Genetic Programming and Evolvable Machines (2024) 25:9
https://doi.org/10.1007/s10710-024-09482-6

An ensemble learning interpretation of geometric semantic genetic programming

Grant Dick[1]

Received: 22 June 2023 / Revised: 20 January 2024 / Accepted: 20 January 2024

Abstract
Geometric semantic genetic programming (GSGP) is a variant of genetic programming (GP) that directly searches the semantic space of programs to produce candidate solutions. GSGP has shown considerable success in improving the performance of GP in terms of program correctness, however this comes at the expense of exponential program growth. Subsequent attempts to address this growth have not fully-exploited the fact that GSGP searches by producing linear combinations of existing solutions. This paper examines this property of GSGP and frames the method as an ensemble learning approach by redefining mutation and crossover as examples of boosting and stacking, respectively. The ensemble interpretation allows for simple integration of regularisation techniques that significantly reduce the size of the resultant programs. Additionally, this paper examines the quality of parse tree base learners within this ensemble learning interpretation of GSGP and suggests that future research could substantially improve the quality of GSGP by examining more effective initialisation techniques. The resulting ensemble learning interpretation leads to variants of GSGP that substantially improve upon the performance of traditional GSGP in regression contexts, and produce a method that frequently outperforms gradient boosting.

Keywords Genetic programming · Boosting · Base learner · Geometric interpretation

1 Introduction

Genetic programming (GP) is an evolutionary computation (EC) approach to produce solutions with unbounded size and structure [1]. In its canonical form, GP conducts search via the syntax of solutions, and this has been shown to produce

✉ Grant Dick
grant.dick@otago.ac.nz

[1] Department of Information Science, University of Otago, Dunedin, New Zealand

potential problems with *locality*, where small changes in syntax often result in large changes in solution behaviour [2]. It is argued that focus should be put on designing representations for GP that strengthen the connection between search and behaviour [3]. Additionally, GP (and EC in general) is often inefficient in its use of available feedback from the problem domain, often limiting the use of such information to fitness evaluation and resulting in search operators that are blind and inefficient. It is known that disaggregating error information, for example, can provide a richer guidance towards determining good search directions [4]. Additionally, a broad class of extensions to GP have been created that examine the *semantics* of program behaviour [5]. These semantic methods attempt to provide fine-grained information to guide not only the selection of good parent candidates, but extend its use into the crossover and mutation operators.

Perhaps the most well-known variant of these semantic methods is geometric semantic GP (GSGP) [6]. The approach of GSGP is to evolve trees through operators that explicitly result in affine combinations of parents. In the case of crossover, this has provable properties of simplifying search and ensuring that offspring are fitter than the worst parent. A significant drawback of GSGP is its tendency towards exponential program growth, which limits the practicality of the approach both in terms of comprehension and computation. While some subsequent research into GSGP has focused on addressing program size [7], much work in GSGP has focused on other areas, particularly in the area of finding ideal linear combinations of parents [8–10].

This paper develops an alternate interpretation of GSGP framed in the context of ensemble learning. Specifically, trees in GSGP are reinterpreted as sets of base models acting as an additive ensemble. New models are introduced into the ensemble through mutation and similarities are identified between mutation in GSGP and gradient boosting [11]. Crossover is positioned as a stacking operator attempting to find good combinations of base models. This paper shows that this interpretation preserves the semantic behaviour of GSGP while offering comparable solution size characteristics to a bloat-reducing variant of GSGP called GSGP-Red [7]. Establishing a strong link between GSGP and ensemble learning offers many benefits: first, there is a rich literature pertaining to ensemble learning theory that can be used to inform improvements in GSGP design; second, the geometric interpretation used by GSGP can also provide insights that may feed back into traditional ensemble learning research; third, identifying analogies between GSGP operators and those from ensemble learning allows researchers to decouple the behaviour of GSGP from an underlying parse tree representation and more effectively examine the behaviour of novel search operators and base learner representations (as done in this paper); finally, ideas related to ensemble learning that have not been rigorously explored in GSGP, such as ensemble regularisation, can be relatively easily adopted to address limitations in GSGP such as the tendency towards exponential growth in model size.

With a relationship between GSGP and ensemble learning clearly defined, this paper extends the basic GSGP concept by introducing regularisation in the form of lasso regression for model coefficients [12]. Lasso regression can be viewed as an enhancer of the stacking function produced by standard GSGP crossover and allows a larger set of linear combinations of models to be explored over the affine combinations that are permitted by standard GSGP. Results over several benchmarks suggest that the incorporation of lasso to learn more effective stacking functions leads to both fitter and more compact solutions. Following the exploration of an improved stacking function learner, the role of the base learner in GSGP (traditionally, a parse tree generator) is assessed, with alternative base model generation techniques proposed. Results over several benchmarks suggest that the typical parse tree generator used for random solutions and mutations in GSGP is not well-behaved and spends considerable time exploring regions of the search space with low fitness. Instead, more informed base learners that acknowledge the residuals of the other models in the ensemble are required, such as those from classical decision tree literature. With these extensions from traditional ensemble learning factored, the resulting *parallel recombinative boosting* (PRB) methods are shown to outperform traditional GSGP by a significant margin in both fitness and model size. In addition, PRB with an appropriate base learner and lasso regularisation is shown to be frequently more effective than equivalent gradient boosting approaches.

The rest of the paper is structured as follows: Sect. 2 examines existing work related to semantics and GSGP; Sect. 3 frames the GSGP approach as an ensemble learning method and demonstrates the functional equivalence of this interpretation; Sect. 4 examines how regularisation techniques can be introduced into an ensemble-framed GSGP method to substantially improve performance; Sect. 5 assesses the suitability of existing parse tree representations in GSGP when it is interpreted as an ensemble learning method; Sect. 6 consolidates the work done in previous sections and examines the newly-developed methods against gradient boosting methods; finally, Sect. 7 concludes the paper and suggests areas of future work.

2 Semantic methods in GP

A common property of genetic programming is the adoption of a solution representation that decouples search from program behaviour. For example, the standard crossover and mutation operations on a parse tree representation operate in a manner that is blind to the behaviour of the corresponding program. Consider the mutation operator on the parse tree shown in Fig. 1: this very small mutation (of a constant node from a value of one to a new node with a value of zero) results in a complete change in behaviour of the tree. The lack of correlation between the size of change in *syntax* against the *behaviour* of the tree is measured in terms of locality [2, 13, 14]. Problems with locality of search operators in GP has been noted in most representations, including parse trees, grammatical evolution and Cartesian representations [14–16].

Locality issues in GP can be managed in several ways. An increasingly common approach is to integrate information about a program's behaviour (i.e., its *semantics*) into search operators. These *semantic methods* have allowed substantial improvement in GP performance [5] but often require substantial work and modification of GP to map between the representation and semantic spaces.

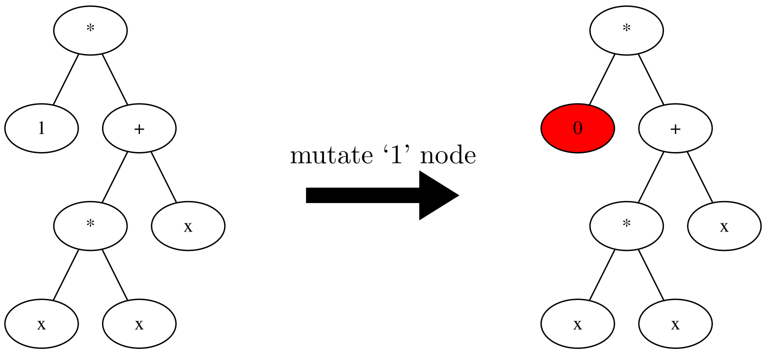

Fig. 1 Example of locality effects in GP. Here the original tree has a functional behaviour of $f(x) = x^2 + x$. Following a single mutation of one node, the resulting behaviour shifts to $f(x) = 0$

2.1 Geometric semantic genetic programming

Geometric semantic genetic programming was introduced to reduce the complexity of integrating semantics into GP search [6]. The solution proposed by GSGP is elegant and simple: consider the semantics of programs as a vector space and allow search to work directly on this vector space. Crossover in GSGP, dubbed *SGXE*, takes two parent functions $A, B : \mathbb{R}^n \rightarrow \mathbb{R}$ and returns a real function O that produces an affine combination of the parents' semantics:

$$O = p \cdot A + (1 - p) \cdot B \tag{1}$$

where p is a uniform random variable in [0, 1). As defined in [6], when the fitness measure and metric in the semantic space are both Euclidean distance, crossover in GSGP is weakly progressive and has the useful property of ensuring that offspring are as least as fit as their worst parent (see Fig. 2) [17].

Mutation in GSGP, dubbed *SGM*, produces a new function $O : \mathbb{R}^n \rightarrow \mathbb{R}$ by generating two new random real functions, R_1 and R_2, and adding their weighted difference to parent function A:

$$O = A + ms \cdot \left(R_1 - R_2\right) \tag{2}$$

where ms is typically a user-defined parameter in [0, 1). *SGM* produces offspring within close proximity to parent A, however it does not guarantee that offspring will be fitter (see Fig. 3).

With crossover and mutation now defined in the semantic space, there needs to be a way for GSGP to construct a representation for offspring that will map to their corresponding semantics. An important feature of GSGP is that "syntax does not matter" [18], so the way that GSGP enables this is through templates that encapsulate the entire representation of each parent into the offspring. This simple approach requires minimal modification to existing GP (as selection and evaluation are largely unchanged), but has the undesirable property that offspring

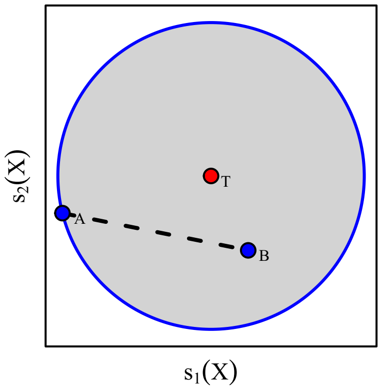

Fig. 2 Geometric semantic crossover (SGXE). Each dimension represents the span of possible behaviour in a single component of problem semantics. The target semantics are represented by the red point, and two parents (A and B) will undergo crossover to produce offspring. All potential offspring from crossover will be on the line formed by A and B. The shaded area represents the space of solutions that are fitter than the worst parent (A): note that all offspring are guaranteed to be fitter than the worst parent

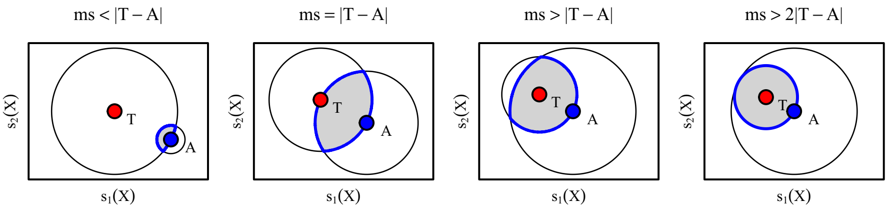

Fig. 3 Geometric semantic mutation (SGM). SGM will produce offspring in a ball centred around parent A. The circle centred on the target semantics T represents the space of solutions fitter than parent A. The circle around A represents the space of solutions produced by mutating A. The shaded interval represents the spaces in which offspring produced by mutation will be fitter than parent A. Multiple conditions are presented for where *ms* (the mutation scale) is smaller than, equal to, or larger than the distance between the target semantics and parent A

size tends to grow exponentially under crossover. With mutation, program growth is linear but still results in large solutions.

Note that this work focuses on GSGP and its use in symbolic regression. As shown in the remainder of this section, regression appears to be a large focus of the use of GSGP, so a restriction to examine regression is warranted. It is likely that the findings of this work are not limited to regression and indeed apply to GSGP in its broader applications (e.g., binary function generation, classification, etc.).

2.2 Related work in GSGP

Work in advancing GSGP has largely focused on two areas: optimisation of the coefficients *p* and *ms*, and approaches to manage and/or reduce tree size. Post-processing of solutions through algebraic simplification can reduce program size, as can representing solutions as anonymous functions and defining calls to parent functions instead of direct embedding of copies [6, 19]. In a somewhat related manner, offspring can be implemented as pointers to their parents and executed recursively [20]. Both the functional and pointer-based solutions reduce storage requirements of offspring to linear complexity, but the execution complexity of offspring remains exponential: the pointer-based approach solves this problem during training by caching the semantics of parents and using these cached results to compute the semantics of offspring. Later, the GSGP-Red approach used hashing of trees to identify cases where the same tree had been embedded into a solution multiple times, and used this concept to combine these instances into a single one with a shared coefficient [7]. This had the effect of substantially reducing tree size without compromising solution quality.

Alongside work to manage program size, several researchers have focused on the parameters used within mutation and crossover in GSGP. The original GSGP defined crossover as an affine combination of parents, and used a fixed weighting parameter *ms* in mutation. Through its geometric interpretation, researchers identified that optimal values for *ms* could be identified through ordinary least squares [8]. Others have noted that the affine crossover operator could be replaced with a linear combination of parents, the parameters of which could also be optimised [9, 10].

Alongside these approaches to improve the efficiency of GSGP, other researchers have considered alternative tree-base representations that "approximate" the geometric operators and permit a more considered approach to tree and population initialisation. The partial semantics of a solution of a given subtree can be computed through *semantic backpropagation*, and these semantics can be used to guide the construction or selection of a new subtree with desired semantics to replace the old one [21, 22]. The same researchers also demonstrated that proper convergence of the population in GSGP requires that the target semantics reside within the convex hull defined by the collective semantics of the population. They use this idea to construct alternative initial population construction techniques [23, 24]. Also related to population initialisation and mutation, other work has observed that interval arithmetic can be used to ensure that all tree generation results in solutions with valid semantics [25].

Finally, some interesting work has explored the idea that GSGP is "overkill" [26]. Specifically, it is recognised that GSGP produces offspring that are linear combinations of parents, and rather than construct these iteratively through a stochastic process, an optimal composition of the initial population can be determined in a single step. The resulting models are smaller and faster to construct than those produced by GSGP and offer equivalent generalisation.[1]

[1] It is beyond the scope of this paper, but personal experience with the "overkill" approach is that it is sensitive to noise and heavily dependent upon the quality of the initial population construction. It will fail to generalise well if low noise or sufficient initialisation cannot be achieved and this limits its practical application. Nonetheless, the paper offers excellent insights into the behaviour of GSGP and helped to inform some of the work done in this paper.

3 Parallel recombinative boosting (PRB)

Despite the vast array of approaches and foci in the work examined in the previous section, they all have one theme in common: the underlying representation used is ultimately a monolithic tree. However, insights hinted to in many of these papers suggest an alternative interpretation of GSGP as an ensemble learning method. As shown in later sections of this paper, an ensemble interpretation of GSGP affords several avenues of exploration that may help to improve the performance of GSGP, but also provide useful insights that may help drive future directions of GSGP research.

Ensemble learning has a long and well-established history in machine learning, with many useful practical theories and applications [27]. Ensemble learning theory has also been useful in advancing the capability of other fields: for example, the concepts of dropout and residual networks, used heavily in modern deep learning approaches, have clear connections to ensemble learning theory [28, 29]. While varied definitions of ensemble learning exist in both machine learning and evolutionary computation literature, three fairly standard families of ensemble learning exist: bagging, boosting, and stacking. Bagging (bootstrap aggregation) takes multiple models learned on resampled instances from the same data set and aggregates them to reduce model variance without compromising bias [30]. Boosting operates by taking a "weak" learner and iteratively applying that learner to the same data set, but rather than learning the entire response function each time, it fits the *residuals* of the previously learned models [11]. Lastly stacking attempts to find a parent model that effectively recombines the outputs of multiple individuals models within the ensemble [31].

An ensemble interpretation of GSGP begins by examining the crossover operator. The authors of GSGP-Red performed a similar step in their research [7], and the "overkill" interpretation of GSGP raised a similar insight [26]. Consider an initial population containing three solutions: A, B, and C. In the next generation, two offspring are produced through crossover, one combining A and B and the other combines A and C:

$$\begin{aligned} O_1 &= p_1 \cdot A + (1 - p_1) \cdot B \\ O_2 &= p_2 \cdot A + (1 - p_2) \cdot C \end{aligned}$$

and in the generation after that, a third offspring is created by crossing O_1 and O_2 as parents:

$$O_3 = p_3 \cdot O_1 + (1 - p_3) \cdot O_2.$$

Clearly, the expressions for O_1 and O_2 can be substituted into O_3:

$$\begin{aligned} O_3 &= p_3 \cdot \left(p_1 \cdot A + (1 - p_1) \cdot B\right) + (1 - p_3) \cdot \left(p_2 \cdot A + (1 - p_2) \cdot C\right) \\ &= \left(p_3 \cdot p_1 + \left(1 - p_3\right) \cdot p_2\right) \cdot A + p_3 \cdot \left(1 - p_1\right) \cdot B + \left(1 - p_3\right) \cdot \left(1 - p_2\right) \cdot C \end{aligned}$$

resulting in O_3 being a linear combination of three parents A, B, and C. A reasonable interpretation of crossover, therefore, is that it is searching for high-performing linear combinations of existing solutions created either in the initial population or through subsequent mutation. A reasonable interpretation of this is that crossover in GSGP is equivalent to learning a *stacking* operator in ensemble learning.

Moving now to the mutation operator, consider a squared error loss function as typically used in regression:

$$L(T, O_m) = \frac{1}{n}\sum_{i=1}^{n}\left(T_i - O_{mi}\right)^2. \tag{3}$$

where T and O_m are the semantics of the target and mutant offspring, and T_i and O_{mi} represent the ith semantic case of the target and mutant. In order for a mutation resulting in O_m to be *effective*, it must improve the fitness (i.e., it must reduce the error in $L(T, O_m)$). Substituting Eq. 2 into the loss function reveals:

$$L(T, O_m) = \frac{1}{n}\sum_{i=1}^{n}\left(T_i - \left(A_i + ms \cdot \left(R_{1i} - R_{2i}\right)\right)\right)^2 \tag{4}$$

$$= \frac{1}{n}\sum_{i=1}^{n}\left(\left(T_i - A_i\right) + ms \cdot \left(R_{2i} - R_{1i}\right)\right)^2 \tag{5}$$

where $\left(T_i - A_i\right)$ is defined as the *residuals* of A on the ith semantic. Therefore, an effective mutation works to minimise the residuals of the parent, acting in the same manner as adding a new model into the ensemble of a gradient boosting method.

Based on these observations, it is reasonable to interpret GSGP as a ensemble learning method performing a combination of boosting and stacking. This paper refers to this ensemble approach as *parallel recombinative boosting* (PRB) to clearly distinguish it from an explicit GSGP representation. To more effectively realise this interpretation, a new representation for solutions is required. This paper adopts a very simple model as follows: each individual contains a set of models, and associated with each model is a weight. The weight corresponding to a model M_j within the set of an individual I is denoted as $w_I(M_j)$. Using this representation, the semantics of an individual I is simply the weighted sum of the semantics of its constituent models:

$$s(I) = \sum_{M_j \in I} w_I(M_j) \cdot M_j \tag{6}$$

An individual in the initial population contains a set with a single model and a corresponding weight of 1. Individuals in subsequent generations will contain sets with multiple models, some of which have been inherited from parents in previous generations. It is therefore possible for a given model M_j to be contained in the sets of multiple individuals, each time with a different weight $w_I(M_j)$. Crossover is then performed as a *weighted set union* operation, as demonstrated in Algorithm 1. The result of this crossover operation is a new individual whose semantics are an linear combination of its parents inline with the behaviour defined in Eq. 1. Mutation in this representation is a simple weighted set insertion operation.

Algorithm 1 Set-based SGXE: perform geometric semantic crossover using a set-based representation.

```
Require: $A$, $B$: two parent individuals;
Ensure: $O$: an offspring individual that forms an affine combination of $A$ and $B$
 1: $p \leftarrow U(0,1)$                                  ▷ generate a uniform random value in $[0,1)$
 2: $O \leftarrow A \bigcup B$
 3: for $i \leftarrow 1..|O|$ do
 4:     if $O_i \in A$ & $O_i \in B$ then                  ▷ $O_i$ present in both parents
 5:         $w_O(O_i) \leftarrow p \cdot w_A(O_i) + (1-p) \cdot w_B(O_i)$
 6:     else if $O_i \in A$ then                           ▷ $O_i$ only in $A$
 7:         $w_O(O_i) \leftarrow p \cdot w_A(O_i)$
 8:     else                                               ▷ $O_i$ must be in $B$
 9:         $w_O(O_i) \leftarrow (1-p) \cdot w_B(O_i)$
10:     end if
11: end for
12: return $O$
```

Algorithm 2 Set-based SGM: perform geometric semantic mutation using a set-based representation.

```
Require: $A$: a parent individual; $ms$: the mutation scale;
Ensure: $O$: an offspring individual formed through mutation of $A$
 1: $R_1 \leftarrow$ newmodel ()                           ▷ newmodel constructs a new random model
 2: $R_2 \leftarrow$ newmodel ()
 3: $w_O(R_1) \leftarrow ms$
 4: $w_O(R_2) \leftarrow -ms$
 5: $O \leftarrow A \bigcup \{R_1, R_2\}$
 6: return $O$
```

To ensure that the PRB representation offers similar behaviour to previous methods, the two representations were examined using a set of benchmark data sets from previous work [7, 25].[2] These problems are outlined in Table 1. Following previous work, each problem was run 30 times in the form of six rounds of five-fold cross validation [7]. Experimental parameter settings were drawn from previous work [25] and are outlined in Table 2. To ensure valid individuals were created throughout a run, both GSGP and PRB used safe initialisation incorporating interval arithmetic [25]. Finally, both standard GSGP and GSGP-Red variants were compared against PRB, as it was expected that PRB would demonstrate identical performance to GSGP-Red. Comparisons were made in terms of both fitness and size: size was measured as root-relative mean squared error (RRSE) of the target t against model predictions y[3]:

[2] The code used to run these experiments is available online: https://github.com/grantdick/.

[3] RRSE is essentially the normalised root-mean-squared error: an RRSE of 1 indicates an error that would be produced by a model performing no better than predicting the mean of the training data.

Table 1 Benchmark problems used in this paper

Problem	#Inputs	#Instances	References
Airfoil	6	1503	[32]
Boston housing[a]	13	506	[33, 34]
CCN	123	1994	[7]
CCUN	125	1994	[7]
Concrete	8	1030	[35]
Energy cooling	8	768	[36]
Energy heating	8	768	[36]
Parkinsons	19	5876	[37]
Tower	25	4999	[38]
Wine quality (Red)	11	1599	[39]
Wine quality (White)	11	4898	[39]
Yacht	6	308	[40]

[a]Previous work used the ‘PPB’ data set: experience with that data set, and its related data sets, suggests that it is not suited to regression modelling, so it was replaced with the Boston Housing data set

Table 2 Parameter settings used in this paper for GSGP and PRB variants

Parameter	Setting
Population size	200
Generations	250
Initialisation	Ramped Half-and-Half (height: 2–6)
Crossover prob	0.3
Mutation prob	0.7
Selection	Tournament (size: 3)
Elitism	1 (best of previous generation)
Function set	$+, -, \times, \div$, sin, cos, log, $\sqrt{}$
Terminal set	$x_1, x_2, \dots, x_p$, plus ephemeral random constants ($\mathfrak{R}$) drawn from $U(-1, 1)$

$$RRSE(t, y) = \sqrt{\frac{\sum_{i=1}^{N}(t_i - y_i)^2}{\sum_{i=1}^{N}(t_i - \bar{t})^2}} \tag{7}$$

while size was measured through the total number of nodes in all trees in an individual. To ensure a fair comparison, the size results for PRB were computed as if the ensemble was being converted back into a tree (i.e., for an ensemble of size n an additional $3 \cdot (2n - 1)$ nodes are added to the total node count to account for the required addition, multiplication, and constant nodes that would be required to turn the ensemble into a tree).

The fitness and size performance of GSGP and PRB are shown in Figs. 4 and 5, respectively. All results are presented as mean performance values with 95%

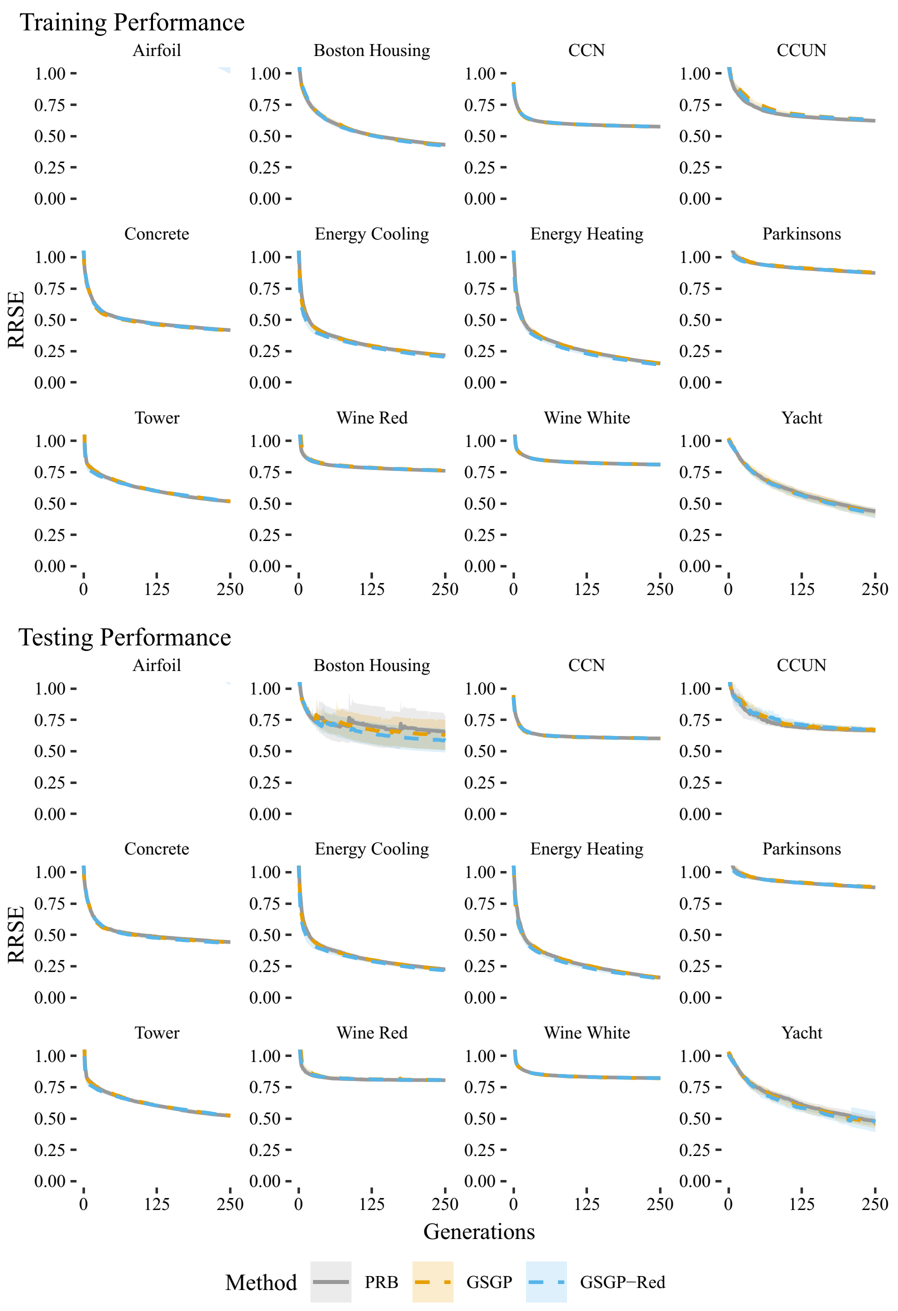

Fig. 4 Fitness performance of PRB. Note the high similarity in general behaviour of PRB relative to GSGP and GSGP-Red. Note also that all three methods perform very poorly on the Airfoil problem, with RRSE scores consistently greater than 1

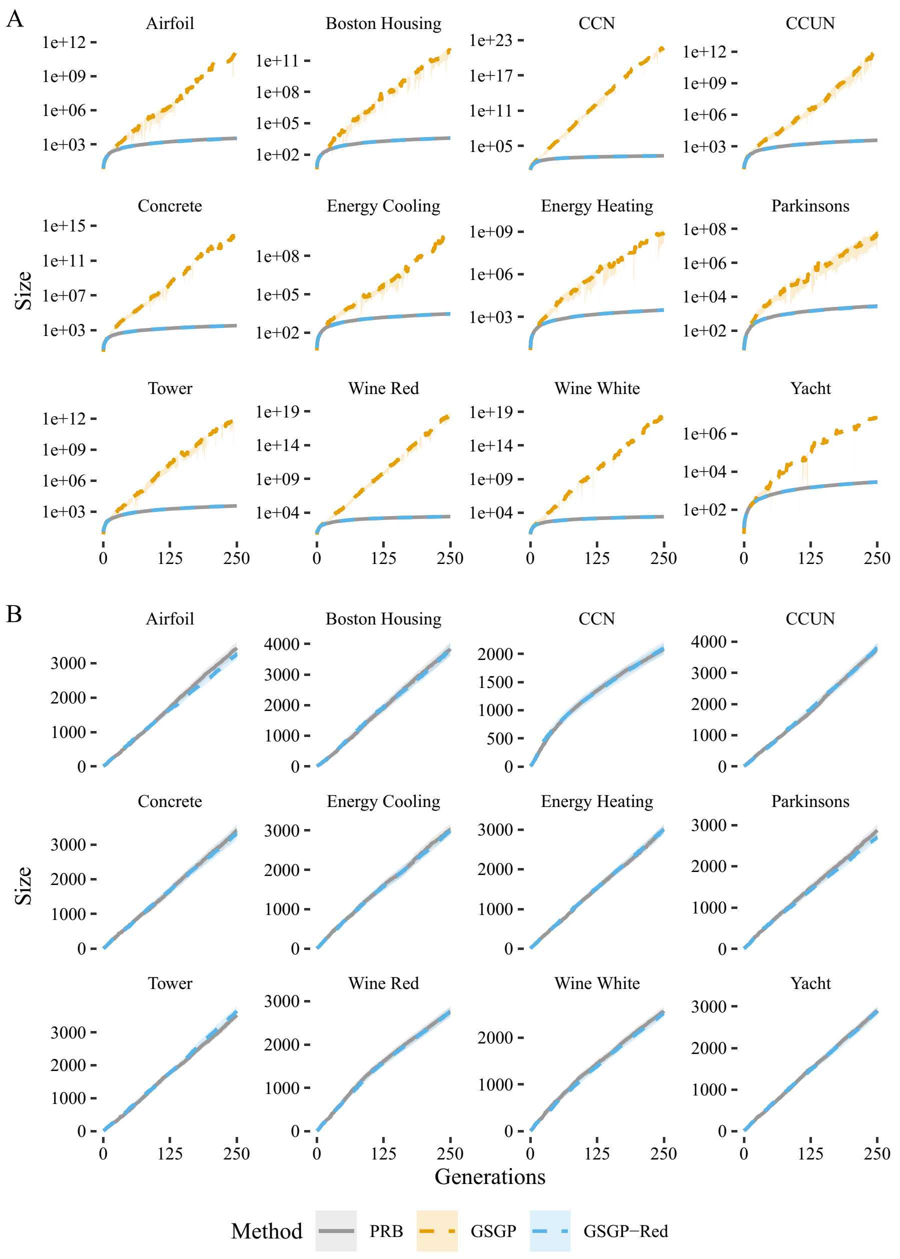

Fig. 5 Size performance of PRB relative to GSGP and GSGP-Red. The overhead of the additional nodes required to convert the set-based representation of PRB into a tree-based representation (as used by GSGP and GSGP-Red) has been factored into PRB's size. In plot group A, all three methods are compared, while plot group B removes GSGP to allow closer comparison of GSGP-Red and PRB

confidence intervals (computed from t-distributions). As shown, the fitness performance for all three methods is essentially identical in both training and testing contexts. Interestingly, none of the methods performed well on the Airfoil data sets, despite GSGP presenting a cone landscape that "GP can easily optimise" [18]. Reasons for this poor performance, and solutions to improving it, are explored in the remainder of this paper. The size performance observed for the three methods suggest that PRB is practically equivalent to GSGP-Red in behaviour with only minor differences of no statistical significance. Both methods are able to substantially reduce the size of models relative to standard GSGP.

The results presented here indicate that PRB behaves in a manner akin to GSGP-Red in terms of both size and fitness. From this base, the set-based additive ensemble model implemented by PRB allows several aspects of GSGP behaviour to be thoroughly explored. Specific to this paper, issues around identifying optimal parameter settings (in place of sampling *p* and fixed *ms*) can be easily examined, as can simple parameter regularisation to induce model shrinkage. Likewise, the efficiency of the base learner used in GSGP (parse trees, by default) can be easily examined.

4 Stacking functions, coefficient optimisation and regularisation

The implementation of PRB in the previous section used an identical setup for modelling coefficients to that used in the original GSGP paper [6]. As suggested earlier, the effect of crossover in GSGP is to learn a stacking function for a set of underlying base models. In GSGP, and likewise the implementation of PRB examined thus far, this stacking function learning is limited to affine combinations of the base models. However, another approach to learning the stacking function would be to optimise these coefficients directly, similar to that done in previous work [9, 10]. While this could be done by modifying the crossover operator, the set-based approach of PRB allows the stochastic search operator within crossover to be preserved, while allowing a new operator to be included. This new operator is straightforward: after a specified number of generations, the best individual in the population is selected, and lasso regression is applied to the coefficients of this model (i.e., L1-penalised regression is used to fit a linear model of individual model semantics against the target semantics) [12]. This serves two purposes: first, it allows all model coefficients to be optimised simultaneously, whereas previous work has had limited scope to achieve this. Second, the L1-penalty applied during regression encourages some coefficients towards zero: from this, any model within the set with a coefficient of zero can be removed from the solution. Therefore, we should reasonably expect the incorporation of lasso regression to learn the stacking function to produce solutions that are both fitter and more compact than those produced by GSGP-Red and PRB. Lasso regression requires a penalty weighting λ: this is established through five-fold cross-validation on a regularisation path estimated from the data, and does not require user intervention. Because the lasso regression is applied to only one individual in the population, it adds little overhead to the algorithm (and, in practice, this overhead should be more than compensated for by the overall reduction in program sizes within the population).

Two variants of PRB with embedded lasso regression were compared to GSGP-Red: the first variant applied lasso regression to the best individual in the population at the end of each generation (referred to as PRB + Lasso(1)), while the second variant only applies lasso regression to the first and last generations of the run (referred to her as PRB + Lasso(250)).

The same experimental setup used in the previous section was used to examine the effect of including lasso regularisation into PRB. Because of the equivalence of PRB and GSGP-Red, comparison with anything other then PRB on its own was not required. As before, both fitness and size were compared. The fitness results are shown in Fig. 6, while the size performance is shown in Fig. 7. The results are very encouraging: both fitness and size are substantially improved over PRB on its own (and hence GSGP). There does not appear to be a consistent end-of-run difference in fitness between using lasso at the end of each generation versus at the end of the run. However, applying lasso after each generation does typically lead to faster fitness improvement and, although not explicitly shown here, it was observed that lasso regression performed in each generation lead to an overall reduction in the total number of nodes processed during an entire run.

5 Effectiveness of base learner in PRB

An important requirement of boosting is that a weak learner needs to provide a fit that is (at least marginally) better than a null prediction (e.g., predicting the mean of the model residuals). In other words, to be effective in boosting, the base learner must produce models that exhibit an RRSE performance that is less than 1. This requirement of weak learning can be related back to mutation in GSGP (and therefore PRB) by examining Fig. 3: note that the shaded areas describe the regions where mutation produces an offspring that is fitter than its parent. Note also that it appears in all circumstances, the region of fitter offspring is outweighed by the region of weaker offspring: in other words, in the standard GSGP approach with parse trees, mutation spends more time exploring weaker solutions than promising regions. In the extreme case of Fig. 3, where the ratio of *ms* against the distance between the parent and target vector is 2:1, mutation leads to a weaker offspring 75% of the time. As problem size increases, this becomes potentially worse: for n components in the semantics, the probability of GSGP mutation generating weaker offspring is $\frac{2^n-1}{2^n}$. For GSGP (and therefore PRB) to be truly effective, the impact on base learner performance on mutation must be explored.

In Sect. 6, the overall effect of incorporating different base learners into GSGP is demonstrated. Prior to that, however, a more direct consideration of the behaviour of base learners is warranted. Given that a clear relationship between boosting and GSGP mutation was identified in Sect. 3, a simple boosting framework (forward step-wise additive boosting (FSAB) [27]) can be used to evaluate the performance of potential base learner approaches, and thus provide insights into how they will influence mutation behaviour in GSGP. This paper considers five potential base learners:

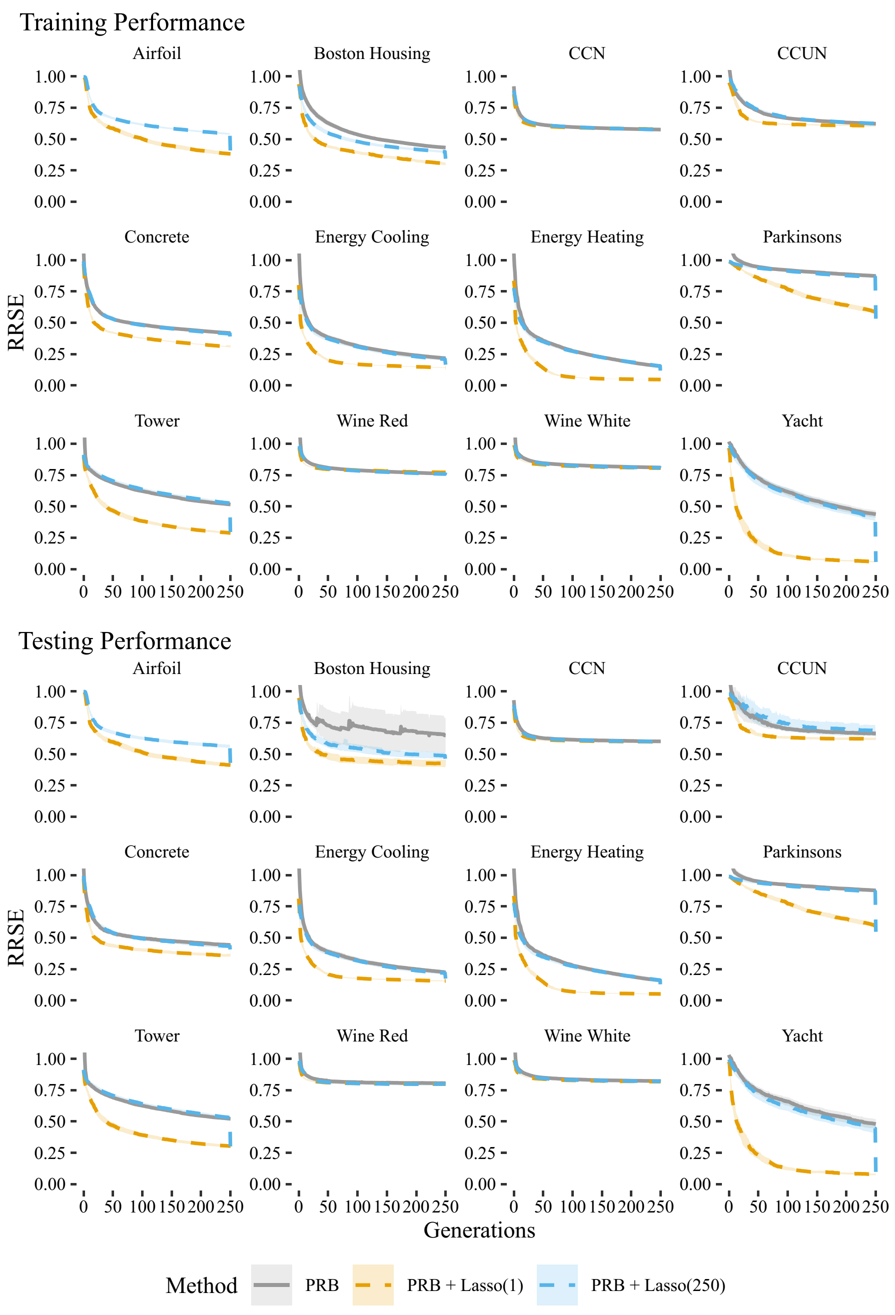

Fig. 6 Impact on fitness performance through using lasso regression for optimisation of coefficients in PRB

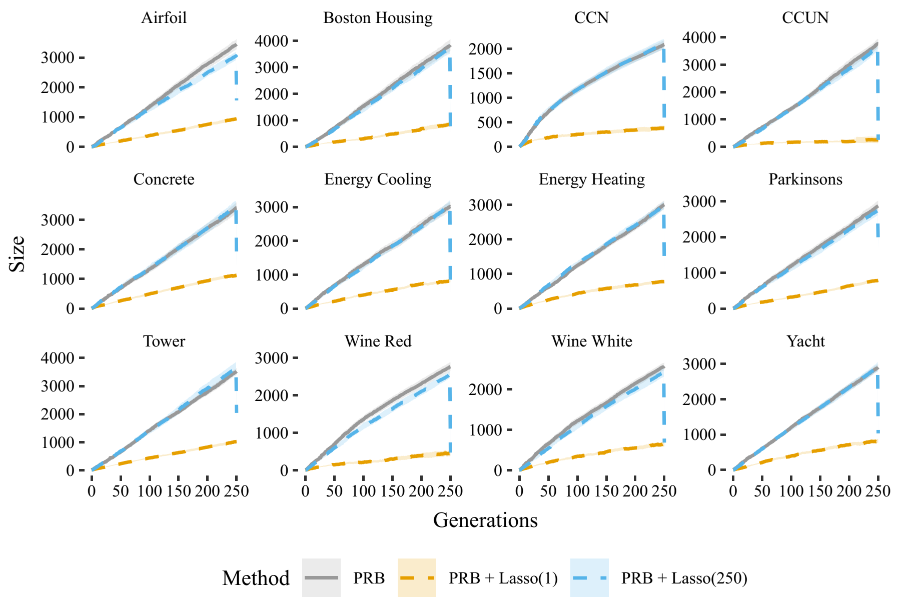

Fig. 7 Impact of size performance of PRB when using lasso regression to encourage model regularisation

1. *Standard parse tree generation*: this is the method used by default in GSGP.
2. *Parse tree + OLS*: essentially the standard parse tree generation with an additional attempt to fit the resulting tree to the target semantics by optimising *ms* using ordinary least squares as done in previous work [8].
3. *Greedy parse tree*: here, standard parse trees are repeatedly generated and rejected until one that improves fitness is found.[4]
4. *Random decision tree*: a non-greedy variant of CART that picks variables for splitting nodes, and the splitting points within said variables, at random.
5. *Stochastic CART*: essentially, the variant of CART that is used to generate trees in random forests through considering a random subset of variables at each splitting node [30].

Typical performance of an effective base learner (i.e., one exhibiting weak learning) would follow a given pattern:

1. The initial models learned would demonstrate fairly rapid adaptation to the problem and would rapidly reduce error.
2. The first few models would show relatively strong individual adaptation to the problem, as the residuals they fit would be relatively large. So if their fitness is

[4] While different in implementation, this greedy approach to generation has some overlapping concepts with the initialisation techniques that use a generate and reject approach to expand the convex hull of program semantics in the initial population in GSGP [23, 24].

measured individually, it would be reasonable to expect each model to have low error.
3. As the residuals get smaller and smaller, there is less useful information for individual models to fit, so they should exhibit relatively high errors.
4. Over time, there should be a gradual convergence towards a stable point of low error for the entire ensemble.

Confirming this pattern of a base learner can be done quite easily through graphing the evolution of the error of an FSAB ensemble and each individual base learner model over subsequent iterations on a given problem. For an effective base learner, the plot of training error of the ensemble should exhibit a fairly rapid monotonic decrease in error, while the error *n*th model should quickly converge on an RRSE close to 1.

To test the efficacy of these base learners, FSAB was used to create ensembles of 50 trees. For standard and greedy parse trees, a *ms* of 0.1 was used (as would be typical in GSGP/PRB), while parse tree + OLS determined this value automatically. Both CART base learner variants were permitted a maximum tree exploration depth of 3. As before, 30 runs were performs (six repeats of five-fold cross validation). The results of these results are shown in Fig. 8. The performance presented is mean RRSE with 95% confidence intervals. The results for standard parse trees as a base learner are particularly interesting and seem to confirm the theory that most of the effort of this type of base learner is spent exploring low-fitness regions of the solution space. The parse tree representation is unable to consistently produce models with RRSE less than one, so does not fulfill the requirement of weak learning needed for boosting. The application of OLS to tune the scaling parameter seems to do little to improve the situation, with the models produced consistently having RRSE scores greater than 1. However, the greedy parse tree approach, while computationally expensive, seems to show some promise and exhibits the necessary properties for a base learner under a boosting framework. Relating this to the scenario presented in Fig. 3, using the greedy parse tree base learner within GSGP mutation would ensures that offspring reside in the shaded regions. The results of the greedy learner suggest that the parse tree representation should not be completely rejected as base learners for GSGP mutation, but rather future work should explore better methods for parse tree generation. The two CART-based decision tree base learners demonstrate behaviour that is well-suited for boosting learning: indeed relating these base learners back to the scenario presented in Fig. 3, if used within SGM, these base learners would only produce within the shaded regions (i.e., an offspring would be guaranteed to be fitter than its parent).

The results presented here strongly suggest that non-greedy variants of parse tree base learners are unlikely to lead to effective implementation of mutation in GSGP. This is further examined in the next section, where different base learners are used within PRB to drive mutation.

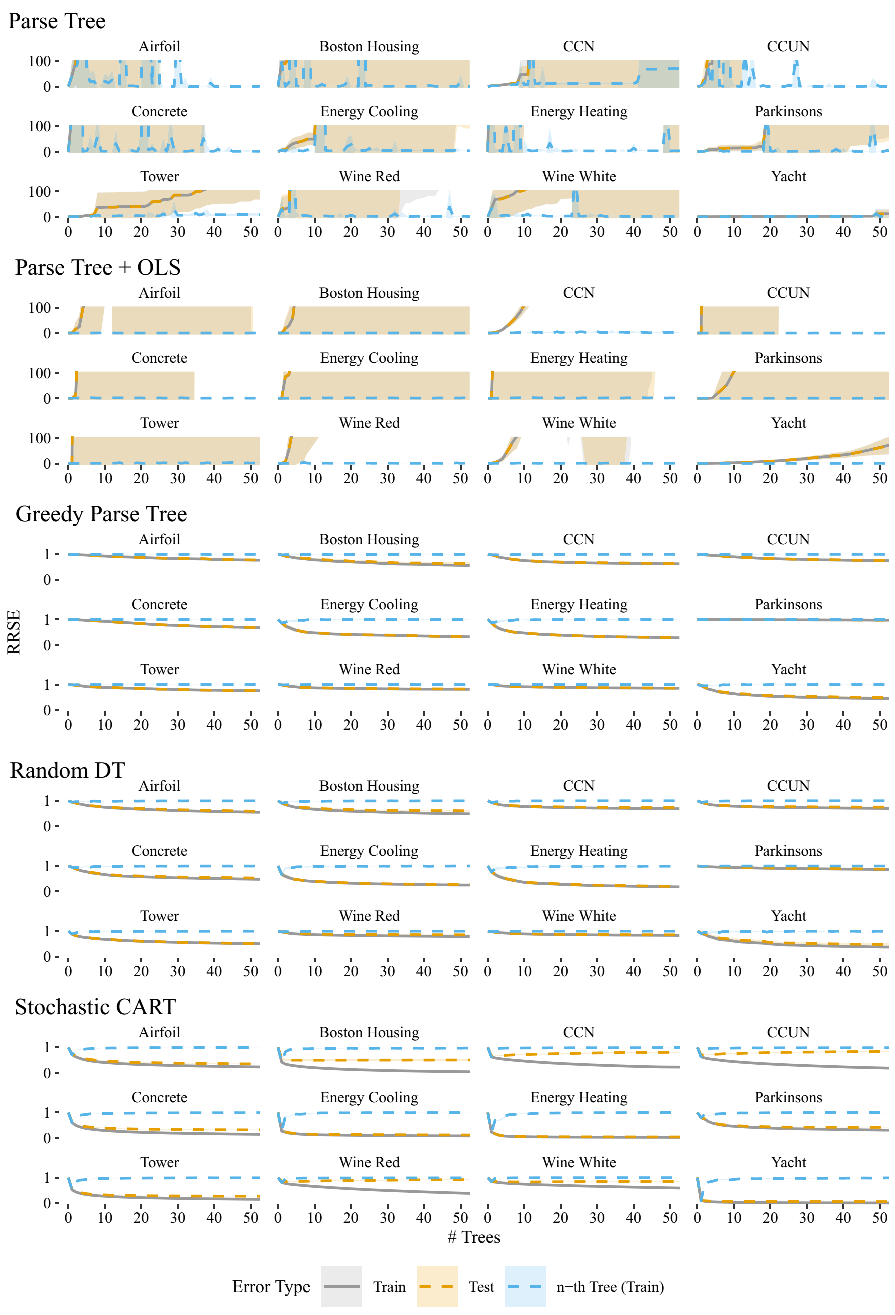

Fig. 8 Boosting performance of investigated parse tree learners. The RRSE performance against ensemble size is compared for each base learner type. Note the increased y-axis scale of [0, 100] for parse tree base learners to indicate their very unstable performance. The bottom three base learners all demonstrate the required *weak learner* behaviour needed to for effective boosting

6 Comparing PRB to contemporary boosting

Based on the results presented in Sects. 4 and 5, it seems that significant improvements in GSGP performance can be obtained through relatively simple modifications. However, the extensions resulting in PRB have the effect of moving it closer to existing boosting frameworks, such as gradient boosting machines (GBM) [11]. Therefore, there is a need to examine the resulting PRB frameworks against GBM to ensure that they present novel and useful behaviour.

The experimental setup from previous sections was used to compare PRB to GBM. The GBM package from CRAN was used for comparison [41]. GBM was permitted to develop ensembles of 250 trees, and trees of maximum depth 3 were used to remain consistent with the decision tree base learners used for PRB. Two variants of GBM were used: GBM on its own without regularisation, and $\mathrm{GBM_L}$, which is GBM followed by lasso regularisation once the ensemble was constructed (this emulates the regularisation process offered to PRB). Four variants of PRB were considered, all using lasso regularisation after each generation:

1. $\mathrm{PRB_P}$: PRB using a standard parse tree representation for the initial population and the random functions generated during mutation.
2. $\mathrm{PRB_G}$: PRB using the greedy parse tree representation defined in Sect. 5 for generating the random functions used in mutation.
3. $\mathrm{PRB_D}$: PRB using a random decision tree base learner for initial solutions and within mutation.
4. $\mathrm{PRB_C}$: PRB using a stochastic CART base learner, again in both population initialisation and subsequently within GSGP mutation.

Note that the only difference between each variant of PRB is the choice of base learner. Therefore, the results presented in this section can also be used to assess the effective performance of mutation produced by using these representations. Note also that, due to its poor performance demonstrated in Sect. 5, the parse tree base learner with OLS adaptation is not explored here.

Boxplots of performance in terms of RRSE and resulting ensemble size are shown in Figs. 9 and 10, respectively. Average rankings of the various methods for prediction performance and size are shown in Table 3. For statistical analyses, pairwise Wilcoxon signed-rank tests were performed for each pair of methods on each problem, with Holm's correction applied to correct for multiple comparisons: results of these tests are shown in Tables 4 and 5. It terms of prediction performance, it appears that PRB offers good performance, certainly on par with GBM. $\mathrm{PRB_C}$ offered statistically significant performance improvements over $\mathrm{GBM_L}$ on six problems, was inferior on five and differences in performance were not significant on the remaining problems. Size performance of all PRB variants were consistently better than that offered by GBM: given the similarity of the final ensemble representation of both PRB and GBM, it is interesting to observe such a difference. Explaining why PRB was more amenable to regularisation than GBM would be a worthy area of future work.

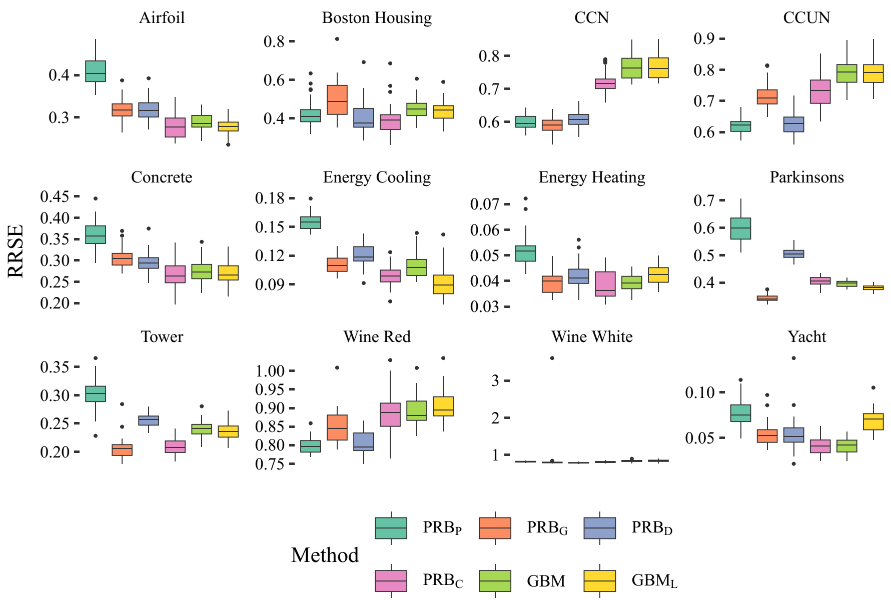

Fig. 9 Final test error performance comparison between PRB variants and GBM

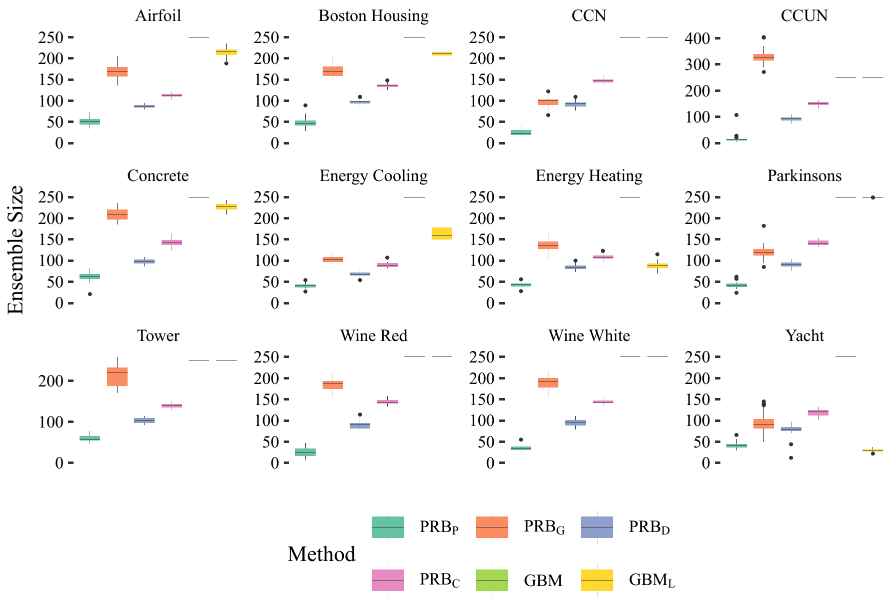

Fig. 10 Final ensemble size comparison between PRB variants and GBM

Table 3 Average rank of each method in the tested problems

Test RRSE performance			Ensemble size		
Method	Mean rank	SD rank	Method	Mean rank	SD rank
PRB_C	2.42	1.24	PRB_P	1.08	0.29
PRB_G	3.42	1.83	PRB_D	2.08	0.29
PRB_D	3.42	1.38	PRB_C	3.42	0.67
GBM	3.67	1.44	PRB_G	4.08	0.79
GBM_L	3.75	1.86	GBM_L	4.83	1.47
PRB_P	4.33	2.15	GBM	5.55	0.67

7 Conclusion

Geometric semantic genetic programming has received considerable attention due to its desirable properties of searching directly in the semantic space. However, previous work exploring GSGP has been largely fixed on using a tree representation. This paper has presented an alternate interpretation of GSGP, one that frames GSGP within the concept of ensemble learning. Specifically, GSGP mutation was recast as a boosting operator, while crossover was examined in the guise of a stacking operator. Coupled with a set-base representation of solutions, the resulting parallel recombinative boosting framework offers new insights into the nature of GSGP. The PRB framework, when coupled with effective regularisation, coefficient optimisation and base learner design as able to provide effective ensemble learning that is competitive with class-leading methods like GBM in terms of prediction quality, but is able to do so with much smaller ensemble sizes.

7.1 Future work

The PRB framework provides several insights into useful areas for future work in GSGP research. Although the standard parse tree base learner was shown to be largely ineffective, the actual representation itself, when coupled with a greedy strategy, was able to be more effective. This suggests that exploring more effective means of constructing parse trees is warranted. Given the natural interpretability of parse trees, finding more effective means of constructing them may provide useful techniques for not only GSGP, but more general genetic programming activities.

Linking with the idea of better tree construction, future work to better integrate GSGP and semantic backpropagation is warranted. While there has been much work exploring the use of coefficient optimisation within GSGP, this seems to be an under-explored area in the approximate geometric semantic methods.[5]

The approach in this paper of using lasso regularisation seems effective, but nonetheless exploring other means of regularisation and coefficient search is warranted. Dropout has been remarkably successful in introducing bagging-like dynamics into large deep learning models [28]. Pursuing dropout-like regularisation in GSGP and

[5] The work of Chen et al. [10] goes someway towards exploring this concept, and could be an excellent launching point from which this future work could be conducted.

Table 4 Comparisons of mean testing RRSE performance

Methods (A-B)		Problem					
		Airfoil	Boston housing	CCN	CCUN	Concrete	Energy cooling
PRB$_{\text{P}}$	PRB$_{\text{G}}$	***0.000≺***	*0.012 ≻*	0.395	***0.000≻***	***0.000≺***	***0.000≺***
	PRB$_{\text{D}}$	***0.000≺***	0.714	0.395	1.000	***0.000≺***	***0.000≺***
	PRB$_{\text{C}}$	***0.000≺***	0.696	***0.000≻***	***0.000≻***	***0.000≺***	***0.000≺***
	GBM	***0.000≺***	0.166	***0.000≻***	***0.000≻***	***0.000≺***	***0.000≺***
	GBM$_{\text{L}}$	***0.000≺***	0.714	***0.000≻***	***0.000≻***	***0.000≺***	***0.000≺***
PRB$_{\text{G}}$	PRB$_{\text{D}}$	1.000	***0.001≺***	*0.027 ≻*	***0.000≺***	0.173	***0.005 ≻***
	PRB$_{\text{C}}$	***0.000≺***	***0.000≺***	***0.000≻***	0.841	***0.000≺***	***0.000≺***
	GBM	***0.000≺***	0.257	***0.000≻***	***0.000≻***	***0.001≺***	0.399
	GBM$_{\text{L}}$	***0.000≺***	0.062	***0.000≻***	***0.000≻***	***0.000≺***	***0.000≺***
PRB$_{\text{D}}$	PRB$_{\text{C}}$	***0.000≺***	0.848	***0.000≻***	***0.000≻***	*0.013 ≺*	***0.000≺***
	GBM	***0.000≺***	*0.022 ≻*	***0.000≻***	***0.000≻***	0.173	***0.004 ≺***
	GBM$_{\text{L}}$	***0.000≺***	0.123	***0.000≻***	***0.000≻***	*0.021 ≺*	***0.000≺***
PRB$_{\text{C}}$	GBM	0.467	***0.008 ≻***	***0.000≻***	***0.000≻***	0.627	*0.012 ≻*
	GBM$_{\text{L}}$	1.000	0.083	***0.000≻***	***0.000≻***	0.766	0.060
GBM	GBM$_{\text{L}}$	0.165	0.714	0.906	1.000	0.766	***0.001≺***
	GBM$_{\text{L}}$	*0.033 ≻*	***0.000≺***	***0.000≻***	0.573	***0.000≻***	***0.000≻***
GBM	GBM$_{\text{L}}$	0.080	***0.000≺***	0.522	0.643	0.376	***0.000≻***

		Energy heating	Parkinsons	Tower	Wine red	Wine white	Yacht
PRB$_{\text{P}}$	PRB$_{\text{G}}$	***0.000≺***	***0.000≺***	***0.000≺***	***0.000≻***	***0.000≺***	***0.000≺***
	PRB$_{\text{D}}$	***0.000≺***	***0.000≺***	***0.000≺***	1.000	***0.000≺***	***0.000≺***
	PRB$_{\text{C}}$	***0.000≺***	***0.000≺***	***0.000≺***	***0.000≻***	0.376	***0.000≺***
	GBM	***0.000≺***	***0.000≺***	***0.000≺***	***0.000≻***	***0.001≻***	***0.000≺***
	GBM$_{\text{L}}$	***0.000≺***	***0.000≺***	***0.000≺***	***0.000≻***	***0.000≻***	0.128
PRB$_{\text{G}}$	PRB$_{\text{D}}$	0.518	***0.000≻***	***0.000≻***	***0.000≺***	*0.011 ≺*	1.000
	PRB$_{\text{C}}$	1.000	***0.000≻***	0.522	0.065	0.124	***0.001≺***
	GBM	1.000	***0.000≻***	***0.000≻***	***0.005≻***	***0.000≺***	***0.001≺***
	GBM$_{\text{L}}$	0.108	***0.000≻***	***0.000≻***	***0.000≻***	***0.000≺***	***0.000≻***
PRB$_{\text{D}}$	PRB$_{\text{C}}$	0.108	***0.000≺***	***0.000≺***	***0.000≻***	***0.000≻***	***0.002 ≺***
	GBM	0.518	***0.000≺***	***0.001≺***	***0.000≻***	***0.000≻***	***0.001≺***
	GBM$_{\text{L}}$	1.000	***0.000≺***	***0.000≺***	***0.000≻***	***0.000≻***	***0.001≻***
PRB$_{\text{C}}$	GBM	1.000	*0.017 ≺*	***0.000≻***	1.000	***0.001≻***	1.000
	GBM$_{\text{L}}$	*0.033 ≻*	***0.000≺***	***0.000≻***	0.573	***0.000≻***	***0.000≻***
GBM	GBM$_{\text{L}}$	0.080	***0.000≺***	0.522	0.643	0.376	***0.000≻***

For each problem, the *p* value (adjusted for multiple comparisons) associated with a Wilcoxon signed-rank test for difference between the two methods A and B is shown. Where relevant, each *p* value value is shown with a symbol ≻ or ≺ to indicate a significant difference in favour of method A or B, respectively

PRB may help to promote similar variance-reducing behaviour and serve as a useful complement to the natural boosting properties of the method.

Table 5 Comparisons of mean ensemble sizes

Methods (A-B)		Problem											
		Airfoil	Boston housing	CCN	CCUN	Concrete	Energy cooling	Energy heating	Parkinsons	Tower	Wine red	Wine white	Yacht
PRB$_{P}$	PRB$_{G}$	***0.000≻***	***0.000≻***	***0.000≻***	***0.000≻***	***0.000≻***	***0.000≻***	***0.000≻***	***0.000≻***	***0.000≻***	***0.000≻***	***0.000≻***	***0.000≻***
	PRB$_{D}$	***0.000≻***	***0.000≻***	***0.000≻***	***0.000≻***	***0.000≻***	***0.000≻***	***0.000≻***	***0.000≻***	***0.000≻***	***0.000≻***	***0.000≻***	***0.000≻***
	PRB$_{C}$	***0.000≻***	***0.000≻***	***0.000≻***	***0.000≻***	***0.000≻***	***0.000≻***	***0.000≻***	***0.000≻***	***0.000≻***	***0.000≻***	***0.000≻***	***0.000≻***
	GBM	***0.000≻***	***0.000≻***	***0.000≻***	***0.000≻***	***0.000≻***	***0.000≻***	***0.000≻***	***0.000≻***	***0.000≻***	***0.000≻***	***0.000≻***	***0.000≻***
	GBM$_{L}$	***0.000≻***	***0.000≻***	***0.000≻***	***0.000≻***	***0.000≻***	***0.000≻***	***0.000≻***	***0.000≻***	***0.000≻***	***0.000≻***	***0.000≻***	***0.000≺***
PRB$_{G}$	PRB$_{D}$	***0.000≺***	***0.000≺***	*0.017 ≺*	***0.000≺***	***0.000≺***	***0.000≺***	***0.000≺***	***0.000≺***	***0.000≺***	***0.000≺***	***0.000≺***	***0.003 ≺***
	PRB$_{C}$	***0.000≺***	***0.000≺***	***0.000≻***	***0.000≺***	***0.000≺***	***0.000≺***	***0.000≺***	***0.000≻***	***0.000≺***	***0.000≺***	***0.000≺***	***0.000≻***
	GBM	***0.000≻***	***0.000≻***	***0.000≻***	***0.000≺***	***0.000≻***	***0.000≻***	***0.000≻***	***0.000≻***	***0.000≻***	***0.000≻***	***0.000≻***	***0.000≻***
	GBM$_{L}$	***0.000≻***	***0.000≻***	***0.000≻***	***0.000≺***	***0.000≻***	***0.000≻***	***0.000≺***	***0.000≻***	***0.000≻***	***0.000≻***	***0.000≻***	***0.000≺***
PRB$_{D}$	PRB$_{C}$	***0.000≻***	***0.000≻***	***0.000≻***	***0.000≻***	***0.000≻***	***0.000≻***	***0.000≻***	***0.000≻***	***0.000≻***	***0.000≻***	***0.000≻***	***0.000≻***
	GBM	***0.000≻***	***0.000≻***	***0.000≻***	***0.000≻***	***0.000≻***	***0.000≻***	***0.000≻***	***0.000≻***	***0.000≻***	***0.000≻***	***0.000≻***	***0.000≻***
	GBM$_{L}$	***0.000≻***	***0.000≻***	***0.000≻***	***0.000≻***	***0.000≻***	***0.000≻***	*0.043 ≻*	***0.000≻***	***0.000≻***	***0.000≻***	***0.000≻***	***0.000≺***
PRB$_{C}$	GBM	***0.000≻***	***0.000≻***	***0.000≻***	***0.000≻***	***0.000≻***	***0.000≻***	***0.000≻***	***0.000≻***	***0.000≻***	***0.000≻***	***0.000≻***	***0.000≻***
	GBM$_{L}$	***0.000≻***	***0.000≻***	***0.000≻***	***0.000≻***	***0.000≻***	***0.000≻***	***0.000≺***	***0.000≻***	***0.000≻***	***0.000≻***	***0.000≻***	***0.000≺***
GBM	GBM$_{L}$	***0.000≺***	***0.000≺***	***0.000≺***	***0.000≺***	***0.000≺***	***0.000≺***	***0.000≺***	0.078	***0.000≺***	***0.000≺***	***0.000≺***	***0.000≺***

For each problem, the p value (adjusted for multiple comparisons) associated with a Wilcoxon signed-rank test for difference between the two methods A and B is shown. Where relevant, each p value value is shown with a symbol ≻ or ≺ to indicate a significant difference in favour of method A or B, respectively

Finally, it seems prudent to conduct a full and thorough examination of the hyperparameter space of GSGP and PRB. While there is some question about the (lack of) sensitivity of evolutionary computation methods towards hyperparameter settings [42], a new interpretation of GSGP under an ensemble learning framework may help to uncover more effective hyperparameter settings to improve both scalability and application.

Author Contributions All work in preparing the paper was done by the sole author.

Funding Open Access funding enabled and organized by CAUL and its Member Institutions.

Code availability All code for final paper version will be published via http://github.com/grantdick/

Declarations

Conflict of interest The authors declare that they have no conflict of interest.

References

1. J.R. Koza, *Genetic Programming: On the Programming of Computers by Means of Natural Selection* (MIT Press, Cambridge, 1992)
2. E. Galván-López, J. McDermott, M. O'Neill, A. Brabazon, Towards an understanding of locality in genetic programming, in *Proceedings of the 12th Annual Conference on Genetic and Evolutionary Computation. GECCO '10* (Association for Computing Machinery, New York, 2010), pp. 901–908. https://doi.org/10.1145/1830483.1830646
3. P.A. Whigham, G. Dick, J. Maclaurin, On the mapping of genotype to phenotype in evolutionary algorithms. Genet. Progr. Evolvable Mach. **18**, 353–361 (2017)
4. K. Krawiec, P. Liskowski, Automatic derivation of search objectives for test-based genetic programming, in *Genet. Program.* ed. by P. Machado, M.I. Heywood, J. McDermott, M. Castelli, P. García-Sánchez, P. Burelli, S. Risi, K. Sim (Springer, Cham, 2015), pp.53–65
5. L. Vanneschi, M. Castelli, S. Silva, A survey of semantic methods in genetic programming. Genet. Progr. Evolvable Mach. **15**, 195–214 (2014)
6. A. Moraglio, K. Krawiec, C.G. Johnson, Geometric semantic genetic programming, in *Parallel Problem Solving from Nature—PPSN XII*. ed. by C.A.C. Coello, V. Cutello, K. Deb, S. Forrest, G. Nicosia, M. Pavone (Springer, Berlin, 2012), pp.21–31
7. J.F.B.S. Martins, L.O.V.B. Oliveira, L.F. Miranda, F. Casadei, G.L. Pappa, Solving the exponential growth of symbolic regression trees in geometric semantic genetic programming, in *Proceedings of the Genetic and Evolutionary Computation Conference. GECCO '18* (Association for Computing Machinery, New York, 2018), pp. 1151–1158. https://doi.org/10.1145/3205455.3205593
8. J. McDermott, A. Agapitos, A. Brabazon, M. O'Neill, Geometric semantic genetic programming for financial data, in *Applications of Evolutionary Computation*. ed. by A.I. Esparcia-Alcázar, A.M. Mora (Springer, Berlin, 2014), pp.215–226
9. M. Castelli, L. Trujillo, L. Vanneschi, S. Silva, E. Z-Flores, P. Legrand, Geometric semantic genetic programming with local search, in *Proceedings of the 2015 Annual Conference on Genetic and Evolutionary Computation. GECCO '15* (Association for Computing Machinery, New York, 2015), pp. 999–1006. https://doi.org/10.1145/2739480.2754795

10. Q. Chen, B. Xue, M. Zhang, Improving generalization of genetic programming for symbolic regression with angle-driven geometric semantic operators. IEEE Trans. Evol. Comput. **23**(3), 488–502 (2019). https://doi.org/10.1109/TEVC.2018.2869621
11. J.H. Friedman, Greedy function approximation: a gradient boosting machine. Ann. Stat. **29**(5), 1189–1232 (2001). https://doi.org/10.1214/aos/1013203451
12. R. Tibshirani, Regression shrinkage and selection via the lasso. J. R. Stat. Soc. Ser. B (Methodol.) **58**(1), 267–288 (1996)
13. E. Galván-López, J. McDermott, M. O'Neill, A. Brabazon, Defining locality as a problem difficulty measure in genetic programming. Genet. Progr. Evolvable Mach. **12**, 365–401 (2011)
14. N.Q. Uy, N.X. Hoai, M. O'Neill, R.I. McKay, D.N. Phong, On the roles of semantic locality of crossover in genetic programming. Inf. Sci. **235**, 195–213 (2013). https://doi.org/10.1016/j.ins.2013.02.008. (**Data-based Control, Decision, Scheduling and Fault Diagnostics**)
15. F. Rothlauf, M. Oetzel, On the locality of grammatical evolution, in *Genetic Programming*. ed. by P. Collet, M. Tomassini, M. Ebner, S. Gustafson, A. Ekárt (Springer, Berlin, 2006), pp.320–330
16. T. Seaton, J.F. Miller, T. Clarke, An ecological approach to measuring locality in linear genotype to phenotype maps, in *Proceedings of the 15th European Conference on Genetic Programming. EuroGP'12* (Springer, Berlin, 2012), pp. 170–181. https://doi.org/10.1007/978-3-642-29139-5_15
17. T.P. Pawlak, K. Krawiec, Progress properties and fitness bounds for geometric semantic search operators. Genet. Progr. Evolvable Mach. **17**, 5–23 (2016)
18. A. Moraglio, K. Krawiec, Semantic genetic programming, in *Proceedings of the 2016 on Genetic and Evolutionary Computation Conference Companion. GECCO '16 Companion* (Association for Computing Machinery, New York, 2016), pp. 639–662. https://doi.org/10.1145/2908961.2926990
19. A. Moraglio, An efficient implementation of gsgp using higher-order functions and memoization, in *SMGP Workshop at PPSN* (2014)
20. L. Vanneschi, M. Castelli, L. Manzoni, S. Silva, A new implementation of geometric semantic GP and its application to problems in pharmacokinetics, in *Genetic Programming*. ed. by K. Krawiec, A. Moraglio, T. Hu, A.Ş. Etaner-Uyar, B. Hu (Springer, Berlin, 2013), pp.205–216
21. K. Krawiec, P. Lichocki, Approximating geometric crossover in semantic space, in *Proceedings of the 11th Annual Conference on Genetic and Evolutionary Computation. GECCO '09* (Association for Computing Machinery, New York, 2009), pp. 987–994. https://doi.org/10.1145/1569901.1570036
22. T.P. Pawlak, B. Wieloch, K. Krawiec, Semantic backpropagation for designing search operators in genetic programming. IEEE Trans. Evol. Comput. **19**(3), 326–340 (2015). https://doi.org/10.1109/TEVC.2014.2321259
23. T.P. Pawlak, K. Krawiec, Semantic geometric initialization, in *Genetic Programming*. ed. by M.I. Heywood, J. McDermott, M. Castelli, E. Costa, K. Sim (Springer, Cham, 2016), pp.261–277
24. T.P. Pawlak, K. Krawiec, Competent geometric semantic genetic programming for symbolic regression and Boolean function synthesis. Evol. Comput. **26**(2), 177–212 (2018). https://doi.org/10.1162/evco_a_00205
25. G. Dick, Improving geometric semantic genetic programming with safe tree initialisation, in *Genetic Programming*. ed. by P. Machado, M.I. Heywood, J. McDermott, M. Castelli, P. García-Sánchez, P. Burelli, S. Risi, K. Sim (Springer, Cham, 2015), pp.28–40
26. T.P. Pawlak, Geometric semantic genetic programming is overkill, in *Genetic Programming*. ed. by M.I. Heywood, J. McDermott, M. Castelli, E. Costa, K. Sim (Springer, Cham, 2016), pp.246–260
27. T. Hastie, R. Tibshirani, J.H. Friedman, J.H. Friedman, *The Elements of Statistical Learning: Data Mining, Inference, and Prediction*, vol. 2 (Springer, Berlin, 2009)
28. N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, R. Salakhutdinov, Dropout: a simple way to prevent neural networks from overfitting. J. Mach. Learn. Res. **15**(56), 1929–1958 (2014)
29. F. Huang, J. Ash, J. Langford, R. Schapire, Learning deep ResNet blocks sequentially using boosting theory, in *Proceedings of the 35th International Conference on Machine Learning. Proceedings of Machine Learning Research*, vol. 80, ed. by J. Dy, A. Krause (2018), pp. 2058–2067
30. L. Breiman, Random forests. Mach. Learn. **45**(1), 5–32 (2001)
31. D.H. Wolpert, Stacked generalization. Neural Netw. **5**(2), 241–259 (1992). https://doi.org/10.1016/S0893-6080(05)80023-1
32. T. Brooks, D. Pope, M. Marcolini, Airfoil Self-Noise. UCI Machine Learning Repository (2014). https://doi.org/10.24432/C5VW2C
33. D. Harrison, D.L. Rubinfeld, Hedonic housing prices and the demand for clean air. J. Environ. Econ. Manag. **5**(1), 81–102 (1978)

34. L. Breiman, J. Friedman, C.J. Stone, R.A. Olshen, *Classification and Regression Trees* (CRC Press, Boca Raton, 1984)
35. I.-C. Yeh, Modeling of strength of high-performance concrete using artificial neural networks. Cem. Concr. Res. **28**(12), 1797–1808 (1998)
36. A. Tsanas, A. Xifara, Energy efficiency. UCI Machine Learning Repository (2012). https://doi.org/10.24432/C51307
37. A. Tsanas, M. Little, Parkinsons Telemonitoring. UCI Machine Learning Repository (2009). https://doi.org/10.24432/C5ZS3N
38. E.J. Vladislavleva, G.F. Smits, D. Hertog, Order of nonlinearity as a complexity measure for models generated by symbolic regression via pareto genetic programming. IEEE Trans. Evol. Comput. **13**(2), 333–349 (2009). https://doi.org/10.1109/TEVC.2008.926486
39. P. Cortez, A. Cerdeira, F. Almeida, T. Matos, J. Reis, Wine Quality. UCI Machine Learning Repository (2009). https://doi.org/10.24432/C56S3T
40. J. Gerritsma, R. Onnink, A. Versluis, Yacht Hydrodynamics. UCI Machine Learning Repository (2013). https://doi.org/10.24432/C5XG7R
41. B. Greenwell, B. Boehmke, J. Cunningham, G. Developers, Gbm: Generalized Boosted Regression Models. R package version 2.1.8.1 (2022). https://CRAN.R-project.org/package=gbm
42. M. Sipper, W. Fu, K. Ahuja, J.H. Moore, Investigating the parameter space of evolutionary algorithms. BioData Min. **11**, 1–14 (2018)