Improving Model-Based Genetic Programming for Symbolic Regression of Small Expressions

M. Virgolin marco.virgolin@cwi.nl
Life Science and Health group, Centrum Wiskunde & Informatica, Amsterdam,
1098 XG, the Netherlands

T. Alderliesten t.alderliesten@lumc.nl
Department of Radiation Oncology, Amsterdam UMC, University of Amsterdam,
Amsterdam, 1105 AZ, the Netherlands
Department of Radiation Oncology, Leiden University Medical Center, Leiden,
2333 ZA, the Netherlands

C. Witteveen c.witteveen@tudelft.nl
Algorithmics Group, Delft University of Technology, Delft, 2628 XE, the Netherlands

P. A. N. Bosman peter.bosman@cwi.nl
Life Science and Health group, Centrum Wiskunde & Informatica, Amsterdam,
1098 XG, the Netherlands
Algorithmics Group, Delft University of Technology, Delft, 2628 XE, the Netherlands

https://doi.org/10.1162/evco_a_00278

Abstract

The Gene-pool Optimal Mixing Evolutionary Algorithm (GOMEA) is a model-based EA framework that has been shown to perform well in several domains, including Genetic Programming (GP). Differently from traditional EAs where variation acts blindly, GOMEA learns a model of interdependencies within the genotype, that is, the linkage, to estimate what patterns to propagate. In this article, we study the role of Linkage Learning (LL) performed by GOMEA in Symbolic Regression (SR). We show that the non-uniformity in the distribution of the genotype in GP populations negatively biases LL, and propose a method to correct for this. We also propose approaches to improve LL when ephemeral random constants are used. Furthermore, we adapt a scheme of interleaving runs to alleviate the burden of tuning the population size, a crucial parameter for LL, to SR. We run experiments on 10 real-world datasets, enforcing a strict limitation on solution size, to enable interpretability. We find that the new LL method outperforms the standard one, and that GOMEA outperforms both traditional and semantic GP. We also find that the small solutions evolved by GOMEA are competitive with tuned decision trees, making GOMEA a promising new approach to SR.

Keywords

Genetic programming, symbolic regression, linkage, GOMEA, machine learning, interpretability.

1 Introduction

Symbolic Regression (SR) is the task of finding a function that explains hidden relationships in data, without prior knowledge on the form of such function. Genetic

Manuscript received: 2 July 2019; revised: 1 December 2019, 27 January 2020, 27 February 2020; accepted: 17 June 2020.

Programming (GP) (Koza, 1992) is particularly suited for SR, as it can generate solutions of arbitrary form using basic functional components.

Much work has been done in GP for SR, proposing novel algorithms (Krawiec, 2015; Zhong et al., 2018; De Melo, 2014), hybrids (Žegklitz and Pošík, 2017; Icke and Bongard, 2013), and other forms of enhancement (Keijzer, 2003; Chen et al., 2015). What is recently receiving a lot of attention is the use of so-called *semantic-aware* operators, which enhance the variation process of GP by considering intermediate solution outputs (Pawlak et al., 2015; Chen et al., 2018; Moraglio et al., 2012). The use of semantic-aware operators has proven to enable the discovery of very accurate solutions, but often at the cost of complexity: solution size can range from hundreds to billions of components (Pawlak et al., 2015; Martins et al., 2018). These solutions are consequently impossible to interpret, a fact that complicates or even prohibits the use of GP in many real-world applications because many practitioners desire to understand what a solution means before trusting its use (Lipton, 2018; Guidotti et al., 2018). The use of GP to discover uninterpretable solutions can even be considered to be questionable in many domains, as many alternative machine learning algorithms exist that can produce competitive solutions much faster (Orzechowski et al., 2018).

We therefore focus on SR when GP is *explicitly constrained* to generate small-sized solutions, that is, mathematical expressions consisting of a small number of basic functional components, to increase the level of interpretability. With size limitation, finding accurate solutions is particularly hard. It is not without reason that many effective algorithms work instead by growing solution size, for example, by iteratively stacking components (Moraglio et al., 2012; Chen and Guestrin, 2016).

A recurring hypothesis in GP literature is that the evolutionary search can be made effective if *salient patterns*, occurring in the representation of solutions (i.e., the genotype), are identified and preserved during variation (Poli et al., 2008). It is worth studying if this holds for SR, to find accurate small solutions.

The hypothesis that salient patterns in the genotype can be found and exploited is what motivates the design of Model-Based Evolutionary Algorithms (MBEAs). Among them, the Gene-pool Optimal Mixing Evolutionary Algorithm (GOMEA) is a recent EA that has proven to perform competitively in different domains: discrete optimization (Thierens and Bosman, 2011; Luong et al., 2014), real-valued optimization (Bouter et al., 2017), but also grammatical evolution (Medvet, Bartoli et al., 2018), and, the focus of this article, GP (Virgolin et al., 2017, 2018). GOMEA embodies within each generation a model-learning phase, where *linkage*, that is, the interdependency within parts of the genotype, is modeled. During variation, the linkage information is used to propagate genotype patterns and avoid their disruption.

The aim of this article is to understand the role of linkage learning when dealing with SR, and consequently improve the GP variant of GOMEA (GP-GOMEA), to find small and accurate SR solutions for realistic problems. We present three main contributions. First, we propose an improved linkage learning approach, that, differently from the original one, is unbiased with respect to the way the population is initialized. Second, we analyze how linkage learning is influenced by the presence of many different constant values, sampled by Ephemeral Random Constant (ERC) nodes (Poli et al., 2008), and explore strategies to handle them. Third, we introduce improvements upon GP-GOMEA's Interleaved Multistart Scheme (IMS), a strategy of multiple evolutionary runs of increasing evolutionary budget that executes them in an interleaved fashion, to better deal with SR and learning tasks in general.

The structure of this article is as follows. In Section 2, we briefly discuss related work on MBEAs for GP. In Section 3, we explain how GP-GOMEA and linkage learning work. Before proceeding with the description of the new contributions and experiments, Section 4 shows general parameter settings and datasets that will be used in the article. Next, we proceed by interleaving our findings on current limitations of GP-GOMEA followed by proposals to overcome such limitations, and respective experiments. In other words, we describe how we improve linkage learning one step at a time. In particular, Section 5 presents current limitations of linkage learning, and describes how we improve linkage learning. Strategies to learn linkage efficiently and effectively when ERCs are used are described in Section 6. We propose a new IMS for SR in Section 7, and use it in Section 8 to benchmark GP-GOMEA with competing algorithms: traditional GP, GP using a state-of-the-art semantic-aware operator, and the very popular decision tree for regression (Breiman et al., 1984). Lastly, we discuss our findings and draw conclusions in Section 9.

2 Related Work

We differentiate today's MBEAs into two classes: Estimation-of-Distribution Algorithms (EDA), and Linkage-based Mixing EAs (LMEA). EDAs work by iteratively updating a probabilistic model of good solutions, and sampling new solutions from that model. LMEAs attempt to capture linkage, that is, interdependencies between parts of the genotype, and proceed by variating solutions with mechanisms to avoid the disruption of patterns with strong linkage.

Several EDAs for GP have been proposed so far. Hauschild and Pelikan (2011) and Kim et al. (2014) are relatively recent surveys on the matter. Two categories of EDAs for GP have mostly emerged in the years: one where the shape of solutions adheres to some template to be able to estimate probabilities of what functions and terminals appear in what locations (called *prototype tree* for tree-based GP) (Salustowicz and Schmidhuber, 1997; Sastry and Goldberg, 2003; Yanai and Iba, 2003; Hemberg et al., 2012), and one where the probabilistic model is used to sample grammars of rules which, in turn, determine how solutions are generated (Shan et al., 2004; Bosman and De Jong, 2004; Wong et al., 2014; Sotto and de Melo, 2017). Research on EDAs for GP appears to be limited. The review of Kim et al. (2014) admits, quoting, that "*Unfortunately, the latter research [EDAs for GP] has been sporadically carried out, and reported in several different research streams, limiting substantial communication and discussion.*"

Concerning symbolic regression, we crucially found no works where it is attempted on realistic datasets (we searched among the work reported by the surveys and other recent work cited here). Many contributions on EDAs for GP have been validated on hard problems of artificial nature instead, such as *Royal Tree* and *Deceptive Max* (Hasegawa and Iba, 2009). Some real-world problems have been explored, but concerning only a limited number of variables (Tanev, 2007; Li et al., 2010). When considering symbolic regression, at most synthetic functions or small physical equations with only few ($\leq$5) variables have been considered (e.g., by Ratle and Sebag, 2001 and Sotto and de Melo, 2017).

The study of LMEAs has emerged the first decade of the millennium in the field of binary optimization, where it remains mostly explored to date (Chen et al., 2007; Thierens and Bosman, 2013; Goldman and Punch, 2014; Hsu and Yu, 2015). Concerning GP, GOMEA is the first state-of-the-art LMEA ever brought to GP (Virgolin et al., 2017).

GP-GOMEA was first introduced in Virgolin et al. (2017), to tackle classic yet artificial benchmark problems of GP (including some of the ones mentioned before), where

Algorithm 1 Outline of GOMEA

```
1 procedure RUNGOMEA($n^{\text{pop}}$)
2     $\mathcal{P} \leftarrow$initializePopulation($n^{\text{pop}}$)
3     while terminationCriteriaNotMet() do
4         $F \leftarrow$learnFOS($\mathcal{P}$)
5         $\mathcal{O} \leftarrow \emptyset$
6         for $i \in \{1, \ldots, n^{\text{pop}}\}$ do
7             $\mathcal{O}_i \leftarrow$GOM($\mathcal{P}_i, \mathcal{P}, F$)
8         $\mathcal{P} \leftarrow \mathcal{O}$
```

the optimum is known. The IMS, largely inspired on the work by Harik and Lobo (1999), was also proposed, to relieve the user from the need of tuning the population size. Population sizing is particularly crucial for MBEAs in general: the population needs to be big enough for probability or linkage models to be reliable, yet small enough to allow efficient search (Harik et al., 1999).

GP-GOMEA has also seen a first adaptation to SR, to find small and accurate solutions for a clinical problem where interpretability is important (Virgolin et al., 2018). There, GP-GOMEA was engineered for the particular problem, and no analysis of what linkage learning brings to SR was performed. Also, instead of using the IMS, a fixed population size was used. This is because the IMS was originally designed by Virgolin et al. (2017) to enable benchmark problems to be solved to optimality. No concern on generalization of solutions to unseen test cases was incorporated.

As to combining LMEAs with grammatical evolution, Medvet, Bartoli et al. (2018) also employed GOMEA, to attempt to learn and exploit linkage when dealing with different types of predefined grammars. In that work, only one synthetic function was considered for symbolic regression, among other four benchmark problems.

There is a need for assessing whether MBEAs can bring an advantage to real-world symbolic regression problems. This work attempts to do this, by exploring possible limitations of GP-GOMEA and ways to overcome them, and validating experiments upon realistic datasets with dozens of features and thousands of observations.

3 Gene-Pool Optimal Mixing Evolutionary Algorithm for GP

Three main concepts are at the base of (GP-)GOMEA: solution representation (genotype), linkage learning, and linkage-based variation. These components are arranged in a common outline that encompasses all algorithms of the GOMEA family.

Algorithm 1 shows the outline of GOMEA. As most EAs, GOMEA starts by initializing a population $\mathcal{P}$, given the desired population size n^{pop}. The generational loop is then started and continues until a termination criterion is met, for example, a limit on the number of generations or evaluations, or a maximum time. Lines 4 to 8 represent a generation. First, the linkage model is learned, which is called Family of Subsets (FOS) (explained in Section 3.2). Second, each solution $\mathcal{P}_i$ is used to generate an offspring solution $\mathcal{O}_i$ by the variation operator Gene-pool Optimal Mixing (GOM). Last, the offspring replace the parent population. Note the lack of a separate selection operator. This is because GOM performs variation and selection at the same time (see Section 3.3).

For GP-GOMEA, an extra parameter is needed, the tree height (or, equivalently, tree depth) h. This is necessary to determine the representation of solutions, as described in the following Section 3.1.

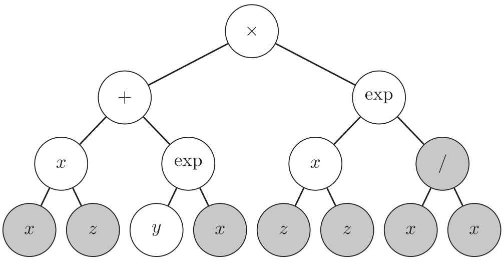


Figure 1: Example of tree for GP-GOMEA with $h = 3$ and $r = 2$. While 15 nodes are present, the nodes that influence the output are only 7: the gray nodes are introns.

3.1 Solution Representation in GP-GOMEA

GP-GOMEA uses a modification of the tree-based representation (Koza, 1992) which is similar to the one used by Salustowicz and Schmidhuber (1997). While typical GP trees can have any shape, GP-GOMEA uses a fixed template that allows linkage learning and linkage-based variation to be performed in a similar fashion as for other, fixed string-length versions of GOMEA.

All solutions are generated as *perfect* r-ary trees of height h, that is, such that all non-leaf nodes have exactly r children, and leaves are all at maximum depth h, with r being the maximum number of inputs accepted by the functions (arity) provided in the function set (e.g., for $\{+, -, \times\}$, $r = 2$), and h chosen by the user. Note that, for any node that is not at maximum depth, r child nodes are appended anyway: no matter if the node is a terminal, or if it is a function requiring less than r inputs (in this case, the leftmost nodes are used as inputs). Some nodes are thus *introns*; that is, they are not executed to compute the output of the tree. It follows that while trees are *syntactically* redundant, they are not necessarily *semantically* so. All trees of GP-GOMEA have the same number of nodes, equal to $\ell = \sum_{i=0}^{h} r^i = \frac{r^{h+1}-1}{r-1}$. Figure 1 shows a tree used by GP-GOMEA.

3.2 Linkage Learning

The linkage model used by GOMEA algorithms is called the Family of Subsets (FOS), and is a set of sets:

$$F = \{F_1, \dots, F_{|F|}\},\ F_i \subseteq \{1, \dots, l\}.$$

Each F_i (called FOS subset) contains indices representing locations in the genotype. For GP-GOMEA, these indices represent node locations. It is sufficient to choose a parsing order to identify the same node locations in all trees, since trees share the same shape.

In GOMEA, linkage learning corresponds to building a FOS. Different types of FOS exist in literature, however, the one recommended as default is the *Linkage Tree* (LT), by, for example, Thierens and Bosman (2013) and Virgolin et al. (2017). The LT captures linkage on hierarchical levels. An LT is learned every generation, from the population. To assess whether linkage learning plays a key role, that is, whether it is better than randomly choosing linkage relations, we also consider the Random Tree (RT) (Virgolin et al., 2017).

3.2.1 Linkage Tree

The LT arranges the FOS subsets in a binary tree structure representing hierarchical levels of linkage strength among genotype locations. The LT is built bottom-up, that is, from the leaves to the root. The bottom level of the LT, that is, the leaves, assume that all genotype locations are independent (no linkage), and is realized by instantiating FOS subsets to singletons, each containing a genotype location i, $\forall i \in \{1, \ldots, \ell\}$.

To build the next levels, mutual information is used as a proxy for linkage strength. Mutual information is a sensible choice to represent linkage strength because it expresses, considering, for example, the pair (i, j) of genotype locations as random variables, the amount of information gained on i given observations on j (and vice versa). In this light, the population can be considered as a set of realizations of the genotype. In particular, the realizations of each genotype location i are what *symbols* appear at location i in the population. In a binary genetic algorithm, symbols are either "0" or "1" while in GP, symbols correspond to the types of function and terminal nodes, for example, "+","−","x_1","x_2". In other words, random variables can assume as many values as there are possible symbols in the instruction set.[1]

Now, the next step is to compute the mutual information between each and every pair of locations in the genotype of the entire population. Mutual information between a pair of locations can be computed after measuring entropy for single locations $\mathrm{H}(i)$, and the joint entropy for locations pairs, $\mathrm{H}(i, j)$ (this aspect will be used in Section 5):

$$\begin{aligned} &\mathrm{MI}(i, j) = \mathrm{H}(i) + \mathrm{H}(j) - \mathrm{H}(i, j), \text{ where} \\ &\quad \mathrm{H}(i) = -\sum \mathrm{P}_i \log \mathrm{P}_i, \qquad \mathrm{H}(i, j) = -\sum \mathrm{P}_{ij} \log \mathrm{P}_{ij}, \end{aligned} \tag{1}$$

and P_i (P_{ij}) is the (joint) probability distribution over the symbols at location(s) i (i, j), which can be estimated by counting occurrences of symbol types in the population genotype. This requires to loop over the entire population, and to use nested loops over location pairs $i \in \{1, \ldots, \ell\}$ and $j \in \{i, \ldots, \ell\}$, leading to a time complexity of $O(n^{\text{pop}}\ell^2)$. The contribution to the entropy of null probability cases $(-0 \log 0)$ is set to 0.

Given mutual information between location pairs, we approximate linkage among higher orders of locations using the Unweighted Pair Group Method with Arithmetic Mean (UPGMA) (Gronau and Moran, 2007). To ease understanding, we now provide an explanation of how UPGMA is used to build the rest of the LT that is primarily meant to be intuitive. In practice, we do not use an implementation that strictly adheres to the following explanation, but we use a more advanced algorithm that achieves the same result while having lower time complexity, called the Reciprocal Nearest Neighbor algorithm (RNN). For details on RNN, see Gronau and Moran (2007).

UPGMA operates in a recursive, hierarchical fashion. Consider each singleton containing a different genotype location i as a cluster C_i, and the mutual information between location pairs as a measure of similarity S between clusters, that is, $S(C_i, C_j) := \mathrm{MI}(i, j)$. Let $\mathcal{C}$ be the collection of clusters to be parsed, initially containing all location singletons. Every iteration, firstly a new cluster $C_{i^\star} \cup C_{j^\star}$ is formed by joining the clusters $C_{i^\star}, C_{j^\star}$ that have maximal similarity. Secondly, $C_{i^\star}$ and $C_{j^\star}$ are removed from $\mathcal{C}$, and $C_{i^\star} \cup C_{j^\star}$ is inserted in $\mathcal{C}$. When this happens, a FOS subset is added in the LT that corresponds to (contains the same locations of) $C_{i^\star} \cup C_{j^\star}$, as parent of the subsets that represent $C_{i^\star}$ and $C_{j^\star}$. Thirdly, the similarity between $C_{i^\star} \cup C_{j^\star}$ and every other cluster C_k is computed, with:

$$S(C_k, C_{i^\star} \cup C_{j^\star}) = \frac{|C_{i^\star}|}{|C_{i^\star}| + |C_{j^\star}|} S(C_k, C_i) + \frac{|C_{j^\star}|}{|C_{i^\star}| + |C_{j^\star}|} S(C_k, C_j).$$

[1] More symbols can be possible than the number of instructions in case ERCs are used, since instantiating an ERC in a solution results in a constant being randomly sampled.

Algorithm 2 Pseudocode of GOM

```
1  procedure GOM($\mathcal{P}_i, \mathcal{P}, F$)
2      $\mathcal{B}_i \leftarrow \mathcal{P}_i$; $f_{\mathcal{B}_i} \leftarrow f_{\mathcal{P}_i}$; $\mathcal{O}_i \leftarrow \mathcal{P}_i$
3      $F \leftarrow$randomShuffle($F$)
4      for $F_j \in F$ do
5          $\mathcal{D} \leftarrow$pickRandomDonor($\mathcal{P}$)
6          $\mathcal{O}_i \leftarrow$overrideNodes($\mathcal{O}_i, \mathcal{D}, F_j$)
7          if $\mathcal{O}_i \neq^\star \mathcal{B}_i$ then
8              $f_{\mathcal{O}_i} \leftarrow$computeFitness($\mathcal{O}_i$)
9              if $f_{\mathcal{O}_i} \leq f_{\mathcal{B}_i}$ then          #Assumption: minimization of $f$
10                 $\mathcal{B}_i \leftarrow \mathcal{O}_i$; $f_{\mathcal{B}_i} \leftarrow f_{\mathcal{O}_i}$
11             else
12                 $\mathcal{O}_i \leftarrow \mathcal{B}_i$; $f_{\mathcal{O}_i} \leftarrow f_{\mathcal{B}_i}$
13         else
14             $\mathcal{B}_i \leftarrow \mathcal{O}_i$
```

Iterations are repeated until no more merging is possible, that is, $\mathcal{C} = \emptyset$. This necessarily happens in $2\ell - 1$ iterations. Note that the last iterations sets the root of the LT, that is, the subset that contains all genotype locations: $\{1, \ldots, \ell\}$. Note also that the structure of the LT is related to the structure of the tree-like genotype of GP solutions only in the sense that the LT contains $2\ell - 1$ FOS subsets and the genotype has length ℓ, but it is not a one-to-one match to the structure of the genotype.

With the efficient implementation of UPGMA by RNN, the time complexity to build the LT remains bounded by $O(n^{\text{pop}}\ell^2)$.

3.2.2 Random Tree

While linkage learning assumes an inherent structural inter-dependency to be present within the genotype that can be captured in an LT, such hypothesis may not be true. In such a scenario, using the LT might be not better than building a similar FOS in a completely random fashion. The RT is therefore considered to test this. The RT shares the same tree-like structure of the LT, but is built randomly rather than using mutual information (taking $O(\ell)$). We use the RT as an alternative FOS for GP-GOMEA.

3.3 Gene-Pool Optimal Mixing

Once the FOS is learned, the variation operator GOM generates the offspring population. GOM varies a given solution $\mathcal{P}_i$ in iterative steps, by overriding the nodes at the locations specified by each F_j in the FOS, with the nodes in the same locations taken from random donors in the population. Selection is performed within GOM in a hill-climbing fashion, that is, variation attempts that result in worse fitness are undone.

The pseudocode presented in Algorithm 2 describes GOM in detail. To begin, a backup $\mathcal{B}_i$ of the parent solution $\mathcal{P}_i$ is made, including its fitness, and similarly an offspring solution $\mathcal{O}_i = \mathcal{P}_i$ is created. Next, the FOS F is shuffled randomly: this is to provide different combinations of variation steps along the run and prevent bias. For each set of node locations F_j, a random donor $\mathcal{D}$ is then picked from the population, and $\mathcal{O}_i$ is changed by replacing the nodes specified by F_j with the homologous ones from $\mathcal{D}$. This process is exemplified in Figure 2. It is then assessed whether at least one (syntactic) non-intron node of the tree has been changed by variation (indicated by $\neq^\star$ in line 7). When that is not the case, $\mathcal{O}_i$ will have the same behavior as $\mathcal{B}_i$, thus the fitness

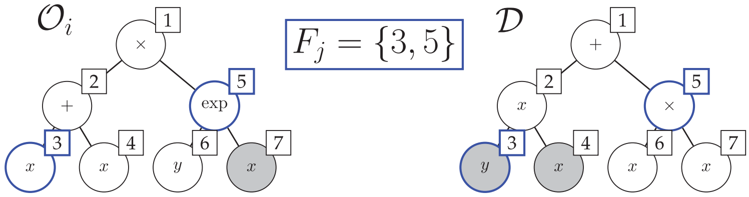


Figure 2: Example of variation step performed by GOM for trees with $h = 2$. Squares on top of each node indicate the node location according to pre-order traversal (depth-first). GOM replaces the nodes of $\mathcal{O}_i$ of which the location is specified by F_j, with the homologous nodes of $\mathcal{D}$ (blue contour).

Table 1: General parameter settings for the experiments.

Parameter	Setting
Function set	$\{+, -, \times, \div_{\text{AQ}}\}$
Terminal set	$\mathbf{x} \cup \{\text{ERC}\}$
ERC bounds	$[\min \mathbf{x}, \max \mathbf{x}]$
Initialization for GP-GOMEA	Half-and-Half as in Virgolin et al. (2018)
Tree height h	4
Train-validation-test split	50%–25%–25%
Experiment repetitions	30

is necessarily identical. Otherwise, the new fitness $f_{\mathcal{O}_i}$ is computed: if not worse than the previous one, the change is kept, and the backup is updated, otherwise the change is reversed.

Note that if a change results in $f_{\mathcal{O}_i} = f_{\mathcal{B}_i}$, the change is kept. This allows random walks in the neutral fitness landscape (Ebner et al., 2001; Sadowski et al., 2013). Note also that differently from traditional subtree crossover and subtree mutation (Koza, 1992), GOM can change unconnected nodes at the same time, and keeps tree height limited to the initially specified parameter h. Finally, GOM *does not consider* any FOS subset that contains all node locations, that is, $F_j = \{1, \dots, \ell\}$, as using such subset would mean to entirely replace $\mathcal{O}_i$ with $\mathcal{D}$.

4 General Experimental Settings

We now describe the general parameters that will be used in this article. Table 1 reports the parameter settings which are typically used in the following experiments, unless specified otherwise. The notation $\mathbf{x}$ represents the matrix of feature values. We use the Analytic Quotient (AQ) (Ni et al., 2013) instead of protected division. This is because the AQ is continuous in 0 for the second operand: $x_1 \div_{\text{AQ}} x_2 := x_1/\sqrt{1 + x_2^2}$. Albeit continuity is not needed by many GP variation operators (including GOM), it is useful at prediction time: Ni et al. (2013) show that using the AQ helps generalization (whereas using protected division does not). However, the AQ may be considered relatively hard to interpret.

Table 2: Regression datasets used in this work.

Name	Abbreviation	# Features	# Examples
Airfoil	Air	5	1503
Boston housing	Bos	13	506
Concrete compres. str.	Con	8	1030
Dow Chemical	Dow	57	1066
Energy cooling	EnC	8	768
Energy heating	EnH	8	768
Tower	Tow	25	4999
Wine red	WiR	11	1599
Wine white	WiW	11	4898
Yacht hydrodynamics	Yac	6	308

As mentioned in the introduction, we focus on the evolution of solutions that are constrained to be small, to *enable* interpretability. We choose $h = 4$ because this results in relatively balanced trees with up to 31 nodes (since $r = 2$). We consider this size limitation a critical value: for the given function set, we found solutions to be already borderline interpretable for us (this is discussed further in Section 9). Larger values for h would therefore play against the aim of this study. When benchmarking GP-GOMEA in Section 8, we also consider $h = 3$ and $h = 5$ for completeness.

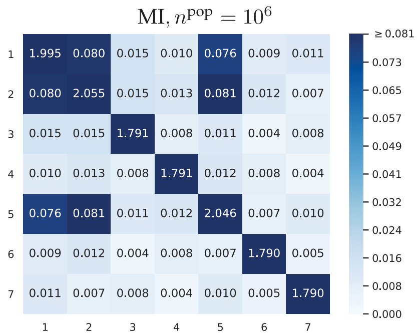
We consider 10 real-world benchmark datasets from literature (Martins et al., 2018) that can be found on the UCI repository[2] (Asuncion and Newman, 2007) and other sources.[3] The characteristics of the datasets are summarized in Table 2.

We use the linearly scaled Mean Squared Error (MSE) to measure solution fitness (Keijzer, 2003), as it can be particularly beneficial when evolving small solutions. This means a fast (cost $O(n)$ with n number of dataset examples) linear regression is applied between the target y and the solution prediction $\tilde{y}$ prior to computing the MSE. We present our results in terms of variance-Normalized MSE (NMSE), that is, $\frac{\text{MSE}(y,\tilde{y})}{var(y)}$, so that results from different datasets are on a similar scale.

To assess statistical significance when comparing the results of multiple executions of two algorithms (or configurations) on a certain dataset, we use the Wilcoxon signed-rank test (Demšar, 2006). This test is set up to compare competing algorithms based on the same prior conditions. In particular, we employ pairs of executions where the dataset is split into identical training, validation, and test sets for both algorithms being tested. This is because the particular split of data determines the fitness function (based on the training set), and the achievable generalization error (for the validation and test sets). We consider a difference to be significant if a smaller p-value than $0.05/\beta$ is found, with β the Bonferroni correction coefficient, used to prevent false positives. If more than two algorithms need to be compared, we first perform a Friedman test on mean performance over all datasets (Demšar, 2006). We use the symbols ▲, ▼ to respectively indicate significant superiority, and inferiority (absence of a symbol means no significant difference). The result *next* to the symbol ▲ (▼) signifies a result being better (worse) than the result obtained by the algorithm that has the same color of the sym-

[2] https://archive.ics.uci.edu/ml/index.php

[3] https://goo.gl/tn6Zxv



	1	2	3	4	5	6	7
1	1.995	0.080	0.015	0.010	0.076	0.009	0.011
2	0.080	2.055	0.015	0.013	0.081	0.012	0.007
3	0.015	0.015	1.791	0.008	0.011	0.004	0.008
4	0.010	0.013	0.008	1.791	0.012	0.008	0.004
5	0.076	0.081	0.011	0.012	2.046	0.007	0.010
6	0.009	0.012	0.004	0.008	0.007	1.790	0.005
7	0.011	0.007	0.008	0.004	0.010	0.005	1.790

Figure 3: Mutual information matrix between pairs of locations in the genotype (x and y labels). Darker blue represents higher values. The matrix is computed for an initialized population of size 10^6. The values suggest the existence of linkage even though no evolution has taken place yet.

bol. Algorithms and/or configurations are color coded in each table reporting results (colors are color-blind safe).

5 Improving Linkage Learning for GP

In previous work on GP-GOMEA, learning the LT was performed the same way it is done for any discrete GOMEA implementation, that is, by computing the mutual information between pairs of locations (i, j) in the genotype (Eq. 1) (Virgolin et al., 2017). However, the distribution of node types is typically not uniform when a GP population is initialized (e.g., function nodes never appear as leaves). In fact, this depends on the cardinality of the function and terminal sets, on the arity of the functions, and on the population initialization method (e.g., *Full*, *Grow*, *Half-and-Half*, *Ramped Half-and-Half* (Luke and Panait, 2001)). Note that it does not depend on the particular dataset in consideration (except in that the number of features determines the size of the terminal set). The lack of uniformity in the distribution leads to the emergence of mutual information between particular parts of the genotype. Crucially, this mutual information is natural to the solution representation, the sets of symbols and the initialization process.

If mutual information is used to represent linkage, then linkage will already be observed at initialization. However, it is reasonable to expect no linkage to be present in an initialized population, as evolution did not take place yet. Figure 3 shows the mutual information matrix between pairs of node locations in an initial population of 1,000,000 solutions with maximum height $h = 2$, using *Half-and-Half*, a function set of size 4 with maximum number of inputs $r = 2$, and a terminal set of size 6 (no ERCs are used). Each tree contains exactly 7 nodes. We index node locations with pre-order tree transversal; that is, 1 is the root, 2 its first child, 5 its second child, 3,4 are (leaves) children of 2, and 6,7 are (leaves) children of 5. Nodes at locations 2 and 5 can be functions only if a function is sampled at node 1. It can be seen that the mutual information matrix of location pairs (correctly) captures the non-uniformity in the initial distribution (i.e., larger mutual information values are present between non-leaf nodes). Using mutual information directly as a proxy for linkage may be undesirable.

5.1 Biasing Mutual Information to Represent Linkage

We propose to overcome the aforementioned problem by measuring linkage with a modified version of the mutual information, such that no linkage is measured at initialization. Our hypothesis is that, if we apply such a correction so that no patterns are identified at initialization, the truly salient patterns will have a bigger chance of emerging during evolution, and better results will be achieved.

Let us consider the scenario where, at initialization, symbols are uniformly distributed. For example, this typically happens in binary genetic algorithms. The mutual information between pairs of genotype locations that is expected at initialization, that is, at generation $g = 1$ before variation and selection, will then correspond to the identity matrix: $\text{MI}^g|_{g=1} = I$ (assuming binary symbols and mutual information in bits as well as a sufficiently large population size). This mutual information matrix is suitable to represent linkage as no linkage should be present at initialization.

We propose to adopt a biased mutual information matrix $\text{MI}_b(i, j)$ to represent the linkage between a pair of genotype locations (i, j), that has the property:

$$\text{MI}_b^g(i, j)|_{g=1} = I,$$

no matter the actual distribution of the initial population.

To this end, we use Equation 1, that is, we manipulate the entropy terms, to represent maximal randomness to be present at initialization for each genotype location. In particular, we propose to use biased entropy metrics such that $\text{H}_b^g(i)|_{g=1} = 1$ and $\text{H}_b^g(i, j)|_{g=1} = 2$ (for $i \neq j$), since

$$\begin{aligned}\text{MI}_b^g(i, j)|_{g=1} &= \left(\text{H}_b^g(i) + \text{H}_b^g(j) - \text{H}_b^g(i, j)\right)|_{g=1} \\ &= 1 + 1 - 2 = 0 \qquad\qquad (\text{for } i \neq j, \text{ else } 1).\end{aligned}$$

We propose to use linear biasing coefficients β_i ($\beta_{i,j}$) to have the general biased entropy for any generation g as $\text{H}_b^g(i) = \beta_i \text{H}(i)$ and $\text{H}_b^g(i, j) = \beta_{i,j}\text{H}(i, j)$, with $\beta_i = \left(\text{H}_b^g(i)|_{g=1}\right)^{-1}$ and $\beta_{i,j} = 2\left(\text{H}^g(i, j)|_{g=1}\right)^{-1}$ to enforce maximal randomness at initialization.

To determine the beta coefficients *exactly* means to know the true distribution inferred by the sampling process used to sample the initial population, and thus the true initial entropy for each genotype location. However, this is generally not trivial to determine for GP, since a number of factors need to be considered. For example, if the *Ramped Half-and-Half* initialization method is used, what symbol is sampled at a location depends on the chance to use *Full* or *Grow*, the chance to pick the function or the terminal set based on the depth, the size of these sets, and possibly other problem-specific factors. Hence, we propose to simply approximate the β coefficients by using the $\text{H}^g(i)|_{g=1}$ measured on the initial population, assuming the population to be large enough.

Summing up, the pairwise linkage estimation we propose to use at generation g, for a pair of locations (i, j), will be:

$$\text{MI}_{\tilde{b}}^g(i, j) = \beta_i \text{H}^g(i) + \beta_j \text{H}^g(j) - \beta_{i,j}\text{H}^g(i, j). \tag{2}$$

The tilde in $\tilde{b}$ is to remark that this is an approximation.

5.2 Estimation of Linkage by $\text{MI}_{\tilde{b}}$

As a preliminary step, we observe what linkage values are obtained between pairs of genotype locations by using $\text{MI}_{\tilde{b}}$. For conciseness, in the following we denote $\text{MI}_{\tilde{b}}^g(i, j)|_{g=\Gamma}$ with $\text{MI}_{\tilde{b}}^\Gamma(i, j)$. We show the MI matrix computed at the second generation of a GP-GOMEA run on the dataset Yac ($\text{MI}_{\tilde{b}}^1 = I$ by construction). We do this for

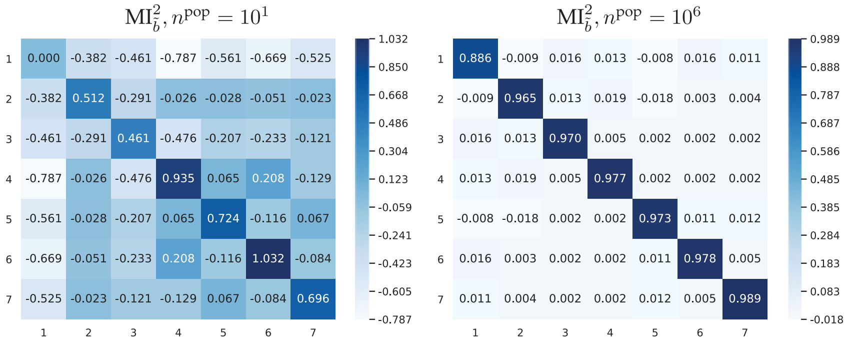


Figure 4: Mutual information matrices at the second generation using our biasing method to better represent linkage, with population size of 10 (left), and of 10^6 (right) for a particular run of GP-GOMEA. The rightmost matrix is closest to the identity I.

two population sizes, $n^{\mathrm{pop}} = 10$ and $n^{\mathrm{pop}} = 10^6$. We expect that, the bigger n^{pop} is, the closer $\mathrm{MI}^2_{\tilde{b}}$ is to I.

We use the parameters of Table 1, a terminal set of size 6 (the features of Yac, no ERC) and $h = 2$; that is, $\ell = 7$ nodes per tree. Figure 4 shows the biased mutual information matrix between location pairs, for the two population sizes. It can be seen that the values can be lower than 0 or bigger than 1. However, while this is particularly marked for $n^{\mathrm{pop}} = 10$, with minimum of -0.787 and maximum of 1.032, it becomes less evident for $n^{\mathrm{pop}} = 10^6$, with minimum of -0.018 and maximum of 0.989. The fact that $\mathrm{MI}^2_{\tilde{b}} \approx I$ for $n^{\mathrm{pop}} = 10^6$ is because, with such a large population size, considerable diversity is still present in the second generation.

5.3 Experiment: LT–$\mathrm{MI}_{\tilde{b}}$ vs LT–MI vs RT

We now test the use of $\mathrm{MI}_{\tilde{b}}$ over the standard MI for GP-GOMEA with the LT. We denote the two configurations with LT–$\mathrm{MI}_{\tilde{b}}$ and LT–MI. We also consider the RT to see if mutual information drives variation better than random information.

We set the population size to 2000 as a compromise between having enough samples for linkage to be learned, and meeting typical literature values, which range from hundreds to a few thousands. We use the function set of Table 1, and a tree height $h = 4$ (thus $\ell = 31$). We set a limit of 20 generations, which corresponds to approximately 1200 generations of traditional GP, as each solution is evaluated up to $2\ell - 2$ times (size of the LT minus its root and non-meaningful changes, see Sections 3.2 and 3.3).

5.4 Results: LT–$\mathrm{MI}_{\tilde{b}}$ vs LT–MI vs RT

The training and test NMSE performances are reported in Table 3. The Friedman test results in significant differences along training and test performance. GP-GOMEA with LT–$\mathrm{MI}_{\tilde{b}}$ is clearly the best performing algorithm, with significantly lower NMSE compared to LT–MI on 8/10 datasets when training, and 7/10 at test time. It is always better than using the RT when training, and in 9/10 cases when testing. The LT–MI is comparable with the RT for these problems.

The result of this experiment is that the use of the new $\mathrm{MI}_{\tilde{b}}$ to build the LT simply enables GP-GOMEA to perform a more competent variation than the use of MI. Also, using the LT this way leads to better results than when making random changes with the RT. Figure 5 shows the evolution of the training NMSE for the dataset Yac. It can

Table 3: Median NMSE of 30 runs for GP-GOMEA with LT–MI$_{\tilde{b}}$, LT–MI, and RT.

	Training			Test		
Dataset	LT–MI$_{\tilde{b}}$	LT–MI	RT	LT–MI$_{\tilde{b}}$	LT–MI	RT
Air	29.9 ▲▲	31.2 ▼	32.7 ▼	31.8 ▲▲	34.8 ▼	34.0 ▼
Bos	15.4 ▲▲	15.4 ▼▲	17.5 ▼▼	24.0 ▼▼	23.0 ▲	22.5 ▲
Con	17.5 ▲▲	18.5 ▼▲	19.0 ▼▼	18.7 ▲▲	19.6 ▼▲	20.1 ▼▼
Dow	20.9 ▼▲	20.3 ▲▲	24.0 ▼▼	22.6 ▼▲	21.1 ▲▲	26.0 ▼▼
EnC	8.42 ▲▲	9.68 ▼▼	9.09 ▼▲	9.18 ▲▲	10.7 ▼▼	10.3 ▼▲
EnH	6.24 ▲▲	6.44 ▼▼	6.40 ▼▲	6.50 ▲▲	7.10 ▼▼	6.70 ▼▲
Tow	12.5 ▼▲	12.5 ▲▲	13.1 ▼▼	13.0 ▲	12.8 ▲	13.2 ▼▼
WiR	60.3 ▲▲	60.9 ▼▲	61.2 ▼▼	62.5 ▲▲	63.0 ▼	63.1 ▼
WiW	68.1 ▲▲	68.4 ▼	68.7 ▼	69.1 ▲▲	69.7 ▼▼	69.5 ▼▲
Yac	0.34 ▲▲	0.37 ▼	0.36 ▼	0.58 ▲▲	0.62 ▼▼	0.62 ▼▲

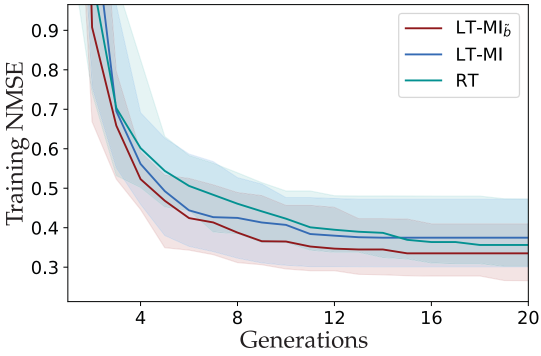


Figure 5: Median fitness of the best solution of 30 runs on Yac, for LT–MI$_{\tilde{b}}$, LT–MI, and RT (10th and 90th percentiles in shaded area).

be seen that the LT–MI$_{\tilde{b}}$ can more quickly reach smaller errors than the other two FOS types. We observed similar training patterns for the other datasets (not shown here).

In the remainder, when we write "LT", we refer to LT–MI$_{\tilde{b}}$.

5.5 Experiment: Assessing Propagation of Node Patterns

The previous experiment showed that using linkage-driven variation (LT) can be favorable compared to random variation (RT). This seems to confirm the hypothesis that, in certain SR problems, salient underlying patterns of nodes exist in the genotype that can be exploited. Another aspect that can be considered with regard to such hypothesis is how final solutions look: if linkage learning identifies specific node patterns, it can be expected that their propagation will lead to the discovery of similar solutions over different runs.

Therefore, we now want to assess whether the use of the LT has a bigger chance to lead to the discovery of a particular best-of-run solution, compared to the use of the RT. We use the same parameter setting as described in Section 5.3, but perform 100 repetitions. While each run uses a different random seed (e.g., for population initialization), we fix the dataset split, as changing the training set results in changing the fitness function. We repeat the 100 runs on 5 random dataset splits, on the smallest dataset Yac. Together with $n^{\text{pop}} = 2000$ as in the previous experiment, we also consider a doubled $n^{\text{pop}} = 4000$.

Table 4: Percentage of best solutions with duplicates found by GP-GOMEA with LT and RT for different splits of Yac.

	$n^{\text{pop}} = 2000$		$n^{\text{pop}} = 4000$	
Split	LT	RT	LT	RT
1	36	18	44	15
2	42	12	49	21
3	40	7	43	8
4	43	8	45	25
5	36	16	49	16
Avg.	39	12	46	17

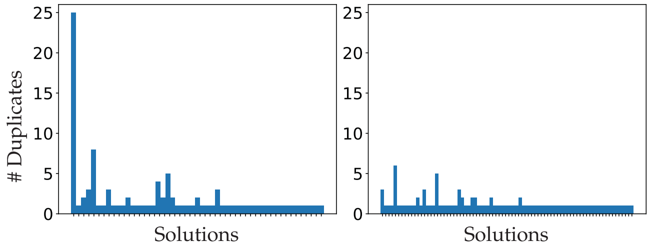


Figure 6: Distribution of best found solutions for 100 runs by using the LT (left) and the RT (right) with $n^{\text{pop}} = 4000$ on the second dataset split of Yac.

Table 4 reports the number of best found solutions that have at least one duplicate, that is, their genotype is semantically equivalent (e.g., $x_1 + x_2 = x_2 + x_1$), along different runs for 5 random splits of Yac (semantic equivalence was determined by automatic tests[4] followed by manual inspection). It can be seen that the LT finds more duplicate solutions than the RT, by a margin of around 30% (difference between averages). Figure 6 shows the distribution of solutions found for the second dataset split with $n^{\text{pop}} = 4000$, that is, where both the LT and the RT found a large number of duplicates. The LT has a marked chance of leading to the discovery of a particular solution, up to one-fourth of the times. When the RT is used, a same solution is found only up to 6 times out of 100.

This confirms the hypothesis that linkage-based variation can propagate salient node patterns more than random variation should such patterns exist, enhancing the likelihood of discovering particular solutions.

6 Ephemeral Random Constants and Linkage

In many GP problems, and in particular in SR, the use of ERCs can be very beneficial (Poli et al., 2008). An ERC is a terminal which is set to a constant only when

[4] Including the use of symbolic simplification with https://andrewclausen.net/computing/deriv.html.

Table 5: Median training NMSE and median test NMSE of 30 runs for GP-GOMEA with the LT using the three strategies all-const, no-const, bin-const, and with the RT.

	Training NMSE				Test NMSE			
Dataset	all-const	no-const	bin-const	RT	all-const	no-const	bin-const	RT
Air	27.7 ▼▲	28.0 ▼▲	27.5 ▲▲▲	31.4 ▼▼▼	28.7 ▲▼▲	29.6 ▼▼▲	27.8 ▲▲▲	32.5 ▼▼▼
Bos	15.2 ▼▲	15.3 ▲	15.0 ▲ ▲	17.6 ▼▼▼	24.2 ▼▼	23.2 ▼▲	21.8 ▲▲▲	24.2 ▲▲▼
Con	17.2 ▲▼▲	17.2 ▼▼▲	17.0 ▲▲▲	18.5 ▼▼▼	18.5 ▲	18.7 ▲	18.8 ▲	19.8 ▼▼▼
Dow	21.4 ▼▲	21.1 ▲	20.7 ▲ ▲	24.5 ▼▼▼	22.8 ▼ ▲	21.9 ▲▲▲	22.5 ▼▲	25.5 ▼▼▼
EnC	5.51 ▲▲▲	5.72 ▼ ▲	5.76 ▼ ▲	6.44 ▼▼▼	6.18 ▲ ▲	6.34 ▼▼▲	6.00 ▲▲	6.77 ▼▼▼
EnH	3.00 ▲▼▲	3.14 ▼▼▲	2.80 ▲▲▲	4.10 ▼▼▼	3.28 ▲▼▲	3.33 ▼▼▲	3.11 ▲▲▲	4.67 ▼▼▼
Tow	12.3 ▼▲	12.2 ▲	12.3 ▲ ▲	13.2 ▼▼▼	12.9 ▲	12.8 ▲	12.8 ▲	13.5 ▼▼▼
WiR	60.3 ▲▲	60.2 ▲	60.2 ▼ ▲	61.2 ▼▼▼	63.6 ▲	62.9	62.9 ▲	63.2 ▼ ▼
WiW	67.6 ▲▲▲	68.1 ▼▼▲	68.0 ▼▲▲	68.5 ▼▼▼	68.9 ▲	69.0 ▲	69.4 ▲	69.9 ▼▼▼
Yac	0.32 ▲▲▲	0.35 ▼▼▲	0.34 ▼▲▲	0.38 ▼▼▼	0.55 ▲	0.61 ▼▲	0.52 ▲▲	0.63 ▼▼▼

instantiated in a solution. In SR, this constant is commonly sampled uniformly at random from a user-defined interval.

Because every node instance of ERC is a different constant, linkage learning needs to deal with a large number of different symbols. This can lead to two shortcomings. First, a very large population size may be needed for salient node patterns to emerge. Second, data structures used to store the frequencies of symbols grow really big and become slow (e.g., hash maps).

We explore three strategies to deal with this: all-const: Ignore the shortcomings, and consider all different constants as different symbols during linkage learning; no-const: Skip all constants during linkage learning, that is, set their frequency to zero. This approximation is reasonable since all constants are unique at initialization, and the respective frequency is almost zero. However, during evolution some constants will be propagated while others will be discarded, making this approximation less and less accurate over time; bin-const: Perform on-line binning. We set a maximum number γ of constants to consider. After γ different constants have been encountered in frequency counting, any further constant is considered to fall into the same bin as the closest constant among the first γ. The closest constant can be determined with binary search in $\log_2(\gamma)$ steps. Contrary to strategy no-const, we expect the error of this approximation to lower over time, because selection lowers diversity, meaning that the total number of different constants will be reduced as generations pass.

6.1 Experiment: Linkage Learning with ERCs

We use the same parameter setup of the experiment in Section 5.3, this time adding an ERC terminal to the terminal set. We compare the three strategies to handle ERCs when learning the LT. For this experiment and in the rest of the article, we use $\gamma = 100$ in bin-const. We observed that for problems with a small number of features (e.g., Air and Yac), that is, where ERC sampling is more likely and thus more constants are produced, this choice reduces the number of constant symbols to be considered by linkage learning in the first generations by a factor of $\sim$50. We also report the results obtained with the RT as a baseline, under the hypothesis that using ERCs compromises linkage learning to the point that random variation becomes equally good or better.

The results of this experiment are shown in Table 5 (training and test NMSE) and Table 6 (running time). The Friedman test reveals significant differences among the configurations for train, test, and time performance. Note that the use of ERCs leads to lower errors compared with not using them (compare with Table 3).

Table 6: Median time of 30 runs for GP-GOMEA with the LT using the three strategies all-const, no-const, bin-const, and with the RT.

	Time (s)			
Dataset	all-const	no-const	bin-const	RT
Air	355.4 ▼▼▼	71.4 ▲▲▲	80.0 ▲▼▲	80.1 ▲▼▼
Bos	63.4 ▼▼▼	29.4 ▲▲▼	30.9 ▲▼▼	24.5 ▲▲▲
Con	154.9 ▼▼▼	56.7 ▲▲	59.8 ▲▼	58.4 ▲
Dow	53.8 ▼▲▼	51.7 ▲▼	54.9 ▼▼▼	37.7 ▲▲▲
EnC	147.2 ▼▼▼	40.5 ▲▲▲	43.5 ▲▼▲	45.6 ▲▼▼
EnH	145.0 ▼▼▼	45.8 ▲▲	49.4 ▲▼▼	45.7 ▲ ▲
Tow	255.9 ▼▼▼	246.6 ▲ ▼	245.6 ▲ ▼	233.9 ▲▲▲
WiR	126.1 ▼▼▼	67.7 ▲▲	80.2 ▲▼▼	70.1 ▲ ▲
WiW	285.0 ▼▼▼	213.3 ▲▲▲	237.2 ▲▼▼	224.1 ▲▼▲
Yac	236.5 ▼▼▼	23.9 ▲▲▼	24.8 ▲▼▼	22.8 ▲▲▲

In terms of training error, the RT is always outperformed by the use of the LT, no matter the strategy. The all-const strategy is significantly better than no-const in half of the problems, and never worse. Overall, bin-const performs best, with 6 out of 10 significantly better results than all-const. The fact that all-const can be outperformed by bin-const supports the hypothesis that linkage learning can be compromised by the presence of too many constants to consider, which hide the true salient patterns. Test results are overall similar to the training ones, but less comparisons are significant.

In terms of time, all-const is almost always significantly worse than the other methods, and often by a consistent margin. This is particularly marked for problems with a small number of features (i.e., Air, Yac). There, more random constants are present in the initial population, since the probability of sampling the ERC from the terminal set is inversely proportional to the number of features.

Interestingly, despite the lack of a linkage-learning overhead, using the RT is not always the fastest option. This is because random variation leads to a slower convergence of the population compared with the linkage-based one, where salient patterns are quickly propagated, and fewer variation attempts result in changes of the genotype that require a fitness evaluation (see Section 3.3). The slower convergence caused by the RT can also be seen in Figure 5 (for the previous experiment), and was also observed in other work, in terms of diversity preservation (Medvet, Virgolin et al., 2018).

Between the LT-based strategies, the fastest is no-const, at the cost of a bigger training error. Although consistently slower than no-const, bin-const is still quite fast, and achieves the lowest training errors. We found bin-const to be preferable in test NMSE as well. In the following, we always use bin-const, with $\gamma = 100$.

7 Interleaved Multistart Scheme

The Interleaved Multistart Scheme (IMS) is a wrapper for evolutionary runs largely inspired by the work of Harik and Lobo (1999) on genetic algorithms. It works by interleaving the execution of several runs of increasing resources (e.g., population size). The main motivation for using the IMS is to make an EA much more robust to parameter settings, and alleviate the need for practitioners to tinker with parameters. In fact, the whole design of GP-GOMEA attempts to promote the aspects of ease-of-use and robustness: the EA has no need for parameters that specify how to conduct variation (e.g., crossover or mutation rates), nor how to conduct selection (e.g., tournament size). The IMS or similar schemes are often used with MBEAs (Lin and Yu, 2018; Goldman and Punch, 2014), where population size plays a crucial role in determining the quality of model building. Note that although the IMS has potential to be parallelized, here it is used in a sequential manner.

An IMS for GP-GOMEA was first proposed in Virgolin et al. (2017), and its outline is as follows. A collection of parameter settings σ_{base} is given as input, which will be used in the first run R_1. The IMS runs until a termination criterion is met (e.g., number of generations, time budget). The run R_i performs one generation if no run that precedes it exists (e.g., because it is the first run or because all previous runs have been terminated), or if the previous run R_{i-1} has executed g generations. The first time R_i is about to execute a generation, it is initialized using the parameter settings σ_{base} scaled by the index i. For example, the population size can be set to $2^{i-1}n^{\text{pop}}_{\text{base}}$ (i.e., doubling the population size of the previous run). Finally, when a run completes a generation, a check is done to determine if the run should be terminated (explained below).

7.1 An IMS for Supervised Learning Tasks

The first implementation of the IMS for GP-GOMEA was designed to deal with GP benchmark problems of pure optimization. That implementation therefore scaled both the population size and the height of trees in an attempt to find the optimal solution (of unknown size) (Virgolin et al., 2017).

In this work, we use the IMS as follows. *Scaling of parameter settings*: We scale only the population size. For run R_i, the population size is set to $n^{\text{pop}}_i = 2^{i-1}n^{\text{pop}}_{\text{base}}$. *Run termination*: A run R_i is terminated if the fitness of its best solution is worse than the one of a run R_j initialized later, that is, with $j > i$, or if it converges to all identical solutions.

Differently from Virgolin et al. (2017) we no longer scale the tree height h because in SR, and in supervised learning tasks in general, no optimum is known beforehand, and it is rather desired to find a solution that generalizes well to unseen examples. Moreover, h bounds the maximum solution size, which influences interpretability. Hence h is left as a parameter for the user to set, and we recommend $h \leq 4$ to increase the chance that solutions will be interpretable (see Section 9).

We set the run termination criteria to be based upon the fitness of best solutions instead of mean population fitness as done by Harik and Lobo (1999) and Virgolin et al. (2017), because in SR it can happen that the error of a few solutions becomes so large that it compromises the mean population fitness. This can trigger the termination criteria even if solutions exist that are competitive with the ones of other runs. Also differently from the other versions of the IMS, when terminating a run, we do not automatically terminate all previous runs. Indeed, some runs with smaller parameter settings may still be very competitive (e.g., due to the fortunate sampling of particular constants when using ERCs).

We lastly propose to exploit the fact that many runs are performed within the IMS to tackle a central problem of learning tasks: generalization. Instead of discarding the best solutions of terminating runs, we store them in an archive. When the IMS terminates, we recompute the fitness of each solution in the archive using a set of examples different from the training set, that is, the validation set, and return the new best performing, that is, the solution that generalized best. The final test performance is measured on a third, separate set of examples (test set).

8 Benchmarking GP-GOMEA

We compare GP-GOMEA (using the new LT) with tree-based GP with traditional subtree crossover and subtree mutation (GP-Trad), tree-based GP using the state-of-the-art, semantic-aware operator Random Desired Operator (GP-RDO) (Pawlak et al., 2015), and Decision Tree for Regression (DTR) (Breiman et al., 1984).

We consider RDO because, as mentioned in the introduction, semantic-aware operators have been studied with interest in the last years. Several works either built upon RDO, or used RDO as a baseline for comparison (see, for example, Chen et al., 2018; Pawlak and Krawiec, 2018; and Virgolin et al., 2019). Yet, consistently large solutions were found. It is interesting to assess how RDO fares when rather strict solution size limits are enforced. Because of such limits, we remark we cannot consider another popular set of semantic-aware operators, that is, the operators used by Geometric Semantic Genetic Programming (GSGP) (Moraglio et al., 2012). These operators work by stacking entire solutions together, necessarily causing extremely large solution growth (even if smart simplifications are attempted (Martins et al., 2018)).

We consider DTR because it is considered among the state-of-the-art algorithms to learn interpretable models (Doshi-Velez and Kim, 2017; Guidotti et al., 2018). We remark that DTR ensembles (e.g., Breiman, 2001; and Chen and Guestrin, 2016) are typically markedly more accurate than single DTRs, but are considered not interpretable.

8.1 Experimental Setup

For the EAs, we use a fixed time limit of 1,000 seconds.[5] We choose a time-based comparison because GP-GOMEA performs more evaluations per generation than other GP algorithms (up to $2\ell - 2$ evaluations per generation with the LT), and so that the overhead of learning the LT (which does not involve evaluations) is taken into account.

We consider maximum solution sizes $\ell = 15, 31, 63$ (tree nodes), that is, corresponding to $h = 3, 4, 5$ respectively, for full r-ary trees. The EAs are run with a typical fixed population size $n^{\text{pop}} = 1000$ and also with the IMS, considering three values for the number of generations in between runs g: 4, 6, and 8. For the fixed population size, if the population of GP-GOMEA converges before the time limit, since there is no mutation, it is randomly restarted. Choices of g between 4 and 8 are standards from literature (Bouter et al., 2017; Virgolin et al., 2017).

Our implementation of GP-Trad and GP-RDO mostly follows the one of Pawlak et al. (2015). The population is initialized with the *Ramped Half-and-Half* method, with tree height between 2 and h. Selection is performed with tournament of size 7. GP-Trad uses a rate of 0.9 for subtree crossover, and of 0.1 for subtree mutation. GP-RDO uses the population-based library of subtrees, a rate of 0.9 for RDO, and of 0.1 for subtree mutation. Subtree roots to be variated are chosen with the *uniform depth mutation* method, which makes nodes of all depths equally likely to be selected (Pawlak et al., 2015). Elitism is ensured by cloning the best solution into the next generation. All EAs are implemented in C^{++}, and the code is available at: https://goo.gl/15tMV7.

For GP-Trad we consider two versions, to account for different types of solution size limitation. In the first version, called GP-Trad^h, we force trees to be constrained within a maximum height ($h = 3, 4$), as done for GP-GOMEA. This way, we can see which algorithm searches better in the same representation space. In the second version, GP-Trad$^\ell$, we allow more freedom in tree shape, by only bounding the number of tree nodes. This limit is set to the maximum number of nodes obtainable in a full r-ary tree of height h ($\ell = 15$ for $h = 3$, $\ell = 31$ for $h = 4$). As indicated by previous literature (Gathercole and Ross, 1996; Langdon and Poli, 1997), and as will be shown later in the results, GP-Trad$^\ell$ outperforms GP-Trad^h. We found that the same holds also for GP-RDO, and present

[5] Experiments were run on an Intel® Xeon® Processor E5-2650 v2.

Table 7: Median validation and test NMSE of 30 runs with $\ell = 15$ for GP-GOMEA (G), GP-Trad^h (T$^\ell$), GP-Trad^h (T$^\ell$), GP-RDO (R) with $n^{\text{pop}} = 1000$ and IMS with $g \in \{4, 6, 8\}$, and DTR. Significance is assessed within each population scheme with regard to GP-GOMEA. The last row reports the number of times the EA performs significantly better (B) and worse (W) than GP-GOMEA.

Validation $\ell = 15$

		$n^{\text{pop}} = 1000$				IMS $g = 4$				IMS $g = 6$				IMS $g = 8$			
	G	T^h	T$^\ell$	R	G	T^h	T$^\ell$	R	G	T^h	T$^\ell$	R	G	T^h	T$^\ell$	R	D
Air	39.2	40.6▼	35.0▲	44.0▼	34.7	38.3▼	31.4▲	42.5▼	34.9	39.7▼	33.6▲	42.0▼	34.4	39.4▼	32.0▲	42.3▼	31.1▲
Bos	21.1	23.4	25.3▼	25.7▼	18.2	21.2▼	19.0▼	20.8▼	19.2	21.2▼	20.4	20.9▼	19.4	21.6▼	19.9▼	22.5▼	22.9▼
Con	23.2	25.3▼	23.4	27.0▼	20.3	23.1▼	19.4▲	26.4▼	20.2	23.3▼	19.9▲	26.2▼	19.4	23.2▼	19.3	26.9▼	22.7▼
Dow	26.7	28.5▼	27.5	30.6▼	24.2	26.8▼	24.2	32.3▼	24.6	26.4▼	24.8▼	31.0▼	24.5	26.3▼	25.2▼	31.0▼	30.6▼
EnC	8.72	10.6▼	7.34	11.0▼	5.86	10.2▼	6.49▼	10.7▼	6.01	10.3▼	6.24	10.5▼	5.87	10.2▼	6.10▼	10.8▼	4.23▲
EnH	4.95	7.45▼	3.83▲	7.65▼	3.33	7.19▼	3.74▼	7.34▼	3.28	7.30▼	3.76▼	7.42▼	3.23	7.24▼	3.72▼	7.54▼	0.43▲
Tow	12.9	14.4▼	13.9▼	20.1▼	12.8	13.6▼	13.6▼	20.5▼	12.7	14.0▼	13.5▼	20.4▼	13.0	14.0▼	13.4▼	20.1▼	11.2▲
WiR	65.3	64.8	64.9	66.5▼	63.9	64.7▼	64.4	65.1▼	63.6	63.9▼	64.4▼	64.9▼	63.9	63.9▼	64.2	65.7▼	71.7▼
WiW	71.4	71.3	70.9	72.6▼	70.8	71.2▼	70.7▼	72.3▼	70.7	71.5▼	70.8▼	72.6▼	71.2	71.4▼	71.2▼	72.6▼	72.2▼
Yac	1.25	1.22	0.70▲	0.96▲	0.89	1.04▼	0.61▲	0.67▲	0.92	1.01▼	0.61▲	0.73▲	0.95	1.03▼	0.62▲	0.76▲	0.88▲
B/W	—	0/6	3/2	1/9	—	0/10	3/5	1/9	—	0/10	3/5	1/9	—	0/10	2/6	1/9	5/5

Test $\ell = 15$

		$n^{\text{pop}} = 1000$				IMS $g = 4$				IMS $g = 6$				IMS $g = 8$			
	G	T^h	T$^\ell$	R	G	T^h	T$^\ell$	R	G	T^h	T$^\ell$	R	G	T^h	T$^\ell$	R	D
Air	38.5	40.7▼	35.3▲	44.1▼	35.8	39.5▼	32.5▲	43.3▼	35.2	39.3▼	33.2▲	42.4▼	35.4	39.7▼	32.8▲	42.8▼	30.8▲
Bos	22.7	23.3▼	24.3▼	26.7▼	22.5	23.1	23.3▼	22.9▼	21.7	23.6▼	22.6▼	25.0▼	22.1	22.5▼	23.6▼	23.3▼	26.1▼
Con	23.1	26.1▼	23.9	27.0▼	20.8	23.9▼	19.3▲	27.7▼	21.2	23.9▼	19.9▲	26.4▼	20.4	24.3▼	19.3▲	27.8▼	21.3▼
Dow	26.3	27.5▼	26.4	31.0▼	24.8	26.1▼	24.7▲	30.7▼	24.5	26.6▼	24.5	30.1▼	24.3	26.8▼	25.1	31.6▼	28.0▼
EnC	9.72	11.2▼	7.86▲	11.8▼	6.36	10.6▼	6.80▼	11.5▼	6.37	10.5▼	6.18	10.9▼	6.02	10.5▼	6.33▼	11.7▼	4.47▲
EnH	5.03	7.19▼	4.04▲	7.85▼	3.45	7.57▼	3.88▼	7.62▼	3.28	7.64▼	3.88▼	7.59▼	3.51	7.56▼	3.86▼	7.65▼	0.33▲
Tow	13.4	14.4▼	13.9▼	20.3▼	13.0	14.1▼	14.0▼	20.8▼	12.9	14.1▼	13.7▼	20.7▼	13.0	14.3▼	13.3▼	20.5▼	11.2▲
WiR	63.1	63.7▼	62.4	64.6▼	63.3	63.4	63.2	64.4▼	63.6	63.5	63.2▲	64.3▼	63.4	63.8	63.3	63.7▼	72.6▼
WiW	70.5	70.5	70.1	71.3▼	70.4	70.0▼	70.3▼	71.6▼	69.7	70.5▼	70.1▼	71.0▼	70.2	70.5▼	70.3▼	71.8▼	72.2▼
Yac	1.23	1.23	0.78▲	0.95▲	1.16	1.24▼	0.73▲	0.77▲	1.17	1.24	0.73▲	0.77▲	1.17	1.23▼	0.71▲	0.86▲	0.91▲
B/W	—	0/8	4/2	1/9	—	0/8	4/5	1/9	—	0/8	4/4	1/9	—	0/9	3/5	1/9	5/5

here only its best configuration, that is, a version where the number of tree nodes is limited like for GP-Trad$^\ell$.

We use the Python Scikit-learn implementation of DTR (Pedregosa et al., 2011), with 5-fold cross-validation grid-search over the training set to tune the following hyperparameters: *splitter* ∈ {'*best*','*random*'}; *max_features* ∈ $\{\frac{1}{2}, \frac{3}{4}, 1\}$; *max_depth* ∈ {3,4,5,6} (documentation available at http://goo.gl/hbyFq2). We do not allow larger depth values because, like for GP solutions, excessively large decision trees are uninterpretable. The best generalizing model found by cross-validation is then used on the test set.

8.2 Results: Benchmarking GP-GOMEA

We consider validation and test NMSE. We now show validation rather than training error because the IMS returns the solution which better generalizes to the validation set among the ones found by different runs (same for DTR due to cross-validation). Tables 7, 8, and 9 show the results for maximum sizes $\ell = 15, 31, 63$ ($h = 3, 4, 5$), respectively. On each set of results, the Friedman test reveals significant differences among the algorithms. As we are only interested in benchmarking GP-GOMEA, we test whether significant performance differences exist only between GP-GOMEA and the other algorithms (with Bonferroni-corrected Wilcoxon signed-rank test).

We begin with some general results. Overall, error magnitudes are lower for larger values of ℓ. This is not surprising: limiting solution size limits the complexity of relationships that can be modeled. Another general result is that errors on validation and test set are generally close. Likely, the validation data is a sufficiently accurate surrogate

Table 8: Median validation and test NMSE of 30 runs with $\ell = 31$. Details as in Table 7.

Validation $\ell = 31$

	$n^{\text{pop}} = 1000$				IMS $g = 4$				IMS $g = 6$				IMS $g = 8$				
	G	T^h	T$^\ell$	R	G	T^h	T$^\ell$	R	G	T^h	T$^\ell$	R	G	T^h	T$^\ell$	R	D
Air	26.4	33.8▼	32.1▼	36.0▼	24.9	25.9▼	23.2▲	37.0▼	25.0	27.1▼	24.9	37.6▼	24.8	28.4▼	24.8	37.1▼	31.1▼
Bos	22.3	21.1	19.2	24.8▼	16.7	18.8▼	16.2▼	20.4▼	17.4	18.3▼	17.3▼	20.6▼	17.3	18.4▼	17.6▼	20.4▼	22.9▼
Con	17.3	18.6▼	17.9▼	20.8▼	16.0	17.6▼	16.7▼	20.5▼	16.6	18.1▼	16.4▼	20.1▼	16.1	18.3▼	17.2▼	20.2▼	22.7▼
Dow	21.3	22.6▼	22.6	24.3▼	19.4	21.6▼	19.2	25.6▼	19.4	21.2▼	19.4▼	25.4▼	19.2	21.9▼	20.1▼	25.8▼	30.6▼
EnC	5.14	5.60▼	4.99▲	7.62▼	4.62	5.51▼	4.82▼	8.04▼	4.35	6.04▼	4.56▼	8.48▼	4.37	5.65▼	4.73▼	7.81▼	4.23▲
EnH	2.29	2.54▼	1.75▲	6.21▼	1.95	3.05▼	1.72▲	4.97▼	2.00	2.84▼	1.65▲	5.93▼	1.88	3.10▼	1.62	6.11▼	0.43▲
Tow	12.0	13.0▼	12.6▼	17.5▼	11.8	12.3▼	11.9	17.8▼	11.7	12.2▼	12.2▼	16.6▼	12.0	12.4▼	11.8	17.6▼	11.2▲
WiR	64.2	64.7▼	64.7▼	65.9▼	62.8	62.4▼	62.6	64.5▼	62.3	63.6▼	62.1	64.1▼	62.6	62.9▼	61.8	64.6▼	71.7▼
WiW	70.2	70.4▼	70.9	71.4▼	69.6	69.7	69.7	71.1▼	70.0	70.2▼	70.0	71.0▼	70.0	70.1▼	69.6	71.2▼	72.2▼
Yac	0.46	0.59▼	0.42▲	0.57▼	0.37	0.51▼	0.40▼	0.59▼	0.38	0.56▼	0.38	0.54▼	0.40	0.54▼	0.42▼	0.52▼	0.88▼
B/W	—	0/9	3/4	0/10	—	0/9	2/4	0/10	—	0/10	1/5	0/10	—	0/10	0/5	0/10	3/7

Test $\ell = 31$

	$n^{\text{pop}} = 1000$				IMS $g = 4$				IMS $g = 6$				IMS $g = 8$				
	G	T^h	T$^\ell$	R	G	T^h	T$^\ell$	R	G	T^h	T$^\ell$	R	G	T^h	T$^\ell$	R	D
Air	26.4	33.5▼	30.8▼	37.1▼	25.9	26.5▼	23.3▲	37.6▼	24.9	27.1▼	24.7	39.2▼	24.9	28.8▼	26.1	38.2▼	30.8▼
Bos	21.4	22.8	21.6▼	26.2▼	20.1	21.3	21.8▼	23.4▼	20.9	21.2▼	22.2▼	23.2▼	20.2	22.3▼	22.6▼	26.0▼	26.1▼
Con	17.6	18.7▼	17.8▼	21.5▼	16.9	18.1▼	17.1▼	21.2▼	16.7	18.8▼	16.9	21.1▼	17.2	18.3▼	17.0	21.5▼	21.3▼
Dow	20.3	21.9▼	22.2▼	24.4▼	19.2	20.7▼	19.1	24.4▼	18.9	21.4▼	18.6	24.4▼	18.7	22.2▼	20.2▼	25.5▼	28.0▼
EnC	5.28	5.91▼	4.76▲	7.00▼	4.43	5.76▼	4.79▼	7.69▼	4.44	6.05▼	4.71	8.73▼	4.60	5.62▼	4.77▼	7.94▼	4.47▲
EnH	2.29	2.49▼	1.83▲	5.98▼	2.05	3.20▼	1.58▲	5.12▼	2.10	3.07▼	1.75▲	5.77▼	2.00	2.91▼	1.55▲	6.51▼	0.33▲
Tow	12.2	13.2▼	13.1▼	18.7▼	12.1	12.6▼	12.0▲	18.2▼	12.1	12.4▼	12.3	16.8▼	12.2	12.7▼	12.0	17.2▼	11.2▲
WiR	62.1	63.1	62.1	63.5▼	62.7	63.1▼	61.9	63.9▼	62.4	62.9▼	63.3▼	64.2▼	61.9	63.0▼	62.9▼	63.4▼	72.6▼
WiW	69.0	69.7▼	69.8▼	70.2▼	69.4	69.3	69.2	70.6▼	69.1	69.4▼	69.2▼	70.7▼	69.1	69.6▼	69.3▲	70.5▼	72.2▼
Yac	0.52	0.66▼	0.49▲	0.66▼	0.50	0.58▼	0.47	0.67▼	0.50	0.64▼	0.48▲	0.63▼	0.53	0.63▼	0.48	0.70▼	0.91▼
B/W	—	0/8	3/6	0/10	—	0/8	3/3	0/10	—	0/10	2/3	0/10	—	0/10	2/4	0/10	3/7

Table 9: Median validation and test NMSE of 30 runs with $\ell = 63$. Details as in Table 7.

Validation $\ell = 63$

	$n^{\text{pop}} = 1000$				IMS $g = 4$				IMS $g = 6$				IMS $g = 8$				
	G	T^h	T$^\ell$	R	G	T^h	T$^\ell$	R	G	T^h	T$^\ell$	R	G	T^h	T$^\ell$	R	D
Air	22.6	25.3▼	25.0▼	33.0▼	20.6	22.4▼	20.8	35.1▼	20.7	23.3▼	21.3▼	34.3▼	20.8	24.5▼	20.1	34.2▼	31.1▼
Bos	21.1	19.5	21.9	22.1	16.5	16.8▼	15.7	19.7▼	16.3	17.6▼	15.4▲	18.9▼	16.2	18.5▼	16.7▼	21.2▼	22.9▼
Con	16.6	17.4▼	16.6	18.5▼	15.2	16.1▼	15.7▼	18.5▼	15.5	16.5▼	15.6▼	19.6▼	15.3	16.3▼	15.9▼	19.0▼	22.7▼
Dow	18.6	19.0	18.8▼	21.7▼	17.4	17.8▼	16.7▲	24.1▼	17.7	18.2▼	17.0▲	24.3▼	17.8	19.8	17.6▲	22.4▼	30.6▼
EnC	4.66	5.15▼	4.26▲	5.55▼	3.67	4.37▼	4.14▼	6.92▼	3.85	4.50▼	4.02▼	7.08▼	3.76	4.88▼	3.99▼	6.78▼	4.23▼
EnH	1.65	1.52▼	1.13▲	2.63▼	0.69	1.54▼	0.84▼	4.02▼	0.92	1.78▼	1.02▼	3.81▼	0.87	1.78▼	0.80	3.68▼	0.43▲
Tow	11.5	11.7▼	11.7▼	15.7▼	11.3	10.9	11.1	16.1▼	11.4	11.3	10.9▲	17.0▼	11.3	11.9▼	11.2▲	16.6▼	11.2▲
WiR	64.4	64.6▲	65.2▼	64.3▲	63.0	62.4	62.5	63.8▼	62.3	62.9	62.8	64.6▼	62.7	62.9▼	62.5▼	64.5▼	71.7▼
WiW	70.1	70.1	68.8▲	70.9▼	69.2	69.2	68.7▲	71.1▼	68.9	69.3▼	69.4	71.6▼	69.1	69.7▼	69.6	71.4▼	72.2▼
Yac	0.46	0.45	0.37▲	0.46	0.32	0.38▼	0.33	0.40▼	0.32	0.39▼	0.33	0.44▼	0.33	0.40▼	0.33▲	0.48▼	0.88▼
B/W	—	1/5	4/4	1/7	—	0/7	2/3	0/10	—	0/8	3/4	0/10	—	0/9	3/4	0/10	2/8

Test $\ell = 63$

	$n^{\text{pop}} = 1000$				IMS $g = 4$				IMS $g = 6$				IMS $g = 8$				
	G	T^h	T$^\ell$	R	G	T^h	T$^\ell$	R	G	T^h	T$^\ell$	R	G	T^h	T$^\ell$	R	D
Air	23.0	25.5▼	25.9▼	31.5▼	21.1	22.5▼	19.6	34.9▼	21.7	22.4▼	21.9▼	33.8▼	21.2	23.4▼	21.6	34.1▼	30.8▼
Bos	22.0	20.0	21.2	21.9	19.2	20.7▼	20.4▼	24.1▼	21.5	20.3	20.3▲	25.0▼	19.8	19.7▼	21.4▼	25.8▼	26.1▼
Con	15.9	17.1▼	16.5▼	18.3▼	15.3	16.2▼	15.5	19.1▼	15.3	16.3▼	15.8▼	19.9▼	15.3	16.6▼	16.1▼	18.9▼	21.3▼
Dow	18.3	18.6▼	17.4▲	22.3▼	17.5	17.9	17.0▲	23.7▼	17.6	18.2▼	17.2▲	24.6▼	17.7	18.2	17.9	22.6▼	28.0▼
EnC	4.49	4.70▼	4.24▲	5.63▼	3.77	4.37▼	3.99▼	6.94▼	3.93	4.42▼	3.95▼	7.42▼	3.95	4.85▼	4.20▼	7.37▼	4.47▼
EnH	1.60	1.59▼	1.12▲	2.74▼	0.80	1.52▼	0.89▼	3.73▼	0.88	1.67▼	0.94	4.12▼	0.89	1.92▼	0.93▼	3.71▼	0.33▲
Tow	11.6	12.2▼	12.1▼	15.9▼	11.5	11.4	11.4	16.8▼	11.6	11.5	11.2▲	16.7▼	11.4	12.2▼	11.4	17.1▼	11.2▲
WiR	63.1	63.0	64.4▼	62.9	62.9	62.5▼	61.7	62.5▼	62.5	63.0	62.3	63.6▼	62.7	63.0	61.8▲	63.2▼	72.6▼
WiW	68.7	69.0	68.0▲	69.9▼	68.3	68.6	68.3▲	70.2▼	69.1	69.4	68.2▲	70.6▼	68.2	69.3▼	69.0	70.3▼	72.2▼
Yac	0.44	0.46	0.40▲	0.46	0.41	0.49▼	0.40▲	0.45▼	0.41	0.45▼	0.42	0.52▼	0.46	0.46	0.44▲	0.50▼	0.91▼
B/W	—	0/6	5/4	0/7	—	0/7	3/3	0/10	—	0/6	4/3	0/10	—	0/7	2/4	0/10	2/8

of the test data in these datasets, and solution size limitations make over-fitting unlikely. Finally, note that the results for DTR are the same in all tables.

We now compare GP-GOMEA with GP-Trad^h, focusing on statistical significance tests (see rows "B/W" of the tables), over all size limit configurations. Recall that these two algorithms work with the same type of limitation, that is, based on maximum tree height. No matter the population sizing method, GP-GOMEA is almost always significantly better than GP-Trad^h. GP-GOMEA relies on the LT with improved linkage learning, which we showed to be superior to using the RT, that is, blind variation, in the previous series of experiments (Sections 5.3, 6.1). Subtree crossover and subtree mutation are blind as well, and can only swap subtrees, which may be a limitation.

GP-GOMEA and GP-Trad$^\ell$ are compared next. Recall that GP-Trad$^\ell$ is allowed to evolve any tree shape, as long as the limit in number of nodes is respected. Having this extra freedom, GP-Trad$^\ell$ performs better than GP-Trad^h (not explicitly reported in the tables), which confirms previous literature results (Gathercole and Ross, 1996; Langdon and Poli, 1997). No marked difference exists between GP-GOMEA and GP-Trad$^\ell$ along different configurations. By counting the number of times one EA is found to be significantly better than the other along *all* 240 comparisons, GP-GOMEA beats GP-Trad$^\ell$ by a small margin: 87 significantly lower error distributions vs. 65 (88 draws).

For the traditional use of a single population ($n^{\text{pop}} = 1000$), GP-Trad$^\ell$ is slightly better than GP-GOMEA for $\ell = 15$ (Table 7), slightly worse for $\ell = 31$ (Table 8), and similar for $\ell = 63$ (Table 9), on both validation and test errors. The performance of the two (and also of the other EAs) improves when using the IMS. Although not explicitly shown in the tables, using the IMS is typically significantly better than not using it. When using a single fixed population size and a single run, only a single best-found solution is found. Depending on the configuration of that run, in particular the size of the population, that final solution may be underfitted or overfitted. When using a scheme such as the IMS, multiple solutions are marked best in the different interleaved runs. These solutions can subsequently be compared more in terms of generalization merits, that is, by observing the associated performance on the validation set. The best performing solution can then ultimately be returned. Essentially, this thus provides a means to mitigate to some extent the problem of underfitting or overfitting. It should be noted, however, that the extent to which the setup of the IMS, particularly in terms of growing population sizes, contributes to this is not immediately clear. This could be studied by comparing with a scheme in which multiple runs are also performed, but all with a single population size. The final results of these runs can then also first be tested against the validation set. Likely, the use of a scheme like the IMS has an advantage because multiple population sizes will be tried. Therefore, likely a larger variety of results will be produced to test against the validation set, but a closer examination of this impact is left for future work.

The comparisons between GP-Trad$^\ell$ and GP-GOMEA tend to shift in favor of the latter when using the IMS, particularly for larger values of g. For $g = 4$, outcomes are still overall mixed along different ℓ limits. For $g = 8$, GP-GOMEA is preferable, with moderately more significant wins for $\ell = 15$, several more wins for $\ell = 31$, and slightly more wins for $\ell = 63$.

To investigate further the comparison between GP-GOMEA and GP-Trad$^\ell$, we consider the effect of g of the IMS for $\ell = 31$ (similar results are found for the other size limits). Figure 7 shows the median maximum population size reached by the IMS for different values of g in GP-GOMEA and GP-Trad$^\ell$. As can be expected, the bigger g, the less runs and the smaller populations at play. GP-Trad$^\ell$ reaches much bigger population sizes than GP-GOMEA when $g = 4$ (on average 3 times bigger). This is because GP-Trad$^\ell$ executes generations much faster than GP-GOMEA: it does not learn a linkage model, and performs n^{pop} evaluations per generation. GP-GOMEA performs $(2\ell - 2)n^{\text{pop}}$ variation steps (size of LT excluding its root times the population size) and up to $(2\ell - 2)n^{\text{pop}}$ evaluations per generation (only meaningful variation steps are evaluated).

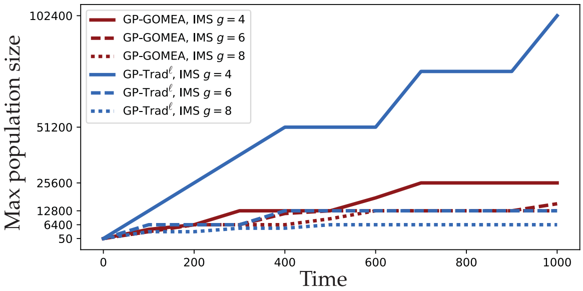


Figure 7: Maximum population size reached (vertical axis) in time (seconds, horizontal axis) with the IMS for GP-GOMEA ($h = 4$ limit) and GP-Trad$^\ell$ ($\ell = 31$ limit), for $g \in \{4, 6, 8\}$. The median among problems and repetitions is shown.

GP-Trad$^\ell$ performs well for small values of g due to huge populations being instantiated with trees of various shape, that is, expensive random search. Note that this behavior may be problematic when limited memory is available, especially if caching mechanisms are desirable to reduce the number of expensive evaluations (e.g., caching the output of each node as in Pawlak et al., 2015 and Virgolin et al., 2017). On the other hand, GP-GOMEA works fairly well with much smaller populations, as long as they are big enough to enable effective linkage learning (the fixed $n^{\text{pop}} = 1000$ is smaller than the population sizes reached with the IMS). Despite the disadvantage of adhering to a specific tree shape, GP-GOMEA is typically preferable than GP-Trad$^\ell$ for larger values of g. Furthermore, Figure 7 shows that GP-GOMEA, population scaling behaves sensibly with regard to g; that is, it does not grow abruptly when g becomes small, nor shrink excessively when g becomes larger. This latter aspect is because in GP-GOMEA, populations ultimately converge to a same solution, and are terminated, allowing for bigger runs to start. In GP-Trad$^\ell$ this is unlikely to happen, because of the use of mutation and stochastic (tournament) selection, stalling the IMS. For the larger $g = 8$, GP-GOMEA reaches on average 1.6 times bigger populations than GP-Trad$^\ell$.

GP-RDO, although allowed to evolve trees of different shape like GP-Trad$^\ell$, performs poorly on all problems, with all settings. It performs significantly worse than GP-GOMEA almost everywhere (it is also worse than GP-Trad$^\ell$). It is known that GP-RDO normally finds big solutions, and it is also reasonable to expect that it needs big solutions to work well, for example, to build a large set of diverse subtrees for the internal library queried by RDO (Virgolin et al., 2019). The strict size limitation basically breaks GP-RDO. However, we remark that this EA was never designed to work under these circumstances. In fact, when solution size is not strictly limited, GP-RDO achieves excellent performance (Pawlak et al., 2015).

DTR is compared with GP-GOMEA using the IMS with $g = 8$. Although GP-GOMEA is not optimized (e.g., by tuning the function set), it performs on par with tuned DTR for $\ell = 15$, and better for $\ell = 31, 63$, on both validation and test sets. Where one algorithm outperforms the other, the magnitude of difference in errors are relatively large compared to the ones between EAs. This is because GP and DTR synthesize models of completely different nature (decision trees only use if-then-else statements).

9 Discussion and Conclusion

We built upon previous work on model-based GP, in particular on GP-GOMEA, to find accurate solutions when a strict limitation on their size is imposed, in the domain of SR.

Tower:

$$4668.49 - 3.56((662.77 + x_{21})x_{12} \div_{\mathrm{AQ}} x_{16} - x_1 - x_{15} + x_5 + 4x_{12} - x_{23}(x_6 \div_{\mathrm{AQ}} x_1 + 1))$$

Yacht:

$$0.73 + 33004.40\left(((x_6^2 \div_{\mathrm{AQ}} (x_5 x_2)) \div_{\mathrm{AQ}} (x_3 x_2 \div_{\mathrm{AQ}} (x_2 \div_{\mathrm{AQ}} x_1)))(x_6 + 0.30)x_6^5 x_5\right)$$

Figure 8: Examples of best solution found by GP-GOMEA ($\ell = 31$, IMS $g = 8$).

We focused on small solutions, in particular much smaller solutions than typically reported in literature, to prevent solutions becoming too large to be (easily) interpretable, a key reason to justify the use of GP in many practical applications.

A first limitation of this work is that to truly *achieve* interpretability may well require different measures. Interpretation is mostly subjective, and many other factors besides solution size are important, including the intuitiveness of the subfunctions composing the solution, potential decompositions into understandable repeating sub-modules, the number of features considered, and the meaning of these features (Lipton, 2018; Doshi-Velez and Kim, 2017). Nonetheless, much current research on GP for SR is far from delivering any interpretable results precisely because the size of solutions is far too large (see, e.g., the work of Martins et al., 2018).

We considered solution sizes up to $\ell = 63$ (corresponding to $h = 5$ for GP-GOMEA with subfunctions of arity $\leq$2). In our opinion, the limit of $\ell = 31$ ($h = 4$) is particularly interesting, as interpreting some solutions at this level can already be non-trivial at times. For example, we show the (manually simplified) best test solution found by GP-GOMEA (IMS $g = 8$) for Tower and Yacht, that is, the biggest and smallest dataset respectively, in Figure 8. The solution for Tower is arguably easier to understand than the one for Yacht. We found solutions with $\ell = 63$ ($h = 5$) to be overly long to attempt interpreting, and solutions with $\ell = 15$ ($h = 3$) to be mostly readable and understandable. We report other example solutions at: http://bit.ly/2IrUFyQ.

We believe future work should address the aforementioned limitation: effort should be put towards reaching some form of interpretability notions, that go beyond solution size or other custom metrics (e.g., Vladislavleva et al., 2009). User studies involving the end users of the model (e.g., medical doctors for a diagnosis model) could guide the design of notions of interpretability. If an objective that represents interpretability can be defined, the design of multiobjective (model-based) GP algorithms may lead to very interesting results.

Another limitation of this work lies in the fact that we did not study how linkage learning behaves in GP for SR in depth. In fact, it would be interesting to assess when linkage learning is beneficial, and when it is superfluous or harmful. To this end, a regime of experiments where linkage-related outcomes are predefined, such as emergence of specific patterns, needs to be designed. Simple problems where the true function to regress is known may need to be considered. Studies of this kind could provide more insights on how to improve linkage learning in GP for SR (and other learning tasks), and are an interesting direction for future work.

Another crucial point to base future research upon is enabling linkage learning and linkage-based mixing in GP with trees of arbitrary shape. In fact, GP-GOMEA was not found to be markedly better than GP-Trad$^\ell$, and a large performance gap was found between GP-Trad$^\ell$ and GP-Trad^h. This is indicative that there is added value to perform evolution directly on non-templated trees, which, from this perspective, may be considered a limitation of GP-GOMEA. Going beyond the use of a fixed tree template, while still enabling linkage identification and exploitation, is a challenging open problem that could bring very rewarding results. On the other hand, we believe it is interesting to see that when GP-GOMEA and GP-Trad are set to work on the same search space, that is, when GP-Trad^h is used, then GP-GOMEA performs markedly better.

In summary and conclusion, we have identified limits and presented ways to improve a key component of a state-of-the-art model-based EA, that is, *GP-GOMEA*, to competently deal with realistic SR datasets, when small solutions are desired. This key component is linkage learning. We showed that solely and directly relying on mutual information to identify linkage may be undesirable, because the genotype is not uniformly distributed in GP populations, and we provided an approximate biasing method to tackle this problem. We furthermore explored how to incorporate ERCs into linkage learning, and found that online binning of constants is an efficient and effective strategy. Lastly, we introduced a new form of the IMS, to relieve practitioners from setting a population size, and from finding a good generalizing solution. Ultimately, our contributions proved successful in improving the performance of GP-GOMEA, leading to the best overall performance against competing EAs, as well as tuned decision trees. We believe our findings set an important first step for the design of better model-based GP algorithms capable of learning interpretable solutions in real-world data.

Acknowledgments

The authors thank the Foundation Kinderen Kankervrij for financial support (project no. 187), and SURFsara for granting access to the Lisa Compute Cluster.

References

Asuncion, A., and Newman, D. (2007). UCI machine learning repository.

Bosman, P. A. N., and De Jong, E. D. (2004). Learning probabilistic tree grammars for genetic programming. In *International Conference on Parallel Problem Solving from Nature*, pp. 192–201.

Bouter, A., Alderliesten, T., Witteveen, C., and Bosman, P. A. N. (2017). Exploiting linkage information in real-valued optimization with the real-valued gene-pool optimal mixing evolutionary algorithm. In *Genetic and Evolutionary Computation Conference (GECCO)*, pp. 705–712.

Breiman, L. (2001). Random forests. *Machine Learning*, 45(1):5–32.

Breiman, L., Friedman, J., Olshen, R. A., and Stone, C. J. (1984). *Classification and regression trees*. Belmont, CA: Wadsworth and Brooks.

Chen, Q., Xue, B., and Zhang, M. (2015). Generalisation and domain adaptation in gp with gradient descent for symbolic regression. In *IEEE Congress on Evolutionary Computation*, pp. 1137–1144.

Chen, Q., Xue, B., and Zhang, M. (2018). Improving generalization of genetic programming for symbolic regression with angle-driven geometric semantic operators. *IEEE Transactions on Evolutionary Computation*, 23(3):488–502.

Chen, T., and Guestrin, C. (2016). XGBoost: A scalable tree boosting system. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pp. 785–794.

Chen, Y., Yu, T.-L., Sastry, K., and Goldberg, D. E. (2007). A survey of linkage learning techniques in genetic and evolutionary algorithms. *IlliGAL Report*, 2007014.

De Melo, V. V. (2014). Kaizen programming. In *Genetic and Evolutionary Computation Conference (GECCO)*, pp. 895–902.

Demšar, J. (2006). Statistical comparisons of classifiers over multiple data sets. *Journal of Machine Learning Research*, 7(Jan):1–30.

Doshi-Velez, F., and Kim, B. (2017). Towards a rigorous science of interpretable machine learning. Retrieved from arXiv preprint arXiv:1702.08608.

Ebner, M., Shackleton, M., and Shipman, R. (2001). How neutral networks influence evolvability. *Complexity*, 7(2):19–33.

Gathercole, C., and Ross, P. (1996). An adverse interaction between crossover and restricted tree depth in genetic programming. In *Genetic and Evolutionary Computation Conference (GECCO)*, pp. 291–296.

Goldman, B. W., and Punch, W. F. (2014). Parameter-less population pyramid. In *Genetic and Evolutionary Computation Conference (GECCO)*, pp. 785–792.

Gronau, I., and Moran, S. (2007). Optimal implementations of upgma and other common clustering algorithms. *Information Processing Letters*, 104(6):205–210.

Guidotti, R., Monreale, A., Ruggieri, S., Turini, F., Giannotti, F., and Pedreschi, D. (2018). A survey of methods for explaining black box models. *ACM Computing Surveys (CSUR)*, 51(5):93.

Harik, G., Cantú-Paz, E., Goldberg, D. E., and Miller, B. L. (1999). The gambler's ruin problem, genetic algorithms, and the sizing of populations. *Evolutionary Computation*, 7(3):231–253.

Harik, G. R., and Lobo, F. G. (1999). A parameter-less genetic algorithm. In *Genetic and Evolutionary Computation Conference (GECCO)*, pp. 258–265.

Hasegawa, Y., and Iba, H. (2009). Latent variable model for estimation of distribution algorithm based on a probabilistic context-free grammar. *IEEE Transactions on Evolutionary Computation*, 13(4):858–878.

Hauschild, M., and Pelikan, M. (2011). An introduction and survey of estimation of distribution algorithms. *Swarm and Evolutionary Computation*, 1(3):111–128.

Hemberg, E., Veeramachaneni, K., McDermott, J., Berzan, C., and O'Reilly, U.-M. (2012). An investigation of local patterns for estimation of distribution genetic programming. In *Genetic and Evolutionary Computation Conference (GECCO)*, pp. 767–774.

Hsu, S.-H., and Yu, T.-L. (2015). Optimization by pairwise linkage detection, incremental linkage set, and restricted/back mixing: DSMGA-II. In *Genetic and Evolutionary Computation Conference (GECCO) 2015*, pages 519–526. ACM.

Icke, I., and Bongard, J. C. (2013). Improving genetic programming based symbolic regression using deterministic machine learning. In *IEEE Congress on Evolutionary Computation*, pp. 1763–1770.

Keijzer, M. (2003). Improving symbolic regression with interval arithmetic and linear scaling. In *European Conference on Genetic Programming*, pp. 70–82.

Kim, K., Shan, Y., Nguyen, X. H., and McKay, R. I. (2014). Probabilistic model building in genetic programming: A critical review. *Genetic Programming and Evolvable Machines*, 15(2):115–167.

Koza, J. R. (1992). *Genetic programming: On the programming of computers by means of natural selection*. Cambridge, MA: MIT Press.

Krawiec, K. (2015). *Behavioral program synthesis with genetic programming*. New York: Springer.

Langdon, W. B., and Poli, R. (1997). An analysis of the max problem in genetic programming. *Genetic Programming*, 1(997):222–230.

Li, X., Mabu, S., Zhou, H., Shimada, K., and Hirasawa, K. (2010). Genetic network programming with estimation of distribution algorithms for class association rule mining in traffic prediction. In *IEEE Congress on Evolutionary Computation*, pp. 1–8.

Lin, Y.-J., and Yu, T.-L. (2018). Investigation of the exponential population scheme for genetic algorithms. In *Genetic and Evolutionary Computation Conference (GECCO)*, pp. 975–982.

Lipton, Z. C. (2018). The mythos of model interpretability. *Queue*, 16(3):30:31–30:57.

Luke, S., and Panait, L. (2001). A survey and comparison of tree generation algorithms. In *Genetic and Evolutionary Computation Conference (GECCO)*, pp. 81–88.

Luong, N. H., La Poutré, H., and Bosman, P. A. N. (2014). Multi-objective gene-pool optimal mixing evolutionary algorithms. In *Genetic and Evolutionary Computation Conference (GECCO)*, pp. 357–364.

Martins, J. F. B. S., Oliveira, L. O. V. B., Miranda, L. F., Casadei, F., and Pappa, G. L. (2018). Solving the exponential growth of symbolic regression trees in geometric semantic genetic programming. In *Genetic and Evolutionary Computation Conference (GECCO)*, pp. 1151–1158.

Medvet, E., Bartoli, A., De Lorenzo, A., and Tarlao, F. (2018). GOMGE: Gene-pool optimal mixing on grammatical evolution. In *International Conference on Parallel Problem Solving from Nature*, pp. 223–235.

Medvet, E., Virgolin, M., Castelli, M., Bosman, P. A. N., Gonçalves, I., and Tušar, T. (2018). Unveiling evolutionary algorithm representation with DU maps. *Genetic Programming and Evolvable Machines*, 19(3):351–389.

Moraglio, A., Krawiec, K., and Johnson, C. G. (2012). Geometric semantic genetic programming. In *International Conference on Parallel Problem Solving from Nature*, pp. 21–31.

Ni, J., Drieberg, R. H., and Rockett, P. I. (2013). The use of an analytic quotient operator in genetic programming. *IEEE Transactions on Evolutionary Computation*, 17(1):146–152.

Orzechowski, P., La Cava, W., and Moore, J. H. (2018). Where are we now?: A large benchmark study of recent symbolic regression methods. In *Genetic and Evolutionary Computation Conference (GECCO)*, pp. 1183–1190.

Pawlak, T. P., and Krawiec, K. (2018). Competent geometric semantic genetic programming for symbolic regression and Boolean function synthesis. *Evolutionary Computation*, 26(2):177–212.

Pawlak, T. P., Wieloch, B., and Krawiec, K. (2015). Semantic backpropagation for designing search operators in genetic programming. *Transactions on Evolutionary Computation*, 19(3):326–340.

Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., Dubourg, V., et al. (2011). Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12(Oct):2825–2830.

Poli, R., Langdon, W. B., McPhee, N. F., and Koza, J. R. (2008). *A field guide to genetic programming*. Lulu. com.

Ratle, A., and Sebag, M. (2001). Avoiding the bloat with stochastic grammar-based genetic programming. In *International Conference on Artificial Evolution*, pp. 255–266.

Sadowski, K. L., Bosman, P. A. N., and Thierens, D. (2013). On the usefulness of linkage processing for solving MAX-SAT. In *Genetic and Evolutionary Computation Conference (GECCO)*, pp. 853–860.

Salustowicz, R., and Schmidhuber, J. (1997). Probabilistic incremental program evolution. *Evolutionary Computation*, 5(2):123–141.

Sastry, K., and Goldberg, D. E. (2003). Probabilistic model building and competent genetic programming. In *Genetic Programming Theory and Practice*, pp. 205–220.

Shan, Y., McKay, R. I., Baxter, R., Abbass, H., Essam, D., and Nguyen, H. (2004). Grammar model-based program evolution. In *IEEE Congress on Evolutionary Computation*, pp. 478–485. Vol. 1.

Sotto, L. F. D. P., and de Melo, V. V. (2017). A probabilistic linear genetic programming with stochastic context-free grammar for solving symbolic regression problems. In *Genetic and Evolutionary Computation Conference (GECCO)*, pp. 1017–1024.

Tanev, I. (2007). Genetic programming incorporating biased mutation for evolution and adaptation of snakebot. *Genetic Programming and Evolvable Machines*, 8(1):39–59.

Thierens, D., and Bosman, P. A. N. (2011). Optimal mixing evolutionary algorithms. In *Genetic and Evolutionary Computation Conference (GECCO)*, pp. 617–624.

Thierens, D., and Bosman, P. A. N. (2013). Hierarchical problem solving with the linkage tree genetic algorithm. In *Genetic and Evolutionary Computation Conference (GECCO)*, pp. 877–884.

Virgolin, M., Alderliesten, T., Bel, A., Witteveen, C., and Bosman, P. A. N. (2018). Symbolic regression and feature construction with GP-GOMEA applied to radiotherapy dose reconstruction of childhood cancer survivors. In *Genetic and Evolutionary Computation Conference (GECCO)*, pp. 1395–1402.

Virgolin, M., Alderliesten, T., and Bosman, P. A. N. (2019). Linear scaling with and within semantic backpropagation-based genetic programming for symbolic regression. In *Genetic and Evolutionary Computation Conference (GECCO)*, pp. 1084–1092.

Virgolin, M., Alderliesten, T., Witteveen, C., and Bosman, P. A. N. (2017). Scalable genetic programming by gene-pool optimal mixing and input-space entropy-based building-block learning. In *Genetic and Evolutionary Computation Conference (GECCO)*, pp. 1041–1048.

Vladislavleva, E. J., Smits, G. F., and Den Hertog, D. (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):333–349.

Wong, P.-K., Lo, L.-Y., Wong, M.-L., and Leung, K.-S. (2014). Grammar-based genetic programming with Bayesian network. In *IEEE Congress on Evolutionary Computation*, pp. 739–746.

Yanai, K., and Iba, H. (2003). Estimation of distribution programming based on Bayesian network. In *IEEE Congress on Evolutionary Computation*, pp. 1618–1625. Vol. 3.

Žegklitz, J., and Pošík, P. (2017). Symbolic regression algorithms with built-in linear regression. Retrieved from arXiv preprint arXiv:1701.03641.

Zhong, J., Feng, L., Cai, W., and Ong, Y.-S. (2018). Multifactorial genetic programming for symbolic regression problems. *IEEE Transactions on Systems, Man, and Cybernetics: Systems*, (99):1–14.