Improving Evaluation of Recombination-based Cartesian Genetic Programming

Duy Long Tran
Chair for AI Methodology,
RWTH Aachen University
Aachen, Germany
duy.long.tran@rwth-aachen.de

Anja Jankovic
Chair for AI Methodology,
RWTH Aachen University
Aachen, Germany
jankovic@aim.rwth-aachen.de

Marie Anastacio
Chair for AI Methodology,
RWTH Aachen University
Aachen, Germany
anastacio@aim.rwth-aachen.de

Holger Hoos
Chair for AI Methodology,
RWTH Aachen University
Aachen, Germany
hh@aim.rwth-aachen.de

Roman Kalkreuth
Chair for AI Methodology,
RWTH Aachen University
Aachen, Germany
kalkreuth@aim.rwth-aachen.de

ABSTRACT

Cartesian Genetic Programming has traditionally been using mutation as its main and often sole genetic operator to drive evolutionary search. Despite advancements in recent years, recombination-based approaches have long been avoided, due to apparent lack of performance gains. This study examines two recently suggested recombination-based operators, subgraph crossover and discrete phenotypic recombination on SRBench, a benchmarking platform for symbolic regression. Using the implementations provided in the TinyverseGP framework, we perform hyperparameter optimisation of the respective representations with these two operators. Our work demonstrates that hyperparameter optimisation can lead to improvements in performance for recombination-based Cartesian Genetic Programming.

KEYWORDS

Genetic Programming, Benchmarking, Hyperparameter Optimisation, Symbolic Regression

ACM Reference Format:
Duy Long Tran, Anja Jankovic, Marie Anastacio, Holger Hoos, and Roman Kalkreuth. 2026. Improving Evaluation of Recombination-based Cartesian Genetic Programming. In *Genetic and Evolutionary Computation Conference (GECCO Companion '26), July 13–17, 2026, San Jose, Costa Rica.* ACM, New York, NY, USA, 7 pages. https://doi.org/10.1145/3795101.3814688

1 INTRODUCTION

Genetic Programming (GP), originally introduced by Cramer [5] and Dickmanns et al. [6], is a search paradigm for program synthesis inspired by biological evolution which evolves computer programs by means of natural selection [15]. The selection pressure is implemented in the form of a fitness function that guides the search for optimised programs and a population of so-called individuals changes gradually under this selective pressure by means of genetic operators that introduce, delete or mix genetic material. The two most prominent operators are mutation and crossover.

Cartesian Genetic Programming (CGP) is a form of GP that encodes programs as directed acyclic graphs, using a grid-based integer representation, where nodes are arranged in columns and rows and evolution selects both the graph topology and the functions computed at each node. Because since the introduction of CGP crossover techniques have been shown to rather hinder than benefit search performance [4], mutation, especially point mutation, has established itself as the main genetic operator for CGP [20]. Recent work has shown that certain forms of crossover that are adapted to CGP are indeed able to beneficially influence the search performance of CGP at least to some extend (see, *e.g.,* [11, 13]). However, recombination in CGP has been demonstrated to be disruptive when compared to recombination in linear-based GP [14]. Furthermore, even if the effectiveness of crossover in CGP has been comprehensively studied quite recently [14], we believe that those studies can be enhanxced in terms of tuning the hyperparameters of the respective methods, in order to ensure that they reach their full performance potential.

We present an implementation and evaluation of recombination techniques for CGP that have been proposed by Kalkreuth et al. [11, 13] and yielded promising results in preliminary studies. We have implemented the proposed crossover methods, subgraph crossover and discrete phenotypic recombination, in the TinyverseGP benchmark framework [12]. Here, we investigate whether the optimisation of CGP hyperparameters included in TinyverseGP provides any benefits over manually configured hyperparameters for recombination-based CGP. To this end, we conducted an empirical evaluation on five different datasets from SRBench.

Our empirical results suggest that hyperparameter optimisation can lead to better results when compared to baseline configurations for recombination-based CGP. Interestingly, the positive impact of hyperparameter optimisation has been most pronounced if any type of crossover is used at all, with mutation-only CGP exhibiting almost identical performance, irrespective of whether hyperparameters were tuned or not. Additionally, we compared the performance of CGP equipped with the subgraph crossover technique to the

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.
GECCO Companion '26, July 13–17, 2026, San Jose, Costa Rica
© 2026 Association for Computing Machinery.
ACM ISBN 979-8-4007-2488-6/2026/07…$15.00
https://doi.org/10.1145/3795101.3814688

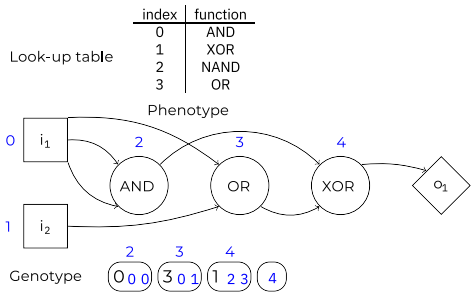


Figure 1: Exemplification of the CGP encoding.

performance of CGP with discrete phenotypic recombination, including as a baseline the mutation-only CGP variant (*i.e.,* without crossover). We observed that the median performance of discrete phenotypic recombination is better than that of subgraph crossover when configured for the symbolic regression problems that we considered. While previous approaches have relied mostly on rules of thumb, convention, and personal experience to configure hyperparameters [11, 16, 20], this is not suitable to determine whether a new representation, variant or genetic operator generally has better performance than another, which motivates our work.

The remainder of this paper is structured as follows: in Section 2, we briefly describe CGP, symbolic regression as our use-case, hyperparameter optimisation and the GP benchmarking framework TinyverseGP. The experiments are described in Section 3, followed by the presentation of the corresponding results in Section 4. Finally, in Section 5, we provide concluding remarks. In addition, an overview of the implementation of the recombination operators in TinyverseGP is presented in the appendix.

2 PRELIMINARIES

We first introduce the methods employed in this work: CGP and the crossover operators we implemented, the symbolic regression problem which we solve with it, the hyperparameter optimisation method we applied on it, and the benchmarking framework we used.

2.1 Cartesian Genetic Programming

CGP, as introduced by Miller and Thompson [21], is a graph-based variant of GP. It makes a distinction between *genotype* and *phenotype* regarding its encoding of computer programs. The genotype encodes the program in terms of a list of integers that describe the functions and connections of the corresponding graph.

The genotype is usually decoded in a backward manner, starting from the output nodes and following the successive input sources to determine the *active nodes* of the program. The result is the actual program encoded by the genotype: the phenotype. Nodes that are not traversed in this process and their corresponding genes are called *non-coding*. An illustrative example of the CGP encoding is given in Figure 1.

We consider two types of crossover in this work. **Subgraph crossover** has been proposed by Kalkreuth et al. [13]; it takes two parents and swaps their genetic material while respecting the active nodes of each parent thus preserving some of their phenotypic properties. **Discrete phenotypic recombination**, also called discrete phenotypic crossover or discrete recombination, has shown promising results when employed as genetic operator in CGP [11]. The method works on two parent individuals and swaps genetic material with a focus on active nodes and creates two offspring individuals.

2.2 Symbolic Regression

According to the definition by Kronberger et al. [16] and La Cava et al. [3], Symbolic Regression (SR) is a type of regression analysis which aims to find a model

$$\hat{y}(\mathbf{x}) = \hat{\phi}(\mathbf{x}, \hat{\theta}) : \mathbb{R}^d \mapsto \mathbb{R}$$

for features $\mathbf{x} \in \mathbb{R}^d$, where the distance between the predicted value $\hat{y}(\mathbf{x})$ and the target value y is as small as possible. This is achieved by continuously altering the parameters and structure of the symbolic representation of a mathematical expression. An illustrative example of such an expression is $10/(5 + (x-3)^2 + (y-3)^2 + (z-3)^2 + (v-3)^2 + (w-3)^2)$, which is also known as Vladislavleva-4 [27], or Nguyen-11: x^y [22]. Symbolic regression problems are often tackled using GP, where different types of error between the predicted value and the target value, *e.g.,* mean squared error (MSE), are used to assess the fitness of the model.

2.3 Hyperparameter Optimisation

Hyperparameter optimisation is the problem of finding a hyperparameter configuration for a machine learning algorithm that leads to optimal performance of that algorithm on the task at hand. It allows to automatically tailor a method, here CGP, to the problem instances of interest, here a symbolic regression task. Hyperparameter optimisation has been shown to be crucial for achieving strong performance across a wide range of algorithms and application domains [2, 17, 26]. In this work, we use SMAC3 [18], a state-of-the-art general-purpose algorithm configurator that uses *Bayesian optimisation* [7, 8] to efficiently search for optimal configurations, balancing exploration and exploitation within a given search space.

2.4 TinyverseGP Benchmarking Framework

TinyverseGP [12] is a Python benchmarking library for GP that incorporates several representations of GP such as CGP, tree-based GP, linear-based GP, and grammatical evolution. It provides a unified framework in which the performance of GP representations (using different measures) can be tested and compared to each other, using problems from several problem domains, including symbolic regression (one of the most prominent problem domains in GP [28]) as well as logic synthesis and policy learning. In particular, TinyverseGP contains interfaces to problems from SRBench [3], a benchmarking suite for symbolic regression problems, which is our use-case of interest in this work, and an interface to SMAC3 [18], which enables hyperparameter optimisation.

Concretely, in this work, the evaluated operators have been implemented in TinyverseGP and required minor adjustments. The pseudocode of the functions `subgraph_crossover` and `discrete_recombination` can be found in Appendix A, and implementation details and necessary modifications are described in Appendix B.

Dataset	# Variables	# Observations
192_vineyard	3	52
210_cloud	6	108
579_fri_c0_250_5	6	250
650_fri_c0_500_50	20	500
678_visualizing_environmental	4	11

Table 1: Datasets from SRBench used in our experiments and their characteristics.

Hyperparameter	Value
population_size	50
levels_back	100
mutation_rate	0.1
cx_rate	0.7
tournament_size	4
num_function_nodes	100 or 200

Table 2: Baseline hyperparameters for the evaluation runs.

3 EXPERIMENTS

This section provides details on the experiments we have conducted in order to answer the following research questions:
RQ 1: What is the impact of hyperparameter optimisation on the performance of recombination-based CGP?
RQ 2: How does the performance of CGP with subgraph crossover compare to that of CGP with discrete phenotypic recombination?

To answer these questions, we performed hyperparameter optimisation using SMAC3 on black-box symbolic regression problems. We used five datasets from the Penn Machine Learning Benchmarks (PMLB) collection of datasets [23, 24]. We then evaluated recombination-based CGP using both manually chosen baseline hyperparameter configurations and the hyperparameter configurations optimised by SMAC3.

3.1 Setup of Experiments

A canonical evolutionary algorithm equipped with tournament selection was adapted to run recombination-based CGP. The algorithm received hyperparameters such as population size, size of the tournament, crossover and mutation probabilities, which defined the hyperparameter-space considered by SMAC3. The HPO scenario for SMAC3 used for the experiments is available in our repository[1].

For each dataset, crossover operator and hyperparameter values, we ran CGP 30 times, in order to obtain a distribution of values and report statistics on this distribution as final performance.

We measured the fitness of a CGP run on a symbolic regression problem as the error between the predicted value of the evolved expression and the target value of the expression; specifically, we used MSE as a minimisation objective.

3.2 Datasets

We used five datasets from the Penn Machine Learning Benchmarks collection (PMLB) [23, 24]. The characteristics, such as number of variables and observations, are shown in Table 1. The selected datasets are part of SRBench [3]. Following [9], we chose small datasets with less than 500 observations and real-valued target functions, *i.e.,* we excluded integer-valued targets and large datasets.

3.3 Baseline Runs

As a baseline, we performed two separate sets of CGP runs with manually configured hyperparameters. The manually chosen values for the hyperparameters can be seen in Table 2. The sets differ in the number of nodes, which is set to 100 and 200, respectively, with all other hyperparameters fixed. They are based on studies conducted in the context of the introduction of the subgraph crossover and discrete phenotypic recombination operators, where the new operators have also been tested on problems from the domain of symbolic regression [10, 11, 13]. A population size of 50 has been determined in the meta-evolution process using the Java Evolutionary Computation Research (ECJ) tools [19, 25] for all symbolic regression problems for discrete recombination [11] and for many of the symbolic regression problems in the study on subgraph crossover [10].

We used the same fixed pseudo-random number seeds for the baseline runs and for the configured runs, to obtain the same training and testing sets for a fair comparison and to ensure reproducibility. We evolved the model on the training set and then took the best individual to calculate for further evaluation on the held-out testing set.

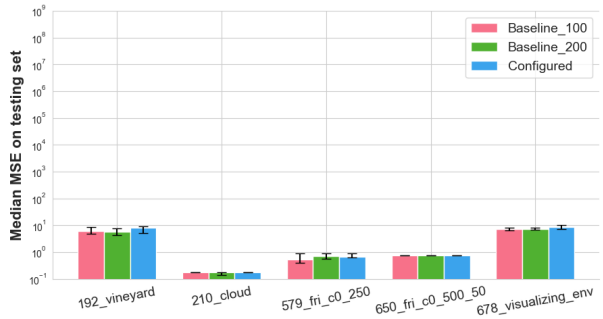
3.4 Hyperparameter Optimisation Runs

We optimised the hyperparameters listed in Table 2. For each dataset, we split the data into a training and testing set using a fixed seed, where the training and testing sets make up 75% and 25% of the dataset, respectively. We tuned the hyperparameters of CGP using SMAC3 on the training set via 5-fold cross-validation with a budget of 200 trials. For reproducibility, the 5-fold cross-validation was seeded with a fixed value for all runs. We optimised for the mean of the fitness values of the best individual on the validation set of each iteration of the 5-fold cross-validation. For each dataset, we performed 10 independent runs of the hyperparameter optimisation procedure times with random seeds, each yielding one optimised hyperparameter configuration. We then performed 30 runs of of each of these 10 optimised hyperparameter configurations for each crossover approach, with seeds in $\{1, 2, \ldots, 10\}$. For all runs, a seed with value 42 was used for the CGP and for the split of the dataset.

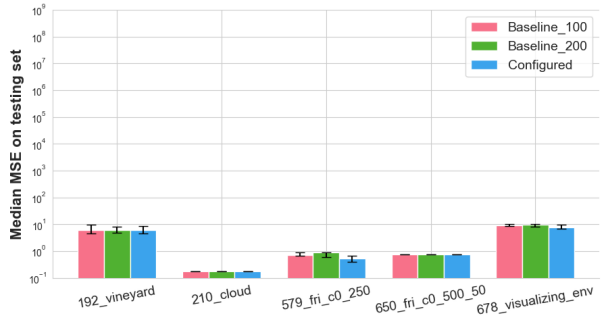
4 RESULTS

To answer **RQ 1**, Figure 2 shows the median fitness (in terms of MSE) of the best hyperparameter configuration found by SMAC3 against the two baseline configurations (100 and 200 nodes) across all five datasets. The error bars indicate the first and third quartiles of the underlying distribution. For both discrete recombination and subgraph crossover, the optimised configuration consistently matches or outperforms at least one of the baselines across all datasets. The benefit of hyperparameter optimisation is particularly visible on datasets such as `579_fri_c0_250` for discrete recombination and `678_visualizing_environmental` for subgraph crossover, where the optimised configurations achieve substantially

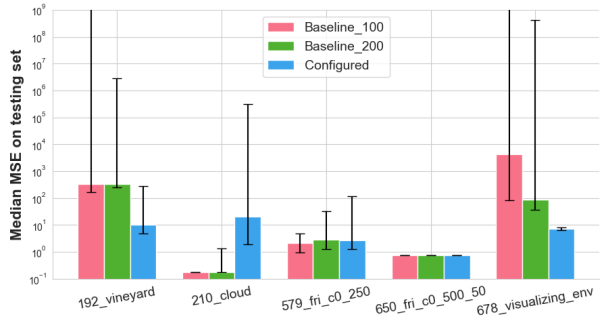
[1] https://github.com/GPBench/TinyverseGP/tree/recombination-based-cgp/src/hpo/hpo_cgp_sr.py



(a) Mutation-only



(b) Discrete recombination



(c) Subgraph crossover

Figure 2: Median fitness of the best found configuration against our baselines. Error bar shows the first and third quartiles. The truncated error bars from `192_vineyard` and `678_visualizing_env` are reaching $1.18 \cdot 10^{11}$ and $1.55 \cdot 10^{19}$, respectively.

lower MSE values than either baseline. These results suggest that hyperparameter optimisation can meaningfully improve the performance of recombination-based CGP beyond what is achievable with manually chosen configurations.

To answer **RQ 2**, Table 3 shows the median MSE of the best configurations found for each operator across our five datasets and highlights the median statistically tied with the best configuration according to a Mann-Whitney U-test with a significance level of 0.05. Discrete recombination achieves lower MSE on four out of the five datasets, with subgraph crossover only performing comparably on the `678_visualizing_environmental` dataset. Overall, discrete phenotypic recombination demonstrates stronger median performance on the symbolic regression problems considered here. In

Dataset	Discrete Recombination		Subgraph Crossover		Mutation-only	
	median	Q1 Q3	median	Q1 Q3	median	Q1 Q3
`192_vineyard`	*6.20*	4.62 8.78	10.55	4.82 284.83	*8.31*	5.06 9.03
`210_cloud`	*0.18*	0.18 0.18	20.34	1.89 314882.07	*0.18*	0.18 0.18
`579_fri_c0_250`	*0.54*	0.40 0.68	2.62	1.28 114.59	0.68	0.63 0.88
`650_fri_c0_500_50`	*0.74*	0.74 0.74	*0.74*	0.74 0.74	*0.74*	0.74 0.74
`678_visualizing_env`	*7.69*	6.95 9.42	*7.26*	6.60 8.23	8.45	7.44 10.14

Table 3: Median performance (in terms of MSE) of the configured CGP models on each dataset (lower is better) with first and third quartile. The best median is underlined, and the medians statistically tied with the best, according to a Mann-Whitney U-test with a significance level of 0.05 are in italic.

comparison to mutation-only CGP, the considered recombination-based approaches overall achieved better or at least equal median performance, while discrete recombination showed more robust performance. Additionally, the variability in performance of CGP is especially pronounced with subgraph crossover as an operator (Figure 2c), as the scale of achieved median performance values with subgraph crossover are up to two order of magnitude larger than the one with discrete recombination. (Figure 2b), with the third quartile being more than ten orders of magnitude larger on the `192_vineyard` and `678_visualizing_environmental` datasets. However, the configured mutation-only CGP, does not to differ much from the mutation-only baselines (Figure 2a).

5 CONCLUSIONS AND FUTURE WORK

We presented a first step towards improved evaluation of recombination-based CGP to facilitate fairer comparisons between operators and algorithms. We have reported results for recombination-based and mutation-only CGP that have been obtained by using a pipeline implemented in TinyverseGP, which enables hyperparameter optimisation using SMAC3 with 5-fold cross-validation and subsequent evaluation of these optimised hyperparameter settings. To ensure fairer comparison of genetic operators, our results demonstrate that hyperparameter optimisation should be used to find suitable configurations for the evaluated operators, and the comparison can be made based on the configurations with the best performance of each operator or method.

In future work, we plan to extend the proposed approach to other recombination operators from the literature, in order to evaluate CGP in the context of a larger study built upon best practices in benchmarking randomised search heuristics [1]. We also intend to carry out fitness landscape analysis to learn more about the optimisation conditions in GP model tuning. Since for some of the datasets we tested, the best found configuration performed worse than our baselines, we would naturally like to better understand the reasons and corresponding challenges.

REFERENCES

[1] T. Bartz-Beielstein, C. Doerr, D. van den Berg, J. Bossek, S. Chandrasekaran, T. Eftimov, A. Fischbach, P. Kerschke, W. La Cava, M. López-Ibáñez, K. M. Malan, J. H. Moore, B. Naujoks, P. Orzechowski, V. Volz, M. Wagner, and T. Weise. 2020. Benchmarking in Optimization: Best Practice and Open Issues. arXiv:2007.03488 [cs.NE]

[2] J. Bergstra and Y. Bengio. 2012. Random Search for Hyper-Parameter Optimization. *Journal of Machine Learning Research* 13 (2012), 281–305.
[3] W. G. La Cava, P. Orzechowski, B. Burlacu, F. O. de França, M. Virgolin, Y. Jin, M. Kommenda, and J. H. Moore. 2021. Contemporary Symbolic Regression Methods and their Relative Performance. In *Proceedings of the 35th International Conference on Neural Information Processing Systems (NeurIPS 2021) Track on Datasets and Benchmarks*.
[4] J. Clegg, J. A. Walker, and J. F. Miller. 2007. A New Crossover Technique for Cartesian Genetic Programming. In *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO'07)*. 1580–1587.
[5] N. L. Cramer. 1985. A Representation for the Adaptive Generation of Simple Sequential Programs. In *Proceedings of the 1st International Conference on Genetic Algorithms*. 183–187.
[6] D. Dickmanns, J. Schmidhuber, and A. Winklhofer. 1987. *Der genetische Algorithmus: Eine Implementierung in Prolog*. Fortgeschrittenenpraktikum. Institut für Informatik, Technische Universität München.
[7] P. I. Frazier. 2018. A Tutorial on Bayesian Optimization. *CoRR* abs/1807.02811 (2018). arXiv:1807.02811
[8] R. Garnett. 2023. *Bayesian Optimization*. Cambridge University Press.
[9] G. S Imai Aldeia, H. Zhang, G. Bomarito, M. Cranmer, A. Fonseca, B. Burlacu, W. G. La Cava, and F. O. de França. 2025. Call for Action: Towards the next generation of symbolic regression benchmarks. In *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO'25) Companion*. 2529–2538.
[10] R. Kalkreuth. 2020. A Comprehensive Study on Subgraph Crossover in Cartesian Genetic Programming. In *Proceedings of the 12th International Joint Conference on Computational Intelligence (IJCCI 2020)*. 59–70.
[11] R. Kalkreuth. 2022. Towards Discrete Phenotypic Recombination in Cartesian Genetic Programming. In *Proceedings of the International Conference on Parallel Problem Solving from Nature (PPSN'22)*. 63–77.
[12] R. Kalkreuth, F. O. de França, J. Dierkes, M. Anastacio, A. Jankovic, Z. Vasícek, and H. H. Hoos. 2025. TinyverseGP: Towards a Modular Cross-domain Benchmarking Framework for Genetic Programming. In *Proceedings of the Genetic and Evolutionary Computation Conference Companion (GECCO '25 Companion)*. 127–130.
[13] R. Kalkreuth, G. Rudolph, and A. Droschinsky. 2017. A New Subgraph Crossover for Cartesian Genetic Programming. In *Genetic Programming - 20th European Conference, EuroGP 2017, Amsterdam, The Netherlands, April 19-21, 2017, Proceedings*, Vol. 10196. 294–310.
[14] M. Kocherovsky and W. Banzhaf. 2024. Crossover Destructiveness in Cartesian versus Linear Genetic Programming. In *Proceedings of the Artificial Life Conference (ALIFE 2024)*, Vol. ALIFE 2024: Proceedings of the 2024 Artificial Life Conference.
[15] J. R. Koza. 1992. Genetic Programming: On the Programming of Computers by Means of Natural Selection. *MA: MIT Press.* (1992), 1–840.
[16] G. Kronberger, B. Burlacu, M. Kommenda, S. M. Winkler, and M. Affenzeller. 2024. Basics of Symbolic Regression. In *Symbolic Regression*. 21–34.
[17] Niklas Lavesson and Paul Davidsson. 2006. Quantifying the Impact of Learning Algorithm Parameter Tuning. In *Proceedings of the 21st National Conference on Artificial Intelligence(AAAI)*. 395–400.
[18] M. Lindauer, K. Eggensperger, M. Feurer, A. Biedenkapp, D. Deng, C. Benjamins, T. Ruhkopf, R. Sass, and F. Hutter. 2022. SMAC3: A Versatile Bayesian Optimization Package for Hyperparameter Optimization. *Journal of Machine Learning Research* 23 (2022), 1–9.
[19] S. Luke. 1998. ECJ Evolutionary Computation Library. http://cs.gmu.edu/$\sim$eclab/projects/ecj/
[20] J. F. Miller. 2011. Cartesian Genetic Programming. In *Cartesian Genetic Programming*. 17–34.
[21] J. F. Miller and P. Thomson. 2000. Cartesian Genetic Programming. In *Genetic Programming, European Conference, Edinburgh, Scotland, UK, April 15-16, 2000, Proceedings*, Vol. 1802. 121–132.
[22] Q. U. Nguyen, X. H. Nguyen, M. O'Neill, R. I. McKay, and E. Galván López. 2011. Semantically-based Crossover in Genetic Programming: Application to Real-Valued Symbolic Regression. *Genetic Programming and Evolvable Machines* 12, 2 (2011), 91–119.
[23] R. S. Olson, W. La Cava, P. Orzechowski, R. J. Urbanowicz, and J. H. Moore. 2017. PMLB: A Large Benchmark Suite for Machine Learning Evaluation and Comparison. *BioData Mining* 10, 1 (2017), 1–36.
[24] J. D. Romano, T. T. Le, W. La Cava, J. T. Gregg, D. J. Goldberg, P. Chakraborty, N. L. Ray, D. Himmelstein, W. Fu, and J. H. Moore. 2021. PMLB v1.0: An Open Source Dataset Collection for Benchmarking Machine Learning Methods. *arXiv preprint arXiv:2012.00058v2* (2021), 1–4.
[25] E. O. Scott and S. Luke. 2019. ECJ at 20: Toward a General Metaheuristics Toolkit. In *Proceedings of the Genetic and Evolutionary Computation Conference Companion, GECCO 2019, Prague, Czech Republic, July 13-17, 2019*. 1391–1398.
[26] J. Snoek, H. Larochelle, and R. P. Adams. 2012. Practical Bayesian Optimization of Machine Learning Algorithms. In *Proceedings of the 26th Conference on Advances in Neural Information Processing Systems (NeurIPS)*, Vol. 25.
[27] E. Vladislavleva, G. Smits, and D. den Hertog. 2009. Order of Nonlinearity as a Complexity Measure for Models Generated by Symbolic Regression via Pareto Genetic Programming. *IEEE Transactions on Evolutionary Computation* 13, 2 (2009), 333–349.
[28] D. R. White, J. McDermott, M. Castelli, L. Manzoni, B. W. Goldman, G. Kronberger, W. Jaskowski, U.-M. O'Reilly, and S. Luke. 2013. Better GP Benchmarks: Community Survey Results and Proposals. *Genetic Programming and Evolvable Machines* 14, 1 (2013), 3–29.

A PSEUDOCODE

Algorithm 1: Subgraph crossover

Input: Genome of the first parent P_1, genome of the second parent P_2
Output: Offspring genome O

```
/* Store the active nodes of each parent in arrays                                   */
```
$M_1 \leftarrow \text{ActiveNodes}(P_1)$
$M_2 \leftarrow \text{ActiveNodes}(P_2)$
```
/* Choose crossover point for each parent                                            */
```
$C_{P_1} \leftarrow \text{RandomChoice}(M_1)$
$C_{P_2} \leftarrow \text{RandomChoice}(M_2)$
```
/* Choose general crossover point to be the minimum of the two crossover points C_P1 and C_P2   */
```
$C_P \leftarrow \text{Min}(C_{P_1}, C_{P_2})$
```
/* Copy the genomes of the parents                                                   */
```
$O \leftarrow \text{CopyGenome}(P_1, \text{start}, C_P)$
$O \leftarrow \text{CopyGenome}(P_2, C_P + 1, \text{end})$
```
/* Perform neighbourhood connect                                                     */
```
if *NumberOfOutputs* = 1 **then**
    ```
    /* Connect first connection gene behind C_P to last active node of the subgraph in front of C_P   */
    ```
    $O[\text{NodePosition}(\text{FirstActive} + 1)] \leftarrow \text{NodePosition}(\text{LastActive})$
```
/* Perform random active connect                                                     */
```
for *gene in genome of the offspring behind* C_P **do**
    if *gene is connection gene* **then**
        gene ← RandomChoice(Inputs, genes in columns in front of current column)
if $NumberOfOutputs > 1$ **then**
    for *gene in output genes of the offspring* **do**
        gene ← RandomChoice(Inputs, function nodes)

Algorithm 2: Discrete phenotypic recombination (adjusted from [11])

Input: Genomes of the parents G_1, G_2
Output: Genomes of the offspring $\tilde{G}_1, \tilde{G}_1$
/* Lists storing the active node numbers of each parent */
$M_1 \leftarrow \text{ActiveNodes}(G_1)$
$M_2 \leftarrow \text{ActiveNodes}(G_2)$
/* Determine the min and max number of active function nodes */
$\text{min} \leftarrow \text{Min}(\text{len}(M_1), \text{len}(M_2))$
$\text{max} \leftarrow \text{Max}(\text{len}(M_1), \text{len}(M_2))$
$i \leftarrow 0$
while $i < 0$ **do**
  /* Randomly choose to either swap genes or not */
  if *RandomBoolean() = true* **then**
    if $i = min - 1$ ***and*** $len(M_1) \neq len(M_2)$ **then**
      /* Boundary extension */
      $r \leftarrow \text{RandomInteger}(0, \text{max} - i)$
      if $len(M_1) < len(M_2)$ **then**
        $n_1 \leftarrow N_1[i]$
        $n_2 \leftarrow N_2[i + r]$
      else
        $n_1 \leftarrow N_1[i + r]$
        $n_2 \leftarrow N_2[i]$
    else
      /* No boundary extension */
      $n_1 \leftarrow N_1[i]$
      $n_2 \leftarrow N_2[i]$
    $p_1 \leftarrow \text{NodePosition}(n_1)$
    $p_2 \leftarrow \text{NodePosition}(n_2)$
    $\tilde{G}_1, \tilde{G}_1 \leftarrow \text{SwapGenes } (G_1, G_2, p_1, p_2)$
  $i \leftarrow i + 1$

B IMPLEMENTATION DETAILS

The operators evaluated in this work have been implemented in the TinyverseGP framework and required minor adjustments described below. The code will be included in the benchmark for future use by the community. The pseudocode of the functions `subgraph_crossover` and `discrete_recombination` can be found in the appendix. The extended version of `tiny_cgp` is provided in our repository[2].

Subgraph crossover: For the subgraph crossover operator, we made some adjustments to the original description by Kalkreuth et al. [13] to fit the broader module of `tiny_cgp`. The implemented method takes the genomes `genome1` and `genome2` of parents P_1 and P_2 as input (as opposed to the parent individuals as a whole). The genomes are represented as lists of type `int` and the function returns an object called `CGPIndividual` that contains the genotype of the generated offspring. The first step is analogous to the original definition and assigns the node numbers of the active nodes of parents P_1 and P_2 to lists `M_1` and `M_2`, respectively.

Discrete recombination: The implementation of discrete phenotypic recombination largely follows the instructions that are lined out in the pseudocode given in the original paper by Kalkreuth [11], with some adjustments to fit the `tiny_cgp` module. Similar to the subgraph crossover method and the original algorithm description, discrete recombination takes as input the genomes of the parents in the form of lists of integers, but returns a tuple of two CGP genotypes for the two offspring.

[2]https://github.com/GPBench/TinyverseGP/tree/recombination-based-cgp/src/gp/tiny_cgp.py