Evolvability degeneration in multi-objective genetic programming for symbolic regression

Liu, Dazhuang; Virgolin, Marco; Alderliesten, Tanja; Bosman, Peter A.N.

DOI
10.1145/3512290.3528787

Publication date
2022

Document Version
Final published version

Published in
GECCO 2022 - Proceedings of the 2022 Genetic and Evolutionary Computation Conference

Citation (APA)
Liu, D., Virgolin, M., Alderliesten, T., & Bosman, P. A. N. (2022). Evolvability degeneration in multi-objective genetic programming for symbolic regression. In *GECCO 2022 - Proceedings of the 2022 Genetic and Evolutionary Computation Conference* (pp. 973-981). (GECCO 2022 - Proceedings of the 2022 Genetic and Evolutionary Computation Conference). ACM. https://doi.org/10.1145/3512290.3528787

Evolvability Degeneration in Multi-Objective Genetic Programming for Symbolic Regression

Dazhuang Liu
Marco Virgolin
dazhuang.liu@cwi.nl
marco.virgolin@cwi.nl
Centrum Wiskunde & Informatica
Amsterdam, the Netherlands

Tanja Alderliesten
t.alderliesten@lumc.nl
Leiden University Medical Center
Leiden, the Netherlands

Peter A. N. Bosman
peter.bosman@cwi.nl
Centrum Wiskunde & Informatica
Amsterdam, the Netherlands
Delft University of Technology
Delft, the Netherlands

ABSTRACT

Genetic programming (GP) is one of the best approaches today to discover symbolic regression models. To find models that trade off accuracy and complexity, the non-dominated sorting genetic algorithm II (NSGA-II) is widely used. Unfortunately, it has been shown that NSGA-II can be inefficient: in early generations, low-complexity models over-replicate and take over most of the population. Consequently, studies have proposed different approaches to promote diversity. Here, we study the root of this problem, in order to design a superior approach. We find that the over-replication of low complexity-models is due to a lack of evolvability, i.e., the inability to produce offspring with improved accuracy. We therefore extend NSGA-II to track, over time, the evolvability of models of different levels of complexity. With this information, we limit how many models of each complexity level are allowed to survive the generation. We compare this new version of NSGA-II, *evoNSGA-II*, with the use of seven existing multi-objective GP approaches on ten widely-used data sets, and find that evoNSGA-II is equal or superior to using these approaches in almost all comparisons. Furthermore, our results confirm that evoNSGA-II behaves as intended: models that are more evolvable form the majority of the population.
Code: https://github.com/dzhliu/evoNSGA-II

CCS CONCEPTS

• **Computing methodologies → Genetic programming**; • **Applied computing → Multi-criterion optimization and decision-making**.

KEYWORDS

Symbolic regression, genetic programming, multi-objective optimization, evolvability

ACM Reference Format:
Dazhuang Liu, Marco Virgolin, Tanja Alderliesten, and Peter A. N. Bosman. 2022. Evolvability Degeneration in Multi-Objective Genetic Programming for Symbolic Regression. In *Genetic and Evolutionary Computation Conference (GECCO '22), July 9–13, 2022, Boston, MA, USA.* ACM, New York, NY, USA, 9 pages. https://doi.org/10.1145/3512290.3528787

GECCO '22, July 9–13, 2022, Boston, MA, USA
© 2022 Copyright held by the owner/author(s).
ACM ISBN 978-1-4503-9237-2/22/07.
https://doi.org/10.1145/3512290.3528787

1 INTRODUCTION

In recent years, we are seeing a renewed interest in symbolic regression (SR), the sub-field of machine learning (ML) which concerns searching for ML models in the form of mathematical expressions [7, 8, 26, 34]. These models are appealing because, by their very nature, they stand a chance of being interpretable. This is increasingly considered important, e.g., to ensure that ML is used in a fair and responsible manner [1, 12, 17].

Today, genetic programming (GP) [16] is one of the best approaches to discover SR models [7]. GP is a bio-inspired metaheuristic that works by *evolving* a population of solutions that, differently from traditional genetic algorithms, need be *executed* to be evaluated, i.e., they are programs. In the case of SR, the solutions evolved by GP encode functions as symbolic models that are evaluated in terms of their accuracy in fitting a (training) data set [16]. However, when maximizing accuracy alone, GP tends to generate solutions that become unnecessarily large in the number of components (arithmetic operations, variables, constants, etc.), a phenomenon known as *bloat*, which harms interpretability [20].

To deal with this problem, GP can be set to optimize different objectives at the same time. Multi-objective GP (MOGP) is typically used with the intention to search for solutions with different trade-offs between accuracy and interpretability [15]. At the end of a single run of MOGP, decision makers can choose the model that strikes the right balance between accuracy and interpretability. Since interpretability is hard or impossible to define (in general terms) [19, 30], the common way by which interpretability in pursued in MOGP for SR is by minimization of solution size (or derivations thereof, see e.g., the related work section in [29]), i.e., the number of components that constitutes the solution. Minimizing size is typically in conflict with maximizing accuracy in (MO)GP, because (MO)GP typically discovers better solutions by refining the function approximation they represent, i.e., by incorporating additional components [18].

The second version of the non-dominated sorting genetic algorithm [10] (NSGA-II) is the most adopted framework to realize MOGP. Unfortunately, as it has been shown by several works before [3, 9, 32, 33] and is confirmed once more in this paper, NSGA-II can be inefficient when adopted for MOGP when one of the objectives is solution size. In particular, small solutions are observed to take over the majority of the population in a few generations, while larger and more accurate solutions are hardly discovered.

In this paper, we tackle this problem at its root. Specifically, we identify that the reason why small solutions over-replicate and hamper the discovery of larger but more accurate solutions is the fact that, besides obviously minimizing size well and thus having high chances of survival, small solutions lack *evolvability*. Here, by evolvability of a solution we mean the likelihood that variation (e.g., subtree crossover and mutation) produces a relatively accurate offspring when using that solution as a parent. We call this cause of inefficiency of NSGA-II *evolvability degeneration*. Consequently, we present a new algorithm, named *evoNSGA-II*, which improves upon standard NSGA-II by restraining the over-replication of solutions whose size is identified to be unhelpful in terms of discovering more accurate solutions. Thanks to this, we find evoNSGA-II to be far more efficient than NSGA-II as well as other algorithms designed to deal with this issue.

2 BACKGROUND & RELATED WORK

2.1 Brief recall on SR and (MO)GP

In SR, we seek a model (or equivalently, function approximation) f that is accurate in terms of fitting a given data set. Accuracy is typically measured in terms of minimizing a loss function, such as the mean-squared-error (MSE). Formally, given a data set $\mathcal{D} = \{(\mathbf{x}_i, y_i)\}_{i=1}^{n}$, where n is the number of observations, $\mathbf{x}_i \in \mathbb{R}^d$ is the vector of d feature values $\mathbf{x}_i = \left(x_i^{(1)}, \ldots, x_i^{(d)}\right)^\top$, and $y_i \in \mathbb{R}$ the label or target variable, we seek an optimal $f^\star$ such that:

$$f^\star := \underset{f \in F}{\operatorname{argmin}} \{\text{MSE}(\mathcal{D}, f)\} = \underset{f \in F}{\operatorname{argmin}} \left\{ \frac{\sum_{i=1}^{n} (y_i - f(\mathbf{x}_i))^2}{n} \right\}.$$

An SR algorithm searches in the space of functions F that is defined in terms of an *encoding* (see next paragraph), and what atomic sub-functions (+, −, ×, ÷, exp, log, etc.), variables ($x^{(1)}$, $x^{(2)}$, etc.), and constants ($\frac{1}{2}$, $-\pi$, 42, etc.) appear in what order in that encoding. Alongside maximizing accuracy, we wish the model to be interpretable. Various metrics have been proposed to seek interpretable/simpler models, see e.g., [29, 31]. However, reducing model size remains a simple and popular approach (e.g., it was recently used in a large SR benchmark [7]).

GP is a popular and often top-performing method for SR [7]. In this work, we adopt traditional GP, where solutions are encoded by trees in which each node contains one of the possible sub-functions, variables, and constants [16, 24]. To discover of multiple solutions with trade-offs between accuracy and interpretability, GP is set to work in a multi-objective fashion (MOGP), where the concept of Pareto-dominance is used to rank solutions. Specifically, we say that solution A Pareto-dominates solution B if A is *equal or better* than B in all objectives, and strictly better in at least one objective. The outcome of MOGP is the best-found *front*, i.e., the set of solutions that are not Pareto-dominated by any other ever found.

NSGA-II is widely considered to be the most popular multi-objective evolutionary algorithm (MOEA). We conducted a small literature survey to assess whether this is indeed the case for MOGP. We detail how the survey was conducted in the supplementary material. We found that, in the last five years, NSGA-II was typically adopted as MOGP algorithm in approximately 70% of the works that we surveyed, either as the main algorithm or as a baseline. We thus believe that our intent of improving NSGA-II for MOGP is amply justified.

2.2 Prior works on improving NSGA-II for GP

Several works in the literature have identified the problem of small solutions over-replicating and hampering further evolution, which we refer to as *evolvability degeneration*. A very-closely related concept was discovered almost twenty years ago in [9], and termed later as *population collapse* [3]. Population collapse refers to the process where the entire population converges to copies of a single solution that has a single component, i.e., the population is unable to evolve any further. As it will be shown in this paper (in Sec. 3), the behavior we observe is less extreme: even though copies of small solutions do initially occupy most of the population in early generations, NSGA-II remains able to recover, i.e., larger solutions are discovered later on, albeit at a very slow rate. To prevent population collapse, the use of a diversity preservation mechanism is advised in [9]. Instead, in [3] it is argued that employing mutation is enough. Here, we find that even if one employs mutation, NSGA-II still suffers from evolvability degeneration.

Other works have also noted, and proposed means to deal with, the problem of small solutions flooding the population. In [4], it is proposed to use SPEA2 for MOGP[35], to overcome the problem just mentioned as well as bloat. SPEA2, which we also consider in our experiments, works in a fundamentally different way than NSGA-II. For example, SPEA2 maintains two separate populations during the search, and measures the performance of a solution based on how many solutions are dominated by that solution.

Recently, [32] and [33] explored the idea of using α-dominance. Instead of the original objectives (here, accuracy and size), these algorithms use linear combinations of the original objectives which are weighted by coefficients (α) that vary over time, so as to be able to put more pressure on finding solutions of a certain trade-off. In particular, α is adapted to increase the importance of accuracy over the importance of size. In the first work [32], fixed schedules are considered to adapt α, according to a function of the number of generations that is linear, a cosine, or a sigmoid. In the second work [33], α is adapted dynamically based on the state of the population: if more small than accurate (and vice versa, accurate than small) solutions are detected, then α is adapted to give more weight to accuracy (respectively, to size).

For NSGA-II applied to discrete optimization, in [13] strategies are explored to remove duplicate solutions from the population. One such strategy is used for MOGP in [30], where NSGA-II is modified so that duplicate solutions are assigned the lowest priority to survive selection. Together with classic NSGA-II, SPEA2, and the α-dominance based algorithms, we also include this algorithm in our comparisons.

To the best of our knowledge, our work differs from the previous ones because it makes an explicit link between the over-replication of small solutions and their lack of evolvability, and proposes an algorithm that uses this information to improve the search.

3 EVOLVABILITY DEGENERATION

In this section, we analyze the phenomenon of evolvability degeneration in NSGA-II for MOGP. First, we describe it by considering a use case. Then, we show what causes it. The latter is done by means of an experiment in which we trace how solutions of different sizes contribute to finding offspring solutions that are relatively accurate.

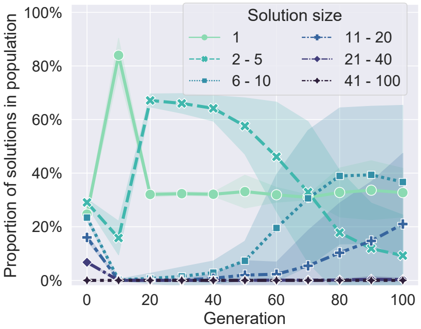

Figure 1: Proportion of solutions of different sizes during the evolution for 30 runs of NSGA-II on Airfoil. Lines indicate means and shaded areas represent standard deviations. Note the exponential scaling of solution size intervals.

3.1 Over-replication of small solutions

We begin by reporting how the size of solutions changes over time when using NSGA-II on an examplary use case. The parameter settings for NSGA-II are those in bold font in Table 1, except for the population size, which is set to 500. We show the behavior of NSGA-II on the data set Airfoil (see Sec. 2 of the supplementary material). We use this data set as a recurring example for no particular reason other than it being first in alphabetic order among the data sets we considered; we observe similar trends also on the other data sets.

Fig. 1 shows that, at the initial stages of the evolution, the proportion of small solutions grows to occupy the majority of the population. Only later small solutions start to diminish and slightly larger solutions start to appear and compete. However, the largest solutions, in this case the ones with more than 20 nodes, are basically not discovered. Importantly, the solutions of size one always occupy a rather large portion of the population (above 30%). This abundance of small solutions can be explained by the fact that, reasonably, small solutions of relatively high accuracy and duplicates thereof are produced by GP relatively quickly; in particular, before larger and more accurate solutions are discovered. Because of how NSGA-II works, solutions that have the best-so-far accuracy for any given size are set to survive the generation with high priority, no matter if they are duplicates or not. Now, this abundance of small solutions would not necessarily be a problem if small solutions would represent fertile grounds to discover larger, more accurate solutions. In the next section, we show that this is not the case.

3.2 Evolvability of small and large solutions

A simple way to understand whether evolution stagnates or proceeds well is to measure evolvability in terms of the frequency by which well-performing offspring solutions are discovered. Here, we particularly want to measure the frequency with which solutions of different size contribute to offspring solutions with an accuracy that is relatively high. Since we aim to improve NSGA-II, we would ideally do this within an NSGA-II evolution. However, as shown in Fig. 1, larger solutions are hardly ever discovered, making it impossible for us to estimate their evolvability. Thus, we design a workflow to collect enough solutions of various sizes.

First, we repeatedly run single-objective GP, 100 times, with different maximal size limitations, for up to a certain number of generations (e.g., 40). This allows us to collect best-found solutions of various sizes which are relatively accurate, and can be imagined to contribute to a best-found front at a certain stage of an "ideal" NSGA-II evolution, where evolvability degeneration does not occur. Second, we collect these solutions in different buckets, based on their size. We also record the 90^{th} percentile of accuracy (acc_{90}) out of all solutions collected, irrespective of their size; we use this information later. Next, for each bucket, we repeatedly (100 times) take a random solution to act as parent, and generate an offspring solution via subtree mutation. We do the same for subtree crossover, this time considering pair of buckets and, importantly, generating a *single* offspring instead of two (this is rather common in GP [24]). Specifically, the offspring is generated by cloning the first parent and transplanting a random subtree from the second parent (which from now on will be called *donor* to avoid confusion) to replace a random subtree of the first parent (which from now on will be simply called *parent*). We perform crossover this way because, for a sufficiently large parent, in expectation the majority of the nodes in the offspring comes from the parent instead of the donor; this may play an important role in terms of evolvability. Lastly, we measure how frequently parents of different sizes produce relatively accurate offspring, using acc_{90} as a threshold.

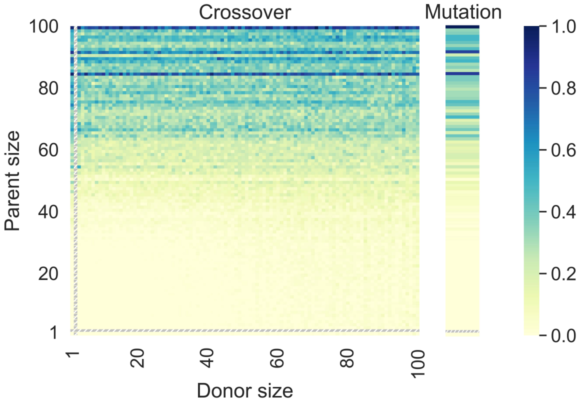

Figure 2: Frequency (normalized between min and max, color coded as depicted by the legend on the right) of producing an offspring with a good accuracy (above the 90^{th} percentile of those obtained in all runs) based on the size of the parent and donor solutions. Left: For subtree crossover. Right: For subtree mutation. Note that a solution of size 2 (to be used as a parent or donor) was never returned by single-objective GP because a more accurate solution of size 1 exists and was systematically discovered.

We apply the proposed workflow and display the result in Fig. 2 (all the parameter settings are as per Sec. 3.1), which concerns Airfoil and best-found solutions at generation 40. We remark that we repeated the same approach on the second data set we consider, Boston, as well as with other termination limits (generation 10, 20, and 30), and means of assessing whether an offspring is relatively accurate (e.g., with respect to the accuracy of the parent); we observed the same general trends as shown in Fig. 2. Note that the heat-map for subtree crossover is not symmetric due to the reasons explained in the previous paragraph. The frequencies found for subtree crossover indicate that the parent needs to be sufficiently large for variation to be successful with large probability, while the donor can be of any size. Similarly, also for mutation larger solutions are more evolvable.

The result just shown confirms our hypothesis that smaller solutions hamper the search. Therefore, the fact that in the early stages of an NSGA-II run, the population is flooded by copies of small solutions, is highly undesirable. We remark that penalizing duplicates altogether, as in fact was done in some earlier approaches (see Sec. 2.2), is not necessarily the optimal strategy. In fact, having duplicates of highly-evolvable solutions may be the best option. This idea is explored in our algorithm, presented in the next section.

4 IMPROVING NSGA-II BASED ON EVOLVABILITY

We now present our proposal to improve NSGA-II, i.e., evoNSGA-II. Since evoNSGA-II mostly follows NSGA-II, we begin by recalling the workings of NSGA-II. Next, we explain what is new in evoNSGA-II, i.e., the estimation of the evolvability of solutions of different size, and the use of this information to prevent the over-replication of solutions with low evolvability.

4.1 NSGA-II

Algorithm 1 shows the pseudo-code of NSGA-II, as well as that of evoNSGA-II: In fact, the only change we apply is regarded to how the population is updated at the end of a generation. In every generation of (evo)NSGA-II, firstly an offspring population O is generated from promising solutions of the current population $\mathcal{P}$. Promising solutions are typically chosen with tournament selection, and then undergo variation, typically by means of subtree crossover and subtree mutation. In (evo)NSGA-II, tournament selection compares solutions based on their non-domination *rank* (explained below) and, if the solutions share the same rank, based on their *crowding distance* (explained below too).

Next, $\mathcal{P}$ and O are merged and undergo non-dominated sorting. Non-dominated sorting is a process that subdivides all solutions into layers called *fronts*, such that for any two solutions in a same front, those two solutions do not Pareto-dominate each other; moreover, for each solution in the i^{th} front, there exists at least one solution in the $(i-1)^{th}$ front that Pareto-dominates it. The rank of a solution represents the front to which that solution belongs, rank 1 being the best. The algorithm proceeds by parsing each front and assigning to each solution in that front a *crowding distance.* The crowding distance is a measure of sparseness (the more a solution is isolated the better) that is computed in the objective space using the L1 norm. A solution for which an objective has the maximum value for that front is assigned an infinite (and thus best) crowding distance.

Finally, the population is updated for the next generation, using an NSGA-II-specific form of *truncation* selection. This is where NSGA-II and evoNSGA-II differ. In NSGA-II, the new population is formed by selecting the solutions with rank 1, then those with rank 2, and so on, until the selection of all solutions with a certain rank would result in exceeding the population size. In that case, the crowding distance is used to discern which subset of solutions with that certain rank still to select for the new population. The remaining solutions are discarded.

Algorithm 1 Workflow of NSGA-II and evoNSGA-II
Note: *Truncation* is the only step that is different between the two.

```
Require: Pop_size, stop_criteria
$\mathcal{P} \leftarrow$ INITIALIZE_POPULATION(Pop_size)
EVALUATE($\mathcal{P}$)
Fronts ← FAST_NON-DOMINATED_SORTING($\mathcal{P}$)
for front in Fronts do
    CROWDING_DISTANCE(front)
end for
while $\neg$ stop_criteria do
    $\mathcal{P}' \leftarrow$ TOURNAMENT($\mathcal{P}$)
    $O \leftarrow$ VARIATION($\mathcal{P}'$)
    EVALUATE($O$)
    Fronts ← FAST_NON-DOMINATED_SORTING($\mathcal{P} \cup O$)
    for front in Fronts do
        CROWDING_DISTANCE(front)
    end for
    $\mathcal{P} \leftarrow \boxed{\text{TRUNCATION}(Fronts)}$
end while
```

evoNSGA-II additionally uses estimates of evolvability for each size of solution to decide whether a solution should be selected. Specifically, we generate a table of bounds $\mathcal{B}$ that tells how many solutions of a certain size can be selected in the truncation selection step. This way we can prevent the over-replication of small, non-evolvable solutions. We proceed by explaining how $\mathcal{B}$ is built.

4.2 Construction of $\mathcal{B}$

We keep track of the evolvability of solutions in terms of their capability of generating accurate offspring of different sizes. Namely, we build a table $\mathcal{B}$ containing pairs (s,b), where s is a size and b is a bound on the number of times that solutions of size s can be selected by truncation selection to form the new population of evoNSGA-II. We want the number b to be proportionate to the (estimated) evolvability of the solutions of size s.

Algorithm 2 shows the construction of $\mathcal{B}$ in detail. For each offspring, the size of *its parent s* is considered. Then, a counter (*successes*) that is dedicated to that s is increased if the accuracy of the offspring is larger than that of the median accuracy computed over $\mathcal{P}$ (we choose the median over the mean because outliers are common in GP for SR). Note that we do not need to re-compute the accuracy of solutions, as they can simply be cached when solutions are evaluated. We also keep track of the number of offspring solutions that was generated from parents of size s (*attempts*). Finally, a simple measure of evolvability is computed for s, as the ratio between the number of successes and the number of attempts (similarly to the concept of *success ratio* in [2]). This ratio is in $[0, 1]$ and the larger its value, the better it is. We fill $\mathcal{B}$ with these ratios, for each size.

Recall that we wish to use $\mathcal{B}$ in the truncation selection process, which is applied to $\mathcal{P} \cup O$. Importantly, $\mathcal{P} \cup O$ contains both solutions that were not selected as parents, and offspring solutions: for those, there may exist a size that is not in $\mathcal{B}$, i.e., for which we have no information on its evolvability. Therefore, we artificially fill this information for potentially-missing sizes in $\mathcal{B}$ (line 13). Namely, for each missing size, we take the weighted average of the ratios observed for the closest smaller and closest larger size. Last but not least, we perform a normalization step on $\mathcal{B}$, transforming the ratios so that their sum amounts to the population size ($|\mathcal{P}|$). This way, for any size s, $\mathcal{B}[s]$ defines how many solutions should be selected at most. In the next section, we illustrate how $\mathcal{B}$ is used in the truncation selection of evoNSGA-II.

Algorithm 2 Build_$\mathcal{B}$

```
Require: $\mathcal{P}, \mathcal{O}$
1:  $max\_size \leftarrow \text{MAX\_SIZE}(\mathcal{P} \cup \mathcal{O})$
2:  $attempts[\text{i}] \leftarrow 0$ for $i \in \{1, \ldots, max\_size\}$
3:  $successes[\text{i}] \leftarrow 0$ for $i \in \{1, \ldots, max\_size\}$
4:  $median\_accuracy \leftarrow \text{MEDIAN\_ACCURACY}(\mathcal{P})$
5:  for $o \in \mathcal{O}$ do
6:      $s \leftarrow \text{FETCH\_PARENT\_SIZE}(o)$
7:      if $\text{ACCURACY}(o) > median\_accuracy$ then
8:          $successes[s] \leftarrow successes[s] + 1$
9:      end if
10:     $attempts[s] \leftarrow attempts[s] + 1$
11: end for
12: $\mathcal{B}[i] \leftarrow \frac{successes[i]}{attempts[i]}$ for $i \in \{1, \ldots, max\_size\} : attempts[i] \neq 0$
13: $\mathcal{B} \leftarrow \text{FILL\_MISSING\_SIZES}(\mathcal{B}, \mathcal{P}, \mathcal{O})$
14: $\mathcal{B} \leftarrow \text{NORMALIZE}(\mathcal{B}, |\mathcal{P}|)$
15: return $\mathcal{B}$
```

4.3 Use of $\mathcal{B}$ during truncation selection

The way truncation selection works in evoNSGA-II is the same as in NSGA-II (as described before, in Sec. 4.1), except for the fact that we will now use $\mathcal{B}$ to decide how many solutions of a certain size can be selected. We build $\mathcal{B}$ after the offspring population $\mathcal{O}$ has been evaluated, so that it is ready to be used for truncation selection. Like in NSGA-II, our truncation selection parses the solutions progressively, based on their rank. Different from NSGA-II, we do not immediately copy the solution that is currently in consideration; first, we consider the size of s of that solution, and the respective bound $\mathcal{B}[s]$. If the number of solutions of size s copied so far is less or equal to $\mathcal{B}[s]$, then the solution is selected; else, the solution is skipped, and the next solution is considered.

It can happen that, in Algorithm 2, large evolvability values are estimated for sizes for which there is a limited number of solutions in $\mathcal{P} \cup \mathcal{O}$, while low values are estimated for sizes for which there is an abundant number of solutions. Consequently, to respect the bounds in $\mathcal{B}$, the number of selected solutions may be lower than the population size. If that happens, we reset the counters for how many solutions of each size have been copied, and start the truncation selection process anew, from rank 1 onwards. This way, even if the bound for the size s is exceeded, we maintain an approximate proportionality between estimated evolvability of s and the number of selected solutions of size s.

Lastly, we remark that a single generation of evoNSGA-II is basically as fast as NSGA-II, as it entails minimial overhead. In fact, from the perspective of computational complexity, all operations needed to build and use $\mathcal{B}$ are linear in the population size, and thus subsumed by the complexity of other operations, particularly non-dominated sorting and evaluation of accuracy.

5 EXPERIMENTAL SETUP

We consider ten data sets that are commonly used in recent literature on GP for SR. The information for these data sets is reported in the supplementary material due to space limitations. For any run, we use a traditional Monte-Carlo split of the data set into training and test set, with respective proportions of 75-25%. Moreover, all data sets are standardized (based on the information in the training set) by subtracting the mean and dividing by the standard deviation for each feature separately, as advised in [11].

For comparison, we consider seven algorithms besides evoNSGA-II: classic NSGA-II [10], SPEA2 [35], α-dominance-based NSGA-II [32] with α varied with a linear (α-dom. lin.), cosine (α-dom. cos.), or sigmoid (α-dom. sig.) schedule, as well as its adaptive version [33] (Adap. α-dom.), and a simple extension of NSGA-II as mentioned in [30], where non-dominated sorting assigns an artificial worst-possible rank to duplicate solutions. We refer to the latter as NSGA-II with penalization of duplicates, NSGA-II+PD in short. For each algorithm, we keep track of the best-ever found non-dominated solutions (with respect to the training set) in an external archive, and return that archive at the end of the evolution.

Solutions are evaluated in terms of accuracy (to maximize) and size (to minimize). To maximize accuracy, we minimize the MSE (Sec. 2.1) augmented by linear scaling [14]. Linear scaling effectively enables to optimize in terms of a form of absolute correlation to the target variable y, typically causing a large improvement when GP is applied on real-world SR data sets [27]. Many state-of-the-art GP algorithms use linear scaling during the evolution [7].

To evaluate the quality of multi-objective search, we compute the hypervolume (HV) of the archive of best-found non-dominated solutions [36]. The HV indicates, for a set of solutions, the area in objective that is Pareto-dominated by that set of solutions, bounded by a reference point. The reference point represents an artificial solution with (very) poor performance in terms of all considered objectives, and should be chosen to be commensurate to the ranges of the objectives at play. We set the reference point to be $(1.1, 1.1)$ (meaning that the best-possible HV will be $1.1^2 = 1.21$) and normalize the MSE and size to be within 0 and 1. Even though the MSE would normally be unbounded from above, performing linear scaling guarantees that the maximal training error corresponds to predicting the mean of y; thus we can achieve the desired normalization by dividing by the variance of y. Regarding size, since very large solutions will likely not be interpretable, we enforce a maximal solution size of 100 (see Table 1) by deleting any offspring that exceeds that limit (and cloning the parent in its place); size is then normalized by dividing by 100.

We perform 30 repetitions for each run, to account for the randomness of train-test splitting and the stochasticity inherent to GP. We strive to present our results in terms of a typical parameter configuration that appears often in GP literature. To that end, we

Table 1: Parameter settings considered for *evoNSGA-II* and the other algorithms. Tournament size of 1 corresponds to random parent selection. SPEA2 does not employ tournament selection. For parameters with multiple possible settings (i.e., the first three), the settings in bold correspond to those that result in evoNSGA-II achieving the average overall performance in terms of hyper-volume on the training set.

Parameter	Considered settings
Population size	250, 500, **1000**, 2000, 5000
Tournament size	1, **2**, 7
Crossover-mutation proportion	0.5-0.5, **0.9-0.1**
Initialization	Ramped half-&-half (2–6)
Maximum solution size	100
Function set	$\{+, -, \times, \div^*, \sqrt{}^*, \log^*\}$
Terminal set	$\{x^{(1)}, \ldots, x^{(d)}, \text{ERC}\}$

actually consider a number of typical configurations; see Table 1, where some parameters have different possible settings (namely, population size, tournament size, and proportion between crossover and mutation). Note that starred operators (e.g., $\div^*$) implement protection and ephemeral random constants (ERC) [24] are sampled within $\mathcal{U}(-5, +5) \times \max_{i,j} |x_i^{(j)}|$. For each algorithm, we find the configuration that leads to the *average* performance for that algorithm (on the training set). The configuration that leads to the *average* performance for an algorithm is found as follows. First, for each data set, we consider the training HV (averaged across 30 runs) obtained on that data set by the different configurations. Configurations are sorted based on their HV, and their sort order is taken as a score. Next, an overall, single score is assigned to each configuration, by averaging the scores across the data sets of that configuration. Finally, we select the configuration whose overall score is closest to the one obtained by averaging the scores of all configurations. For example, the parameter settings in bold in Table 1 represent the configuration obtained for evoNSGA-II; The configurations for the other algorithms are reported in the supplementary material.

We use the Mann-Whitney-U test [21] to assess whether the distribution of HVs obtained by an algorithm is better than that of another, determining significance for p-value < 0.05, with Bonferroni correction [5].

6 RESULTS

6.1 Benchmarking results

6.1.1 Results and analysis. Tables 2 and 3 show the results obtained when considering the accuracy as measured on the training set and on the test set, respectively, for the parameter settings that result in the average performance. At training time, evoNSGA-II performs significantly better than any other algorithm in a vast number of cases, sometimes substantially so (e.g., when compared to Adap. α-dom., NSGA-II, and SPEA2 on several data sets). In fact, evoNSGA-II is found to be significantly better than another algorithm 64 times, worse only 1 times, and not significantly different 5 times. When it comes to the test set, evoNSGA-II remains vastly superior, although the number of statistical comparisons that are not significantly different raises to 19 (better 50 times, worse 1 time). This is due to the generalization gap between the training and the test set. In fact, improving generalization is not on focus in this paper.

Overall, only NSGA-II+PD is capable of coming close to the performance of evoNSGA-II. At training time, evoNSGA-II is better than NSGA-II+PD on 1 data sets, worse on 4, and equal on 5. At test time, the difference between the two shrinks even more, due to the generalization gap. Nevertheless, except for in one case (Yacht), evoNSGA-II is essentially equal or better than NSGA-II+PD.

We proceed by briefly describing what we observe for the other parameter configurations. More detailed information is reported in the supplementary material. Across the algorithms, using a larger population size and larger tournament size contribute to improve the performance, while it is unclear whether using more or less crossover than mutation is preferable. Across the configurations, evoNSGA-II remains the best-performing approach, although it is sometimes matched by NSGA-II+PD. However, we observe that with larger population sizes, the gap between evoNSGA-II and NSGA-II+PD grows in favor of the former. For example, at training time with the best-possible parameter configurations (for both algorithms, using a population size of 5000), evoNSGA-II is significantly superior to NSGA-II+PD on 6 data sets, equal on 4, and worse on none. This is likely because larger population sizes allow for better estimations of evolvability.

6.1.2 Further analysis: convergence of HV. To provide further evidence that evoNSGA-II is typically superior to the other algorithms, Fig. 3 (left) shows the convergence of the (training) HV, again with using parameter configurations that represent average performance, on Airfoil. Due to space limitations, we show the respective plots for other data sets in the supplementary material. For clarity, since the non-adaptive α-dominance algorithms perform similarly, we report only the one with linear scheduling (α-dom. lin.) in Fig. 3. As can be seen, the HV obtained by NSGA-II, SPEA2, and Adap. α-dom. tends to converge to a suboptimal value very soon after the first dozen generations. The other algorithms, i.e., evoNSGA-II, α-dom. lin., and NSGA-II+PD perform similarly, however evoNSGA-II is slightly superior throughout the whole search.

Furthermore, Fig. 3 (right) shows the distribution of the solutions in the final archives (for the 30 runs). The most apparent result is that only evoNSGA-II is capable of reliably discovering accurate solutions with a larger size than 30 (approximately). Interestingly, NSGA-II and NSGA-II+PD can be better than evoNSGA-II in discovering some relatively accurate solutions of size between 10 and 20 (approximately). This is because the search of NSGA-II and NSGA-II+PD can concentrate more in that area, as they discover larger and even more accurate solutions less frequently than evoNSGA-II.

6.2 Did it work as expected?

As last result, we show that evoNSGA-II does not, in fact, exhibit evolvability degeneration. Fig. 4 shows, on Airfoil, the evolvability that is estimated using the workflow proposed in Sec. 3.2 (left panel), and also the proportion of solutions of different sizes in the population of evoNSGA-II during the evolution (right panel), again using the parameter configuration that represents average performance. As can be seen, the proportion of solutions of larger

Table 2: Mean (standard deviation) of the HV computed on the training set for 30 runs of the considered algorithms. This table corresponds to the settings in bold in Table 1. The symbols +, −, = indicate, for each algorithm other than evoNSGA-II, whether the corresponding distribution of results for evoNSGA-II is, respectively, significantly better, worse, or not significantly different. The last row summarizes this information.

Data set	evoNSGA-II	Adap. α-dom.	α-dom. cos.	α-dom. lin.	α-dom. sig.	NSGA-II	NSGA-II+PD	SPEA2
Airfoil	0.799(0.021)	0.595(0.023)-	0.744(0.016)-	0.745(0.019)-	0.736(0.016)-	0.652(0.035)-	0.780(0.017)-	0.624(0.049)-
Boston	1.023(0.012)	0.895(0.024)-	0.984(0.010)-	0.986(0.014)-	0.980(0.014)-	0.954(0.015)-	1.017(0.010)=	0.929(0.019)-
Concrete	0.939(0.018)	0.684(0.033)-	0.884(0.029)-	0.876(0.025)-	0.864(0.030)-	0.792(0.037)-	0.941(0.018)=	0.725(0.053)-
Dow chemical	0.972(0.013)	0.714(0.043)-	0.914(0.013)-	0.914(0.018)-	0.908(0.023)-	0.779(0.057)-	0.977(0.007)=	0.836(0.037)-
Energy: cooling	1.119(0.008)	1.043(0.013)-	1.094(0.010)-	1.094(0.010)-	1.088(0.014)-	1.058(0.009)-	1.097(0.015)-	1.040(0.013)-
Energy: heating	1.152(0.004)	1.076(0.017)-	1.125(0.010)-	1.125(0.008)-	1.117(0.012)-	1.098(0.010)-	1.131(0.010)-	1.054(0.014)-
Tower	1.027(0.013)	0.824(0.046)-	0.994(0.012)-	0.985(0.027)-	0.984(0.024)-	0.941(0.030)-	1.029(0.006)=	0.867(0.047)-
Wine: red	0.513(0.005)	0.446(0.007)-	0.491(0.004)-	0.490(0.008)-	0.486(0.008)-	0.461(0.009)-	0.509(0.004)-	0.459(0.009)-
Wine: white	0.449(0.005)	0.378(0.008)-	0.426(0.007)-	0.422(0.008)-	0.416(0.009)-	0.403(0.007)-	0.446(0.004)=	0.387(0.010)-
Yacht	1.177(0.004)	1.141(0.019)-	1.174(0.001)-	1.174(0.001)-	1.174(0.001)-	1.163(0.013)-	1.178(0.001)+	1.150(0.011)-
Total +/−/=	—	0/10/0	0/10/0	0/10/0	0/10/0	0/10/0	1/4/5	0/10/0

Table 3: Results for the test set, formatting similar to that of Table 2.

Data set	evoNSGA-II	Adap. α-dom.	α-dom. cos.	α-dom. lin.	α-dom. sig.	NSGA-II	NSGA-II+PD	SPEA2
Airfoil	0.813(0.021)	0.669(0.027)-	0.781(0.018)-	0.782(0.017)-	0.781(0.013)-	0.708(0.030)-	0.795(0.019)-	0.690(0.041)-
Boston	0.969(0.019)	0.895(0.028)-	0.966(0.013)=	0.951(0.073)=	0.959(0.017)=	0.949(0.012)-	0.976(0.019)=	0.923(0.023)-
Concrete	0.930(0.025)	0.692(0.040)-	0.892(0.027)-	0.883(0.025)-	0.875(0.025)-	0.815(0.040)-	0.939(0.015)=	0.734(0.056)-
Dow chemical	0.920(0.020)	0.696(0.050)-	0.864(0.016)-	0.862(0.017)-	0.855(0.025)-	0.752(0.055)-	0.927(0.026)=	0.785(0.036)-
Energy: cooling	1.111(0.009)	1.033(0.017)-	1.092(0.009)-	1.089(0.012)-	1.082(0.016)-	1.052(0.009)-	1.097(0.016)-	1.028(0.016)-
Energy: heating	1.144(0.007)	1.087(0.016)-	1.128(0.010)-	1.126(0.009)-	1.124(0.012)-	1.101(0.009)-	1.136(0.009)=	1.067(0.013)-
Tower	1.022(0.020))	0.812(0.049)-	0.986(0.035)-	0.981(0.039)-	0.985(0.025)-	0.941(0.037)-	1.031(0.005)=	0.859(0.050)-
Wine: red	0.629(0.059)	0.591(0.009)-	0.634(0.009)=	0.633(0.013)=	0.632(0.011)=	0.615(0.014)-	0.647(0.013)=	0.613(0.011)-
Wine: white	0.359(0.074)	0.354(0.010)=	0.390(0.020)=	0.378(0.050)=	0.387(0.012)=	0.379(0.006)=	0.400(0.025)=	0.361(0.012)=
Yacht	1.170(0.002)	1.131(0.024)-	1.167(0.004)-	1.167(0.002)-	1.167(0.003)-	1.155(0.013)-	1.172(0.001)+	1.137(0.013)-
Total +/−/=	—	0/9/1	0/7/3	0/7/3	0/7/3	0/9/1	1/2/7	0/9/1

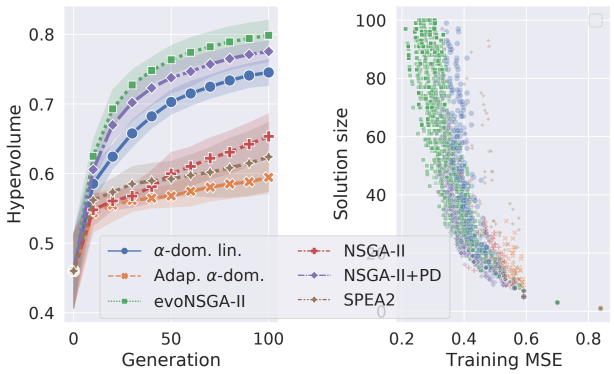

Figure 3: Comparison between the algorithms in terms of HV during the evolution (left) and final front (right) for 30 runs on Airfoil at training time. Left: Lines represent means and shaded areas represent standard deviations. Right: All solutions in the archives from the 30 runs are shown.

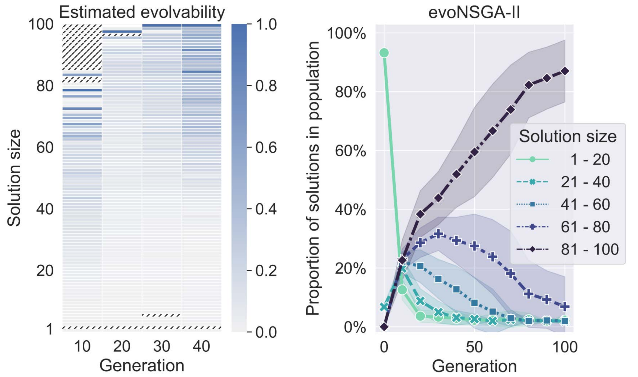

Figure 4: Left: Each column of the heat-map shows, for a given generation, the average evolvability between crossover and mutation computed with the workflow of Sec. 3.2 on Airfoil, and normalized across solution size (dashed entries represent absent sizes). Right: Proportions of solutions of different sizes in evoNSGA-II during 30 evolutions on Airfoil (lines are means, shaded areas are standard deviations).

sizes increases over time during the evolution process of evoNSGA-II, which is in agreement with the expected evolvability from our analysis. This result is in stark contrast with the one displayed in Fig. 1 (note the different scale in sizes), where NSGA-II could not discover larger and more accurate solutions.

We produced the same plots for the other algorithms and included them in the supplementary material; there, it can be seen that SPEA2 and Adap. α dom., like NSGA-II, suffer from evolvability degeneration. The other algorithms perform better, yet still assign less copies to larger solutions than evoNSGA-II.

7 DISCUSSION

In this work, we investigated *evolvability degeneration*, i.e., the phenomenon by which small solutions over-replicate and hamper search progress because they represent unfruitful parents for the discovery of larger and more accurate offspring. Next, we proposed to extend NSGA-II into evoNSGA-II, which estimates the evolvability of solutions based on their size and, based on this, bounds how many solutions of any given size can be selected for the next generation. Lastly, we found that evoNSGA-II is largely superior to other recent MOGP algorithms, and is indeed capable of allowing solutions of highly-evolvable size to thrive.

The reason for evolvability degeneration can be linked to the fact that the algorithm has insufficient time to discover more accurate solutions (because the probability that variation succeeds is low) compared to the speed by which small solutions duplicate. Such hypothesis is strongly supported by the findings of [25], where it is shown that GP tends to fail when the pressure surpasses a certain threshold. It is thus natural that MOGP algorithms that improve the diversity of the population perform better than classic NSGA-II. In fact, the algorithms that we used in our comparisons that were built to improve NSGA-II, essentially realize some form of diversity preservation. However, none of them considers tracking and using evolvability to decide which solutions to keep and which to discard. In our view, this is the fundamental reason why evoNSGA-II performed best. Interestingly, NSGA-II+PD, which is perhaps an even simpler approach than evoNSGA-II, performed similarly to evoNSGA-II in many data sets. Still, evoNSGA-II performed typically equal to or better than NSGA-II+PD, suggesting that one may not always want to discard all duplicate solutions: keeping a number of copies for highly-evolvable solutions seems to be generally more helfpul. Moreover, we observed that the performance gap between evoNSGA-II and NSGA-II+PD tends to increase when the population size is larger. We believe that this happens because larger population sizes allow for better estimations of evolvability.

There exist a number of limitations in this paper that call for future research. Firstly, our estimations of evolvability are repeated every generation, using solely the current population. We attempted to use exponential-moving-averages to incorporate estimations from previous generations but preliminary findings indicated no statistically significant improvement. However, one could study whether other approaches can lead to an improvement, such as learning an accurate model of evolvability of solution size across multiple data sets and parameter configurations, and using that model as starting point when dealing with a new problem. A second important limitation is that we considered minimizing solution size, which is a simple but coarse way of pursuing interpretability. Future work should consider other and better proxies of interpretability (e.g., [6, 29]), and assess whether the good performance found here for evoNSGA-II transfers to those settings. Transferability of the quality of our approach should also be assessed when other variation operators are used, such as geometric semantic- [22, 23] or linkage-based ones [28], as well as, e.g., gradient descent to optimize coefficients [11]. A third limitation is that evoNSGA-II makes no attempt to limit bloat. If bloated solutions have larger evolvability, they will replicate more than others. For SR this is not necessarily a problem, since it is reasonable to impose a cap on the maximally allowed size, above which solutions would certaintly not be interpretable. However, capping the size might not be desirable for other problems. There, evoNSGA-II might keep discovering larger and larger solutions, and thus fail to find medium-sized ones. Thus, bloat-control mechanisms may need to be considered.

Finally, we conclude this work by reflecting on the fact that our results may, in principle, transfer to problems of very different nature than GP for SR. Indeed, we remark that maximizing accuracy and minimizing solution size is an *imbalanced* multi-objective problem: on the one hand, minimizing solution size is easily done, since random deletion of components suffices to improve this objective; on the other hand, maximizing accuracy is almost always challenging, since the right components need to appear in the right order to obtain an accurate model. There might exist a number of problems where a similar situation happens, i.e., the objective that is easy-to-optimize inhibits the search of solutions with respect to the objective that is hard-to-optimize. For any given problem, tracking and exploiting information on the evolvability in terms of the hard-to-optimize objective that is associated with the easy-to-optimize objective might be a consistent way to improve multi-objective evolutionary search.

8 CONCLUSION

We studied an important cause of inefficiency in the use of the non-dominated sorting genetic algorithm II (NSGA-II) for the discovery of symbolic regression models with trade-offs between accuracy and simplicity. Namely, we experimentally found that simpler models over-replicate and take over the majority of the population, because they lack *evolvability*, i.e., they represent infertile grounds for larger but more accurate models to be discovered. We named this phenomenon *evolvability degeneration*, and proposed *evoNSGA-II*, an algorithm that is explicitly built to prevent it. With comparisons to NSGA-II and six other algorithms, upon ten real-world data sets, and across different parameter configurations, we found evoNSGA-II to be the superior approach. The working principles of evoNSGA-II are not limited to symbolic regression: studying their transferability to other imbalanced multi-objective problems represents an interesting avenue for future research.

ACKNOWLEDGMENTS

This research was funded by the European Commission within the HORIZON Programme (TRUST-AI Project, Contract No.:952060). We further thank Maurits and Anna de Kock Foundation for financing a high-performance computing system.

REFERENCES

[1] Amina Adadi and Mohammed Berrada. 2018. Peeking Inside the Black-box: A Survey on eXplainable Artificial Intelligence (XAI). *IEEE Access* 6 (2018), 52138–52160.

[2] Michael Affenzeller and Stefan Wagner. 2005. Offspring selection: A new self-adaptive selection scheme for genetic algorithms. In *Adaptive and Natural Computing Algorithms*. Springer, 218–221.

[3] Khaled M. S. Badran and Peter I. Rockett. 2007. The Roles of Diversity Preservation and Mutation in Preventing Population Collapse in Multiobjective Genetic Programming. In *Proceedings of the 9th Annual Conference on Genetic and Evolutionary Computation*. Association for Computing Machinery, 1551–1558.

[4] Stefan Bleuler, Martin Brack, Lothar Thiele, and Eckart Zitzler. 2001. Multiobjective Genetic Programming: Reducing Bloat using SPEA2. In *Proceedings of the 2001 Congress on Evolutionary Computation*, Vol. 1. IEEE, 536–543.

[5] Carlo Bonferroni. 1936. Teoria Statistica delle Classi e Calcolo delle Probabilita: *Pubblicazioni del R Istituto Superiore di Scienze Economiche e Commericiali di Firenze* 8 (1936), 3–62.

[6] Bogdan Burlacu, Gabriel Kronberger, Michael Kommenda, and Michael Affenzeller. 2019. Parsimony Measures in Multi-Objective Genetic Programming for Symbolic Regression. In *Proceedings of the Genetic and Evolutionary Computation Conference Companion*. Association for Computing Machinery, 338–339.

[7] William La Cava, Patryk Orzechowski, Bogdan Burlacu, Fabricio Olivetti de Franca, Marco Virgolin, Ying Jin, Michael Kommenda, and Jason H. Moore. 2021. Contemporary Symbolic Regression Methods and their Relative Performance. In *Thirty-fifth Conference on Neural Information Processing Systems Datasets and Benchmarks Track*.

[8] Stéphane d'Ascoli, Pierre-Alexandre Kamienny, Guillaume Lample, and François Charton. 2022. Deep Symbolic Regression for Recurrent Sequences. arXiv:2201.04600

[9] Edwin D. De Jong and Jordan B. Pollack. 2003. Multi-Objective Methods for Tree Size Control. *Genetic Programming and Evolvable Machines* 4, 3 (2003), 211–233.

[10] Kalyanmoy Deb, Amrit Pratap, Sameer Agarwal, and TAMT Meyarivan. 2002. A fast and elitist multiobjective genetic algorithm: NSGA-II. *IEEE transactions on evolutionary computation* 6, 2 (2002), 182–197.

[11] Grant Dick, Caitlin A. Owen, and Peter A. Whigham. 2020. Feature Standardisation and Coefficient Optimisation for Effective Symbolic Regression. In *Proceedings of the 22th Annual Conference on Genetic and Evolutionary Computation*. Association for Computing Machinery, 306–314.

[12] David Gunning and David W. Aha. 2019. DARPA's Explainable Artificial Intelligence (XAI) Program. *AI Magazine* 40, 2 (2019), 44–58.

[13] Hisao Ishibuchi, Kaname Narukawa, and Yusuke Nojima. 2005. An Empirical Study on the Handling of Overlapping Solutions in Evolutionary Multiobjective Optimization. In *Proceedings of the 7th Annual Conference on Genetic and Evolutionary Computation*. Association for Computing Machinery, 817–824.

[14] Maarten Keijzer. 2003. Improving Symbolic Regression with Interval Arithmetic and Linear Scaling. In *European Conference on Genetic Programming*. Springer, 70–82.

[15] Michael Kommenda, Gabriel Kronberger, Michael Affenzeller, Stephan M Winkler, and Bogdan Burlacu. 2016. Evolving Simple Symbolic Regression Models by Multi-objective Genetic Programming. In *Genetic Programming Theory and Practice XIII*. Springer, 1–19.

[16] John R Koza. 1992. *Genetic programming: on the programming of computers by means of natural selection*. Vol. 1. MIT press, Cambridge, MA, USA.

[17] William La Cava and Jason H. Moore. 2020. Genetic programming approaches to learning fair classifiers. In *Proceedings of the 2020 Genetic and Evolutionary Computation Conference*. 967–975.

[18] William B. Langdon. 2021. Genetic programming convergence. *Genetic Programming and Evolvable Machines* (2021), 1–34.

[19] Zachary C Lipton. 2018. The Mythos of Model Interpretability: In machine learning, the concept of interpretability is both important and slippery. *Queue* 16, 3 (2018), 31–57.

[20] Sean Luke and Liviu Panait. 2006. A Comparison of Bloat Control Methods for Genetic Programming. *Evolutionary Computation* 14, 3 (2006), 309–344.

[21] Henry B. Mann and Donald R. Whitney. 1947. On a Test of Whether One of Two Random Variables is Stochastically Larger than the Other. *The Annals of Mathematical Statistics* (1947), 50–60.

[22] Alberto Moraglio, Krzysztof Krawiec, and Colin G Johnson. 2012. Geometric semantic genetic programming. In *International Conference on Parallel Problem Solving from Nature*. Springer, 21–31.

[23] Tomasz P. Pawlak, Bartosz Wieloch, and Krzysztof Krawiec. 2014. Semantic backpropagation for designing search operators in genetic programming. *IEEE Transactions on Evolutionary Computation* 19, 3 (2014), 326–340.

[24] Riccardo Poli, William B. Langdon, and Nicholas Freitag McPhee. 2008. *A Field Guide to Genetic Programming*. Lulu Enterprises, UK Ltd.

[25] Terence Soule and James A. Foster. 1998. Effects of Code Growth and Parsimony Pressure on Populations in Genetic Programming. *Evolutionary Computation* 6, 4 (1998), 293–309.

[26] Silviu-Marian Udrescu and Max Tegmark. 2020. AI Feynman: A physics-inspired method for symbolic regression. *Science Advances* 6, 16 (2020).

[27] Marco Virgolin, Tanja Alderliesten, and Peter A. N. Bosman. 2019. Linear Scaling with and within Semantic Backpropagation-based Genetic Programming for Symbolic Regression. In *Proceedings of the Genetic and Evolutionary Computation Conference*. 1084–1092.

[28] Marco Virgolin, Tanja Alderliesten, Cees Witteveen, and Peter A. N. Bosman. 2021. Improving Model-based Genetic Programming for Symbolic Regression of Small Expressions. *Evolutionary Computation* 29, 2 (2021), 211–237.

[29] Marco Virgolin, Andrea De Lorenzo, Eric Medvet, and Francesca Randone. 2020. Learning a formula of interpretability to learn interpretable formulas. In *International Conference on Parallel Problem Solving from Nature*. Springer, 79–93.

[30] Marco Virgolin, Andrea De Lorenzo, Francesca Randone, Eric Medvet, and Mattias Wahde. 2021. *Model Learning with Personalized Interpretability Estimation (ML-PIE)*. Association for Computing Machinery, 1355–1364.

[31] Ekaterina J. Vladislavleva, Guido F. Smits, and Dick 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.

[32] Shaolin Wang, Yi Mei, and Mengjie Zhang. 2020. A Multi-Objective Genetic Programming Hyper-Heuristic Approach to Uncertain Capacitated Arc Routing Problems. In *Proceedings of the 2020 IEEE Congress on Evolutionary Computation (CEC)*. 1–8.

[33] Shaolin Wang, Yi Mei, and Mengjie Zhang. 2021. A Multi-Objective Genetic Programming Approach with Self-Adaptive α Dominance to Uncertain Capacitated Arc Routing Problem. In *Proceedings of the 2021 IEEE Congress on Evolutionary Computation (CEC)*. 636–643.

[34] Hengzhe Zhang and Aimin Zhou. 2021. RL-GEP: Symbolic Regression via Gene Expression Programming and Reinforcement Learning. In *2021 International Joint Conference on Neural Networks (IJCNN)*. IEEE, 1–8.

[35] Eckart Zitzler, Marco Laumanns, and Lothar Thiele. 2001. SPEA2: Improving the Strength Pareto Evolutionary Algorithm. *TIK-report* 103 (2001).

[36] Eckart Zitzler and Lothar Thiele. 1998. Multiobjective optimization using evolutionary algorithms — A comparative case study. In *International Conference on Parallel Problem Solving from Nature*, Agoston E. Eiben, Thomas Bäck, Marc Schoenauer, and Hans-Paul Schwefel (Eds.). Springer Berlin Heidelberg, Berlin, Heidelberg, 292–301.