Towards Discrete Phenotypic Recombination in Cartesian Genetic Programming

Roman Kalkreuth(✉)

Computational Intelligence Research Group, Chair XI Algorithm Engineering,
Department of Computer Science, TU Dortmund University,
Dortmund, North Rhine-Westphalia, Germany
roman.kalkreuth@tu-dortmund.de
https://ls11-www.cs.tu-dortmund.de/

Abstract. The tree-based representation model of Genetic Programming (GP) is largely used with subtree crossover for genetic variation. Unlike Cartesian Genetic Programming (CGP) which is commonly used merely with mutation. Compared to comprehensive knowledge about recombination in the field of tree-based GP, the state of knowledge in CGP appears to be comparatively poor. Even if CGP was officially introduced over twenty years ago, the role of recombination in CGP has been recently considered an open issue. Several promising steps have been taken in recent years, but more research is needed to develop towards a more comprehensive and holistic perspective on crossover in CGP. In this work, we propose a phenotypic variation method for discrete recombination in CGP. We compare our method to the traditional mutation-only CGP approach on a set of well-known symbolic regression problems. The initial results presented in this work demonstrate that the use of our proposed discrete recombination method performs significantly better than the traditional mutation-only approach.

Keywords: Cartesian Genetic Programming · Crossover · Phenotypic variation

1 Introduction

Cartesian Genetic Programming can be considered a well-established graph-based GP variant. Initial work towards CGP was done by Miller, Thompson, Kalganova, and Fogarty [8,14,15] by the introduction of a two-dimensional graph encoding model of functional nodes. CGP can be seen as an extension to the traditional tree-based GP representation model since its representation allows many graph-based applications such as digital circuit design [26], evolution of neural network topologies [16,28] and synthesis of cryptographic Boolean functions [5,7]. CGP has introduced over two decades ago but is still predominantly used only with a probabilistic point mutation operator. The reason for this is that various standard

G. Rudolph et al. (Eds.): PPSN 2022, LNCS 13399, pp. 63–77, 2022.
https://doi.org/10.1007/978-3-031-14721-0_5

genotypic crossover operators failed to improve the search performance of standard CGP in the past [3,15]. Overall, the state of knowledge about recombination in CGP appears to be weak when compared to the number of publications in tree-based GP. The role of recombination in CGP was recently surveyed by Miller [17] and is still considered to be an open issue. Even if some progress has been made in recent years, comprehensive and advanced knowledge about recombination in CGP is still missing [17]. In the field of evolutionary computation (EC), discrete recombination is a well-established form of recombination in various subfields. Discrete recombination typically selects each gene from one of the two parents with equal probability. According to Rudolph [22], this method can be therefore considered as a dynamic n-point crossover since each gene for the chromosome of the offspring is selected from the first or second parent with equal probability. In this work, we take a step forward on the issue of crossover and introduce a method for the adaption of discrete recombination in CGP. We initially evaluate our method on a set of well-known symbolic regression benchmarks. Our results demonstrate the effectiveness of our approach for these problems.

Section 2 of this work describes CGP. Related work on crossover in CGP is surveyed in Sect. 3. This section also gives a brief historical overview of discrete recombination in the field of EC. In Sect. 4, we introduce our new method. Section 5 is devoted to the description of our experiments and the presentation of our results. Our findings are discussed in Sect. 6. Finally, Sect. 7 gives a conclusion and outlines our future work.

2 Cartesian Genetic Programming

In contrast to tree-based GP, CGP represents a genetic program via genotype-phenotype mapping as an indexed, acyclic, and directed graph. In this way, CGP can be seen as an extension of the traditional tree-based GP approach. The CGP representation model is based on a rectangular grid or row of nodes. Each genetic program is encoded in the genotype of an individual and is decoded to its corresponding phenotype. A definition of a cartesian genetic program $\mathcal{P}$ is given in Definition 1. Let $\phi : \mathcal{P} \mapsto \Psi$ be a decode function which maps $\mathcal{P}$ to a phenotype Ψ. Originally, the structure of the graph was represented by a rectangular grid of n_r rows and n_c columns, but later work focused on a representation with one row. The CGP decoding procedure processes groups of genes, and each group refers to a node of the graph, except the last one, which represents the outputs of the phenotype. Each node is represented by two types of genes that index the function number in the GP function set and the node inputs. These nodes are called *function nodes* and execute functions on the input values. The number of input genes depends on the maximum arity n_a of the function set.

Definition 1 (Cartesian Genetic Program). *A cartesian genetic program $\mathcal{P}$ is an element of the Cartesian product $\mathcal{N}_\mathrm{i} \times \mathcal{N}_\mathrm{f} \times \mathcal{N}_\mathrm{o} \times \mathcal{F}$:*

- *$\mathcal{N}_\mathrm{i}$ is a finite non-empty set of input nodes*
- *$\mathcal{N}_\mathrm{f}$ is a finite set of function nodes*

– $\mathcal{N}_o$ *is a finite non-empty set of output nodes*
– $\mathcal{F}$ *is a finite non-empty set of functions*

A backward search is conducted to decode the corresponding phenotype. The decoding itself starts at the output nodes and continues until the inputs nodes are reached. The decoding procedure is done for all output genes. The result of the decoding procedure can be described as a set of directed paths Ω. Given the input set I and the output set O, let $\omega = I \times \Omega \mapsto O$ be an output function. An example of the backward search of the most popular one-row integer representation is illustrated in Fig. 1. The backward search starts from the program output and processes all nodes which are linked in the genotype. In this way, only active nodes are processed during evaluation. The genotype in Fig. 1 is grouped by its function nodes. The first (underlined) gene of each group refers to the function number in the corresponding function set. The non-underlined genes represent the input connections of the node. Inactive function nodes are shown in gray color and with dashed lines.

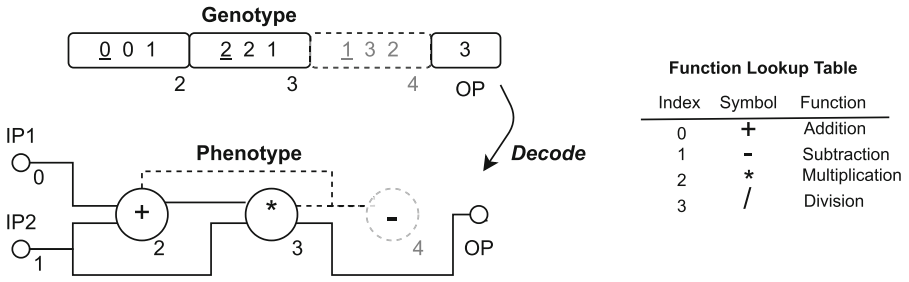

Fig. 1. Example of the decoding procedure of a CGP genotype to its corresponding phenotype. The identifiers IP1 and IP2 stand for the two input nodes with node index 0 and 1. The identifier OP stands for the output node of the graph.

The number of inputs n_i, outputs n_o, and the length of the genotype is fixed. Every candidate program is represented with $n_r * n_c * (n_a + 1) + n_o$ integers. Even if the length of the genotype is fixed for each candidate program, the length of the corresponding phenotype in CGP is variable, which can be considered as an advantage of the CGP representation. CGP is traditionally used with a $(1+\lambda)$ evolutionary algorithm (EA). The $(1+\lambda)$-EA is often used with a selection strategy called *neutrality*, which is based on the idea that genetic drift yields to diverse individuals having equal fitness. The genetic drift is implemented into the selection mechanism in a way that individuals which have the same fitness as the normally selected parent are determined, and one of these same-fitness individuals is returned uniformly at random. The new population in each generation consists of the best individual of the previous population and the λ created offspring. The breeding procedure is mostly done by a point mutation that swaps genes in the genotype of an individual in the valid range by chance.

Another point mutation is the flip of the functional gene, which changes the functional behavior of the corresponding function node.

3 Related Work

3.1 Recombination in CGP

According to the reports of Clegg et al. [3], the first attempts of recombination in standard CGP included testing of various genotypic crossover techniques. For instance, the genetic material was recombined by swapping parts of the genotypes of the parent individuals or randomly exchanging selected nodes. Clegg et al. reported that all techniques failed to improve the convergence of CGP and that merely swapping the integers disrupts the search performance. In comparison to mutation only CGP, the addition of genotypic crossover techniques hindered the performance. In one of the first empirical studies about CGP, Miller [15] analyzed its computational efficiency on Boolean function problems. More precisely, Miller analyzed and studied the influence of population size on the efficiency of CGP. The key finding of his study was that extremely low populations perform most effectively for the tested problems. The experiments of this study also demonstrated that the addition of a genotypic crossover reduces the computational effort only marginally.

This was the motivation for the introduction of a real-valued representation and intermediate recombination for CGP by Clegg et al. The real-valued representation of CGP represents the directed graph as a fixed-length list of real-valued numbers in the interval [0, 1]. The genes are decoded to the integer-based representation by their normalization values (number of functions or maximum input range). The recombination of two CGP genotypes is performed by intermediate recombination with a random weighting factor. Clegg et al. demonstrated that the new representation in combination with crossover improves the convergence behavior of CGP on one of the two tested symbolic regression problems. However, for the later generations, Clegg et al. found that the use of crossover in real-valued CGP disrupts the convergence on one problem. Later work by Turner [30] presented results with intermediate recombination on three additional classes of computational problems, digital circuit synthesis, function optimization, and agent-based wall avoidance. On these problems, it was found that the real-valued representation together with the crossover operation performed worse than mutation-only CGP.

Kalkreuth et al. [10] introduced and investigated subgraph crossover in CGP which exchanges and links subgraphs of active function nodes between two selected parents and the block crossover exchanges blocks of active function genes. In recent comparative studies, its use has been found beneficial for several symbolic regression benchmarks since it led to a significant decrease in the number of fitness evaluations needed to find the ideal solution [9,11]. Contrarily, the gain of the search performance was considerably lower for the tested Boolean function problems [9,11]. Moreover, the results of the experiments clearly showed that the subgraph crossover failed to improve the search performance on some of

the tested Boolean benchmarks when compared to the results of the traditional $1 + \lambda$ selection strategy.

Husa and Kalkreuth [6] proposed block crossover which selects active function nodes by chance in accordance with a predefined block size but without any order. The function genes of the selected active nodes are then swapped. The block crossover has been compared to mutation-only CGP on a suite of Boolean functions and symbolic regression problems. The outcome of the study gave significant evidence that the $(1+\lambda)$-CGP cannot be considered the most efficient CGP algorithm in the Boolean function domain, although it seems to be often a good choice. The outcome of the study gave the first evidence, that it is possible for crossover operators to outperform the standard $1 + \lambda$ selection strategy.

Sivla et al. [27] introduced a form of crossover for multiple output problems. The proposed method combines the subgraphs of the best outputs of the parent individuals produce an offspring. The proposed crossover technique was applied to the synthesis of combinational logic circuits with multiple outputs. The so-called X-CGP obtained the best results when compared to single chromosome CGP representations and performed better than the multi-chromosome representation for some of the tested problems. The experiments of Siliva et al. indicate that the proposed method is promising. On the other hand, the authors concluded that more studies are needed since X-CGP performed no better than the mutation-only multi-chromosome techniques on the majority of the tested problems.

3.2 Historical Background of Discrete Recombination

Discrete recombination in EC was first described by Rechenberg [20,21] for the simulation of the first type of a multimembered evolutionary strategy (ES) called $(\mu + 1)$ or steady-state ES. Rechenberg demonstrated that recombination can improve the speed of the evolutionary process if the measure is taken per generation rather than per function evaluation [2]. Schwefel [23,24] later utilized discrete recombination among five types of recombination for two further versions of the multimembered ES, called $(\mu + \lambda)$- and (μ, λ)-ES [1]. Schwefel [24] performed an empirical study with 50 uni- and multimodal test functions and compared ESs to the most traditional direct optimization strategies and the outcome showed good results for ESs. According to Bäck et al. [1], the best results were achieved with the use of several types of recombination. In the field of GAs, discrete recombination is commonly referred to as *uniform crossover* and has been found to be a useful search operator [4]. Uniform crossover was first proposed for the binary encoding model of GA by Syswerda [29] and its search performance was found superior to the one- and two-point crossover in the most cases. Uniform recombination in GA inspired the adaption in tree-based GP [18,19] where function nodes and subtrees are exchanged between two parent individuals in accordance with a uniform rate. If the uniform rate is set to 50%, this method represents the tree-based GP equivalent of the uniform crossover for binary strings.

4 The Proposed Method

We adapt discrete recombination in CGP by means of phenotypic functional variation which is performed through the exchange of function genes of active function nodes. The phenotype of a CGP individual is represented by its active function nodes which are determined before the crossover procedure. After selecting two individuals, the minimum and a maximum number of active function nodes of the two individuals is determined. The reason for this is that the size of the phenotype in CGP is not fixed and can vary among individuals. To perform the exchange of active function genes, the crossover procedure iterates over the minimum number of active nodes. A binary decision is made by chance in each iteration whether the function genes are swapped or kept. In the case that both phenotypes differ in size, our method performs a special step in the last iteration called *boundary extension* which extends the selection of active function genes. The idea behind this step is to include active function genes of the larger phenotype into the selection which would not be considered if the lists of active function nodes are merely interated in order. Just like the uniform crossover in GA, our method produces two offspring. The algorithmic implementation of our method is described in Algorithm 1. Exemplifications of the procedure on genotypic and phenotypic level are illustrated in Fig. 2 and 3. An implementation for the CGP extension package of the Java Evolutionary Computation Research System (ECJ) [25] is provided in the ECJ GitHub repository[1].

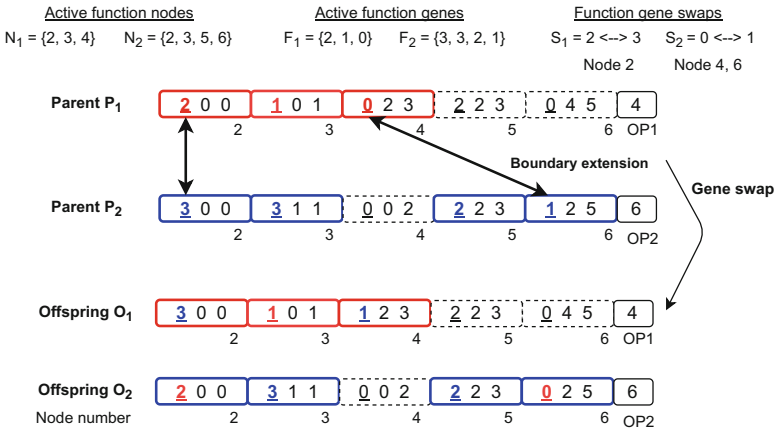

Fig. 2. Exemplification of discrete recombination in CGP: Active function genes of two CGP genotypes are recombined by means of discrete recombination. Function genes, which have been randomly selected for the exchange, are connected with a double-sided arrow in the figure. The active function nodes and genes of the respective parent and offspring individuals are highlighted in red and blue color. (Color figure online)

[1] https://github.com/GMUEClab/ecj.

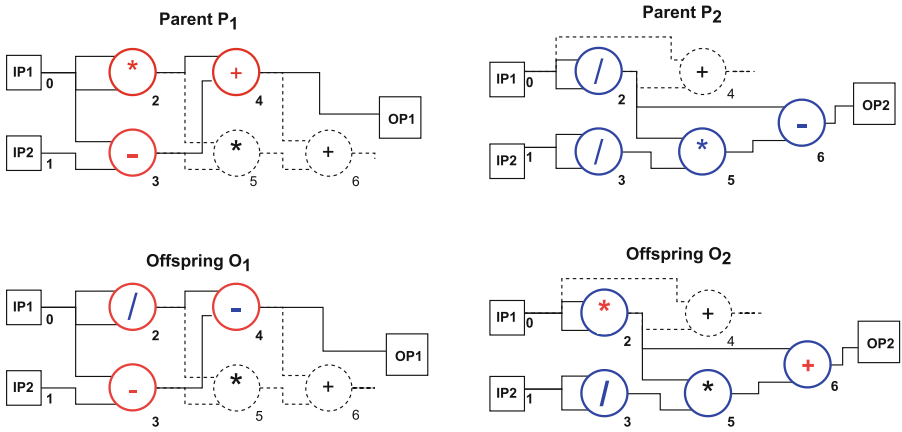

Fig. 3. Illustration of discrete recombination in CGP on the phenotypic level based on the genotypic exemplification presented in Fig. 2. Active function nodes of the respective parent and offspring individuals are highlighted in red and blue color. (Color figure online)

5 Experiments

5.1 Experimental Setup

We performed experiments with symbolic regression problems. We compared the traditional $(1 + \lambda)$-CGP to a canonical EA equipped with our proposed discrete recombination and tournament selection. The algorithms which we used in our experiments are listed in Table 1. To evaluate the search performance, we measured the number of fitness evaluations until the CGP algorithm terminated successfully as recommended by McDermott et al. [13]. Termination was triggered when an ideal solution was found or a predefined budget of fitness evaluation was exceeded. We defined a maximum number of 10^8 fitness evaluations for our experiments and calculated the success rate (SR). In addition to the mean values of the measurements, we also calculated the standard deviation (SD), median (Q2) as well as lower and upper quartile (Q1 and Q3). Meta-optimization experiments have been performed to compare the algorithms fairly and are described in more detail in the following subsection. All tested algorithms were compared on the same number of function nodes to exclude conditions, which can distort the search performance comparison. Our method was tested against the traditional $(1 + \lambda)$-CGP which we declared as the baseline algorithm for our experiments. In our experiments, we exclusively used the single-row standard integer-based representation of CGP. Since we cannot guarantee normally distributed values in our samples, we used the nonparametric two-tailed Mann-Whitney U test to evaluate statistical significance. More precisely, we tested the null hypothesis that two samples come from the same population (i.e. have the same median). We performed 100 independent runs with different random seeds. The levels back parameter l was set to ∞.

Algorithm 1. Discrete phenotypic crossover

Arguments
G_1, G_2: Genomes of the first parent individuals
N_1, N_2: List of active function node numbers of the first parent inidividuals
Return
$\widetilde{G}_1$, $\widetilde{G}_2$: Genomes of the offspring

```
1: function DiscreteCrossover(G1, G2, N1, N2)
2:     l1 ← |N1|                                   ▷ Number of active nodes of the first parent
3:     l2 ← |N2|                                   ▷ Number of active nodes of the second parent
4:     min ← Min(l1, l2)                           ▷ Determine the minimum
5:     max ← Max(l1, l2)                           ▷ Determine the maximum
6:     i ← 0
7:     while i < min do                            ▷ Iterate over the minimum number of active nodes
8:         if RandomBoolean() = true then          ▷ Decision by chance to keep or swap genes
9:             ▷ Check if conditions for boundary extension are satisfied
10:            if i = min − 1 and l1 ≠ l2 then
11:                r ← RandomInteger(0, max - i)   ▷ Determine a random offset
12:                if l1 < l2 then                 ▷ If the first parent has the minimum of active nodes
13:                    n1 ← N1[i]
14:                    ▷ Extend node selection for the second, phenotypically larger, parent
15:                    n2 ← N2[i + r]
16:                else                            ▷ Otherwise, extend the selection for the first parent
17:                    n1 ← N1[i + r]
18:                    n2 ← N2[i]
19:                end if
20:            else                                ▷ Without boundary extension, just select the nodes in order
21:                n1 ← N1[i]
22:                n2 ← N2[i]
23:            end if
24:            p1 ← PositionFromNodeNumber(n1)     ▷ Function gene position of n1
25:            p2 ← PositionFromNodeNumber(n2)     ▷ Function gene position of n2
26:            G̃1, G̃2 ← SwapGenes(G1, G2, p1, p2)  ▷ Swap the function genes
27:        end if
28:        i ← i + 1                               ▷ Loop counter increment
29:    end while
30:    return G̃1, G̃2
31: end function
```

Table 1. Identifiers for the tested CGP algorithms.

Identifier	Description
$1 + \lambda$	$1 + \lambda$ selection strategy with neutral genetic drift
Canonical	Canonical EA with phenotypic uniform crossover and tournament selection

5.2 Benchmarks

We chose eleven symbolic regression problems from the work of McDermott et al. [13] for better GP benchmarks. The reason for our choice of these problems is the fact that we can find an ideal solution more likely on average and evaluate the search performance of the whole evolutionary process. Our set of benchmarks covers uni- as well as bivariate polynomial, trigonometric, logarithmic, and power functions. The functions of the problems are shown in Table 2. A training data set $\mathrm{U}[a, b, c]$ refers to c uniform random samples drawn from a to b inclusive. We used the extended Koza function set as recommended by McDermott et al. The function set is shown in Table 3. The fitness of the individuals was represented by a cost function value. The cost function was defined by the sum of the absolute difference between the real function values and the values of an evaluated individual. Let $T = \left\{x_p\right\}_{p=1}^{\mathcal{P}}$ be a training dataset of $\mathcal{P}$ random points and $f_{\mathrm{ind}}(x_p)$ the value of an evaluated individual and $f_{\mathrm{ref}}(x_p)$ the true function value. Let

$$C := \sum_{p=1}^{\mathcal{P}} |f_{\mathrm{ind}}(x_p) - f_{\mathrm{ref}}(x_p)|$$

be the cost function. When the difference of all absolute values becomes less than 0.01, the algorithm is classified as converged.

5.3 Meta-optimization

We tuned relevant parameters for all tested CGP algorithms on the set of benchmark problems. Moreover, we used the meta-optimization toolkit of ECJ. The parameter space for the respective algorithms, explored by meta-optimization, is presented in Table 4. For the meta-level, we used a canonical GA equipped with intermediate recombination and point mutation. Since GP benchmark problems can be very noisy in terms of finding the ideal solution, we oriented the meta-optimization with a common approach that has been used in previous studies [6,11,12]. The meta-evolution process at the base level was repeated multiple times for each candidate setting and the most effective settings were compared to find the best setting. For the problems Koza 1–3 and Nguyen 4–7, we selected effective settings of certain parameters for the $(1 + \lambda)$-CGP from previous parametrization studies [11,12].

Table 2. List of symbolic regression benchmarks.

Problem	Objective function	Vars	Training set	Function set
Koza-1	$x^4 + x^3 + x^2 + x$	1	U[−1, 1, 20]	Koza
Koza-2	$x^5 - 2x^3 + x$	1	U[−1, 1, 20]	Koza
Koza-3	$x^6 - 2x^4 + x^2$	1	U[−1, 1, 20]	Koza
Nguyen-4	$x^6 + x^5 + x^4 + x^3 + x^2 + x$	1	U[−1, 1, 20]	Koza
Nguyen-5	$\sin(x^2)\cos(x) - 1$	1	U[−1, 1, 20]	Koza
Nguyen-6	$\sin(x) + \sin(x + x^2)$	1	U[−1, 1, 20]	Koza
Nguyen-7	$\ln(x + 1) + \ln(x^2 + 1)$	1	U[0, 2, 20]	Koza
Nguyen-8	$\sqrt{x}$	1	U[0, 4, 20]	Koza
Nguyen-9	$\sin(x^2) + \sin(y^2)$	2	U[0, 2, 20]	Koza
Nguyen-10	$2 * \sin(x) * \cos(x)$	2	U[0, 2, 20]	Koza
Nguyen-11	x^y	2	U[0, 2, 20]	Koza

Table 3. Function set used for the experiments.

Name	Functions	Constants
Koza	$+ \quad - \quad * \quad / \quad \sin \quad \cos \quad e^n \quad \ln(\lvert n \rvert)$	Constant input with a value of 1

Table 4. Parameter space explored by meta-optimization for the $1 + \lambda$ and canonical CGP algorithm.

Algorithm	Parameter	Description	Range
$1 + \lambda$	λ	Number of offspring	[1, 1024]
	N	Number of function nodes	[10, 1000]
	M	Point mutation rate [%]	[1.0, 30.0]
Canonical	N	Number of function nodes	[10, 1000]
	M	Point mutation rate [%]	[1.0, 30.0]
	C	Crossover rate [%]	[10, 100]
	P	Population size	[10, 500]
	T	Tournament size	[2, 20]

5.4 Results

The results of our meta-optimization and search performance evaluation are presented in Table 5 and it is clearly visible that the Canonical-CGP with discrete recombination reduces the number of fitness evaluations to termination significantly on all tested problems. Moreover, on the more complex problems, the Canonical-CGP achieves higher success rates. Violin plots are provided in Fig. 4.

Table 5. Results of the meta-optimization and search performance evaluation.

Problem	Algorithm	Parametrization						Search performance evaluation						p
		N	λ	M [%]	C [%]	P	T	MFE	SD	1Q	2Q	3Q	SR	
Koza-1	$1+\lambda$	10	4	20	–	–	–	3,285,238	8,974,193	518,516	1,408,326	3,460,391	1.0	
	Canonical	10	–	20	70	50	4	**532,957**	**652,332**	**76,868**	**311,983**	**724,563**	**1.0**	10^{-9}
Koza-2	$1+\lambda$	10	4	20	–	–	–	2,325,581	7,830,950	340,608	1,260,496	2,463,527	1.0	
	Canonical	10	–	10	50	50	4	**733,925**	**934,455**	**67,387**	**394,075**	**982,325**	**1.0**	10^{-6}
Koza-3	$1+\lambda$	10	4	20	–	–	–	428,778	663,576	26,527	159,290	502,686	1.0	
	Canonical	10	–	20	50	50	4	**122,629**	**264,791**	**17,113**	**41,282**	**113,324**	**1.0**	10^{-4}
Nguyen-4	$1+\lambda$	100	16	10	–	–	–	91,228,744	24,303,588	100,000,000	100,000,000	100,000,000	0.16	
	Canonical	100	–	8	50	50	4	**59,767,376**	**38,075,889**	**23,060,887**	**59,816,675**	**100,000,000**	**0.62**	10^{-10}
Nguyen-5	$1+\lambda$	60	16	7	–	–	–	64,092,121	42,126,017	14,078,020	96,894,232	100,000,000	0.50	
	Canonical	60	–	7	70	50	4	**9,758,166**	**23,157,856**	**190,312**	**833,400**	**6,072,437**	**0.96**	10^{-13}
Nguyen-6	$1+\lambda$	100	16	10	–	–	–	16,757,903	18,877,924	2,980,764	10,508,376	23,852,124	0.95	
	Canonical	100	–	8	70	50	4	**1,634,090**	**4,399,397**	**21,962**	**132,575**	**888,900**	**1.0**	10^{-15}
Nguyen-7	$1+\lambda$	200	16	7	–	–	–	64,033,983	35,411,800	30,458,912	67,583,400	100,000,000	0.67	
	Canonical	200	–	7	50	50	7	**23,424,276**	**32,155,768**	**2,622,975**	**7,966,750**	**26,935,237**	**0.93**	10^{-13}
Nguyen-8	$1+\lambda$	150	16	15	–	–	–	1,554,341	1,745,877	93,096	911,720	2,386,644	1.0	
	Canonical	150	–	15	50	50	7	**764,404**	**890,860**	**149,262**	**415,800**	**1,028,275**	**1.0**	0.02
Nguyen-9	$1+\lambda$	150	16	15	–	–	–	1,141,109	1,681,517	32,288	560,416	1,572,280	1.0	
	Canonical	150	–	15	50	50	4	**291,008**	**613,343**	**14,350**	**50,975**	**255,450**	**1.0**	10^{-5}
Nguyen-10	$1+\lambda$	60	128	20	–	–	–	905,799	1,659,653	26,144	130,176	1,201,152	1.0	
	Canonical	60	–	15	70	50	4	**139,754**	**178,352**	**22,837**	**76,375**	**185,050**	**1.0**	0.002
Nguyen-11	$1+\lambda$	50	64	10	–	–	–	155,608	165,428	14,944	111,488	224,784	1.0	
	Canonical	50	–	10	70	50	4	**56,685**	**65,100**	**111,62**	**37,175**	**75,225**	**1.0**	10^{-4}

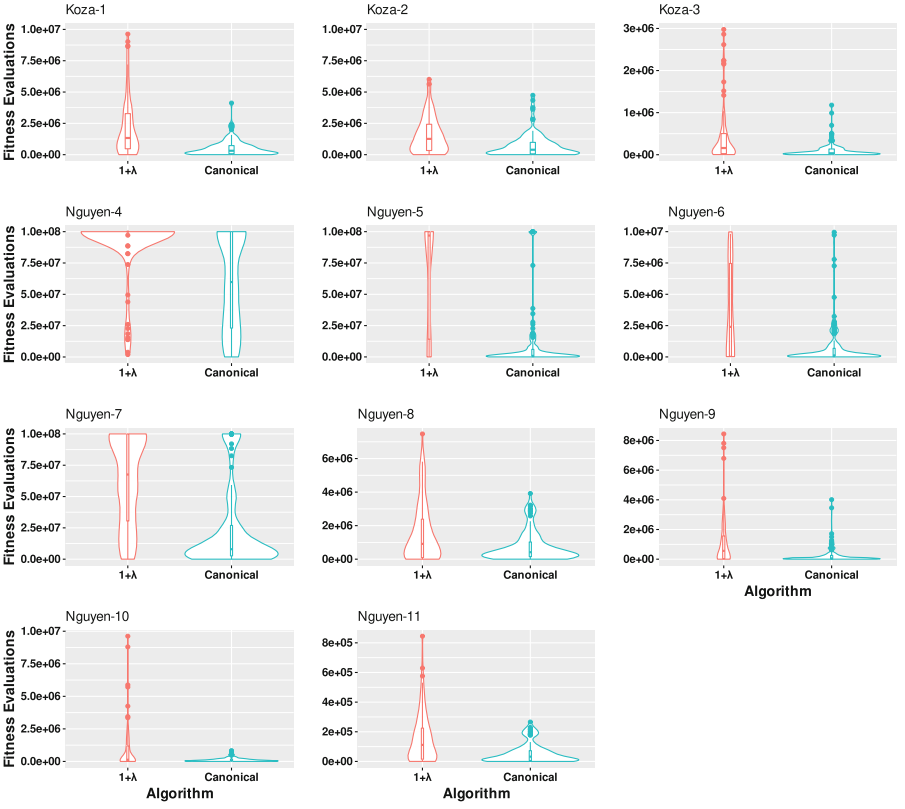

Fig. 4. Violin plots for all tested problems and algorithms of our experiments.

6 Discussion

The experiments presented in this work allow certain points that are worthy of discussion. Even if the initial results of our proposed method are promising we have to emphasize that more experiments are needed to achieve insight into how our method performs in other problem domains. Since former work [10] on recombination in CGP presented promising results with symbolic regression problems, we initially tested our proposed method in this problem domain. However, we have to evaluate our method in problem domains where the search space differs from the continuous search spaces of our tested symbolic regression problems. Recent work [9,11] led to more insight into the antagonism between continuous and discrete search spaces and its implications for the success of crossover-based algorithms in CGP. For our experiments, we also did not include the comparison to other crossover operators that have been proposed for CGP. For our initial evaluation and generally as a first step we concentrated on comparisons to the most commonly used algorithm in CGP and ensuring fair conditions with meta-optimization. But since several crossover operators have been proposed in recent years, more comparative studies are needed in the field of CGP and should be addressed by future work.

Another point that should be discussed is the parametrization of our method. Based on our meta-optimization experiments, we can derive some essential generalizations for our tested problems. In our experiments, moderate to high crossover rates performed best in combination with mid-size populations. We also tested low and very high rates of crossover but obtained no further improvement in the search performance. Likewise, we also experimented with bigger and smaller populations but the size of 50 individuals turned out to be the best choice. Overall, our results give more evidence that mid-size populations can be used effectively in CGP which depicts a significant shift from the popular dogma that only very small populations can perform effectively in CGP. Moreover, our results are coherent with the work of Kalkreuth [11] on population sizes in CGP and reinforce his findings. Nevertheless, we again, have to point out that our findings are based on results that have been obtained in merely one problem category.

7 Conclusions and Future Work

In this work, we presented initial results of a method for phenotypic discrete recombination in CGP. The effectiveness of our approach has been evaluated on a diverse set of well-known symbolic regression benchmarks, covering uni- and bivariate functions. Overall, our results indicate that the use of our proposed methods can be beneficial for symbolic regression. This work primarily focused on an initial evaluation of the search performance and ensuring fair conditions through meta-optimization. The next natural following step is the evaluation of our method in other problem domains and in comparison to other crossover

operators. Therefore, our future work will primarily focus on comparative studies. Another part of our future work will be devoted to analytical experiments to study the effects caused by the phenotypic discrete crossover.

References

1. Bäck, T., Hoffmeister, F., Schwefel, H.: A survey of evolution strategies. In: Belew, R.K., Booker, L.B. (eds.) Proceedings of the 4th International Conference on Genetic Algorithms, San Diego, CA, USA, July 1991, pp. 2–9. Morgan Kaufmann (1991)
2. Beyer, H., Schwefel, H.: Evolution strategies - a comprehensive introduction. Nat. Comput. **1**(1), 3–52 (2002). https://doi.org/10.1023/A:1015059928466
3. Clegg, J., Walker, J.A., Miller, J.F.: A new crossover technique for cartesian genetic programming. In: Thierens, D., et al. (eds.) Proceedings of the 9th Annual Conference on Genetic and Evolutionary Computation, GECCO 2007, London, 7–11 July 2007, vol. 2, pp. 1580–1587. ACM Press (2017). https://doi.org/10.1145/1276958.1277276. http://www.cs.bham.ac.uk/~wbl/biblio/gecco2007/docs/p1580.pdf
4. De Jong, K., Spears, W.: On the virtues of parameterized uniform crossover. In: Proceedings of the 4th International Conference on Genetic Algorithms, pp. 230–236. Morgan Kaufmann Publishers, San Mateo (1991)
5. Hrbacek, R., Dvorak, V.: Bent function synthesis by means of cartesian genetic programming. In: Bartz-Beielstein, T., Branke, J., Filipič, B., Smith, J. (eds.) PPSN 2014. LNCS, vol. 8672, pp. 414–423. Springer, Cham (2014). https://doi.org/10.1007/978-3-319-10762-2_41
6. Husa, J., Kalkreuth, R.: A comparative study on crossover in cartesian genetic programming. In: Castelli, M., Sekanina, L., Zhang, M., Cagnoni, S., García-Sánchez, P. (eds.) EuroGP 2018. LNCS, vol. 10781, pp. 203–219. Springer, Cham (2018). https://doi.org/10.1007/978-3-319-77553-1_13
7. Husa, J., Sekanina, L.: Evolving cryptographic boolean functions with minimal multiplicative complexity. In: IEEE Congress on Evolutionary Computation, CEC 2020, Glasgow, United Kingdom, 19–24 July 2020, pp. 1–8. IEEE (2020). https://doi.org/10.1109/CEC48606.2020.9185517.
8. Kalganova, T.: Evolutionary approach to design multiple-valued combinational circuits. In: Proceedings of the 4th International Conference on Applications of Computer Systems, ACS 1997, Szczecin, Poland, pp. 333–339 (1997)
9. Kalkreuth, R.: A comprehensive study on subgraph crossover in cartesian genetic programming. In: Guervós, J.J.M., Garibaldi, J.M., Wagner, C., Bäck, T., Madani, K., Warwick, K. (eds.) Proceedings of the 12th International Joint Conference on Computational Intelligence, IJCCI 2020, Budapest, Hungary, 2–4 November 2020, pp. 59–70. SCITEPRESS (2020). https://doi.org/10.5220/0010110700590070.
10. Kalkreuth, R., Rudolph, G., Droschinsky, A.: A new subgraph crossover for cartesian genetic programming. In: McDermott, J., Castelli, M., Sekanina, L., Haasdijk, E., García-Sánchez, P. (eds.) EuroGP 2017. LNCS, vol. 10196, pp. 294–310. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-55696-3_19
11. Kalkreuth, R.T.: Reconsideration and Extension of Cartesian Genetic Programming. Ph.D. thesis (2021). https://doi.org/10.17877/DE290R-22504. http://dx.doi.org/10.17877/DE290R-22504

12. Kaufmann, P., Kalkreuth, R.: An empirical study on the parametrization of cartesian genetic programming. In: Proceedings of the Genetic and Evolutionary Computation Conference Companion, GECCO 2017, pp. 231–232. ACM, New York (2017). https://doi.org/10.1145/3067695.3075980. http://doi.acm.org/10.1145/3067695.3075980
13. McDermott, J., et al.: Genetic programming needs better benchmarks. In: Proceedings of the 14th International Conference on Genetic and Evolutionary Computation Conference, GECCO 2012, Philadelphia, Pennsylvania, USA, 7–11 July 2012, pp. 791–798. ACM (2012). https://doi.org/10.1145/2330163.2330273
14. Miller, J.F., Thomson, P., Fogarty, T.: Designing electronic circuits using evolutionary algorithms. arithmetic circuits: a case study. In: Genetic Algorithms and Evolution Strategies in Engineering and Computer Science, pp. 105–131. Wiley (1997)
15. Miller, J.F.: An empirical study of the efficiency of learning boolean functions using a cartesian genetic programming approach. In: Banzhaf, W., et al. (eds.) Proceedings of the Genetic and Evolutionary Computation Conference, Orlando, Florida, USA, 13–17 July 1999, vol. 2, pp. 1135–1142. Morgan Kaufmann (1999). http://citeseer.ist.psu.edu/153431.html
16. Miller, J.F., Wilson, D.G., Cussat-Blanc, S.: Evolving programs to build artificial neural networks. In: Adamatzky, A., Kendon, V. (eds.) From Astrophysics to Unconventional Computation. ECC, vol. 35, pp. 23–71. Springer, Cham (2020). https://doi.org/10.1007/978-3-030-15792-0_2
17. Miller, J.F.: Cartesian genetic programming: its status and future. Genet. Program. Evolvable Mach. **21**(1), 129–168 (2020). https://doi.org/10.1007/s10710-019-09360-6
18. Poli, R., Langdon, W.B.: On the ability to search the space of programs of standard, one-point and uniform crossover in genetic programming. Technical report CSRP-98-7, University of Birmingham, School of Computer Science (January 1998). ftp://ftp.cs.bham.ac.uk/pub/tech-reports/1998/CSRP-98-07.ps.gz. Presented at GP-98
19. Poli, R., Langdon, W.B.: On the search properties of different crossover operators in genetic programming. In: Koza, J.R., et al. (eds.) Genetic Programming 1998: Proceedings of the 3rd Annual Conference, University of Wisconsin, Madison, Wisconsin, USA, 22–25 July 1998, pp. 293–301. Morgan Kaufmann (1998). http://www.cs.essex.ac.uk/staff/poli/papers/Poli-GP1998.pdf
20. Rechenberg, I.: Evolutionsstrategie: Optimierung technischer Systeme nach Prinzipien der biologischen Evolution. Dr.-Ing. Ph.D. thesis, Thesis, Technical University of Berlin, Department of Process Engineering (1971)
21. Rechenberg, I.: Evolutionsstrategie Optimierung technischer Systeme nach Prinzipien der biologishen Evolution. Frommann Holzboog Verlag, Stuttgart (1973)
22. Rudolph, G.: Global optimization by means of distributed evolution strategies. In: Schwefel, H.-P., Männer, R. (eds.) PPSN 1990. LNCS, vol. 496, pp. 209–213. Springer, Heidelberg (1991). https://doi.org/10.1007/BFb0029754
23. Schwefel, H.P.: Evolutionsstrategien für die numerische Optimierung, pp. 123–176. Birkhäuser Basel, Basel (1977). https://doi.org/10.1007/978-3-0348-5927-1_5
24. Schwefel, H.P.: Numerical Optimization of Computer Models. Wiley, USA (1981)
25. Scott, E.O., Luke, S.: ECJ at 20: toward a general metaheuristics toolkit. In: López-Ibáñez, M., Auger, A., Stützle, T. (eds.) Proceedings of the Genetic and Evolutionary Computation Conference Companion, GECCO 2019, Prague, Czech Republic, 13–17 July 2019, pp. 1391–1398. ACM (2019). https://doi.org/10.1145/3319619.3326865

26. Sekanina, L., Walker, J.A., Kaufmann, P., Platzner, M.: Evolution of electronic circuits. In: Miller, J.F. (ed.) Cartesian Genetic Programming. Natural Computing Series, pp. 125–179. Springer, Heidelberg (2011). https://doi.org/10.1007/978-3-642-17310-3_5
27. da Silva, J.E.H., Bernardino, H.: Cartesian genetic programming with crossover for designing combinational logic circuits. In: 7th Brazilian Conference on Intelligent Systems, BRACIS 2018, São Paulo, Brazil, 22–25 October 2018, pp. 145–150. IEEE Computer Society (2018). https://doi.org/10.1109/BRACIS.2018.00033
28. Suganuma, M., Kobayashi, M., Shirakawa, S., Nagao, T.: Evolution of deep convolutional neural networks using cartesian genetic programming. Evol. Comput. **28**(1), 141–163 (2020). https://doi.org/10.1162/evco_a_00253
29. Syswerda, G.: Uniform crossover in genetic algorithms. In: Schaffer, J.D. (ed.) Proceedings of the 3rd International Conference on Genetic Algorithms, George Mason University, Fairfax, Virginia, USA, June 1989, pp. 2–9. Morgan Kaufmann (1989)
30. Turner, A.J.: Improving crossover techniques in a genetic program. Master's thesis, Department of Electronics, University of York (2012)