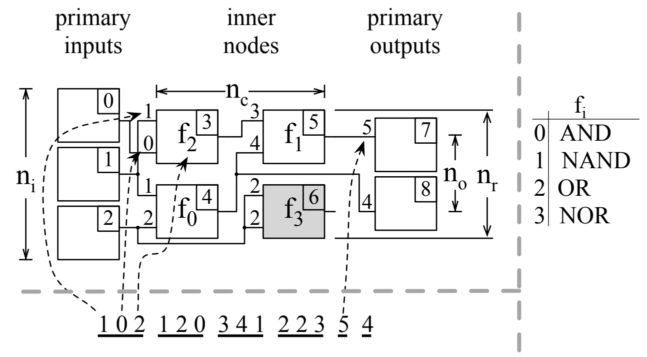

Fig. 1: Cartesian Genetic Program (top) and its encoding (bottom). All nodes contributing to primary outputs are called "active" and all other nodes "inactive". For example, node 6 is an inactive node.

III. Cartesian Genetic Programming

This section introduces the representation model, the evolutionary operator, and the optimization procedure of CGP.

A. The Representation Model

Cartesian Genetic Programming is a representation model that encodes the goal function as a directed acyclic graph (DAG) [22]. An exemplification of the encoding is illustrated in Fig. 1. The graph nodes are arranged as a $n_c \times n_r$ grid and are connected by feed-forward wires (c.f. top of Fig. 1). Inputs to the graph are sourced in by n_i primary inputs, and the graph's outputs are fed into n_o primary output nodes. All nodes are enumerated column-wise and from left to right. A wire can be restricted to span at most l columns. Each node may have up to n_a ingoing wires and computes a single output. The lengths of wires connecting primary outputs with inner nodes and inner nodes with primary inputs are restricted by the levels back parameter l. Usually, l is set to span the entire genotype ($l = \infty$). Each node may implement a function from the functional set.

The graph in CGP is encoded by a linear list of integers, as shown at the bottom of Fig. 1. The first $n_a + 1$ integers encode the n_a input wires (connection genes) and the function gene of the upper left node of the grid. In Fig. 1, the upper left node with the index 3 connects to the primary inputs 1 and 0 and computes the function $f_2 = OR$. The encoding of node three is, therefore "1 0 2". The next $n_a + 1$ integers encode the configuration of node four and so on. The encoding scheme proceeds column-by-column and from left to right until all inner nodes are specified. The n_o primary outputs are encoded by n_o connection genes. For example, the first primary output in Fig. 1 with the index 7 connects to node five and is therefore coded as "5" in the genotype.

The genotype size of CGP is fixed and amounts for $n_r n_c(n_a + 1) + n_o$ genes. However, not all genes are lying on paths that are contributing to primary outputs. For instance, the output of node 6 in Fig. 1 is not connected directly or transitively to a primary output. Such nodes and their genes are called "inactive" because they are not required to implement the phenotype. All other nodes and genes are called "active".

The size of CGP's genotype can be reduced, and in consequence, the convergence improved, by setting the number of rows to $n_r = 1$. Related work almost exclusively uses this "single-line" CGP model.

B. The Mutation Operator

CGP relies mostly on the mutation operator [22]. The operator works on the linear list of genes. Given the mutation rate and the length of the genotype, the operator computes how many genes have to be mutated. These genes are then selected randomly. A function gene is mutated by choosing a function from a set of valid functions randomly. A connection gene is mutated by randomly rewiring it to a preceding node within the range given by the levels back parameter.

C. The Optimization Algorithm

CGP is traditionally optimized by an (1+λ) Evolutionary Algorithm (EA). The selection scheme is implemented such that off-spring individuals that are as fit or better as the parent are preferred when selecting the new parent for the next generation. In [7] it has been shown by Miller that Kozas's Computational Effort (CE) [23] becomes minimal for $\mu = 1$ and $\lambda = 4$ [10]. Since then, the $(1 + 4)$ EA has been used predominantly by works employing CGP.

The characteristic property of CGP's $(1 + \lambda)$ selection scheme is that those off-spring individuals with changes only in inactive genes have identical fitness as the current parent and are preferred when choosing the parent for the next generation. As λ is usually set to a small number (i.e., four), CGP's EA is executed for thousands to millions of generations. Because only in very few generations, relative to the overall number, fitness can be improved, in a lot of remaining generations random changes of inactive genes are propagated to the next generation. The combination of CGP's selection scheme with the mutation of inactive genes creates a randomization operator that forces a steady influx of random genetic material into CGP's genotypes. This, as we will show later, can help to improve the convergence of CGP.

IV. Preliminaries

This section describes the motivation and the methodology for the experiments done in this work.

A. Parametrization of CGP's selection scheme

In related work, CGP is often configured according to the findings of Turner and Miller [10]. This includes setting the number of rows to $n_r = 1$, configuring the levels back parameter as $l = n_c$, and using the $(1 + 4)$ selection scheme. Later work by Goldman and Punch [12] introduced the parameter-free *single-mutation* operator and showed that it is often more efficient than the original mutation algorithm. The idea behind *single-mutation* is to repeatedly mutate genes until an active gene has been changed for the first time. This helps avoiding redundant evaluation of candidate solutions that encode the same phenotype as their parent. Since the introduction of *single-mutation*, more and more works resorted to the new operator. There has been done, however, no study on how *single-mutation* interacts with other parameters of CGP. Previous work by Kaufmann and Kalkreuth [14], [13] showed that the effectiveness of the traditional $(1+4)$-CGP couldn't be generalized. Because Kaufmann and Kalkreuth have used an automated parameter tuner, identification of general parameter dependencies is limited in their study.

In our first experiment, we extend the work of Kaufmann and Kalkreuth by a grid search on μ and λ for a better understanding of the interplay between of parameters and their impact on CGP when using "single-mutation". Those values for μ and λ that generalize best are then selected for the subsequent experiment to configure CGP's EA.

B. Decomposition of CGP's Mutation Operator

The original mutation operator of CGP works at the genotype level, which is defined as a sequence of integers (cf. Sec. III). This is different to traditional GP, where evolutionary operators act on the graph of a candidate solution. Depending on the effects an operator may induce in a GP graph, one can differentiate between, for instance, recombination and a subtree mutation operators. In this work, we argue that CGP's mutation operator can be configured better when isolating the different effects it has in a CGP graph and parametrizing these components independently, as it is done in GP.

We propose the decomposition of CGP's mutation operator into three different components. The mutation of function genes of active nodes can be seen as the *node function mutation* operator. The mutation of connection genes of active nodes is from the graph topology perspective very similar to GP's recombination applied on two identical individuals. We define, therefore, the mutation of active connection genes as the *self-recombination operator*. Finally, the mutation of inactive genes can be seen as a *randomization operation*, because CGP's selection scheme emphasizes off-spring individuals with mutated inactive genes over the parent with the same fitness. Mutated inactive genes experience, therefore, no evolutionary selection pressure and may proceed to the next generations as well as accumulate in a genotype over time.

The focus of the second experiment is to gain insight to which degree the respective components of the mutation operator contribute to the search performance of CGP.

C. Parametrization of CGP's Levels Back Parameter

Goldman and Punch have observed that the distribution of active genes is uneven in CGP and that active genes tend to locate more likely towards the beginning of a genotype. The consequence of an unequal active gene distribution are shrinking genotype regions of inactive, i.e., randomized genes between active genes. A smaller amount of randomized genetic material hampers the effectiveness of, for instance, the self-recombination operator. A more even distribution of active genes among the genotype would induce more significant

TABLE I: List of benchmarks and CGP parameters according to [10], if not otherwise stated.

Parameter	Value
n_r	1
n_c, Boolean benchmarks	100
n_c, symb. regression bench.	10
n_a	2
F, Boolean benchmarks	AND, NAND, OR, NOR
F, symb. regression bench.	+, -, *, /, sin, cos, e^x, $ln(\|x\|)$
l	∞
no. generations,	10.000.000
mut. rate, symb. regression	20%
no. repetitions per benchmark	100
μ, λ, Boolean benchmarks	$\{1, 2, 4, 8, 16, 32\}$
μ, λ, symb. regression bench.	$\{1, 2, 4, 8, 16, 32, 64, 128\}$
Boolean benchmarks	add2...add4, mult2, mult3, epar4...epar8, mux6, mux11, 4cmp
symb. regression bench.	Koza-1...Koza-3, Nguyen-4...Nguyen-6

TABLE II: List of symbolic regression benchmarks.

Benchmark	**Objective Function**	**Vars**	**Training Set**
Koza-1	$x^4 + x^3 + x^2 + x$	1	U[-1,1,20]
Koza-2	$x^5 - 2x^3 + x$	1	U[-1,1,20]
Koza-3	$x^6 - 2x^4 + x^2$	1	U[-1,1,20]
Nguyen-4	$x^6 + x^5 + x^4 + x^3 + x^2 + x$	1	U[-1,1,20]
Nguyen-5	$\sin(x^2)\cos(x) - 1$	1	U[-1,1,20]
Nguyen-6	$\sin(x) + \sin(x + x^2)$	1	U[-1,1,20]

regions of randomized genes between active genes and facilitate the self-recombination operator. There is already a built-in mechanism of CGP that can be used to enforce a better distribution of active genes: the levels back parameter l. Setting l to a fraction of the genotype length n_c would pull active genes apart and let self-recombination explore the search space more freely.

The goal of the third experiment is to investigate, whether the standard way of configuring $l = n_c$ is appropriate and if smaller values for l would result in better convergence.

V. Experiments

This section starts with the search for the best μ and λ parameters for CGP's selection scheme. Using μ and λ that generalize best, CGP's original mutation operator is then decomposed, and the impacts of its principal components (node function mutation, self-recombination, and randomization operators) are measured. Finally, the behavior of the levels back parameter is investigated. The section starts with the description of the experimental setup.

A. Setup

We have configured the experiments according to original findings of Miller [10] and to be consistent with related work (cf. Tab. I and Tab. II). The benchmarks subdivide into Boolean and symbolic regression functions. We have selected popular benchmarks to make the results of this paper comparable to the findings of other authors.

The Boolean benchmarks include the n-adder (2n+1 inputs, n+1 outputs), n-multiplier (2n inputs, 2n outputs), n-even-parity (n inputs, 1 output), 6-multiplexer (2+4 inputs, 1 output), 11-multiplexer (3+8 inputs, 1 output), and 4-comparator (4 inputs, 18 outputs) circuits. The last benchmark computes for all 2-bit input tuples the "<", "==", and ">" relations. The evolution of a Boolean circuit is considered as completed if CGP can evolve a correct solution. Otherwise, ∞ is logged as the required number of fitness evaluations of the current CGP run.

Symbolic regression benchmarks have been selected and configured according to [24]. The functions are shown in Tab. II. A training data set U$[a, b, c]$ in Tab. II refers to c uniform random samples drawn from $[a, b]$. The cost function is defined by the sum of the absolute differences between the real function values and the values of an evaluated individual. When the cost function computes a value lower than 0.01, the algorithm is classified as converged.

The benchmarks are compared using the median number of fitness evaluations and Koza's Computational Effort metric at $z = 99\%$ [23]. To classify the significance of our results, we use the Mann-Whitney U test at the significance level of $1 - \alpha = 95\%$.

TABLE III: Which $(\mu + \lambda)$ is best? Configuration and the median number of fitness evaluations of the best and the $(1 + 4)$ selection schemes. For all Boolean benchmarks, there is a significant difference under the Mann-Whitney U test at $\alpha = 0.05$ between the two best and all remaining $(\mu + \lambda)$ configurations. The best $(\mu+\lambda)$ configurations are $(1+1)$ and $(1+2)$. There is no significant difference between them, except for epar5. For all symbolic regression benchmarks, there is **no** significant difference under the Mann-Whitney U test at $\alpha = 0.05$ between the best and the $(1 + 4)$ selection schemes.

	best	conf.	(1+4)	diff.
add2	36.824	$(1 + 2)$	45.970	24,8%
add3	209.184	$(1 + 1)$	257.078	22,9%
add4	848.098	$(1 + 1)$	1.009.954	19,1%
mult2	5.074	$(1 + 1)$	7.366	45,2%
mult3	773.675	$(1 + 1)$	836.214	8,1%
epar4	8.276	$(1 + 1)$	10.426	26,0%
epar5	30.517	$(1 + 1)$	38.810	27,2%
epar6	103.163	$(1 + 1)$	141.462	37,1%
epar7	319.846	$(1 + 2)$	391.850	22,5%
epar8	861.860	$(1 + 1)$	1.051.970	22,1%
mux6	5.150	$(1 + 2)$	6.490	26,0%
mux11	135.669	$(1 + 1)$	164.610	21,3%
4cmp	44.079	$(1 + 1)$	54.066	22,7%
koza1	38.257	$(128 + 8)$	48.530	26,8%
koza2	150.138	$(1 + 8)$	171.094	13,9%
koza2	131.395	$(1 + 1)$	140.434	6,8%
nguyen4	2.343.344	$(128 + 1)$	7.965.366	239,9%
nguyen5	2.384.210	$(1 + 8)$	2.444.646	2,5%
nguyen6	389.434	$(1 + 1)$	573.234	47,2%

B. Finding a Good $(\mu + \lambda)$ Selection Scheme

Tab. III and IV summarize results of the grid search experiments. The tables compare the best and the $(1 + 4)$ selection schemes regarding the median number of fitness evaluations and CE.

The general trend for Boolean benchmarks is that if a selection scheme is configured with small μ's and λ's, it tends to perform better. The conventional $(1+4)$ selection scheme is always inferior to the $(1+1)$ and the $(1+2)$ selection schemes,

TABLE IV: Which $(\mu + \lambda)$ is best? Configuration and the Computational Effort at $z = 99\%$ of the best and the $(1 + 4)$ selection schemes.

	best	conf.	(1+4)	diff.
add2	193.305	$(1+1)$	235.083	21,6%
add3	978.322	$(1+1)$	1.195.924	22,2%
add4	4.058.649	$(1+1)$	5.105.176	25,8%
mult2	30.596	$(1+1)$	44.094	44,1%
mult3	2.751.027	$(1+1)$	3.094.606	12,5%
epar4	39.617	$(1+1)$	50.893	28,5%
epar5	152.206	$(1+1)$	184.074	20,9%
epar6	527.471	$(1+1)$	635.396	20,5%
epar7	1.416.719	$(1+2)$	1.883.893	33,0%
epar8	4.413.641	$(1+2)$	5.661.376	28,3%
mux6	24.661	$(1+1)$	31.130	26,2%
mux11	466.200	$(1+1)$	594.617	27,5%
4cmp	109.395	$(1+1)$	157.653	44,1%
koza1	112.357	$(1+1)$	124.704	11,0%
koza2	464.443	$(1+4)$	464.443	0,0%
koza2	620.968	$(1+1)$	716.012	15,3%
nguyen4	1.845.381	$(1+1)$	2.796.467	51,5%
nguyen5	10.903.192	$(1+16)$	15.135.145	38,8%
nguyen6	647.106	$(1+2)$	828.551	28,0%

which in return perform best among all configurations, regarding the median number of fitness evaluations (8%-45%) and CE (20%-44%). The difference between the two best selection schemes and $(1+4)$ is also always statistically significant. The $(1+1)$ selection strategy generalizes best among all Boolean benchmarks and will be used in the remaining experiments of this paper.

In preliminary experiments, we have tested whether Goldman and Punch's *single-mutation* outperforms CGP's original mutation operator. While this is always the case for Boolean benchmarks, it has also been observed by other authors [12]. For symbolic regression benchmarks *single-mutation* underperforms. We, therefore, rely on CGP's original mutation operator with a mutation rate of 20% for the symbolic regression benchmarks.

For symbolic regression benchmarks, the $(1 + 4)$ selection scheme is inferior to the best selection strategy except for the CE of the Koza-2 benchmark. We could observe, however, no statistically significant difference between $(1 + 4)$ the best configurations, except for Nguyen-6. The search for a well-generalizable selection scheme configuration is more difficult in this case. There is no clear trend as being observed in the Boolean experiments. For Koza-2, Koza-3, Nguyen-5, and Nguyen-6, $\mu = 1$ results in good performances. For Koza-1 and Nguyen-4, $\mu = 1, 2, 64, 128$ performs best. After an analysis of which configuration is as close to the best configuration as possible over all benchmarks, we identified the $(1+1)$ selection scheme as the best candidate. Except for the Nguyen-4 benchmark, where $(1 + 1)$ excel for the CE, $(1+1)$ is also statistically not significantly different from the best configuration. We, therefore, configure CGP's selection strategy as $(128+1)$ for the Nguyen-4 and as $(1+1)$ for the remaining symbolic regression benchmarks for the rest of this paper.

C. Investigating Success Rates of CGP's Evolutionary Operators

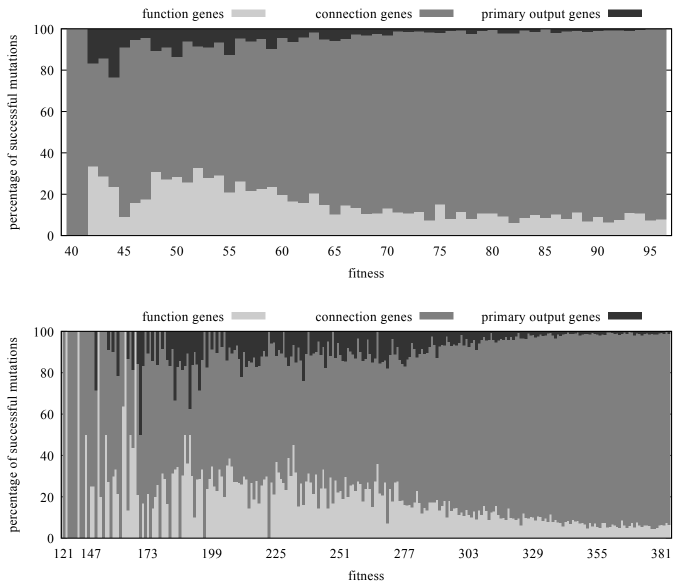

Fig. 2: Which mutation type leads to most fitness improvements? The diagrams plot for the 2-adder (top) and 3-multiplier (bottom) benchmarks the relative fitness improvement rates (y-axes) at different fitness levels (x-axis) for the mutation of node functions, the incoming wires of inner nodes, and the incoming wires of primary outputs. The illustrated behavior can be observed in slight variations for all benchmarks.

When decomposing CGP's original mutation into node function, self-recombination, and randomization operators, one of the first questions is to what extent do these operators contribute to the convergence of CGP. In the first experiment, we log for CGP's original mutation operator, if a mutation of a function or a connection gene results in improvements of the goal function. Fig. 2 shows for 2-adder and 3-multiplier benchmarks and for the different fitness levels, the relative fitness improvement rates for the mutation of function, and connection genes. It can be observed that the functional quality is improved predominantly by mutating the connection genes of inner nodes and primary outputs. Very similar observations can be made for all remaining benchmarks. Switching off the mutation of connection genes stops the convergence of CGP for Boolean and the symbolic regression benchmark completely.

D. The Impact of the Node Function Mutation Operator

With the insight from Fig. 2, one of the next questions is: If mutation of connection genes contributes more frequently to the improvement of the goal function, would CGP's convergence improve further by switching off the mutation of function genes? Tab. V and VI show the development of the median number of fitness evaluations and the CE, when switching off the mutation of function genes ("no node function mutation" column). The left column ("mutation") of Tab. V and VI presents the reference performance of CGP configured according to the findings in the previous Section. Switching off the mutation of function genes improves CGP's median performance for Boolean and decreases CGP's median performance for symbolic regression benchmarks. The CE follows the trend, although with a few small exceptions (mux11, 4cmp, Koza-1, and Nguyen-4).

TABLE V: Comparison of median number of fitness evaluations when using single-mutation (Boolean benchmarks) / regular CGP mutation (symbolic regression benchmarks), switching off the mutation of function genes ("no node function mutation" column), randomization of inactive genes ("rand. operator" column), and switching off function gene mutation as well as switching on inactive gene randomization (two rightmost columns).

	mutation	no node function mutation	diff.	rand. operator	diff.	rand. operator +no node function mutation	diff.
add2	36.824	30.383	-17,49%	32.808	-10,91%	28.637	-22,23%
add3	209.184	189.930	-9,20%	189.834	-9,25%	169.400	-19,02%
add4	848.098	775.500	-8,56%	835.130	-1,53%	736.587	-13,15%
mult2	5.074	4.252	-16,20%	4.817	-5,07%	4.260	-16,04%
mult3	773.675	615.263	-20,48%	747.698	-3,36%	557.560	-27,93%
epar4	8.276	6.431	-22,29%	7.593	-8,25%	6.449	-22,08%
epar5	30.517	26.730	-12,41%	29.630	-2,91%	25.815	-15,41%
epar6	103.163	91.001	-11,79%	103.367	0,20%	84.402	-18,19%
epar7	319.846	284.569	-11,03%	288.625	-9,76%	293.132	-8,35%
epar8	861.860	761.476	-11,65%	828.991	-3,81%	715.667	-16,96%
mux6	5.150	3.771	-26,78%	4.737	-8,02%	3.699	-28,17%
mux11	135.669	113.440	-16,38%	125.387	-7,58%	105.657	-22,12%
4cmp	44.079	38.785	-12,01%	41.230	-6,46%	36.071	-18,17%
koza1	45.911	73.718	60,6%	31.033	-32,4%	68.302	48,8%
koza2	156.415	1.915.520	1124,6%	165.427	5,8%	1.538.417	883,5%
koza3	131.395	628.200	378,1%	187.617	42,8%	471.458	258,8%
nguyen4	2.343.344	4.756.925	103,0%	3.410.237	45,5%	3.207.054	36,9%
nguyen5	2.673.635	∞	∞	3.107.270	16,2%	∞	∞
nguyen6	389.434	4.331.564	1012,3%	369.065	-5,2%	3.252.557	735,2%

TABLE VI: Comparison of the CE when using single-mutation (Boolean benchmarks) / regular CGP mutation (symbolic regression benchmarks), switching off the mutation of function genes ("no node function mutation" column), randomization of inactive genes ("rand. operator" column), and switching off function gene mutation as well as switching on inactive gene randomization (two rightmost columns).

	mutation	no node function mutation	diff.	rand. operator	diff.	rand. operator +no node function mutation	diff.
add2	193.305	155.999	-19,30%	148.319	-23,27%	146.688	-24,12%
add3	978.322	939.585	-3,96%	948.170	-3,08%	767.922	-21,51%
add4	4.058.649	3.512.000	-13,47%	4.297.296	5,88%	3.651.637	-10,03%
mult2	30.596	25.409	-16,95%	27.323	-10,70%	25.386	-17,03%
mult3	2.751.027	2.419.276	-12,06%	2.757.074	0,22%	1.922.387	-30,12%
epar4	39.617	32.421	-18,16%	33.737	-14,84%	30.665	-22,60%
epar5	152.206	117.219	-22,99%	122.577	-19,47%	96.551	-36,57%
epar6	527.471	430.443	-18,39%	482.472	-8,53%	435.160	-17,50%
epar7	1.416.719	1.327.559	-6,29%	1.169.705	-17,44%	1.356.460	-4,25%
epar8	4.413.641	3.852.772	-12,71%	4.085.368	-7,44%	3.733.203	-15,42%
mux6	24.661	17.871	-27,53%	23.866	-3,22%	15.479	-37,23%
mux11	466.200	481.671	3,32%	431.743	-7,39%	391.074	-16,11%
4cmp	109.395	117.333	7,26%	118.856	8,65%	110.355	0,88%
koza1	112.357	109.524	-2,5%	104.420	-7,1%	79.398	-29,3%
koza2	501.239	1.192.581	137,9%	467.027	-6,8%	1.651.625	229,5%
koza3	620.968	1.716.582	176,4%	632.260	1,8%	800.486	28,9%
nguyen4	4.485.606	4.323.409	-3,6%	4.406.429	-1,8%	4.699.820	4,8%
nguyen5	15.270.132	26.922.151	76,3%	16.740.551	9,6%	68.120.064	346,1%
nguyen6	722.230	1.862.043	157,8%	634.519	-12,1%	539.234	-25,3%

E. Randomization of Inactive Genes

The role of CGP's randomization operator has been already extensively studied in literature [20], [25], [26], [27], [28]. However, the works usually concentrate on figuring out what happens if the mutation of inactive genes is switched off. Given the results of Goldman and Punch regarding the uneven distribution of active genes, higher mutation rate could facilitate the effectiveness of CGP's self-recombination operator.

The randomization operator mutates all inactive genes in an individual at the end of a generation. Redundant mutations of already mutated genes can be avoided by mutating only those genes that have experienced a transition from an active to an inactive state, i.e., have been rendered by the evolutionary process as not beneficial for the functional quality.

Tab. V and VI shows that indeed, randomizing inactive genes improves CGP's performance for most Boolean and symbolic regression benchmarks regarding the CE as well as for most Boolean benchmarks regarding the median number of fitness evaluations. The improvements are, however, not as pronounced as switching off the node function mutation operator. The median number of fitness evaluations cannot be improved for the 6-parity benchmark (0.2%), Koza-2 (5.8%), Koza-3 (42.8%), Nguyen-4 (45.5%), and Nguyen-5 (16.2%). Regarding the CE metric, 4-adder (5.88%), 3-multiplier (0.22%), 4-cmp (8.65%), Koza-3 (1.8%), and Nguyen-5 (9.6%) benchmarks witness a degradation.

The last two columns of Tab. V and VI show the results when skipping the node function mutation and enabling the randomization operators. The cumulative results are often superior to the improvements of principle operators for the Boolean benchmarks. Only for the 7-parity benchmark, the combined improvement regarding the median number of fitness evaluations (-8.35%) lies below the improvement of skipping the function gene mutations (-11.03%). Same holds for CE and the 4-adder benchmark (-10.03% vs. -13.47%). The combined improvements usually follow the improvements when skipping mutation of function genes. Notable exceptions are the CE for 3-multiplier, 5-parity, and the multiplexer benchmarks, where the combined improvement is more than 10% better than the improvements of the principal operators.

For symbolic regression benchmarks, switching off node function mutation is usually better than switching off node function and switching on the randomization operators. Only for the Koza-1 (-29.3%) and Nguyen-6 (-25.3%) benchmarks, the CE can be improved in comparison to the CE of the principal operators (Koza-1: -2.5% and -7.1%, Nguyen-6: 157.8% and -12.1%).

F. Exploring Levels Back Parameter

The question of this Section is: What are the excellent regions for the levels back parameter l and will distributing active genes more evenly in the genotype by setting l to a fraction of n_c improve self-recombination's efficiency?

For Boolean benchmarks we are using the standard CGP parameters showed in Tab. I, Goldman and Punch's single-mutation operator, no node function mutation operator, and the randomization operator. We conduct experiments for every $l = 1, 2 \ldots 100$. The top line in Fig. 3 shows for the adder benchmark the median number of fitness evaluations

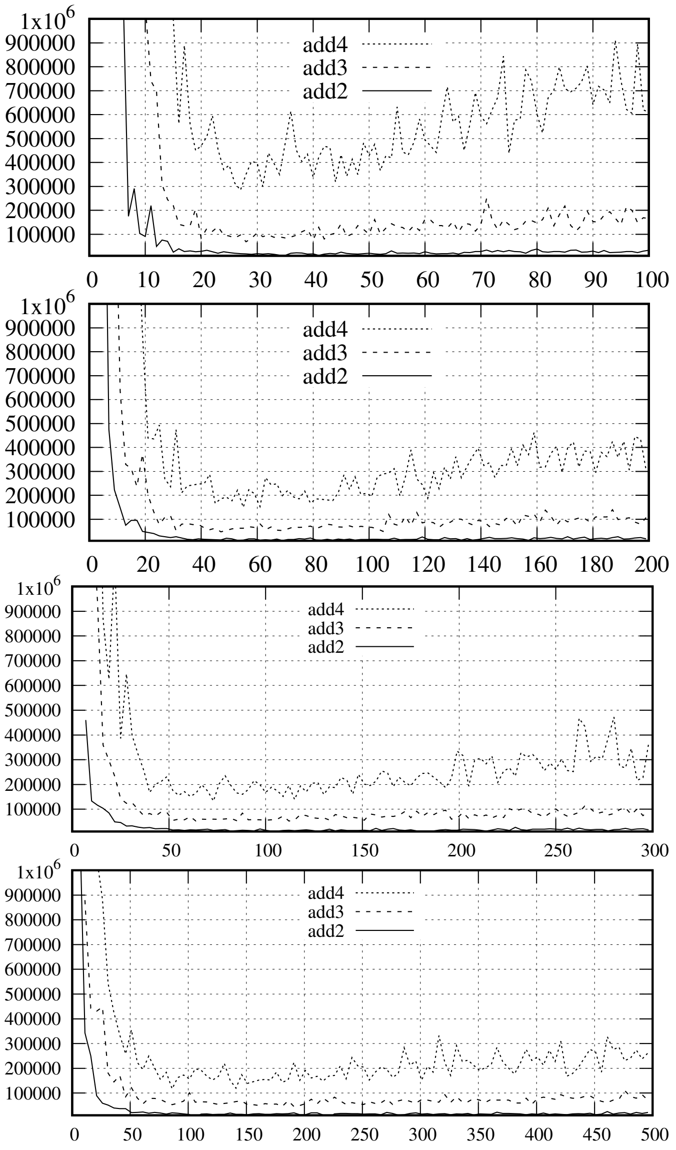

Fig. 3: Which levels back parameter is best? Increasing $l = 1, 2 \dots n_c$ (x-axis) for $n_c = 100, 200, 300, 500$ for the adder benchmarks. y-axis encodes the median number of fitness evaluations. Similar behavior can be observed for all Boolean benchmarks that are investigated in this paper.

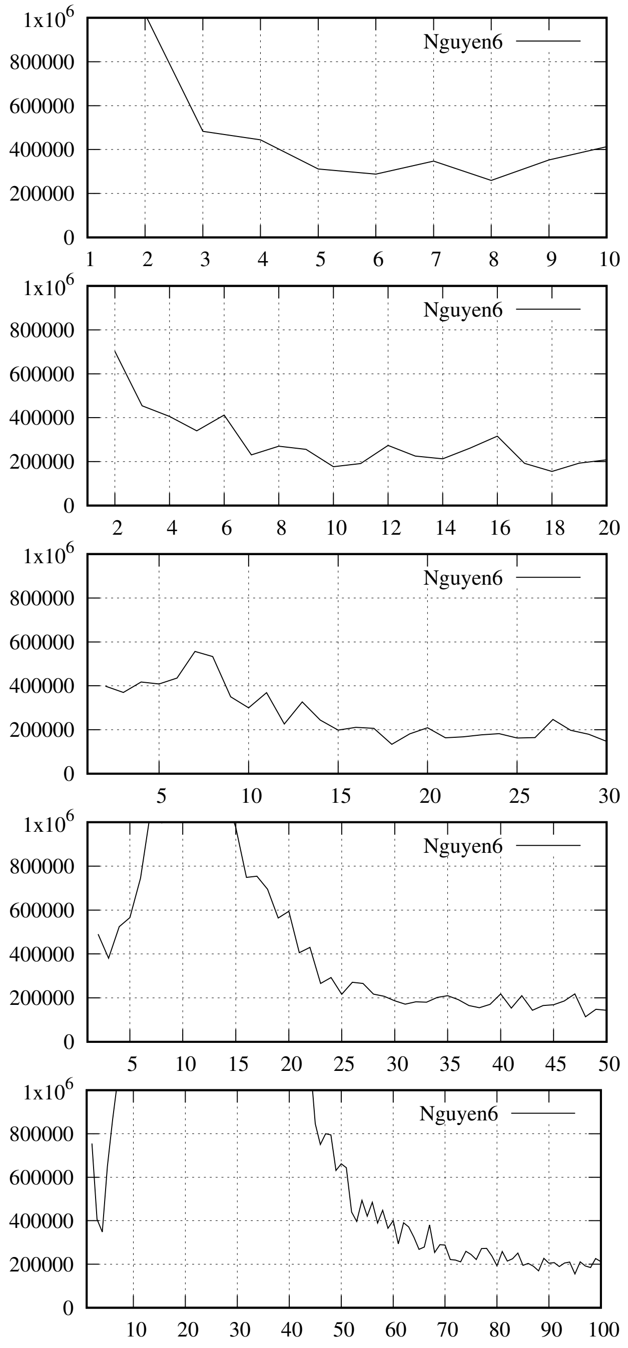

Fig. 4: Which levels back parameter is best? Increasing $l = 10, 20, 30, 50$ for the Nguyen6 benchmark.

for solving the goal function. The first observation is that for small goal functions such as the 2-adder, the reduction of l has no substantial effect. With the increasing complexity of the goal function, however, reducing l to 25 to 40 will improve CGP's converges dramatically. Setting l to a fraction of n_c shows that a more even distribution of active genes helps to improve the effectiveness of CGP's self-recombination operator. This behavior can be observed for all Boolean benchmarks investigated in this paper.

Another question is: Is there a relation between the genotype length n_c and the levels back parameter l? Repeating the experiments of the previous paragraph for $n_c = 200, 300, 500$ and $l = 1, 2, \dots n_c$ shows in the second to fourth rows of Fig. 3, how with rising n_c the sensitivity for a properly selected levels back parameter drops. This means that the self-recombination operator becomes effective for larger n_c's despite the clustered active genes at the beginning of the genotype as a consequence of $l \to n_c$. All other Boolean benchmarks exhibit similar behavior. The conclusion of the development of the convergence rate for $n_c = 100 \to 500$ is that given a sufficiently long CGP genotype, there are enough random genes even between clustered active genes at the beginning of a genotype to render the self-recombination operator effective.

The effects of scaling l and n_c for the Nguyen-6 benchmark can be seen in Fig. 4. Remaining symbolic regression benchmarks behave similarly. The first observation is that rising genotype lengths n_c may improve CGP's convergence, but not to the extent as it happened for Boolean benchmarks. Interestingly, CGP's efficiency drops for smaller l values with rising n_c. Setting l close to n_c is, therefore, more critical for symbolic regression benchmarks.

VI. Conclusion

This work investigates the question, whether CGP's performance can be improved by taking recent developments of Goldman and Punch [15] and Kaufman and Kalkreuth [14] into account. We show that the *single-mutation* operator can have a pronounced impact on CGP's selection scheme. Boolean functions can be evolved significantly faster when utilizing the $(1+1)$ EA. Decomposing CGP's mutation operator into node function, self-recombination, and randomization operators and enabling the most efficient of them, the randomization and self-recombination operators, can further increase CGP's convergence. This shows that for Boolean benchmarks random search is one of the major mechanisms of CGP. The performance of CGP for symbolic regression benchmarks, however, cannot be improved consistently with these measures.

Common to both benchmarks is that more extensive genotype lengths n_c in the single-line CGP model allow for faster convergences and that CGP is more sensitive to decreasing values of the levels back parameter l.

References

[1] R. Forsyth, "Beagle a darwian approach to pattern recognition," *Kybernetes*, vol. 10, no. 3, pp. 159–166, 1981.

[2] N. L. Cramer, "A representation for the adaptive generation of simple sequential programs," in *Proceedings of the 1st International Conference on Genetic Algorithms*. Hillsdale, NJ, USA: L. Erlbaum Associates Inc., 1985, pp. 183–187.

[3] J. Hicklin, "Application of the genetic algorithm to automatic program generation," Master's thesis, 1986.

[4] J. Koza, "Genetic Programming: A paradigm for genetically breeding populations of computer programs to solve problems," Dept. of Computer Science, Stanford University, Technical Report STAN-CS-90-1314, Jun. 1990.

[5] J. F. Miller, P. Thomson, T. Fogarty, and I. Ntroduction, "Designing Electronic Circuits Using Evolutionary Algorithms. Arithmetic Circuits: A Case Study," 1997.

[6] P. Kaufmann, K. Glette, T. Gruber, M. Platzner, J. Torresen, and B. Sick, "Classification of Electromyographic Signals: Comparing Evolvable Hardware to Conventional Classifiers," *IEEE Trans. Evolutionary Computation*, vol. 17, no. 1, pp. 46–63, 2013.

[7] Z. Vasícek and L. Sekanina, "Evolutionary Design of Complex Approximate Combinational Circuits," *Genetic Programming and Evolvable Machines*, vol. 17, no. 2, pp. 169–192, 2016. [Online]. Available: http://dx.doi.org/10.1007/s10710-015-9257-1

[8] L. Sekanina, "Image Filter Design with Evolvable Hardware," in *Proceedings of the Applications of Evolutionary Computing on EvoWorkshops 2002*. London, UK, UK: Springer-Verlag, 2002, pp. 255–266.

[9] M. M. Khan, G. M. Khan, and J. F. Miller, "Evolution of neural networks using cartesian genetic programming," in *IEEE Congress on Evolutionary Computation (CEC)*. IEEE, 2010, pp. 1–8.

[10] J. F. Miller, "An Empirical Study of the Efficiency of Learning Boolean Functions Using a Cartesian Genetic Programming Approach," in *Genetic and Evolutionary Computation (GECCO)*, vol. 2. Morgan Kaufmann, 1999, pp. 1135–1142.

[11] B. W. Goldman and W. F. Punch, "Length bias and search limitations in cartesian genetic programming," in *Proceedings of the 15th Annual Conference on Genetic and Evolutionary Computation*, ser. GECCO '13. ACM, 2013, pp. 933–940.

[12] ——, "Reducing wasted evaluations in cartesian genetic programming," in *Genetic Programming*. Springer Berlin Heidelberg, 2013, pp. 61–72.

[13] P. Kaufmann and R. Kalkreuth, "Parametrizing Cartesian Genetic Programming: An Empirical Study," in *KI 2017: Advances in Artificial Intelligence: 40th Annual German Conference on AI*. Springer International Publishing, 2017.

[14] ——, "An Empirical Study on the Parametrization of Cartesian Genetic Programming," in *Genetic and Evolutionary Computation (GECCO)*. (Compendium). ACM, 2017.

[15] B. W. Goldman and W. F. Punch, "Analysis of cartesian genetic programming's evolutionary mechanisms," *IEEE Transactions on Evolutionary Computation*, vol. 19, no. 3, pp. 359–373, 2015.

[16] S. Yazdani and J. Shanbehzadeh, "Balanced cartesian genetic programming via migration and opposition-based learning: Application to symbolic regression," *Genetic Programming and Evolvable Machines*, vol. 16, no. 2, pp. 133–150, 2015.

[17] A. J. Turner and J. F. Miller, "Recurrent Cartesian Genetic Programming," in *International Conference on Parallel Problem Solving from Nature (PPSN)*. Springer International Publishing, 2014, pp. 476–486.

[18] D. Izzo, F. Biscani, and A. Mereta, "Differentiable genetic programming," Arxiv Preprint arxiv:1611.04766, 2016.

[19] J. F. Miller, "Cartesian genetic programming: its status and future," *Genetic Programming and Evolvable Machines*, pp. 1 – 40, 2019.

[20] A. J. Turner and J. F. Miller, "Neutral genetic drift: an investigation using cartesian genetic programming," *Genetic Programming and Evolvable Machines*, vol. 16, no. 4, pp. 531–558, 2015. [Online]. Available: https://doi.org/10.1007/s10710-015-9244-6

[21] M. López-Ibáñez, J. Dubois-Lacoste, L. P. Cáceres, M. Birattari, and T. Stützle, "The irace package: Iterated racing for automatic algorithm configuration," *Operations Research Perspectives*, vol. 3, pp. 43 – 58, 2016.

[22] J. Miller and P. Thomson, "Cartesian Genetic Programming," in *European Conf. on Genetic Programming (EuroGP)*. Springer, 2000, pp. 121–132.

[23] J. R. Koza, "Evolving a Computer Program to Generate Random Numbers Using the Genetic Programming Paradigm," in *Proc. Intl. Conf. on Genetic Algorithms*. Morgan Kaufmann, 1991, pp. 37–44.

[24] J. McDermott, D. R. White, S. Luke, L. Manzoni, M. Castelli, L. Vanneschi, W. Jaskowski, K. Krawiec, R. Harper, K. De Jong, and U.-M. O'Reilly, "Genetic Programming Needs Better Benchmarks," in *Genetic and Evolutionary Computation (GECCO)*. ACM, 2012, pp. 791–798. [Online]. Available: http://doi.acm.org/10.1145/2330163.2330273

[25] R. M. Downing, "On Population Size and Neutrality: Facilitating the Evolution of Evolvability," in *EuroGP*. Berlin, Heidelberg: Springer-Verlag, 2007, pp. 181–192. [Online]. Available: http://dl.acm.org/citation.cfm?id=1763756.1763774

[26] T. Yu and J. F. Miller, "Through the interaction of neutral and adaptive mutations, evolutionary search finds a way," *Artificial Life*, vol. 12, no. 4, pp. 525–551, 2006.

[27] T. Yu and J. Miller, "Neutrality and the Evolvability of Boolean Function Landscape," in *European Conf. on Genetic Programming (EuroGP)*, ser. LNCS, vol. 2038. Springer, 2001, pp. 204–217.

[28] V. K. Vassilev and J. F. Miller, "The advantages of landscape neutrality in digital circuit evolution," in *ICES*, ser. LNCS, vol. 1801. Springer, 2000, pp. 252–263.