This is the authors' version of the paper that has been accepted for publication in
Genetic Programming and Evolvable Machines (2021) 22:463–493
https://doi.org/10.1007/s10710-021-09419-3

A Semantic Genetic Programming Framework Based on Dynamic Targets

Stefano Ruberto · Valerio Terragni · Jason H. Moore

Received: date / Accepted: date

Abstract Semantic GP is a promising branch of GP that introduces semantic awareness during genetic evolution to improve various aspects of GP. 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 of previous runs. 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 final solutions with low approximation error and computational cost. We evaluate SGP-DT on eleven well-known data sets and compare with ϵ-LEXICASE, a state-of-the-art evolutionary technique, and seven Machine Learning techniques. SGP-DT achieves small RMSE values, on average 23.19% smaller than the one of ϵ-LEXICASE. Tuning SGP-DT's configuration greatly reduces the computational cost while still obtaining competitive results.

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

Acknowledgments National Institute of Health grant NIH R01 LM010098

Stefano Ruberto
University of Pennsylvania (currently at Joint Research Centre - European Commission)
Philadelphia, PA, USA
E-mail: stefano.ruberto@gmail.com

Valerio Terragni
University of Auckland
Auckland, New Zealand
E-mail: v.terragni@auckland.ac.nz

Jason Moore
University of Pennsylvania
Philadelphia, PA, USA
E-mail: jhmoore@upenn.edu

1 Introduction

Recently, researchers successfully applied Semantic methods to Genetic Programming (SGP) on different domains, showing promising results [31, 33, 47]. 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 [47]. 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 [47].

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 [47], we intend *"semantics"* as the set of output values of a program on the training cases [22]. Such an approach obtains a richer feedback during the evolution relying on the evaluation of the individuals on the training cases. More formally, the semantics of an individual $\mathcal{I}$ is a vector $sem(\mathcal{I}) = \langle y_1, y_2, \cdots, y_m \rangle$ of responses to the m inputs of the training cases. Let $sem(\hat{y}) = \langle \hat{y_1}, \hat{y_2}, \cdots, \hat{y_m} \rangle$ denote the semantic vector of the target (as defined in the training set), where $\hat{y_1}, \hat{y_2}, \cdots, \hat{y_m}$ are the desired outputs. SGP defines *semantic space* [47] 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 [25].

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. [25]. 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 [25]. However, such crossover operator has the major drawback of producing individuals with an exponentially increasing size (i.e., *exponential bloat*) [25, 47]. To avoid the exponential bloat, researchers proposed variants of this operator that minimize bloating [33] but at the cost of dropping the important guarantee of non-worsening crossover operations.

In this paper, we present a SGP approach called **SGP-DT** [41, 42] (***S**emantic **G**enetic **P**rogramming based on **D**ynamic **T**argets*) 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 $\hat{y}_i = \langle \hat{y_1}, \hat{y_2}, \cdots, \hat{y_m} \rangle$ 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* [14]. 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 [14]. 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 avoids 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 [36]. Despite the absence of crossovers, SGP-DT implicitly recombines different functionalities, similarly to a geometric crossover [25]. 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.

This paper summarizes and extends our previous conference paper that presented SGP-DT [41]. In particular, in the conference paper 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. [19]. 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× fewer tree computations than ϵ-LEXICASE (4.81× on average).

This paper presents an additional set of experiments that investigate alternative configurations of SGP-DT to reduce the computational cost while maintaining good performance. More specifically, we considered configurations of SGP-DT with a reduced computational cost by three orders of magnitude, with respect to the experiments presented in our previous conference paper. Then, we explore different configurations to understand what are the aspects of

Algorithm 1: SGP-DT

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

our technique that influence the accuracy and overfitting. We evaluated these new configuration of SGP-DT by comparing them with seven well known ML techniques on five datasets (three different from the previous ones) obtaining competitive results and precious insight on the SGP-DT characteristics. The paper is organized as follows. Section 2 describes our proposed approach. Section 3 discusses the related work. Section 4 reports our experimental evaluation and discusses the results. The experimental evaluation is divided into two parts. the first part (Cost-effectiveness experiments) summarizes the results presented in the conference paper. The second part (Sensitivity experiments) discusses the new results. Section 5 concludes the paper.

2 Methodology

This section describes the SGP-DT framework using the symbolic regression as the targeted ML problem.

Algorithm 1 overviews the SGP-DT approach. Given the values of the independent ($\overline{x}$) 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 [14, 27]. 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 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 $\mathcal{I}$ [14]. 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 [14]. 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 [38].

Linear scaling has another important property: it gives an upper bound on the error [14]. 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 [14], we compute the linear scaling of an individual $\mathcal{I}$ as follows:

$$\mathcal{I}_{ls} = a + b \cdot \mathcal{I} \tag{1}$$

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

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

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

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 [15]:

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

where m is the number of training cases (y).
At each new external iteration the residual error becomes the new target (line 21).

$$target = \hat{y} - sem(\mathcal{I}^{\star}_{ls}(\overline{x})) \tag{5}$$

where $sem(\mathcal{I}^{\star}_{ls}(\overline{x}))$ 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 $\sigma^2(error) \leq \sigma^2(\hat{y})$, where $error = target - sem(\mathcal{I}^{\star}_{les}(\overline{x}))$. 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 [11]. 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 [18, 33, 36]. According to McPhee et al. [22] and Ruberto et al. [38] 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* [22] among the building blocks of the individuals exchanged during crossover, which

is important to increase the chance of obtaining a semantically meaningful offspring [18]. The idea of determining a common context has been introduced by Poli and Langdon with the one-point crossover operator [36]. 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.

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 ($\overline{x}_{val}$ 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 a 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. [16] and Liskowski et al. [20] 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 [30]. 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 [17] 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 [17] 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" [17] 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 [17].

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. [30] is the only one that progressively builds the solution but it uses a strategy that works for Boolean trees only.

Linear combinations MRGP [1] uses multiple linear regression to combine the semantics of sub-programs (subtrees) to form the semantic of an individual.

Ruberto et al. proposed ESAGP [37], 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 [48], which performs linear combinations between two aligned chromosomes belonging to the same individual.

Gandomi et al. proposed MGGP [10], 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 [25], 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 [25].

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 [25]). 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

Table 1: Data sets of regression problems.

name	# attributes	# instances	source	name	# attributes	# instances	source
airfoil	5	1,503	UCI [2]	housing	14	506	UCI [2]
concrete	8	1,030		tower	25	3,135	
enc	8	768		yacht	6	309	
enh	8	768		uball5d	5	6,024	[49]

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) [28] uses the crossover operator GSGP [25] to iteratively transform the target using a semantic distance that resembles the classical residual approach. However, no statistically significant difference (on the errors) from the classical GP approach was found [28]. 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 [23, 24] 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 "fuzzifying" 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 Heterogeneous nature of WAVE emulates this dynamic evolutionary environment by simulating periods of a rapid change [23, 24]. The effectiveness of such an approach requires specific combinations of system parameters that converge to a fitter solution. Due to the huge space of possible system parameters, finding such combinations often requires a large number of iterations [23, 24]. 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.

4 Experiments

We performed two sets of experiments. The first set of experiments aims to evaluate the *cost-effectiveness* of our approach compared to state-of-the-art methods.We compared the approach's performance in terms of Root Square Mean Error (RMSE) and computational cost measured with the number of evaluated nodes. The results show that SGP-DT outperforms state-of-the-art

evolutionary approaches. However, evolutionary techniques are computationally expensive compared to other *Machine Learning* (ML) methods because they require the evaluations of populations of solutions. SGP-DT is not an exception to this rule. In the second set of experiments, we drastically reduce the computations required by SGP-DT constraining the population size to 100 individuals. From this starting point, we investigate key parameters of SGP-DT to understand if it's possible to maintain a good accuracy of the models while drastically reducing the computational cost. The results from the first set of experiments might suggest that SGP-DT suffers from the overfitting problem. We investigate if reducing the computations also helps with overfitting. Having identified the configurations with an interesting trade-off between effectiveness and cost of the analysis, we perform a final test comparing SGP-DT with seven well-known ML techniques. The final tests show that SGP-DT has a competitive accuracy, having reduced its computational cost by three orders of magnitude with respect to the first set of experiments.

4.1 Cost-Effectiveness Experiments

4.1.1 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 [1, 10, 19, 23, 24, 48]. Table 1 shows the name, number of attributes, and number of instances for each data set. All data sets expect *uball5d* are from the *UCI* repository [2].

For *uball5d*[1], we followed the same configuration used by Cava et al. [6].

4.1.2 Methods

We compared SGP-DT with two techniques (LASSO [8] and ϵ-LEXICASE [19]) 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 [19]. 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 [19]. Each parent selection begins with a randomized ordering of both the training cases and the solutions in the selection pool (i.e., population).

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

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 [19] outperforms many GP-inspired algorithms [29]. We relied on the publicly available implementation of ϵ-LEXICASE, *ellyn*[2], 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.

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

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

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.

4.1.3 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

[2] https://github.com/EpistasisLab/ellyn

Table 2: Median RMSE of the 50 trials.

	Root Mean Square Error (RMSE)					Median RMSE % decrease of SGP-DT over:			
Data set	**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 %**

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.

4.1.4 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 techniques[3]. 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 trials[4]. When comparing the RMSE values we performed 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 statistical 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

[3] 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

[4] 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

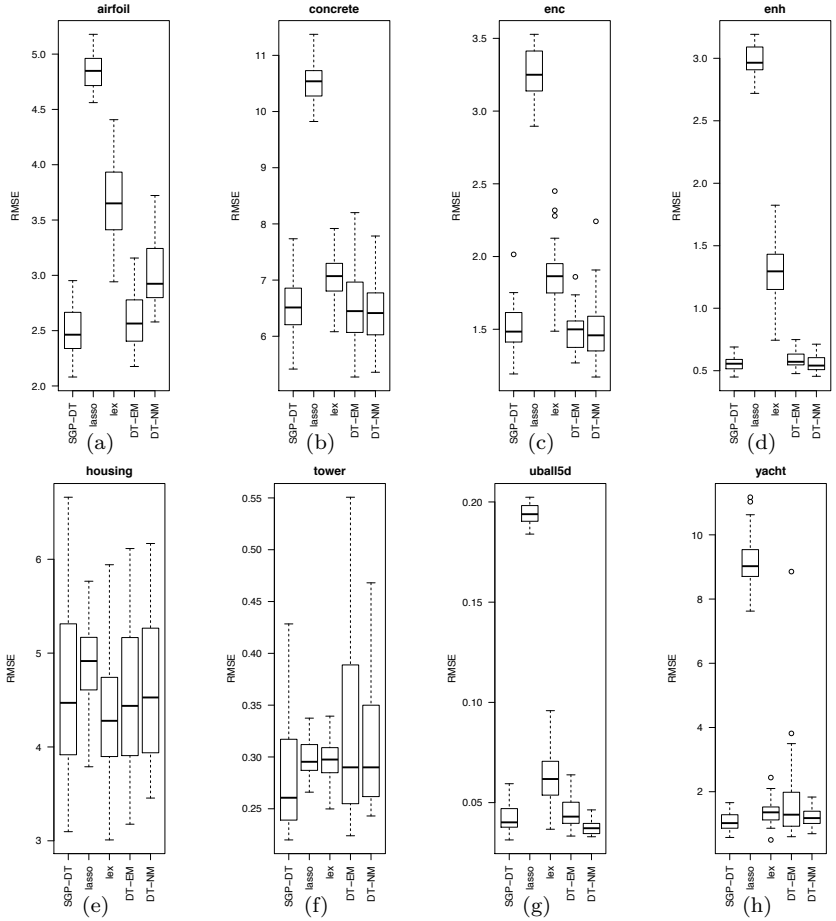


Fig. 1: RMSE of test set for all the techniques and for all the eight data sets.

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, but *uball5d* where the difference is statistically significant (the decrease is -7.87%).

Error comparison with related work Unfortunately, the implementation of WAVE [23, 24] is not publicly available, and thus a direct comparison would be difficult. We extracted the median RMSE from the GECCO 2016 paper [24] 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. [48], we extracted the median RMSE on the data set *concrete* of the following GP techniques: 10.44 (NA-GP [48]),

8.1 (NA-GP-50 [48]), 12.50 (GSGP [25]), and 9.43 (GSGP-LS [5]). 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.

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. [38], 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 shows that all the comparisons between each pair of techniques are statistically significant, except for the comparison with SGP-DT and DT-NM on subject *uball5d*.

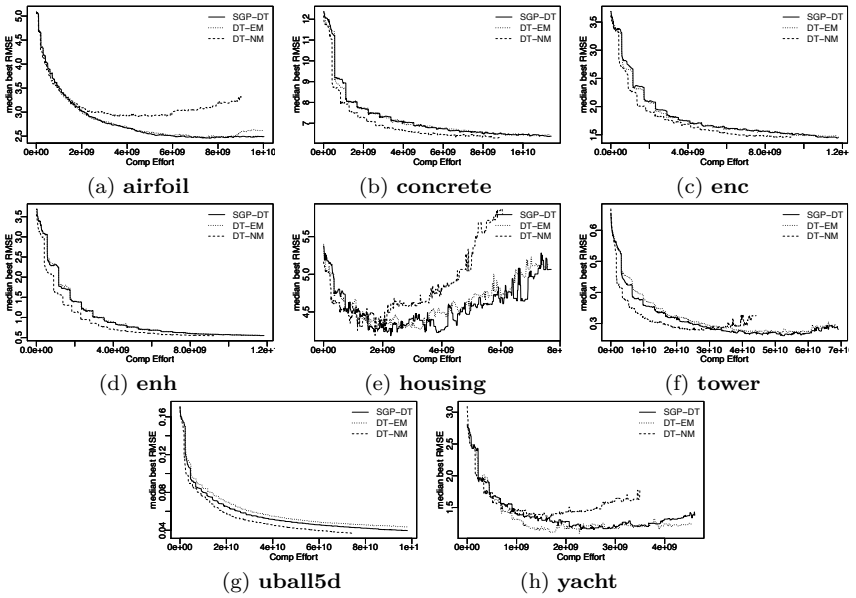
Table 3 reports the median number of nodes (of the 50 runs) that the GP techniques evaluate to produce the final solution. The last three columns of Table 3 report the ratio between the number of node evaluations of SGP-DT with those of ϵ-LEXICASE, DT-EM and DT-NM. A ratio greater (lower) than one means 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× and 9.26×, 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× 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 fewer nodes than SGP-DT (0.77× 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 [19]. SGP-DT produces solutions with size ranging from 442 to 1,184 nodes (760 on average), which is on average 15× 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 to the performance of the algorithm. We are investigating a post-processing phase to simplify the final solutions.

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

	Median number of evaluated nodes				Reduction ratio of SGP-DT over		
Data set	**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**×



(a) **airfoil** (b) **concrete** (c) **enc**

(d) **enh** (e) **housing** (f) **tower**

(g) **uball5d** (h) **yacht**

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

On average, DT-EM produces solutions with 806 nodes and DT-NM with 591. DT-NM generates smaller solutions than DT-EM, perhaps because DT-NM has a smaller search space (DT-NM omits the *Min* and *Max* symbols). Evaluating smaller solutions requires less computation; this explains why DT-NM requires fewer computations 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.

The eight plots indicate that SGP-DT slightly overfits 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 the data sets *yacht* (Fig.2h) and *housing* (Fig.2e). This can be explained 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 computation 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 the smaller 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 overfitting [14].

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 reducing RMSE (5.39% on average, see Table 2) but it does not influence overfitting.

4.2 Sensitivity Experiments

In this section, we perform a series of experiments to investigate whether we can further reduce the computational cost of SGP-DT while maintaining high performance. Similarly to many evolutionary approaches, SGP-DT is computationally more expensive than other Machine Learning methods. In this set of experiments, we decrease the population size from 1,000 to 100 because this is one of the factors that greatly influence the computational cost of evolutionary approaches. After setting this value, we investigated how other parameters of the algorithms can be adjusted to maintain an acceptable level of accuracy and overfitting.

We took a subset of the datasets from the *cost-effectiveness* experiments (Section 4.1) in which SGP-DT manifests different degrees of overfitting. Then, we evaluate the change in performances with respect to the previous tests. We added three new datasets from the *UCI* repository [2] to evaluate SGP-DT in other scenarios. Using these datasets, we perform a sensitivity analysis from three viewpoints: 1) the size of the validation set and the validation strategy itself, 2) the number of internal iterations, and 3) the maximum depth of the trees representing the individuals in the population. These are key parameters

that directly influence the performance of SGP-DT and the computational costs.

All the experiments in this section use a training set equal to 70% of the dataset. The other parameters that are not specified explicitly are kept the same as in the *cost-effectiveness* test section.

4.2.1 Datasets

For the experiments in section 4.2, we consider five datasets: *airfoil*, *yacht*, *ccpp*, *aq_tox*, *real_estate*.

airfoil and *yacht* are the same datasets used in the experiments of Section 4.1. We choose them because SGP-DT manifests neither strong nor weak overfit. We think that in this intermediate condition would be easier to understand what factors would make SGP-DT overfit more or less. Moreover, *airfoil* is among the datasets in Table 1 with the highest number of instances (1,503). On the contrary, *yacht* is the dataset with fewer instances (309), giving us the possibility to explore these two different conditions.

The *ccpp* [46] dataset consists of measurements of environmental variables influencing the production of energy in a Combined Cycle Power Plant over a period of six years (2006-2011). To predict the net hourly electrical energy output, we use the following features: temperature, ambient pressure, relative humidity, and exhaust vacuum. This dataset has the highest number of data points, so we expect lower overfitting.

To test the performance of SGP-DT in challenging conditions, we included the datasets *aq_tox* [4] and *real_estate* [50], which are known to be prone to overfitting. The *aq_tox* dataset aims to predict the acute aquatic toxicity towards Daphnia Magna (LC50 data) given eight molecular descriptors. The variable to predict in the *real_estate* dataset is the price of unit area in the selected cities. The dataset includes six features: transaction age, house age, distance from the nearest subway-like stations, and geographical coordinates.

4.2.2 Validation set sensitivity analysis

A common approach to mitigate the overfitting problem is using a validation set during training as a stopping criterion. In our experiments, SGP-DT uses a validation set of 10% of the training set. More specifically, after all the external iterations, the Function VALIDATE-AND-SELECT at line 22 of Algorithm 1 scans the sequence of *partial 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 produced before the smallest MSE. Such a sequence represents the transformation chain of the dynamic targets.

The size of the validation set is a key parameter when overfitting is concerned. Theoretically, the larger the validation set, the lower the risk of overfitting.

Validation and training sets are complementary. Large validation sets would lead to a small training set, hence to an underperforming model.

To understand this trade-off when using SGP-DT, we perform a sensitivity analysis on the size of the validation set. We run SGP-DT with five configurations, varying the validation size while keeping fixed the other parameters values. We choose the following values of validation size: 5%, 10%, 20%, 30% and 40% (the experiments in Section 4.1 uses 10%).

For all the other settings, we choose a default configuration. We set the maximum depth of the trees to 5, while the number of internal iteration is 50 and the total number of iterations 2000.

Another key design choice of SGP-DT is the validation strategy to mitigate the overfitting problem. It is important to clarify that SGP-DT relies on the validation set to construct the final model after completing a predefined number of external iterations. We call this strategy *global minimum*. SGP-DT does not use the validation set at runtime to decide when to stop the internal or external iterations. Although a stopping criterion could drastically reduce the computational cost, it is often hard to find the correct threshold that stops the evolution at the right moment without preventing the algorithm from finding a model in which the validation set further reduces the RMSE. To confirm our intuition, we repeated the experiments using as stopping criterion the first local minima in the validation errors of the partial models. We call this strategy *first minimum*, which monitors the validation error at runtime and stops the evolution if the error is higher than a given threshold. We used the same rolling mean (with a window size of 20) used by the strategy *global minimum*. The rolling mean helps to give some tolerance against error fluctuations.

We run SGP-DT 30 times for each of the five configurations, each strategy (*global minimum* and *first minimum*), and each dataset (1,500 runs in total). Figure 3 shows the distributions of the RMSE values of the 30 runs for different combinations of validation sizes and stopping criteria. The distributions marked with only the percentage of the validation set represent the configurations with the *global minimum* strategy, and the ones marked with '%f' are the configurations with the *first minimum*.

Global minimum. As expected, the smallest and largest validation sizes (5% and 40%) lead to the worst performance (the median RMSE and the variance are higher). This confirms that a small validation set does not help mitigate the overfitting problem and that a large validation set leads to a model that does not generalize well on unseen data. This phenomena is more obvious for the datasets prone to high overfitting (*real_estate*, *aq_tox* and *yacht*). Overall, the validation sizes that lead to better performance are 10% and 20%.

First minimum. The results confirm our hypothesis that the *first minimum* strategy performs much worse than the *global minimum* one. For every dataset and validation size, the original strategy of SGP-DT performs better. This validates our hypothesis that an early stopping criterion based on the first minimum is not effective. However, the *global optimum* strategy improvements

come at the cost of performing more computations for partial models not included in the final model.

Comparing the configurations in Figure 3 we performed a non-parametric pairwise Wilcoxon rank-sum test with Holm correction for multiple testing, with a confidence level of 95% (p-value <0.05). The P-values confirm significant differences between the RMSE computed considering *global minimum* and *first minimum* strategies. Exceptions to this general finding are found in the dataset that overfit more (i.e., *real_estate* and *aq_tox*) where all the patterns are less clear. For this dataset, the *first minimum* and the *global minimum* strategies are not statistically different for all the configurations. Differences within the same type of validation strategy having different validation sizes are generally non-significant. We believe that more marked differences could be observed performing experiments with more external iterations (N_{ext}).

We noticed the presence of some outlier not represented in Figure 3 and this influence the variance of the results. Considering the outliers, the RMSE variance is higher for the methods relying on the *global minimum* strategy. That happens because this method generally considers more partial models and, in some cases, may produce overfit even if not detected by the validation set. The first minimum method is more conservative and thus generates fewer outliers.

We observe that the validation strategy (*global or first minimum*) is more important than the validation set size and that the performance does not degrade abruptly modifying this parameter.

4.2.3 Parameter sensitivity

To better understand how SGP-DT works, we analyzed two characteristic parameters of this algorithm.

One key parameter of SGP-DT is the maximum depth of the tree that represents each partial model. Such a parameter influences two essential aspects of the algorithm.

First, evaluating and mutating deeper trees increases the computational costs of the algorithm. Second, by considering deeper trees, we increase the expressiveness of the models that can capture complex data patterns. However, a well-known drawback of complex solutions is that they are prone to overfit the training data. Simpler (smaller) solutions often generalize better with unseen data.

In the experiments that evaluated the *cost-effectiveness* of SGP-DT, we set no limit on the maximum depth of the tree that represents each partial model. However, setting a maximum depth of the tree could substantially reduce the computational cost of SGP-DT and, at the same time, improve the generalization ability of the final model. To shed light on this opportunity, we performed a sensitivity analysis on the maximum depth of the tree. We choose the values 5, 10, and 15.

Another essential parameter of SGP-DT is the number of internal iterations (N_{int}), i.e., the number of generations that evolve each partial model. The

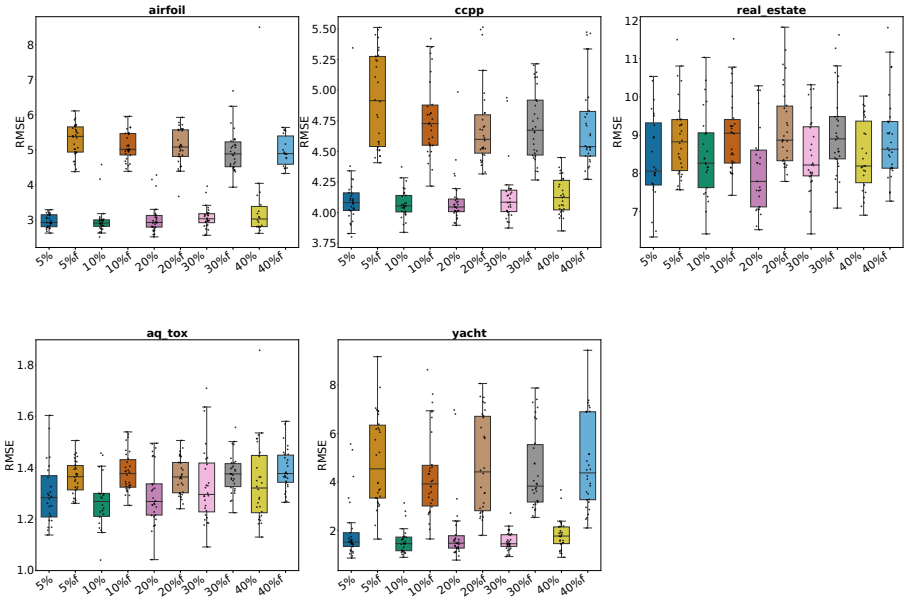


Fig. 3: RMSE for different sizes of the validation set is expressed as the portion of the training set (in percentage). The final results have been obtained considering the point where the validation error is lower

choice of such a parameter is much more crucial than the number of external iterations (N_{ext}). Indeed, the validation set helps to identify the best number of external iterations that leads to the final model with the lowest MSE. Conversely, the number of internal iterations (N_{int}) is a fixed value, which SGP-DT does not automatically adjust for the particular problem at hand.

Similarly to the maximum depth of the tree, the number of internal iterations (N_{int}) influences both the computation cost and the generalization ability of the final model. Indeed, the more generations each GP run performs the higher are the computational cost and the chance to obtain a fitter model for the training set.

We choose the values: 25, 50, 100, 150 for N_{int}. We performed a set of 30 runs for every combination of N_{int} and maximum depth of the tree (dth) parameters totaling 12 different configurations. For the other parameters, we choose the following default configuration: a validation set size equal to the 20% of the training and the number of total iterations equal to 5000.

Using the global minimum method for validation we computed the results showed in Figure 4. A first look at *ccpp* and *yacht* results reveal two opposite pattern: in *ccpp* increasing the number internal iterations worsen the performance while for *yacht* the opposite is true. These results underline the importance of the dataset's characteristics for the performance of SGP-DT. To reveal other patterns, we build a grid showing the relative performance of each parameter combination. Figure 5 represent better performance in green

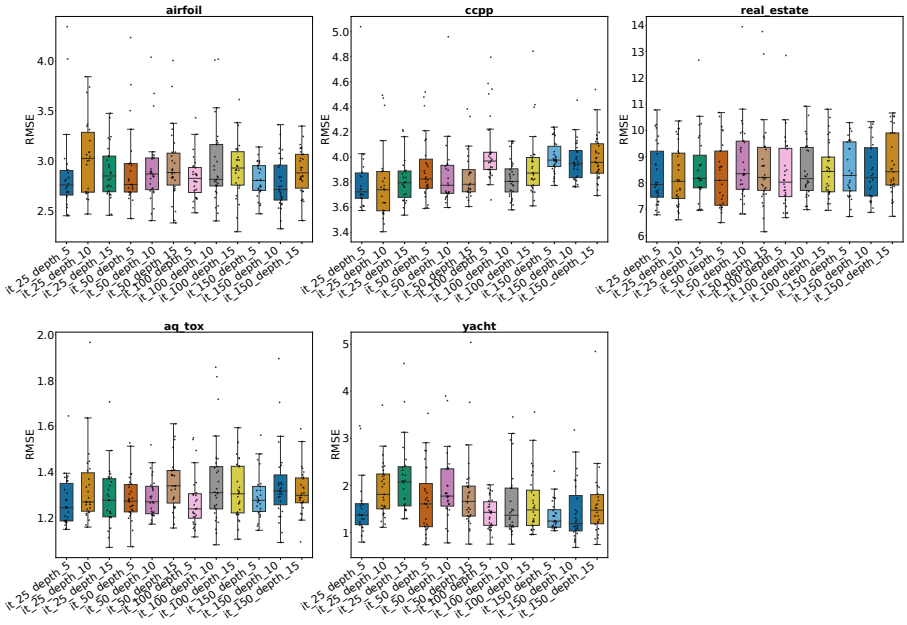


Fig. 4: Test results showing RMSE for all the problems and all the combinations of parameters (it = Internal Iterations and depth = maximum tree depth)

and worst performance in red. We notice that the combination with a depth of 5 are more advantageous in three dataset (i.e., *airfoil*, *aq_tox*, and *yacht*). High-performance combinations with a depth of 10 are fewer, and even fewer are the combinations having a depth of 15. Looking at Figure 5 we have the feeling that a higher depth requires more internal iterations to be effective, probably because optimizing deeper trees requires more iterations.

Comparing the results in Figure 4, we performed a non-parametric pairwise Wilcoxon rank-sum test with Holm correction for multiple testing, with a confidence level of 95% (p-value <0.05). There are only a few results that show statistically significant differences. Considering the parameters couple formed by internal iterations and max trees depth ($N_{\text{int}}, depth$) in Figure 5, for the dataset *airfoil* the combination $(150, 10)$ is significantly better than $(25, 10)$. For the dataset *yacht* combination $(100, 5)$ is better than $(50, 10)$, $(25, 10)$ and $(25, 15)$ while $(150, 5)$ is better than $(25, 15)$. The other comparisons are not statistically significant.

To further clarify the role of these parameters in SGP-DT, we observe the evolution's dynamic by reporting the Median RMSE of the 30 runs with respect to the number of nodes evaluated during the whole evolutionary process, see Figures 6, 7, 8. Different tree depth and internal iterations imply different computational costs. For this reason, we prefer to refer to the x axes the number of nodes evaluated as a measure of the computational costs. We set in advance the number of total iterations in our experiments, and so the lines in Figures 6, 7, 8 vary in length. Looking directly at the errors' dynamics exclude

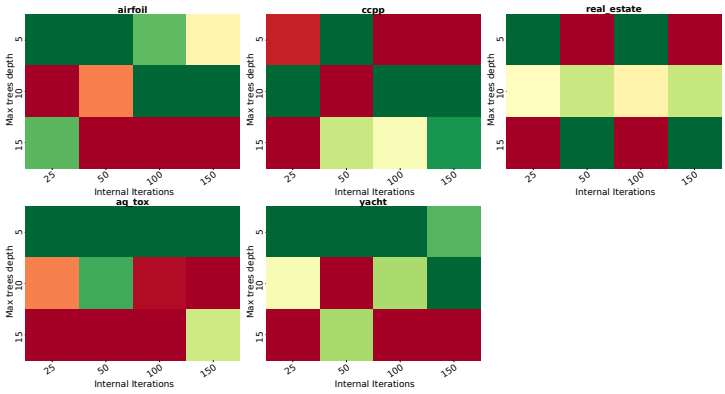


Fig. 5: RMSE comparison for the combination of the Internal Iteration and Max Tree Depth parameters. Green combinations are better than red ones for the given problem

any possible effect of the validation process on the final result, thus highlighting the internal iterations and max tree depth parameters influence.

Figure 8 shows the training results. We observe that combinations of fewer internal iterations with a limited tree depth lead to lower RMSE in the training set. Moreover, employing a higher number of partial models looks advantageous for the training.

We observe a different pattern for the test and validation error dynamics. In Figures 6 and 7, the combinations that lead to better performances (the lowest RMSE in the graph) in three cases out of five are the ones with a higher number of internal iterations (e.g., in Figure 6, 150 for *airfoil* and *yacht* while 100 for *aq_tox*). Furthermore, in all cases being equal to the number of computations, combinations with fewer internal iterations overfit sooner. More internal iterations lead to more stable results. In this case, the validation procedure can be more robust with respect to fluctuations in the errors' value. Figure 6 shows that SGP-DT can reach lower test errors when using fewer partial models but more optimized through more internal iterations. Overall, the overfitting phenomena appear to be related mainly with the number of partial models included in the final model and less with the internal iterations used to produce the partial models. Finally, the comparison of Figure 2 with Figure 6 shows that the constrained configurations of this section are less computationally expensive by three orders of magnitude.

4.2.4 Comparison with other ML techniques

This section presents a more general comparison between off-the-shelf Machine Learning techniques (ML) and SGP-DT.

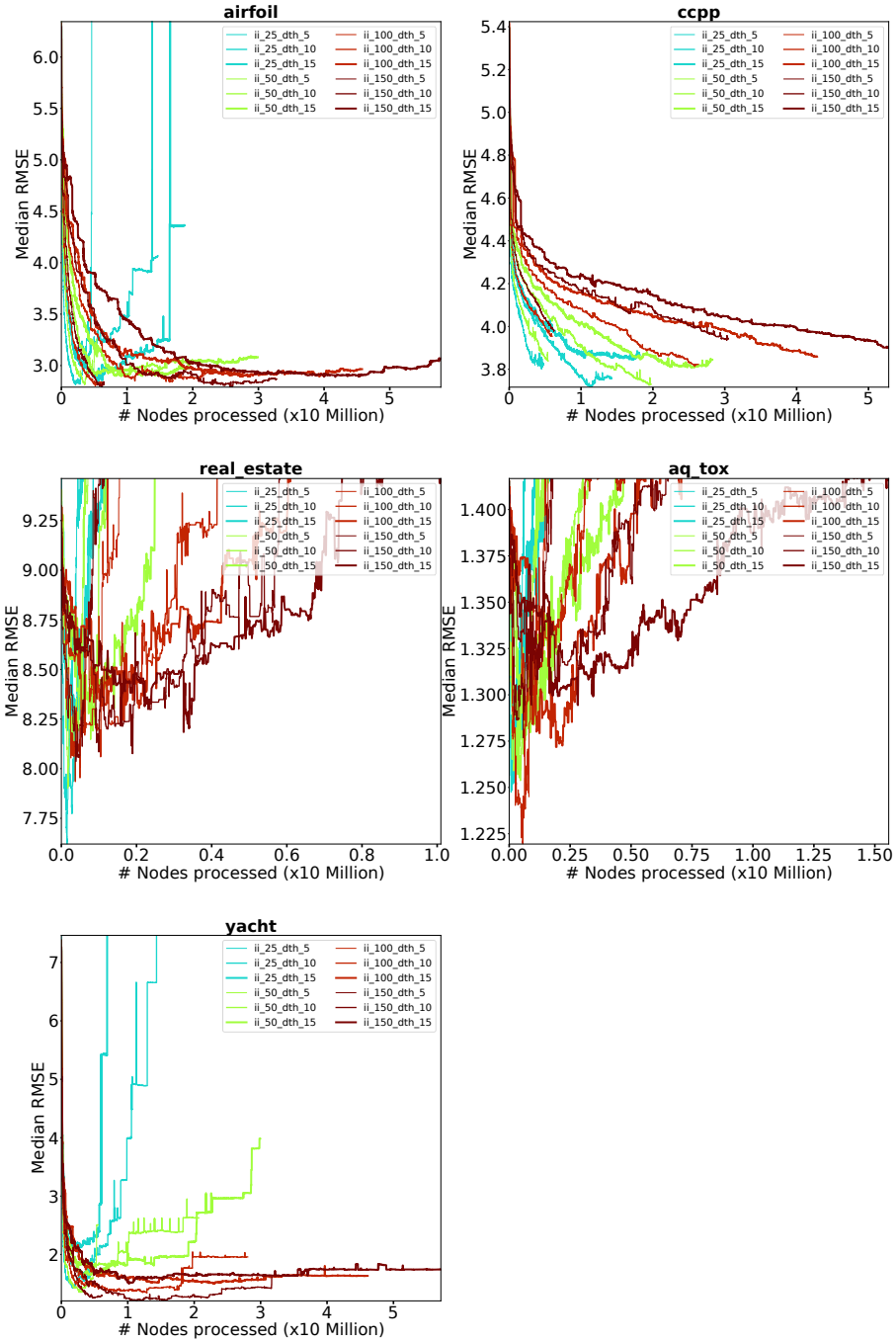


Fig. 6: Median RMSE test error for different combinations of Internal Iteration and Max Tree Depth parameters

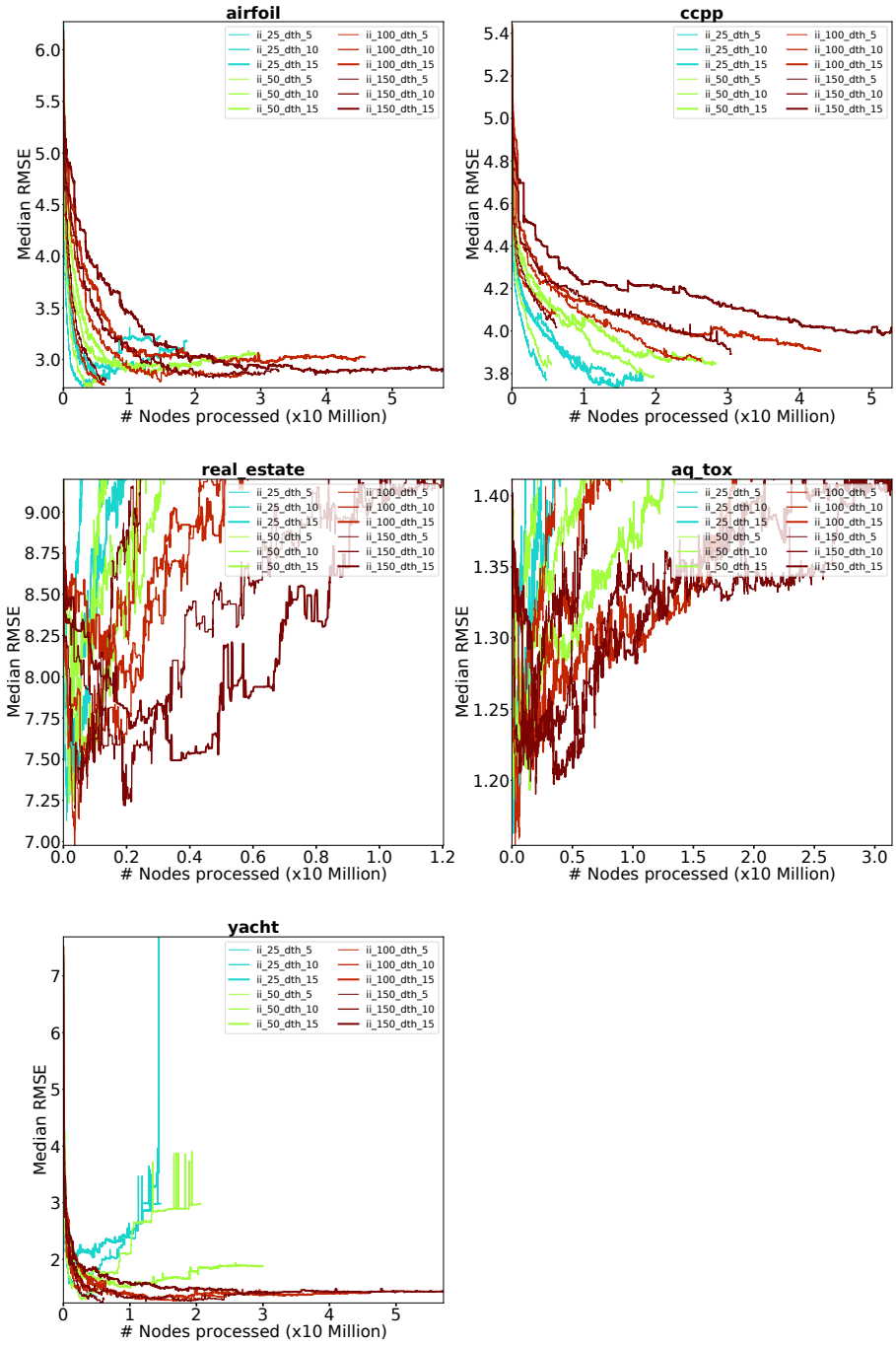


Fig. 7: Median RMSE validation error for different combinations of Internal Iteration and Max Tree Depth parameters

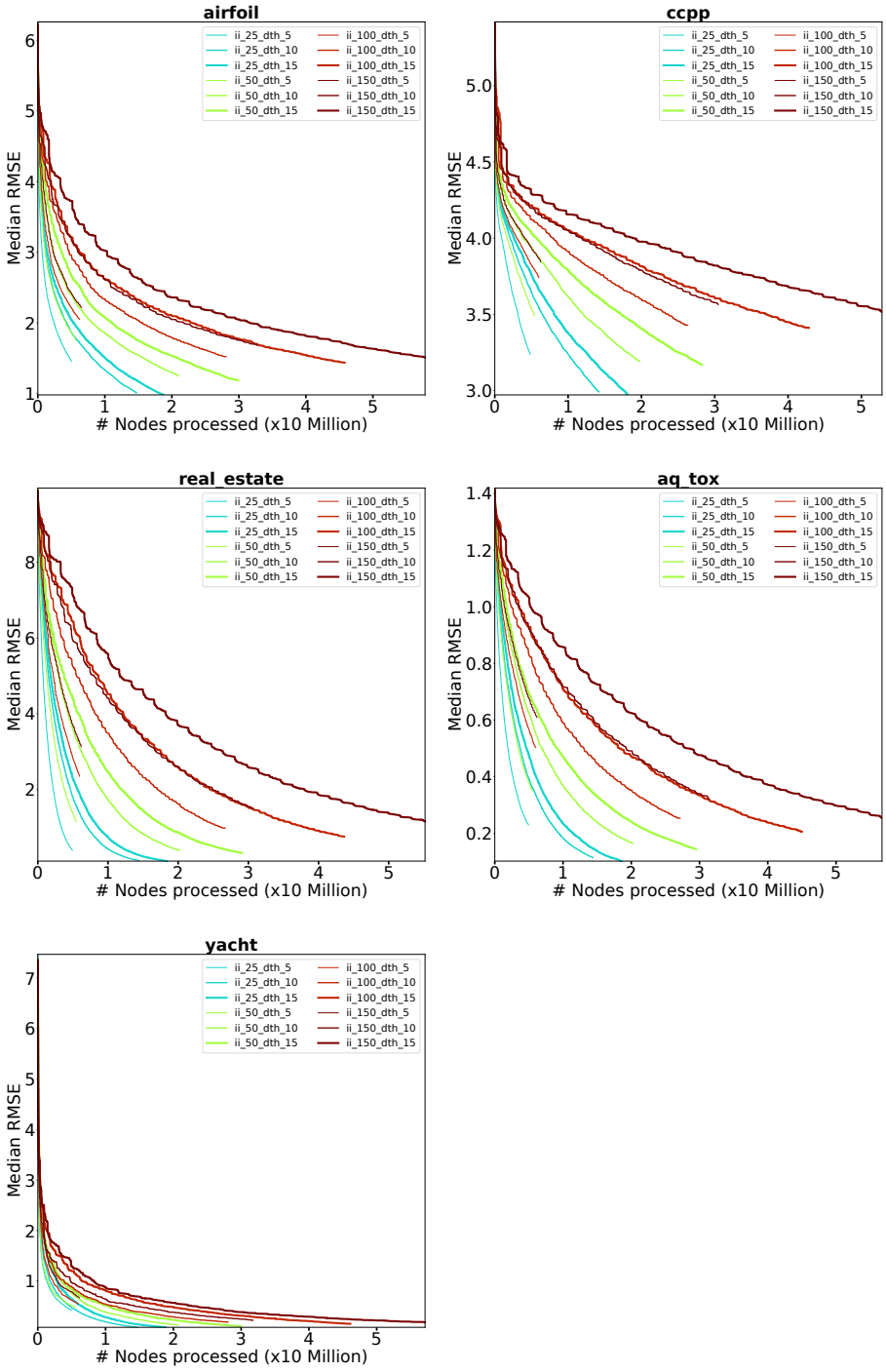


Fig. 8: Median RMSE train error for different combinations of Internal Iteration and Max Tree Depth parameters

Table 4: SGP-DT Variance of ML techniques and SGP-DT. Outliers in SGP-DT results push the value of the variance

	AdaBoost	GradientBoosting	Lasso	MLP	NearestNeighbors	RBFSVM	RandomForest	SGPDT
airfoil	0.02	0.02	0.02	0.61	0.05	0.02	0.01	479440.14
ccpp	0.07	0.01	0.01	0.02	0.01	0.02	0.01	63.65
real_estate_valuation	1.58	1.40	1.54	24.48	0.63	1.20	1.56	50.96
qsar_aquatic_toxicity	0.00	0.00	0.01	0.20	0.01	0.01	0.00	0.01
yacht	0.05	0.05	2.77	5.76	2.44	2.37	0.09	144.96

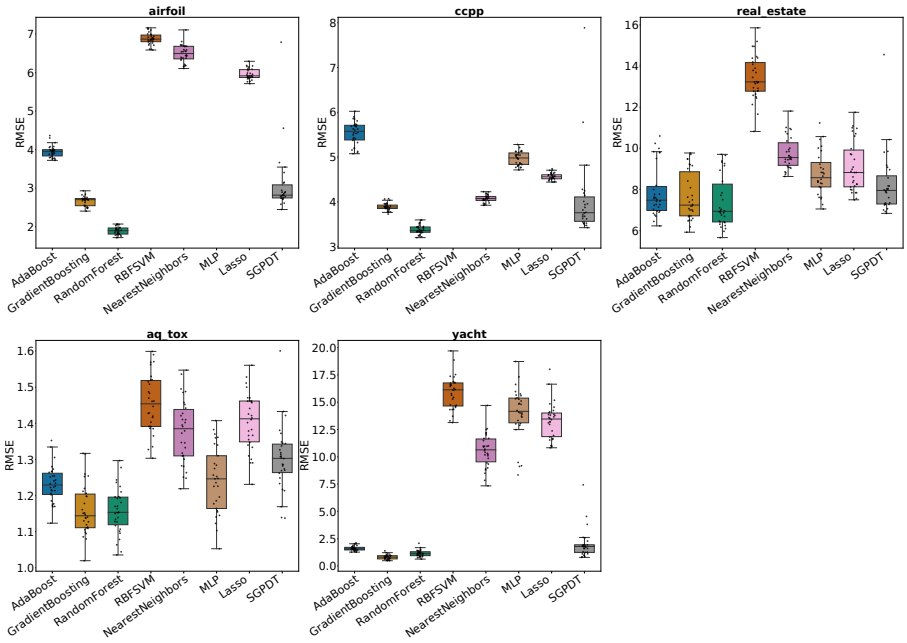


Fig. 9: RMSE comparison of SGP-DT with seven ML techniques

In the previous sections, we have explored different parameter values to reduce the computational cost of SGP-DT while maintaining a similar performance. We limited the maximum depth of the trees, reduced the internal iterations and population size, achieving a reduction of the computational cost of three orders of magnitude with respect to the configuration of SGP-DT used in the first set of experiments. We estimate new parameter values based on the previous tests to understand the impact that this reduction in computational cost has to the performance of the SGP-DT.

Evolutionary approaches are know to have higher computational costs than other ML approaches. In our second set of experiments, the attempt to abruptly compress such costs could results in an unacceptable loss of accuracy. The goal of this last section is to compare the performance of SGP-DT with other ML techniques to be informed about the feasibility of this cost reduction strategy for SGP-DT. We do not expect leading performance because of the very aggressive configuration. Instead, we aim at depicting an informative framework that

Table 5: SGP-DT configurations used for the final comparison test

	Internal Iterations	Total Iterations	Max Tree Depth	Validation Size	Population Size
airfoil	150	21500	5	20%	100
cccp	50	10000	10	20%	100
real_estate	25	200	5	20%	100
aq_tox	100	200	5	20%	100
yacht	150	2000	10	20%	100

help us to validate this line of thinking. On the other end, to understand the full potential of SGP-DT, the reader should consider that the performance of our main evolutionary competitor, ϵ-LEXICASE, has already been evaluated by Orzechowski et al. [29] in comparison with other ML techniques. As such, it is easy to have a clear idea of SGP-DT performance with respect to other ML techniques when the typical evolutionary cost is not an issue.

Table 5 shows the configurations of SGP-DT for each dataset. We select the configuration that had the lowest RMSE in Figure 6 for the problems overfitting soon (i.e. *real_estate*,*aq_tox*,*yacht*) and we set the maximum iterations to a point after the minimum test RMSE. For the remaining problems (i.e., *airfoil* and *ccpp*), we select the lowest RMSE configuration that did not overfit. The idea is that with more iterations, these configurations can improve significantly. In these latter cases, the number of iterations is set so that it potentially achieves the minimum RMSE. The computational costs are still very limited. As before, we run SGP-DT and all the ML techniques 30 times for each of the five problems. For this comparison we used the following ML regression methods from scikit-learn [34]: AdaBoost [9], Gradient Boosting [12], random forest [3] (using 100 estimators), Epsilon-Support Vector Regression with Radial Basis Functions Kernel [35] (RBFSVM), K-Nearest Neighbors, Multi-layer Perceptron (MLP with 3 thousand iterations) [13], Lasso [8]. All the ML parameters not specified here were set as the default. This comparison is not fair because we did not try to tune the ML parameters and the primary goal of SGP-DT tuning is a drastic saving in computational power. However, this test is useful to understand the potential of a more balanced trade-off between the initial test configurations and these last ones. We show the results on the test set in Figure 9. The performance of SGP-DT looks competitive in all the datasets, although it is never the best option.

As expected, the accuracy of SGP-DT decreased. Anyway, being still competitive, we believe that a more balanced parameters' setting, especially for the population size, would deliver a good compromise, also optimizing computational costs. In the light of this result, we think that this setup represents a sort of lower bound for the computational cost of SGP-DT. Below this threshold, probably, the performance of SGP-DT would degrade too much.

For the dataset *airfoil*, this constrained configuration of SGP-DT is still better than ϵ-LEXICASE in the first set of experiments. We observed more outliers in the test results of SGP-DT not shown in Figure 9. This condition is reflected in the anomalous value of the variance showed in Table 4. Even if we

did not perform a specific experiment, we believe that the presence of outliers is an issue connected with the population size. In the comparison of SGP-DT with ϵ-LEXICASE this was a minor issue even with the anecdotal test conducted with different parameters but the same population size (1000). Probably the selection acting on vast populations can promote individuals that generalize better.

5 Conclusion

This paper presents 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 evolution. Notably, SGP-DT does not rely on any form of crossover, and thus without suffering from its intrinsic limitations [33, 36]. 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].

As a classical evolutionary technique, SGP-DT incurs computational cost issues for evaluating the individuals in the population. In this paper we investigated the possible trade-off between computational cost and accuracy of the model. With this in mind, in the second part of this study, we explored key parameters and strategies of SGP-DT. We reduced the population's size and tuned the other parameters to maintain a good models' accuracy. This lead to a decrease of the computation cost by three orders of magnitude, while keeping acceptable performance. The final comparison shows that SGP-DT has competitive results with respect to ϵ-LEXICASE and seven well-known Machine Learning techniques.

SGP-DT could employ additional strategies to reduce the computational cost. For example, bloat control techniques can be used during internal iterations to reduce the number of nodes computed to obtain the partial models [7, 21, 26, 32, 44, 45]. An interesting future work would be to investigate the synergy of SGP-DT and one of such bloat control techniques. For instance, Nguyen et al.'s approach [26] reduces computation costs without sacrificing the performance of the GP models.

Our sensitivity analysis results show that the behavior of SGP-DT with respect to overfitting and performance, in general, is rather predictable given different parameter combinations: configurations having fewer internal iterations and smaller trees get to the minimum error point and overfit sooner (see Figures 6, 7, 8). This observation, in our opinion, opens the possibility of learning a computational model of the parameters or at least some heuristic to guide the SGP-DT's parameter tuning task.

We showed that the validation procedure is critical for SGP-DT and sometimes fails. With smaller populations, this produces outliers in the test error

results. A better validation strategy will increase the reliability of SGP-DT, even in applications with smaller datasets, like the ones in this paper.

Algorithms based on SGP-DT have already been developed for different applications, ranging from image analysis [39, 40] to multi-class classification [43], showing promising results. As such, SGP-DT can be considered as a framework for semantic GP.

References

1. Arnaldo I, Krawiec K, O'Reilly UM (2014) Multiple regression genetic programming. In: Proceedings of the 2014 Annual Conference on Genetic and Evolutionary Computation, ACM, New York, NY, USA, GECCO '14, pp 879–886, DOI 10.1145/2576768.2598291, URL http://proxy.library.upenn.edu:4604/10.1145/2576768.2598291
2. Asuncion A, Newman D (2007) Uci machine learning repository
3. Breiman L (2001) Random forests. Machine learning 45(1):5–32
4. Cassotti M, Ballabio D, Consonni V, Mauri A, Tetko IV, Todeschini R (2014) Prediction of acute aquatic toxicity toward daphnia magna by using the ga-knn method. Alternatives to Laboratory Animals 42(1):31–41, DOI 10.1177/026119291404200106, URL https://doi.org/10.1177/026119291404200106, pMID: 24773486
5. Castelli M, Trujillo L, Vanneschi L, Silva S, Z-Flores E, Legrand P (2015) Geometric semantic genetic programming with local search. In: Proceedings of the Genetic and Evolutionary Computation Conference, GECCO 2015, Madrid, Spain, July 11-15, 2015, ACM, pp 999–1006, DOI 10.1145/2739480.2754795, URL https://doi.org/10.1145/2739480.2754795
6. Cava WL, Helmuth T, Spector L, Moore JH (2018) A probabilistic and multi-objective analysis of lexicase selection and ε-lexicase selection. Evolutionary computation pp 1–28
7. Dignum S, Poli R (2008) Operator equalisation and bloat free gp. In: European Conference on Genetic Programming, Springer, pp 110–121
8. Efron B, Hastie T, Johnstone I, Tibshirani R, et al. (2004) Least angle regression. The Annals of statistics 32(2):407–499
9. Freund Y, Schapire RE (1995) A decision-theoretic generalization of on-line learning and an application to boosting. In: Computational Learning Theory, Second European Conference, EuroCOLT '95, Barcelona, Spain, March 13-15, 1995, Proceedings, Springer, Lecture Notes in Computer Science, vol 904, pp 23–37, DOI 10.1007/3-540-59119-2_166, URL https://doi.org/10.1007/3-540-59119-2_166
10. Gandomi AH, Alavi AH (2012) A new multi-gene genetic programming approach to nonlinear system modeling. part i: materials and structural engineering problems. Neural Computing and Applications 21(1):171–187, DOI 10.1007/s00521-011-0734-z, URL https://doi.org/10.1007/s00521-011-0734-z

11. Gerules G, Janikow C (2016) A survey of modularity in genetic programming. In: 2016 IEEE Congress on Evolutionary Computation (CEC), pp 5034–5043, DOI 10.1109/CEC.2016.7748328
12. Hastie T, Tibshirani R, Friedman J (2009) The elements of statistical learning: data mining, inference, and prediction. Springer Science & Business Media
13. Hinton GE (1990) Connectionist learning procedures. In: Machine learning, Elsevier, pp 555–610
14. Keijzer M (2003) Improving symbolic regression with interval arithmetic and linear scaling. In: European Conference on Genetic Programming, Springer, pp 70–82
15. Keijzer M (2004) Scaled symbolic regression. Genetic Programming and Evolvable Machines 5(3):259–269
16. Krawiec K, Liskowski P (2015) Automatic derivation of search objectives for test-based genetic programming. In: European Conference on Genetic Programming, Springer, pp 53–65
17. Krawiec K, O'Reilly UM (2014) Behavioral programming: a broader and more detailed take on semantic gp. In: Proceedings of the 2014 Annual Conference on Genetic and Evolutionary Computation, ACM, pp 935–942
18. Krawiec K, Pawlak T (2013) Locally geometric semantic crossover: a study on the roles of semantics and homology in recombination operators. Genetic Programming and Evolvable Machines 14(1):31–63, DOI 10.1007/s10710-012-9172-7, URL https://doi.org/10.1007/s10710-012-9172-7
19. La Cava W, Spector L, Danai K (2016) Epsilon-lexicase selection for regression. In: Proceedings of the Genetic and Evolutionary Computation Conference 2016, ACM, pp 741–748
20. Liskowski P, Krawiec K (2017) Online discovery of search objectives for test-based problems. Evolutionary Computation 25(3):375–406, DOI 10.1162/evco_a_00179, URL https://doi.org/10.1162/evco_a_00179, pMID: 26953882
21. Luke S, Panait L (2006) A comparison of bloat control methods for genetic programming. Evolutionary Computation 14(3):309–344
22. McPhee NF, Ohs B, Hutchison T (2008) Semantic Building Blocks in Genetic Programming. In: Genetic Programming, vol 4971, Springer Berlin Heidelberg, Berlin, Heidelberg, pp 134–145, DOI 10.1007/978-3-540-78671-9-12, URL http://link.springer.com/10.1007/978-3-540-78671-9_12
23. Medernach D, Fitzgerald J, Azad RMA, Ryan C (2015) Wave: A genetic programming approach to divide and conquer. In: Proceedings of the Companion Publication of the 2015 Annual Conference on Genetic and Evolutionary Computation, ACM, New York, NY, USA, GECCO Companion '15, pp 1435–1436, DOI 10.1145/2739482.2764659, URL http://doi.acm.org/10.1145/2739482.2764659
24. Medernach D, Fitzgerald J, Azad RMA, Ryan C (2016) A new wave: A dynamic approach to genetic programming. In: Proceedings of the Genetic and Evolutionary Computation Conference 2016, ACM, New York,

NY, USA, GECCO '16, pp 757–764, DOI 10.1145/2908812.2908857, URL http://doi.acm.org/10.1145/2908812.2908857
25. Moraglio A, Krawiec K, Johnson CG (2012) Geometric semantic genetic programming. In: Parallel Problem Solving from Nature - PPSN XII, Springer Berlin Heidelberg, Berlin, Heidelberg, pp 21–31
26. Nguyen QU, Chu TH (2020) Semantic approximation for reducing code bloat in Genetic Programming. Swarm and Evolutionary Computation 58:100729, DOI 10.1016/j.swevo.2020.100729, URL https://www.sciencedirect.com/science/article/pii/S2210650220303825
27. Nicolau M, Agapitos A (2018) On the effect of function set to the generalisation of symbolic regression models. In: Proceedings of the Genetic and Evolutionary Computation Conference Companion, ACM, New York, NY, USA, GECCO '18, pp 272–273, DOI 10.1145/3205651.3205773, URL http://doi.acm.org/10.1145/3205651.3205773
28. Oliveira LOV, Otero FE, Pappa GL, Albinati J (2015) Sequential symbolic regression with genetic programming. In: Genetic Programming Theory and Practice XII, Springer, pp 73–90
29. Orzechowski P, Cava WL, Moore JH (2018) Where are we now?: a large benchmark study of recent symbolic regression methods. In: Proceedings of the Genetic and Evolutionary Computation Conference, GECCO 2018, Kyoto, Japan, July 15-19, 2018, pp 1183–1190, DOI 10.1145/3205455.3205539, URL https://doi.org/10.1145/3205455.3205539
30. Otero FEB, Johnson CG (2013) Automated problem decomposition for the boolean domain with genetic programming. In: Genetic Programming, Springer Berlin Heidelberg, Berlin, Heidelberg, pp 169–180
31. O'Neill M (2016-3) Semantic methods in genetic programming. Genetic programming and evolvable machines 17(1):3,4
32. Pawlak TP, Wieloch B, Krawiec K (2014) Semantic backpropagation for designing search operators in genetic programming. IEEE Transactions on Evolutionary Computation 19(3):326–340
33. Pawlak TP, Wieloch B, Krawiec K (2015) Review and comparative analysis of geometric semantic crossovers. Genetic Programming and Evolvable Machines 16(3):351–386, DOI 10.1007/s10710-014-9239-8, URL https://doi.org/10.1007/s10710-014-9239-8
34. Pedregosa F, Varoquaux G, Gramfort A, Michel V, Thirion B, Grisel O, Blondel M, Prettenhofer P, Weiss R, Dubourg V, Vanderplas J, Passos A, Cournapeau D, Brucher M, Perrot M, Duchesnay E (2011) Scikit-learn: Machine learning in Python. Journal of Machine Learning Research 12:2825–2830
35. Platt JC (1999) Probabilistic outputs for support vector machines and comparisons to regularized likelihood methods. In: ADVANCES IN LARGE MARGIN CLASSIFIERS, MIT Press, pp 61–74
36. Poli R, Langdon WB (1998) Schema theory for genetic programming with one-point crossover and point mutation. Evolutionary Computation 6(3):231–252

37. Ruberto S, Vanneschi L, Castelli M, Silva S (2014) Esagp – a semantic gp framework based on alignment in the error space. In: Genetic Programming, Springer Berlin Heidelberg, Berlin, Heidelberg, pp 150–161
38. Ruberto S, Vanneschi L, Castelli M (2019) Genetic programming with semantic equivalence classes. Swarm and Evolutionary Computation 44:453 – 469, DOI https://doi.org/10.1016/j.swevo.2018.06.001, URL http://www.sciencedirect.com/science/article/pii/S2210650216300384
39. Ruberto S, Terragni V, Moore JH (2020) Image feature learning with a genetic programming autoencoder. In: Proceedings of the Genetic and Evolutionary Computation Conference, GECCO 2020, Cancun, Mexico, July 8-12, 2020, pp 245–246, DOI 10.1145/3377929.3389981
40. Ruberto S, Terragni V, Moore JH (2020) Image Feature Learning with Genetic Programming. In: Parallel Problem Solving from Nature – PPSN XVI, Springer International Publishing, Cham, Lecture Notes in Computer Science, pp 63–78, DOI 10.1007/978-3-030-58115-2_5
41. Ruberto S, Terragni V, Moore JH (2020) SGP-DT: Semantic Genetic Programming Based on Dynamic Targets. In: Proceedings of the 23rd European Conference on Genetic Programming, EuroGP 2020, Springer, Lecture Notes in Computer Science, vol 12101, pp 167–183, DOI 10.1007/978-3-030-44094-7_11, URL https://doi.org/10.1007/978-3-030-44094-7_11
42. Ruberto S, Terragni V, Moore JH (2020) Sgp-dt: Towards effective symbolic regression with a semantic gp approach based on dynamic targets. In: Proceedings of the Genetic and Evolutionary Computation Conference (Hot Off the Press track), GECCO 2020, Cancun, Mexico, July 8-12, 2020, pp 25–26, DOI 10.1145/3377929.3397486
43. Ruberto S, Terragni V, Moore JH (2021) Towards effective gp multi-class classification based on dynamic targets. In: Proceedings of the 2021 Genetic and Evolutionary Computation Conference, ACM, DOI 10.1145/3449639.3459324
44. Silva S, Costa E (2009) Dynamic limits for bloat control in genetic programming and a review of past and current bloat theories. Genetic Programming and Evolvable Machines 10(2):141–179
45. Silva S, Dignum S, Vanneschi L (2012) Operator equalisation for bloat free genetic programming and a survey of bloat control methods. Genetic Programming and Evolvable Machines 13(2):197–238
46. Tufekci P (2014) Prediction of full load electrical power output of a base load operated combined cycle power plant using machine learning methods. International Journal of Electrical Power and Energy Systems 60:126–140, DOI https://doi.org/10.1016/j.ijepes.2014.02.027, URL https://www.sciencedirect.com/science/article/pii/S0142061514000908
47. Vanneschi L, Castelli M, Silva S (2014) A survey of semantic methods in genetic programming. Genetic Programming and Evolvable Machines 15(2):195–214, DOI 10.1007/s10710-013-9210-0, URL https://doi.org/10.1007/s10710-013-9210-0

48. Vanneschi L, Castelli M, Scott K, Trujillo L (2019) Alignment-based genetic programming for real life applications. Swarm and Evolutionary Computation 44:840 – 851, DOI https://doi.org/10.1016/j.swevo.2018.09.006, URL http://www.sciencedirect.com/science/article/pii/S2210650218300208
49. White DR, Mcdermott J, Castelli M, Manzoni L, Goldman BW, Kronberger G, Jaśkowski W, O'Reilly UM, Luke S (2013) Better gp benchmarks: community survey results and proposals. Genetic Programming and Evolvable Machines 14(1):3–29
50. Yeh IC, Hsu TK (2018) Building real estate valuation models with comparative approach through case-based reasoning. Applied Soft Computing 65:260–271, DOI https://doi.org/10.1016/j.asoc.2018.01.029, URL https://www.sciencedirect.com/science/article/pii/S1568494618300358