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@amsterdamumc.nl
Department of Radiation Oncology, Amsterdam UMC, University of Amsterdam, Amsterdam, 1105 AZ, 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.

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, i.e., 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 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, 2016; 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, i.e. 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, e.g., 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 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 et al., 2018a), and, the focus of this article, GP (Virgolin et al., 2017, 2018). GOMEA embodies within each generation a model-learning phase, where *linkage*, i.e. the inter-dependency 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 instance 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 w.r.t. 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 scheme 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 along 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 limits of linkage learning, and describes how we improve linkage learning, and how we deal with ERCs. 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, 2017). In our comparison, the GP algorithms are run with a strict limitation on solution size. 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, i.e., inter-dependencies 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 is constrained to some template to be able to estimate probabilities of what functions and terminals appear in what positions (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 (≤ 5) variables have been considered (e.g., by (Ratle and Sebag, 2001; 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 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 et al., 2018a) also employed GOMEA, to attempt to learn and exploit linkage when dealing with different types of pre-defined grammars. Only one synthetic function is considered for symbolic regression, among other four benchmark problems.

There is a need of 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 standard 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 P, given the desired population size p. The generational loop is then started and continues until a termination criterion is met, e.g., 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 Sec. 3.2). Second, each solution P_i is used to generate an offspring solution 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 Sec 3.3).

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

Algorithm 1 Outline of GOMEA

```
1 procedure RUNGOMEA($p$)
2     $P$ ←initializePopulation($p$)
3     while terminationCriteriaNotMet() do
4         $F$ ←learnFOS($P$)
5         $O$ ← $\emptyset$
6         for $i \in \{1, \dots, p\}$ do
7             $O_i$ ←GOM($P_i$, $P$, $F$)
8         $P$ ← $O$
```

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 full r-ary trees of height h, with r being the maximum number of inputs accepted by the functions (arity) provided in the function set

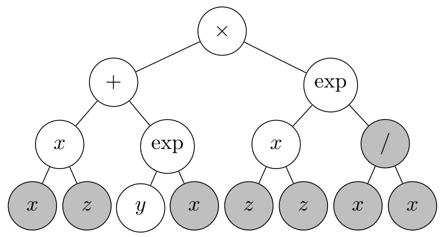


Figure 1: Example of tree for GP-GOMEA with $h = 3$ and $r = 2$. While 15 nodes are present, the nodes that influence the output are only 7: the gray nodes are introns.

(e.g., for $\{+, -, \times\}$, $r = 2$), and h chosen by the user. This is achieved by appending r child nodes to any node that is not at maximum depth, even 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*, i.e., they are not executed to compute the output of the tree. It follows that, while trees are *syntactically* full, they are not necessarily *semantically* so. All trees of GP-GOMEA have a number of nodes, equal to

$$\ell = \sum_{i=0}^{h} r^i. \tag{1}$$

Figure 1 shows an example of a tree for 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:

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

Each F_i contains indices representing 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 by-far most adopted one is the *Linkage Tree* (LT) (Thierens and Bosman, 2013; Virgolin et al., 2017). The LT captures linkage in hierarchical levels. An LT is typically learned every generation, from the population. To assess whether linkage learning plays a key role, i.e. 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 sets F_i in a binary tree structure. The LT uses mutual information as a proxy for linkage strength, as follows. Initially, the leaves of the LT are initialized to singletons $F_i = \{i\}, \forall i \in \{1, \ldots, \ell\}$. To build the next levels of the LT, mutual information is measured for any pair of genotype locations i, j in the population, from the distribution of *symbols*. While in a binary genetic algorithm symbols are either '0' or '1', symbols in GP correspond to the types of function and terminal node, e.g., '$+$','$-$','x_1','x_2'. Mutual information between a pair of locations can be computed after measuring entropy for single locations $\mathrm{H}(i)$, and pairs of locations, $\mathrm{H}(i, j)$:

$$\mathrm{MI}(i, j) = \mathrm{H}(i) + \mathrm{H}(j) - \mathrm{H}(i, j), \tag{2}$$

where

$$\mathrm{H}(i) = -\sum \mathrm{P}_i \log \mathrm{P}_i, \quad \mathrm{H}(i,j) = -\sum \mathrm{P}_{ij} \log \mathrm{P}_{ij}, \tag{3}$$

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 in the population genotype.

Once mutual information is computed for all location pairs, the next levels of the LT are built by the clustering algorithm Unweighted Pair Group Method with Arithmetic Mean (UPGMA) (Gronau and Moran, 2007). UPGMA only coarsely approximates the real mutual information between higher-order tuples of locations, however it is much faster than calculating mutual information exactly. The LT can be built in $O(\ell^2 p)$, and has proven to find higher-order dependencies with sufficient accuracy to efficiently and effectively solve many real-world problems (Thierens and Bosman, 2013). We remove the root of the LT which corresponds to a set that contains all node locations, and thus provides no linkage information. Thus, the LT contains $2\ell - 2$ sets.

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 may 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 variates a given solution 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, i.e., variation attempts that result in worse fitness are undone.

The pseudo-code presented in Algorithm 2 describes GOM in detail. To begin, a backup B_i of the parent solution P_i is made, including its fitness, and similarly an offspring solution $O_i = P_i$ 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 D is then picked from the population, and O_i is changed by replacing the nodes specified by F_j with the homologous ones from D. It is then assessed whether at least one non-intron node of the tree has been changed by variation (indicated by $\neq^\star$ in line 9). When that is not the case, O_i will have the same behavior as B_i, thus the fitness is necessarily identical. Otherwise, the new fitness f_{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 $f_{O_i} = f_{B_i}$, 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.

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

Algorithm 2 Pseudocode of GOM

```
procedure GOM($P_i, P, F$)
    $B_i \leftarrow P_i$
    $f_{B_i} \leftarrow f_{P_i}$
    $O_i \leftarrow P_i$
    $F \leftarrow$randomShuffle($F$)
    for $F_j \in F$ do
        $D \leftarrow$pickRandomDonor($P$)
        $O_i \leftarrow$overrideNodes($O_i, D, F_j$)
        if $O_i \neq^{\star} B_i$ then
            $f_{O_i} \leftarrow$computeFitness($O_i$)
            if $f_{O_i} \leq f_{B_i}$ then                #Assumption: minimization of $f$
                $B_i \leftarrow O_i$
                $f_{B_i} \leftarrow f_{O_i}$
            else
                $O_i \leftarrow B_i$
                $f_{O_i} \leftarrow f_{B_i}$
        else
            $B_i \leftarrow O_i$
```

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 because it has been shown to lead to much better generalization in GP. This is because the AQ is continuous in 0 for the second operand: $x_1 \div_{\text{AQ}} x_2 := x_1/\sqrt{1+x_2^2}$.

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 Sec. 9). Larger values for h would therefore play against the aim of this study. When benchmarking GP-GOMEA in Sec. 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[1] (Asuncion and Newman, 2007) and other sources[2]. 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:

$$\text{MSE}(y, \tilde{y}) = \frac{1}{n}\sum_{i}^{n}\left(y_i - (a + b\tilde{y}_i)\right)^2,$$

where y_i is the value of the variable to regress for the ith example, and $\tilde{y}_i$ the respective solution prediction. The constants a and b can be calculated in $O(n)$ with:

$$a = \mu(y) - b\mu(\tilde{y}),$$
$$b = \sum_{i}^{n}\frac{(y_i - \mu(y))(\tilde{y}_i - \mu(\tilde{y}))}{(\tilde{y}_i - \mu(\tilde{y}))^2},$$

with μ computing the mean. We present our results in terms of variance-Normalized MSE (NMSE), i.e. $\frac{\text{MSE}(y,\tilde{y})}{var(y)}$, so that results from different datasets are on a similar scale.

To assess statistical significance when comparing two algorithms (or configurations) on a certain dataset, we use the Wilcoxon signed-rank test (Demšar, 2006), paired

[1]https://archive.ics.uci.edu/ml/index.php
[2]https://goo.gl/tn6Zxv

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

Table 2: Regression datasets used in this work

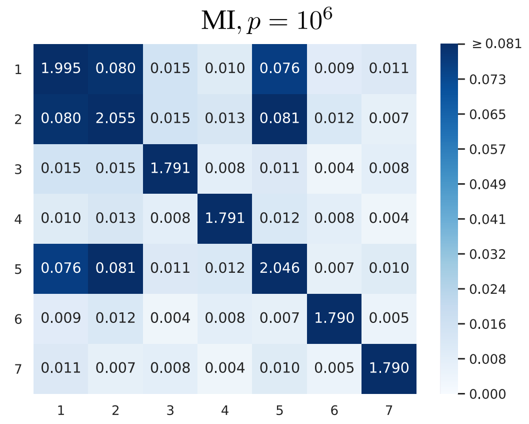
Name	Abbreviation	# Features	# Examples
Airfoil	Air	5	1503
Boston housing	Bos	13	506
Concrete compres. str.	Con	8	1030
Dow chemical	Dow	57	1066
Energy cooling	EnC	8	768
Energy heating	EnH	8	768
Tower	Tow	25	4999
Wine red	WiR	11	1599
Wine white	WiW	11	4898
Yacht hydrodynamics	Yac	6	308

by random seed. The seed determines the particular split of the examples between training, validation, and test sets, and also the sampling of the initial population. 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 symbol. 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, i.e. by computing the mutual information between pairs of locations (i, j) in the genotype (Eq. 2) (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, it depends on the cardinality of the function and terminal sets, and on the population initialization method, (e.g., *Full*, *Grow*, *Half-and-Half*, *Ramped Half-and-Half* (Luke and Panait, 2001)). This lack of uniformity leads to an imbalance in mutual information, suggesting the presence of linkage. However, it is reasonable to expect no linkage to be present in an initialized population, as evolution did not take place yet.

Figure 2 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 (Eq. 1). We index node locations with pre-order tree traversal, i.e., 1 is the root, 2 its first child, 5 its



	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 2: 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 suggests the existence of linkage even though no evolution has taken place yet.

second child, $3, 4$ are (leaves) children of 1, and $6, 7$ are (leaves) children of 5. Nodes at positions 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 captures this aspect (i.e., larger mutual information values are present between non-leaf nodes), thus using it directly as linkage metric is undesirable.

We hypothesize that, if we correct linkage learning so that no patterns emerge at initialization, the truly salient patterns will have a bigger chance of emerging during evolution, and better results will be achieved. We now discuss how such a correction can be performed.

5.1 Not uniform sampling and mutual information

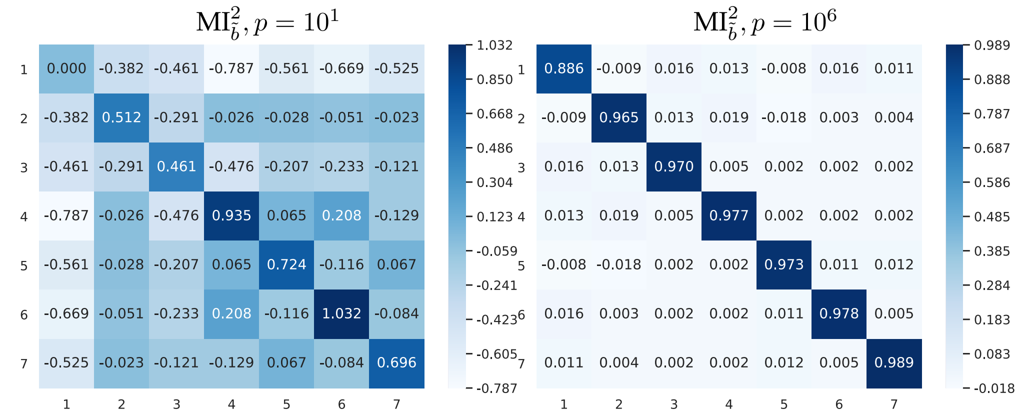
A possibility to overcome the aforementioned problem is to modify the mutual information to take into account the non-uniformity of the distribution of node types in the initial population. Since mutual information depends on entropy (Eq. 2 and 3), we can attempt to normalize the entropy.

We begin by considering the fact that, in principle, symbols at a specific location i can be sampled by a location-specific set Ω_i. For example, in GP, tree leaves are necessarily terminal nodes. For brevity, we now focus on univariate entropy $\mathrm{H}(i)$, but similar considerations hold for the joint entropy $\mathrm{H}(i, j)$. We rewrite Eq. 3 as:

$$\mathrm{H}(i) = -\sum_{\omega \in \Omega_i} \mathrm{P}_i(\omega) \log \mathrm{P}_i(\omega).$$

A normalized entropy is then given by using a proper base b_i for the logarithm. Under normal circumstances, it suffices to set $b_i = |\Omega_i|$.

For an initial population of GP, several issues are present. For a location i, Ω_i is typically not fixed, rather it depends on the probability that the function or the terminal set is used to sample nodes at that location (e.g., due to using the Ramped Half-and-Half initialization method). Even if b_i can be exactly determined, locations $i, j, i \neq j$ may have different a priori probability distributions $\mathrm{P}_i \neq \mathrm{P}_j$. Thus, we propose a simple approximation method.



	1	2	3	4	5	6	7
1	0.000	-0.382	-0.461	-0.787	-0.561	-0.669	-0.525
2	-0.382	0.512	-0.291	-0.026	-0.028	-0.051	-0.023
3	-0.461	-0.291	0.461	-0.476	-0.207	-0.233	-0.121
4	-0.787	-0.026	-0.476	0.935	0.065	0.208	-0.129
5	-0.561	-0.028	-0.207	0.065	0.724	-0.116	0.067
6	-0.669	-0.051	-0.233	0.208	-0.116	1.032	-0.084
7	-0.525	-0.023	-0.121	-0.129	0.067	-0.084	0.696

	1	2	3	4	5	6	7
1	0.886	-0.009	0.016	0.013	-0.008	0.016	0.011
2	-0.009	0.965	0.013	0.019	-0.018	0.003	0.004
3	0.016	0.013	0.970	0.005	0.002	0.002	0.002
4	0.013	0.019	0.005	0.977	0.002	0.002	0.002
5	-0.008	-0.018	0.002	0.002	0.973	0.011	0.012
6	0.016	0.003	0.002	0.002	0.011	0.978	0.005
7	0.011	0.004	0.002	0.002	0.012	0.005	0.989

Figure 3: Mutual information matrices at the second generation using our normalization method, with population size of 10 (middle), and of 10^6 (right) for a particular run of GP-GOMEA. The rightmost matrix is closest to the identity I.

5.2 Sample-based normalization of mutual information

Let us consider the ideal case where a proper logarithm base b is known for each location (we now drop the subscript i from b_i for the sake of readability), and symbols in each location are distributed uniformly. We get $\mathrm{H}_b(i) = 1$, $\mathrm{H}_b(i,j) = 2$ for $i \neq j$, and $\mathrm{H}_b(i,j) = 1$ for $i = j$. The mutual information must then be (Eq. 2):

$$\mathrm{MI}_b(i,j) = 1 + 1 - 2 = 0 \text{ for } i \neq j, \text{ else } 1,$$

i.e. the identity matrix I.

Now, consider that H_b can be written as:

$$\mathrm{H}_b(i) = -\sum \mathrm{P}_i \log_b \mathrm{P}_i = -\sum \mathrm{P}_i \frac{\log \mathrm{P}_i}{\log b} = \\ = -\frac{1}{\log b} \sum \mathrm{P}_i \log \mathrm{P}_i = \frac{1}{\log b} \mathrm{H}(i).$$

This means we can determine a normalization coefficient $\frac{1}{\log b} =: \beta_i$ to transform $\mathrm{H}(i)$ into $\mathrm{H}_b(i)$. We now make the assumption that the initial population sample is sufficiently representative of the true distribution of nodes. We want the initial population to have $\mathrm{H}_b^{\text{init}}(i) = 1$, and $\mathrm{H}_b^{\text{init}}(i,j) = 2$. For this, it suffices to set:

$$\beta_i = \frac{1}{\mathrm{H}^{\text{init}}(i)}, \beta_{i,j} = \frac{2}{\mathrm{H}^{\text{init}}(i,j)}.$$

Thus, we simply compute the β coefficients for the initial population, and then use them during the linkage learning phase (i.e., to build the LT) each generation. The mutual information computed at generation g will then be:

$$\mathrm{MI}^g_{\tilde{b}}(i,j) = \beta_i \mathrm{H}^g(i) + \beta_j \mathrm{H}^g(j) - \beta_{i,j} \mathrm{H}^g(i,j). \quad (4)$$

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

Table 3: Results of linkage learning with correction of mutual information: 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	8.4 ▲▲	9.7 ▼▼	9.1 ▼▲	9.2 ▲▲	10.7 ▼▼	10.3 ▼▲
EnH	6.2 ▲▲	6.4 ▼▼	6.4 ▼▲	6.5 ▲▲	7.1 ▼▼	6.7 ▼▲
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	0.3 ▲▲	0.4 ▼	0.4 ▼	0.6 ▲▲	0.6 ▼▼	0.6 ▼▲

5.3 Approximation error of MI$_{\tilde{b}}$

As a preliminary step, we observe the error associated with using the approximately normalized mutual information MI$_{\tilde{b}}$. To have a crude estimate of the approximation error, we report a typical mutual information matrix computed at the second generation of a GP-GOMEA run on the dataset Yac (at the first one $\text{MI}^1_{\tilde{b}} = I$ by construction). We repeat this for two population sizes, $p = 10$ and $p = 1,000,000$. We expect that, the bigger p, the lower the approximation error.

We use the parameters of Table 1, a terminal set of size 6 (the features of Yac, no ERC) and $h = 2$, i.e. $\ell = 7$ nodes per tree. Figure 3 shows the mutual information matrix between location pairs, for the two population sizes. It can be seen that the values can erroneously be lower than 0 or bigger than 1. However, while this is particularly marked for $p = 10$, with minimum of -0.787 and maximum of 1.032, it becomes less evident for $p = 10^6$, with minimum of -0.018 and maximum of 0.989. The fact that $\text{MI}^2_{\tilde{b}} \approx I$ for $p = 10^6$ is because, with such a large population size, a lot of diversity is still present in the second generation.

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

We now test the use of MI$_{\tilde{b}}$ over the standard MI for GP-GOMEA with the LT. We use the notation LT–MI$_{\tilde{b}}$ and LT–MI to refer to the two configurations. We further consider the RT to see if using mutual information to drive variation is any better than random.

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 is approximately equivalent to 1200 generations of traditional GP, since each solution is evaluated up to $2\ell - 2$ times (size of the LT minus non-meaningful changes, see Sec. 3.2 and Sec. 3.3).

5.5 Results: LT–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–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.

The result of this experiment is that the use of the new MI$_{\tilde{b}}$ to build the LT simply

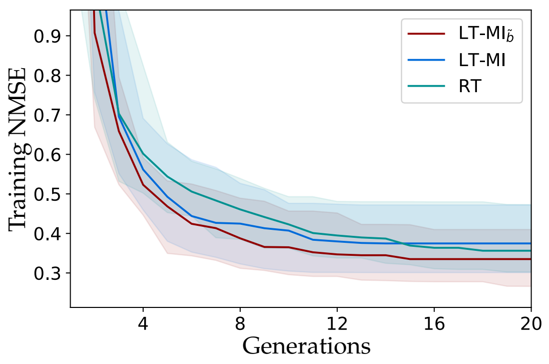


Figure 4: Median fitness of the best solution of 30 runs on Yac, for LT–MI$_{\tilde{b}}$, LT–MI, and RT (10^{th} and 90^{th} percentiles in shaded area).

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 4 shows the evolution of the training NMSE for the dataset Yac. It can be seen that the LT–MI$_{\tilde{b}}$ allows to 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$_{\tilde{b}}$.

5.6 Experiment: impact of pattern propagation

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 in the genotype exist that can be exploited. Another aspect that can be considered w.r.t. such hypothesis is how final solutions look: if linkage learning identifies specific patterns, it can be reasonably 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 solution, compared to the use of the RT. We use the same parameter setting as described in Sec. 5.4, 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 $p = 2000$ as in the previous experiment, we also consider a doubled $p = 4000$.

5.7 Results: impact of pattern propagation

Table 4 reports the number of solutions that have at least one duplicate, i.e. their genotype is semantically equivalent (e.g., $x_1 + x_2 = x_2 + x_1$), along different runs for 5 random splits of Yac. It can be seen that the LT finds more duplicate solutions than the RT, by a margin of around 30%.

Figure 5 shows the distribution of solutions found for the second dataset split with $p = 4000$, i.e. 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 patterns more than random variation should such patterns exist, enhancing the likeli-

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

	$p = 2000$		$p = 4000$	
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

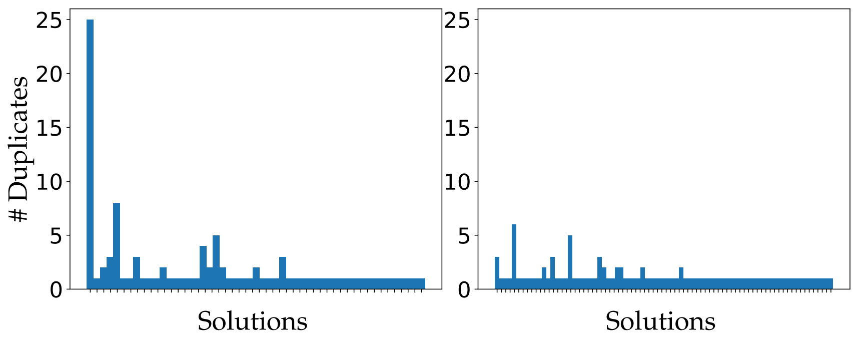


Figure 5: Distribution of solutions found for 100 runs by using the LT (left) and the RT (right) with $p = 4000$ on the second dataset split of Yac.

hood of discovering particular solutions.

6 Ephemeral random constants & 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 special terminal which is set to a constant only when 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 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:

1. all-const: Ignore the shortcomings, and consider all different constants as different symbols during linkage learning.

2. no-const: Skip all constants during linkage learning, i.e. 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.

3. 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, the error of this approximation lowers over time, as less useful constants get discarded during evolution.

6.1 Experiment: linkage learning with ERCs

We use the same parameter setup of the experiment in Sec. 5.4, 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), i.e., 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 ~ 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.

6.2 Results: linkage learning with ERCs

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 to not using them (compare with Table 3).

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 to the linkage-based one, where salient patterns are quickly propagated, and less variation attempts result in changes of the genotype that require a fitness evaluation (see Sec. 3.3). The slower convergence caused by the RT can also be seen in Figure 4 (for the previous experiment), and was also observed in other work, in terms of diversity preservation (Medvet et al., 2018b).

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 Interlaved 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), each performing one generation only after g generations of the run with one level less of

Table 5: Results of linkage learning with ERCs: 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	5.5 ▲▲▲	5.7 ▼ ▲	5.8 ▼ ▲	6.4 ▼▼▼	6.2 ▲ ▲	6.3 ▼▼▲	6.0 ▲▲	6.8 ▼▼▼
EnH	3.0 ▲▼▲	3.1 ▼▼▲	2.8 ▲▲▲	4.1 ▼▼▼	3.3 ▲▼▲	3.3 ▼▼▲	3.1 ▲▲▲	4.7 ▼▼▼
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	0.3 ▲▲▲	0.3 ▼▼▲	0.3 ▼▲▲	0.4 ▼▼▼	0.6 ▲	0.6 ▼▲	0.5 ▲▲	0.6 ▼▼▼

Table 6: Results of linkage learning with ERCs in terms of time (details as in Table 5).

	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 ▲▲▲

computational resources ($g \geq 2$). The main motivation for using the IMS is to make an EA much more robust to parameter selection, 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.

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

7.1 Adapting the IMS to learning tasks

The first implementation of the IMS for GP-GOMEA was designed to deal with GP benchmark problems of pure optimization. Such implementation works as follows:

1. Scaling parameters — The scaling parameters are population size and tree height.

The base population size scales with $p_i = 2^{i-1} p_{\text{base}}$, with i the index of run R_i. The tree height scales with $h_i = h_{\text{base}} + \lfloor \frac{i}{2} \rfloor$.

2. Termination — A run R_i is terminated if the mean population fitness is worse than the one of a subsequent run R_j, $j > i$, or if it converges to all identical solutions. All smaller runs $R_k, k < i$ are terminated as well.

Scaling the tree height was implemented to ensure that solutions will eventually be large enough to accommodate all necessary symbols to find the optimum for benchmark problems (Virgolin et al., 2017). However, in SR and in general learning tasks, no optimum is known beforehand, and it is rather desired to find a solution that generalizes well to unseen examples. Thus, we remove the scaling of h from the IMS (only p scales). This is also motivated by the fact that h bounds the maximum solution size, which then plays a role in interpretability. For these reasons, we do let the user decide what h should be, and, if interpretability is desired, $h \leq 4$ is recommended (see Sec. 9).

We then change the run termination criteria. In SR, it can happen that the error of a few solutions becomes extremely big, compromising the mean population fitness. This can trigger the termination criteria even if solutions exist that are competitive with the ones of other runs. For this reason, we now consider the fitness of the best solution rather than the mean population fitness. Also different from the first version of the IMS, when terminating a run, we do not automatically terminate all previous runs. Indeed, some smaller runs 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 re-compute the fitness of each solution in the archive using a set of examples different from the training set, i.e. the validation set, and return the new best performing, i.e., 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, 2017).

We consider RDO because, as mentioned in the introduction, much work on semantic-aware operators has been proposed recently, often for symbolic regression. 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 set of semantic-aware operators that raised much interest lately, i.e., the geometric operators employed by Geometric Semantic Genetic Programming (GSGP) (Moraglio et al., 2012). Indeed 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; Chen and Guestrin, 2016)) are typically markedly more accurate than single DTRs, but are considered not intepretable.

8.1 Experimental setup

For the EAs, we use a fixed time limit of $1,000$ seconds[3], so that the overhead of learning the LT in GP-GOMEA is not a limiting factor. We consider maximum solution sizes $\ell = 15, 31, 63$ (tree nodes), i.e. corresponding to $h = 3, 4, 5$ respectively, for full r-ary trees. The EAs are run with a typical fixed population size $p = 1000$ and also with the IMS, considering three values for the number of subgenerations 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 re-started. 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 dephts 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 here only its best configuration, i.e., a version where the number of tree nodes is limited like for GP-Trad$^\ell$.

We use the Python Scikit-learn implemention of DTR (Pedregosa et al., 2011), with 5-fold cross-validation grid-search over the training set to tune the following hyper-parameters: *splitter* $\in$ {*'best'*,*'random'*}; *max_features* $\in \{\frac{1}{2}, \frac{3}{4}, 1\}$; *max_depth* $\in$ {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 rela-

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

tionships that can be modeled. Another general result is that errors between validation and test set are generally close. Likely, the validation data is a sufficiently accurate surrogate 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 size limitation, i.e., 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, i.e., blind variation, in the previous series of experiments (Sec. 5.4, 6.1). Subtree crossover and subtree mutation are blind as well, and can only swap connected substructures, 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 fixed population size ($p = 1000$), 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 the IMS is adopted, the comparisons between GP-Trad$^\ell$ and GP-GOMEA tend to shift in favor of the latter, in particular for larger values of g. For $g = 4$, outcomes are still overall mixed along different ℓ limits. For $g = 8$, GP-GOMEA is overall 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 6 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$^\ell$ tends to reach much bigger population sizes than GP-GOMEA when $g = 4$ (on average 3 times bigger). This is because GP-Trad$^\ell$ executes subgenerations much faster than GP-GOMEA: it does not learn a linkage model, and performs p evaluations per subgeneration. GP-GOMEA performs $(2\ell - 2)p$ variation steps (size of LT times population size) and up to $(2\ell - 2)p$ evaluations per subgeneration (only meaningful variation steps are evaluated).

GP-Trad$^\ell$ performs well for small values of g due to huge populations being instantiated with trees of various shape, i.e., 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; 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 $p = 1000$ is smaller than the population sizes reached with the IMS). Despite the disadvantage of adhering to a

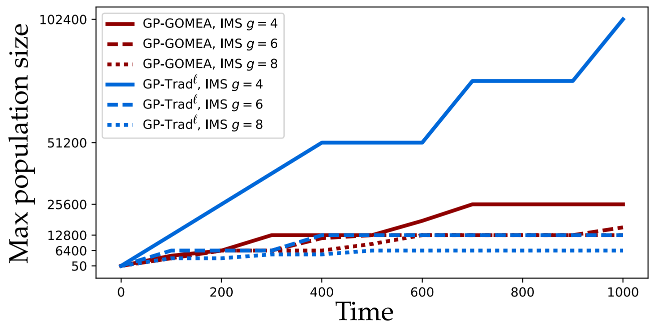


Figure 6: Maximum population size reached (vertical axis) in time (seconds, horizontal axis) with the IMS for GP-GOMEA ($h = 4$ limit) and GP-Trad$^\ell$ ($\ell = 31$ limit), for $g \in \{4, 6, 8\}$. The median among problems and repetitions is shown.

specific tree shape, GP-GOMEA is typically preferable than GP-Trad$^\ell$ for larger values of g. Furthermore, Figure 6 shows that GP-GOMEA population scaling behaves sensibly w.r.t. g, i.e., 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, e.g., 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 & 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. 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, the number of features considered, and the meaning of

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 $p = 1000$ and IMS with $g \in \{4, 6, 8\}$, and DTR. Significance is assessed within each population scheme w.r.t. GP-GOMEA. The last row reports the number of times the EA performs significantly better (B) and worse (W) than GP-GOMEA.

Validation $\ell = 15$

	$p = 1000$				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	8.7	10.6▼	7.3	11.0▼	5.9	10.2▼	6.5 ▼	10.7▼	6.0	10.3▼	6.2	10.5▼	5.9	10.2▼	6.1 ▼	10.8▼	4.2 ▲
EnH	4.9	7.4 ▼	3.8 ▲	7.7 ▼	3.3	7.2 ▼	3.7 ▼	7.3 ▼	3.3	7.3 ▼	3.8 ▼	7.4 ▼	3.2	7.2 ▼	3.7 ▼	7.5 ▼	0.4 ▲
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	1.3	1.2	0.7 ▲	1.0 ▲	0.9	1.0 ▼	0.6 ▲	0.7 ▲	0.9	1.0 ▼	0.6 ▲	0.7 ▲	0.9	1.0 ▼	0.6 ▲	0.8 ▲	0.9 ▲
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 $\ell = 15$

	$p = 1000$				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	9.7	11.2▼	7.9 ▲	11.8▼	6.4	10.6▼	6.8 ▼	11.5▼	6.4	10.5▼	6.2	10.9▼	6.0	10.5▼	6.3 ▼	11.7▼	4.5 ▲
EnH	5.0	7.2 ▼	4.0 ▲	7.8 ▼	3.4	7.6 ▼	3.9 ▼	7.6 ▼	3.3	7.6 ▼	3.9 ▼	7.6 ▼	3.5	7.6 ▼	3.9 ▼	7.6 ▼	0.3 ▲
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	1.2	1.2	0.8 ▲	0.9 ▲	1.2	1.2 ▼	0.7 ▲	0.8 ▲	1.2	1.2	0.7 ▲	0.8 ▲	1.2	1.2 ▼	0.7 ▲	0.9 ▲	0.9 ▲
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

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

Validation $\ell = 31$

	$p = 1000$				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	5.1	5.6 ▼	5.0 ▲	7.6 ▼	4.6	5.5 ▼	4.8 ▼	8.0 ▼	4.4	6.0 ▼	4.6 ▼	8.5 ▼	4.4	5.7 ▼	4.7 ▼	7.8 ▼	4.2 ▲
EnH	2.3	2.5 ▼	1.7	6.2 ▼	2.0	3.1 ▼	1.7 ▲	5.0 ▼	2.0	2.8 ▼	1.6 ▲	5.9 ▼	1.9	3.1 ▼	1.6 ▲	6.1 ▼	0.4 ▲
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	0.5	0.6 ▼	0.4 ▲	0.6 ▼	0.4	0.5 ▼	0.4	0.6 ▼	0.4	0.6 ▼	0.4	0.5 ▼	0.4	0.5 ▼	0.4	0.5 ▼	0.9 ▼
B/W	—	0/7	2/4	0/10	—	0/9	2/2	0/10	—	0/10	1/2	0/10	—	0/10	1/4	0/10	3/7

Test $\ell = 31$

	$p = 1000$				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	5.3	5.9 ▼	4.8 ▲	7.0 ▼	4.4	5.8 ▼	4.8	7.7 ▼	4.4	6.0 ▼	4.7	8.7 ▼	4.6	5.6 ▼	4.8 ▼	7.9 ▼	4.5 ▲
EnH	2.3	2.5 ▼	1.8	6.0 ▼	2.0	3.2 ▼	1.6 ▲	5.1 ▼	2.1	3.1 ▼	1.8	5.8 ▼	2.0	2.9 ▼	1.6 ▲	6.5 ▼	0.3 ▲
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	0.5	0.7 ▼	0.5 ▲	0.7 ▼	0.5	0.6 ▼	0.5	0.7 ▼	0.5	0.6 ▼	0.5 ▲	0.6 ▼	0.5	0.6 ▼	0.5	0.7 ▼	0.9 ▼
B/W	—	0/9	2/4	0/10	—	0/8	3/1	0/10	—	0/9	1/2	0/10	—	0/9	1/5	0/10	3/7

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

Validation $\ell = 63$

	$p = 1000$				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	4.7	5.2 ▼	4.3 ▲	5.5 ▼	3.7	4.4 ▼	4.1 ▼	6.9 ▼	3.9	4.5 ▼	4.0 ▼	7.1 ▼	3.8	4.9 ▼	4.0 ▼	6.8 ▼	4.2 ▼
EnH	1.7	1.5 ▼	1.1 ▲	2.6 ▼	0.7	1.5 ▼	0.8 ▼	4.0 ▼	0.9	1.8 ▼	1.0 ▼	3.8 ▼	0.9	1.8 ▼	0.8	3.7 ▼	0.4 ▲
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	0.5	0.4	0.4 ▲	0.5	0.3	0.4 ▼	0.3	0.4 ▼	0.3	0.4 ▼	0.3	0.4 ▼	0.3	0.4 ▼	0.3 ▲	0.5 ▼	0.9 ▼
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 $\ell = 63$

	$p = 1000$				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	4.5	4.7 ▼	4.2 ▲	5.6 ▼	3.8	4.4 ▼	4.0 ▼	6.9 ▼	3.9	4.4 ▼	4.0 ▼	7.4 ▼	3.9	4.9 ▼	4.2 ▼	7.4 ▼	4.5 ▼
EnH	1.6	1.6 ▼	1.1 ▲	2.7 ▼	0.8	1.5 ▼	0.9 ▼	3.7 ▼	0.9	1.7 ▼	0.9	4.1 ▼	0.9	1.9 ▼	0.9 ▼	3.7 ▼	0.3 ▲
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	0.4	0.5	0.4 ▲	0.5	0.4	0.5 ▼	0.4 ▲	0.4 ▼	0.4	0.4 ▼	0.4	0.5 ▼	0.5	0.5	0.4 ▲	0.5 ▼	0.9 ▼
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

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 ≤ 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, i.e. the biggest and smallest dataset respectively, in Figure 7. 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 multi-objective (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

Tower:

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

Yacht:

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

Figure 7: Examples of best solution found by GP-GOMEA ($\ell = 31$, IMS $g = 8$).

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. Our comparisons between the EAs showed that traditional crossover and mutation perform poorly compared to linkage-based mixing when the same limitation on the shape of solutions (imposed by the maximum height) is used (GP-GOMEA clearly outperforms GP-Trad^h). However, allowing more freedom in solution shape is beneficial (GP-GOMEA is not markedly better than GP-Trad$^\ell$). Going beyond the use of fixed tree templates in GP-GOMEA is a challenging open problem that could bring very rewarding results.

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, i.e. *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 introduces bias, because the genotype is not uniformly distributed in GP populations, and we provided an approximate normalization method to tackle this problem. We furthermore explored how to incorporate ERCs into linkage learning, and found that on-line 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.

Acknowledgement

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*, pages 192–201. Springer.

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) 2017*, pages 705–712. ACM.

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

Breiman, L. (2017). *Classification and regression trees*. Routledge.

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 (CEC) 2015*, pages 1137–1144. IEEE.

Chen, Q., Xue, B., and Zhang, M. (2018). Improving generalisation of genetic programming for symbolic regression with angle-driven geometric semantic operators. *IEEE Trans. Evol. Comput.*

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*, pages 785–794. ACM.

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) 2014*, pages 895–902. ACM.

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. *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) 1996*, pages 291–296. MIT Press.

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

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) 1999*, pages 258–265. Morgan Kaufmann Publishers Inc.

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) 2012*, pages 767–774. ACM.

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 (CEC) 2013*, pages 1763–1770. IEEE.

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

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*. MIT Press, Cambridge, MA, USA.

Krawiec, K. (2016). *Behavioral program synthesis with genetic programming*, volume 618. 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 (CEC) 2010*, pages 1–8. IEEE.

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

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) 2001*, pages 81–88. Morgan Kaufmann Publishers Inc.

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) 2014*. ACM.

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) 2018*, pages 1151–1158, New York, NY, USA. ACM.

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

Medvet, E., Virgolin, M., Castelli, M., Bosman, P. A. N., Gonçalves, I., and Tušar, T. (2018b). 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*, pages 21–31. Springer.

Ni, J., Drieberg, R. H., and Rockett, P. I. (2013). The use of an analytic quotient operator in genetic programming. *IEEE Trans. Evol. Comput.*, 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) 2018*, pages 1183–1190, New York, NY, USA. ACM.

Pawlak, T. P., Wieloch, B., and Krawiec, K. (2015). Semantic backpropagation for designing search operators in genetic programming. *IEEE Trans. Evol. Comput.*, 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 (Evolution Artificielle)*, pages 255–266. Springer.

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) 2013*, pages 853–860. ACM.

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*, pages 205–220. Springer.

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 (CEC) 2004*, volume 1, pages 478–485. IEEE.

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) 2017*, pages 1017–1024. ACM.

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) 2011*, pages 617–624. ACM.

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

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) 2018*, pages 1395–1402. ACM.

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) 2019*, New York, NY, USA. ACM.

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) 2017*, pages 1041–1048, New York, NY, USA. ACM.

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 (CEC) 2014*, pages 739–746. IEEE.

Yanai, K. and Iba, H. (2003). Estimation of distribution programming based on bayesian network. In *IEEE Congress on Evolutionary Computation (CEC) 2003*, volume 3, pages 1618–1625. IEEE.

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

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