Contents lists available at ScienceDirect

Information Sciences

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

A study of dynamic populations in geometric semantic genetic programming

Davide Farinati [*], Illya Bakurov, Leonardo Vanneschi

NOVA Information Management School (NOVA IMS), Universidade Nova de Lisboa, Campus de Campolide, 1070-312 Lisboa, Portugal

A R T I C L E I N F O

Keywords:
Dynamic populations
Genetic programming
Geometric semantic genetic programming
Semantic neighbourhood

A B S T R A C T

Allowing the population size to variate during the evolution can bring advantages to evolutionary algorithms (EAs), retaining computational effort during the evolution process. Dynamic populations use computational resources wisely in several types of EAs, including genetic programming. However, so far, a thorough study on the use of dynamic populations in Geometric Semantic Genetic Programming (GSGP) is missing. Still, GSGP is a resource-greedy algorithm, and the use of dynamic populations seems appropriate. This paper adapts algorithms to GSGP to manage dynamic populations that were successful for other types of EAs and introduces two novel algorithms. The novel algorithms exploit the concept of semantic neighbourhood. These methods are assessed and compared through a set of eight regression problems. The results indicate that the algorithms outperform standard GSGP, confirming the suitability of dynamic populations for GSGP. Interestingly, the novel algorithms that use semantic neighbourhood to manage variation in population size are particularly effective in generating robust models even for the most difficult of the studied test problems.

1. Introduction

Evolutionary Algorithms (EAs) are bioinspired population-based methods of Computational Intelligence [1]. The effects of hyperparameterisation on EAs have been studied since the dawn of the field [2]. In succinct terms, two main approaches to hyperparameter setting exist: hyperparameter tuning and hyperparameter control. The former consists of testing different hyperparameters in a restrained manner, for instance, using cross-validation and then using the configuration that led to the best results to solve the underlying problem. The latter consists of dynamically changing the hyperparameter values throughout the evolutionary process. Following the taxonomy proposed in [2], this can be done in at least two ways: deterministically and adaptively. Deterministic hyperparameter control relies upon some time-dependent function to update hyperparameters. Adaptive hyperparameter control, instead, adapts the hyperparameter values based on the information obtained from the evolutionary process itself, so the values are modified depending on the feedback from evolutionary dynamics. A significant amount of work suggests the superiority of hyperparameter control over tuning [2]. Traditionally, both practitioners and researchers focus on hyperparameters related to genetic operators and selection algorithms [3,4]. However, the population size plays a significant role in the algorithms' performance [5,6]. If the population size is too small, the evolutionary process may quickly get stuck in a sub-optimal solution, typically due to diversity loss [7,8]; if it is too large, an unnecessarily large amount of computational resources may be wastefully used to solve the problem. Population

* Corresponding author.
E-mail address: dfarinati@novaims.unl.pt (D. Farinati).

https://doi.org/10.1016/j.ins.2023.119513
Received 23 February 2023; Received in revised form 30 June 2023; Accepted 9 August 2023

Available online 14 August 2023

size control soon became a reality in EAs, and several works aimed at presenting deterministic and adaptive systems for dynamic population size control have been proposed (many of which are discussed in Section 2).

Genetic Programming (GP) [9] is an EA that evolves computer programs to solve a specific task. Typically, the solutions in a GP population are complex structures, whose dynamic dimension often tends to grow during the evolution [10,11]. In such a context, any method aimed at saving computational effort without compromising performance should be beneficial for GP. In fact, although less frequently than for other types of EAs, deterministic and adaptive dynamic methods to control the population size have also been applied to GP, with encouraging results (also these previous studies are discussed in Section 2). However, to the best of our knowledge, an extensive study of dynamic populations has never been performed for one of the most promising developments of GP: Geometric Semantic GP (GSGP) [12,13]. GSGP is a variant of GP in which standard crossover and mutation are replaced by so-called geometric semantics operators (GSOs). Exploiting semantic awareness, GSOs have the interesting property of inducing a unimodal error surface on any supervised learning problem, bestowing on GSGP a noteworthy optimisation power [13,14]. However, GSOs have the drawback of generating offspring that are always larger than their parents. Thus, the dimension of the individuals in a GSGP population steadily grows during the evolution, and many previous works have proposed techniques to decrease the computational effort or speed up the algorithm [14–19]. In this context, developing methods to dynamically update the population size in GSGP, both deterministically and adaptively, looks like a reasonable next research step. This is particularly true considering that the population size has already been recognized as a crucial hyperparameter for GSGP [20].

In this paper, we introduce methods that dynamically variate the population size of GSGP. The motivation for this study is that GSGP can (possibly more than other methods) benefit from a wise use of computational resources. Some of the methods that we propose are adaptions to GSGP of already existing algorithms that are promising for GP or other forms of EAs. Others are novel use the concept of semantic neighbourhood, well-known in the literature [50]. Our objective is to allow GSGP to save a significant amount of computational effort, while maintaining its excellent optimisation ability and its attractive properties (like exploiting semantic awareness and inducing a unimodal error surface). The main contributions of this work are:

- Extending dynamic population systems to GSGP, confirming their appropriateness also for this GP variant.
- Introducing novel dynamic population systems, specifically for GSGP, able to outperform preexisting systems.
- Demonstrating that dynamic populations can help GSGP use computational resources more effectively, while still maintaining the interesting properties that made GSGP popular.

The paper is organized as follows: in Section 2, we revise the existing literature in the area of dynamic populations in EAs. At the end of that section, a special focus is dedicated to the analysis of previous work employing dynamic populations in GP. Section 3 introduces GSGP, including the definition of GSOs, and also discusses the concept of semantic neighbourhood. Section 4 describes the dynamic population algorithms studied in this work. In Section 5, we first present the case studies on which the algorithms' performance will be tested. Those case studies consist of five real-life symbolic regression problems, plus three theoretically hand-tailored symbolic regression benchmarks. Then, the section discusses the parameter set used in our study, with the objective of fostering complete replicability of our experiments. Section 6 presents and discusses the obtained experimental results. Finally, Section 7 concludes the work and suggests ideas for future research.

2. Previous and related work

The first efforts at developing EAs with dynamic population size date back to the 1990s. For instance, based on the hypothesis that different stages of the evolutionary process might require different appropriate population sizes, Arabas et al. [21] proposed a novel heuristic for the dynamic adaptation of population size in genetic algorithms (GAs), called GAVaPS. GAVaPS introduced the concept of the candidate solutions' lifetime, which was meant to replace the selection mechanism. The authors based their rationale on the ageing process, common to natural environments. Three strategies were explored for solutions' lifetime estimation. GAVaPS was tested on four continuous optimisation problems and compared to the Goldberg's simple GAs [22]. The results indicated that GAVaPS outperforms simple GAs in terms of fitness, however at a significantly higher computational cost.

In 2000, Back et al. [5] investigated the feasibility of dynamically controlling three GAs' hyperparameters: population size, crossover rate and mutation rate. While most of the existing contributions focused on adapting one parameter at a time, the authors proposed a joint hyperparameter control. Both crossover and mutation rates were encoded as extra genes in the linear chromosome representing the candidate solution (aka self-adaptive hyperparameter control); for the population size, however, the authors updated GAVaPS by relaxing the ageing of the fittest individual in the population. The results indicated that adapting the population size alone is very effective and is almost as effective as controlling the tree hyperparameters simultaneously. The authors concluded that it is possible to use GAs without the need of a user-specified population size, crossover and mutation rates, provided that hyperparameter control is used jointly. However, they also pointed out that the main source of this flexibility comes from the adaptation of the population size.

In 2004, Eiben et al. performed a comparative assessment of several existing adaptive methods for the population size, and proposed a new one [6]. As baselines, the authors selected the methods previously introduced in [5], while the proposed method was called the Population Resizing on Fitness Improvement GA (PRoFIGA). PRoFIGA dynamically increases the population size proportionally with the improvement of the best fitness in the population or when there is stagnation for a given number of generations, and decreases the population size otherwise. By increasing the population size, the authors aimed to skew the search towards exploration, which was considered particularly useful in the early stages of the search, where fitness typically improves, or when there is strong evidence for stagnation. Alternatively, when there is a short-term lack of fitness improvement, PRoFIGA aimed at concentrating the algorithm's resources in exploitation by reducing the population size and, consequently, the genetic diversity. To increase the population, the authors cloned solutions that were chosen by means of tournament selection, whereas an anti-tournament was used to remove solutions. PRoFIGA did not decrease the population size after a certain minimal population size was reached. The results indicated that GAVaPS extended as in [5] outperformed all the other studied algorithms, followed by PRoFIGa and standard GAs. The method proposed in [23] was found to be competitive only on relatively simple problems.

An extension of PRoFIGA was proposed in [24], by taking an opposite approach for the population size adaptation. Specifically, the authors suggested reducing the population size when the best fitness increases, which would allow for reduction in the computational costs of the search while still obtaining fair levels of performance. The authors also advocated that the initial landscape of an optimisation problem is typically rugged, and one might need a large population size to better explore it during the early stages of the search. After some period of time, the candidate solutions tend to concentrate on a smaller region of the space, which may correspond to a smoother fitness landscape. According to the authors, this relative reduction in the ruggedness was a valid justification for decreasing the population size, and therefore the computational effort, without a significant deterioration of the algorithm's effectiveness. The proposed approach was compared against a fixed population size GA and two strategies based on the number of generations: linear and exponential decrease. The results indicated the superiority of the proposed approach in terms of both effectiveness and efficiency. Surprisingly, the authors did not compare the performance of their method against PRoFIGA [6], which they initially criticized and compared their method with simple predetermined systems that can only decrease the population size, without ever increasing it. A smaller population size may incapacitate the EA to discriminate between candidate solutions at the basin of attraction of the best peak and solutions at the basin of attraction of other peaks [25].

More recently, Tanabe et al. [26] introduced L-SHADE. This algorithm is a variation of the well-known differential evolution (DE) algorithm SHADE, in which the population size is decreased linearly with generations. The results indicate that L-SHADE significantly outperforms SHADE, but it also has some drawbacks. Starting the linear decrease at the first generation does not give the required time for the individuals to evolve effectively, and it wastes computational power by evaluating solutions that will then be removed right away. The authors also pointed out that linear reduction may increase the possibility of premature convergence.

In [27], Awad et al. proposed a solution to the drawbacks of L-SHADE introducing of niching-based reduction. Instead of removing the worst individuals according to fitness, the individuals were divided into two niches, both of the same size (half of the current population size each). The first niche was composed of the best individuals of the population and its closest neighbours in the population. The second niche was made of the remaining individuals. Then a set of closest neighbours of the best individual of the second niche were removed. A similar approach was presented in [28], where a slightly different equation was used to decide the number of individuals to be removed at each generation.

The advantage of continuously decreasing is that it allows having a bigger population in the early generations so that exploring the search space is encouraged, while a smaller number of individuals promotes exploitation later on. However, it is clear that only decreasing the population size usually leads the algorithm to get stuck in local optima. For this reason, contributions such as [29] have recently proposed variants of dynamic population EAs that can also increase the population size, if needed during the evolution. Those algorithms regulate the population size according to different parameters, such as fitness improvement and the population density in neighbourhood areas of each individual.

2.1. *Previous studies on dynamic populations in genetic programming*

In GP, the dimension of the individuals usually tends to increase during the evolution, often leading to a phenomenon called bloat [10,11].[1] So, the computational cost needed to evaluate the solutions increases with generations as well. For this reason, any method aimed at saving computational effort can be beneficial to GP, including the dynamic adaption of the population size. Nevertheless, dynamic populations have been relatively less studied in GP, compared to other EAs. The first attempt dates back to 2003, when Fernandez et al. introduced a system, called Plague, to reduce the computational cost of GP by dynamically reducing the population size [30,31]. In Plague, at each generation, the k worst individuals are discarded, being k a new hyperparameter. The results showed a decrease of the computational cost as the performance of the algorithm remained comparable to the one of standard GP, at least for the case studies that were considered. The major drawback of this approach is the loss of diversity through generations, caused by the continuous decrease of the population size.

A few years later, two contributions [32,33] proposed modifications to the previously discussed approach. In [32], four new population fitness stagnation assessment methods were introduced. A new gradient pivot function was also proposed, to regulate whether to add or remove individuals from the population. In [33], together with the introduction of an exponential pivot function and a new stagnation assessment phase, a new population variation equation was introduced. This equation induces an exponential variation of the population size in order to accelerate convergence. These two approaches both showed promising results, as they were able to reduce the computational effort while keeping the performance at a similar level as standard GP.

Dynamic populations in GP have also been proposed as a viable strategy to address issues associated with dynamic environments. This type of problem is usually associated with a target function that undergoes changes over time. In [34], a GP system was proposed, able to adjust the population size in response to changes in fitness. Specifically, the population size was reduced as fitness improves

[1] The most accepted definition of bloat is the uncontrolled growth in the dimension of the evolving individuals, without a corresponding improvement in their fitness [10,11].

and incremented as fitness plateaus. In this way, the algorithm was able to adapt to changes of the target function, while at the same time, preserving computational resources and preventing overfitting in time intervals where the target function remains constant. The decision to add or remove individuals from the population was based on an equation that takes into account the current fitness values of the individuals in the population and the current population size.

In this work, we have selected a number of existing population resizing mechanisms. The methods were selected based on their recency and the evidence regarding their relative effectiveness in previous studies. In particular, we included the approaches proposed in [34,6,24]. As a comparison baseline, we selected simple linear decrease, following [30,31].

3. Geometric semantic genetic programming

Let $\mathbf{X} = \{\mathbf{x}_1, \mathbf{x}_2, ..., \mathbf{x}_n\}$ be the set of input data (training instances, observations or fitness cases) of a symbolic regression problem, and $\mathbf{t} = [t_1, t_2, ..., t_n]$ the vector of the respective expected output or target values (in other words, for each $i = 1, 2, ..., n$, t_i is the expected output corresponding to input $\mathbf{x}_i$). A GP individual (or program) P can be seen as a function that, for each input vector $\mathbf{x}_i$ returns the scalar value $P(\mathbf{x}_i)$. Following [12], termed *semantics* of P the vector $s_P = [P(\mathbf{x}_1), P(\mathbf{x}_2), ..., P(\mathbf{x}_n)]$. This vector can be represented as a point in an n-dimensional space, that we call *semantic space*. Note that the target vector $\mathbf{t}$ itself is a point in the semantic space.

As explained above, GSGP is a variant of GP where the standard crossover and mutation are replaced by GSOs. The objective of GSOs is to define modifications to the syntax of GP individuals that have a precise and measurable effect on their semantics. In particular, geometric semantic crossover (GSC) generates one offspring whose semantics stands in the line joining the semantics of the two parents in the semantic space. Instead, geometric semantic mutation (GSM), by mutating an individual i, allows obtaining another individual j such that the semantics of j stands inside a ball of a given predetermined radius centred in the semantics of i. One of the reasons why GSOs became popular is because GSOs induce a unimodal error surface (on training data) for any supervised learning problem where fitness is calculated using an error measure between outputs and targets. In other words, when using GSOs the error surface on training data is guaranteed not to have any locally optimal solution. It was shown that this property holds for any regression or, more recently, classification problem [35], independently of how big and how complex data are. Reference [13] contains a detailed explanation of the reason why the error surface is unimodal and why this is important. The definitions of the GSOs are, as given in [12], respectively:

Geometric semantic crossover (GSC) Given two parent functions $T_1, T_2 : \mathbb{R}^n \rightarrow \mathbb{R}$, GSC returns the function $GSC(T) = (T_1 \cdot T_R) + ((1 - T_R) \cdot T_2)$, where T_R is a random function whose output values range in the interval $[0, 1]$.

Geometric semantic mutation (GSM) Given a parent function $T : \mathbb{R}^n \rightarrow \mathbb{R}$, GSM with mutation step ms returns the function $GSM(T) = T + ms \cdot (T_{R1} - T_{R2})$, where T_{R1} and T_{R2} are random functions.

The reason why GSM uses a difference between two random trees, T_{R1} and T_{R2}, is that the amount of modification it causes must be centred in zero. In other words, a random expression is needed that has the same probability of being positive or negative. Even though this is not in the original definition of GSM, later contributions [36,13,37] have clearly shown that limiting the codomain of T_{R1} and T_{R2} in a predefined interval (for instance $[0, 1]$, as it is done for T_R in GSC) helps improve the generalisability of GSGP. For this reason, as in several previous works [13,14], also in this paper we constrain the outputs of T_R, T_{R1}, and T_{R2} by wrapping them in a logistic function. Only the definitions of the GSOs for symbolic regression problems are given here since they are the only ones used in this work. For the definition of GSOs for other domains, the reader is referred to [12].

As reported in [12,13], GSOs have the drawback of generating larger offspring than the parents, and this entails a rapid growth of the size of the individuals in the population. To counteract this problem, in [14,38–40] implementations of GSOs were proposed, that make GSGP significantly faster than standard GP. This is possible through a smart representation of GP individuals that allows us to not store their genotypes during the evolution. This implementation is particular to GSGP because it is based on the fact that GSC and GSM compose the ancestors and the needed random programs by means of a precise and previously fixed expression [13]. The implementation presented in [40] is the one used here.

3.1. Semantic neighbourhood

The semantic neighbourhood of an individual P (denoted by $\mathcal{SN}(P)$) indicates the set of individuals that can be generated by applying GSM to P. In other words:

$$\mathcal{SN}(P) = \{T \mid T = GSM(P)\}$$

Given a semantic neighbourhood $S = \mathcal{SN}(P)$, P is called the reference individual of S and the individuals belonging to S are called the semantic neighbours of P. In [50], semantic neighbourhood was used to introduce the concept of Training Improvement Effectiveness (TIE). TIE was used to measure the effectiveness of GSM at a given stage of the search process. It consists of the percentage of semantic neighbours that have a better fitness on the training set than their reference individual, calculated either on a sample of individuals or on the best individual in the population. TIE was used as an early stopping criterion for the Semantic Stochastic Hill Climber (SSHC) and the Semantic Learning Machine (SML) in [50]. In other words, the training was terminated when

TIE was smaller than a previously fixed threshold. The experimental findings based on three real-world regression problems showed that the TIE criterion yields the most robust generalisations [50].

In this work, we use TIE as a guideline to decide whether we should add or remove individuals from the population, instead of terminating the execution as it would happen in [50]. Moreover, in order to shed light on the ambiguity generated in [24], we propose to assess our approach following two population management policies. In the first, we adapt the strategy proposed in [6]: when TIE yields high values, this might indicate that the algorithm is still in the phase of exploration; therefore adding more individuals to the population is expected to support exploratory search. A low value of TIE might suggest convergence, and removing individuals from the population might emphasize exploitation by reducing genetic diversity. In the second population management strategy, we adopt the opposite rationale (as proposed in [24]): remove the individuals from the population when TIE yields high values to save the computational costs; add the individuals when TIE yields low values to escape local optima. Note that neither [2] nor [24] have used TIE in their methods. It is important to highlight, however, that the experimental results (see Section 6) only report the former policy because both yield comparable performance and we did not want to overwhelm the reader with redundant results. Following [50], in the continuation we calculate the TIE using the best individual in the population.

4. Dynamic population methods

Six different methods to manage dynamic populations in GSGP are studied in this paper. Those methods are presented in the continuation of this section.

4.1. LD-GSGP

The first method we present uses a simple linear decrease (LD) of the population size that is steadily applied during the evolution. Called LD-GSGP, it follows the idea of Plague, proposed by Fernandez et al. [30,31]. Starting with the initial population size, which is a parameter to be chosen before the beginning of the execution, the method functions by removing from the population a constant number of individuals at each generation. The number N of individuals to be removed from the population is chosen with the objective of terminating the evolution with a given target final population size. The final population size is another parameter of the algorithm. Following [34], the individuals that are removed from the population at each generation are the N individuals with the worst fitness on the training set. Following the taxonomy proposed in [2], LD-GSGP is the only deterministic method of dynamic variation of the population size that we study in this work. In fact, in LD-GSGP the population size control relies on a time-dependent linear function. Instead, all the remaining approaches, presented in the continuation of this section, are adaptive methods. In fact, they adapt the population size based on some information obtained during the evolutionary process itself.

4.2. DP-GSGP

The second method, referred to as Dynamic Population GSGP (DP-GSGP), is an extension to GSGP of the system presented in [34]. DP-GSGP utilizes an adaptive approach, by incorporating the ability to add or remove individuals from the population based on specific events that occur during the evolution. Specifically, it is assumed that the optimisation is effective if the best fitness value in the current generation is found to be superior to the one in the previous generation; in this situation, the individuals are removed from the population. The rationale is that this may allow us to save computational effort without compromising the effectiveness of the search. Conversely, the evolutionary process is assumed to stagnate if the best fitness in the population remains unchanged; in this situation, adding new individuals to the population can increase the population's and improve the exploration at the cost of extra computational effort. The use of elitism prevents the best fitness in the population from worsening one generation to the next. Exactly as in [34], the number of individuals to be added/removed is regulated by the following equation:

$$\Delta_{pop} = pivot \times strength \times best\_fit\_contribution \times pop\_size\_contribution \tag{1}$$

where:

- Δ_{pop} is the difference in the population size from one generation to the next.
- $pivot$ is either -1, if the best fitness improved in the last generation, or 1, otherwise. This parameter regulates whether to add or remove individuals from the population.
- $strength$ is a parameter used to scale the value of $best\_fit\_contribution$ and $pop\_size\_contribution$. It is used to add or remove from the population a number of individuals that is a percentage of $best\_fit\_contribution \times pop\_size\_contribution$ at each generation.
- $best\_fit\_contribution$ is a parameter controlled by Algorithm 1. It always returns a value in the range $[min\_coeff_{fit}, max\_coeff_{fit}]$, where min_coeff_{fit} and max_coeff_{fit} are parameters. It returns max_coeff_{fit} when the current fitness is "bad", i.e. large (given that we want to minimize the error). In particular, $best\_fit\_contribution$ is equal to max_coeff_{fit} when the best fitness in the population is larger than a previously fixed value, called max_fit. Analogously, $best\_fit\_contribution$ is equal to min_coeff_{fit} when the best fitness in the population approximates the optimal fitness in a satisfactory way. In particular, when it is smaller than another previously fixed value called tgr_fit. When the best fitness in the population is included in $[tgr\_fit, max\_fit]$, $best\_fit\_contribution$ is equal to the output of the linear function shown in Fig. 1, which is directly proportional to the best fitness in the population.
- $pop\_size\_contribution$ is the output of a linear function built in a similar way to $best\_fit\_contribution$. The pseudo-code of the method that regulates this function is shown in Algorithm 2. Analogously to the $best\_fit\_contribution$, this linear function returns

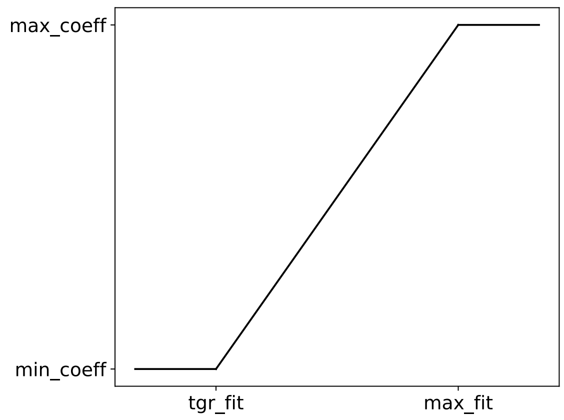


Fig. 1. Graphical representation of the function used to adapt $best\_fit\_contribution$ in DP-GSGP, as defined in Algorithm 1.

Algorithm 1 Pseudo code for $best\_fit\_contribution$.

```
if ($fitness(best) \le tgr\_fit$) then
    return $min\_coeff_{fit}$;
else if ($fitness(best) \ge max\_fit$) then
    return $max\_coeff_{fit}$;
else
    return $max\_coeff_{fit} - min\_coeff_{fit} \times \frac{fitness(best)-tgr\_fit}{max\_fit-tgr\_fit} + min\_coeff_{fit}$
end if
```

Algorithm 2 Pseudo code for $pop\_size\_contribution$.

```
if $current\_pop\_size \le \frac{initial\_pop}{4}$ then
    return $max\_coeff_{size}$;
else if $current\_pop\_size \ge initial\_pop$ then
    return $min\_coeff_{size}$;
else
    return $max\_coeff_{size} - min\_coeff_{size} \times \frac{current\_pop\_size-initial\_pop}{\frac{initial\_pop}{4}-initial\_pop} + min\_coeff_{size}$
end if
```

the maximum possible value (i.e., max_coeff_{size}) when the current population size is minimal (i.e., $\frac{initial\_pop}{4}$) and the minimum possible value (i.e., min_coeff_{size}) when the current population size is maximal (i.e., the initial value of the population size $initial\_pop$). When the current population size is included in $[min\_coeff_{size}, max\_coeff_{size}]$, $pop\_size\_contribution$ is inversely proportional to the current population size itself.

When individuals have to be removed from the population, DP-GSGP chooses the worst individuals in terms of fitness on the training set. When new individuals need to be added to the population, random individuals are created, using the same algorithm as the one employed in the population initialisation phase.

4.3. TIE-DP-GSGP

The third method we present, called TIE-DP-GSGP, is novel and it is a variation of DP-GSGP discussed in Section 4.2. The only difference between TIE-DP-GSGP and DP-GSGP is in the way the method decides if the algorithm is stagnating or not. Instead of comparing the best fitness from the current generation with the one from the previous generation, TIE-DP-GSGP uses the TIE criterion presented in Section 3.1 to take this decision. To calculate the TIE, TIE-DP-GSGP uses a number of neighbours equal to the initial population size. At each generation, TIE is compared with a previously fixed tolerance threshold. If it is smaller than this threshold, then $|\Delta_{pop}|$ individuals are removed from the population, where Δ_{pop} is defined in Equation (1); contrarily, $|\Delta_{pop}|$ individuals are added. Different values of the tolerance threshold are tested in our experiments.

4.4. TIE-DP-GSGP-SEI

The fourth method studied in this work, called TIE-DP-GSGP-SEI (in which the acronym SEI stands for store eliminated individuals) is also novel, and it is a variation of the TIE-DP-GSGP presented in Section 4.3. The only difference is that TIE-DP-GSGP-SEI uses a repository, where all the individuals that have been removed from the population in the previous generations are stored. When the population size needs to be increased, instead of adding random individuals to the population, TIE-DP-GSGP-SEI chooses the best individuals in the repository, in terms of training fitness. The rationale behind this choice is that the genetic material of the

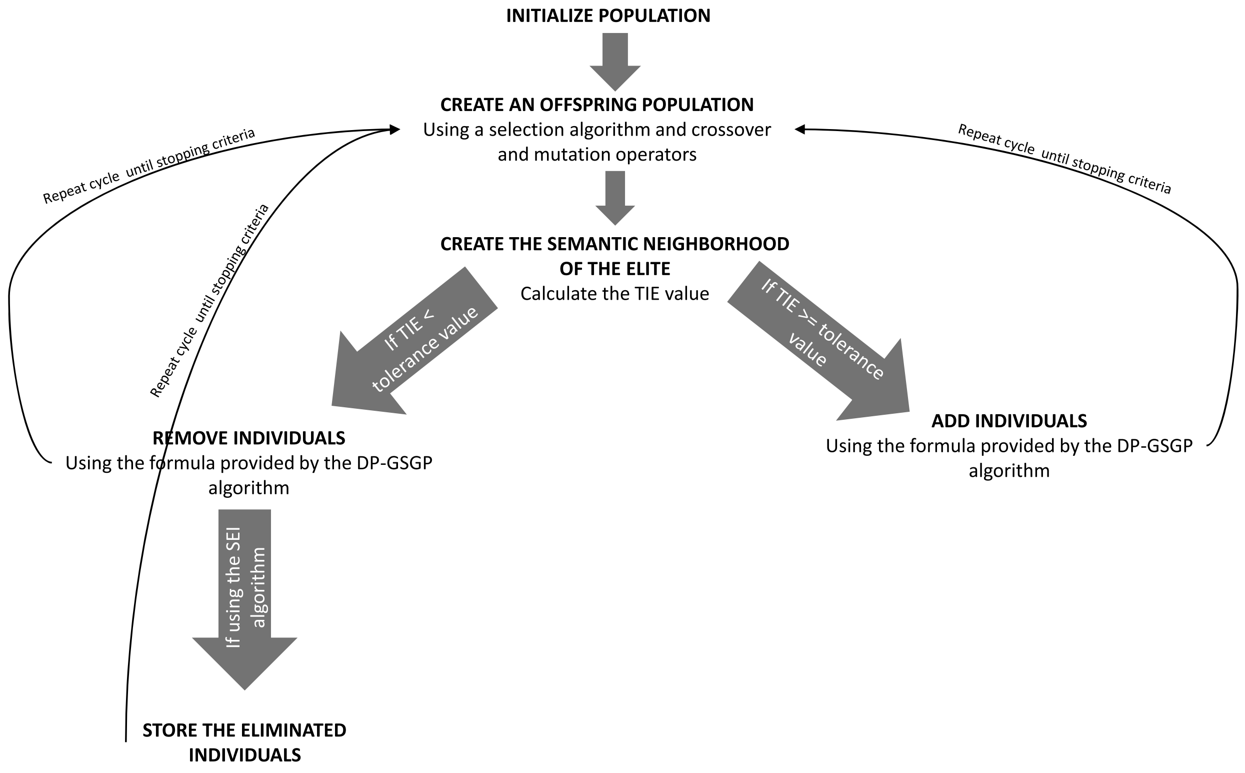

INITIALIZE POPULATION
CREATE AN OFFSPRING POPULATION
Using a selection algorithm and crossover and mutation operators
CREATE THE SEMANTIC NEIGHBORHOOD OF THE ELITE
Calculate the TIE value
If TIE < tolerance value
If TIE >= tolerance value
REMOVE INDIVIDUALS
Using the formula provided by the DP-GSGP algorithm
ADD INDIVIDUALS
Using the formula provided by the DP-GSGP algorithm
If using the SEI algorithm
STORE THE ELIMINATED INDIVIDUALS
Repeat cycle until stopping criteria
Repeat cycle until stopping criteria
Repeat cycle until stopping criteria

Fig. 2. Flowchart of TIE-DP-GSGP and TIE-DP-GSGP-SEI.

Algorithm 3 Pseudo code for PRoFIGSGP.

```
if $best\_fitness(g_t) \leq best\_fitness(g_{t-1})$ then
    $grow\_population$
else if fitness did not improve for $G$ generations or more then
    $grow\_population$
else
    $shrink\_population$
end if
```

individuals in the repository has already evolved, and so it is supposed to have better fitness than random individuals. We do realize, however, that such an approach can eventually contribute to diversity loss, in comparison to the TIE-DP-GSGP method. In order to clarify whether this will happen and to what extent, we decided to include this variant in our experiments. A flowchart of the main working process of TIE-DP-GSGP and TIE-DP-GSGP-SEI is presented in Fig. 2.

4.5. PRoFIGSGP

PRoFIGSGP is an adaptation to GSGP of the PRoFIGA method, presented in [6] and discussed in Section 2. PRoFIGSGP (Population Regulation on-the-Fly in GSGP) uses a dynamic population size modified based on the current performance of the evolutionary process. The population size is increased when the algorithm is making slow progress and decreased when the algorithm is making rapid progress. This allows the algorithm to focus computational resources on areas of the search space where progress is being made while avoiding wasting resources on areas where progress is not being made. PRoFIGSGP works as shown in Algorithm 3, where g_t is the current generation, and g_{t-1} is the previous generation. The population increase, $grow\_population$, is the result of Equation (2), where $increaseFactor$ is a random value, drawn with uniform distribution from (0, 1), and $maxEvalNum$ and $currEvalNum$ denote the given maximum number of fitness evaluations and the current number of evaluations, respectively. $maxFitness_{new}$, $maxFitness_{old}$ and $initMaxFitness$ are the best fitness values in the current generation, the previous generation and the initial population, respectively.

$$grow\_population = increaseFactor \times (maxEvalNum - currEvalNum) \times \frac{maxFitness_{new} - maxFitness_{old}}{initMaxFitness} \tag{2}$$

The new individuals to be added to the population are clones of good individuals that are chosen by tournament selection from the actual population. The value $shrink\_population$, corresponding to the number of removed individuals, is equal to a percentage of the current population. The individuals to be removed are chosen according to their fitness: the ones with higher values are removed.

Table 1
Similarities and differences between the various presented methods.

Algorithm	Fitness-based	Use semantic neighbourhood	Time-dependent function
LD-GSGP			✓
DP-GSGP	✓		
TIE-DP-GSGP		✓	
TIE-DP-GSGP-SEI		✓	
PRoFIGSGP	✓		
SPS-GSGP	✓		

Table 2
Datasets used in this experimental study.

Dataset	#Observations	#Features	Target range	Reference
Boston	506	13	[5, 50]	[42]
Concrete	1005	8	[2.3318, 82.5992]	[43]
Bioav	358	241	[0.4, 100.0]	[44]
LD50	234	626	[0.25, 8900.0]	[44]
PPB	5875	19	[7.0, 54.992]	[44]
f4	6060	1	[-0.3, 0.3]	[41]
f11	200	2	[0.0028, 1.0]	[41]
f16	200	2	[-1.3102, 0.0]	[41]

The algorithm also includes a mechanism for preventing the population size from becoming too large, which can lead to a decrease in performance.

4.6. *SPS-GSGP*

SPS-GSGP is another variant of population size adjustment strategies developed for GAs [24]. In SPS-GSGP, the population size is reduced proportionally to the change in its best fitness. Let N_t be the population size at generation t. Denote the change in best fitness at generation t by $\Delta f_t^{best} = |f_{t-1}^{best} - f_{t-2}^{best}|/|f_{t-2}^{best}|$. Then the new population size, N_{t+1} is determined as follows:

$$N_{t+1} = \begin{cases} (1 - \Delta f_t^{best})N_t, & \text{if } \Delta f_t^{best} \leq \Delta f_{max}^{best} \\ (1 - \Delta f_{max}^{best})N_t, & \text{if } \Delta f_t^{best} > \Delta f_{max}^{best} \\ min\_popsize, & \text{if } N_{t+1} < min\_popsize \end{cases} \tag{3}$$

where Δf_{max}^{best} is a parameter that corresponds to the fitness tolerance level and $min\_popsize$ is another parameter that corresponds to the minimum population size allowed.

Table 1 aims to distinguish between the different methods used in our work. The first column indicates whether the population adjustment is purely based on the fitness values of the individuals. The second column indicates whether the population adjustment method uses the semantic neighbourhood. The third column indicates whether the method uses a time-dependent function or whether the decision-making is supported by evolutionary dynamics. It is worth highlighting that the only difference between TIE-DP-GSGP and TIE-DP-GSGP-SEI is the fact that the latter stores eliminated individuals.

5. Experimental setup

5.1. *Test problems*

In our experimental study, eight datasets are used, five of which are real-world datasets, while the remaining three are artificial datasets introduced in [41]. The main characteristics of these datasets are presented in Table 2.

Contrarily to the other chosen test problems, that are well-established and widely used for benchmarking in GP, the Bioavailability dataset deserves a specific discussion. In fact, that dataset was criticised in [45], partially because of a lack of preprocessing because it includes features that contain no information as well as contradictory relationships between the dependent and independent variables. However, according to many authors who have used this dataset, these characteristics are interesting and should be integrated into a reasonable benchmark suite, because they allow us to test the ability of our algorithms to deal with the difficulties and ambiguities typical of real-world data. It is not our objective to discuss what characteristics a good benchmark suite should possess (refer to [46,47]). We simply observe that the Bioavailability dataset, as well as the PPB and LD50 datasets, has been used in several previous GP studies, clearly indicating a trend for overfitting to emerge [14,36,37]. We thus use these three datasets as a sort of stress test case to assess the generalisability of the studied algorithms.

Table 3
Parameters used in our experimental study.

Algorithm	Parameter	Value
Common to all the algorithms	Initial population size	200 individuals
	Minimum population size	50 individuals
	Number of Generations	100
	Crossover probability	0.2 for GSGP, 0.8 for GP
	Mutation probability	0.8 for GSGP, 0.2 for GP
	Mutation step (ms)	Random number in [0, 1)
	Selection	Tournament
	Tournament size	2 individuals
	Initialisation method	Ramped Half-and-Half
DP-GSGP & TIE-DP-GSGP & TIE-DP-GSGP-SEI	$strength$	0.3
	tgr_fit	0.01
	max_fit	60
	min_coeff_{fit}	1
	max_coeff_{fit}	10
	min_coeff_{size}	0
	max_coeff_{size}	10
PRoFIGSGP	$increaseFactor$	0.1
	$maxEvalNum$	20000
	$shrinkPopulation$	5%
	G	5
SPS-GSGP	Δf_{max}^{best}	0.08

5.2. Performance measures

In all the different methods presented in Section 4, the fitness of the evolving individuals is assessed using the Root Mean Squared Error (RMSE) on the training set, and the experimental comparison is carried out using the RMSE on training and unseen data (training set and test set, respectively). However, it is clear that at each generation, each one of the methods may be employing a population that contains a different number of individuals. So, different generations may cost very disparate computational efforts to be evaluated. For this reason, it would not be fair to compare the RMSE across generations. A much better approach is the one adopted in [30], where the RMSE is compared at corresponding values of the computational effort. In this paper, we adopt the same definition of computational effort as in [30]. The computational effort is calculated by computing the average number of nodes of the individuals in the population at generation g (indicated as avg_lenght_g) and then by computing the partial effort (indicated as PE_g) as follows:

$$PE_g = n \times avg\_lenght_g \tag{4}$$

where n is the number of individuals in the population at generation g. Then, the computational effort E_g is calculated as the cumulative sum of the partial efforts until generation g, as expressed by the following equation:

$$E_g = PE_g + PE_{g-1} + \ldots + PE_1 + PE_0 \tag{5}$$

To assess the computational effort spent by an algorithm, this measure takes into account both the population size and the dimension (expressed as a number of nodes) of the individuals in the population. It was considered in [30] as a faithful model for the actual running time of the algorithm.

In the next section, the experimental results will be presented by reporting curves of the RMSE against the computational effort. In other words, the plots presented in the next section will have a point at each generation g, and that point will have E_g on the horizontal axis and the training or test RMSE of the best individual in the population on the training set at generation g on the vertical axis. This allows us to compare the training and test RMSE of the different studied methods for the same values of the computational effort.

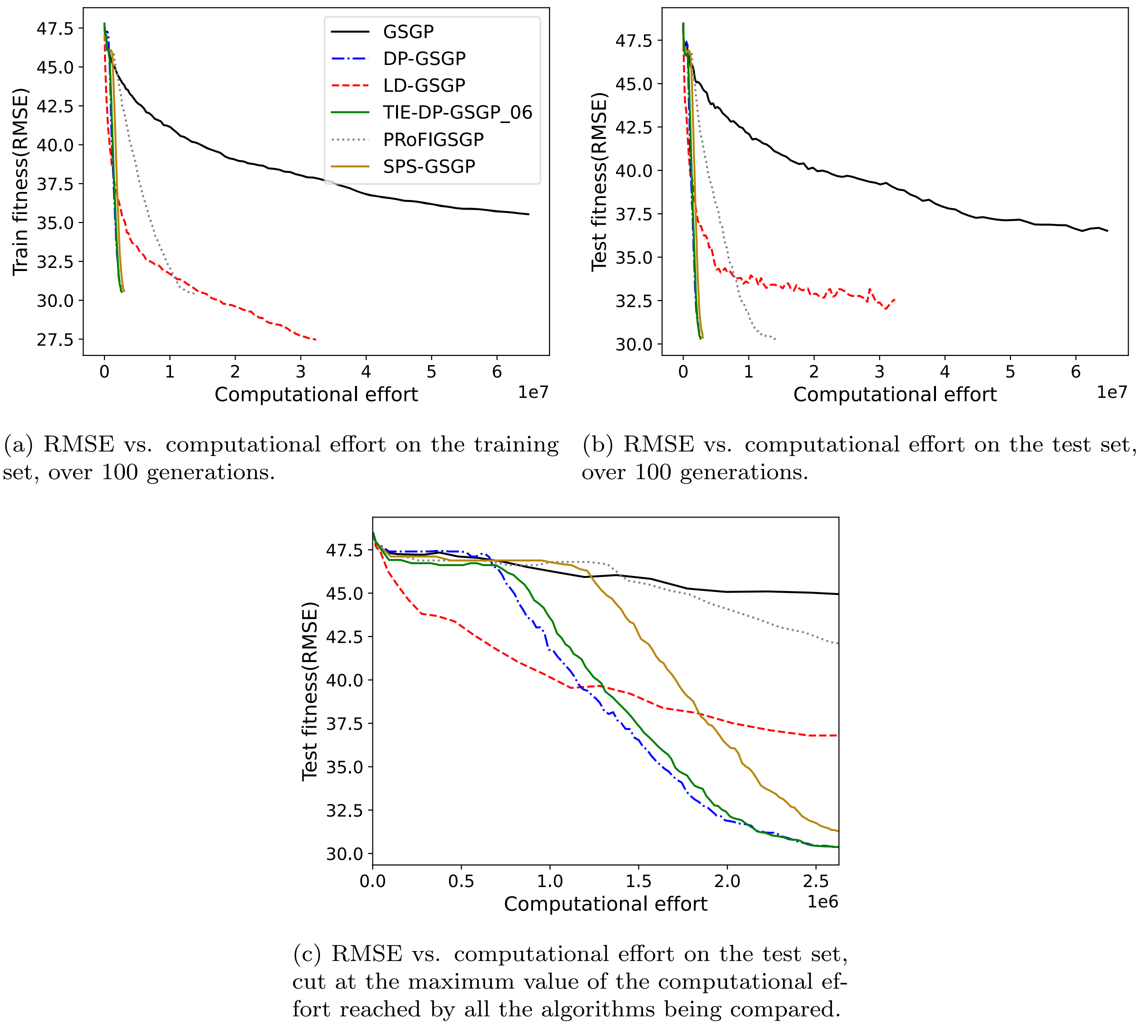
In order for the comparison to be completely fair, for the algorithms that involve the creation of a semantic neighbourhood, i.e., TIE-DP-GSGP and TIE-DP-GSGP-SEI, the individuals evaluated from the semantic neighbourhood are included in the calculation of the computational effort. So, the equation of PE_g for TIE-DP-GSGP and TIE-DP-GSGP-SEI is:

$$PE_g = n \times avg\_lenght_g + n_{neigh} \times avg\_lenght_{neigh} \tag{6}$$

where n_{neigh} is the total number of neighbours of the best solution in the population at generation g and avg_lenght_{neigh} is the average number of nodes of those neighbours.

5.3. Parameter settings

The parameters used in our experimental study are presented in Table 3. The objective of our work is to compare the traditional version of GSGP, which uses a population with constant size, with the dynamic population variants. In the last part of our experimen-



(a) RMSE vs. computational effort on the training set, over 100 generations.

(b) RMSE vs. computational effort on the test set, over 100 generations.

(c) RMSE vs. computational effort on the test set, cut at the maximum value of the computational effort reached by all the algorithms being compared.

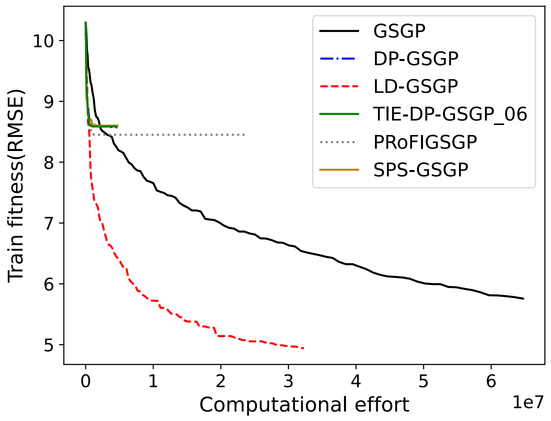
Fig. 3. Experimental results of the *Bioavailability* dataset.

tal study, we will also compare all the studied GSGP variants with the corresponding algorithms applied to standard GP. Our focus is not on obtaining the best possible results on the studied test problems. For this reason, instead of optimizing the parameters, we have preferred to use a relatively standard parameter setting, taken from the literature [14,34,24,6]. In particular, we have decided to use a configuration in which GSM is applied with a higher probability than GSC, as recommended in the literature [14,13]. By its very definition, in fact, GSM is constrained to generate "offspring" *very close* to its "parent" in the semantic space. In other words, the outputs calculated by the "offspring" program are indeed equal to the outputs of the "parent" program, except for a weak perturbation in some of them, whose importance can be tuned by means of the mutation step parameter, p_m. This guarantees the *locality* of the GSM operator, preventing it from becoming something comparable to a random search. Furthermore, given its ability to induce a unimodal error surface, GSM is typically used as the leading genetic operator for GSGP. However, besides this configuration (that uses a GSM rate of 0.8 and a GSC rate of 0.2), we have also tested a configuration in which GSC is applied more often than GSM (i.e., a GSM rate of 0.2 and a GSC rate of 0.8). The obtained results (not shown here to save space) have confirmed that the configuration in which GSM is applied with a higher probability than GSC outperforms the other. Besides these parameters, TIE-DP-GSGP and TIE-DP-GSGP-SEI have been tested with the following values of the tolerance threshold: 0.1, 0.2, 0.3, 0.4, 0.5 and 0.6.

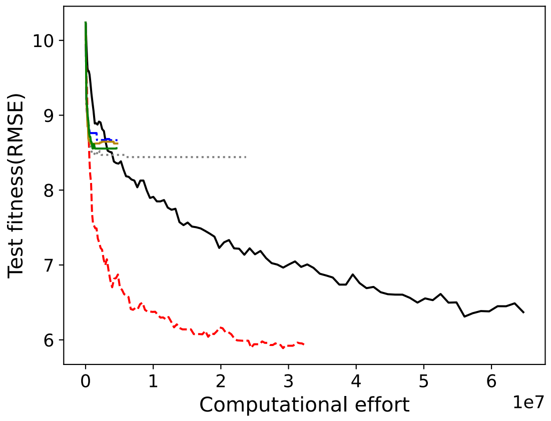
All the results reported in the next section are medians over 30 independent runs. At every run, the datasets have been split into a different random training/test partition, where 70% of the observations were randomly selected, with uniform probability, to form the training set, while the remaining 30% form the test set. The same partition has been used for all the studied methods at each particular run.

6. Experimental results

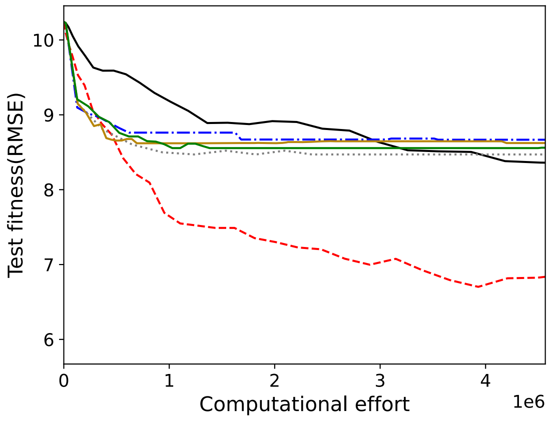
The obtained experimental results are reported in Figs. 3, 4, 5, 6, 7, 8, 9 and 10 for the Bioavailability, Boston, Concrete, PPB, LD50, f11, f16 and f4 test cases, respectively.



(a) RMSE vs. computational effort on the training set, over 100 generations.



(b) RMSE vs. computational effort on the test set, over 100 generations.

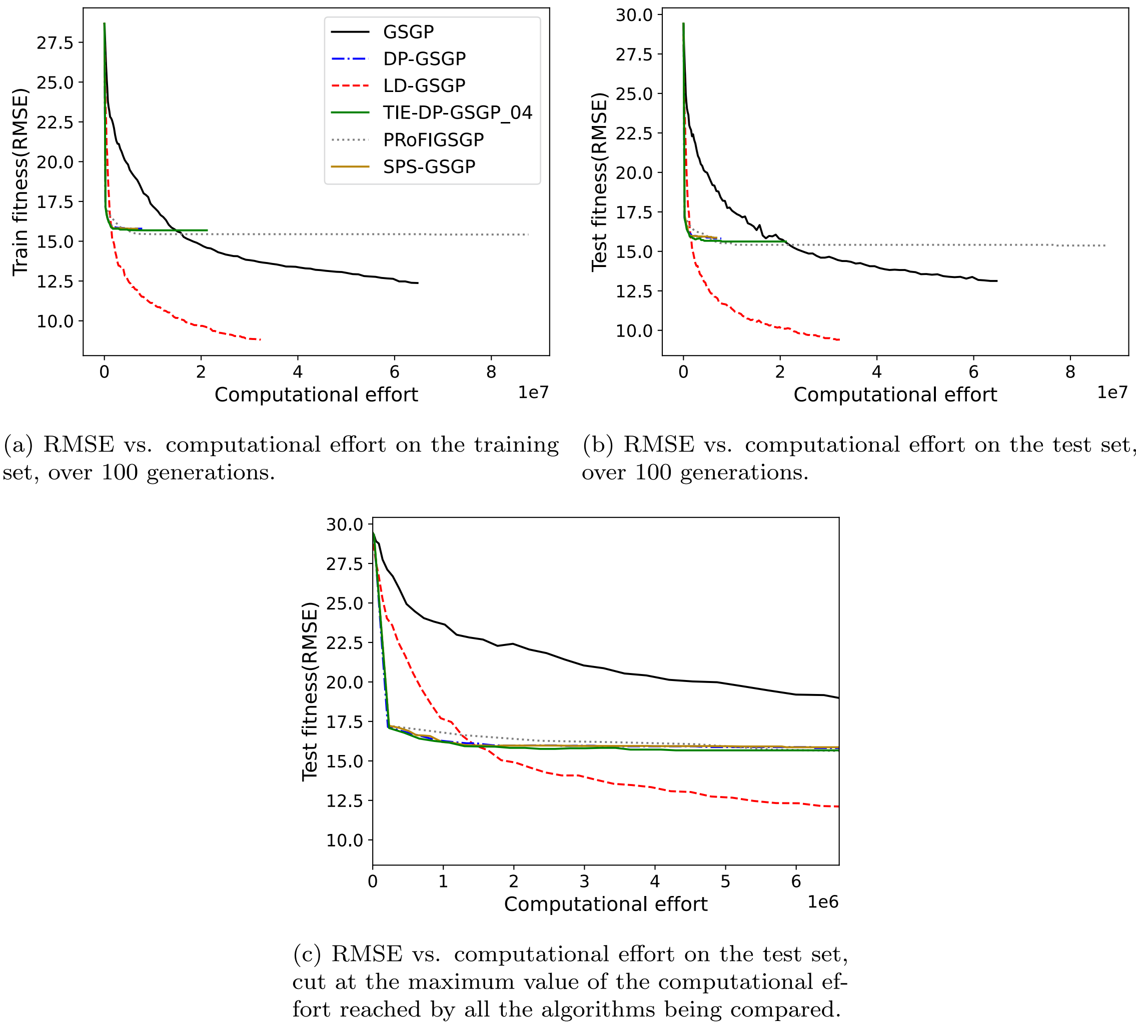


(c) RMSE vs. computational effort on the test set, cut at the maximum value of the computational effort reached by all the algorithms being compared.

Fig. 4. Experimental results of the *Boston* dataset.

Each one of these figures shows a comparison between six algorithms: GSGP, LD-GSGP, DP-GSGP, PRoGSGP, SPS-GSGP and the version of TIE-DP-GSGP or TIE-DP-GSGP-SEI using the tolerance threshold value that has returned the best result on a given problem. For example, when the legend reports TIE-DP-GSGP_06, the result corresponds to the TIE-DP-GSGP algorithm with a tolerance threshold equal to 0.6. Also, each figure contains three plots. The two plots in the upper part of the figure are the curves of the fitness against the computational effort, calculated respectively on the train and test partitions over all the generations. The third plot, in the lower part of the figure, is the curve of the test fitness against the computational effort, cut at the minimum value (min_{eff}) of the computational effort reached by any of the considered algorithms at the last generation of its execution. For example, if TIE-DP-GSGP is the first to complete the 100 generations of the evolutionary process and uses 1,000,000 units of computational effort, then the third plot will report test fitness (RMSE) for all six algorithms at that computational effort to allow a fair comparison. Table 4 reports p-values of the Wilcoxon test for pairwise comparison of the methods. These p-values are calculated at computational effort min_{eff}. The p-values are in bold when they are statistically significant, using a significance level $\alpha = 0.05$, with the Bonferroni correction [49].

The results clearly show the advantage of using adaptive methods for controlling the population size in GSGP: from the figures, it is clear that at least one of the five adaptive methods significantly outperforms standard GSGP on unseen data for all the studied test problems, except for the unique, isolated, case of the f4 synthetic benchmark, for which all the studied methods have comparable performance. Even if observing algorithms' performance at their maximum computational effort (this corresponds to the sub-figure (b)), there is always at least one adaptive method that achieves better generalisability than GSGP. When compared to standard GSGP, the studied population control methods allow saving a considerable amount of computational effort and yield solutions of better quality for the same computational budget. Following the *p-values* from Table 4, the proposed TIE-DP-GSGP and TIE-DP-GSGP-SEI methods significantly outperform standard GSGP on unseen data on the majority of the studied problems (five test cases in eight),



(a) RMSE vs. computational effort on the training set, over 100 generations.

(b) RMSE vs. computational effort on the test set, over 100 generations.

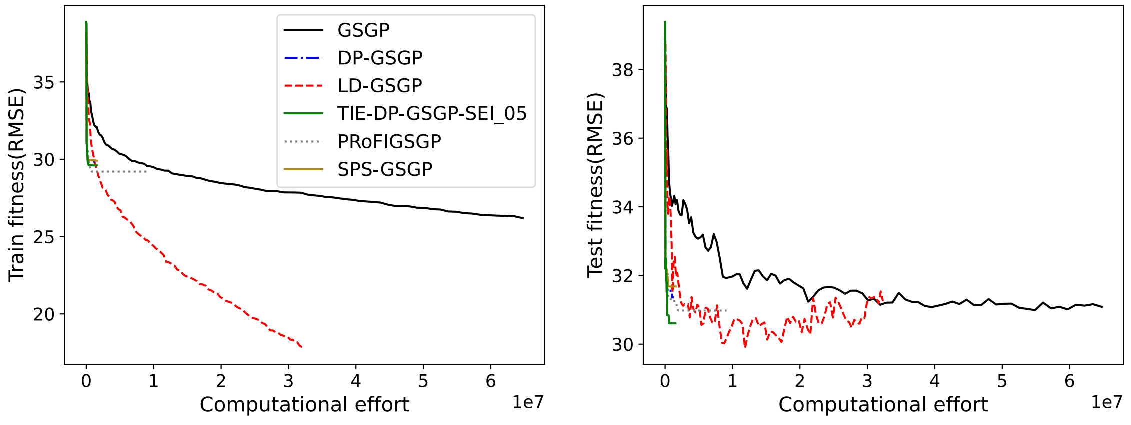
(c) RMSE vs. computational effort on the test set, cut at the maximum value of the computational effort reached by all the algorithms being compared.

Fig. 5. Experimental results of the *Concrete* dataset.

while having comparable performance on the three remaining cases. Interestingly, for the datasets known from the literature to be prone to overfitting [14,36,37], i.e., Bioavailability, PPB and LD50, the proposed TIE-DP-GSGP and TIE-DP-GSGP-SEI methods are always able to counteract overfitting, significantly outperforming the standard version of GSGP on unseen data. When compared with other adaptive methods, the proposed methods consistently achieve comparable or even better results. Although there is no parameter setting for the proposed TIE-DP-GSGP and TIE-DP-GSGP-SEI methods that outperforms the remaining adaptive approaches on all the problems, the comparison of the training and test curves between the first two sub-figures i each figure suggests that the proposed method is particularly adverse to overfitting. Our interpretation is that semantic awareness, implemented by the TIE criterion, is very important to limit overfitting for those datasets (consistently with what was already observed in the literature [14,36]). Besides corroborating the hypothesis that TIE-DP-GSGP and TIE-DP-GSGP-SEI do not overfit, this last observation suggests that an appropriate parameter setting of TIE-DP-GSGP and TIE-DP-GSGP-SEI on the training, data should also correspond to a reasonable configuration of these methods on unseen data.

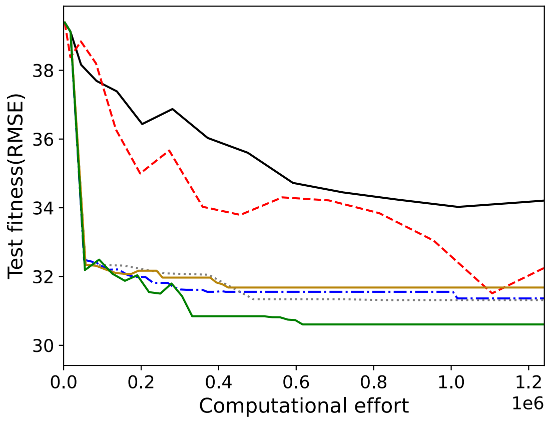
We now deepen the analysis of the results by discussing them problem by problem. The analysis of Fig. 3 shows that the proposed TIE-DP-GSGP_06 and two other adaptive methods (DP-GSGP and SPS-GSGP) achieve comparable generalisability, both in terms of min_{eff} and when assessed at the maximum computational effort (b). The PRoFIGSGP method arrives at similar test results (b) but with a larger computational effort. At min_{eff}, the proposed approach is statistically better than LD-GSGP, PRoFIGSGP and standard GSGP. It is also relevant to highlight the consistency of these four adaptive methods when comparing the training (3a) and test curves (3b). All the presented methods save a considerable amount of computational effort and find solutions that are statistically better than standard GSGP.

For both Figs. 4 and 5, LD-GSGP allows for achieving the best levels of generalisability when considering both maximum (b) and minimum computational (c) effort; the difference was found to be statistically significant (Table 4). The remaining adaptive methods exhibit comparable test fitness, although ProFIGSGSP consumes more computational effort to complete 100 generations. Regarding



(a) RMSE vs. computational effort on the training set, over 100 generations.

(b) RMSE vs. computational effort on the test set, over 100 generations.



(c) RMSE vs. computational effort on the test set, cut at the maximum value of the computational effort reached by all the algorithms being compared.

Fig. 6. Experimental results of the *PPB* dataset.

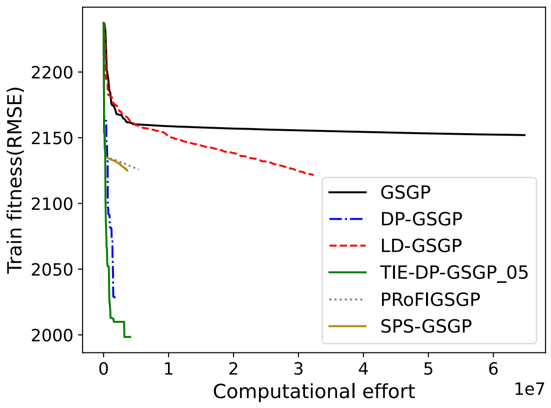
the standard GSGP, however, all the adaptive methods happen to be statistically better at min_{eff} than their static counterpart on the *Concrete* problem; on the *Boston* problem, statistical difference was observed only with LD-GSGP.

For the last pharmacokinetic dataset, PPB (Fig. 6), all the novel algorithms significantly outperform the standard GSGP while having comparable performances between each other. Moreover, the comparison between the training (3a) and test curves (3b) favours the adaptive methods in terms of consistency (except LD-GSGP).

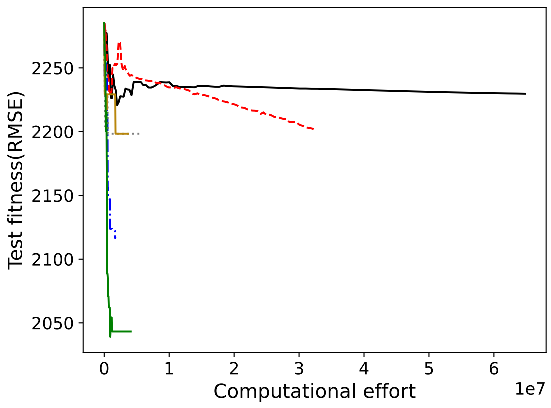
The analysis of Fig. 7 shows that the proposed TIE-DP-GSGP_05 method achieves statistically superior generalisability at min_{eff} (7c), except for DP-GSGP; nevertheless, visually, the proposed TIE-DP-GSGP_05 is clearly more advantageous. When considering the six algorithms at their maximum computational effort (7b), it becomes clear that the adaptive methods achieve notably lower fitness values using substantially fewer resources when compared to their static counterpart. Once again, one can note the consistency of the adaptive methods (except LD-GSGP) when comparing the training (3a) and test curves (3b).

Finally, we discuss the results obtained on the studied hand-tailored benchmarks. On the f11 function, TIE-DP-GSGP-SEI_01, DP-GSGP, PRoFIGSGP and SPS-GSGP perform comparably to each other and significantly outperform the other two methods on the test set. While there is no statistically significant difference between the test RMSE of LD-GSGP and standard GSGP on this problem, standard GSGP clearly appears to be more prone to overfitting, shown by the wide oscillations of the GSGP curve on the test set. On the f16 function, TIE-DP-GSGP-SEI_02, DP-GSGP, PRoFIGSGP and SPS-GSGP perform comparably to each other and to standard GSGP as well. All these algorithms significantly outperform LD-GSGP, which is clearly affected by overfitting, as can be observed by the wide oscillations of its curve. Finally, on the f4 function, all the studied methods perform in a comparable way to each other, since no statistically significant difference can be observed in their respective performance.

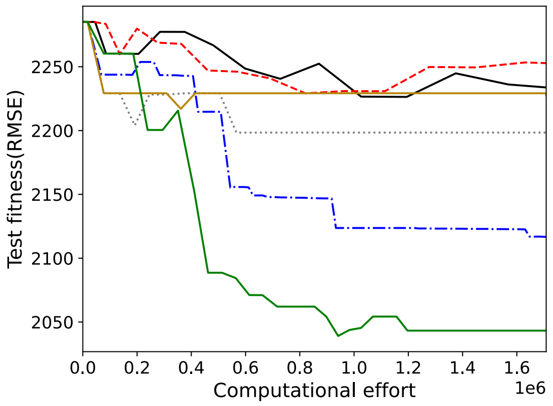
Several potential factors lead to the differences in experimental results between different adaptive methods across the test problems. Of all the implemented adaptive methods, LD-GSGP reported the least consistent results: while on *Boston* and *Concrete* it



(a) RMSE vs. computational effort on the training set, over 100 generations.



(b) RMSE vs. computational effort on the test set, over 100 generations.

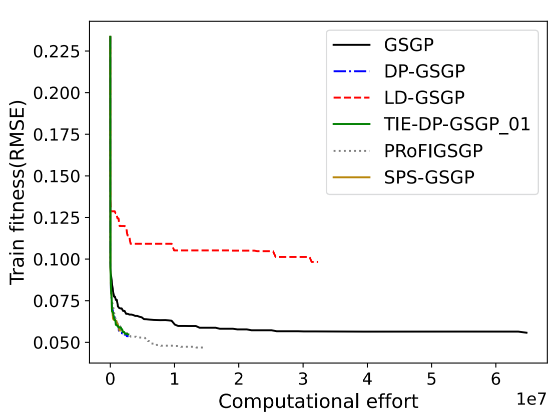


(c) RMSE vs. computational effort on the test set, cut at the maximum value of the computational effort reached by all the algorithms being compared.

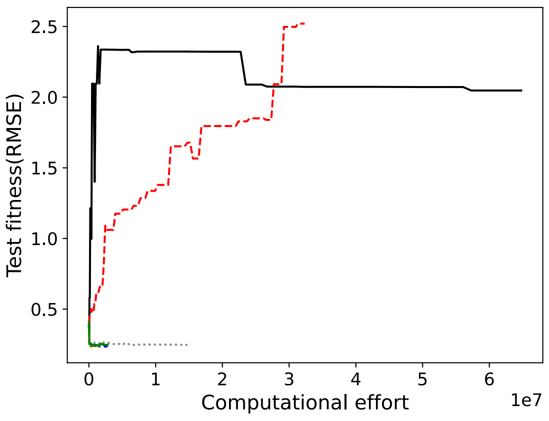
Fig. 7. Experimental results of the *LD50* dataset.

reported the best results at min_{eff}, clearly overfit on *PPB*, *LD50*, *f11* and *f16* problems. We consider that this discrepancy relates to the underlying "adaptive" mechanics of this algorithm: a simple time-dependent linear decrease that does not take the evolutionary dynamics into account. Thus, for some problems, it might happen to work well, whereas for others it might lead to overfitting. Frequently, the SPS-GSGP and DP-GSGP methods seem to exhibit similar performance. We consider that the main reason for this resides in the conceptual similarities between the two approaches: the population size is reduced based on the fitness improvement from one generation to another. The main difference consists of how exactly the number of individuals to be removed is regulated (see Sections 4.2 and 4.6 for more details). In this sense, we speculate that the different approaches to estimating the number of removed individuals lead to comparable performances when evaluated on different problems. The generalisability of the PRoFIGSGP method was found to be substantially different from the aforementioned methods. In our opinion, this relates to the guiding rationale behind the algorithm's population adaptation, which seems to contrast with that of DP-GSGP and SPS-GSGP: the population grows if there is an improvement of the elite from one generation to another, or then there is no improvement after G generations (long period); the population decreases otherwise (no improvement in the short-term). While the three methods that exploit evolutionary dynamics for adjusting the population size (DP-GSGP, SPS-GSGP, PRoFIGSGP) rely upon rules essentially based on fitness improvement from one generation to another, the proposed TIE-DP-GSGP method performs population adjustment using a conceptually different approach – by exploring the semantic neighbourhood of the current elite and estimating the proportion of better neighbours. In our opinion, such a degree of semantic awareness justifies its empirical superiority assessed on eight symbolic regression problems, in particular the robustness to overfitting.

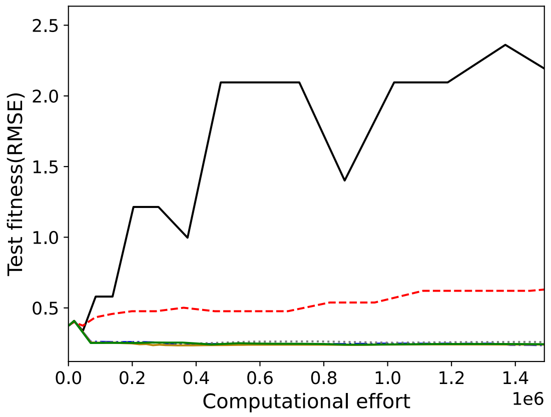
All these results, and in particular the ones obtained on the five real-life applications, show that the novel introduced methods, i.e. TIE-DP-GSGP and TIE-DP-GSGP-SEI, were very successful. This outcome represents a strong argument in favour of the use of semantic awareness as a criterion to decide whether to increment or decrement the population size of GSGP during the evolution.



(a) RMSE vs. computational effort on the training set, over 100 generations.



(b) RMSE vs. computational effort on the test set, over 100 generations.



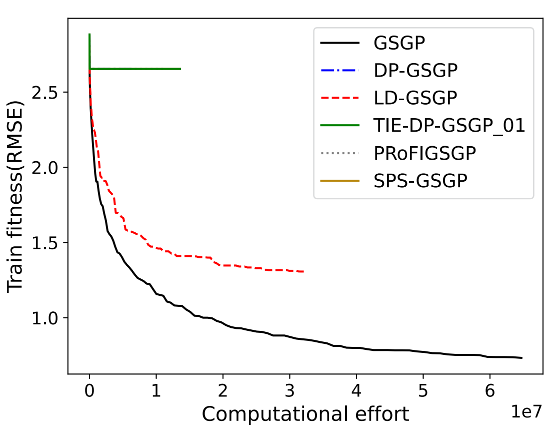
(c) RMSE vs. computational effort on the test set, cut at the maximum value of the computational effort reached by all the algorithms being compared.

Fig. 8. Experimental results of the *f11* dataset.

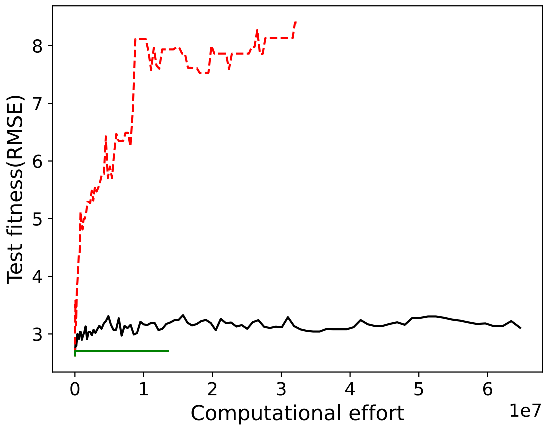
Furthermore, the presented results indicate that TIE is an appropriate measure to capture the information needed to adapt the population size appropriately.

6.1. A comparison with standard GP

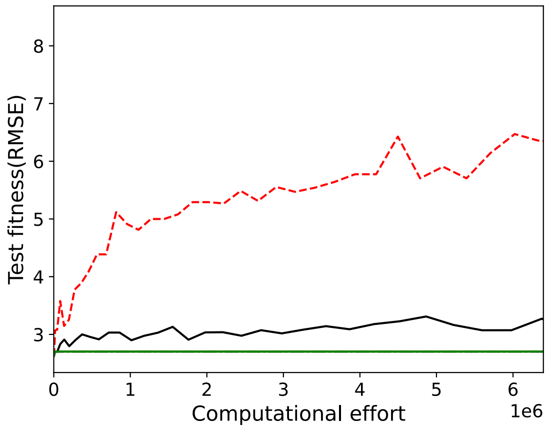
All the methods presented and studied so far can be applied in a very natural way to standard GP. This extension, however, is impractical for the approaches used in TIE-DP-GSGP and TIE-DP-GSGP-SEI, because TIE-based methods are deeply rooted in the concept of semantic neighbourhood, introduced in Section 3.1, and are constructed using GSM applied to a reference individual. Given that GSM is a defining operator of GSGP, it would be unsuitable to extend the TIE-DP-GSGP and TIE-DP-GSGP-SEI methods to standard GP. Although the focus of our work is on GSGP, still it is interesting to investigate how differently GSGP and standard GP benefit from all these different ways of managing dynamic populations. Table 5 reports the RMSE on the test set at the maximum level of the computational effort reached by all the algorithms for the different studied variants applied to standard GP, together with the analogous results for the GSGP variants. The p-values are also reported in the table. The first observation is that the best GSGP variant consistently outperforms the best standard GP variant on all the studied test problems, except for the PPB, Boston and f4 datasets, in which no statistically significant difference is visible. Another interesting outcome apparent in Table 5 is that the difference between the static population and the dynamic population variants of GSGP is generally more marked than in the case of standard GP. Actually, for standard GP the variant that uses a static population size even outperforms the dynamic population variants in most of the test cases. In other words, using dynamic populations looks more convenient for GSGP than for standard GP. This outcome is a further corroboration of the suitability of this research and encourages us to pursue the study.



(a) RMSE vs. computational effort on the training set, over 100 generations.



(b) RMSE vs. computational effort on the test set, over 100 generations.



(c) RMSE vs. computational effort on the test set, cut at the maximum value of the computational effort reached by all the algorithms being compared.

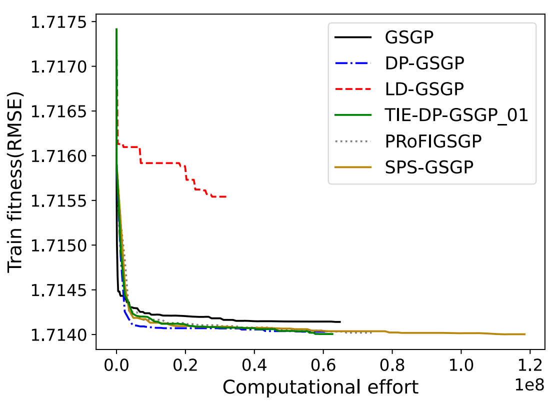
Fig. 9. Experimental results of the *f16* dataset.

7. Conclusion and future work

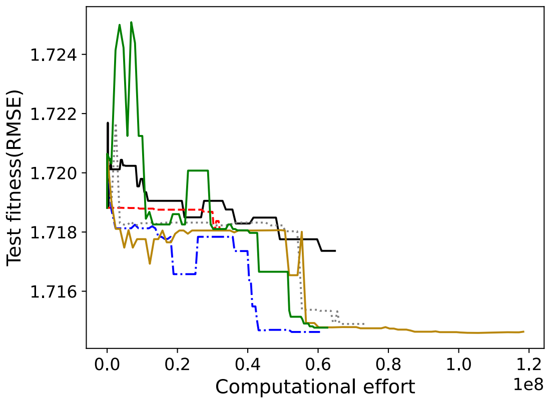
The use of dynamic populations is widespread in evolutionary computation. It can be seen either simply as a method to save computational effort (in the cases where the population size is steadily decreased during evolution) or, in slightly more sophisticated terms, as a way to preserve computational effort in some particular phases of evolution, to spend it more effectively in others (the case in most advanced proposed methods, where the population size can increase or decrease dynamically during evolution). Many studies exist that exploit dynamic populations in several EAs such as genetic algorithms and differential evolution. The particularity of GP, compared to other EAs, is that it evolves complex structures, such as computer programs, that have a dynamic and highly variable size. Furthermore, the evolving programs typically grow during the evolution, often giving rise to the well-known phenomenon of bloat, which cause a large waste of computational effort. Thus, GP seems a particularly appropriate algorithm to be improved, with methods aimed at saving computational effort, and it is only natural that studies apply dynamic populations to GP. However, up to date, no systematic study of dynamic populations in GSGP has been published. Nevertheless, GSGP is a particularly resource-greedy variant of GP, given that its genetic operators create offspring always larger than their parents, causing fast growth in the code that is evolving in the population.

The first element of novelty of this paper is that it applied dynamic populations to GSGP. Different approaches were tested on eight datasets, five of which correspond to real-world problems, while the remaining three are artificial benchmarks. Some of the studied approaches were inspired by methods previously been applied to other variants of EAs, while two of them were novel. The six algorithms used are:

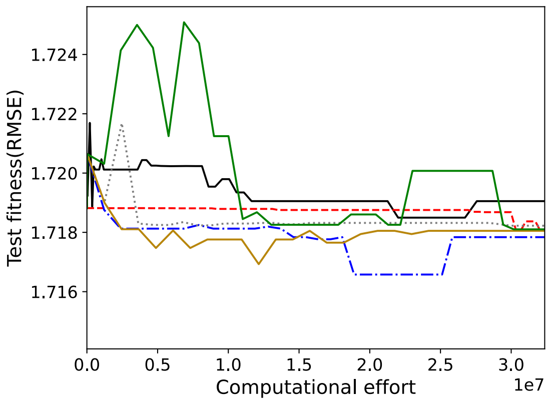
- *LD-GSGP*: a simple linear decrease of the population size, inspired by [30,31].



(a) RMSE vs. computational effort on the training set, over 100 generations.



(b) RMSE vs. computational effort on the test set, over 100 generations.



(c) RMSE vs. computational effort on the test set, cut at the maximum value of the computational effort reached by all the algorithms being compared.

Fig. 10. Experimental results of the *f4* dataset.

- *DP-GSGP*: an adaptation to GSGP of the DP algorithm presented in [34].
- *TIE-DP-GSGP*: a novel variant of the DP-GSGP, with a different fitness stagnation assessment method, based on the TIE measure presented in [50].
- *TIE-DP-GSGP-SEI*: another novel variation of the TIE-DP-GSGP algorithm, that stores the removed individuals in order to reintegrate the most promising ones when needed.
- *PRoFIGSGP*: an adaptation to GSGP of the algorithm presented in [6].
- *SPS-GSGP*: an adaptation to GSGP of the algorithm presented in [24].

Experimental results show that, given a common budget of computational effort, all the studied algorithms that use a dynamic population outperformed, or in the worst case performed comparably, to the standard version of GSGP. Among the different algorithms, the novel ones, based on the TIE measure (i.e., TIE-DP-GSGP and TIE-DP-GSGP-SEI), seem particularly interesting. In fact, besides being among the best-performing algorithms on all the studied problems, they are able to return robust models, that show a very good performance on unseen data, on problems where overfitting is a known issue, such as Bioavailability, PPB and LD50. Last but not least, the results have indicated a clear advantage in using dynamic populations in GSGP compared to standard GP.

In the future, we plan to study other criteria to decide on the variation of the population size, distinct from fitness. For instance, the adaption of the population size could be guided by other principles like diversity (genotypic diversity, as it already happens for other EAs, or semantic diversity, which would be novel and seems promising for GSGP), model complexity or model interpretability. Another interesting topic to extend this work would be to better investigate what individuals should be added to the population when the population size needs to grow. The advantages of using old individuals, such as in TIE-DP-GSGP-SEI, compared to random genetic material, encourage us to hypothesize that more effective strategies could be developed.

Table 4
p-values of the Wilcoxon test for pairwise comparison of GSGP and its variants. The GSGP algorithm is adopted from Moraglio's paper [48], whereas LD-GSGP, DP-GSGP, PRoFIGSGP and SPS-GSGP algorithms are inspired by Fernandez et al. [30], Vanneschi et al. [34], Eiben et al. [6], and Hallam et al. [24], respectively.

Problem		Algorithms					
		GSGP	LD-GSGP	DP-GSGP	PRoFIGSGP	SPS-GSGP	TIE-DP-GSGP_06
Bioav.	GSGP	–	**1.7e-06**	**1.7e-06**	**1.7e-06**	**1.7e-06**	**1.7e-06**
	LD-GSGP	**1.7e-06**	–	**1.7e-06**	0.339	**1.7e-06**	**1.7e-06**
	DP-GSGP	**1.7e-06**	**1.7e-06**	–	**4.3e-06**	0.734	0.766
	PRoFIGSGP	**1.7e-06**	0.339	**4.3e-06**	–	**3.5e-06**	**3.2e-06**
	SPS-GSGP	**1.7e-06**	**1.7e-06**	0.734	**3.5e-06**	–	0.614
	TIE-DP-GSGP_06	**1.7e-06**	**1.7e-06**	0.766	**3.2e-06**	0.614	–
		GSGP	LD-GSGP	DP-GSGP	PRoFIGSGP	SPS-GSGP	TIE-DP-GSGP-SEI_04
Boston	GSGP	–	**1.9e-06**	0.125	0.36	0.094	0.086
	LD-GSGP	**1.9e-06**	–	**1.7e-06**	**2.6e-06**	**1.7e-06**	**1.9e-06**
	DP-GSGP	0.125	**1.7e-06**	–	0.107	0.469	0.642
	PRoFIGSGP	0.360	**2.6e-06**	0.107	–	0.088	0.086
	SPS-GSGP	0.094	**1.7e-06**	0.469	0.088	–	0.845
	TIE-DP-GSGP_06	0.0859	**1.9e-06**	0.642	0.086	0.845	–
		GSGP	LD-GSGP	DP-GSGP	PRoFIGSGP	SPS-GSGP	TIE-DP-GSGP-SEI_04
Concrete	GSGP	–	**1.7e-06**	**5.79e-05**	**3.11e-05**	**6.89e-05**	**4.86e-05**
	LD-GSGP	**1.7e-06**	–	**1.7e-06**	**1.9e-06**	**1.7e-06**	**1.7e-06**
	DP-GSGP	**5.79e-05**	**1.7e-06**	–	**1e-04**	0.82	0.02
	PRoFIGSGP	**3.11e-05**	**1.9e-06**	**1e-04**	–	**4.58e-05**	0.053
	SPS-GSGP	**6.89e-05**	**1.7e-06**	0.82	**4.58e-05**	–	0.013
	TIE-DP-GSGP_04	**4.86e-05**	**1.7e-06**	0.024	0.053	0.013	–
		GSGP	LD-GSGP	DP-GSGP	PRoFIGSGP	SPS-GSGP	TIE-DP-GSGP_05
LD50	GSGP	–	0.644	**4.3e-06**	**0.002**	**0.001**	**1.9e-06**
	LD-GSGP	0.644	–	**3.2e-06**	**0.004**	**0.005**	**1.7e-06**
	DP-GSGP	**4.3e-06**	**3.2e-06**	–	**0.003**	**0.002**	0.075
	PRoFIGSGP	**0.002**	**0.004**	**0.003**	–	0.837	**2.4e-04**
	SPS-GSGP	**0.0014839**	**0.0046818**	**0.002278**	0.8373006	–	**6.89e-05**
	TIE-DP-GSGP_05	**1.9e-06**	**1.7e-06**	0.075	**2.4e-04**	**6.89e-05**	–
		GSGP	LD-GSGP	DP-GSGP	PRoFIGSGP	SPS-GSGP	TIE-DP-GSGP-SEI_04
PPB	GSGP	–	**5.7e-04**	**1e-04**	**1.2e-04**	**1.1e-04**	**5.31e-05**
	LD-GSGP	**5.7e-04**	–	0.125	0.106	0.159	0.017
	DP-GSGP	**1e-04**	0.125	–	0.905	0.77	0.068
	PRoFIGSGP	**1.2e-04**	0.106	0.905	–	0.820	0.150
	SPS-GSGP	**1.1e-04**	0.159	0.77	0.82	–	0.37
	TIE-DP-GSGP-SEI_05	**5.31e-05**	0.017	0.068	0.15	0.37	–
		D GSGP	LD-GSGP	DP-GSGP	PRoFIGSGP	SPS-GSGP	TIE-DP-GSGP-SEI_01
F11	GSGP	–	0.072	**1.8e-05**	**9.3e-06**	**3.9e-06**	**8.5e-06**
	LD-GSGP	0.072	–	**1.8e-05**	**4.17e-05**	**1.8e-05**	**1.02e-05**
	DP-GSGP	**1.8e-05**	**1.8e-05**	–	0.309	0.781	0.441
	PRoFIGSGP	**9.3e-06**	**4.17e-05**	0.309	–	0.36	0.453
	SPS-GSGP	**3.9e-06**	**1.8e-05**	0.781	0.36	–	0.614
	TIE-DP-GSGP_01	**8.5e-06**	**1.02e-05**	0.441	0.453	0.614	–
		GSGP	LD-GSGP	DP-GSGP	PRoFIGSGP	SPS-GSGP	TIE-DP-GSGP_02
F16	GSGP	–	**0.001**	0.254	0.254	0.254	0.254
	LD-GSGP	**0.001**	–	**1.1e-04**	**1.1e-04**	**1.1e-04**	**1.1e-04**
	DP-GSGP	0.254	**1.1e-04**	–	0.128	0.176	0.235
	PRoFIGSGP	0.254	**1.1e-04**	0.128	–	0.670	0.671
	SPS-GSGP	0.254	**1.1e-04**	0.176	0.670	–	0.752
	TIE-DP-GSGP_01	0.254	**1.1e-04**	0.235	0.671	0.752	–
		GSGP	LD-GSGP	DP-GSGP	PRoFIGSGP	SPS-GSGP	TIE-DP-GSGP_02
F4	GSGP	–	0.254	0.453	0.975	0.688	0.894
	LD-GSGP	0.254	–	0.943	0.185	0.959	0.644
	DP-GSGP	0.453	0.943	–	0.184	0.552	0.393
	PRoFIGSGP	0.975	0.185	0.184	–	0.569	0.658
	SPS-GSGP	0.688	0.959	0.552	0.569	–	0.820
	TIE-DP-GSGP_01	0.894	0.644	0.393	0.658	0.820	–

CRediT authorship contribution statement

Conceptualization: DF, LV, IB; Methodology: DF, LV, IB; Software: DF, IB; Validation: DF, LV, IB; Formal analysis: DF, LV, IB; Investigation: DF, LV, IB; Resources: DF, LV, IB; Data curation: DF, LV, IB; Writing draft preparation: DF, LV, IB; Supervision: LV. All authors have read and agreed to the submitted version of the manuscript.

Table 5
Comparison of the variants of GSGP and the variants of GP on the different test problems, together with the *p*-values (in bold when the difference is statistically significant ($p < 0.05$)). The GSGP algorithm is adopted from Moraglio's paper [48], while the LD-GP, DP-GP, PRoFIGP, and SPS-GP algorithms as well as their adaptation to GSGP, are inspired by Fernandez et al. [30], Vanneschi et al. [34], Eiben et al. [6], and Hallam et al. [24] respectively.

Problem	GSGP and its variations	Test fitness	GP and its variations	Test fitness	*p-values*
Bioavailability	GSGP	45.028	GP	37.227	**0.025**
	DP-GSGP	30.368	DP-GP	38.715	**1.9e-06**
	LD-GSGP	36.795	LD-GP	38.557	**0.036**
	PRoFIGSGP	42.181	PRoFIGP	37.771	0.22
	SPS-GSGP	30.376	SPS-GP	42.425	**1.7e-06**
	TIE-DP-GSGP_06	30.376			
Boston	GSGP	8.362	GP	5.86	**2.12e-06**
	DP-GSGP	8.667	DP-GP	5.73	**3e-04**
	LD-GSGP	6.824	LD-GP	7.061	**1.7e-04**
	PRoFIGSGP	8.47	PRoFIGP	7.231	**3.2e-06**
	SPS-GSGP	8.623	SPS-GP	8.375	0.13
	TIE-DP-GSGP_06	8.56			
Concrete	GSGP	19.167	GP	14.473	**4e-04**
	DP-GSGP	15.817	DP-GP	15.582	0.558
	LD-GSGP	12.143	LD-GP	15.261	**1.9e-06**
	PRoFIGSGP	15.724	PRoFIGP	17.682	**0.007**
	SPS-GSGP	15.866	SPS-GP	20.017	**7.6e-06**
	TIE-DP-GSGP_04	15.669			
LD50	GSGP	2236.109	GP	2129.449	0.009
	DP-GSGP	2116.802	DP-GP	2088.826	0.082
	LD-GSGP	2253.352	LD-GP	2168.068	**0.003**
	PRoFIGSGP	2198.44	PRoFIGP	2107.628	0.135
	SPS-GSGP	2229.225	SPS-GP	2189.461	0.44
	TIE-DP-GSGP_05	2043.26			
PPB	GSGP	34.163	GP	30.185	**0.001**
	DP-GSGP	31.005	DP-GP	31.261	0.64
	LD-GSGP	33.123	LD-GP	31.417	0.86
	PRoFIGSGP	32.155	PRoFIGP	30.776	0.7
	SPS-GSGP	32.224	SPS-GP	32.77	0.19
	TIE-DP-GSGP_05	30.606			
f11	GSGP	2.361	GP	0.633	0.068
	DP-GSGP	0.236	DP-GP	0.663	**3.8e-06**
	LD-GSGP	0.621	LD-GP	1.145	0.1
	PRoFIGSGP	0.26	PRoFIGP	2.025	**1.2e-05**
	SPS-GSGP	0.241	SPS-GP	1.306	**1.1e-05**
	TIE-DP-GSGP_01	0.243			
f4	GSGP	1.719	GP	1.719	0.44
	DP-GSGP	1.718	DP-GP	1.719	0.909
	LD-GSGP	1.718	LD-GP	1.719	0.106
	PRoFIGSGP	1.71	PRoFIGP	1.719	0.614
	SPS-GSGP	1.718	SPS-GP	1.719	0.75
	TIE-DP-GSGP_01	1.718			
f16	GSGP	3.273	GP	7.281	**0.002**
	DP-GSGP	2.703	DP-GP	5.082	**2e-04**
	LD-GSGP	6.35	LD-GP	6.475	0.45
	PRoFIGSGP	2.703	PRoFIGP	4.424	0.001
	SPS-GSGP	2.703	SPS-GP	3.553	0.106
	TIE-DP-GSGP_01	2.703			

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.

Data availability

Data will be made available on request.

Acknowledgements

This work was supported by national funds through FCT (Fundação para a Ciência e a Tecnologia), under the project – UIDB/04152/2020 – Centro de Investigação em Gestão de Informação (MagIC)/NOVA IMS.

References

[1] L. Vanneschi, S. Silva, Lectures on Intelligent Systems, Natural Computing Series, Springer, 2023.

[2] A. Eiben, R. Hinterding, Z. Michalewicz, Parameter control in evolutionary algorithms, IEEE Trans. Evol. Comput. 3 (2) (1999) 124–141, https://doi.org/10.1109/4235.771166.

[3] S. Oh, W.-H. Suh, C.-W. Ahn, Self-adaptive genetic programming for manufacturing big data analysis, Symmetry 13 (4) (2021), https://doi.org/10.3390/sym13040709.

[4] I. Bakurov, M. Castelli, O. Gau, F. Fontanella, L. Vanneschi, Genetic programming for stacked generalization, Swarm Evol. Comput. 65 (2021) 100913, https://doi.org/10.1016/j.swevo.2021.100913.

[5] T. Bäck, A.E. Eiben, N.A.L. van der Vaart, An emperical study on gas “without parameters”, in: M. Schoenauer, K. Deb, G. Rudolph, X. Yao, E. Lutton, J.J. Merelo, H.-P. Schwefel (Eds.), Parallel Problem Solving from Nature – PPSN VI, Springer Berlin Heidelberg, Berlin, Heidelberg, 2000, pp. 315–324.

[6] A.E. Eiben, E. Marchiori, V.A. Valkó, Evolutionary algorithms with on-the-fly population size adjustment, in: X. Yao, E.K. Burke, J.A. Lozano, J. Smith, J.J. Merelo-Guervós, J.A. Bullinaria, J.E. Rowe, P. Tiňo, A. Kabán, H.-P. Schwefel (Eds.), Parallel Problem Solving from Nature – PPSN VIII, Springer Berlin Heidelberg, Berlin, Heidelberg, 2004, pp. 41–50.

[7] A. Piotrowski, L-shade optimization algorithms with population-wide inertia, Inf. Sci. 468 (08 2018), https://doi.org/10.1016/j.ins.2018.08.030.

[8] R.C. Lonsinger, J.R. Adams, L.P. Waits, Evaluating effective population size and genetic diversity of a declining kit fox population using contemporary and historical specimens, Ecol. Evol. 8 (23) (2018) 12011–12021, https://doi.org/10.1002/ece3.4660.

[9] J.R. Koza, Genetic Programming: On the Programming of Computers by Means of Natural Selection, vol. 1, MIT press, 1992.

[10] S. Silva, E. Costa, Dynamic limits for bloat control in genetic programming and a review of past and current bloat theories, Genet. Program. Evol. Mach. 10 (2) (2009) 141–179, https://doi.org/10.1007/s10710-008-9075-9.

[11] 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, https://doi.org/10.1007/s10710-011-9150-5.

[12] A. Moraglio, K. Krawiec, C. Johnson, Geometric semantic genetic programming, in: C. Coello, V. Cutello, K. Deb, S. Forrest, G. Nicosia, M. Pavone (Eds.), Parallel Problem Solving from Nature – PPSN XII, in: Lecture Notes in Computer Science, vol. 7491, Springer Berlin Heidelberg, 2012, pp. 21–31.

[13] L. Vanneschi, An Introduction to Geometric Semantic Genetic Programming, Springer International Publishing, Cham, 2017, pp. 3–42.

[14] M. Castelli, S. Silva, L. Vanneschi, A C++ framework for geometric semantic genetic programming, Genet. Program. Evol. Mach. 16 (1) (2015) 73–81.

[15] 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, NY, USA, 2015, pp. 999–1006.

[16] L. Trujillo, J.M. Muñoz Contreras, D.E. Hernandez, M. Castelli, J.J. Tapia, GSGP-CUDA – a CUDA framework for geometric semantic genetic programming, SoftwareX 18 (2022) 101085, https://doi.org/10.1016/j.softx.2022.101085.

[17] G. Pietropolli, L. Manzoni, A. Paoletti, M. Castelli, Combining geometric semantic GP with gradient-descent optimization, in: E. Medvet, G. Pappa, B. Xue (Eds.), Genetic Programming, Springer International Publishing, Cham, 2022, pp. 19–33.

[18] L. Vanneschi, I. Bakurov, M. Castelli, An initialization technique for geometric semantic GP based on demes evolution and despeciation, in: 2017 IEEE Congress on Evolutionary Computation (CEC), 2017, pp. 113–120.

[19] I. Bakurov, L. Vanneschi, M. Castelli, F. Fontanella, EDDA-V2 – an improvement of the evolutionary demes despeciation algorithm, in: A. Auger, C.M. Fonseca, N. Lourenço, P. Machado, L. Paquete, D. Whitley (Eds.), Parallel Problem Solving from Nature – PPSN XV, Springer International Publishing, Cham, 2018, pp. 185–196.

[20] M. Castelli, L. Manzoni, S. Silva, L. Vanneschi, A. Popovič, The influence of population size in geometric semantic GP, Swarm Evol. Comput. 32 (2017) 110–120, https://doi.org/10.1016/j.swevo.2016.05.004.

[21] J. Arabas, Z. Michalewicz, J. Mulawka, GAVaPS – a genetic algorithm with varying population size, in: Proceedings of the First IEEE Conference on Evolutionary Computation, IEEE World Congress on Computational Intelligence, vol. 1, 1994, pp. 73–78.

[22] D.E. Goldberg, Genetic Algorithms in Search Optimization and Machine Learning, 1988.

[23] G.R. Harik, F.G. Lobo, A parameter-less genetic algorithm, in: Proceedings of the 1st Annual Conference on Genetic and Evolutionary Computation – Volume 1, GECCO’99, Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1999, pp. 258–265.

[24] J.W. Hallam, O. Akman, F. Akman, Genetic algorithms with shrinking population size, Comput. Stat. 25 (2010) 691–705.

[25] F.G. Lobo, C.F. Lima, On the utility of the multimodal problem generator for assessing the performance of evolutionary algorithms, in: Proceedings of the 8th Annual Conference on Genetic and Evolutionary Computation, GECCO ’06, Association for Computing Machinery, New York, NY, USA, 2006, pp. 1233–1240.

[26] R. Tanabe, A.S. Fukunaga, Improving the search performance of shade using linear population size reduction, in: 2014 IEEE Congress on Evolutionary Computation (CEC), 2014, pp. 1658–1665.

[27] N. Awad, M. Ali, P. Suganthan, Ensemble of parameters in a sinusoidal differential evolution with niching-based population reduction, Swarm Evol. Comput. 39 (09 2017), https://doi.org/10.1016/j.swevo.2017.09.009.

[28] W. Bingchuan, Z.-Y. Shui, Y. Feng, Z. Ma, Evolutionary algorithm with dynamic population size for constrained multiobjective optimization, Swarm Evol. Comput. 73 (2022) 101104, https://doi.org/10.1016/j.swevo.2022.101104.

[29] O. Montiel Ross, O. Castillo, P. Melin, R. Sepúlveda, Intelligent Control of Dynamic Population Size for Evolutionary Algorithms, 2006, pp. 551–557.

[30] F. Fernandez, L. Vanneschi, M. Tomassini, The effect of plagues in genetic programming: a study of variable-size populations, in: C. Ryan, T. Soule, M. Keijzer, E. Tsang, R. Poli, E. Costa (Eds.), Genetic Programming, Springer Berlin Heidelberg, Berlin, Heidelberg, 2003, pp. 317–326.

[31] F. Fernandez, M. Tomassini, L. Vanneschi, Saving computational effort in genetic programming by means of plagues, in: The 2003 Congress on Evolutionary Computation, vol. 3, CEC ’03, 2003, pp. 2042–2049.

[32] P. Kouchakpour, A. Zaknich, T. Braunl, Dynamic population variation in genetic programming, Inf. Sci. 179 (2009) 1078–1091, https://doi.org/10.1016/j.ins.2008.12.009.

[33] Y.-y. Tao, J. Cao, M.-L. Li, Genetic programming using dynamic population variation for computational efforts reduction in system modeling, J. Shanghai Jiaotong Univ. 17 (04 2012), https://doi.org/10.1007/s12204-012-1251-7.

[34] L. Vanneschi, G. Cuccu, Reconstructing dynamic target functions by means of genetic programming using variable population size, in: Computational Intelligence, vol. 343, 2011, pp. 121–134.

[35] I. Bakurov, M. Castelli, F. Fontanella, A. Scotto di Freca, L. Vanneschi, A novel binary classification approach based on geometric semantic genetic programming, Swarm Evol. Comput. 69 (2022) 101028, https://doi.org/10.1016/j.swevo.2021.101028.

[36] L. Vanneschi, S. Silva, M. Castelli, L. Manzoni, Geometric Semantic Genetic Programming for Real Life Applications, Springer New York, New York, NY, 2014, pp. 191–209.

[37] I. Gonçalves, S. Silva, C.M. Fonseca, On the generalization ability of geometric semantic genetic programming, in: P. Machado, M.I. Heywood, J. McDermott, M. Castelli, P. García-Sánchez, P. Burelli, S. Risi, K. Sim (Eds.), Genetic Programming, Springer International Publishing, Cham, 2015, pp. 41–52.
[38] A. Moraglio, An efficient implementation of GSGP using higher-order functions and memoization, in: Semantic Methods in Genetic Programming, Workshop at Parallel Problem Solving from Nature, 2014.
[39] 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, ACM, New York, NY, USA, 2018, pp. 1151–1158.
[40] I. Bakurov, M. Buzzelli, M. Castelli, L. Vanneschi, R. Schettini, General purpose optimization library (GPOL): a flexible and efficient multi-purpose optimization library in Python, Appl. Sci. 11 (11) (2021), https://doi.org/10.3390/app11114774.
[41] M. Keijzer, Improving symbolic regression with interval arithmetic and linear scaling, in: Genetic Programming, 2003, pp. 70–82.
[42] D. Harrison, D.L. Rubinfeld, Hedonic housing prices and the demand for clean air, J. Environ. Econ. Manag. 5 (1) (1978) 81–102, https://doi.org/10.1016/0095-0696(78)90006-2.
[43] I.-C. Yeh, Modeling of strength of high-performance concrete using artificial neural networks, Cem. Concr. Res. 28 (12) (1998) 1797–1808, https://doi.org/10.1016/S0008-8846(98)00165-3.
[44] F. Archetti, S. Lanzeni, V. Messina, L. Vanneschi, Genetic programming for computational pharmacokinetics in drug discovery and development, Genet. Program. Evol. Mach. 8 (2007) 413–432, https://doi.org/10.1007/s10710-007-9040-z.
[45] G. Dick, A.P. Rimoni, P.A. Whigham, A re-examination of the use of genetic programming on the oral bioavailability problem, in: Proceedings of the 2015 Annual Conference on Genetic and Evolutionary Computation, 2015, pp. 1015–1022.
[46] J. McDermott, D.R. White, S. Luke, L. Manzoni, M. Castelli, L. Vanneschi, W. Jaskowski, K. Krawiec, R. Harper, K. De Jong, U.-M. O'Reilly, Genetic programming needs better benchmarks, in: Proceedings of the 14th Annual Conference on Genetic and Evolutionary Computation, GECCO '12, Association for Computing Machinery, New York, NY, USA, 2012, pp. 791–798.
[47] J. McDermott, G. Kronberger, P. Orzechowski, L. Vanneschi, L. Manzoni, R. Kalkreuth, M. Castelli, Genetic programming benchmarks: looking back and looking forward, SIGEVOlution 15 (3) (dec 2022), https://doi.org/10.1145/3578482.3578483.
[48] A. Moraglio, K. Krawiec, C. Johnson, Geometric semantic genetic programming, in: Parallel Problem Solving from Nature – PPSN XII, vol. 7491, 2012, pp. 21–31.
[49] C. Bonferroni, Teoria statistica delle classi e calcolo delle probabilità, in: Pubblicazioni del R. Istituto superiore di scienze economiche e commerciali di Firenze, Seeber, 1936.
[50] Ivo Gonçalves, Sara Silva, Carlos M. Fonseca, Mauro Castelli, Unsure when to stop?, in: Proceedings of the Genetic and Evolutionary Computation Conference, ACM, 2017, https://doi.org/10.1145/3071178.3071328.