

SGP-DT: Semantic Genetic Programming Based on Dynamic Targets*

Stefano Ruberto¹, Valerio Terragni², and Jason H. Moore¹

¹ Institute for Biomedical Informatics, University of Pennsylvania, United States
stefano.ruberto@penmedicine.upenn.edu jhmoore@upenn.edu

² Faculty of Informatics, Università della Svizzera italiana USI, Switzerland
valerio.terragni@usi.ch

Abstract. Semantic GP is a promising approach that introduces semantic awareness during genetic evolution. This paper presents a new Semantic GP approach based on Dynamic Target (SGP-DT) that divides the search problem into multiple GP runs. The evolution in each run is guided by a new (dynamic) target based on the residual errors. To obtain the final solution, SGP-DT combines the solutions of each run using linear scaling. SGP-DT presents a new methodology to produce the offspring that does not rely on the classic crossover. The synergy between such a methodology and linear scaling yields to final solutions with low approximation error and computational cost. We evaluate SGP-DT on eight well-known data sets and compare with ϵ -LEXICASE, a state-of-the-art evolutionary technique. SGP-DT achieves small RMSE values, on average 23.19% smaller than the one of ϵ -LEXICASE.

Keywords: Semantic GP · Genetic Programming · Natural Selection · Symbolic Regression · Residuals · Linear Scaling · Crossover · Mutation

1 Introduction

Recently, researchers successfully applied Semantic methods to Genetic Programming (SGP) on different domains, showing promising results [1, 2, 3]. While the classic GP operators (e.g., selection, crossover and mutation) act at the syntactic level, blindly to the semantic (behavior) of the individuals (e.g., programs), the key idea of SGP is to apply semantic evaluations [1]. More specifically, classic GP operators ignore the behavioral characteristic of the offspring, focusing only on improving the fitness of the individuals. Differently, SGP uses a richer feedback during the evolution that incorporates semantic awareness, which has the potential to improve the power of genetic programming [1].

In this paper, we are considering the Symbolic Regression domain, and thus assuming the availability of training cases (defined as m pairs of inputs and desired output). Following the most popular SGP approaches [1], we intend “*semantics*” as the set of output values of a program on the training cases [4]. Such an approach obtains a richer feedback during the evolution relying on the

* This is the authors version of this work. It was later published in: European Conference on Genetic Programming (EuroGP '20)

evaluation of the individuals on the training cases. More formally, the semantics of an individual \mathcal{I} is a vector $sem(\mathcal{I}) = \langle y_1, y_2, \dots, y_m \rangle$ of responses to the m inputs of the training cases. Let $sem(\hat{y}) = \langle \hat{y}_1, \hat{y}_2, \dots, \hat{y}_m \rangle$ denote the semantic vector of the target (as defined in the training set), where $\hat{y}_1, \hat{y}_2, \dots, \hat{y}_m$ are the desired outputs. SGP defines *semantic space* [1] with a metric that characterizes the distance between the semantic vectors of the individuals $sem(\mathcal{I})$ and the target $sem(\hat{y})$. SGP often relies on such a distance to compute the fitness score, inducing a unimodal fitness landscape, which avoids local optima by construction [5].

The effectiveness of SGP depends on the availability of GP operators that can move in the semantic space towards the global optimum. An example of semantic operator is the geometric crossover proposed by Moraglio et al. [5]. It produces an offspring with a semantic vector that lies on the line connecting the parents in the semantic space. Thus, it guarantees that the offspring is no worse than the worst of the parents [5]. However, such crossover operator has the major drawback of producing individuals with an exponentially increasing size (i.e., *exponential bloat*) [5, 1]. To avoid the exponential bloat, researchers proposed variants of this operator that minimize bloating [2] but at the cost of dropping the important guarantee of non-worsening crossover operations.

In this paper, we present a new SGP approach called **SGP-DT** (*Semantic Genetic Programming based on Dynamic Targets*) that minimizes the exponential bloat problem and at the same time gives a bound on the worsening of the offspring. SGP-DT divides the search problem into multiple GP runs. Each run is guided by a different dynamic target, which SGP-DT updates at each run based on the residual errors of the previous run. Then, SGP-DT combines the results of each run into a “*optimized*” final solution.

In a nutshell, SGP-DT works as follows. SGP-DT runs the GP algorithm (see Algorithm 1) a fixed number of times (N_{ext}) depending on the available budget. We call these runs *external* iterations. As opposed to the *internal* iterations (i.e., generations) that the GP algorithm performs to evolve the individuals. Each GP run performs a fixed number of internal iterations and returns a model (i.e., the best solution) that we call *partial model*. The next external iteration runs the GP algorithm with a modified training set, where SGP-DT replaces the m desired outputs $\hat{y}_i = \langle \hat{y}_1, \hat{y}_2, \dots, \hat{y}_m \rangle$ with the residual errors of the partial model returned by the previous iteration. That is, the difference between $sem(\mathcal{I}_i)$ and $sem(\hat{y}_{i-1})$, where \mathcal{I}_i is the partial model at the i^{th} iteration. Thus, at each external iteration, the fitness function evaluates differently the individuals (because the fitness functions predicates on different training sets). As such, each partial model focuses on a different portion of the problem, the one that most influences the fitness value. As a result, our approach leads to dynamic targets that change at each external iteration incorporating the semantic information. SGP-DT obtains the final solution after N_{ext} iterations with a linear combination in the form $\sum_{i=0}^{N_{\text{ext}}} a_i + b_i \cdot \mathcal{I}_i$, where a_i and b_i are computed with the well-known *linear scaling* [6]. There is a key advantage of using linear scaling. Keijzers showed that linear scaling gives a bound on the error of those generated individuals that

are linear scaled [6]. Therefore, SGP-DT entails a bound on the worsening of the offspring at each internal and external iteration.

To reduce the exponential bloat problem, SGP-DT performs the internal GP iterations relying on classic mutation operators only. It does not rely on any form of crossover, neither geometric nor classic, and thus avoiding their fundamental limitations. Geometric crossover leads to exponential bloat and classic crossover decreases the chance to obtain a fitness improvement because it exchanges random functionalities at random points [7]. Despite the absence of crossovers, SGP-DT implicitly recombines different functionalities, similarly to a geometric crossover [5]. This is because, each partial model focuses on a different characteristic of the problem that the fitness function recognized as important (at that iteration). This makes the search more efficient because the evolution focuses on a single characteristic at a time leaving unaltered other (already optimized) characteristics.

We evaluated our approach on eight well-known regression problems. We compared SGP-DT with two baselines: LASSO a least square regression technique by Efron et al. [8]; and ϵ -LEXICASE a state-of-the-art SGP approach by La Cava et al. [9]. The results show that our approach obtains a median RMSE on 50 runs that is, on average, 51.47% and 23.19% smaller than the one of LASSO and ϵ -LEXICASE, respectively. Moreover, SGP-DT requires as much as $9.26 \times$ fewer tree computations than ϵ -LEXICASE ($4.81 \times$ on average).

The remainder of this paper is organized as follows. Section 2 describes our approach. Section 3 discusses the related work. Section 4 reports our experimental evaluation and discusses the results. Section 5 concludes the paper.

2 Methodology

Algorithm 1 overviews the SGP-DT approach. Given the values of the independent (\bar{x}) and dependent (\hat{y}) variables of the training cases, and the number of external (N_{ext}) and internal (N_{int}) iterations, it returns the final solution (*finalModel*).

SGP-DT considers tree-like individuals with the usual non-terminal symbols: $+$, $-$, \cdot , $/$ (the protected division), *ERC* (between -1 and 1). In addition, SGP-DT considers the functions *Min* and *Max* that returns the minimum and maximum between two numbers, respectively. The rationale of adding the two latter symbols is to inject *discontinuity* to make the linear combinations more adaptable. Although also the protected division adds discontinuity in the form of asymptotes, such discontinuity often promotes overfitting [6, 10]. With *Min* and *Max* functions, we introduce valid discontinuities alternatives that do not suffer from the limitation of the protected division.

Algorithm 1 holds out a portion of the training cases for validation (lines 1-3). SGP-DT will use such validation sets to construct the final solution (line 22). Lines 4-5 initialize the current target with \hat{y} and the lists of the best models with the empty list. Line 6 starts the external loop, which re-assigns \mathcal{P} to a fresh randomly generated population with the *ramped-half-and-half* approach (function GET-RANDOM-INITIAL-POPULATION of Algorithm 1). Starting every

Algorithm 1: SGP-DT

```

input  :  $\bar{x}$  : values of the independent variables of the training cases
           $\hat{y}$  : values of the dependent variables of the training cases
           $N_{ext}$  : number of external iterations
           $N_{int}$  : number of internal iterations
output : finalModel : final regression model

1  $\langle \bar{x}_{val}, \hat{y}_{val} \rangle \leftarrow \text{SPLIT}(\bar{x}, \hat{y})$ 
2  $\bar{x} \leftarrow \{\bar{x} \setminus \bar{x}_{val}\}$ 
3  $\hat{y} \leftarrow \{\hat{y} \setminus \hat{y}_{val}\}$ 
4  $\text{target} \leftarrow \hat{y}$ 
5  $\text{models} \leftarrow \emptyset$ 
6 for ext-iter 1 ...  $N_{ext}$  do
7    $\mathcal{P} \leftarrow \text{GET-RANDOM-INITIAL-POPULATION}()$ 
8   for int-iter 1 ...  $N_{int}$  do
9     for each  $\mathcal{I} \in \mathcal{P}$  do
10       $\mathcal{I}_{ls} \leftarrow \text{COMPUTE-LS}(\mathcal{I}, \bar{x}, \text{target})$  // linear scaling
11       $\text{fitness}(\mathcal{I}) \leftarrow \sigma^2(\text{sem}(\mathcal{I}_{ls}(\bar{x})) - \text{target})$  //  $\sigma^2$  variance
12       $\mathcal{I}_{ls}^* \leftarrow \text{GET-BEST-INDIVIDUAL}(\mathcal{P})$ 
13       $\text{error} \leftarrow \text{target} - \text{sem}(\mathcal{I}_{ls}^*(\bar{x}))$ 
14      add  $\mathcal{I}_{ls}^*$  to  $\text{models}$ 
15       $\mathcal{P}' \leftarrow \emptyset$ 
16      add ELITE( $\mathcal{P}$ ) to  $\mathcal{P}'$ 
17      while  $\mathcal{P}'$  is not full do
18         $\mathcal{I} \leftarrow \text{TOURNAMENT-SELECTION}(\mathcal{P})$ 
19        add MUTATE( $\mathcal{I}$ ) to  $\mathcal{P}'$ 
20       $\mathcal{P} \leftarrow \mathcal{P}'$ 
21    $\text{target} \leftarrow \text{error}$  // update the target
22  $\text{bestModels} \leftarrow \text{VALIDATE-AND-SELECT}(\bar{x}_{val}, \hat{y}_{val}, \text{models})$  // best MSE models on val
23  $\text{finalModel} \leftarrow \sum_{\text{model} \in \text{bestModels}} \text{model}$ 
24 return finalModel

```

external iteration with a new population alleviates the overfitting problem. Indeed, the syntactic structures of already evolved individuals can be too complex to adapt to a new fitness landscape or to generalize on unseen data. To further reduce overfitting and the cost of fitness evaluation, SGP-DT generates the initial population with individuals with low complexity (i.e., a few nodes).

At line 8, SGP-DT starts the N_{int} internal iterations, which resembles the classic GP but with the addition of linear scaling and the absence of crossover. Before line 11 computes the fitness of each individual \mathcal{I} in \mathcal{P} , line 10 performs the linear scaling of \mathcal{I} [6]. Linear scaling has the advantage of transforming the semantic of individuals so that their potential fit with the current target is immediately given: we do not need to wait for GP to produce a partial model that reaches the same result [6]. And thus, linear scaling reduces the number of both external and internal iterations. Fewer iterations means populations with simpler structural complexity and less computational cost. Reducing the complexity of the solutions may reduce overfitting [11].

Linear scaling has another important property: it gives an upper bound on the error [6]. Recall that SGP-DT considers errors on dynamic targets, which change

at each iteration (at the first iteration the dynamic target is \hat{y}). To exploit such a situation, we propose a fitness function based on this upper bound. Following Keijzer [6], we compute the linear scaling of an individual \mathcal{I} as follows:

$$\mathcal{I}_{ls} = a + b \cdot \mathcal{I} \quad (1)$$

$$\text{where } a = \bar{\hat{y}} - b \cdot \bar{y} \quad \text{and} \quad b = \frac{\sum_{i=1}^n [(\hat{y}_i - \bar{\hat{y}}) \cdot (y_i - \bar{y})]}{\sum_{i=1}^n [(y_i - \bar{y})^2]} \quad (2)$$

We define the following fitness function of an individual \mathcal{I} :

$$\text{fitness}(\mathcal{I}) = \sigma^2(\text{sem}(\mathcal{I}_{ls}(\bar{x})) - \hat{y}) \quad (3)$$

The rationale of this function is that the Mean Square Error (MSE) of \mathcal{I}_{ls} has the variance (σ^2) of the current target as an upper bound [12]:

$$\text{MSE} = \frac{\sum_{i=0}^m (y_i - \hat{y}_i)^2}{m} \leq \sigma^2(\hat{y}) \quad (4)$$

where m is the number of training cases (y).

At each new external iteration the residual error becomes the new target (line 21).

$$\text{target} = \hat{y} - \text{sem}(\mathcal{I}_{ls}^*(\bar{x})) \quad (5)$$

where $\text{sem}(\mathcal{I}_{ls}^*(\bar{x}))$ is the evaluation of the best individual at the current iteration, which we call *partial model*.

The inequality 4 does not guarantee that the external iterations converge to a lower MSE because we do not know if $\sigma^2(\text{error}) \leq \sigma^2(\hat{y})$, where $\text{error} = \text{target} - \text{sem}(\mathcal{I}_{ls}^*(\bar{x}))$. Thus, by optimizing the variance of the error shown in equation 3, we act directly on the minimization of the upper bound, so that the next external iteration can benefit from a lower bound.

At lines 17-19, Algorithm 1 runs a classic GP algorithm without crossovers, using only mutations. We use a tree-based mutation operator because SGP-DT uses trees as syntactic structures for the individuals. The operator randomly generates a subtree from a randomly chosen node. To increase the synergy with linear scaling, we set two constraints during mutation. First, the node selection is biased towards the leaves of the tree, so that the mutated tree does not diverge too much from the original semantic (*locality principle*). Producing a mutation that is close to the original semantic of the tree preserves the validity of the selection performed after the linear scaling. And thus, we only allow minor changes to improve the fitness. Second, for the same reason, the mutation is biased towards replacing the selected node with a sub-tree of limited depth. Note that, we decided not to limit the maximum size (number of nodes in the tree) or depth of an individual. By doing so, GP can grow and choose the right solution complexity for the problem at hand. These two constraints help us to mitigate the overfitting and bloat problem without preventing the SGP-DT to effectively search for competitive individuals. As linear scaling helps GP to find useful individuals (thanks to the upper bound). Moreover, additional external iterations will further refine other aspects of the problem not yet addressed.

We decided to exclude the classic crossover operator in the internal iterations, as several researchers argued about the effectiveness of crossover in relation to the problem of modularity of GP [13]. There is a consensus that an effective GP algorithm needs a crossover that preserves the semantics of the parts swapped among individuals respecting the boundaries of a useful functionality within the individual’s structure [7, 2, 14]. According to McPhee et al. [4] and Ruberto et al. [11] most classic crossover operators do not obtain a meaningful variation (or any variation at all) in the program semantics, when dealing with Boolean and real value symbolic regression domains. The main issue is that classic crossover operators do not preserve a *common context* [4] among the building blocks of the individuals exchanged during crossover, which is important to increase the chance of obtaining a semantically meaningful offspring [14]. The idea of determining a common context has been introduced by Poli and Langdon with the one-point crossover operator [7]. But how to identify a meaningful common context among trees structures is still an open problem.

Instead, SGP-DT exchanges functionalities among individuals by relying on the linear combination of the *partial models* (i.e., the fittest individuals at each external iteration, line 12 Algorithm 1) and on a specific mechanism for selecting and mutating the individuals during the GP runs. In light of this, we exclude the crossover operators in the presence of these semantic recombination alternatives. To have an effective exchange of functionalities among individuals we need to: (i) preserve building blocks semantics (ii) preserve the context of building blocks (iii) make the exchange of functionalities directed towards producing new and interesting semantics. SGP-DT achieves these objectives by (i) mapping each building block to a single partial model (this would avoid arbitrary fragmentations of the blocks); (ii) preserving the context of the building blocks because in our scenario the partial models obtained at previous iterations represent the context; and (iii) using mutation only, which promotes diversity in the population. Despite the absence of crossover, SGP-DT exchanges building blocks because each partial model is a building block. Differently from the classical crossover that exchanges random fragments, SGP-DT obtains the final model by summing the linear scaled partial models. This approach makes the exchange of functionalities more effective, as each partial model (building block) characterizes a specif functionality.

The for-loop at line 6 terminates when SGP-DT concludes all external iterations. We decide not to introduce a different stopping criterion based on the stagnation of fitness improvement. This is because it is difficult to predict if the fitness will not escape stagnation in future iterations. After all the external iterations, the function VALIDATE-AND-SELECT at line 22 of Algorithm 1 returns the partial models that will be combined into the final solution. Such models are selected as follows. The validation takes in input the ordered sequence of best individuals (*models*) collected after each internal iteration (line 14 Algorithm 1) and the validation sets (\bar{x}_{val} and \hat{y}_{val}) obtained at line 1. Note that, SGP-DT saves the computed linear scaling parameters (a and b equations (2)) at line 10 and do not recompute them during the validation and test phases. Internally, the validation scans the sequence *models* and progressively computes the MSE

evaluating the individuals on the validation set to find the point in the sequence where MSE is the smallest. SGP-DT finds the smallest MSE using the rolling mean of the validation set error at a fixed window size to minimize the short-term fluctuations. The function `VALIDATE-AND-SELECT` returns the sequence (*bestModels*) of the partial models that were produced before the smallest MSE. Such sequence represents the transformation chain of the dynamic targets. In case SGP-DT obtained the model with the smallest MSE during the internal iterations, it appends this individual at the end of *bestModels*. Line 23 of Algorithm 1 computes the final model by summing all the models in *bestModels*.

3 Related Work

This section divides the related work of SGP-DT in three groups. Each group refers to techniques that are relevant to a main characteristic of SGP-DT: (i) having dynamic or semantic objectives, (ii) using linear combinations or geometric operators, (iii) using an iterative approach on residual errors.

Dynamic or semantic objectives The GP techniques proposed by Krawiec et al. [15] and Liskowski et al. [16] present semantic approaches that consider interactions between individuals and the training set. These approaches cluster such interactions to derive new targets for a multi-objective GP.

Otero et al. proposed an approach with dynamic objectives that combines intermediate solutions in a final Boolean tree [17]. This technique progressively eliminates from the training cases the ones perfectly predicted from the current intermediate solution and operates exclusively in a Boolean domain.

Krawiec and O’Reilly [18] proposed a GP approach that explicitly models the semantic behavior of a solution during the computation of training cases.

BPGP by Krawiec and O’Reilly [18] explicitly models the semantic behavior of a solution during the computation of training cases. BPGP proposes an operator that mutates an individual by replacing a randomly selected sub-tree with a random one. According to Krawiec and O’Reilly this “mutation-like” [18] operator is intended as a “form of crossover”. We think that this is similar in principle to our design choice of dropping crossover altogether and instead choosing among mutated alternatives in the population. However, Krawiec and O’Reilly still use the traditional crossover alongside with this new mutation [18].

We differ from all of these techniques because we build our solution progressively crystallizing the intermediate achievements. Most of these approaches use auxiliary objectives during their search and use a single GP run. Conversely, SGP-DT uses a non-predetermined number of objectives in subsequent GP runs. The approach of Otero et al. [17] is the only one that progressively builds the solution but it uses a strategy that works for Boolean trees only.

Linear combinations MRGP [19] uses multiple linear regression to combine the semantics of sub-programs (subtrees) to form the semantic of an individual.

Ruberto et al. proposed ESAGP [20], which derives the target semantics by relying on a specific linear combination between two “optimally aligned” individuals in the error space. Leveraging such geometric alignment property,

Vanneschi et al. proposed NA-GP [21], which performs linear combinations between two aligned chromosomes belonging to the same individual.

Gandomi et al. proposed MGGP [22], where each individual is composed of multiple trees. MGGP produces the final solution with a linear combination of the tree’s semantics, deriving the values of the coefficients from the training data with a classic least squares method. However, the number of trees in the linear combination is fixed and the fitness landscape is not dynamic.

Moraglio et al. proposed the Geometric Semantic GP (GSGP) crossover operator [5], which uses linear combinations to guarantee offspring that is not worse than the worst of the parents. Unfortunately, GSGP suffers from the exponential bloat problem and requires many generations to converge, especially if the target is not in the convex hull spanned by the initial population [5].

Notably, all the approaches described in this second group use a single run to search for the final solution. Differently from SGP-DT, they fix the number of components in advance (the only exception is GSGP but it suffers from the exponential bloat problem [5]). In addition, all of the techniques in the first and second groups have a static target, and thus they continuously evolve a population without re-initialization. This limits the diversity of the genetic alternatives when the population converges at later generations. Conversely, SGP-DT has a dynamic target and it starts with a fresh population at each internal iteration (see Algorithm 1).

Iterative approaches based on residual errors Sequential Symbolic Regression (SSR) [23] uses the crossover operator GSGP [5] to iteratively transform the target using a semantic distance that resembles the classical residual approach. However, no statistical difference (on the errors) from the classical GP approach was found [23]. Differently from SGP-DT, SSR considers residuals that do not optimize the linear combinations with a least square method. Although SSR overcomes the exponential bloat, it weakens the advantage of using residuals.

Medernach et al. presented the WAVE technique [24, 25] that similarly to SGP-DT, executes multiple GP runs using the same definition of residual errors (equation 5) and obtains the final model by summing the intermediate models. WAVE produces a sequence of short and heterogeneous GP runs, obtained by “fuzzing” the settings of system parameters (e.g, population size, number of internal iterations) and by alternating the use of linear scaling. However, SGP-DT drastically differs from WAVE. The heterogeneity nature of WAVE emulates this dynamic evolutionary environment by simulating periods of a rapid change [24, 25]. The effectiveness of such an approach requires specif combinations of system parameters that converges to a fitter solution. Due to the huge space of possible system parameters, finding such combinations often requires a large number of iterations [24, 25]. Conversely, SGP-DT steers the evolution with a novel approach that gradually evolves the building blocks of the final solution without exploring the huge space of possible combinations of system parameters.

All the techniques of this group use residuals differently from SGP-DT. Moreover, they rely on the classic or geometric crossover. Conversely, one of the key novel aspects of SGP-DT is to avoid crossover altogether.

Table 1: Data sets of regression problems.

name	# attributes	# instances	source	name	# attributes	# instances	source
airfoil	5	1,503	UCI [26]	housing	14	506	UCI [26]
concrete	8	1,030		tower	25	3,135	
enc	8	768		yacht	6	309	
enh	8	768		uball5d	5	6,024	

4 Evaluation

Data sets We performed our experiments on eight well-known data sets of regression problems that have been used to evaluate most of the techniques discussed in Section 3 [9, 19, 21, 22, 24, 25]. Table 1 shows the name, number of attributes, and number of instances for each data set. For *uball5d*³ we followed the same configuration used by Cava et al. [28].

4.1 Methods

We compared SGP-DT with two techniques (LASSO [8] and ϵ -LEXICASE [9]) and two variants of SGP-DT (DT-EM and DT-NM).

LASSO Both SGP-DT and LASSO [8] use the least square regression method to linearly combine solution components. More specifically, LASSO incorporates a regularization penalty into least-squares regression using an ℓ_1 norm of the model coefficients and uses a tuning parameter λ to specify the weight of this regularization [8]. We relied on the LASSO implementation by Efron et al. [8], which automatically chooses λ using cross-validation.

ϵ -LEXICASE This evolutionary technique adapts the *lexicase* selection operator for continuous domains [9]. The idea behind ϵ -LEXICASE selection is to promote candidate solutions that perform well on unique subsets of samples in the training set, and thereby maintain and promote diverse building blocks of solutions [9]. Each parent selection begins with a randomized ordering of both the training cases and the solutions in the selection pool (i.e., population). Individuals are iteratively removed from the selection pool if they are not within a small threshold (ϵ) of the best performance among the pool on the current training sample. The selection procedure terminates when all but one individual is left in the pool, or until all individuals have tied performance. In the latter case, a random one is chosen. The recent study of Orzechowski et al. shows that ϵ -LEXICASE [9] outperforms many GP-inspired algorithms [29]. We relied on the publicly available implementation of ϵ -LEXICASE, *ellyn*⁴, which uses stochastic hill climbing to tune the scalar values of each generated individual. It also relies on a 25% validation hold-out from the training data to choose the final model from a bi-dimensional *Pareto archive*, which *ellyn* constantly updates during the evolution. The two dimensions are the number of nodes and the fitness.

³ $f(x) = 10/(5 + \sum_{i=1}^5 (x_i - 3)^2)$

⁴ <https://github.com/EpistasisLab/ellyn>

DT-EM We considered a variant of SGP-DT (called DT-EM) with a modified fitness function as the only difference with SGP-DT:

$$fitness(\mathcal{I}) = MSE = \frac{\sum_{i=0}^m (y_i - \hat{y}_i)^2}{m} \quad (6)$$

While the original fitness of SGP-DT minimizes the upper bound of the MSE in equation 3, this function directly minimizes the MSE in equation 6. This variant helps to evaluate the impact of a direct error minimization with respect to a more qualitative and indirect measure of the error, such as the variance (σ^2).

DT-NM We considered another variant, called DT-NM, that excludes the *Min* and *Max* non-terminal symbols (as the only difference with SGP-DT), and thus evaluating the advantage of different discontinuity types during the evolution.

4.2 Evaluation setup

Following the setup of Orzechowski et al. [29] for ϵ -LEXICASE, we set for all the four GP techniques (SGP-DT, ϵ -LEXICASE, DT-EM, and DT-NM) a population size of 1,000 and a budget of 1,000 generations. We ran 50 trials for every technique on each data set using 25% of the data for testing and 75% for training.

SGP-DT and its two variants share the same configuration: We divided the 1,000 generations in 20 external iterations ($N_{\text{ext}} = 20$), and thus the number of internal iterations (N_{int}) is 50. We used ramped half&half initialization up to a maximum depth of four (function GET-RANDOM-INITIAL-POPULATION at line 7 of Algorithm 1). The probability of mutation is 100% and the maximum depth of the sub-trees generated by the mutation operators is five. The probability of a sub-tree mutation happening at the leaf level is 70%. We set no limits on the number of nodes in the trees and on the depth of the trees. We set the Elitism to keep only the best individual at each internal iteration (function ELITE at line 16 of Algorithm 1). We obtained the validation set by extracting 10% of the training cases (function SPLIT at line 1 of Algorithm 1). The fixed window size for the rolling-mean is 20. We chose this configuration after a preliminary tuning phase and kept uniform for all the eight data sets.

4.3 Results and discussion

Errors' Comparison Following previous work we use the Root Mean Square Error (RMSE) to evaluate the final solution with the test set. The first five columns of Table 2 show for each technique the median RMSE of the 50 trials. The last four columns of Table 2 indicate the percentage decrease of the RMSE medians with respect to the competitor techniques⁵. A positive percentage value means that the RMSE median of SGP-DT is lower (i.e., better), while a negative value means a worst median RMSE. Figure 1 shows the box plots of the RMSE values of the 50 trials⁶. When comparing the RMSE values we performed

⁵ calculated with $((M_T - M_D)/M_T) \cdot 100$, where M_D is the median RMSE of SGP-DT and M_T is the one of the competing technique

⁶ for readability reasons we omitted 4 out-layers for LASSO, 13 for ϵ -LEXICASE, 30 for SGP-DT, 30 for DT-NM and 35 for DT-EM

Table 2: Median RMSE of the 50 trials.

Data set	Root Mean Square Error (RMSE)				Median RMSE % decrease of SGP-DT over:				
	SGP-DT	LASSO	ϵ -LEXICASE	DT-EM	DT-NM	LASSO	ϵ -LEXICASE	DT-EM	DT-NM
airfoil	2.4634	4.8484	3.6505	2.5643	2.9237	49.19 %	32.52 %	3.94 %	15.75 %
concrete	6.5123	10.5383	7.0707	6.4476	6.4132	38.20 %	7.90 %	-1.00 %	-1.55 %
enc	1.4838	3.2498	1.8647	1.4993	1.4584	54.34 %	20.43 %	1.03 %	-1.75 %
enh	0.5560	2.9645	1.2952	0.5714	0.5410	81.25 %	57.07 %	2.70 %	-2.76 %
housing	4.4700	4.9155	4.2785	4.4377	4.5273	9.06 %	-4.48 %	-0.73 %	1.26 %
tower	0.2606	0.2953	0.2975	0.2900	0.2900	11.75 %	12.39 %	10.12 %	10.12 %
uball5d	0.0402	0.1939	0.0618	0.0430	0.0372	79.29 %	35.00 %	6.63 %	-7.87 %
yacht	1.0221	9.0237	1.3577	1.2849	1.1786	88.67 %	24.72 %	20.45 %	13.28 %
Average RMSE % decrease:						51.47 %	23.19 %	5.39 %	3.31 %

a non-parametric pairwise Wilcoxon rank-sum test with Holm correction for multiple-testing, with a confidence level of 95% (p-value <0.05).

SGP-DT achieves a smaller RMSE than LASSO for all the data sets, obtaining always statistical significance. The decrease of the RMSE medians ranges from 9.06% for *housing* to 88.67% for *yacht* (51.47% on average). SGP-DT has smaller RMSE medians than ϵ -LEXICASE for all data sets but *housing* (decrease -4.48%). This is the only comparison of SGP-DT and ϵ -LEXICASE without statistically significance. The decrease of the RMSE medians ranges from -4.48% for *housing* to 57.07% for *ench* (23.19% on average). This is a remarkable result considering that ϵ -LEXICASE outperforms many GP-inspired algorithms [29]. Comparing with the variant DT-EM, SGP-DT achieves the only statistically significant differences with DT-EM on the data sets *uball5d* and *yacht*, with percentage decreases of 6.63% and 20.45%, respectively. For such datasets SGP-DT performs better than DT-EM indicating that our fitness function that minimizes the upper bound achieves a better final solution. SGP-DT has statistically significant differences of the median RMSE with DT-NM only with the data sets *airfoil*, *tower* and *uball5d*. SGP-DT performs better than DT-NM on the *airfoil* and *tower* datasets: 3.94% and 10.12% of percentage decrease, respectively. This means that the *Min* and *Max* non-terminal symbols provide an advantage only in these two datasets. However, Figure 1 indicates that using such non-terminal symbols does not penalize the outcome in any other dataset, except for *uball5d* where the difference is statistically significant (the decrease is -7.87%).

Error comparison with related work Unfortunately, the implementation of WAVE [24, 25] is not publicly available, and thus a direct comparison would be difficult. We extracted the median RMSE from the GECCO 2016 paper [25] for our two common subjects: 4.1 (*concrete*) and 8.7 (*yacht*). SGP-DT achieves a median RMSE percentage decrease of 25.17% (*concrete*) and 75.12% (*yacht*), see Table 2 for the reference values. Note that, the computational cost reported in the GECCO paper has the same order of magnitude with the one of SGP-DT.

From the paper of Vanneschi et al. [21], we extracted the median RMSE on the data set *concrete* of the following GP techniques: 10.44 (NA-GP [21]), 8.1 (NA-GP-50 [21]), 12.50 (GSGP [5]), and 9.43 (GSGP-LS [30]). SGP-DT has a percentage decrease of 37.64%, 19.62%, 47.92% and 30.96%, respectively. These results are only indicative because their evaluation setup differs from ours.

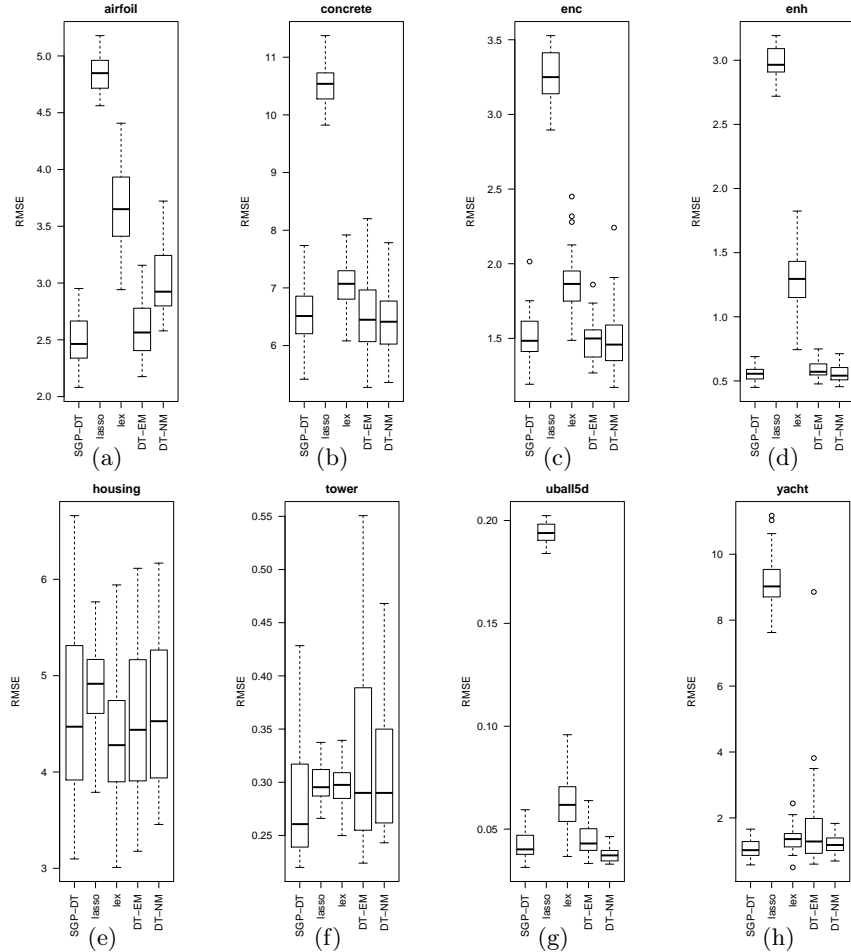


Fig. 1: RMSE of test set for all the techniques and for all the eight data sets.

Computational effort To evaluate the computational effort of the evolutionary techniques we decided not to rely on execution time because it depends on implementation details. Instead, we relied on the total number of evaluated nodes (being not a GP technique this metric is not applicable to LASSO). Both SGP-DT and ϵ -LEXICASE operate on nodes, SGP-DT on tree-like data structures, while ϵ -LEXICASE on stack-based ones. Following Ruberto et al. [11], we count a node operation every time a technique evaluates a node regardless the purpose of the operation (e.g., mutation, fitness computation). We excluded the computational effort of linear scaling because it does not perform operations on nodes. However, it has a linear computational cost of $\mathcal{O}(m \cdot P)$, where m is the size of the training set and P the population size. For comparing the number of evaluated nodes, we used the Wilcoxon rank-sum test with Holm correction for multiple-testing, with a confidence level of 95% (p-value < 0.05). The test show that all the comparisons between each pair of techniques are statically significance, except the comparison with SGP-DT and DT-NM on subject *uball5d*.

Table 3: Median number of evaluated nodes and reduction ratio of SGP-DT.

Data set	Median number of evaluated nodes				Reduction ratio of SGP-DT over		
	SGP-DT	ϵ -LEXICASE	DT-EM	DT-NM	ϵ -LEXICASE	DT-EM	DT-NM
airfoil	1.00E+10	9.28E+10	1.00E+10	9.03E+09	9.26×	1.00×	0.90×
concrete	1.14E+10	6.43E+10	1.14E+10	8.82E+09	5.64×	1.00×	0.77×
enc	1.18E+10	4.99E+10	1.17E+10	9.37E+09	4.25×	0.99×	0.80×
enh	1.18E+10	5.08E+10	1.17E+10	9.27E+09	4.30×	0.99×	0.78×
housing	7.70E+09	3.09E+10	7.63E+09	6.03E+09	4.02×	0.99×	0.78×
tower	7.21E+10	1.94E+11	7.12E+10	4.45E+10	2.69×	0.99×	0.62×
uball5d	9.83E+10	3.94E+11	9.76E+10	7.50E+10	4.01×	0.99×	0.76×
yacht	4.62E+09	2.00E+10	4.58E+09	3.47E+09	4.34×	0.99×	0.75×
Average reduction ratio:					4.81×	0.99×	0.77×

Table 3 reports the median number of nodes (of the 50 runs) that the GP techniques evaluate to produce the final solution. The last three columns of Table 3 report the ratio between the number of node evaluations of SGP-DT with those of ϵ -LEXICASE, DT-EM and DT-NM. A ratio greater (lower) than one means that SGP-DT evaluates a lower (higher) number of nodes. Comparing with ϵ -LEXICASE, SGP-DT reduces the amount of node evaluations by a factor between $4.01\times$ and $9.26\times$, obtaining statistically significant better RMSE values than ϵ -LEXICASE for seven out of eight data sets. This result can be explained by (i) SGP-DT computes only a fraction of the entire solution (partial models) at a time; (ii) the size of the individuals is kept at minimum (see Section 2).

The number of evaluated nodes of SGP-DT and DT-EM are almost identical ($0.99\times$ on average). This indicates that guiding the evolution with the fitness function of SGP-DT and with the one of DT-EM yield to the same computational cost but SGP-DT achieves better median RMSE (5.39% on average). DT-NM always evaluated less nodes than SGP-DT ($0.77\times$ on average).

Size of the final solutions SGP-DT has no limits on the maximum complexity of the individuals, while ϵ -LEXICASE has a limit of 50 nodes because at higher limits the computational effort of ϵ -LEXICASE becomes prohibitively expensive [9]. SGP-DT produces solutions with size ranging from 442 to 1,184 nodes (760 on average), which is on average $15\times$ larger than the one produced by ϵ -LEXICASE and is not large enough to be considered (exponential) bloat. This extra complexity of the final solutions positively contributes at the performance of the algorithm. We are investigating a post-processing phase to simplify the final solutions.

On average, DT-EM produces solutions with 806 nodes and DT-NM with 591. DT-NM generates smaller solutions than DT-EM, this could be due to the fact that DT-NM has a smaller search space (DT-NM omits the *Min* and *Max* symbols). Evaluating smaller solutions require less computation, this explains why DT-NM requires less computation than SGP-DT and DT-EM (see Table 3).

Overfitting Figure 2 plots for each data set the median of the best RMSE by computational effort (number of evaluated tree nodes) for SGP-DT and its two variants. Unfortunately, the implementation of ϵ -LEXICASE that we used does not report the intermediate RMSE on test. We use the computational effort, rather the number of generations, for a fair comparison of the three techniques. This is because the number of evaluated nodes is not uniform across the generations.

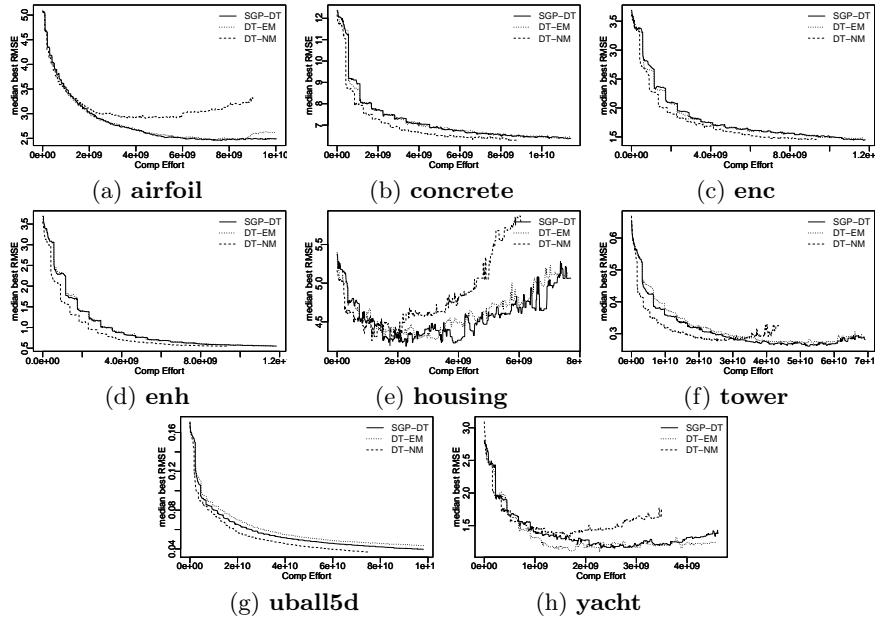


Fig. 2: Median RMSE of the best so far on the test set by computational effort.

The eight plots indicate that SGP-DT slightly overfits on the data sets *tower* and *yacht*, while on *housing* produces a substantial overfitting, which is comparable to the one of DT-EM but less severe than the one of DT-NM. DT-EM overfits four data sets: *airfoil* (Fig.2a), *housing* (Fig.2e), *tower* (Fig.2f), *yacht* (Fig.2h). The worst performance is from DT-NM that shows severe overfitting on *airfoil* (Fig.2a), *housing* (Fig.2e), *tower* (Fig.2f) and *yacht* (Fig.2h). Note that all three techniques overfit for the data sets *yacht* (Fig.2h) and *housing* (Fig.2e). This can be explain by their relatively low number of instances (see Table 1).

For the data sets *concrete* (Fig.2b), *enc* (Fig.2c) and *enh* (Fig.2d) all three techniques do not manifest overfitting (yet). Interestingly, in these three cases DT-NM arrives to a low RMSE with less computations than SGP-DT and DT-EM. We conjecture that this is because *concrete*, *enc* and *enh* are problems that do not need the additional expressiveness of the *Min* and *Max* symbols.

DT-NM is the technique that yields to the smallest individuals, as such we would expect less overfitting. Surprisingly, this is not the case. We believe that, to compensate the absence of discontinuity that *Max* and *Min* introduce, DT-NM used the protected divisions more frequently. This may lead to many asymptotic discontinuities, which are known to increase the overfitting [6].

When considering each data set individually, SGP-DT and DT-EM mostly manifest similar overfitting, while DT-NM manifests overfitting much earlier. This suggests that (i) the non-terminal symbols *Max* and *Min* help to alleviate the overfitting problem; and (ii) relying on the variance (SGP-DT) rather than MSE (DT-EM) in the fitness function indeed contributes to reduce RMSE (5.39% on average, see Table 2) but not to influence overfitting.

5 Conclusion

In this paper, we proposed SGP-DT, a new evolutionary technique that dynamically discovers and resolves intermediate dynamic targets. Our key intuition is that the synergy of the linear scaling and mutation helps to exchange good genetic materials during the evolution. Notably, SGP-DT does not rely on any form of crossover, and thus without suffering from its intrinsic limitations [2, 7]. Our experimental results confirm our intuitions and show that SGP-DT outperforms ϵ -LEXICASE in both lower RMSE and less computational cost. This is a promising result as ϵ -LEXICASE outperforms many GP-inspired algorithms [29].

This paper sparks interesting future work:

We do not perform any type of post-processing of the final solutions to reduce their size. Indeed, the solutions may contain redundant elements. We are currently investigating a post-processing step to minimize the size of the final solutions.

A possible future research direction is to automatically identify the proper number of iterations of SGP-DT. Indeed, problems with different complexity and nature may require a different number of external and internal iterations.

References

1. Vanneschi, L., Castelli, M., Silva, S.: A Survey of Semantic Methods in Genetic Programming. *Genetic Programming and Evolvable Machines* 15(2), pp. 195–214 (2014)
2. Pawlak, T.P., Wieloch, B., Krawiec, K.: Review and Comparative Analysis of Geometric Semantic Crossovers. *Genetic Programming and Evolvable Machines* 16(3), pp. 351–386 (2015)
3. Neill, M.: Semantic Methods in Genetic Programming. *Genetic Programming and Evolvable Machines* 17(1), pp. 3–4 (2016)
4. McPhee, N.F., Ohs, B., Hutchison, T.: Semantic Building Blocks in Genetic Programming. In: *European Conf. on Genetic Programming (EuroGP '08)* pp. 134–145 (2008)
5. Moraglio, A., Krawiec, K., Johnson, C.G.: Geometric Semantic Genetic Programming. In: *Parallel Problem Solving from Nature - PPSN XII*, pp. 21–31 (2012)
6. Keijzer, M.: Improving Symbolic Regression with Interval Arithmetic and Linear Scaling. In: *European Conf. on Genetic Programming (EuroGP '03)* pp. 70–82 (2003)
7. Poli, R., Langdon, W.B.: Schema Theory for Genetic Programming with One-Point Crossover and Point Mutation. *Evolutionary Computation* 6(3), pp. 231–252 (1998)
8. Efron, B., Hastie, T., Johnstone, I., Tibshirani, R., et al.: Least Angle Regression. *The Annals of Statistics* 32(2), pp. 407–499 (2004)
9. La Cava, W., Spector, L., Danai, K.: Epsilon-Lexicase Selection for Regression. In: *Proc. of the Conf. on Genetic and Evolutionary Computation (GECCO '16)* pp. 741–748 (2016)
10. Nicolau, M., Agapitos, A.: On the Effect of Function Set to the Generalisation of Symbolic Regression Models. In: *Proc. of the Companion of the Conf. on Genetic and Evolutionary Computation (GECCO '18)* pp. 272–273 (2018)
11. Ruberto, S., Vanneschi, L., Castelli, M.: Genetic Programming with Semantic Equivalence Classes. *Swarm and Evolutionary Computation* 44, pp. 453–469 (2019)

12. Keijzer, M.: Scaled Symbolic Regression. *Genetic Programming and Evolvable Machines* 5(3), pp. 259–269 (2004)
13. Gerules, G., Janikow, C.: A Survey of Modularity in Genetic Programming. In: the IEEE Congress on Evolutionary Computation (CEC '16) pp. 5034–5043 (2016)
14. Krawiec, K., Pawlak, T.: Locally Geometric Semantic Crossover: A Study on the Roles of Semantics and Homology in Recombination Operators. *Genetic Programming and Evolvable Machines* 14(1), pp. 31–63 (2013)
15. Krawiec, K., Liskowski, P.: Automatic Derivation of Search Objectives for Test-Based Genetic Programming. In: European Conf. on Genetic Programming (EuroGP '15) pp. 53–65 (2015)
16. Liskowski, P., Krawiec, K.: Online Discovery of Search Objectives For Test-Based Problems. *Evolutionary Computation* 25(3), pp. 375–406 (2017)
17. Otero, F.E.B., Johnson, C.G.: Automated Problem Decomposition for the Boolean Domain with Genetic Programming. In: *Genetic Programming* pp. 169–180 (2013)
18. Krawiec, K., O'Reilly, U.M.: Behavioral Programming: A Broader and More Detailed Take on Semantic GP. In: Proc. of the Conf. on Genetic and Evolutionary Computation. (GECCO '14) pp. 935–942 (2014)
19. Arnaldo, I., Krawiec, K., O'Reilly, U.M.: Multiple Regression Genetic Programming. In: Proc. of the Conf. on Genetic and Evolutionary Computation. (GECCO '14) pp. 879–886. (2014).
20. Ruberto, S., Vanneschi, L., Castelli, M., Silva, S.: Esagp – A Semantic GP Framework Based on Alignment in the Error Space. *Genetic Programming* pp. 150–161 (2014)
21. Vanneschi, L., Castelli, M., Scott, K., Trujillo, L.: Alignment-Based Genetic Programming for Real Life Applications. *Swarm Evolutionary Computation* 44, pp. 840–851 (2019)
22. Gandomi, A.H., Alavi, A.H.: A New Multi-Gene Genetic Programming Approach to Nonlinear System Modeling. *Neural Computing and Applications* 21(1), pp. 171–187 (2012)
23. Oliveira, L.O.V., Otero, F.E., Pappa, G.L., Albinati, J.: Sequential Symbolic Regression with Genetic Programming. In: *Genetic Programming Theory and Practice XII*, pp. 73–90 (2015)
24. Medernach, D., Fitzgerald, J., Azad, R.M.A., Ryan, C.: Wave: A Genetic Programming Approach to Divide and Conquer. In: Proc. of the Companion of the Conf. on Genetic and Evolutionary Computation. (GECCO '15) pp. 1435–1436 (2015)
25. Medernach, D., Fitzgerald, J., Azad, R.M.A., Ryan, C.: A new wave: A Dynamic Approach to Genetic Programming. In: Proc. of the Conf. on Genetic and Evolutionary Computation. (GECCO '16) pp. 757–764 (2016)
26. Asuncion, A., Newman, D.: UCI Machine Learning Repository (2007)
27. White, D.R., Mcdermott, J., Castelli, et al. Better GP Benchmarks: Community Survey Results and Proposals. *Genetic Programming and Evolvable Machines* 14(1), pp. 3–29 (2013)
28. Cava, W.L., Helmuth, T., Spector, L., Moore, J.H.: A Probabilistic and Multi-Objective Analysis of Lexicase Selection and ϵ -lexicase Selection. *Evolutionary Computation* pp. 1–28 (2018)
29. Orzechowski, P., Cava, W.L., Moore, J.H.: Where Are We Now?: A Large Benchmark Study of Recent Symbolic Regression Methods. In: Proc. of the Conf. on Genetic and Evolutionary Computation. (GECCO '18) pp. 1183–1190 (2018)
30. Castelli, M., Trujillo, L., Vanneschi, L., Silva, S. Geometric Semantic Genetic Programming with Local Search. In: Proc. of the Conf. on Genetic and Evolutionary Computation. (GECCO '15) pp. 999–1006 (2015)