On the Effectiveness of Crossover Operators in Cartesian Genetic Programming

Mark Kocherovsky(✉), Marzieh Kianinejad, Illya Bakurov, and Wolfgang Banzhaf

Department of Computer Science and BEACON Center for the Study of Evolution in Action, Michigan State University, East Lansing, MI 48824, USA
kocherov@msu.edu
https://banzhaf-lab.github.io/

Abstract. This study investigates the effectiveness of Cartesian Genetic Programming by analyzing numerous indicators of evolutionary dynamics when using different crossover operators and the canonical mutation-only $(1 + 4)$ strategy. Specifically, we examine a traditional crossover operator which is based on the random selection of parental genes; Subgraph Crossover, where points in the range of active nodes are considered; and the recently-proposed Deep Neural Crossover (DNC) approach which utilizes a transformer network to learn correlations between genes and predict potentially beneficial crossover points. The performance of these different crossovers is evaluated on 11 standard and one real-world regression problem.

Keywords: Cartesian Genetic Programming · Crossover · Neural Crossover · Crossover Effects · Reinforcement Learning

1 Introduction

Introduced by Miller in the late 1990 s [28,30], Cartesian Genetic Programming (CGP) is one of several Genetic Programming (GP) paradigms currently heavily used and under investigation in the field. This paradigm relies on a directed acyclic graph (DAG) representation, which allows the reuse of intermediate results and multiple outputs. Since its introduction, it was observed that traditional crossover operators were detrimental to search in CGP and the scientific community mostly avoids their usage [29]. Nevertheless, there were several attempts to design an effective crossover for CGP that could consistently outperform the canonical $(1 + \lambda)$ strategy [8–10,19–22,36,38], where there is no crossover, and only one parent is selected to be mutated several times. Despite numerous efforts, the scientific community shares the *assumption* that standard crossover is simply too destructive for CGP, and as we will see in Sect. 5, certain other assumptions. In summary, the literature skirts around the main question: *why is crossover so destructive in CGP in the first place?*

B. Xue et al. (Eds.): EuroGP 2025, LNCS 15609, pp. 68–84, 2025.
https://doi.org/10.1007/978-3-031-89991-1_5

In an attempt to understand why traditional crossover is so destructive in CGP, the authors of [24] compared the performance of different crossover operators in CGP and Linear Genetic Programming (LGP). The latter is a GP paradigm that represents and executes solutions as a sequence of instructions, akin to a low-level programming language, where each instruction specifies an operation and the memory registers/locations [7]. Although it was shown that CGP and LGP can be mutually convertible [7], allowing direct comparison between their DAGs [40], crossover in LGP is frequently found to benefit search. In [24], this was linked to a more drastic node connectivity disruption in CGP when the genetic operators are applied, which results in children being far more genetically different from their parents than in LGP.

Previous research about the crossover in CGP relied upon methods that perform random selection of parental genes. For example, n-point crossover randomly selects n crossover points and swaps segments in between. Uniform Crossover swaps parent genes at each index with uniform probability. The real-valued crossover [9] performs a randomly weighted average of the two parents. Subgraph Crossover [19,21] essentially performs a one-point crossover between active function nodes of the two parents, aiming at recombining the subgraphs with semantic value.

In this work, we seek to deepen the community's understanding of crossover destructiveness in CGP following [24]. In particular, we adapt and utilize a recently proposed Deep Neural Crossover (DNC) [35] to CGP for the first time and compare it to the aforementioned crossover methods. DNC contains an encoder-decoder neural network trained in real-time to learn structural correlations between parent genes in order to select crossover points that maximize offspring fitness. DNC was originally developed for Genetic Algorithms (GAs) to guide the probability of swapping genes in Uniform Crossover (making it, effectively, non-uniform), and was shown to outperform traditional operators substantially. Additionally, we include the Real-Valued Crossover proposed in [9] to investigate the suitability of a floating-point representation of CGP chromosomes. We also analyze whether allowing a variable-length representation for CGP, akin to LGP, helps to improve the program search. Finally, we try to shed light on how different crossover methods influence evolutionary dynamics beyond commonly used learning curves, phenotypic diversity, and the proportion of active nodes. Here we also observe the relationships between the parent-child similarity, the fitness landscape properties of individuals, and the spatio-temporal distribution of crossover and mutation events over parent genotypes categorized by offspring fitness (i.e., whether worse, near-equal, or improved fitness was observed).

We find that (i) DNC does not allow CGP to synthesize better programs than mutation-only CGP, (ii) Cui et al.'s [10] results that CGP nodes are positionally dependent, i.e. the probability of whether a node will be active depends on the node's position, are confirmed, (iii), traditional crossover operators fail because they assume that nodes close to each other are semantically related, which is not true in CGP.

The remainder of the paper is structured as follows: Sect. 2 introduces the established CGP methods and the literature on CGP crossover. Section 3 discusses our experimental methods, whose results are shown in Sect. 4. Finally, in Sect. 5, we discuss the implications of our results and provide future directions for investigation.

2 Background

2.1 CGP

The basis of a Cartesian program is an instruction that is commonly referred to as a *node*. Each node has a structure of $n_m \in N = [o_m\ a_{m,1}\ a_{m,2}]$, which determines the operator and operand nodes, assuming an arity of 2. Note the lack of memory: Each node takes its input directly from the result of the execution of earlier nodes. However, nodes that are neighbors in the genotype (i.e. neighbors in the chromosome) are not necessarily functionally related due to the DAG structure: Each node can connect to any preceding node, regardless of its physical position in the genotype, and some nodes may not contribute to the final output at all (acting as *introns*), referred to as *inactive nodes*. Thus, a program p is defined as $p \in P = \{n_0, n_1, n_2, ..., n_\beta\} \cup [o_0, o_1, o_2, ..., o_n]$, where each output node takes a single operand, and can be an input or program node. CGP programs are necessarily of constant size, determined by parameter β. In CGP, each node is typically concatenated to form a one-dimensional vector. However, for ease of use, each program is represented as a two-dimensional matrix, except during crossover, where the matrix is flattened into a one-dimensional vector and then reshaped into a matrix once the operator is finished.

2.2 Crossover in CGP

Even though they are not commonly used, several existing crossover methods can be applied to CGP, each representing different assumptions about the connectivity in a DAG. Initially, CGP researchers and practitioners adapted GA-like crossover operators, such as n-Point Crossover [31]. Here, different values of n allow a different treatment of gene order and positional dependence in the chromosomes. For a small n, the operator swaps a few consecutive gene segments, allowing neighboring genes to remain together in the offspring. This assumes that genes closer to each other in the genome have some functional relationship and should be passed together to preserve it. However, as n increases, the relationships and positional dependence progressively fade. In an extreme case, such as Uniform Crossover, each gene is treated as having no positional or relational dependence on neighboring genes, discarding any notion of order or connectivity. Because CGP's DAG structure allows each node to connect with any preceding node, crossover operators that assume neighboring genes have functional relationships can disrupt essential dependencies. In contrast, operators that completely disregard positional relationships can amplify these destructive effects, although they introduce greater genetic diversity.

Clegg *et al.* [9] adapted the Real Valued Crossover for CGP, frequently used in GA literature for solving continuous optimization problems. This crossover essentially performs a randomly-weighted average of the two parent genes. Given that traditional CGP uses an integer-based representation, the authors proposed an algorithm for converting it into a [0–1] range of floating-points. The real-valued CGP crossover was assessed on two symbolic regression problems (Koza2 and Koza3) and was found to outperform the mutation-alone strategy only on one problem. Nevertheless, it was found to achieve faster convergence in earlier iterations, which led the authors to adapt the crossover rate over generations. The adaptive variant turned out to outperform the mutation-alone strategy. However, a later study examined real-valued crossover on three additional classes of problems and found no advantage in using it [39].

In [8], Cai *et al.* highlight other challenges of traditional CGP representation and its destructive effects on recombination. They argue that a node's location in the CGP chromosome can influence whether it contributes to the final solution. They also note that node position does not correlate with behavior, as identical positions across chromosomes do not imply the same functionality and the same functionality can be observed across different positions. As a result, position-based gene exchanges, typical in traditional crossovers, often disrupt useful sub-structures. To circumvent this limitation, the authors proposed a more flexible representation where gene location does not influence phenotype, demonstrating superior performance on the even-3 parity problem using the 2-point crossover.

Goldman and Punch [13] observed that the probability of a given node being active varies significantly by position, with a bias toward nodes near the input, which receive more connections due to the DAG structure in CGP, where each node can only connect to preceding nodes to avoid cycles. Two strategies were proposed to mitigate this bias. `DAG` mutates connections to every other node in the genome, as long as the new connection does not create a cycle; `Reorder` shuffles active nodes through the chromosome in a way that phenotype does not change. Both strategies increased the amount of active nodes and achieved faster convergence with a smaller genotype. Although Goldman and Punch [13] studied CGP$(1+\lambda)$ with point mutation, it is reasonable to argue that crossover under-performance can also be attributed to positional bias. To this end, the authors of [10] examined whether the reorder strategy could improve its performance. They compared two standard CGP$(1+\lambda)$ variants against four crossover operators with the reorder strategy on four boolean benchmarks and concluded that crossover with reordering benefits search. In three out of four problems, the crossover was found to outperform CGP$(1+\lambda)$, and Uniform Crossover generally benefits the most.

In 2017, Kalkreuth et al. introduced Subgraph Crossover [21], which exchanges active subgraphs between parents by selecting crossover points within active nodes, focusing recombination on genome segments that impact offspring fitness. Later, Kalkreuth [19] evaluated Subgraph Crossover with large genotypes and small population sizes on symbolic regression and Boolean problems, finding that it generally outperformed mutation-only approaches with faster convergence and better final fitness in regression tasks. He concluded that medium (50) and

large (250) population sizes perform best, while larger genotypes offered no significant advantage. In [18], Husa and Kalkreuth presented the *block crossover*, which was inspired by the Subgraph Crossover and the cone-based module creation proposed in [22], which enables CGP to evolve and reuse modules (subcomponents of the program). The method essentially swaps blocks of consecutive active nodes between two parents. The authors assessed the performance of the block, Subgraph, One-Point, and real-valued crossover, and compared them with mutation-only strategies on a subset of problems taken from [21]. Testing block, Subgraph, One-Point, and real-valued crossovers against mutation-only strategies, they found $(1 + \lambda)$ mutation was the best overall, possibly due to the lack of crossover parameter tuning, such as crossover rate.

Preserving gene structures that remain unchanged across generations can be beneficial, as they likely confer a fitness advantage. In biological systems, this concept appears during meiosis, where Homologous Crossover conserves similar sequences while exchanging non-similar parts, promoting genetic diversity [17]. Homologous Crossover has long been explored in GP [6] using both trees [32] and linear representations [12]. Recently, a Homologous Crossover was proposed for CGP in [38], where multiple sequence alignment (MSA) is used to interleave gaps between the sequences so that they achieve the same length and have the maximum possible similarity. After allowing initial generations to stabilize on some meaningful, high-fitness building blocks, MSA identifies intact/conserved sequences without any gaps that appear consistently across multiple genomes. A single representation is then extracted for commonly conserved and non-conserved genes, acting as a footprint of areas of stability and variability in the population's genetic pool. During crossover, two individuals with the highest fitness values that align well with this footprint are selected as parents. In their work, Uniform Crossover is used. The new crossover was explored in the context of neural architecture search (NAS) for image classification, and the results were compared with those found in the literature using human-designed architectures and other types of NAS. The comparison included CGP$(1 + \lambda)$ strategy from a related study [37], where similar CGP hyper-parameters were used. While the accuracy of CGP with the new Homologous Crossover was found to be about 2% above that reported in [37], the evolved network was also found to be slightly more complex. Thus, it is not obvious whether the proposed crossover effectively improves CGP search.

3 Methods

3.1 Problems

We tested each method by evolving models to find eleven univariate functions over 10,000 generations. We chose known optimization problems by virtue of being complex enough to be challenging for the GP to find, as well as to match the literature [20,24]. At each run, we randomly sampled 20 points for simpler problems and 40 for more complex ones from the respective domain. For more details, including functional forms, see Table 1. Also, we assessed the methods on a real-world problem, the Diabetes dataset [11]. A 70–30% train-test split

was used and models were trained for 3,000 generations. Note that over time, only training fitness was collected, and testing fitness was collected at the end.

Table 2 shows our notation and basic hyperparameters, and Table 3 shows our hyperparameters for the DNC [35].

Table 1. Problems tested; each model was given a set of random points within the given domain. *: problem too complex to be written out here, see the reference. Mostly reproduced from [24].

Problem	Function	Domain	Points
Koza-1	$x^4 + x^3 + x^2 + x$	$[-1, 1]$	20
Koza-2	$x^5 - 2x^3 + x$	$[-1, 1]$	20
Koza-3	$x^6 - 2x^4 + x^2$	$[-1, 1]$	20
Nguyen-4	$x^6 + x^5 + x^4 + x^3 + x^2 + x$	$[-1, 1]$	20
Nguyen-5	$\sin(x^2)\cos(x) - 1$	$[-1, 1]$	20
Nguyen-6	$\sin(x) + \sin(x + x^2)$	$[-1, 1]$	20
Nguyen-7	$\ln(x + 1) + \ln(x^2 + 1)$	$[0, 2]$	20
Ackley [2]	$-20\exp(-0.2x^2) - \exp(\cos 2\pi x) + 20 + \exp 1$	$[-32.768, 32.768]$	40
Rastrigin [33]	$10 + x^2 - 10\cos 2\pi x$	$[-5.12, 5.12]$	40
Levy [25]	*	$[-10, 10]$	40
Griewank [14]	$x^2/4000 - \cos x + 1$	$[-600, 600]$	40
Diabetes [11]	*	See [11]	442 (70/30% Split)

Table 2. Evolutionary parameters to demonstrate the effects of crossover. Mostly reproduced from [24].

Notation	Crossover	Mutation
CGP(1+4)	None (Canonical)	$\mu = 100\%$ $(x4)$
CGP-1x(40+40)	One-Point (50%)	$\mu = 2.50\%$
CGP-2x(40+40)	Two-Point (50%)	$\mu = 2.50\%$
CGP-VL1x(40+40)	One-Point | Variable-Length (50%)	$\mu = 2.50\%$
CGP-VL2x(40+40)	Two-Point | Variable Length (50%)	$\mu = 2.50\%$
CGP-Ux(40+40)	Uniform (50%)	$\mu = 2.50\%$
CGP-SGx(40+40)	Subgraph (50%)	$\mu = 2.50\%$
CGP-DNC(40+40)	Uniform Deep Neural Crossover (50%)	$\mu = 2.50\%$
CGP-DNC-1x(40+40)	One-Point Deep Neural Crossover (50%)	$\mu = 2.50\%$
CGP-RV(40+40)	Real-Valued (50%)	$\mu = 2.50\%$

3.2 Deep Neural Crossover (DNC)

Previous research about the crossover in CGP relied upon methods that select parental genes at random. In recent years, conceptually novel evolutionary operators have been proposed that leverage deep learning (DL) capabilities to process

Table 3. Neural Network Hyperparameters for DNC [34]

Hyperparameter	Value
Embedding Dimension	64
Sequence Length	193
Number of Embeddings	75
Running Mean Decay	0.95
Sample Size for Training	820 pairs of parents
Learning Rate	1e−4
Greedy ϵ	0.2
Number of Parents	2

sequence data, like genes, to identify complex relationships. NeuroCrossover [26] utilizes Dual-Aspect Collaborative Transformer architecture [27], coupled with online reinforcement learning (RL), which was shown to be able to optimize the selection of points for order and 2-point crossover applied for routing and packing problem-solving. Specifically, the attention modules learn embedded representations of the parent genomes using what the authors call Cross Information Synergistic Attention (CISA), where each parent embedding is learned separately but can be cross-referenced by the other to find common features. Once the encoder and decoder are trained, the network can then work on finding children with better fitness than the parents, outputting relevant crossover points.

Later, Shem-Tov and Elyasaf extended NeuroCrossover by proposing to learn a probability distribution of exchanging each gene, akin to Uniform Crossover [35]. Their method, called Deep Neural Crossover (DNC), is motivated by the assumptions that distinguish One-Point and Uniform Crossover (previously discussed in Sect. 2.2). However, there are more differences. Although DNC also learns through policy-based online reinforcement learning, a different neural architecture based on an LSTM Pointer Network is employed. The latter uses a set of softmax modules with attention to produce the probability of selecting a gene from each parent. In addition, the authors use what they call an $\epsilon-$greedy method to randomly perform actions with the probability ϵ. Moreover, DNC can operate in a multiple-parent setting, which is different from the two-parent approach in [26]. Diagrams of the encoding and decoding network are shown in Fig. 1.

There are some minor differences between the baseline DNC code and ours. First, where DNC produces a single child from each pairing, we produce two children to match the other forms of crossover we measure. For the same reason, we use a two-parent setting rather than the multi-parent option provided. Second, minor structural alterations are made to accommodate the CGP structures we were already using: we enabled ourselves to plug in our fitness function, mutation operator, and data recording methods. The ANN methods were untouched as DNC is genome-agnostic. In addition, we also test these strategies with *fixed par-*

ents: in short, each starting set of parents in each replicate are identical instead of randomly initialized each time; and we tested different neural network learning rates. In our work, we test both Uniform and One-Point DNC (as in [26]).

3.3 Other Design Details

We also tested the One-Point and Two-Point crossovers with *variable-length individuals*, which uses a mutation operator close to LGP, in that it can add and remove instructions as well as perform point mutation; and crossover between nodes, not genes. All methods except DNC used *forced diversity*, making sure each parent returned by the selection operator has a unique genotype [3,5]. DNC is excluded as we did not want to tamper with the authors' existing selection method, which is standard tournament selection. For selection, the CGP$(1+4)$ strategy uses elite selection, DNC uses the tournament selection as in [35], and the others use tournament selection with elitism following [24]. All trials were run using the MSU High-Powered Computing Cluster [1]. Point mutation is used at a gene level.

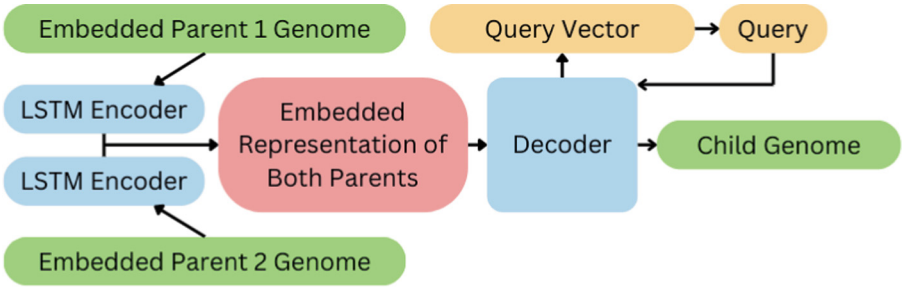

Fig. 1. Encoder and Decoder architecture in DNC. Each parent is converted into an embedding. Embedded child genomes are generated and converted back into a typical genomic representation [35].

In each run and for each generation, we record several metrics in relation to the evolutionary dynamics. These are the fitness and number of active nodes of the best-performing individuals, the change in fitness between the best parent and child, and parent-offspring similarity as stated by [4] (*Similarity*). We also count the amount of deleterious, near-neutral, or beneficial crossover and mutation events at each generation (*Crossover* and *Mutation Impact*); for determining the impact category, the percent difference of the parent and child fitness was measured, and labeled as deleterious if $\%\Delta f > 0.001$, beneficial if $\%\Delta f < -0.001$, and near-neutral otherwise. Finally, we measure the distribution of crossover indices over each generation categorized by impact. These metrics are briefly summarized in Table 4.

For our fitness function, we use the Pearson correlation as suggested in [15, 23], which focuses selection on expressions that are close in shape to the training

Table 4. Summary of Metrics Recorded.

Metric	Description
Best Fitness	Smallest fitness at the end of evolution
Best Fitness per Generation	Smallest fitness during each generation
Semantic Diversity	Standard Deviation of population semantics (fitnesses) in each generation
Instructions	Number of active instructions in the best model after evolution
Program Size per Generation	Number of active nodes in the best program in each generation
Similarity	Similarity score (using alignment [4]) between parents and their best child
Xover Density Distribution	Temporal frequency of indices used as crossover points categorized by impact direction
Mutation Density Distribution	Temporal frequency of indices mutated categorized by impact direction

target, instead of merely requiring that aggregate error is small, as can happen using RMSE simply because the scale of model output is close to targets, despite the quality of fit being poor. To minimize the fitness measure, we use $1 - r^2$, where r is the Pearson Correlation.

4 Results

4.1 Median Performance

For each replicate, we saved the fitness of the best model in the population. Median fitness values of the best models for each crossover method for each problem are shown in Table 5 and in more detail, together with the full Mann-Whitney significance tables, in the supplementary material https://github.com/MarkKocherovsky/cgp_crossover. From Table 5, we can conclude that Canonical CGP is the best performer for most of the problems considered, except being significantly worse at the Levy problem, not significantly different from other results for the Ackley problem, and being insignificantly worse than Subgraph on Nguyen 6. However, One-Point-Variable-Length, Uniform, and One-Point Crossovers show competitive performance on several problems, with Two-Point-Variable-Length and Subgraph also performing well in some cases. The Real-Valued and Deep Neural Crossovers perform significantly worse. On the Diabetes problem, both Variable-Length crossovers perform significantly better than the other methods, and DNC performs significantly worse.

4.2 Temporal Plots

For our assessment of the metrics discussed in Table 4, we have chosen to limit our figures in this paper to Koza 3 and Rastrigin because of the details in each plot. Both problems are difficult to solve and are therefore adequate representatives of the evolutionary dynamics. Full plots are available at https://github.com/MarkKocherovsky/cgp_crossover.

Figure 2a shows the median fitness of the best individuals in the population at each generation. For the paper, we will look at our two example problems starting at generation 100, as the most interesting development occurs between 500—2000 generations into evolution: Canonical CGP, which starts with much

Table 5. Median fitness values of the best models for each crossover method in each problem after 50 runs. The best value is shown in the deepest green and in **bold**, and the worst in the deepest red.

	Koza 1	Koza 2	Koza 3	Nguyen 4	Nguyen 5	Nguyen 6
CGP(1+4)	**6.76E-04**	**5.11E-03**	**9.91E-03**	**8.91E-04**	2.84E-04	5.06E-04
CGP-1x	1.12E-03	1.67E-02	4.18E-02	1.33E-03	1.11E-03	8.76E-04
CGP-DNC	1.70E-03	2.03E-02	1.17E-01	2.02E-03	1.35E-03	9.71E-04
CGP-DNC-1x	1.48E-03	2.73E-02	8.30E-02	1.93E-03	1.51E-03	9.78E-04
CGP-1xVL	1.26E-03	6.60E-03	2.74E-02	1.58E-03	2.36E-04	5.68E-04
CGP-2xVL	1.93E-03	8.94E-03	1.98E-02	2.05E-03	6.78E-04	1.01E-03
CGP-2x	1.12E-03	1.71E-02	4.89E-02	1.30E-03	1.04E-03	8.80E-04
CGP-RVx	3.62E-03	1.00E-01	3.38E-01	9.48E-03	2.77E-03	2.10E-03
CGP-SGx	1.21E-03	2.31E-02	4.16E-02	1.63E-03	**1.09E-04**	8.45E-04
CGP-Ux	9.34E-04	1.19E-02	1.75E-02	1.55E-03	7.55E-04	**4.99E-04**
	Nguyen 7	Ackley_1D	Rastrigin_1D	Levy_1D	Griewank_1D	Diabetes
CGP(1+4)	**8.89E-06**	2.92E-02	**3.61E-01**	1.61E-01	**6.00E-04**	5.09E-01
CGP-1x	1.53E-05	2.78E-02	4.01E-01	**9.78E-02**	6.25E-04	5.22E-01
CGP-DNC	2.59E-05	3.39E-02	4.16E-01	1.91E-01	6.39E-04	5.34E-01
CGP-DNC-1x	4.47E-05	3.38E-02	4.11E-01	1.94E-01	6.45E-04	5.38E-01
CGP-1xVL	1.33E-05	2.70E-02	4.10E-01	1.02E-01	6.31E-04	**4.87E-01**
CGP-2xVL	1.81E-05	2.88E-02	4.24E-01	1.06E-01	6.19E-04	4.92E-01
CGP-2x	1.28E-05	2.85E-02	4.10E-01	1.04E-01	6.30E-04	5.25E-01
CGP-RVx	5.43E-04	3.16E-02	4.41E-01	1.07E-01	6.39E-04	5.24E-01
CGP-SGx	1.02E-04	2.77E-02	4.14E-01	1.06E-01	6.37E-04	5.09E-01
CGP-Ux	1.30E-05	**2.68E-02**	4.01E-01	1.01E-01	6.12E-04	5.18E-01

worse models on average, likely due to its small population size (as it is a (1+4) ES), overtakes all crossover methods rather early in the evolutionary process. Even though each method shows improvement of fitness over time, crossover clearly stunts the rate of improvement. In Fig. 2b, we show the median amount of active nodes for the elite model in each replicate over time. Canonical CGP produces more complex models, which may correspond to its high performance, but some competitive crossovers, namely Uniform Crossover in Koza 3 and Two-Point crossover in Rastrigin, evidently produce less complex models, indicating a complicated solution landscape.

At the population level, we measured *semantic diversity*, defined as the standard deviation of fitnesses in the population. This is shown in Fig. 2c. We can see that the greatest diversity comes from the Variable-Length group, closely followed by Subgraph and Canonical CGP. Though the high diversity of the latter is likely caused by its small population, the former two, as we will see, have high amounts of deleterious events when compared to the other methods, so the higher diversity is reasonable. Figure 2d shows the median *similarity* between

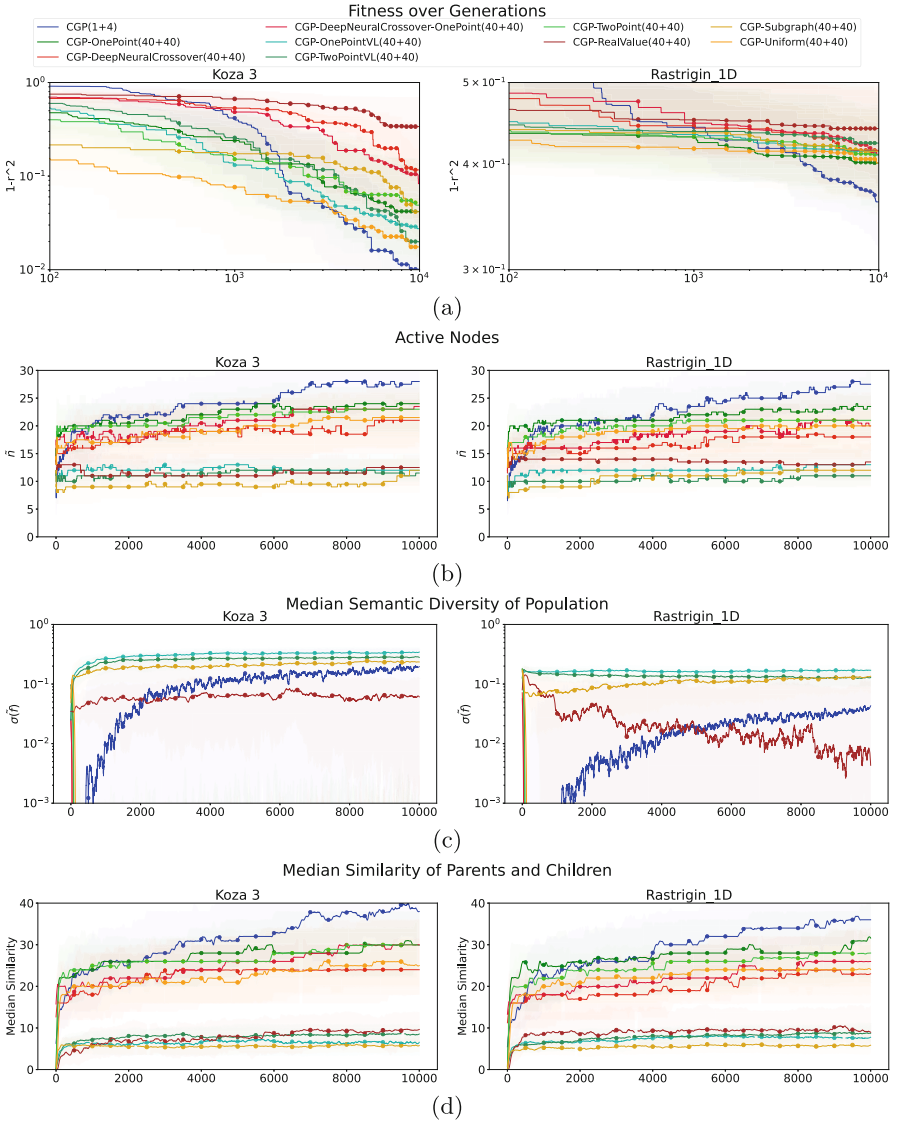

Fig. 2. (**a**): Median fitness of the elite model in the population starting at generation 100, with markers indicating every 500 generations. The error ribbons fill in the area between the first and third quartiles. Full plots are available in the supplementary material. (**b**): Median amount of active nodes in the elite model at each generation. CGP canonical tends to be the most complex by the end of evolution. (**c**): Median Semantic Diversity of the Population at each generation. One-Point-Variable-Length, Subgraph, and Canonical CGP have the most diverse populations. (**d**): Median similarity of the best parent and child in each couple at each generation. Canonical CGP produces the most similar children from each parent, as expected.

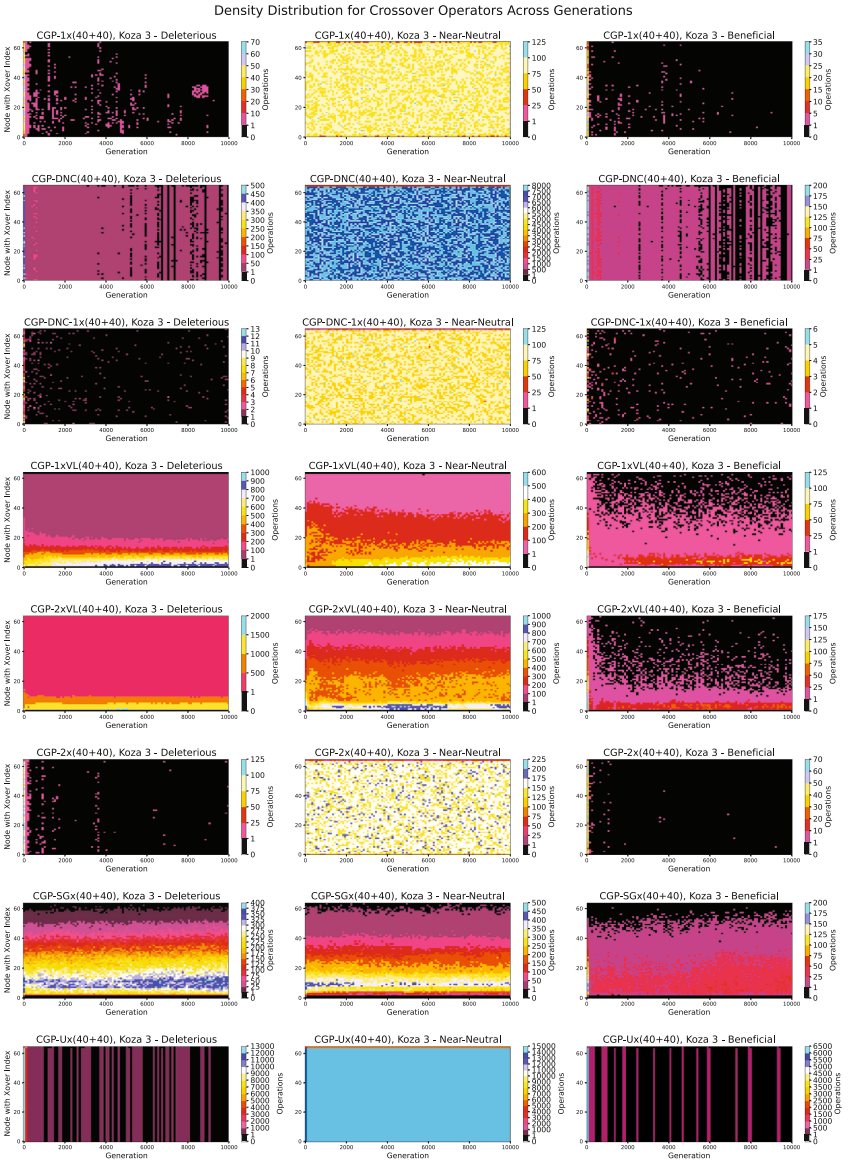

Fig. 3. Density of crossover points over time for Koza 3, grouped by algorithm and effectiveness. The data is condensed into points containing 100 generations and, except for Subgraph and One-Point-Variable-Length,three genes, thus each pixel represents a single node. This decision was made to keep events visible.

the best parent and the best child for each pair of parents, where Canonical CGP, Two-Point methods, and One-Point DNC produce more similar children by the end of evolution.

To check whether there are any patterns in the crossover events themselves, Fig. 3 shows the density of crossover points over time for each crossover method and their effectiveness. The effectiveness of an operation is determined by finding the percent change between the best parent in the couple and the best of that couple's offspring. The operation is then deleterious if $\Delta f < -0.001\%$, beneficial if $\Delta f > 0.001\%$, and near-neutral otherwise, if $-0.001\% < \Delta f < 0.001\%$. First, it is immediately clear that for most cases the overwhelming majority of events are near-neutral, and for the most part, those events are nearly uniformly spread across the genome at all times, which makes sense given the use of a uniform random number generator. It is also obvious that deleterious and beneficial operations are very common at the start of evolution (within the first hundred generations) and from there become increasingly uncommon. Two crossover methods stray from this pattern. Variable-Length methods are more likely to have deleterious than they are to have neutral operations, and operations are more likely closer to the front of the genome. Both observations make sense given that the genome in this case can change size, thus it is more likely to be smaller and more likely to have events at smaller indices, which would make its evolution less stable. Subgraph Crossover, on the other hand, does not have such capabilities: always having a fixed length genome and a randomly selected crossover point (within constraints), the distribution *should* be uniform, yet we see crossover events being much more common closer to the front of the genome. Since crossover points are restricted to active nodes, this seems to confirm Cui et al.'s observations that an instruction closer to the front of the genome is more likely to be active than those at the back [10].

Finally, we collect similar density distributions for mutation operations. However, since we are not testing different mutation operators, the results are relatively uniform across problems and crossover methods: operations are more likely to be deleterious or beneficial if performed near the front of the genome, and more likely to be neutral if performed towards the back.

5 Discussion

Our results indicate that Deep Neural Crossover does not seem to be an effective crossover operator for Cartesian Genetic Programming; DNC and Real-Value Crossover perform significantly worse than Canonical CGP(1+4) and other crossover methods, particularly Two-Point-Variable-Length and Uniform Crossovers. Subgraph Crossover, on the other hand, performs competitively in some scenarios, even when problems are more complex. We posit that the CGP genome does not contain any patterns that can be found with lightweight neural networks. As intimated in Sect. 1, previous literature has largely established constraints on selection and existing crossover methods instead of new ones. We argue that these new constraints assume *both* the semantic equivalence of genes in CGP regardless of position and that nodes closer to each other are likely to be connected. Following [8] and [10], it is most likely that *a node's semantics also rely on its position in the graph.* Even if the phenotype is kept intact, moving a

gene from one position to another may have a more dramatic effect on fitness than previously recognized. This might occur because the Cartesian representation lacks a grounding mechanism such as the internal calculation registers in LGP [24].

From here we can posit that the literature operates on each node without considering the context of its connected nodes. A single instruction can be simultaneously dependent on other instructions and itself a dependency for other nodes. *A crossover method that does not take into account the interdependence of nodes is therefore unlikely to succeed.* This may be why Subgraph Crossover is competitive, as it tries to preserve relationships between active nodes. In this vein, we suggest an exploration of the relationship between positional bias and semantics following [10], which used the `REORDER` operator introduced in Goldman and Punch (2013) [13] to mitigate positional bias in relation to active nodes. We also propose to develop a method that focuses on the semantics (output) of each node rather than simply the genotype and fitness of the phenotype. Positive results have already been published for a semantic *mutation* operator in a $1 + \lambda$ framework [16].

5.1 Summary of Contributions

To summarize, we return to our question from Sect. 1: *why is crossover so destructive in CGP in the first place?* We believe, following [13] and [10], that the problem lies in the assumptions of traditional crossover. Standard n-Point Crossover assumes that genes that are closer to each other are more likely to be positionally related, which in CGP is not the case. We also confirm the findings of Cui *et al.*, namely that the activity of a node is negatively correlated with its position in the genome.

From a theoretical perspective, it would be prudent to run further experiments designed to test the dynamics of positional bias. Regardless of whether this is intuitive, the knowledge of *how* and *why* we observe positional bias is still rather vague at this time. It would also be interesting to put into practice a contextual perspective of CGP nodes and design a crossover operator around the semantics and interconnectivity of individual nodes, .i.e. using a phenotypic perspective on the crossover operator.

Acknowledgments. We would like to thank the Institute for Cyber-Enabled Research at Michigan State University and the John R. Koza Endowment fund administered at MSU.

Disclosure of Interests. The authors declare no conflicts of interest.

References

1. Hardware | institute for cyber-enabled research. https://icer.msu.edu/hpcc/hardware (2024)
2. Ackley, D.: A Connectionist Machine for Genetic Hillclimbing, vol. 28. Springer (2012)
3. Alfaro-Cid, E., Merelo, J., De Vega, F.F., Esparcia-Alcázar, A.I., Sharman, K.: Bloat control operators and diversity in genetic programming: a comparative study. Evol. Comput. **18**(2), 305–332 (2010)
4. Aygün, E., Ecer, D.: Python-alignment (2017). https://github.com/eseraygun/python-alignment
5. Banzhaf, W., Bakurov, I.: On the nature of the phenotype in tree genetic programming. In: Handl, J., Li, X., Wagner, M., Garza-Fabre, M., et al. (eds.) Proceedings of the Genetic and Evolutionary Computation Conference (GECCO 2024), Melbourne, Australia, pp. 868–877. ACM Press (2024)
6. Banzhaf, W., Nordin, P., Keller, R.E., Francone, F.D.: Genetic Programming: An Introduction—On the Automatic Evolution of Computer Programs and its Applications. Morgan Kaufmann Publishers Inc. (1998)
7. Brameier, M., Banzhaf, W.: Linear Genetic Programming, pp. 36–37. Springer (2007)
8. Cai, X., Smith, S.L., Tyrrell, A.M.: Positional independence and recombination in Cartesian Genetic Programming. In: Collet, P., Tomassini, M., Ebner, M., Gustafson, S., et al. (eds.) European Conference on Genetic Programming, pp. 351–360. Springer (2006)
9. Clegg, J., Walker, J.A., Miller, J.F.: A new crossover technique for Cartesian Genetic Programming. In: Thierens, D., Beyer, H.G., Branke, J., et al. (eds.) Proceedings of the 9th Annual Conference on Genetic and Evolutionary Computation, pp. 1580–1587. ACM Press (2007)
10. Cui, H., Margraf, A., Heider, M., Hähner, J.: Towards understanding crossover for Cartesian Genetic Programming. In: van Stein, N., Marcelloni, F., Lam, H.K., Cottrell, M., et al. (eds.) Proceedings of the 15th International Joint Conference on Computational Intelligence (IJCCI 2023), pp. 308–314. SCITEPRESS (2023)
11. Efron, B., Hastie, T., Johnstone, I., Tibshirani, R.: Least angle regression. Ann. Stat. **32**(2), 407–451 (2004)
12. Francone, F.D., Conrads, M., Banzhaf, W., Nordin, P.: Homologous crossover in genetic programming. In: Banzhaf, W., Daida, J.M. (eds.) Proceedings of the 1st Annual Conference on Genetic and Evolutionary Computation, GECCO 1999, vol. 2, pp. 1021-1026. Morgan Kaufmann Publishers Inc., San Francisco (1999)
13. Goldman, B.W., Punch, W.F.: Length bias and search limitations in Cartesian Genetic Programming. In: Silva, S., Esparcia-Alcazar, A.I., Lopez-Ibanez, M., Mostaghim, S., et al. (eds.) Proceedings of the 15th Annual Conference on Genetic and Evolutionary Computation, pp. 933–940 (2013)
14. Griewank, A.O.: Generalized descent for global optimization. J. Optim. Theory Appl. **34**, 11–39 (1981)
15. Haut, N., Banzhaf, W., Punch, B.: Correlation versus RMSE loss functions in symbolic regression tasks. In: Trujillo, L., Winkler, S.M., Silva, S., Banzhaf, W. (eds.) Genetic Programming Theory and Practice XIX, pp. 31–55. Springer (2023)
16. Hodan, D., Mrazek, V., Vasicek, Z.: Semantically-oriented mutation operator in Cartesian Genetic Programming for evolutionary circuit design. In: Coello Coello, C.A., Aguirre, A.H., Uribe, J.C., Fabre, M.G. (eds.) Proceedings of the 2020 Genetic and Evolutionary Computation Conference, pp. 940–948 (2020)

17. Hunter, N.: Meiotic recombination: The Essence of Heredity. Cold Spring Harb. Perspect. Biol. **7 12** (2015). https://api.semanticscholar.org/CorpusID:33542130
18. 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
19. Kalkreuth, R.: A comprehensive study on subgraph crossover in Cartesian Genetic Programming. In: Merelo Guervós, J.J., Garibaldi, J., Wagner, C., et al. (eds.) Proceedings of the 12th International Joint Conference on Computational Intelligence (IJCCI 2020) - Volume 1: ECTA, pp. 59–70. INSTICC, SciTePress (2020). https://doi.org/10.5220/0010110700590070
20. Kalkreuth, R.: Reconsideration and extension of Cartesian Genetic Programming. Ph.D. thesis, Technical University of Dortmund, Germany (2021)
21. 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.) Genetic Programming, pp. 294–310. Springer, Cham (2017)
22. Kaufmann, P., Platzner, M.: Advanced techniques for the creation and propagation of modules in Cartesian Genetic Programming. In: Keijzer, M., Antoniol, G., Congdon, C.B., Deb, K., et al. (eds.) Proceedings of the 10th Annual Conference on Genetic and Evolutionary Computation, GECCO 2008, pp. 1219–1226. Association for Computing Machinery, New York (2008). https://doi.org/10.1145/1389095.1389334
23. Keijzer, M.: Scaled symbolic regression. Genet. Program Evolvable Mach. **5**(3), 259–269 (2004)
24. Kocherovsky, M., Banzhaf, W.: Crossover destructiveness in cartesian versus Linear Genetic Programming. In: Faíña, A., Risi, S., Medvet, E., Stoy, K., et al. (eds.) ALIFE 2024: Proceedings of the 2024 Artificial Life Conference, p. 20. Artificial Life Conference Proceedings, The International Society for Artificial Life (2024). https://doi.org/10.1162/isal_a_00735
25. Laguna, M., Marti, R.: Experimental testing of advanced scatter search designs for global optimization of multimodal functions. J. Glob. Optim. **33**, 235–255 (2005)
26. Liu, H., Zong, Z., Li, Y., Jin, D.: NeuroCrossover: An Intelligent genetic locus selection scheme for genetic algorithm using reinforcement learning. Appl. Soft Comput. **146**, 110680 (2023)
27. Ma, Y., et al.: Learning to iteratively solve routing problems with dual-aspect collaborative transformer. In: Ranzato, M., Beygelzimer, A., Dauphin, Y., Liang, P.S., et al. (eds.) Proceedings of the 35th International Conference on Neural Information Processing Systems, NIPS 2021, Curran Associates Inc., Red Hook (2024)
28. Miller, J.F., et al.: An empirical study of the efficiency of learning Boolean functions using a cartesian genetic programming approach. In: Banzhaf, W., Daida, J.M., Eiben, A.E., Garzon, M.H., Honavar, V. (eds.) Proceedings of the Genetic and Evolutionary Computation Conference, pp. 1135–1142. Morgan Kaufmann (1999)
29. Miller, J.F.: Cartesian genetic programming: Its Status and Future. Genet. Program Evolvable Mach. **21**(1–2), 129–168 (2020)
30. Miller, J.F., Harding, S.L.: Cartesian genetic programming. In: Keijzer, M., Antoniol, G., Bates Congdon, C., Deb, K., et al. (eds.) Proceedings of the 10th Annual Conference Companion on Genetic and Evolutionary Computation, pp. 2701–2726. ACM Press (2008)

31. Oltean, M., Groşan, C., Oltean, M.: Encoding multiple solutions in a Linear Genetic Programming chromosome. In: Bubak, M., Albada, G.D., Sloot, P.M.A., Dongarra, J. (eds.) International Conference on Computational Science, pp. 1281–1288. Springer (2004)
32. Poli, R., Langdon, W.B.: Schema theory for genetic programming with one-point crossover and point mutation. Evol. Comput. **6**(3), 231–252 (1998). https://doi.org/10.1162/evco.1998.6.3.231
33. Rudolph, G.: Globale optimierung mit parallelen Evolutionsstrategien. Ph.D. thesis, Department of Computer Science, University of Dortmund (1990)
34. Shem-Tov, E., Elyasaf, A.: Deep neural crossover: A multi-parent operator that leverages gene correlations. In: Handl, J., Li, X., Wagner, M., Garza-Fabre, M., et al. (eds.) Proceedings of the Genetic and Evolutionary Computation Conference, pp. 1045–1053 (2024)
35. Shem-Tov, E., Sipper, M., Elyasaf, A.: Deep learning-based operators for evolutionary algorithms. In: Winkler, S.M., Banzhaf, W., Hu, T., Lalejini, A. (eds.) Genetic Programming Theory and Practice XXI. Springer (2025)
36. da Silva, J.E., Bernardino, H.S.: Cartesian genetic programming with crossover for designing combinational logic circuits. In: 2018 7th Brazilian Conference on Intelligent Systems (BRACIS), pp. 145–150. IEEE Press (2018)
37. Suganuma, M., Shirakawa, S., Nagao, T.: Designing convolutional neural network architectures using cartesian genetic programming, pp. 185–208. Springer, Singapore (2020). https://doi.org/10.1007/978-981-15-3685-4_7
38. Torabi, A., Sharifi, A., Teshnehlab, M.: Using cartesian genetic programming approach with new crossover technique to design convolutional neural networks. Neural Process. Lett. **55**(5), 5451–5471 (2023)
39. Turner, A.J.: Improving crossover techniques in a genetic program. Master's thesis, Department of Electronics, University of York (2012)
40. Wilson, G., Banzhaf, W.: A comparison of Cartesian Genetic Programming and Linear Genetic Programming. In: O'Neill, M., Vanneschi, L., Gustafson, S., Esparcia Alcázar, A.I., et al. (eds.) Genetic Programming: Proceedings of the 11th European Conference, EuroGP 2008, Naples, Italy, pp. 182–193. Springer (2008)