Applied Soft Computing Journal 104 (2021) 107198

Contents lists available at ScienceDirect

Applied Soft Computing Journal

journal homepage: www.elsevier.com/locate/asoc

Conference

Evolving simple and accurate symbolic regression models via asynchronous parallel computing

Aliyu Sani Sambo [a,*], R. Muhammad Atif Azad [a], Yevgeniya Kovalchuk [a], Vivek Padmanaabhan Indramohan [b], Hanifa Shah [c]

[a] School of Computing and Digital Technology, Birmingham City University, UK
[b] School of Health, Education and Life Sciences, Birmingham City University, UK
[c] Faculty of Computing, Engineering and the Built Environment, Birmingham City University, UK

ARTICLE INFO

Article history:
Received 9 October 2020
Received in revised form 9 February 2021
Accepted 13 February 2021
Available online 24 February 2021

Keywords:
Genetic programming
Model complexity
Parallel computing
Evaluation time

ABSTRACT

In machine learning, reducing the complexity of a model can help to improve its computational efficiency and avoid overfitting. In genetic programming (GP), the model complexity reduction is often achieved by reducing the size of evolved expressions. However, previous studies have demonstrated that the expression size reduction does not necessarily prevent model overfitting. Therefore, this paper uses the *evaluation time* – the computational time required to evaluate a GP model on data – as the estimate of model complexity. The evaluation time depends not only on the size of evolved expressions but also their *composition*, thus acting as a more nuanced measure of model complexity than the expression size alone. To discourage complexity, this study employs a novel method called asynchronous parallel GP (APGP) that introduces a race condition in the evolutionary process of GP; the race offers an evolutionary advantage to the simple solutions when their accuracy is competitive. To evaluate the proposed method, it is compared to the standard GP (GP) and GP with bloat control (GP+BC) methods on six challenging symbolic regression problems. APGP produced models that are significantly more accurate (on 6/6 problems) than those produced by both GP and GP+BC. In terms of complexity control, APGP prevailed over GP but not over GP+BC; however, GP+BC produced simpler solutions at the cost of test-set accuracy. Moreover, APGP took a significantly lower number of evaluations than both GP and GP+BC to meet a target training fitness in all tests. Our analysis of the proposed APGP also involved: (1) an ablation study that separated the proposed measure of complexity from the race condition in APGP and (2) the study of an initialisation scheme that encourages functional diversity in the initial population that improved the results for all the GP methods. These results question the overall benefits of bloat control and endorse the employment of both the evaluation time as an estimate of model complexity and the proposed APGP method for controlling it.

1. Introduction

The key challenges in managing the complexity of machine learning (ML) models include defining what complexity is and constructing a mechanism to control it; however, because the motivations behind existing definitions vary significantly, these definitions fail to transfer across various applications or algorithms. For example, a common reason for managing the complexity of ML models is attaining models that are just complex enough to explain the phenomenon generating the given data but not too complex to reflect noise in the training data. This way, the predictions on unseen data will be accurate [1]. Often in standard regression methods, the complexity is constrained by penalising the increase in the magnitudes of the coefficients of a fixed model. However, in methods such as Genetic Programming [2] the model is not fixed and hence such a penalty does not make sense. Another reason for controlling complexity is interpretability because simple models can be more interpretable [3]; interpretability of ML models is now a legal requirement due to frameworks such as the EU General Data Protection Regulation (GDPR).[1] Although very useful, such research produces post-hoc methods to explain pre-trained models on each training instance [4]. Therefore, this interpretability does not concern accuracy on test data. A yet

* Corresponding author.
E-mail address: aliyu.sambo@mail.bcu.ac.uk (A.S. Sambo).

https://doi.org/10.1016/j.asoc.2021.107198
1568-4946/

[1] The EU General Data Protection Regulation includes a *right to explanation* in situations, where ML algorithms are applied to make a decision affecting a person.

another incentive for managing complexity is computational constraints. For example, *Internet of Things* (IoT) devices constrain the evaluation time of an acceptable model even if this compromises the model's accuracy [5]. However, a question arises as to whether constraining evaluation times can effectively decrease complexity and improve accuracy. In summary, the reasons for controlling model complexity vary and so does the notion of complexity [6].

This study focuses on the complexity of models produced by Genetic Programming (GP) [7]. In GP, the concern is that models may grow too complex and render ineffective evolutionary search [8]. Bloat control is thus the most common way of controlling model complexity in GP, and it limits the sizes of the evolved expressions. However, previous studies have demonstrated that bloat control alone does not always overcome the model overfitting problem (that is, a good performance on the training/seen data but a poor performance on the test/unseen data) [9,10].

To address such ineffective control of model complexity in GP, this paper presents a novel method called *Asynchronous Parallel Genetic Programming* (APGP). Instead of model size, APGP employs the *evaluation time* – the computational time required to evaluate a GP model on data – as a notion of its complexity. This notion is based on the observation that a model made up of computationally expensive building blocks or having large structures takes a long time to be evaluated, and hence is computationally complex. Therefore, the evaluation time control discourages both the structural as well as functional complexity.

The next question is how to control the evaluation times. Instead of subjectively penalising the slow evaluations, APGP takes a simple view: induce a *race* among competing models that allows a model to join the breeding population as soon as it has finished evaluating. Hence, the faster models can (fitness permitting) join the breeding population before their slower counterparts and gain an evolutionary advantage. This advantage arises because the competing models compete in terms of not only their accuracy but also their evaluation times due to the race; this is quite unlike in standard GP where each evaluation (or a batch of evaluations, as in generational replacement) is allowed to finish before the next evaluation (or a batch of evaluations) can start. Note, however, that selection is solely based on accuracy; therefore, APGP facilitates a dynamic interplay between accuracy and simplicity. To induce this race, APGP evaluates multiple models simultaneously across multiple asynchronous threads.

To evaluate the effectiveness of the proposed APGP method, this work first compares APGP with standard GP and GP with a very effective bloat control mechanism (GP+BC). The results indicate that APGP is capable of breeding models that are simpler than those produced with GP, yet more accurate on both training and test data than the models produced by the other two methods. Moreover, APGP takes a lower number of evaluations to match the training accuracy of GP compared to GP+BC.

This work then further analyses the proposed APGP in two ways. The first is an ablation study that analyses containing complexity *explicitly* with evaluation time instead of using APGP, which controls complexity *implicitly* with a race. To do that we employ four effective bloat-control techniques but to control evaluation time instead of expression sizes. The results indicate that while evaluation time control performs better than size control, the APGP still produces more accurate models (both training and test) than explicit control of evaluation time but while allowing greater model complexity. This suggests that the complexity control in APGP balances the accuracy-simplicity dilemma better than its counterparts.

This work also explored an initialisation scheme where the initial population comprises of identically sized individuals. This is because when models in a population are sized identically, the evaluation times are determined primarily by functional complexity; therefore, an evaluation time control can discourage functional complexity more than it can in a size-varying population. The results indicate that this initialisation scheme benefited all the methods, even the standard bloat control methods that do not employ time control.

The rest of this paper is organised as follows. Section 2 provides some background on GP and some notions of complexity related to it. Also in Section 2, the concept of evaluation time is introduced and the challenge of measuring it reliably is discussed. Section 3 presents the proposed APGP method. The experimental setup is outlined in Section 4. The results of the experiments are reported and discussed in Section 5. Sections 6 and 7 provide further analysis of the proposed method. Finally, Section 8 concludes the paper and outlines some directions for further work.

2. Background

2.1. Genetic Programming (GP)

GP enables computers to program themselves and build models automatically. It is also known as an evolutionary algorithm (EA) because it loosely imitates the Darwinian principle of natural selection to automatically search the space of possible models without any prior knowledge [2]. The process involves probabilistically choosing better performing individuals (models) from a given population, and producing offspring via simulated genetic operators of crossover and mutation. GP has been used to produce solutions to many practical problems. A detailed account of practical and innovative results produced by GP can be found in [11].

A variable tree-styled structure is the traditional and most popular representation of GP individuals [12], although other variations are also common [13–18]. These representations produce models of different sizes; in fact, these sizes may grow without improving accuracy (also known as code bloat) [19]. To address this issue, a lot of research efforts have been dedicated to developing methods for measuring and managing the complexity of GP models. The existing notions of complexity in GP and ways of its control are discussed below in turn.

2.2. Complexity in genetic programming

Some motivations for managing the complexity of computational models were discussed in Section 1. Here, we consider how some of these motivations are being addressed in GP. This involves examining the most widely used measures of complexity and mechanisms employed to control them.

2.2.1. Structural complexity

Traditionally, controlling complexity in GP means controlling the size of evolved structures, which includes limiting the number of nodes, encapsulated sub-trees and layers making up the evolved expressions, while ignoring the underlying functional or computational complexity.

The size is a simple and easy-to-calculate measure of complexity [20]. Early in the development of GP, it was observed that the size of GP models can grow unnecessarily (a phenomenon termed as bloat), severely constraining the computational resources [8]. The study of bloat control in GP has produced a large body of literature that cannot be covered extensively here. Instead, we provide a quick summary of some of the popular bloat control methods.

Many bloat-control techniques either set an arbitrary size (or depth) limit for models, or penalise large models [21]. However, such setups also encourage growing up to these size limits because that guarantees survival of such individuals after

crossover [22,23]. Other bloat-control techniques either limit the search space or reduce the likelihood of producing large models through customisation of the evolutionary process [21]. Another approach is to use multi-objective genetic programming (MOGP) [20] that optimises the twin objectives of fitness and size to obtain Pareto optimal solutions. MOGP is sometimes used in combination with other measures of complexity.

Dynamic Operator equalisation (DynOpEQ) [24], a recent and advanced bloat-control technique, dynamically changes the distribution of size in a population to admit more individuals in those size ranges that are producing fitter models. First, the individuals of the population are segmented into bins according to their sizes. When a new generation is produced, the number of individuals allowed in each bin is determined by the average fitness score of the corresponding bin in the parent generation; bins with higher average fitness scores are allowed more individuals. However, an offspring will always be admitted if its fitness score is higher than the best of the bin it belongs to, even if the bin's quota has been exhausted. Further, if an offspring is the new best and its size does not fit into any existing bin, then a new bin is created for it. In effect, DynOpEQ does not guarantee that bloated individuals will not exist in the population; however, the population is controlled in such a way that growth in size is somewhat contained unless fitness is improving. DynOpEQ is inefficient as it discards a large portion of evaluated candidates; an evaluated offspring is likely to be discarded if the bin it belongs to is full. To alleviate this problem, Mutation Operator Equalisation [24] was introduced; it mutates candidates (in small steps) to fit into the nearest bin with available space. However, the required changes may be exceedingly destructive to the model's fitness when the distance between the individual's original size and the size it needs to be is too large.

Some studies have explored the concept of Kolmogorov complexity [25,26] to manage complexity in GP. This concept is adopted from algorithmic information theory and relates to specifying an object such as a binary string or a sequence of numbers. The Kolmogorov complexity of such an object is the length of the shortest computer program that can produce the object and nothing more. An abstract coding language (universal Turing machine) is used as a reference. However, this measure is uncomputable [27,28] because the output of every program cannot be computed to definitively determine the shortest. Hence, the minimum description length (MDL) [29,30], a computable form of the Kolmogorov complexity, has been applied in GP instead [31]. The study [31] calculated the MDL value of an individual by summing the length of code required to encode it and the length of code required to encode its classification errors. Thereafter, a scaling technique [32] was used to transform the MDL value to a windowed MDL value; the windowed MDL value is relative to the maximum MDL value of the individuals produced up to that point of the GP run. The MDL-based fitness function, which was setup to minimise the windowed MDL value, controlled the growth of the individuals. However, the implementation only works under two conditions: (1) where performance of individuals improves with the growth of the trees, and (2) where the fitness of the substructures of the trees are well-defined.

Restricting the structural representation of GP solutions is not an effective way of managing their complexity. In symbolic regression, for example, the model size does not represent the underlying functional and computational complexity of the model. For instance, restricting size would mean penalising the large yet linear expression $9x + 6x + 3x + 2x + x$, which is computationally less complex than the smaller expression $sin(x)$ [10], which is computationally equivalent to the implementation of its Taylor series expansion $\sum_{n=0}^{\infty}(-1)^n \frac{x^{2n+1}}{(2n+1)!}$. Moreover, the response surface of $sin(x)$ is more complex than that of the linear function that is larger in size. Further, two expressions with the same size may have different response surfaces; the response of the function $sin(9x)$ has greater fluctuations than $sin(2x)$. Thus, model complexity in GP is more than their representation.

2.2.2. Functional complexity

Approaches based on functional complexity focus on the functionality of models rather than their representation. To elicit functional complexity, one method approximates the evolved expressions with Chebyshev polynomials [33] such that functionally complex expressions are approximated by polynomials of a higher degree. The degree of approximating polynomials is termed as the *order of non-linearity* of the corresponding GP expressions [34], which should be minimised. However, the minimisation procedure requires the evolved expressions to be twice differentiable, a property that is not always guaranteed. The study [33] aimed to optimise three objectives: accuracy, the order of non-linearity and expressional complexity (size). A trade-off between accuracy and the order-of-non-linearity to guide selection led to improved generalisation over bloat control. Further, the study introduced a system that alternates between the optimisation of two sets of objectives during the evolution: (1) the optimisation of accuracy and the order of non-linearity, and (2) the optimisation of accuracy with size. This 2-D optimisation framework showed improvements in both managing the size of evolved models and their generalisation ability.

To avoid the constraint of twice-differentiability in [33] Vanneschi et al. [9] defined a less rigorous measure of functional complexity, whereby the slope of an expression along each feature dimension is approximated with a simpler but error-prone measure that approximates second order partial derivatives with a finite difference method that uses unequal intervals; however, the eventual measure of complexity that the work proposes is mathematically questionable because it simply averages these approximations to partial derivatives across all the feature dimensions to get an average complexity. To what extent that average diverges from a Hessian is not discussed. To avoid computational expense, this measure was only computed for the best individual in each generation. Crucially, however, the paper reported that a decrease in this measure did not necessarily decrease overfitting.

Castelli et al. [35] proposed other complexity measures that relate to curvature. The authors quantified the *degree of curvature* by examining the output of the pairs of close training points for a given model. The number of pairs with very different outputs were used to reflect the curvature. This formed the basis of two measures: (1) *graph-based complexity* to measure functional complexity and (2) *graph-based learning ability* to quantify the ability to learn difficult training points. The result of the experiments showed some generalisation gains over standard GP.

2.2.3. Complexity based on statistical learning

Methods for managing model complexity based on statistical learning theory, including generalisation error-bound Vapnik–Chervonenkis (VC) theory and Rademacher complexity theory [36,37], are designed to find a balance between model complexity and its generalisation capability; while models should be complex enough to find patterns in training data, those that are too complex do not generalise well. The VC dimension is a general measure of the capacity or complexity of a learning machine [38,39]. According to the original definition proposed for a set of indicator functions, the VC dimension is the maximum number of vectors that can be separated (shattered) into two classes in all possible ways by a set of functions [38]. The VC dimension is used to provide various estimations of generalisation errors.

Structural risk minimisation (SRM) is a framework that uses the VC dimension to assess the generalisation ability of a learning

machine [40]. The assessment involves predicting the distance between the training and test errors. To do this, SRM defines the upper bound of the generalisation error based on the empirical risk (training error) and confidence interval. The confidence interval measures the difference between the empirical risk (training error) and the expected risk (generalisation error); it depends on (1) the size of the training set and (2) VC dimension. When the size of the training data is fixed, the generalisation bound is indicated by the VC dimension only; therefore, it is also called the VC generalisation bound or VC bound. SRM is used to produce an optimal model that finds a balance between minimising the upper bound of the generalisation error and minimising the training error. SRM was also used to manage the complexity of evolved models in [41]. While the proposed method outperformed standard GP by producing smaller sized models with better generalisation, the authors acknowledged the expensive computational cost of the method and lack of exploration of the parameters used in measuring the VC dimension.

Rademacher complexity extends the VC dimension to handle real-valued functions; therefore, it can be applied to both classification and regression problems. This measure of complexity has a tighter bound on the generalisation error than the VC dimension, and unlike the VC dimension, it depends on the data distribution. Rademacher complexity was used in [42] to drive the evolutionary process, which led to models with significantly smaller sizes and better generalisation ability compared to those generated by standard GP. However, the implementation has a lengthy process of tuning the parameter used to increase the pressure when overfitting occurs. Similar to the VC dimension, this is not a trivial task to compute.

Azad and Ryan [43] used the variance of the output values of evolving expressions to infer their complexity. They combined the variance and the fitness as twin objectives to optimise. Although variance differs with mathematical smoothness (a straight line can have a greater variance than a sinusoid), its combination with fitness meant that expressions within similar functional space may normally be compared in later generations with greater genetic convergence; however, this needs further verification. The method improved generalisation. Moreover, since the method does not require specialised multi-objective optimisation methods, it is simple to implement and computationally inexpensive.

Ni and Rockett [44] coined a measure of complexity that incorporates Tikhonov regularisation (as a functional complexity indicator) and size (a structural complexity measure). Tikhonov regularisation (also known as ridge regression) is a simple and common L^2 parameter regularisation strategy. This work was motivated by the understanding that addressing functional complexity does not necessarily address structural complexity and vice versa; also both are important in GP. Therefore, they used a two-dimensional vector that consisted of the Tikhonov regulariser (an indicator of the smoothness of response of the function) and the size of the model to represent its complexity; the Pareto optimal individual is considered the simpler individual. The two-dimensional vector was then used as an objective in conjunction with accuracy in a traditional multi-objective optimisation approach. This method led to generalisation gains over GP with size control. It also attained higher accuracy over GP with Tikhonov regularisation.

In summary, the popular techniques based on structural complexity tend to reduce the model sizes but not overfitting; likewise, the techniques based on functional complexity show some generalisation gains but often need to be married up with structural complexity and require parameter tuning overhead; also, these techniques are often tailored to specific tasks such as regression whereas automatic programming in GP extends beyond that. In contrast, the present study aims to induce non-complex models naturally regardless of the application domain by using the model evaluation time as a measure of its complexity.

The use of the evaluation time as an indicator of complexity is relatively new in GP such as in the proof of concept of this work [45]. At around the time of publication of [45] another study [46] also used evaluation times to control bloat, where the correlation between the size and the evaluation time was employed to manage bloat in a variety of problems. Although that second study corroborates our proposal that the evaluation time can be used to control model complexity, this work goes beyond bloat control to assess how evaluation time reflects both size and functional complexity, and proposes a new method, called APGP, for managing model complexity by enabling a race condition among evolved models.

2.3. Time is not size

Since Section 2.2.1 explains that model size does not necessarily represent functional complexity, this section empirically demonstrates how, instead, evaluation time can discriminate between different functional complexities. This demonstration is important because it verifies whether considerable time differences exist between different functional complexities of identically sized expressions so that a GP system can exploit these differences to promote functional simplicity. After all, evaluation time is also a function of model size, and if functional differences of identically sized expressions do not show up in evaluation times, then measuring time is just another way of measuring expression sizes. Clearly, this is undesirable.

To this end, four different *function sets* were used to generate symbolic regression models of different complexities. Functional complexity of these sets decreases in the following order: $\{cos, sin\}$, $\{cos, sin, +, -\}$, $\{\times, \div, +, -\}$ and $\{+, -\}$. Then, differently sized expressions (10, 20, 30, …, 300) were generated for each function set, and 30 random expressions were generated for each size. All models (expressions) were then evaluated 50 times, each with the same set of data. The four plots in Fig. 1 represent the average evaluation times of the individuals according to their size and complexity.

Two trends are clearly visible in Fig. 1: (1) for a given size, the higher the functional complexity the greater are the evaluation times; and (2) evaluation times are strongly correlated with the expression sizes, as expected. Hence, the evaluation time indeed discriminates between different functional complexities. However, if a simple function is represented inefficiently by an excessively large expression, it evaluates slowly. Therefore, evaluation time control impacts conditionally: it curbs functional complexity when individual sizes are similar and allows greater sizes for functionally simpler models up to a certain tolerance (or range); however, in the presence of very different sizes (such as during early generations) it curbs growth in size (controls bloat). To estimate the tolerance for greater sizes of simpler models, compare different curves at identical values along y-axis of Fig. 1: for example, at evaluation time = 300 ms, ADD-SUB models can be more than twice as large as COS-SIN models.

The above findings also reveal the limiting behaviour of evaluation time control in GP. In a functionally diverse but size-converged population – where bloat control is impotent – evaluation times discriminate between functional complexities, whereas in a functionally converged but size-diverse population, evaluation times discriminate between sizes.

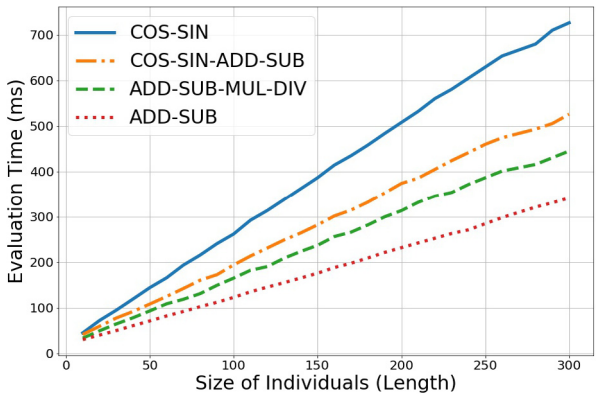

Fig. 1. Relationship between the evaluation time, size and composition of models. Individuals made up of COS and SIN operators have higher average evaluation times than same-sized individuals from other function sets. Furthermore, the size correlates with the evaluation time.

2.4. Measuring evaluation time reliably

Consistency in the measurement of the evaluation times is critical to estimate the complexity of models reliably. However, evaluation times vary across multiple executions, and if this variability is high, the reliability of the complexity estimate is low. However, this variability cannot be completely eliminated because it results from CPU scheduling decisions that are beyond our control. Still, we managed to significantly minimise this variation across evaluations.

In particular, we found that certain CPU management options can help minimise the evaluation time variation. These options include: stopping all background services; locking the CPU speed to prevent the operating system power management from interfering; executing the experiments on dedicated processors; and assigning the experimental tasks a high priority. Fig. 2 illustrates the impact of these options. In the figure, each box-plot represents multiple evaluation times for an individual of a given size. Fig. 2(a) shows that when no CPU management options were applied the variation in the evaluation times was high. In contrast, Fig. 2(b) shows that when the CPU management options were applied the variation clearly decreased. Thus, we were able to use a single evaluation to measure the evaluation time in the later reported experiments.

The time measurements were made using a CPU-time-based function that employs CPU performance counters [47]. This function is available in Python 3.3 and above. The function offers high resolution (in nanoseconds) across platforms, while the returned values are in fractional seconds.

The techniques used to improve time readings may not work in all situations. For example, stabilising time measurements becomes more challenging when the models being evaluated are exceptionally large, like those used in [48], where trees with millions of nodes were evolved. In this case, CPU memory caching comes into play. This is when the processor, while executing a task, has to access CPU memory caches (L1, L2 and L3) having different speeds. Moving from one type of cache to another may introduce delays and inconsistencies in the time measurements. In any case, the achieved improved stability enabled us to explore the proposed concepts and systems.

2.5. Parallel genetic programming

Because we leverage parallel computing for our proposed system, we first review its use in existing GP literature and contrast it with our objectives.

Although the use of parallel computing is not new in GP [49–51], it is typically to improve the run times [52] of GP, as opposed to reducing the complexity of individual models, which is the target of this study. As GP runs can take a long time to complete, parallelising these runs reduces the overall run time. Most commonly, generational replacement schemes parallelise the evaluation of offspring populations. However, the generational replacement requires the *entire* offspring population to be ready before its members can start breeding, which means that all evaluation threads join at a single *point of synchronisation* before the evolution proceeds further. Hence, this parallelisation confers no advantage to simpler individuals and is inefficient in terms of resource utilisation: while a complex individual is taking an excessive amount of time to be evaluated on one CPU, the remaining CPUs stay idle after other, less complex, individuals have been evaluated.

Some EAs have employed asynchronous parallel computing in non-GP setups to alleviate this problem of idle time. For example, an evaluation time bias favouring smaller evaluation times can be observed in [50,53]; however, these studies are concerned with real parameter optimisation using fixed-length chromosomes and do not study the impact of asynchronous parallelism on the functional complexity of variable length structures in GP.

EAs also use parallel computing in the so called *island model* [54,55], where the evolved populations are divided into multiple distinct islands, and the *sub-population* in each island can be evolved in a separate parallel thread. Regardless of parallelisation, the island model offers advantages such as greater diversity in the overall population because each island remains oblivious to other islands except at discrete intervals when selected individuals are sent to other islands. Parallelisation in this model just speeds up the run times.

3. Asynchronous Parallel Genetic Programming (APGP)

The APGP method evaluates GP individuals asynchronously to allow simple individuals that are evaluated faster an opportunity to get into the breeding population prior to their more complex (and still evaluating) counterparts. This mechanism is also natural; after all, natural populations do not simply halt breeding while one of their members is being tested against the environment. Yet, this is precisely what happens in traditional GP: while an individual of any complexity is being evaluated, the evolution stops regardless of how long that evaluation takes. Thus traditional GP confers no advantage to fast evaluations (and hence potentially low complexity). However, APGP aims to leverage the advantage of fast evaluations to breed simple yet accurate models.

APGP is based on *Steady State GP* [56], a replacement scheme in which the offspring immediately competes for a place in the parent population after being evaluated. In APGP steady state replacement allows multiple breeding operations and fitness evaluations to be executed in parallel and asynchronously. For example, a maximum of 50 of such breed operations (followed immediately by the corresponding fitness evaluation; note our setup produces one offspring per breed operation) can be allowed to run at the same time; as soon as one operation finishes, another one starts. However, these operations may finish at different times due to the varying times taken to evaluate different models. This difference is due only to the different make up of models because all models are evaluated on exactly the same dataset. Thus, a race condition develops such that less complex individuals taking less time to be evaluated can *apply* for a place into the population before their slower counterparts; applying for a place means checking if the new model is fitter than the

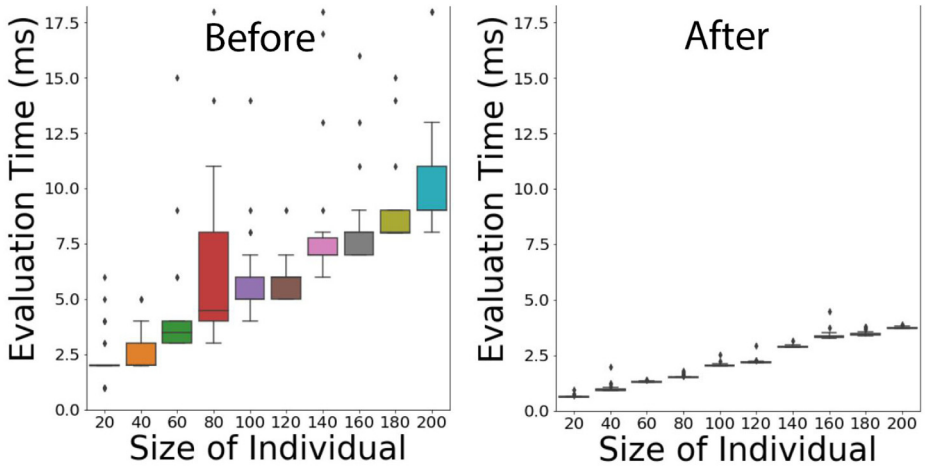

Fig. 2. Measuring evaluation time reliably: applying CPU management options leads to more consistent measurements.

"winner"[2] of an inverse tournament, and if so replacing that winner with the new model. Hence, the fast evaluating individuals may reproduce earlier than the more complex individuals taking longer to be evaluated.

Algorithm 1 lists the pseudo code of the APGP algorithm. The algorithm begins by initialising the number of allowed concurrent evaluations, population size and total number of offspring (total number of fitness evaluations in the run). This is followed by creating the initial population and evaluating it. The parallel breeding then begins by initiating multiple breed operations up to the allowed limit. These breed operations evaluate offspring in independent threads. As soon as an offspring completes evaluating, it replaces the winner of an inverse tournament in the current population if it is fitter than the winner, and the corresponding thread is released to allow another breed operation to commence. As these parallel operations work over the same population, a temporary lock is set on the position of the individual being replaced to avoid clashes.

As discussed above, the decision on whether an offspring finds a place in the population is made based on its accuracy only. As such, speed becomes an advantage only when it is accompanied by high accuracy. Where complex candidates are more accurate, they will eventually get in, get selected and propagate. Thus, complex models are not excluded, and the simplicity of good models is constrained by the possibilities within the specific problem.

Algorithm 1: Asynchronous Parallel Genetic Programming (APGP) Algorithm

```
/* Initialise                                                    */
N ← set total number of offspring to produce;
threadpool ← set number of concurrent operations allowed;
popsize ← set population size;
/* Generate and evaluate initial population                      */
population ← generate initial population of size popsize;
Evaluate(population);
/* Generate and evaluate offspring in parallel                   */
count = 0;
while count < N do
    if threadpool > 0 then
        Thread(threadpool ← threadpool − 1 &&
        offspring ← Breed_and_evaluate() );
        count ← count + 1;
        if offspring_evaluated then
            replace ← To_Replace(population);
            if fitness(offspring) > fitness(population[replace]) then
                Lock population[replace] memory position;
                population[replace] ← offspring;
                Release locked position;
            else
                Discard offspring;
            Release thread: threadpool ← threadpool + 1;
    else
        Wait
end
```

4. Experiments

The performance of APGP was evaluated against standard GP and GP with an effective bloat control mechanism (GP+BC) on a suite of symbolic regression problems. Identical parameters were adopted across the three methods except: the race condition, which was present only in APGP; and the bloat control mechanism, which was present only in GP+BC.

4.1. Test problems

We considered the recommendations provided in [57] when choosing the test problems. Five multi-dimensional problems and one bi-dimensional problem were used. The datasets for Problems 1–5 are described in [58]; Problem 6 is a bi-variate version of the function used in [59]. The datasets are summarised in Table 1. As the results reported in Section 5 show, all but Dow are hard for GP (the accuracy scores are less than 41%). Hence, these problems require GP to run long and thus present a good test bed for evaluating complexity control because complexity in GP grows with the increase in the duration of runs.

4.2. Configuration and parameters

The basic parameters for all the methods are summarised in Table 2. Other key experimental decisions included the following.

[2] The winner of an inverse tournament is the worst member of the tournament; we use an inverse tournament of size 5.

Table 1
Overview of the test problems.

ID	Label	Test problem	No. of variables	No. of instances
1	*Airfoil*	Airfoil self-noise	5	1503
2	*Boston*	Boston housing	13	506
3	*Concrete*	Concrete strength	8	1030
4	*Dow*	Dow chemical	57	1066
5	*Energy*	Energy efficiency	8	768
6	*Y2X6*	$y^2x^6 - 2.13y^4x^4 + y^6x^2$	2	250 (x = min:-0.3, step: 0.012; $y = x + 0.03$)

- *Bloat Control for GP+BC*: *Double tournament* [60] was employed to control bloat. This is a method that has been very successful on a variety of benchmark problems [60,61], and the results of our experiments reported in Section 5 also show that this method indeed limits sizes aggressively. The best problem-independent settings for this method were used as recommended in [60]: in the first round, run n probabilistic tournaments, each with a tournament of size 2, to select a set of n individuals; then, in the second round, select the fittest out of the n individuals. The tournaments in the first round choose the smallest individual with a probability of 0.7. We also considered using Operator Equalisation (OpEq), a more recent bloat control method [62,63]; however, unlike APGP, which uses steady-state replacement, OpEq requires generational replacement.
- *Degree of Concurrency:* The size of the thread pool (number of parallel threads available) can vary in APGP. We tested pool sizes 5, 25, 50, 75 and 100. While size 50 produced the best or competitive results generally, some problem-specific improvement may be possible with other thread sizes. Future work can further investigate this.
- *System Configuration:* The experiments were run on Windows 10 (64-bit) with 32GB RAM, and Intel Core i7-6700 CPU @ 3.40 GHz (quad-core).
- *Additional Configuration and Parameters*: The other key experimental decisions were the following. First, the individuals with divide-by-zero errors were assigned the worst fitness; as discussed in [64], the protected operators commonly used in GP lead to poor generalisation. Second, the datasets were randomly split into training and test sets using the 80:20 ratio. Finally, the fitness was calculated as $\frac{1}{1+\frac{1}{n}\Sigma_{i=1}^{n}(y_i-\hat{y}_i)^2}$ such that y_i and $\hat{y}_i$ represent ideal and model's outputs respectively; the fitness value stays between 0 and 1, where a higher value is better.

Table 2
Summary of parameters used in the tested methods.

Parameter	Setting
Number of runs	50
Population size	500
Run terminates	After 35,000 evaluations (≡ 70 generations)
Random tree/ subtree generation	Ramped half-and-half (depth $min = 1, max = 4$)
Tree depth limit	17
Operators & probabilities	One point crossover = 0.9 Point mutation = 0.1
Function set	$+, -, *, /$, sin, cos, neg
Ephemeral random constants (ERC)	$\|ERC\| = 100$ (min = 0.05, step: 0.05)
Terminal set	{Input variables} U ERC
Selection	tournament size = 3
Replacement	steady state, inverse tournament size = 5

= Favourable to APGP = Not Favourable to APGP = Difference Not Significant

		APGP	GP		GP+BC	
		Mean Values	Mean Values	vs APGP p-values	Mean Values	vs APGP p-values
Airfoil	Evaluation Time:	0.02505	0.02699	2.04E-266	0.02648	3.46E-27
	Size:	226.65	245.71	7.16E-271	167.71	0.00E+00
	Training Fitness:	0.01925	0.01694	1.73E-57	0.01426	0.00E+00
	Test Fitness:	0.02219	0.01976	4.79E-60	0.01634	0.00E+00
Boston	Evaluation Time:	0.01279	0.01316	0.266448	0.01038	0
	Size:	135.52	141.5	0.000884	46.65	0
	Training Fitness:	0.03737	0.03445	0	0.02698	0
	Test Fitness:	0.0511	0.04567	0.00E+00	0.03759	0.00E+00
Concrete	Evaluation Time:	0.02484	0.02638	1.07E-34	0.01308	0.00E+00
	Size:	138.07	144.33	7.43E-17	52.03	0.00E+00
	Training Fitness:	0.00927	0.00819	6.40E-107	0.00698	0.00E+00
	Test Fitness:	0.00803	0.00703	4.5E-133	0.0061	0
Dow	Evaluation Time:	0.01126	0.00951	2.63E-60	0.01049	0.166377
	Size:	121.18	95.8	1.3E-132	44.44	1.89E-07
	Training Fitness:	0.91224	0.90422	0	0.90209	4.56E-05
	Test Fitness:	0.91468	0.90669	0	0.90909	0.006098
Energy	Evaluation Time:	0.01655	0.01672	2.15E-04	0.01399	1.31E-02
	Size:	113.64	117.93	3.84E-19	45.86	1.95E-08
	Training Fitness:	0.16938	0.16521	2.04E-01	0.10305	7.18E-05
	Test Fitness:	0.16488	0.15913	1.26E-05	0.09901	5.90E-05
Y2X6	Evaluation Time:	0.00158	0.00167	2.24E-129	0.00224	3.34E-03
	Size:	54.07	57.67	4.26E-79	32.14	9.74E-07
	Training Fitness:	0.46448	0.41959	9.44E-66	0.33744	5.65E-03
	Test Fitness:	0.39266	0.32868	3.12E-151	0.27194	3.86E-03

Fig. 3. Results of Mann–Whitney U tests on the final populations (APGP vs GP; APGP vs GP+BC). The results show that APGP produced significantly more accurate models (in terms of their training and test fitness) on all datasets compared to both GP and GP+BC. While APGP produced significantly simpler models than GP in five out of six test, GP+BC produced significantly simpler models than APGP at the price of accuracy.

5. Experimental results

To evaluate the performance of the proposed APGP method, we compared the accuracy (fitness) scores of its evolved models over the training and test sets against those of models evolved by standard GP and GP+BC. In addition, we noted the average number of evaluations it took APGP and GP+BC to reach the average training accuracy of GP (target accuracy); this shows the efficiency of the other methods relative to GP.

5.1. Accuracy of the tested methods

As fitness in our experiments was set up as a maximisation function (where higher fitness scores are better), we use the term fitness and accuracy interchangeably. Fig. 3 shows the colour-coded results of the Mann–Whitney U statistical test on the final populations (of models) that were output by APGP, standard GP and GP+BC for all test problems. The results include the evaluation time, size, training and test fitness of the models. The

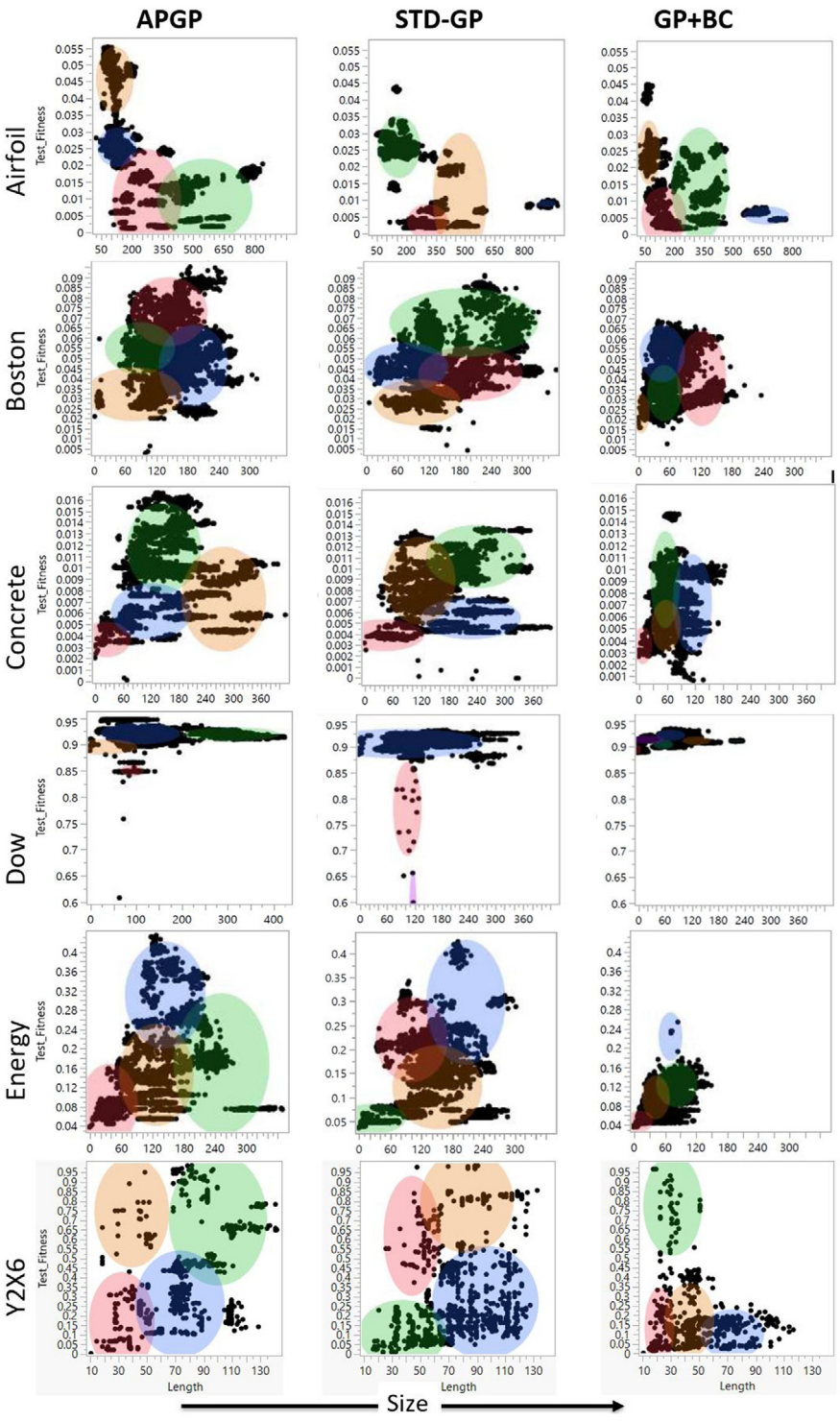

Fig. 4. Scatter-plot Test-fitness by Size of final populations of APGP, GP and GP+BC. The preferred individuals are in the top left corners of the charts (individuals with high test fitness accuracy and small sizes). The clusters discovered by K-means clustering are indicated.

p-values shown in Fig. 3 correspond to the statistical difference between APGP and GP, as well as APGP and GP+BC. The rows coloured in green represent the cases when APGP is significantly better (i.e. outputs less complex models or achieves higher accuracy) than either GP or GP+BC, red when it is significantly worse, and yellow when it is not significantly different.

The dominating green colour in columns 3 and 4 of Fig. 3 indicate that APGP significantly outperforms GP in the vast majority of the tests. In particular, the final population output by APGP is simpler than that output by GP for all the problems except Dow; this is indicated by significantly smaller mean evaluation times and sizes. Furthermore, the training and test accuracy scores (fitness) of APGP are significantly better than those of GP for all the problems except for the training fitness of Energy, where the improvement is not significant. Finally, the APGP test fitness values, which is the major concern when evaluating models, are significantly better than those of GP for all the problems. This indicates that the models output by APGP tend to generalise

Table 3
Comparison of the best individuals that were output by the three considered methods. Positive/negative numbers correspond to the increase/decrease in the corresponding metric values of APGP relative to GP or GP+BC. APGP produced more accurate models than GP and GP+BC for all the problems, however, sometimes, at the expense of longer evaluation times and/or larger model sizes.

Test ID	Test fitness gain	Eval. time increase	Size increase
Relative performance of APGP compared to GP			
Airfoil	+24.99%	−43.26%	−47.17%
Boston	+6.64%	+12.31%	−0.896%
Concrete	+22.43%	−36.99%	−44.96%
Dow	+2.04%	−78.13%	−85.71%
Energy	+3.13%	−22.92%	−18.38%
Y2X6	+0.11%	+42.57%	−11.11%
Relative performance of APGP compared to GP+BC			
Airfoil	+19.93%	+18.00%	+15.0%
Boston	+43.24%	+95.17%	+452.5%
Concrete	+11.56%	+72.70%	+100.0%
Dow	+2.43%	−78.56%	−65.82%
Energy	+69.10%	−34.51%	+40.91%
Y2X6	+1.71%	+49.45%	+695.0%

better on unseen data compared to the models output by GP. It can be noticed that the reported p-values are very small, which indicates that the differences between the results of the two methods are very significant.

While APGP produced more complex individuals than GP for Dow, it demonstrated better training and test fitness values. The relative increase in complexity (size and evaluation time) in this one instance comes with an associated gain in fitness; therefore, it is not necessarily bloat — bloat is growth in size without an associated gain in accuracy.

When comparing APGP with GP+BC, as captured in columns 5 and 6 of Fig. 3, GP+BC produced significantly smaller models for all the problems since it aggressively and solely targets the model size. However, these simpler models achieved significantly lower training and test accuracy values. In other words, simplicity in GP+BC came at the expense of accuracy.

While APGP outperformed GP and GP+BC *on average*, we were also interested in identifying the method that produced the better individuals; GP applications are often interested in the best solutions and not averages. We used the test fitness scores as a criteria for identifying the best of each method. A comparison is summarised in Table 3; the values show how APGP differs from the other compared methods in percentages. It can be noticed from the table that APGP produced individuals with the overall best test accuracy (fitness) values for all the six problems. As noted before, the best models that were output by APGP were also smaller than those output by GP but sometimes larger than GP+BC.

5.2. Cost of producing accurate models

Using the average training accuracy of GP as a benchmark, we compared the average evaluations taken by each method to match that training accuracy. Where a particular run did not achieve that target, it was assigned the maximum number of budgeted evaluations. As summarised in Table 4, APGP used 10% to 40% fewer evaluations than GP, and 15% to 84% fewer than GP+BC. Thus, APGP is the fastest to train, whereas GP+BC despite producing smaller individuals is the slowest; note, as in Fig. 3, both GP and GP+BC are also consistently less accurate than APGP on both training and test sets.

5.3. Constitution of the populations

The proposed APGP is based on the idea that simple models that are competitive will be allowed to thrive and that complex models that are accurate will not be excluded. In this section we examine the makeup of the final populations in terms of the accuracy and simplicity of the individual models.

Scatter-plots that plot test fitness against model-size in the final populations are presented in Fig. 4. Each method has a separate plot for each problem. Columns 1, 2 and 3 are plots for APGP, GP and GP+BC respectively. The plots also have clusters indicated (by ovals) that were identified using K-means algorithm [65].

The GP+BC plots (column 3) show more convergence towards the origin than the other methods. In other words, the aggressive restriction of size by GP+BC also restricts test fitness accuracy. On the other hand, GP allows size to grow with gains in accuracy. Whereas, APGP produced the highest test fitness scores while gently managing complexity.

These trends are similar to those observed in the population in Section 5.1, where it was established that the difference in the populations is statistically significant.

These observations suggest that the best test fitness values are not associated with either extreme of size restriction. They also suggest that APGP normally offers the best trade-off in terms of accuracy and simplicity.

As all the methods show variation within the populations and an appropriate size is problem specific, finding ideal solutions is more a matter of investigation than persistently controlling size. APGP aims to control complexity in an organic approach, where simple solutions can thrive when their training accuracy is better than their more complex counterparts.

From the above results the merit of the ideas behind the APGP system are apparent. However, a questions arises: if we use *time-control* methods that explicitly minimise evaluation time instead of sizes, can we reproduce the previous results without using the race condition that APGP uses to induce a more *implicit* time control? The next section explores this question.

6. Explicit time control

The proposed APGP controls complexity by implicitly controlling evaluation time. Therefore, it offers novelty in two fronts: (1) the idea that evaluation time can serve as a measure of complexity and (2) the race condition that APGP employs to control this indicator of complexity. Here we decouple the two to examine how explicit control of evaluation time compares with APGP. This will also enable us to explore the behaviour of evaluation time.

As in [66] that introduced explicit time control, we use four well-known techniques for bloat-control to now instead control evaluation time; hence, the techniques effect an explicit time-control. However, unlike in [66] that compared time-control with only standard GP methods, here we compare the explicit time-control against a relatively implicit time-control in APGP.

6.1. Techniques for explicit time control

The bloat-control techniques that were adapted to control evaluation time are as follows:

1. *Death by Size* (DS) [60] increases the probability of replacing the larger individuals from the present population. To replace an individual, DS randomly selects two individuals and replaces the larger with a given probability (typically 0.7; we use the same).

Table 4
APGP used significantly fewer evaluations than both GP and GP+BC to reach the same target accuracy. .

No. of evaluations to reach target accuracy					
Test ID	GP mean	GP+BC mean	APGP mean	Difference APGP /GP	Difference APGP /GP+BC
Airfoil	29407	33041	23941	18.59%	38.01%
Boston	20761	27792	17762	14.44%	56.47%
Concrete	31668	33595	20422	35.51%	64.50%
Dow	17861	19762	10719	39.99%	84.36%
Energy	28852	34842	25658	11.07%	35.79%%
Y2X6	33077	33895	29546	10.68%	14.72%

= Favourable to APGP = Not Favourable to APGP = Difference Not Significant

		APGP	DS-TC		DT-TC		OpEq-TC		Tp-TC	
		Mean Values	Mean Values	vs APGP p-values	Mean Values	vs APGP p-values	Mean Values	vs APGP p-values	Mean Values	vs APGP p-values
Airfoil	Evaluation Time:	0.02505	0.00408	0.00E+00	0.00967	0.00E+00	0.02214	7.21E-02	0.00745	0.00E+00
	Size:	226.65	64.03	0.00E+00	154.5	0.00E+00	113.86	0.00E+00	107.01	0.00E+00
	Training Fitness:	0.01925	0.00252	0.00E+00	0.0136	0.00E+00	0.01169	0.00E+00	0.00478	0.00E+00
	Test Fitness:	0.02219	0.00278	0.00E+00	0.01576	0.00E+00	0.01336	0.00E+00	0.00529	0.00E+00
Boston	Evaluation Time:	0.01279	0.00154	0.00E+00	0.00469	0.00E+00	0.01148	0.00E+00	0.00711	0.00E+00
	Size:	135.52	9.41	0.00E+00	58.99	0.00E+00	86.57	0.00E+00	86.31	0.00E+00
	Training Fitness:	0.03737	0.00674	0.00E+00	0.02729	0.00E+00	0.02479	0.00E+00	0.02258	0.00E+00
	Test Fitness:	0.0511	0.00944	0.00E+00	0.03565	0.00E+00	0.03347	0.00E+00	0.03176	0.00E+00
Concrete	Evaluation Time:	0.02484	0.00274	0.00E+00	3.66E-03	0.00E+00	0.0205	1.01E-233	0.007	0.00E+00
	Size:	138.07	28.79	0.00E+00	4.05E+01	0.00E+00	82.3	0.00E+00	86.11	0.00E+00
	Training Fitness:	0.00927	0.0031	0.00E+00	6.87E-03	0.00E+00	0.00594	0.00E+00	0.00465	0.00E+00
	Test Fitness:	0.00803	0.00282	0.00E+00	0.00607	0.00E+00	0.00515	0.00E+00	0.00417	0.00E+00
Dow	Evaluation Time:	0.01126	0.0012	0.00E+00	0.00339	0.00E+00	0.00339	0.00E+00	0.00439	0.00E+00
	Size:	121.18	3	0.00E+00	41.19	0.00E+00	41.19	0.00E+00	59.3	0.00E+00
	Training Fitness:	0.91224	0.60018	0.00E+00	0.89898	0.00E+00	0.89898	0.00E+00	0.75892	0.00E+00
	Test Fitness:	0.91468	0.60631	0.00E+00	0.90303	0.00E+00	0.90303	0.00E+00	0.76499	0.00E+00
Energy	Evaluation Time:	0.01655	0.00126	0.00E+00	0.00428	0.00E+00	0.00149	0.00E+00	0.00724	0.00E+00
	Size:	113.64	4.04	0.00E+00	40.94	0.00E+00	27.27	0.00E+00	76.37	0.00E+00
	Training Fitness:	0.16938	0.04352	0.00E+00	0.09914	0.00E+00	0.3057	0.00E+00	0.05165	0.00E+00
	Test Fitness:	0.16488	0.04099	0.00E+00	0.09512	0.00E+00	0.25417	0.00E+00	0.0495	0.00E+00
Y2X6	Evaluation Time:	0.00158	0.00139	0.00E+00	0.00149	6.91E-04	0.00954	6.91E-04	0.00303	0.00E+00
	Size:	54.07	25.09	0.00E+00	27.27	0.00E+00	77.04	0.00E+00	86.52	0.00E+00
	Training Fitness:	0.46448	0.01339	0.00E+00	0.3057	0.00E+00	0.22432	0.00E+00	0.10232	0.00E+00
	Test Fitness:	0.39266	0.01032	0.00E+00	0.25417	0.00E+00	0.18866	0.00E+00	0.07608	0.00E+00

Fig. 5. APGP vs explicit time-control methods. Detailed here are the results of the test for significance of difference in the final populations of APGP against those of the explicit time control methods. The results show that APGP produced significantly more accurate (training and test fitness) models on all tests against and against all time-control methods. The time-control methods produced simpler models at the cost of accuracy.

2. *Double Tournament* (DT) [60,61] encourages the reproduction of small offspring by increasing the probability of choosing smaller individuals as parents. This is achieved with two rounds of tournaments. In the first round, it runs n probabilistic tournaments each with a tournament of size 2 to select a set of n individuals. Each of these tournaments selects the smaller individual with a probability of 0.7. Then, in the second round, DT selects the fittest out of the n individuals.
3. *Operator Equalisation* (OpEq) [62,63] allows the sizes of individuals to grow only when fitness is improving. It controls the distribution of size in the population by employing two core functions. The first determines the target distribution (by size) of individuals in the next generation; the second ensures that the next generation matches the target distribution. To define the target distribution, OpEq puts the current individuals into bins according to their sizes and calculates the average fitness score of each bin. This average score is then used to calculate the number of individuals to be allowed in a corresponding bin in the next generation (target distribution). Thus, from one generation to the next the target distribution changes to favour the sizes that produce fitter individuals. In our experiments we used Dynamic OpEq, which is the variant that produces higher accuracy [63].
   To adapt OpEq to control evaluation time, we had to estimate the time equivalent of the bin width at the beginning of the run; we used *bin width* = 5 in our experiments.
4. *The Tarpeian* (TP) [19] discourages growth in size by penalising larger individuals in the population and making them noncompetitive. This is effected by calculating the average size of the population at every generation and then assigning the worst fitness to a fraction W of the individuals that have above-average size (recommended $W = 0.3$; we use the same).

Similar to APGP, Death by Size and Double Tournament use Steady-state replacement strategy. However, although Operator

Equalisation and Tarpeian use the generational replacement strategy that differs from the steady state in APGP, we still report their results as benchmarks due to their popularity as bloat control methods.

6.2. Results of comparing APGP with explicit time control

APGP produced more accurate models (both training and test fitness) on all tests against all time-control methods. Fig. 5 details the results of the tests for significance of difference in the final populations. The improvement in accuracy by APGP was significant. Also, similar to the result of comparing APGP with bloat control in Section 5, the control of complexity by APGP was not as aggressive as the explicit time-control methods. The explicit time-control techniques produced significantly simpler models but lost out on accuracy.

Although [66] reported that the explicit control of evaluation time instead of size outperforms the standard GP methods with and without bloat control, these results show that APGP is still more accurate. Thus, the asynchronous parallel breeding in APGP manages complexity in such a way that allows greater accuracy and while keeping complexity less than that in standard GP. In the next section, we explore how evaluation time can be manoeuvred to enhance APGP.

7. APGP for functionally diverse populations

In Section 2.3 we demonstrated that the evaluation time can reflect both the size and functional complexity of models. This finding suggests that when the individuals are identically sized but functionally diverse, then differences in evaluation times will largely reflect the differences in functional or computational complexity of the functions that make up the models. Therefore, restricting evaluation time in this environment will restrict the complexity of the functions that make up the individuals. This in turn may lead to simpler functional behaviour of the models and better generalisation. Therefore, in [66] we proposed a *Fixed-Length Initialisation (FLI)* scheme that can be used to start the evolution with a size converged and functionally diverse population. The results demonstrated that using FLI significantly improves the test fitness on a variety of GP techniques both with bloat-control and time-control. Also, the results indicated that time-control with FLI produces simpler functions.

However, FLI was not tested with APGP in [66]; therefore, we test the impact of FLI on APGP in this paper.

7.1. Fixed Length Initialisation scheme (FLI)

We use FLI to produce an initial population of unique individuals each having the same length (or size) of 10 nodes. Given the functions set size of 7 a length of 10 can easily produce populations of a few hundred unique individuals. Later in Section 7.4, we explored the impact of varying the lengths.

To encourage functional diversity, two individuals that only differ by numeric constants are considered the same.

7.2. Impact of fixed length initialisation on APGP

APGP with FLI (APGP-FLI) produced significantly more accurate models (on both training and test data) than plain APGP on five out of the six test problems; see Fig. 6. In terms of sizes the results were mixed (though APGP-FLI is better 4 out 6 times); however, in terms of evaluation times, APGP-FLI was significantly better than APGP on 5 out of six problems.

= Favourable to APGP-FLI = Not Favourable to APGP-FLI

		APGP-FLI	APGP	
		Mean Values	Mean Values	vs APGP-FLI p-values
Airfoil	Evaluation Time:	0.0137	0.0251	0.00E+00
	Size:	187.08	226.65	6.79E-25
	Training Fitness:	0.01968	0.01925	6.14E-06
	Test Fitness:	0.02251	0.02219	3.62E-04
Boston	Evaluation Time:	0.00766	0.0128	0.00E+00
	Size:	164.52	135.52	0.00E+00
	Training Fitness:	0.03974	0.03737	0.00E+00
	Test Fitness:	0.05269	0.05110	0.00E+00
Concrete	Evaluation Time:	0.01182	0.0248	0.00E+00
	Size:	147.55	138.07	5.50E-49
	Training Fitness:	0.00915	0.00927	6.16E-10
	Test Fitness:	0.00766	0.00803	9.65E-04
Dow	Evaluation Time:	0.00741	0.0113	0.00E+00
	Size:	87.67	121.18	0.00E+00
	Training Fitness:	0.91654	0.91224	5.43E-152
	Test Fitness:	0.91654	0.91468	2.35E-18
Energy	Evaluation Time:	0.01474	0.0166	6.62E-133
	Size:	149.96	113.64	0.00E+00
	Training Fitness:	0.22043	0.16938	0.00E+00
	Test Fitness:	0.21552	0.16488	0.00E+00
Y2X6	Evaluation Time:	0.00257	0.0016	0.00E+00
	Size:	53.66	54.07	2.88E-08
	Training Fitness:	0.48126	0.46448	2.97E-03
	Test Fitness:	0.41305	0.39266	5.49E-05

Fig. 6. Impact of FLI on APGP. Statistical tests showed that apply FLI to APGP improved the mean accuracy values (both training and test) of the population in 5 out of 6 problems. Also there was a reduction in size and evaluation times in 4 out of 6 and 5 out of 6 respectively.

7.3. APGP with fixed length initialisation vs. time control with fixed length initialisation

Next, we compare APGP-FLI against the time-control methods with FLI. For this, we re-ran time-control methods with FLI and tested for a significance in difference in the final populations against APGP-FLI; the results of the Mann-U Whitney tests are captured in Fig. 7.

The results show that APGP-FLI produced significantly more accurate individuals than all the compared time-control techniques with FLI. This is both in terms of training and test fitness accuracy. Out of the twenty-four comparisons of accuracy with APGP-FLI, only once another method outperformed APGP-FLI. Also, the accuracy difference was not significant in one out of twenty-four tests.

As with earlier comparisons with time-control methods, mostly APGP-FLI produced more complex individuals; however, again this complexity also brings higher accuracy on the test data.

7.4. Varying the lengths in the fixed length initialisation

In all the previous FLI experiments all the individuals in the initial generation were made up of ten nodes. Here we explore the impact of varying the initial size on APGP. Therefore, we re-ran the APGP-FLI with ten different sizes from five to fifty (FLI-5 to FLI-50, in steps of 5). Fig. 8 details the results. For each problem, box-plots of the average test fitness values and average lengths of the final populations are shown; the X-axis show the different FLI sizes.

= Favourable to APGP-FLI = Not Favourable to APGP-FLI = Difference Not Significant

		APGP-FLI	DS-TC-FLI		DT-TC-FLI		OPEQ-TC-FLI		TP-TC-FLI	
		Mean Values	Mean Values	vs APGP-FLI p-values	Mean Values	vs APGP-FLI p-values	Mean Values	vs APGP-FLI p-values	Mean Values	vs APGP-FLI p-values
Airfoil	Evaluation Time:	0.0137	0.00500	0.00E+00	0.00864	0.00E+00	0.02737	0.00E+00	0.00745	0.00E+00
	Size:	187.08	80.97	0.00E+00	135.25	0.00E+00	120.75	0.00E+00	107.01	0.00E+00
	Training Fitness:	0.01968	0.00359	0.00E+00	0.01488	0.00E+00	0.0137	0.00E+00	0.00478	0.00E+00
	Test Fitness:	0.02251	0.00397	0.00E+00	0.01718	0.00E+00	0.01578	0.00E+00	0.00529	0.00E+00
Boston	Evaluation Time:	0.00766	0.00388	0.00E+00	0.00343	0.00E+00	0.01450	0.00E+00	0.00787	3.34E-27
	Size:	164.52	42.16	0.00E+00	83.90	0.00E+00	91.46	0.00E+00	91.49	0.00E+00
	Training Fitness:	0.03974	0.01354	0.00E+00	0.03769	7.24E-162	0.02558	0.00E+00	0.02417	0.00E+00
	Test Fitness:	0.05269	0.01966	0.00E+00	0.05424	2.09E-07	0.035	0.00E+00	0.03416	0.00E+00
Concrete	Evaluation Time:	0.01182	0.00366	0.00E+00	0.00680	0.00E+00	0.02666	0.00E+00	0.00956	0.00E+00
	Size:	147.55	41.98	0.00E+00	89.10	0.00E+00	87.03	0.00E+00	88.13	0.00E+00
	Training Fitness:	0.00915	0.00357	0.00E+00	0.00841	4.30E-229	0.00593	0.00E+00	0.005	0.00E+00
	Test Fitness:	0.00766	0.00322	0.00E+00	0.00743	4.49E-71	0.00515	0.00E+00	0.0045	0.00E+00
Dow	Evaluation Time:	0.00741	0.00326	0.00E+00	0.00422	0.00E+00	0.01623	0.00E+00	0.00499	0.00E+00
	Size:	87.67	36.88	0.00E+00	55.44	0.00E+00	55.72	0.00E+00	61.19	0.00E+00
	Training Fitness:	0.91654	0.71385	0.00E+00	0.9118	1.40E-109	0.7579	0.00E+00	0.78253	0.00E+00
	Test Fitness:	0.91654	0.72139	0.00E+00	0.91575	7.88E-02	0.76226	0.00E+00	0.79031	0.00E+00
Energy	Evaluation Time:	0.01474	0.00193	0.00E+00	0.00428	0.00E+00	0.02931	1.27E-02	0.00956	0.00E+00
	Size:	149.96	13.69	0.00E+00	40.94	0.00E+00	89.22	0.00E+00	97.27	0.00E+00
	Training Fitness:	0.22043	0.05655	0.00E+00	0.09914	0.00E+00	0.08433	0.00E+00	0.06431	0.00E+00
	Test Fitness:	0.21552	0.05381	0.00E+00	0.09512	0.00E+00	0.08261	0.00E+00	0.0621	0.00E+00
Y2X6	Evaluation Time:	0.00257	0.00148	0.00E+00	0.00153	0.00E+00	0.00954	0.00E+00	0.00298	0.00E+00
	Size:	53.66	27.86	0.00E+00	30.21	0.00E+00	77.04	1.88E-19	81.39	0.00E+00
	Training Fitness:	0.48126	0.02694	0.00E+00	0.28112	0.00E+00	0.22432	0.00E+00	0.11028	0.00E+00
	Test Fitness:	0.41305	0.0204	0.00E+00	0.23543	0.00E+00	0.18866	0.00E+00	0.08446	0.00E+00

Fig. 7. Results of Mann–Whitney U test for significance in the differences between the final populations of APGP with FLI and explicit time-control with FLI. APGP produced more accurate solutions (both training and test) in all cases. Explicit time-control produced simpler (smaller sizes and evaluation times) models than APGP in xx out of 24 tests.

The lengths of individuals in the final populations are not affected significantly despite varying FLI lengths. However, there is some activity in test-fitness: with the exception of Boston, the results show that starting with individuals of size five (5 nodes each) appears disadvantageous. The lengths that produced the best results are problem specific. However, it is reassuring to note that relatively lower lengths in FLI-10, FLI-15 and FLI-20 are at least competitive with the greater lengths.

7.5. Summary of APGP in a functionally diverse population

The results in this section show that FLI significantly improved APGP. Similarly, our related work [66] showed that FLI improved the performance of time-control methods. However, APGP retained its leading position in the environment that FLI creates.

The analysis of time in Section 2.3 indicated that in a functionally diverse and size converged environment, controlling time will distinguish complexity based more on composition than on size; FLI creates such an environment at the beginning of the evolution but does not control the remaining generations. Future work will study if encouraging such an environment in the remaining evolution will further intensify the effect of time-control.

8. Conclusions and future work

This paper exemplifies redefining complexity in GP with the evaluation time. The evaluation time is both a function of the size, as well as functional and computational effort of a model. Although we only applied the measure here to regression problems, it is also applicable to other domains.

A criticism of the proposed evaluation time measure is the variability in its repeated measurements. To address this issue, the paper demonstrated a way to minimise this variability, so that the evaluation time could be measured reliably.

Leveraging evaluation times, this paper further presented a novel method, called Asynchronous Parallel GP (APGP), which uses asynchronous parallel computing to induce a race between concurrent executions of multiple models to discourage complexity. According to this race condition, models are allowed to join a steady-state population as soon as they are evaluated rather than waiting for other models to be evaluated. This condition provides an evolutionary advantage to simpler models, that are also competitive in terms of accuracy. APGP thus challenges the conventional, but ultimately unnatural, way of evaluating individuals one after the other or in a lock-step.

Unlike aggressive bloat control techniques that penalise sizes at the expense of achieving lower accuracy and hence increasing training times, APGP produces models that are simpler than those from standard GP and are more accurate than those from GP both with or without complexity control; complexity was controlled with standard bloat control methods or their time variants that we defined as *time-control* methods. Against standard GP, APGP improved test fitness accuracy on 6/6 test; the solutions were also significantly simpler in 5/6 tests, both in terms of size and evaluation times. Against GP with bloat control (GP+BC), APGP improved test fitness accuracy on 6/6 test; but the solutions were not simpler than BC (in 3/6 for evaluation times and 6/6 for size). Thus, instead of aggressively going after complexity control, the APGP decreases complexity but not to the extent that it starts hurting accuracy on test data.

The APGP trained faster than GP and GP+BC on all 6 problems; it reached the average training scores of GP with 11% to 40%

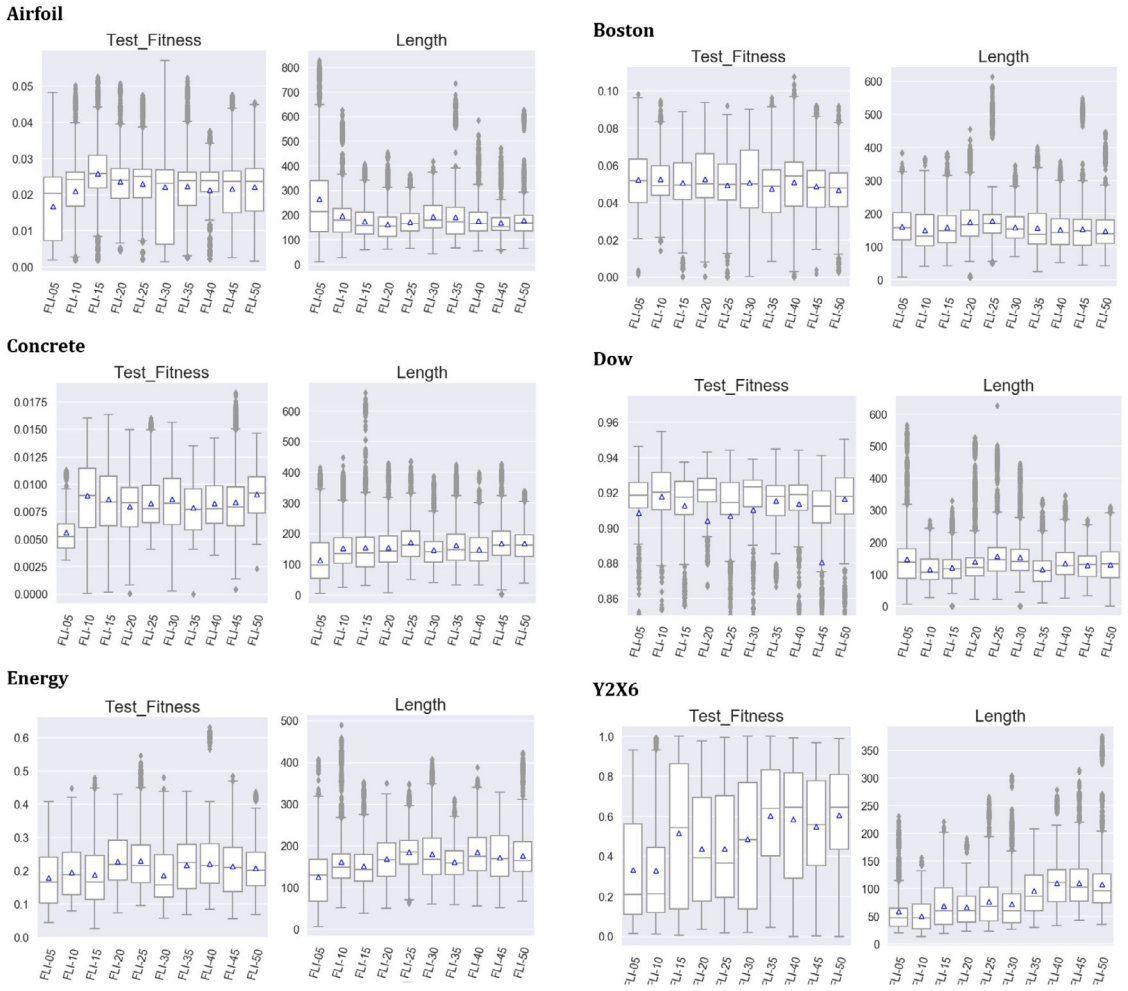

Fig. 8. Impact of changing the sizes of individuals for FLI. Despite varying FLI lengths, the lengths of individuals in the final populations has remained fairly stable. However, some changes were observed with test-fitness accuracy.

fewer evaluations than GP and 15% to 84% fewer evaluations than GB+BC.

When APGP was compared against the explicit control of evaluation time using four well-known and effective bloat control techniques, APGP produced significantly more accurate solutions on all tests (24/24). The explicit control methods controlled complexity more aggressively; they produced significantly simpler models in all tests but 1/24 for size and 3/24 for evaluation times. Again, this shows the merit of APGP's approach of not directly targeting and penalising the complexity of models but rather encouraging simplicity in competitive models.

Motivated by the idea that the evaluation time can distinguish functional complexity in a functionally diverse and size-converged environment, we tested APGP with a new and effective initialisation scheme called the Fixed Length Initialisation (FLI). The FLI improved APGP even further: The test set accuracy increased significantly in 5/6 tests; and the complexity decreased significantly in 8/12 tests (3/6 for size and 5/6 for evaluation times).

We also tested the sensitivity of the FLI to different initial lengths and we found that the performance of APGP remains largely robust against the variation. However, usefully, the relatively smaller lengths of 10, 15, and 20 produced at least as good performance as the greater lengths. While FLI allows functional complexity to be detected in the earliest generations, future work can explore similar possibilities throughout the evolution.

In summary, the results of the experiments presented in this paper demonstrated that APGP has the potential for generating simple and accurate models fast. While this study focused on regression problems only, in principle, the evaluation time can represent complexity in other domains as well. In the future, we plan to test the proposed method on classification problem with discrete fitness values and non-machine-learning problems such as automatic programming and design.

CRediT authorship contribution statement

Aliyu Sani Sambo: Conceptualization, Software, Investigation, Methodology, Writing - original draft. **R. Muhammad Atif Azad:** Conceptualization, Methodology, Supervision, Writing - review & editing. **Yevgeniya Kovalchuk:** Supervision, Writing - review & editing. **Vivek Padmanaabhan Indramohan:** Supervision. **Hanifa Shah:** Supervision.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

References

[1] G. Paris, D. Robilliard, C. Fonlupt, Exploring overfitting in genetic programming, in: P. Liardet, P. Collet, C. Fonlupt, E. Lutton, M. Schoenauer (Eds.), Evolution Artificielle, 6th International Conference, in: Lecture Notes in Computer Science, 2936, Springer, Marseilles, France, 2003, pp. 267–277, http://dx.doi.org/10.1007/b96080, Revised Selected Papers.

[2] J.R. Koza, Genetic programming: On the programming of computers by means of natural selection, MIT Press, Cambridge, MA, USA, 1992, http://mitpress.mit.edu/books/genetic-programming.

[3] Z.C. Lipton, The mythos of model interpretability, Commun. ACM 61 (10) (2018) 36–43, http://dx.doi.org/10.1145/3233231, http://doi.acm.org/10.1145/3233231.

[4] J. Hatwell, M.M. Gaber, R.M.A. Azad, CHIRPS: Explaining random forest classification, Artif. Intell. Rev. (2020) 1–42, http://dx.doi.org/10.1007/s10462-020-09833-6, https://doi.org/10.1007/s10462-020-09833-6.

[5] A. Kumar, S. Goyal, M. Varma, Resource-efficient machine learning in 2 KB RAM for the internet of things, in: D. Precup, Y.W. Teh (Eds.), Proceedings of the 34th International Conference on Machine Learning, in: Proceedings of Machine Learning Research, 70, PMLR, International Convention Centre, Sydney, Australia, 2017, pp. 1935–1944.

[6] M. Couture, Complexity and Chaos-State-Of-The-Art; Formulations and Measures of Complexity, Tech. rep., Defence research and development Canada Valcartier (QUEBEC), 2007.

[7] J.R. Koza, Genetic Programming: On the Programming of Computers by Means of Natural Selection, MIT Press, Cambridge, MA, USA, 1992, http://mitpress.mit.edu/books/genetic-programming.

[8] T. Soule, J.A. Foster, J. Dickinson, Code growth in genetic programming, in: J.R. Koza, D.E. Goldberg, D.B. Fogel, R.L. Riolo (Eds.), Genetic Programming 1996: Proceedings of the First Annual Conference, MIT Press, Stanford University, CA, USA, 1996, pp. 215–223, http://cognet.mit.edu/sites/default/files/books/9780262315876/pdfs/9780262315876_chap26.pdf.

[9] L. Vanneschi, M. Castelli, S. Silva, Measuring bloat, overfitting and functional complexity in genetic programming, in: J.B. et al (Ed.), GECCO '10: Proceedings of the 12th Annual Conference on Genetic and Evolutionary Computation, ACM, Portland, Oregon, USA, 2010, pp. 877–884, http://dx.doi.org/10.1145/1830483.1830643.

[10] R.M.A. Azad, C. Ryan, A simple approach to lifetime learning in genetic programming based symbolic regression, Evol. Comput. 22 (2) (2014) 287–317, http://dx.doi.org/10.1162/EVCO_a_00111, http://www.mitpressjournals.org/doi/abs/10.1162/EVCO_a_00111.

[11] J.R. Koza, Human-competitive machine invention by means of genetic programming, Artif. Intell. Eng. Des. Anal. Manuf. 22 (3) (2008) 185–193, http://dx.doi.org/10.1017/S0890060408000127.

[12] N.X. Hoai, R.I.B. McKay, D. Essam, Representation and structural difficulty in genetic programming, IEEE Trans. Evol. Comput. 10 (2) (2006) 157–166, http://dx.doi.org/10.1109/TEVC.2006.871252, http://sc.snu.ac.kr/courses/2006/fall/pg/aai/GP/nguyen/Structdiff.pdf.

[13] C. Ryan, M. O'Neill, J.J. Collins (Eds.), Handbook of Grammatical Evolution, Springer, 2018, http://dx.doi.org/10.1007/978-3-319-78717-6.

[14] R.M.A. Azad, A Position Independent Representation for Evolutionary Automatic Programming Algorithms - The Chorus System (Ph.D. thesis), University of Limerick, Ireland, 2003, http://www.cs.ucl.ac.uk/staff/W.Langdon/ftp/papers/azad_thesis.ps.gz.

[15] G. Chennupati, R.M.A. Azad, C. Ryan, Performance optimization of multi-core grammatical evolution generated parallel recursive programs, in: S.S. et al (Ed.), GECCO '15: Proceedings of the 2015 Annual Conference on Genetic and Evolutionary Computation, ACM, Madrid, Spain, 2015, pp. 1007–1014, http://dx.doi.org/10.1145/2739480.2754746, http://doi.acm.org/10.1145/2739480.2754746.

[16] L. Spector, A. Robinson, Genetic programming and autoconstructive evolution with the push programming language, Genet. Program. Evol. Mach. 3 (1) (2002) 7–40, http://dx.doi.org/10.1023/A:1014538503543, http://hampshire.edu/lspector/pubs/push-gpem-final.pdf.

[17] T. Hu, J. Payne, W. Banzhaf, J. Moore, Evolutionary dynamics on multiple scales: a quantitative analysis of the interplay between genotype, phenotype, and fitness in linear genetic programming, Genet. Program. Evol. Mach. 13 (3) (2012) 305–337, http://dx.doi.org/10.1007/s10710-012-9159-4, Special issue on selected papers from the 2011 European conference on genetic programming.

[18] J.A. Walker, J.F. Miller, The automatic acquisition, evolution and reuse of modules in cartesian genetic programming, IEEE Trans. Evol. Comput. 12 (4) (2008) 397–417, http://dx.doi.org/10.1109/TEVC.2007.903549, http://results.ref.ac.uk/Submissions/Output/3354578.

[19] R. Poli, A simple but theoretically-motivated method to control bloat in genetic programming, in: C. Ryan, T. Soule, M. Keijzer, E. Tsang, R. Poli, E. Costa (Eds.), Genetic Programming, Proceedings of EuroGP'2003, in: LNCS, 2610, Springer-Verlag, Essex, 2003, pp. 204–217, http://dx.doi.org/10.1007/3-540-36599-0_19.

[20] A. Ekart, S.Z. Nemeth, Selection based on the Pareto nondomination criterion for controlling code growth in genetic programming, Genet. Programm. Evol. Mach. 2 (1) (2001) 61–73, http://dx.doi.org/10.1023/A:1010070616149.

[21] S. Luke, L. Panait, A comparison of bloat control methods for genetic programming, Evol. Comput. 14 (3) (2006) 309–344, http://dx.doi.org/10.1162/evco.2006.14.3.309, http://cognet.mit.edu/system/cogfiles/journalpdfs/evco.2006.14.3.309.pdf.

[22] S. Dignum, R. Poli, Crossover, sampling, bloat and the harmful effects of size limits, in: European Conference on Genetic Programming, Springer, 2008, pp. 158–169.

[23] N.F. McPhee, A. Jarvis, E.F. Crane, On the strength of size limits in linear genetic programming, in: K. Deb (Ed.), Genetic and Evolutionary Computation – GECCO 2004, Springer Berlin Heidelberg, Berlin, Heidelberg, 2004, pp. 593–604.

[24] S. Silva, S. Dignum, L. Vanneschi, Operator equalisation for bloat free genetic programming and a survey of bloat control methods, Genetic Program. Evol. Mach. 13 (2) (2012) 197–238, http://dx.doi.org/10.1007/s10710-011-9150-5.

[25] A.N. Kolmogorov, Three approaches to the quantitative definition ofinformation', Probl. Inf. Transm. 1 (1) (1965) 1–7.

[26] T.M. Cover, J. Thomas, Joint entropy and conditional entropy, in: Elements of Information Theory, second ed., John Wiley & Sons: Hoboken, NJ, USA, 2006, p. 16.

[27] P. Vitányi, How incomputable is Kolmogorov complexity?, Entropy 22 (4) (2020) 408.

[28] A.K. Zvonkin, L.A. Levin, The complexity of finite objects and the development of the concepts of information and randomness by means of the theory of algorithms, Russian Math. Surveys 25 (6) (1970) 83.

[29] J. Rissanen, Modeling by shortest data description, Automatica 14 (5) (1978) 465–471.

[30] V. Nannen, A short introduction to model selection, Kolmogorov complexity and minimum description length (MDL), 2010, arXiv preprint arXiv:1005.2364.

[31] H. Iba, H. de Garis, T. Sato, Genetic programming using a minimum description length principle, in: K.E. Kinnear, Jr. (Ed.), Advances in Genetic Programming, MIT Press, Cambridge, MA, USA, 1994, pp. 265–284, http://cognet.mit.edu/sites/default/files/books/9780262277181/pdfs/9780262277181_chap12.pdf.

[32] N.N. Schraudolph, J.J. Grefenstette, A user's guide to GAucsd 1.4, in: Computer Science & Engineering Department, University of California, San Diego, 1992, p. 1991.

[33] E.J. Vladislavleva, G.F. Smits, D. den Hertog, Order of nonlinearity as a complexity measure for models generated by symbolic regression via Pareto genetic programming, IEEE Trans. Evol. Comput. 13 (2) (2009) 333–349, http://dx.doi.org/10.1109/TEVC.2008.926486.

[34] T. Rivlin, The Chebyshev Polynomials, in: A Wiley-Interscience publication, Wiley, 1974.

[35] M. Castelli, L. Manzoni, S. Silva, L. Vanneschi, A quantitative study of learning and generalization in genetic programming, in: S. Silva, J.A. Foster, M. Nicolau, M. Giacobini, P. Machado (Eds.), Proceedings of the 14th European Conference on Genetic Programming, EuroGP 2011, in: LNCS, 6621, Springer Verlag, Turin, Italy, 2011, pp. 25–36, http://dx.doi.org/10.1007/978-3-642-20407-4_3.

[36] S.R. Kulkarni, G. Harman, Statistical learning theory: a tutorial, Wiley Interdiscip. Rev. Comput. Stat. 3 (6) (2011) 543–556.

[37] V.N. Vapnik, Statistical learning theory, in: Adaptive and learning systems for signal processing, communications, and control, Wiley, New York, NY, 1998, OCLC: 845016043.

[38] V. Vapnik, The Nature of Statistical Learning Theory, in: Information Science and Statistics, Springer New York, 2013.

[39] V. Vapnik, Statistical learning theory. 1998, Vol. 3, Wiley, New York, 1998.

[40] Q. Chen, M. Zhang, B. Xue, Structural risk minimisation-driven genetic programming for enhancing generalisation in symbolic regression, IEEE Trans. Evol. Comput. 23 (4) (2019) 703–717, http://dx.doi.org/10.1109/TEVC.2018.2881392.

[41] Q. Chen, M. Zhang, B. Cue, Improving generalisation of genetic programming for symbolic regression with structural risk minimisation, in: T. Friedrich (Ed.), GECCO '16: Proceedings of the 2016 Annual Conference on Genetic and Evolutionary Computation, ACM, Denver, USA, 2016, pp. 709–716, http://dx.doi.org/10.1145/2908812.2908842.

[42] C. Raymond, Q. Chen, B. Xue, M. Zhang, Genetic programming with rademacher complexity for symbolic regression, in: C.A.C. Coello (Ed.), 2019 IEEE Congress on Evolutionary Computation, CEC 2019, IEEE Press, Wellington, New Zealand, 2019, pp. 2657–2664, http://dx.doi.org/10.1109/CEC.2019.8790341.

[43] R.M.A. Azad, C. Ryan, Variance based selection to improve test set performance in genetic programming, in: N. Krasnogor, P.L. Lanzi, A. Engelbrecht, D. Pelta, C. Gershenson, G. Squillero, A. Freitas, M. Ritchie, M. Preuss, C. Gagne, Y.S. Ong, G. Raidl, M. Gallager, J. Lozano, C. Coello-Coello,

D.L. Silva, N. Hansen, S. Meyer-Nieberg, J. Smith, G. Eiben, E. Bernado-Mansilla, W. Browne, L. Spector, T. Yu, J. Clune, G. Hornby, M.-L. Wong, P. Collet, S. Gustafson, J.-P. Watson, M. Sipper, S. Poulding, G. Ochoa, M. Schoenauer, C. Witt, A. Auger (Eds.), GECCO '11: Proceedings of the 13th Annual Conference on Genetic and Evolutionary Computation, ACM, Dublin, Ireland, 2011, pp. 1315–1322, http://dx.doi.org/10.1145/2001576.2001754.

[44] J. Ni, P. Rockett, Tikhonov regularization as a complexity measure in multiobjective genetic programming, IEEE Trans. Evol. Comput. 19 (2) (2014) 157–166.

[45] A.S. Sambo, R.M.A. Azad, Y. Kovalchuk, V.P. Indramohan, H. Shah, Leveraging asynchronous parallel computing to produce simple genetic programming computational models, in: Proceedings of the 35th Annual ACM Symposium on Applied Computing, in: SAC '20, Association for Computing Machinery, New York, NY, USA, 2020, pp. 521–528, http://dx.doi.org/10.1145/3341105.3373921, https://doi.org/10.1145/3341105.3373921.

[46] F.F. de Vega, G. Olague, D. Lanza, W. Banzhaf, E. Goodman, J. Menendez-Clavijo, A. Martinez, et al., Time and individual duration in genetic programming, IEEE Access 8 (2020) 38692–38713.

[47] C. Simpson, J. Jewett, S. Turnbull, V. Stinner, PEP 418: Add monotonic time, performance counter, and process time functions, Website, https://www.python.org/dev/peps/pep-0418/.

[48] W.B. Langdon, Genetic Improvement of Genetic Programming, in: 2020 IEEE Congress on Evolutionary Computation (CEC), 2020, pp. 1–8, doi:10.1109/CEC48606.2020.9185771.

[49] J.R. Koza, D. Andre, Parallel Genetic Programming on a Network of Transputers, Technical Report CS-TR-95-1542, Stanford University, Department of Computer Science, 1995, http://www.genetic-programming.com/jkpdf/tr1542parallelsuversion.pdf.

[50] E.O. Scott, K.A. De Jong, Evaluation-time bias in asynchronous evolutionary algorithms, in: T. Tusar, B. Naujoks (Eds.), GECCO'15 Student Workshop, ACM, Madrid, Spain, 2015, pp. 1209–1212, http://dx.doi.org/10.1145/2739482.2768482.

[51] J. Kim, J. Kim, S. Yoo, GPGPGPU: Evaluation of parallelisation of genetic programming using GPGPU, in: T. Menzies, J. Petke (Eds.), Proceedings of the 9th International Symposium on Search Based Software Engineering, SSBSE 2017, in: LNCS, 10452, Springer, Paderborn, Germany, 2017, pp. 137–142, http://dx.doi.org/10.1007/978-3-319-66299-2_11.

[52] M. Oussaidène, B. Chopard, O.V. Pictet, M. Tomassini, Parallel genetic programming and its application to trading model induction, Parallel Comput. 23 (8) (1997) 1183–1198, http://dx.doi.org/10.1016/S0167-8191(97)00045-8.

[53] E.O. Scott, K.A. De Jong, Evaluation-time bias in quasi-generational and steady-state asynchronous evolutionary algorithms, in: T.F. et al (Ed.), GECCO '16: Proceedings of the 2016 on Genetic and Evolutionary Computation Conference, ACM, Denver, USA, 2016, pp. 845–852, http://dx.doi.org/10.1145/2908812.2908934.

[54] E. Cantú-Paz, A survey of parallel genetic algorithms, Calc. Paralleles Res. Syst. Repar. 10 (2) (1998) 141–171.

[55] D. Power, C. Ryan, R.M.A. Azad, Promoting diversity using migration strategies in distributed genetic algorithms, in: 2005 IEEE Congress on Evolutionary Computation, 2, IEEE Press, Edinburgh, Scotland, UK, 2005, pp. 1831–1838, http://dx.doi.org/10.1109/CEC.2005.1554910.

[56] G. Syswerda, A study of reproduction in generational and steady-state genetic algorithms, in: Foundations of Genetic Algorithms, 1, Elsevier, Amsterdam, 1991, pp. 94–101.

[57] D.R. White, J. McDermott, M. Castelli, L. Manzoni, B.W. Goldman, G. Kronberger, W. Jaskowski, U.-M. O'Reilly, S. Luke, Better GP benchmarks: community survey results and proposals, Genet. Program. Evol. Mach. 14 (1) (2013) 3–29, http://dx.doi.org/10.1007/s10710-012-9177-2.

[58] D. Dua, E. Karra Taniskidou, UCI Machine Learning Repository, University of California, Irvine, School of Information and Computer Sciences, 2017, http://archive.ics.uci.edu/ml.

[59] S. Gustafson, E.K. Burke, N. Krasnogor, On improving genetic programming for symbolic regression, in: D.C. et al (Ed.), Proceedings of the 2005 IEEE Congress on Evolutionary Computation, 1, IEEE Press, Edinburgh, Scotland, UK, 2005, pp. 912–919.

[60] S. Luke, L. Panait, A comparison of bloat control methods for genetic programming, Evol. Comput. 14 (3) (2006) 309–344, http://dx.doi.org/10.1162/evco.2006.14.3.309.

[61] S. Luke, L. Panait, Fighting bloat with nonparametric parsimony pressure, in: J.J. Merelo-Guervos, P. Adamidis, H.-G. Beyer, J.-L. Fernandez-Villacanas, H.-P. Schwefel (Eds.), Parallel Problem Solving from Nature - PPSN VII, in: Lecture Notes in Computer Science, LNCS, (2439) Springer-Verlag, Granada, Spain, 2002, pp. 411–421, http://dx.doi.org/10.1007/3-540-45712-7_40.

[62] S. Dignum, R. Poli, Operator equalisation and bloat free GP, in: M.O. et al (Ed.), Proceedings of the 11th European Conference on Genetic Programming, EuroGP 2008, in: Lecture Notes in Computer Science, 4971, Springer, Naples, 2008, pp. 110–121, http://dx.doi.org/10.1007/978-3-540-78671-9_10.

[63] S. Silva, S. Dignum, L. Vanneschi, Operator equalisation for bloat free genetic programming and a survey of bloat control methods, Genet. Program. Evol. Mach. 13 (2) (2012) 197–238, http://dx.doi.org/10.1007/s10710-011-9150-5.

[64] M. Keijzer, Improving symbolic regression with interval arithmetic and linear scaling, in: European Conference on Genetic Programming, Springer, EuroGP, Essex, UK, 2003, pp. 70–82.

[65] T. Kanungo, D.M. Mount, N.S. Netanyahu, C.D. Piatko, R. Silverman, A.Y. Wu, An efficient k-means clustering algorithm: analysis and implementation, IEEE Trans. Pattern Anal. Mach. Intell. 24 (7) (2002) 881–892.

[66] A.S. Sambo, R.M.A. Azad, Y. Kovalchuk, V.P. Indramohan, H. Shah, Time control or size control? Reducing complexity and improving accuracy of genetic programming models, in: European Conference on Genetic Programming (Part of EvoStar), Springer, 2020, pp. 195–210, http://dx.doi.org/10.1007/978-3-030-44094-7_13.