Applied Soft Computing 122 (2022) 108825

Contents lists available at ScienceDirect

Applied Soft Computing

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

Semantic schema based genetic programming for symbolic regression

Zahra Zojaji [a,*], Mohammad Mehdi Ebadzadeh [b], Hamid Nasiri [b]

[a] *Department of Computer Engineering, University of Isfahan, Isfahan, Iran*
[b] *Department of Computer Engineering, Amirkabir University of Technology (Tehran Polytechnic), Tehran, Iran*

ARTICLE INFO

Article history:
Received 23 June 2021
Received in revised form 2 March 2022
Accepted 1 April 2022
Available online 22 April 2022

Keywords:
Genetic programming
Schema theory
Locality
Semantic genetic programming
Symbolic regression

ABSTRACT

Despite the empirical success of Genetic programming (GP) in various symbolic regression applications, GP is not still known as a reliable problem-solving technique in this domain. Non-locality of GP representation and operators causes ineffectiveness of its search procedure. This study employs semantic schema theory to control and guide the GP search and proposes a local GP called semantic schema-based genetic programming (SBGP). SBGP partitions the semantic search space into semantic schemas and biases the search to the significant schema of the population, which is gradually progressing towards the optimal solution. Several semantic local operators are proposed for performing a local search around the significant schema. In combination with schema evolution as a global search, the local in-schema search provides an efficient exploration–exploitation control mechanism in SBGP. For evaluating the proposed method, we use six benchmarks, including synthesized and real-world problems. The obtained errors are compared to the best semantic genetic programming algorithms, on the one hand, and data-driven layered learning approaches, on the other hand. Results demonstrate that SBGP outperforms all mentioned methods in four out of six benchmarks up to 87% in the first set and up to 76% in the second set of experiments in terms of generalization measured by root mean squared error.

1. Introduction

As a kind of evolutionary algorithm, genetic programming is inspired by natural evolution. Although the motivation behind its innovation was to achieve the success of natural evolution in problem-solving and has been applied to solve numerous real-world problems [1–4], some features of this algorithm contradict natural evolution and keep genetic programming from having an effective search within search space. Non-locality and non-gradual optimization are important samples of such features.

One of the features of natural evolution is its gradualism. According to Darwin's Theory, evolution is a slow but gradual process in which environmental compatibility occurs via gradual corrections on the individuals. Natural selection is considered the most important mechanism of evolution [5]. Darwin states that natural selection takes place only when consequent, but slight changes happen. Evolution can never have significant and sudden mutations or saltation. It should proceed with small, steady, and indeed slow steps [6]. He also takes into consideration the gradualism of changes in the phenotype. Although gradualism is one of the main principles of Darwin's theory of evolution [7], it has not been included in standard genetic programming.

The locality is both the feature of representation and operators. It has been shown in previous researches [8–14] that the locality of representation and genetic operators significantly affect an evolutionary algorithm. The locality of representation means that small changes in genotype cause small changes of phenotype. It means that the neighborhood is preserved during genotype–phenotype mapping and referred to as the continuity of the genotype–phenotype map. This map is continuous when there is a causal relationship between the data structure of an individual and its equivalent function. On the other hand, a genetic operator is called local when the offspring it produces is semantically close to the parents. Non-locality within the evolutionary algorithms domain is stated under various titles, such as the absence of causality and imbalance between exploration and exploitation.

The representation of standard genetic programming is not local [15] because small changes in programs cause intense changes in their behavior [16]. For instance, in Fig. 1 an individual has a tree view of $(*x(*x\ x))$ representing $F_1(x) = x_3$. If $*$ symbol (multiplication) is substituted with $+$ symbol (addition) in this individual's root, the resultant individual will be $((+x(*x\ x)))$. Although there has been a small change in genotype, the equivalent function of the individual turns from $F_1(x) = x_3$ to $F_2(x) = x_2 + x$, which is significantly different from F_1 in semantically.

In standard genetic programming, operators also have weak locality (causality) [15]. In genetic programming, usually, the

* Corresponding author.
E-mail address: z.zojaji@eng.ui.ac.ir (Z. Zojaji).

https://doi.org/10.1016/j.asoc.2022.108825
1568-4946/© 2022 Elsevier B.V. All rights reserved.

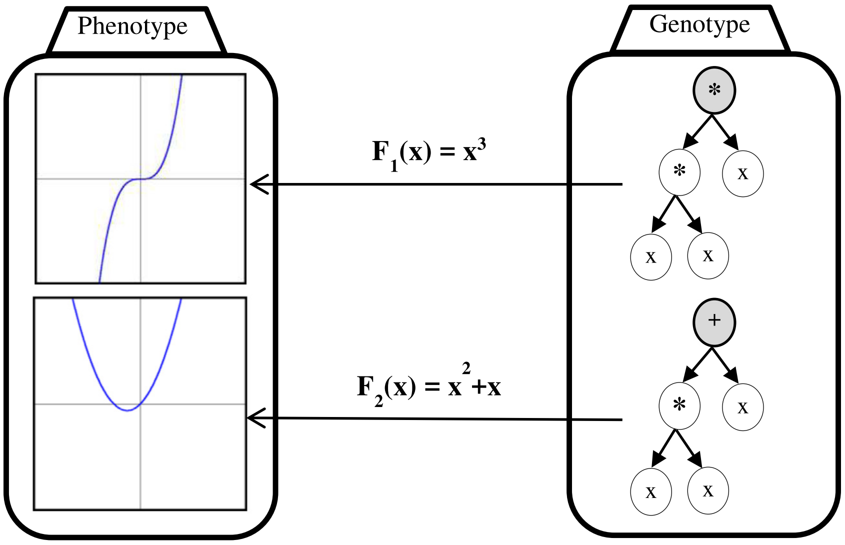

Fig. 1. An example of discontinuity of non-locality of GP representation.

offsprings that are specifically generated through recombination differ significantly from their parents, from both syntactic and semantic aspects. An essential reason for these operators' non-locality is that semantic units are not clear in genetic programming. In the genetic algorithm, genes play the role of semantic units. In contrast, in genetic programming, one node has no independent meaning on its own but becomes meaningful according to the context. Therefore, an almost-independent meaning has to be sought in subtrees and building blocks, considered sub-solutions of the final solution. Yet, there is still no specific definition for semantic building blocks, and recombination, the main operator of genetic programming, has no information of semantic borders of the subtrees. As a result, it chooses the crossover points randomly and usually destroys the building blocks. Hence, the offspring has a significant semantic difference from the parents. The difference is not beneficial, as the fitted parts of the parents are often destructed not combined to produce better offsprings. In other words, the main mechanism of evolutionary algorithms is the gradual discovery of partial solutions and their automatic combination (using the operators) to achieve more complete solutions. However, when applied to the tree representation of genetic programming, this approach becomes quite challenging since recombination alters genomes or semantic units and decreases the chance of this operator's usefulness to a great extent [17]. Locality (causality) is a key feature, effective on exploration and exploitation of evolutionary algorithms [18]. The stronger the locality of an operator, the greater its exploitation, and the weaker its locality, the higher its exploration [15]. In turn, higher exploration causes greater diversity in the population; and higher exploitation results in lower diversity [19].

In this paper, we propose the semantic schema-based genetic programming (SBGP) algorithm. To solve the non-gradual evolution of genetic programming, it partitions the semantic search space into some sub-spaces and biases the search to one of these sub-spaces, gradually progressing towards the optimal solution. The proposed algorithm utilizes schema theory to define the partitions in a way that it divides the semantic search space into smaller sub-spaces. For biasing the search towards the schema, it introduces new local operators.

While many researchers used genetic programming for classification [20–22], we focus on the symbolic regression problem in this paper. The main idea of symbolic regression is finding equations that describe the variables relations. These formulas can provide scientifically meaningful models, particularly when combined with the domain knowledge. Although the idea of symbolic regression has existed for a long time, it has begun to make a noticeable impact on real world research areas such as medicine, physics, chemistry, biology, and engineering, in the last decade. Therefore, developing effective symbolic regression algorithms can make significant progress on successful solving of a wide range of real-world problems. In this paper, in addition to synthesized benchmarks, we applied the proposed method on four real world problems taken from the biology, engineering and pharmacokinetics domains to show the capabilities of SBGP in deploying in the real world applications.

The rest of this paper is organized as follows. Section 1 provides an introduction to the research problem. Section 2 explains the necessary background concepts. In Section 3, related researches are described. The proposed method is presented in Section 4. In Section 5, the experimental results are reported and analyzed. Finally, Section 6 is the conclusion of the paper.

2. Background

In this section, schema theory is first described, the notion of semantics is then clarified, and semantic schema is finally introduced.

2.1. Schema theory

A schema is a pattern of similarity, matching a set of search space points. The schema theorem describes how the number of schema instances changes through evolution. Based on the genetic algorithm's schema theory [23], Koza proposed the first genetic programming's schema theory in 1992 [24]. He defined a set of complete subtrees as schema, for instance, $H = (+xy)$, $(+x(\times xy))$. An individual matches H if it contains both subtrees. The schema theorem's general form is given in (1).

$$E[m(H, t+1)] = M\alpha(H, t). \tag{1}$$

Where $E[m(H, t+1)]$ is the expected number of instances of schema H in $(t+1)$th generation, M denotes the population size, and the probability of a newly generated individual instantiates H (the transition probability) denoted by α [24]. Schema theory in [24] can be used to calculate a lower bound for the expected number of individuals belonging to a particular schema.

The building block hypothesis was first proposed by Goldberg [25]; according to this hypothesis, GA works by combining building blocks, low order, short, and highly fit schemas [24]. The notion is extended in several works to introduce semantic building blocks. In [26], these building blocks are considered points of the semantic space with high value that frequently occur in the population.

2.2. Semantics

Semantics is a knowledge that determines the concept of syntactic symbols [27], demonstrating what the tree really does [28]. A tree's semantics refers to its behavior. The semantics of a tree in Boolean domains is defined in [29] by McPhee et al. in 2008. They represent the semantics as the tree's output values for all possible input values combinations. Yet, in the domain of real numbers, one cannot calculate the tree output for all possible inputs since there is an infinite number of possible input cases. Although for the real-valued domain, there is no widely accepted definition for semantics, in several researches including [27,29–33], a tree's semantics is regarded as its output vector for various input fitness cases. This definition is called sampling semantics [30]. According to this definition, an N-dimensional vector represents each tree in semantic space, where N denotes the number of fitness cases. Some researchers [34–36], rather than focusing on the program's output, used its execution's behavioral features to define semantics for the real-valued domain. In [26], the semantics of an individual is described in terms of the normalized mutual information between its output and the target.

2.3. Semantic schema

As described before, a traditional definition of schema refers to a subset of points in the search space that shares some syntactic characteristics [24], which we call *a syntactic schema*. Although the instances of a syntactic schema are syntactically similar, they do not behave similarly [26,37]. Since there are several major issues relying on syntactic schema theories to model the population's behavior, semantic schema theory was originally proposed in [26]. The semantic schema can reliably describe the semantically similar individuals who also behave similarly. In further researches [37,38], authors improved their proposed schema in terms of generalization and computational complexity. A comprehensive comparison between these schemata is provided in [38]. As discussed in [38], the Information Clustering-based Semantic Schema (ICSS) is the best proposed semantic schema in terms of generalization and computational complexity. In this work, the notion of semantics for a tree is considered the mutual information between its output and the target. For modeling schema instances, *semantic building blocks* were extracted and employed. Semantic building block refers to a semantics value that frequently occurs in promising individuals of the population when the subtrees of these individuals are mapped into semantic space. Building blocks propagate on the promising individuals of the population dependently, and combine to produce the target solution. A novel approach called *information-based clustering* was then utilized for clustering the discovered building blocks of the population.

A semantic schema is composed of two modules: (1) a cluster set of semantic building blocks (2) a schema instantiation function that describes the distribution model of building blocks in schema instance.

For extracting the major schema of an arbitrary population, called the *significant schema*, some steps should be taken, as illustrated in Fig. 2. First, the initial schema instances should be specified, which are considered individuals with more than average semantics of the population. Next, the semantic building blocks are extracted from the set of initial samples, and some schema-related parameters are estimated. After that, information-based clustering is applied to extracted building blocks. The instantiation function is then specified according to learned schema parameters. This function considers the effect of a building block on its cluster. Accordingly, a semantic schema is a set including all individuals with effective enough building blocks from various clusters. Refer to [38] for more details on information clustering-based semantic schema.

3. Related works

There have been numerous studies on integrating semantics within genetic programming. They have chiefly shown that the use of semantic**s** can improve the power of this algorithm. As discussed in Section 2.2, various definitions of semantics were proposed by researchers. The majority of previous genetic programming versions has ignored trees' semantics and has only operated in the syntactical space. Yet, the use of semantic methods in genetic programming has gained attention and extensive study in recent years. A great deal of work concentrated on incorporating semantics in computing diversity, initializing the population, selection strategy and bloat control mechanism which are described in Section 3.1. Many of the conducted researches in semantic genetic programming are dedicated to the development of semantic operators. These studies are divided into two main groups [28]: indirect and direct methods described in Sections 3.2 and 3.3 respectively. The former is based on trial and error and, using the operators, manipulating trees at the syntactic level to produce new offsprings. The generated trees enter the next generation population if they satisfy specific semantic constraints, which are usually defined concerning diversity, locality, and geometric relations within semantic space. The latter denotes operators that produce the offspring directly in semantic space.

3.1. Semantic methods

Some researchers have investigated semantic diversity, they defined this concept based on the distribution of fitness values [39] or tree behavior at runtime [40]. In [41], a new semantic diversity metric was proposed based on the transformed semantics of models. Many researchers [42] concluded that a semantically diverse initial population could increase genetic programming's performance. Pawlak and Krawiec [43] have presented a semantic method for initializing the population, suitable for genetic programming with geometric semantic recombination [31]. Galvan-Lopezet al. [44] introduced a simple method to consider trees' semantics in the selection process, aiming to enhance semantic diversity. Ruberto et al. [45] proposed SGP-DT, a new semantic genetic programming algorithm based on the dynamic target. Nguyen and Chu [46] proposed two semantic approximation-based methods for controlling bloat in genetic programming. Miranda et al. [47] introduced an instance selection method for reducing the dimensionality of geometric semantic genetic programming space.

3.2. Indirect methods

Some studies [29,32,48] have been conducted based on indirect methods with diversity constraints. In this way, the offspring should differ from their parents semantically. Semantic constraints in some researches are considered based on the locality. The goal of such methods is to produce offsprings with no significant semantic difference from their parents. Nguyen et al. [49] introduced various semantic aware crossovers (SAC) for Boolean problems. Afterward, they expanded these operators for

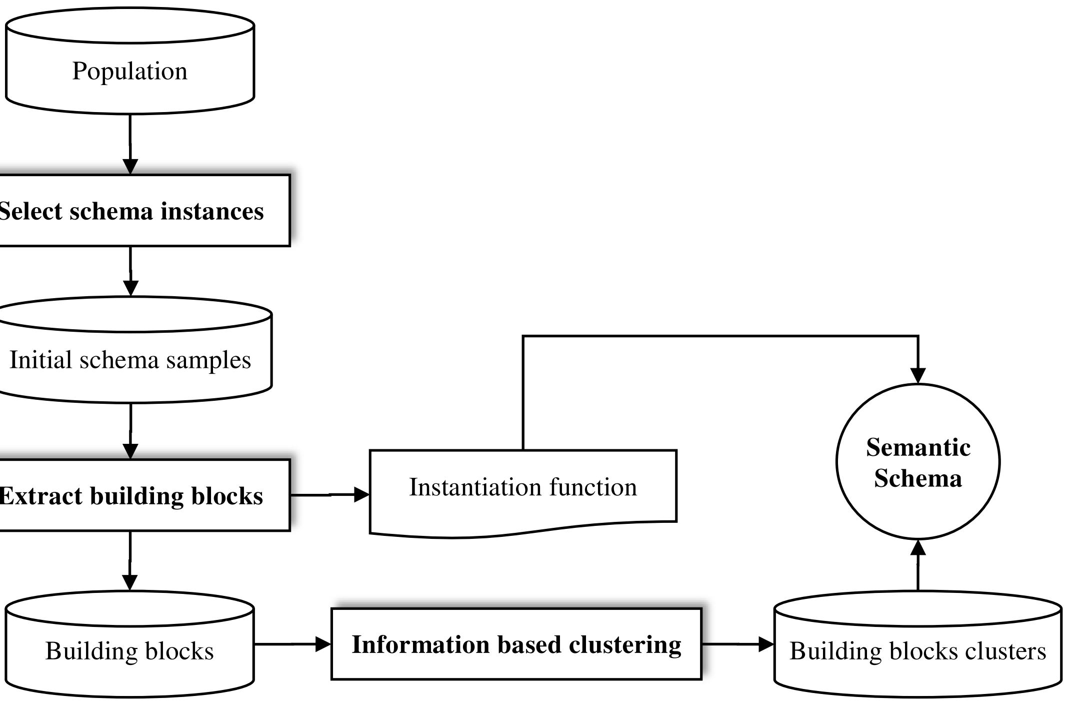

Fig. 2. The semantic schema extraction [38].

the domain of real numbers [50]. Instead of accurate semantics, they used an approximate one, in which semantics included a tree's output for a random set of input samples. Later, the authors presented Semantic Similarity-based crossover (SSC) [30], which emphasized local behavior of crossover and swapped some subtrees in the parents that differed semantically, but not significantly. Afterward, an improvement of SSC, the most semantic similarity-based crossover (MSSC), was proposed [51], and eventually, the semantic mutation was introduced [30]. The impact of SSC on population diversity got analyzed in [52]. As a result of this study, Nguyen et al. [53] found that increasing semantic locality would improve the performance of genetic programming.

Some of the conducted studies are focused on geometric semantic operators [27]. In these methods, the operators are designed to allow the produced offspring to occur in the semantic space between the parents. Krawiec and Lichocki [33] claimed that the offspring produced by a complete geometric crossover would be better than at least one of the parents. Due to the complexity of genotype–phenotype mapping, they proposed an Approximating Geometric Crossover (AGC). In this crossover, the offsprings with maximum semantic similarity to the parents will be accepted, even if they are not placed exactly between them. Moreover, a locally geometric semantic crossover [27] was presented that produced a bank of all possible subtrees until a specific depth. After selecting the parents' subtrees for swap, the closest subtree to their semantic average was selected from the bank and was replaced in the offspring. Later, Krawiec [54] proposed a crossover operator that produced medial offsprings. Afterward, two approximating geometric crossovers called AGX and RDO [55,56] were introduced that used semantic backpropagation to select suitable subtrees for swap. In the AGX operator, firstly, the parents' output average vector, known as M, is calculated. Then the crossover points are selected, and using backpropagation, the swapping subtree's semantics are computed in the parents to allow the offsprings to be similar to M as much as possible. Then the prebuilt library is searched for the most similar subtree to the calculated semantics, and it is substituted with parents' subtree. RDO operator is applied on one parent. At first, it randomly selects a subtree in the parent. Then, employing backpropagation, it determines this subtree's semantics in a way that after replacing in the parent, the tree's output will have the highest similarity with the problem's objective.

3.3. Direct methods

Direct semantic methods belong to the ones that produce the offspring directly at a semantic level. The idea of direct methods was first proposed in 2012 by Moraglio et al. [31]. They offered accurate geometric operators, called Semantic Geometric Crossovers (SGX), which searched the semantic space directly. In symbolic regression, the result of recombination in semantic space is a weighted average of parents' semantics. Offspring's semantic in the syntactic space could be realized by generating a tree that calculates the weighted average on top of the parents' root that cause the offsprings' size in each generation to face extreme growth. Some researchers [33,55] have proposed approximate geometric semantic recombinations to overcome this problem, which had a good performance like accurate geometric operators. A strategy to control the exponential growth of programs in [31] is to reduce the size of generated offsprings during the recombination, which was previously studied in [57]. Another proposed strategy is the direct search of the semantic space while saving the initial population simultaneously and applying changes to it during the evolution [58]. In this method, after the evolution, the best individual of the population is generated by applying some changes to its predecessors from the first generation to the last one. This kind of genetic programming was later tested on several real-world problems like [59,60]. In addition, a geometric mutation was designed and introduced in the semantic space [61], which produced offspring smaller than its parents or equal in size. Later on, Angwin et al. [62] presented a subtree semantic geometric crossover (SSGX), which approximated the parent's semantics with one of its smaller subtrees, then applied SGX. Moreover, in this method, the standard crossover has a probability of occurrence, so that diversity could be kept high. The effect of population size on geometric semantic genetic programming has been studied in [63] and the operators probabilities have been adjusted by the proposed algorithm of [64].

3.4. Layered learning genetic programming

In what follows, some of the best versions of genetic programming will be introduced that tried to make gradual evolution through data layering, have high generalization, and are employed for comparison with the proposed method.

Random Sampling Technique (RST) is proposed by Gonçalves et al. to control overfitting in genetic programming [65]. RST employs a random subset of training samples in each generation. Thus, the chance of survival in the population is higher for those members with higher performance over different subsets of training samples. Castelli et al. [66] designed a quantitative test on learning and generalization of genetic programming, thus proposing a criterion for function complexity, called "Graph-Based Complexity" (GBC). This criterion is based on the idea that functions that are more complex should have less degree of curvature than smoother ones. They introduced a fitness function based on this criterion to increase the generalization GP. Hein et al. [67] studied the influence of stage-based layered learning with incremental sampling. They then improved this method, using the Progressive Sampling (PS) to determine the initial set and the number of layers. PS provides a method to work with big data. Their results have shown that layered learning along with incremental sampling could increase learning speed without harming generalization. A modified version of genetic programming to improve generalization is VBLL-GP [68], in which some simpler datasets are generated by taking the root of initial data. At the beginning of the evolution, the simplest dataset is used and during the evolution, more complex data are gradually provided to the algorithm. SGP [69] is a gradual variant of GP that uses statistical information of the trees to improve generalization. A well-made tree is introduced as the tree whose output's correlation with target increases while moving from the leaf to the root. SGP search is limited in a way to find trees that are as well-made as possible. Thus, it limits trees' complexity. Recently, RCGP [70] presented by Hosseini Amini et al. is a variant of GP developed based on the imperialist competitive algorithm. RCGP also benefits from a set of predefined rules for producing better offsprings. In this algorithm, local search is first carried out on countries and then combined into a global search through operations like assimilation and revolution.

4. Proposed method

For solving the problem of non-gradual evolution of genetic programming, we propose SBGP, which partitions the semantic search space into some sub-spaces and biases the search to one of them that is changing during the evolution, gradually progressing towards the optimal solution. This approach conforms to the raised theories in natural evolution. Our introduced sub-spaces imply "species" in nature. In this paper, we employed the schema to play the role of the species concept in GP evolution so that reproduction takes place in one species and the species evolve gradually. In the definition given by Thompson [71], a species is a group of individuals detected using a series of constant and inducible properties and connected via hereditary and genetic relationships. In nature, recombination mostly occurs inside a species and the offspring belonging to the parents' species. In fact, species' evolution from one to another has rarely occurred in some specific cases with the passage of considerable time as all flora and fauna have appeared from primitive unicellular organisms. The proposed method has considered the species' alteration and evolution, which comply with the natural evolution process.

We utilize schema theory to partition the semantic search space into sub-spaces. This paper regards locality as being a member of the schema. Instead of considering producing offsprings between or close to the parents, it takes both parents and their offspring as members of the schema so that they will have a meaningful local relationship. In a nutshell, for the following reasons, the schema theory has been chosen as a tool to make genetic programming's search gradual: (1) presenting sub-spaces from the search space, (2) strong theoretical background, and (3) modeling the operators' effect in the form of mathematical equations.

After partitioning the semantic search space into sub-spaces, the next step will be to bias the search towards the schema, achievable by introducing local operators. To localize genetic programming's search, we introduce several local operators, each with a probability of occurrence. In fact, the appropriate algorithm's prerequisite is the presence of schema-biasing local operators.

4.1. SBGP algorithm

In this section, Schema-Based Genetic Programming (SBGP) is introduced and evaluated. SBGP algorithm uses schema theory to divide the space into some sub-spaces. Furthermore, the proposed semantic schema is based on the building blocks, some genes that produce appropriate semantic characteristics in an individual. To produce individuals matching the schema means to propel the operators towards producing offsprings that actively incorporate the building blocks of the schema within themselves. Fig. 3 shows the flowchart of the SBGP algorithm. Various steps of this algorithm are described below:

- **Generating the initial population** The initial population is generated employing the ramped-half and half method with limited depth in the first step. The initial samples of the schema should be identified in this population.
- **Semantic schema Extraction:** Once the initial population is generated, the individuals, semantically higher than the population average, are taken as preliminary samples of the schema, and ICSS Schema is extracted from the mentioned population as it was illustrated in Fig. 2. More precisely, at first, the building blocks are extracted from the population and clustered. Afterward, the obtained clusters are used to form membership functions and identify schema samples.
- **Generating the next generation with schema bias:** In this step, the local operators, introduced in the next section, are used by considering the corresponding probability of occurrence. Moreover, the standard recombination and mutation also have a chance to occur to keep the population diverse. Some of the introduced local operators need schema instances as the parents, some operate on any parent, and some generate the samples without any parent. In fact, in this step, the search mechanism should exploit the search space; however, in the case of the low diverse population, more exploitation causes premature convergence. As a result, we adjusted the probability of using standard recombination and mutation inversely proportional to the population diversity. Given in (2), this diversity is a semantic diversity, calculated as the standard deviation of the population's individuals' semantic. In addition, (3) gives the probability of using standard operators. This probability is calculated adaptively during the evolution. At the beginning of the evolution, when the population's diversity is high, the probability of using standard operators is low. As generations pass, this probability increases. More offsprings with standard operators are generated to establish the necessary potentials for the next schemas and provide an appropriate balance between exploration and exploitation.

$$\mathrm{diversity}(pop^g) = \sqrt{\frac{1}{popSize}\sum_{i\in pop}(s(i) - \bar{s}(i))^2} \tag{2}$$

$$p_{std} = -\frac{\log(\mathrm{diversity}(pop))}{10} \tag{3}$$

- **Conditions for schemas transition:** In natural evolution, a transition from one species to another occurs as an effect of events such as a change in gene frequency, ecological environment changes, and intensive mutations. Various genetic theories believe that this transition might occur smoothly (Phyletic Gradualism) or suddenly (Punctuated Equilibria) [72]. The former is equivalent to implicit bias towards a new schema, while the latter is equivalent to explicit mutation of the schema. Considering the flowchart in Fig. 3, the conditions of transition to the next schema should be evaluated in each generation. These conditions should be defined in a way to show whether the evolution within the schema has taken place sufficiently or not. The greater the local evolution within the schema, the more the exploitation and the less the population diversity. Therefore, we define transition criterion as semantic diversity of the population, given in (2). Afterward, *diversityRatio* shows the proportion of current population diversity in generation g to the average diversity of recent generations, provided in (4). The diversity is averaged over a sliding window, with length w. If *diversityRatio* is less than the threshold *diversityThr*, a new schema is extracted from the population. Given that the population's average fitness increases through generations, the new schema's fitness is higher than the previous one. As a matter of fact, the schema itself evolves. Schema transition propels the search towards exploration to consider new parts of the search space.

$$diversityRatio = \frac{\mathrm{diversity}(pop^g)}{\frac{1}{|w|}\sum_{i=1}^{w}\mathrm{diversity}(pop^{g-i})} \tag{4}$$

- **Fitness**

In SBGP evolving is performed in semantic space which leads to discover individuals that are semantically equal to the optimal solution and high semantic value does not necessarily denotes low error. Since SBGP is presented for symbolic regression domain, a linear correction is applied to individuals' output prior to the fitness calculation. This correction is not valid in other domains like classification but it can be customized in further researches.

In order to evaluate the fitness of individuals in the population, linear regression coefficients of each individual is calculated and incorporated into the individual's function. Then root mean squared error (RMSE) over the modified individual is considered as the performance measure. Applying regression transformation as a linear correction on the individuals with a high semantic value also results in obtaining high fitness calculated in terms of RMSE. This transformation, however, does not involve the genotype space of the tree, but is considered only in calculation of individuals' fitness.

We denote the output of tree t by Y for input samples, target output by Z, and the output of tree after transformation by Y' that is calculated using (5):

$$\begin{aligned} &Y' = \alpha Y + \beta, \\ &\alpha = \frac{\mathrm{cov}(Y,Z)}{\mathrm{var}(Y)},\ \ \beta = \overline{Z} - a\overline{Y} \end{aligned} \tag{5}$$

where cov and var symbols stand for covariance and variance operations, respectively, α and β are the regression coefficients initially applied on the tree's output. Then the error is calculated using Y', as the transformed output of the tree.

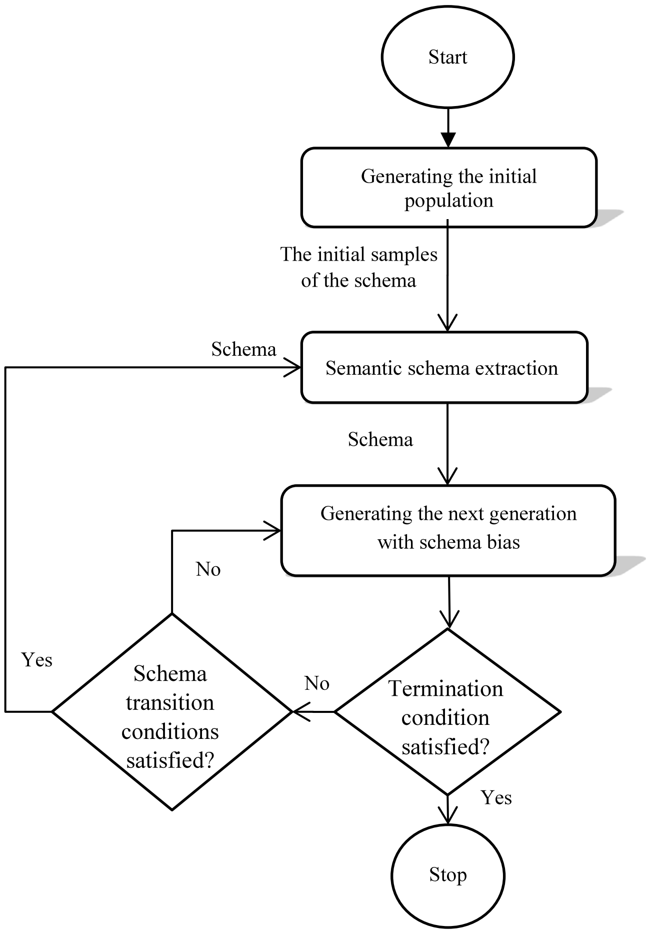
Start
Generating the initial population
The initial samples of the schema
Schema
Semantic schema extraction
Schema
Generating the next generation with schema bias
No
Yes
Schema transition conditions satisfied?
No
Termination condition satisfied?
Yes
Stop

Fig. 3. Flowchart of the SBGP algorithm.

The pseudo-code of the SBGP algorithm is shown in Fig. 4. In this pseudo-code, the parameters of genetic programming, training data, and validation data are given as input, and the best individual is returned as output.

4.2. Local operators

This paper introduces several local operators to localize the genetic programming's search. Each has a probability of occurrence in the SBGP algorithm. These operators encourage the building blocks to occur and prevent them from destruction. The proposed operators are: 1. indirect local recombination, 2. indirect local mutation, 3. direct local mutation, and 4. parentless child-production operators, each elaborated below. The goal of local recombination and mutation operators is to bias the offspring towards the schema. The aim of the parentless child-production operator is to generate individuals that are instances of the schema.

4.2.1. Indirect local recombination operator (ILRO)

This recombination is a single-child one that gets two schema-instances parents as input and generates an offspring, which is itself an instance of the schema. This operator uses a trial and error method, finding the recombination points in the parents, wherein at least one of the offsprings is a sample of the schema via the recombination of these points. If both offspring are samples of the schema, the one with a higher membership value in that schema is selected. Fig. 5 gives the pseudo-code of this recombination.

```
SBGP
Inputs:
        diversityThr, popSize, MaxGeneretion, xoProb, muProb, ...: GP parameters
        T: training data
        V: validation data
Outputs:
        best : the best individual
pop = generateInitialPopulation()
offspring = { }
pValError = inf
for g = 0 : MaxGeneretion
    evaluatePop(T)
    best = findBestAfterTransformation(pop, T)
    if error(best, V) > pValError then
        break;
    end
    pValError = error(best, V)
    diversityAvg = calculateAverageDiversityOfLastGenerations()
    diversity = calulateDiversity()
    diversityRatio = diversity/diversityAvg
    if diversityRatio < diversityThr || g=0 then
        H = extractSchema3(pop)
    end
    schInstances = getSchemaInstances(H, pop)
    for i = 0 : popSize
        r = rand()
        if r < xoProb then
            if rand() < -log( diversity)/10 then
                p_1 = tournomentSelection(pop)
                p_2 = tournomentSelection(pop)
                off = crossover(p_1, p_2)
            else
                p_1 = tournomentSelection(schInstances)
                p_2 = tournomentSelection(schInstances)
                off = In-schemaCrossover (p_1, p_2)
            end
        else
            if r < xoProb + muProb then
                if rand() < -log( diversity)/10 then
                    p = tournomentSelection(pop)
                    off = mutation(p)
                else
                    p = tournomentSelection(schInstances)
                    off = apply direct or indirect In-schemaMutations randomly on p
                end
            else
                off = generateDirectOffspring(H)
            end
        end
        add off to offspring
    end
    pop = offspring + elites of pop
end
```

Fig. 4. Pseudo-code of SBGP algorithm.

As the figure shows, the offspring should have a maximum permissible size. Subtrees selected for recombination are checked for semantic equality, and it is guaranteed that recombination generates semantically different individuals from parents. Therefore, this operator increases the semantic diversity, leading to the exploration of new semantic space points.

4.2.2. Indirect local mutation operator (ILMO)

This operator gets a schema instance as the parent and applies the subtree mutation to generate another instance of the schema as offspring (Fig. 6). For doing so, a random node of the parent is selected and then replaced by a new randomly generated subtree. This procedure is repeated until the generated offspring is a schema instance too. If after *maxCount* attempts, no schema-instance offspring is generated, instead of a random subtree, a subtree that is biased towards building blocks is generated. The mutant generation is repeated until the final offspring becomes a schema sample. To generate a random tree, *generateRandomSubTreeWithBBInTerminals* function considers the building blocks as terminals. Thus it causes the tree generation process to include more building blocks, increasing the probability of schema membership. This operator tries to generate the schema instances as offspring without any bias, keeping the population's diversity high. In the case that this goal cannot be achieved, the operator becomes biased. The depth between the root and the selected point for the mutation (mp) is calculated by the depth (mp) function. By subtracting this value from the maximum depth, the maximum permissible depth (*offDepth*) is determined. When generating the random subtree, this depth is taken into consideration.

The following example demonstrates a schematic view of applying indirect local operators.

Example

Assuming function $x^4 + x^3 + x^2 + x$ as the target, consider a semantic schema H which is defined based on two semantic building block clusters along with an instantiation function that checks the active occurrence of these clusters in a tree (refer to Section 2.3 for detail definition of the semantic schema). Each cluster center is a semantic building block that is associated with a minimal tree known as representative tree. Let $c_1 = x^3$ and

```
ILRO
Inputs:
        p_1,p_2 : parents that are schema instances
        Semantic schema H
        maxDepth
Outputs:
        offspring
do
    do
        xop_1 = selectRandomNode(p_1)
        xop_2 = selectRandomNode(p_2)
    while (Tree(xop_1) semantically equals to Tree(xop_2))

    [off_1, off_2] = swap(xop_1, xop_2)
    off_1IsAcceptable = isSchemaInstance(off_1, H) & depth(off_1) < maxDepth
    off_2IsAcceptable = isSchemaInstance(off_2, H) & depth(off_2) < maxDepth
while (!off_1IsAcceptable & !off_2IsAcceptable )
if off_1IsAcceptable & off_2IsAcceptable then
    return offspring with greater membership in H
end
if off_1IsAcceptable then
    return off_1
end
return off_2
```

Fig. 5. Indirect local recombination pseudo-code.

```
ILMO
Inputs:
        p : parent that is schema instances
        H: Semantic schema
        BBSet = {b_1,…,b_n} (set of extracted building blocks)
        maxDepth, maxCount : threshold
Outputs:
        offspring
count = 0
do
    mp = selectRandomNode(p)
    offDepth = maxDepth - depth(mp)
    if count<maxCount then
        t = generateRandomSubtree(offDepth)
    else
        t = generateRandomSubtreeWithBBInTerminals(offDepth ,BBSet)
    end
    [off] = replace (mp, t)
    offIsAcceptable = isSchemaInstance(off, H) & depth(off) < maxDepth
    count++
while (!off1IsAcceptable)
return off
```

Fig. 6. Indirect local mutation pseudo-code.

$c_2 = x^2 + x$ be representative trees for the centers of clusters $C1$ and $C2$, respectively. Consider P_1 and P_2, illustrated in Fig. 7, as two instances of schema H in which building blocks of clusters 1 and 2 occurred, actively. It can be seen in the figure that P_1 includes both c_1 and c_2. P_2 includes $x^3 + 1$ and $2x^2 + 2x$ which are semantically equal to c_1 and c_2, respectively. According to the algorithm of ILRO provided in Fig. 5, crossover points are selected randomly until a pair of points are founded upon which applying crossover generates at least one offspring that is an instance of H. Consider for example the first selected points in P_1 and P_2 are XOP_1 and XOP_2. None of the resulting offsprings namely O_1 and O_2 are schema instances. Because crossover points are selected such that one building block of each parent is destroyed and generated trees do not contain any subtree in one of the required clusters of H.

Meanwhile if the crossover points XOP'_1 and XOP'_2 are selected, O_3 and O_4 are generated. O3 does not contain any subtree from cluster C_1 but O_4 consists of both x^3 and $x^2 + 2x$ subtrees, which are semantically equal to c_1 and c_2, respectively. Therefore, O_4 is an instance of H. The role of ILRO is finding randomly crossover points such as XOP'_1 and XOP'_2 that leads to at least one schema instance. For operator ILMO represented in Fig. 6, similar argument holds. If the selected subtree for mutation is replaced with a random subtree such that building blocks are destroyed, the offspring is not a schema instant anymore and other subtrees should be generated. If the alternate generation and replacement of the subtree do not result in the production of schema instance, the subtree generation is biased towards the building block clusters that do not exist in the context of the parent. For example, if in P_1, XOP'_1 is selected as mutation point it should be replaced with a subtree containing a building block of cluster C_1.

4.2.3. Direct local mutation operator (DLMO)

The direct local mutation is an operator that generates an offspring, which is an instance of the schema from any arbitrary parent. Fig. 8 gives the pseudo-code of this operator. Initially, this operator selects a node for mutation in the parent. Then, it

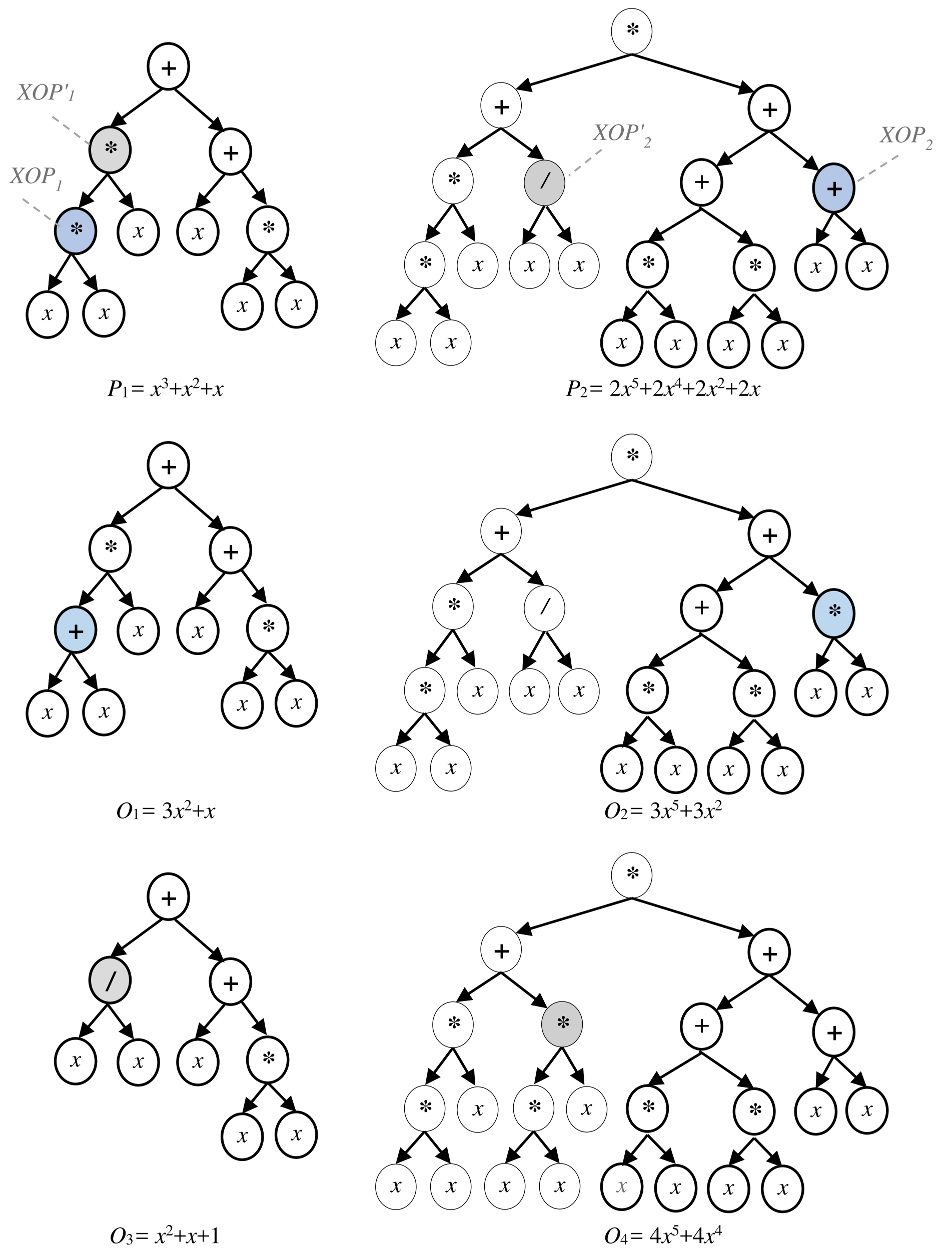

Fig. 7. Example of applying ILRO, P_1 and P_2: parents that are instances of schema H, O_1 and O_2: produced as a result of selecting unsuitable points, both are non-schema instances, O_1 and O_2: produced as a result of selecting suitable points, O_4 is a schema instance.

identifies the building blocks in the context of the tree. By "context", we mean part of the tree that remains after removing the selected subtree for mutation. At this time, the cluster centers, which have no member actively present in the parent, are added to *neccessaryBBs* set, and the random subtree that is biased towards this set is generated. When generating the random subtree, given the remaining depth of the tree, it is decided whether the building block could be chosen or not. At the depths in which the building block can be selected, a probability for building blocks, functions, and terminals is considered to keep the population diversity high.

4.2.4. Parentless child production operator (PCPO)

This operator directly generates a tree that is an instance of the specified schema, needless of any parent (Fig. 9). The operator instantiates the schema to generate offspring. To this aim, a random tree is generated from the set constructed from the union of terminals and the building blocks. Building blocks have the chance of being selected as terminals with the probability of ϵ. When a building block is inserted, first, a building block cluster is chosen randomly. Afterward, a block from this cluster is inserted in the tree randomly. Then, this cluster is removed from the set of remaining clusters. This algorithm helps generate some offspring that include building blocks from different clusters. The mentioned random tree is generated in a recursive function called GenerateRandomTreeWithRemainingClusters; however, given that the presence of all clusters in the tree is not necessary, the PCPO function checks the schema's membership to reconstruct trees that are not samples of the schema, a practice inspired by the advances of genetic engineering in producing parentless artificial creatures.

4.3. Generalization

To preserve the generalization of SBGP, we used a validation set to avoid overfitting in addition to training and testing sets. To

```
DLMO
Inputs:
        p : parent
        H: Semantic schema
        maxDepth, contributionThr : threshold
        $C = \{c_1,\dots,c_k\}$: set of cluster centroids
        $BBSet = \{b_1,\dots,b_n\}$ (set of extracted building blocks)
Outputs:
        offspring
do
    mp = selectRandomNode(p)
    includedBBs = getBuildingBlocks(p,mp)
    neccessaryBBs = { }
    for i = 1: k
        flag = false
        for each building block bb in includedBBs
            if l(bb) == i & SS(bb,p)>contributionThr then
                flag = true
                break
            end
        end
        if !flag then
            add $c_i$ to neccessaryBBs
        end
    end
    if isEmpty(neccessaryBBs) then
        offDepth = maxDepth - depth(mp)
        t = generateRandomSubTree(offDepth)
    else
        t = generateRandomSubTreeWithBBInTerminals(offDepth ,neccessaryBBs)
    end
    [off] = replace(xop, t)
    offIsAcceptable = isSchemaInstance(off, H) & depth(off) < maxDepth
while (!off1IsAcceptable)
return off
```

Fig. 8. Direct local mutation operator pseudo-code.

put it more precisely, the validation set has been utilized in two ways: (1) Stopping criteria In each generation of the algorithm, the error of the best individual of the population is calculated on the validation set. If this error is greater than the error in the previous generation, the algorithm is terminated because more training results in overfitting and subsequently increases the test error. (2) Elitism: In each generation, the best individual of the population upon the validation set enters the next generation without any change so that its spread in the population can transform the population towards better generalization. Apart from using the validation set, the offspring's bias towards the schema causes individuals' production to become constrained both in semantic and genotypic spaces. This, in turn, prevents unrestrained growth of the trees and overfitting to the training set.

5. Experimental results

To evaluate the proposed method, we compare SBGP's performance in terms of training error, test error, and trees' size with many versions of genetic programming, which fall under two categories apart from standard genetic programming. The first category includes the best versions of semantic genetic programming. The second one involves those algorithms that have somehow performed gradualism via data-driven layered learning and focused on the algorithm's generalization.

The main performance measure is the RMSE error of the best individual. Moreover, during the evolution, changes in the size and semantics of the best tree are studied. The number of dominant schema samples in each generation, and the rate of schema extraction are also some items to be studied. Furthermore, the locality of the proposed method is compared with standard genetic programming.

5.1. Benchmark functions

When selecting benchmark functions, we tried to select six well-known problems for symbolic regression that are used to evaluate most of the similar works. Four of them are real world applications and the remaining ones are synthetic complicated benchmarks. The complexity of benchmarks was also of a great concern in dataset suit preparation from various aspects such as the number of dimensions, the number of instances, and the fitness landscape. Table 1 shows the employed benchmark functions. It reveals the number of dimensions, the size of training, testing, and validation datasets, as well as the sampling method in the case of synthetic functions. Moreover, some important related works that used each benchmark are reported in the table.

The pollen dataset is a real world dataset from StatLib,[1] which includes observed geometric features of the pollen grain in 3 dimensions and its weight as the target. The Concrete compressive strength dataset is chosen from UCI Machine Learning Repository,[2] which predicts the quantitative value of compressive strength of a concrete based on its composition. Bioavailability dataset is also another real world dataset in pharmacokinetics area. The features of each record in this dataset are molecular descriptors that identify a drug, the output is the percentage of the drug that can effectively enter systemic blood circulation. The dataset is also known as %F in some references. Toxicity dataset is also a real world dataset in pharmacokinetics domain. Each feature in this dataset is a molecular descriptor for a specified drug, and its output is the drug's median lethal dose. That is the

[1] http://lib.stat.cmu.edu/datasets/pollen.data

[2] https://archive.ics.uci.edu/ml/datasets/Concrete+Compressive+Strength

```
PCPO
Inputs:
        H: Semantic schema
        maxDepth, contributionThr : threshold
        $C = \{c_1,\dots,c_k\}$: set of cluster centroids
        $L = \{l(b_i)| i = 1,2,\dots,n\}$ (set of cluster labels of BBSet)
        $BBSet = \{b_1,\dots,b_n\}$ (set of extracted building blocks)
        $\epsilon$: the probability of choosing building blocks
Outputs:
        offspring
do
    off = generateRandomTreeWithRemainingClusters(maxDepth, BBSet, L, C, $\epsilon$, 0)
    offIsAcceptable = isSchemaInstance(off, H) & depth(off) < maxDepth
while (!off1IsAcceptable)
return off
```

```
GenerateRandomTreeWithRemainingClusters
Inputs:
        maxDepth, contributionThr : threshold
        $C = \{c_1,\dots,c_k\}$: set of remaining cluster centroids
        $BBSet = \{b_1,\dots,b_n\}$ (set of extracted building blocks)
        $L = \{l(b_i)| i = 1,2,\dots,n\}$ (set of cluster labels of BBSet)
        $\epsilon$: the probability of choosing building blocks
        level: current level of the tree
Outputs:
        n : root of generated tree
if level = maxDepth-1 then
    n = randomLeafNode()
else
    if rand()>$\epsilon$ || |C|=0 then
        n = randomNode()
        if n is operator then
            n.left = generateRandomTreeWithRemainingClusters(maxDepth, BBSet, L, C, $\epsilon$, level+1)
            n.right = generateRandomTreeWithRemainingClusters(maxDepth, BBSet, L,C, $\epsilon$, level+1)
        end
    else
        c = selectRandomCluster(C)
        bb = selectRandomBBFromCluster(c, BBset, L)
        $C = C - \{c\}$
        n = rootOf (bb)
    end
end
return n
```

Fig. 9. Parentless child-production pseudo-code.

Table 1
Benchmark functions to evaluate SBGP.

Name	No. of dimension	Training data	Testing data	Validation data	Used in
Pollen	4	1927	1154	766	[68,70,73]
Concrete	8	517	309	204	[62,67,68,70,74–76]
Bioavailability	241	180	104	70	[66,68,70,74]
Toxicity	626	118	70	45	[65,66,68,70,75]
UBall5D	5	100 random points in [0, 6]	500 random points in [0, 6]	100 random points in [0, 6]	[30,68,70,75–77]
RatPol2D	2	50 random points in [0.05, 6.05]	1156 points [−0.25 : 0.2 : 6.35]	50 random points in [0.05, 6.05]	[30,68,70,78]

required amount of compound to kill half of the considered test organisms. This dataset is also known as LD50. All the mentioned datasets have been sampled with ratios of 50%, 20%, and 30% for training, validation, and testing datasets, respectively. The UBall5D and RaTPol2D benchmark functions are given in (6), and (7), in order.

$$f(x) = \frac{10}{5 + \sum_{i=1}^{5}(x_i - 3)^2} \tag{6}$$

$$f(x, y) = \frac{(x-3)^4 + (y-3)^3 - (y-3)}{(y-2)^4 + 10} \tag{7}$$

The four first benchmarks are corresponding to the real world problems from different domains. Concrete benchmark represents a highly non-linear function of concrete age and ingredients. Bioavailability and Toxicity are from the area of pharmacokinetics. In addition to the applicative importance of these benchmarks in the domain of drug discovery, the difficulty of the problems motivates their choice. In both datasets the number of dimensions is high but the proportion of the number of dimensions to the number of available instances is extremely high. This makes constructing and generalizing the related function very challenging. From these benchmarks, UBall5D and RatPol2D are synthesized functions. UBall5D is a function that requires extrapolation, not just interpolation [79]. In addition, it is stated in the literature (e.g. [79,80]) that this problem has much difficulties in discovering the harmonious input–output relationship. RatPol2D is a difficult problem from the generalization point of view, since in this benchmark the testing data are distributed in a wider range than training data.

5.2. Settings

Table 2 shows the configuration of genetic programming and the parameter settings of SBGP. The parameter "number of bins"

Table 2
Genetic programming and SBGP parameter settings.

Parameter	Value
Population size	200
Number of generations	100
Fitness measure	RMSE
Initial population generation	Ramped half-and-half
Probability of selecting the functions for the generation of the tree	0.66
Parent selection	Tournament selection (size 4)
Survival selection	All offspring
Recombination probability (local and standard)	0.7
Mutation probability	0.2
Parentless child production probability	0.1
Maximum initial depth	6
Maximum permissible depth	17
Function set	}+, -, *,/{
Number of elites	3
Diversity threshold for schema transition (diversityThr)	0.9
Number of bins	$\sqrt{training\ \ data}$
Length of the diversity sliding window (w)	3
Number of ERC	100 points within [0, 1]

utilized for estimating continues mutual information is set adaptively to the square root of the number of training samples as suggested in [81–83]. Some parameters (like the population size, fitness measure, parent selection method, trees' max depth, and the number of constant random numbers) have been set equal to the compared approaches to provide a fair evaluation framework. Other parameters, however, (like the probability of mutation and diversity threshold) have been set through trial and error to the best values.

The population model is a generational model, in which the produced offspring form the next generation directly. The three best trees of the previous generation replace the worst offspring as elites. Two of them are the best individuals based on training data, and one, the best individual in accordance with validation data. In all tests, the results are an average of 20 independent runs.

For each benchmark, the number of terminal variables is equal to the problem dimensions. Moreover, at the beginning of each run, 100 ephemeral random constants (ERC) are generated between 0 and 1 to be added to the terminal set. Due to providing fair comparison conditions, similar to compered methods, no constant is considered for Bioavailability and Toxicity problems; instead, the dataset features are utilized as terminals.

5.3. Accuracy

Since an individual's accuracy is inversely related to the error between its output and the target, for evaluating the accuracy of SBGP, we measure the individual's training error in terms of RMSE criterion. The lower error indicates the greater accuracy. Fig. 10 illustrates the SBGP convergence process in terms of the error over different benchmarks. For each benchmark, the average error of the population has been compared with the error of the best individual of the population during the evolution. As a result of using elitism, the best individual's error obeys a non-ascending trend in all benchmarks. It can be also inferred from the figure that diversity preserving mechanism of SBGP prevents the algorithm from premature convergence in all benchmarks. Furthermore, since validation set has been used to stop evolution, in some benchmarks (e.g. UBall5D in generation 50 and RatPol2D in generation 70), the evolution has been terminated sooner than 100 generations and prior to the convergence to prevent overfitting.

5.4. Evolution in semantic space

The semantic changes of the best individual along with the semantic average of the population are demonstrated in Fig. 11. As it can be inferred from the figure, by proceeding the evolution, the semantics of the population's individuals increase. The rising trend of the semantics values is the result of evolving the significant schema of the population towards the optimal solution along with individuals' evolution. Extracting a new schema may cause a jump in the population's semantics plot, which shows the explicit schema evolution leading to the explicit changes in generated offsprings in correspondence to the schema.

5.5. Controlling tree growth

Fig. 12 illustrates the average size of trees in the population during the evolution. Since SBGP uses local recombination, biasing the offsprings towards the schema, is less affected by the destructive effect of standard recombination. Moreover, the number of introns in the tree decreases because the presence of active building blocks in the trees is encouraged. Ultimately, this algorithm essentially controls bloat, as can be seen in most of the charts.

5.6. Semantic diversity

Fig. 13 illustrates the semantic diversity of the population during the evolution. In [38], it is shown that ICSS schema keeps semantic diversity high despite producing individuals with high semantic averages. Furthermore, the definition of semantic diversity has been introduced as the standard deviation of the population's semantic. Since considering the probability of local and standard operators based on the diversity helps preserve the population's diversity, these probabilities have been set accordingly based on Eqs. (2) and (3). In addition, considering the conditions for the transition of the significant schema based on diversity is another step towards increasing the population's diversity.

Results indicate that the diversity has been kept high until the end of the evolution. Based on the charts, simultaneous with schema's evolution, the diversity increases because semantic diversity is the criterion for schema transition. In fact, this diversity is calculated in each generation, and if it decreases compared to previous generations, the next schema is extracted. Producing a new schema generates trees with new building blocks leading to the exploration of new points in the semantic space. Therefore, the diversity increases, and this process continues until the evolution terminates. In RatPol2D and UBall5D benchmark functions, the diversity in the last generations is unchanged, and the population has converged, yet in more complicated benchmarks like Concrete and Toxicity, the diversity changes until the last generations and even does increase at the end of the evolution.

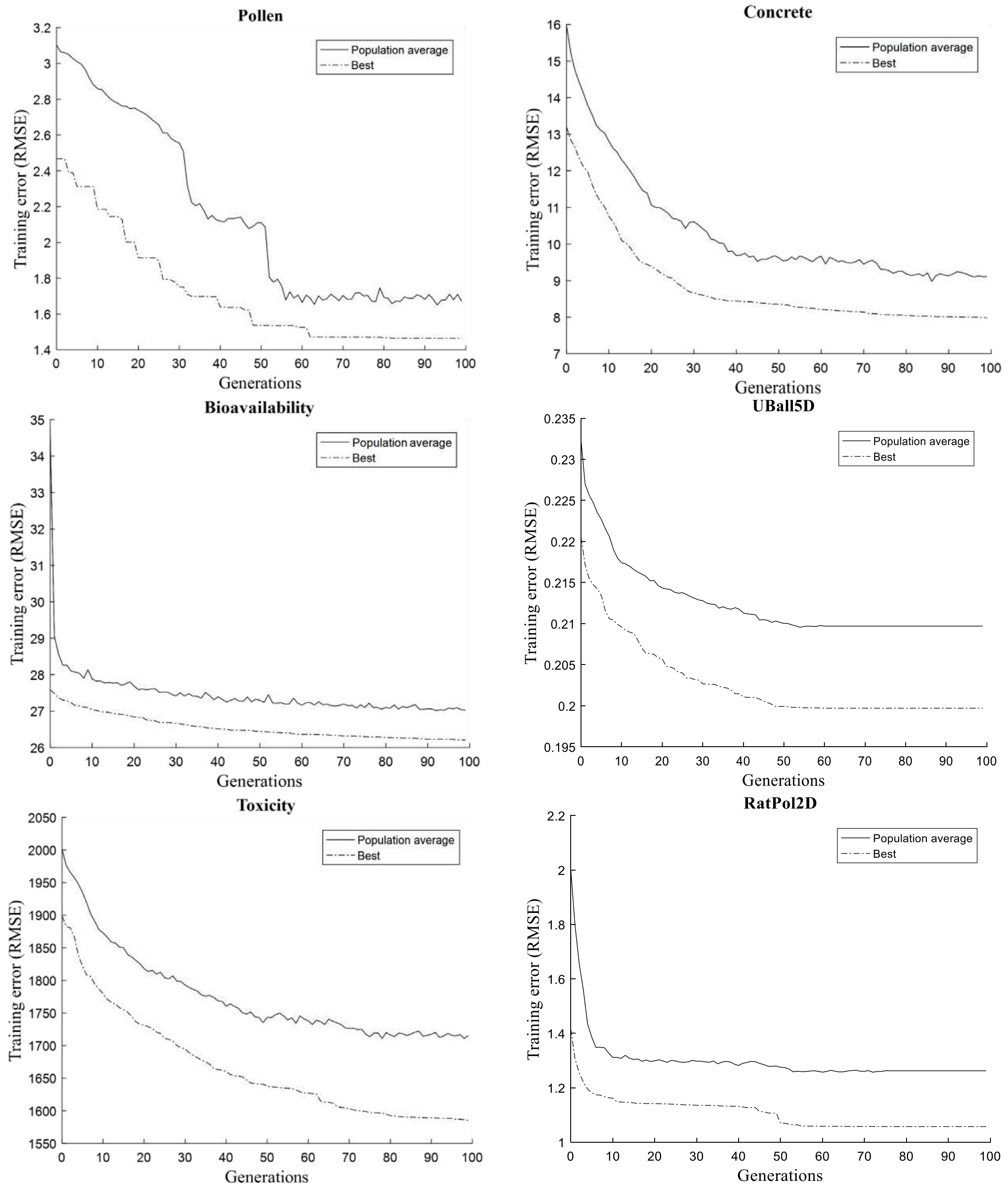

Fig. 10. SBGP convergence trends (comparison of the best individual and population average).

5.7. Number of schema samples

Fig. 14 demonstrates the number of samples of the significant schema of the population in different generations. Immediately after schema extraction, the number of schema instances in the population decreases. As the evolution proceeds, local operators generate and distribute schema samples within the population. As a result, the number of samples increases gradually. This process continues until the semantic diversity of the population decreases, and the conditions for transition of the schema are met. Here, the next schema is extracted, and once again, the number of samples of newly extracted significant schema decreases, and this process repeats until the end of the evolution.

5.8. Schema extraction rate

Diagrams of Fig. 15 illustrate the schema extraction rate in each generation. In fact, in different runs, the schema extraction occurs in different generations. The schema extraction rate in a generation refers to the fraction of the runs in which the schema has been extracted in that generation. This rate is proportionate to the changes in the number of schema samples and the semantic diversity studied in previous experiments.

5.9. Comparison with standard genetic programming

Fig. 16 compares the evolution convergence of the best individual in SBGP and standard genetic programming in the terms of

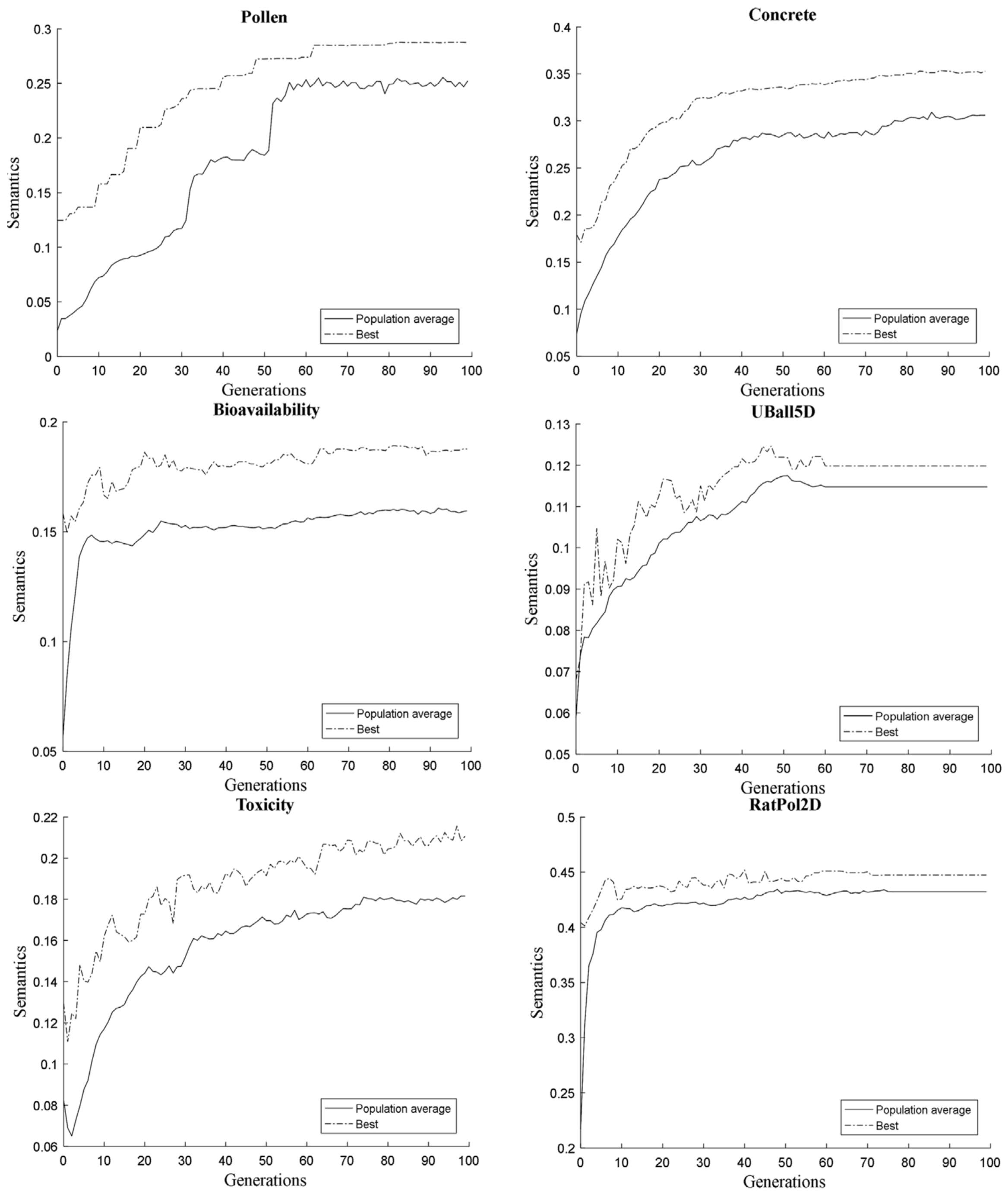

Fig. 11. Evolution process of the best individual and population average in semantic space.

training error measured by RMSE. As it is illustrated in the figure, the best individual of the evolution found in the last generation of SBGP is far better than standard GP, in all benchmarks. The improvement rates for Pollen, Concrete, Bioavailability, UBall5D, Toxicity and RatPol2D are 65%, 39%, 24%, 65%, 14% and 26%, respectively. It is also revealed in the figure that the convergence speed of SBGP is much more than standard GP. It seems that the gradual search strategy of SBGP along with local operators leads to improve the performance of the proposed algorithm compared to standard GP.

5.10. Locality

To evaluate the locality of SBGP, we defined the Semantic Distance (SD) according to (8), which measures the distance for each two consecutive generations. Fig. 17 gives the results of this evaluation.

$$SD = \sum_{g=1}^{popSize} \left| s(pop^{g}) - s(pop^{g-1}) \right| \tag{8}$$

The shorter the distance, the higher the locality of the algorithm. Diagrams in Fig. 17 depict this measure for SBGP,

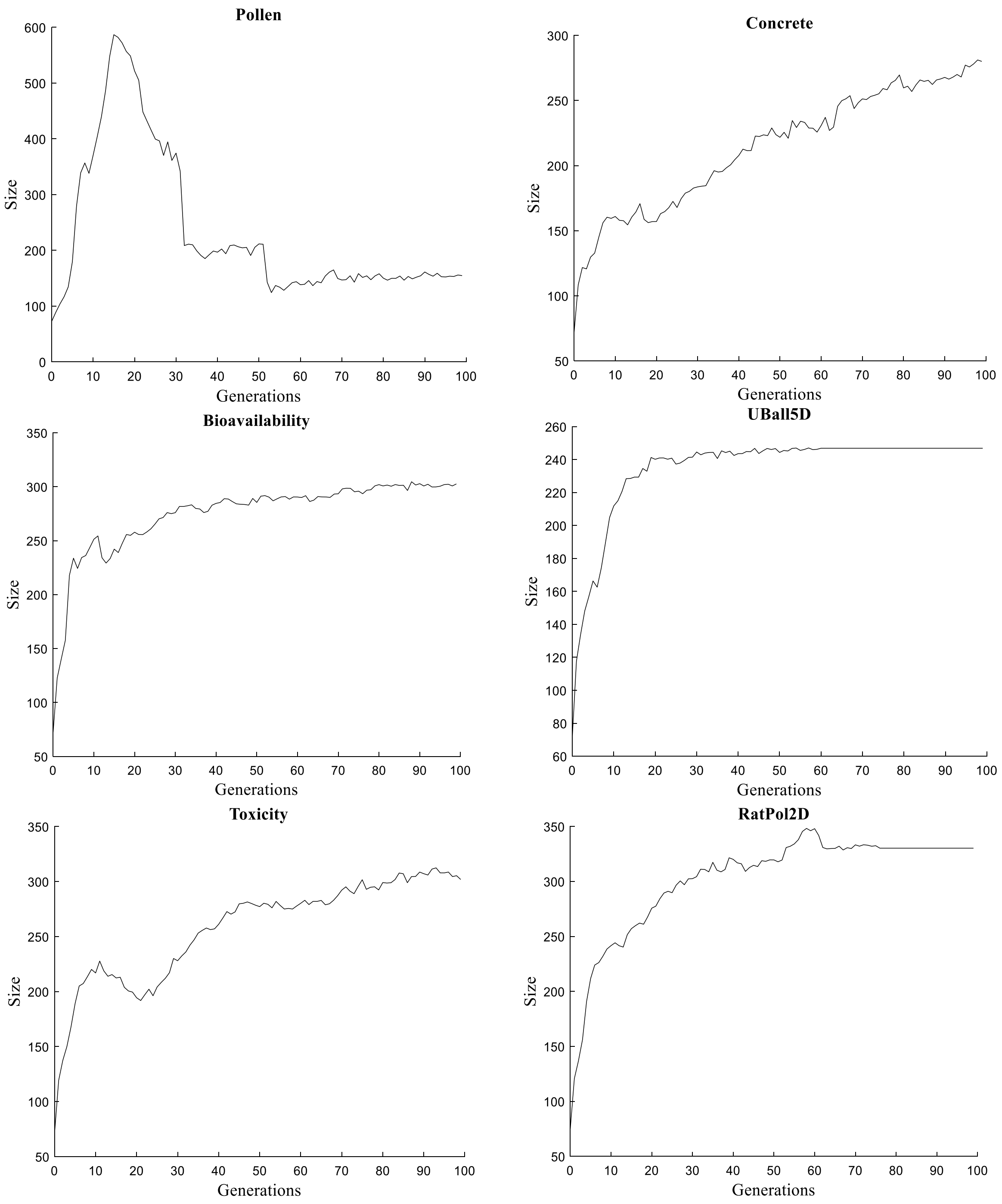

Fig. 12. Average population size during the evolution.

comparing it with standard genetic programming. As it can be inferred from the figure, the semantic locality of SBGP is more than standard GP in most generations.

5.11. Comparing SBGP performance with other methods

In this section, we compare the accuracy of SBGP with other versions of genetic programming in terms of RMSE for training and testing sets, respectively. Moreover, the average size of trees in the population of the last generation is reported in the tables. Genetic programming versions, which we compare with SBGP, include two general categories. The first consists of the best semantic genetic programming algorithms, and the second contains the algorithms which somehow perform gradualism via data-driven layered learning. Furthermore, standard genetic programming, abbreviated as GP, can be seen in the tables for comparison. The general settings of genetic programming have been set identical for all methods in these experiments. The probability of recombination and mutation for methods other than SBGP have been set to 0.9 and 0.1, respectively.

To evaluate whether the superiority of a method is statistically significant, a two-tailed Welch's t-test with a significance level was applied for training and test error between the proposed method and other methods in all comparisons. Welch's t-test is a nonparametric univariate statistical test, which is useful when the two samples have unequal variances. The last two columns of Tables 3–5 show the *p*-value of the two-tailed Welch's t-test. In most cases the null hypothesis is clearly rejected based on the

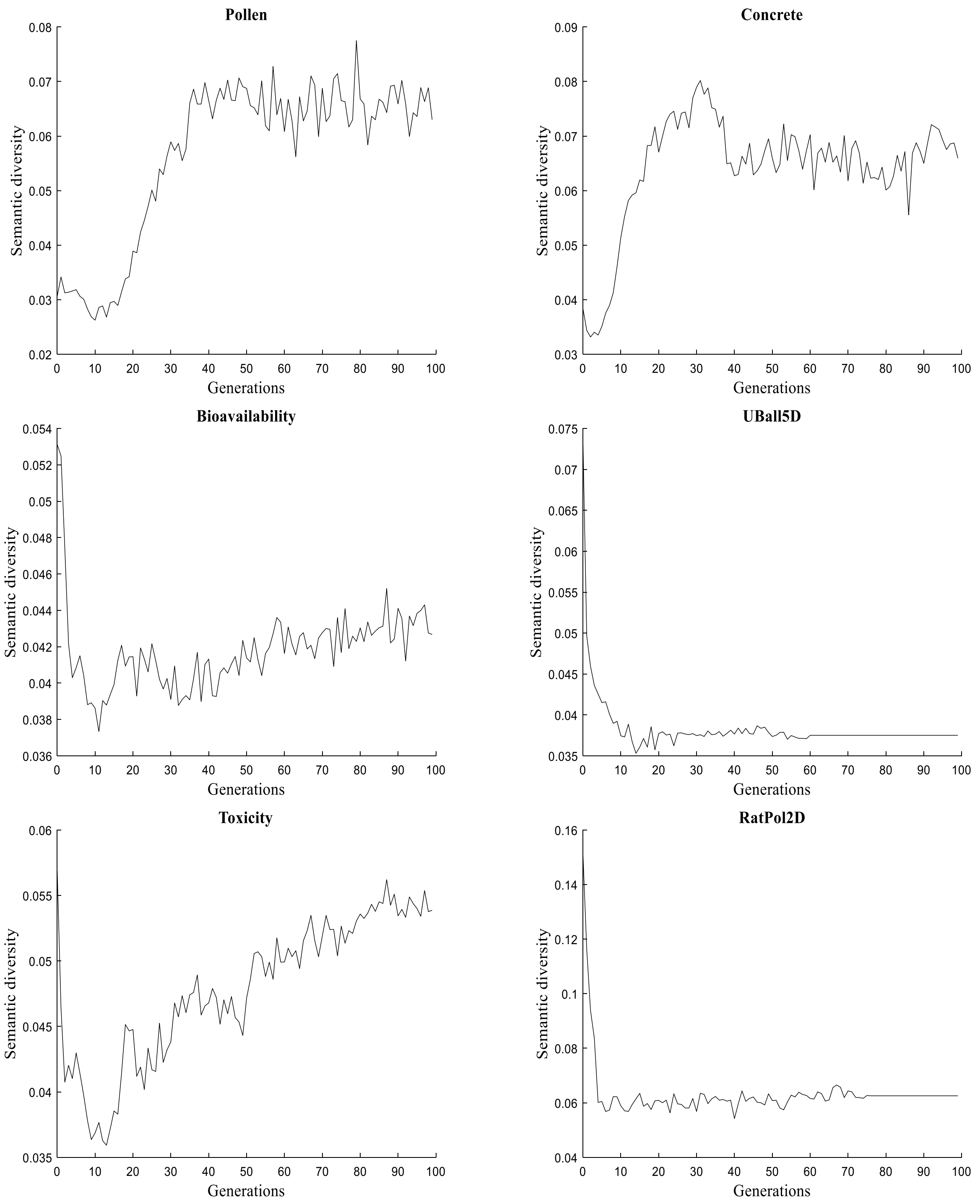

Fig. 13. Changes of semantic diversity in the evolution process.

tests with a 95% confidence level (p-value < 0.05), and the results are statistically significant.

5.12. Comparison with geometric semantic genetic programming methods

Semantic genetic programming methods have gained attention and extensive study in recent years. These approaches are deliberately established on localization and the use of geometric properties in the semantic space. From these methods, SGX [31] proposed a direct geometric semantic recombination operator for the first time, along with SSGX [62], its improved version introduced in 2016, which has attracted much attention. Moreover, AGX and RDO operators [55,56], the latest and most reliable semantic recombination operators, have been utilized for comparison. To implement these methods, we used SSGX paper source code available at github.[3] The library size for AGX and RDO is set to 1000, and the library's trees' depth is set to 4, based on the authors' recommendations. The probability of using standard recombination has been set to 0.3 for SSGX. The selected subtree's size for tree estimation is considered to be between 2 and 80, which complies with the authors' suggestion in [62].

3 https://github.com/jmmcd/GP-SSGX.

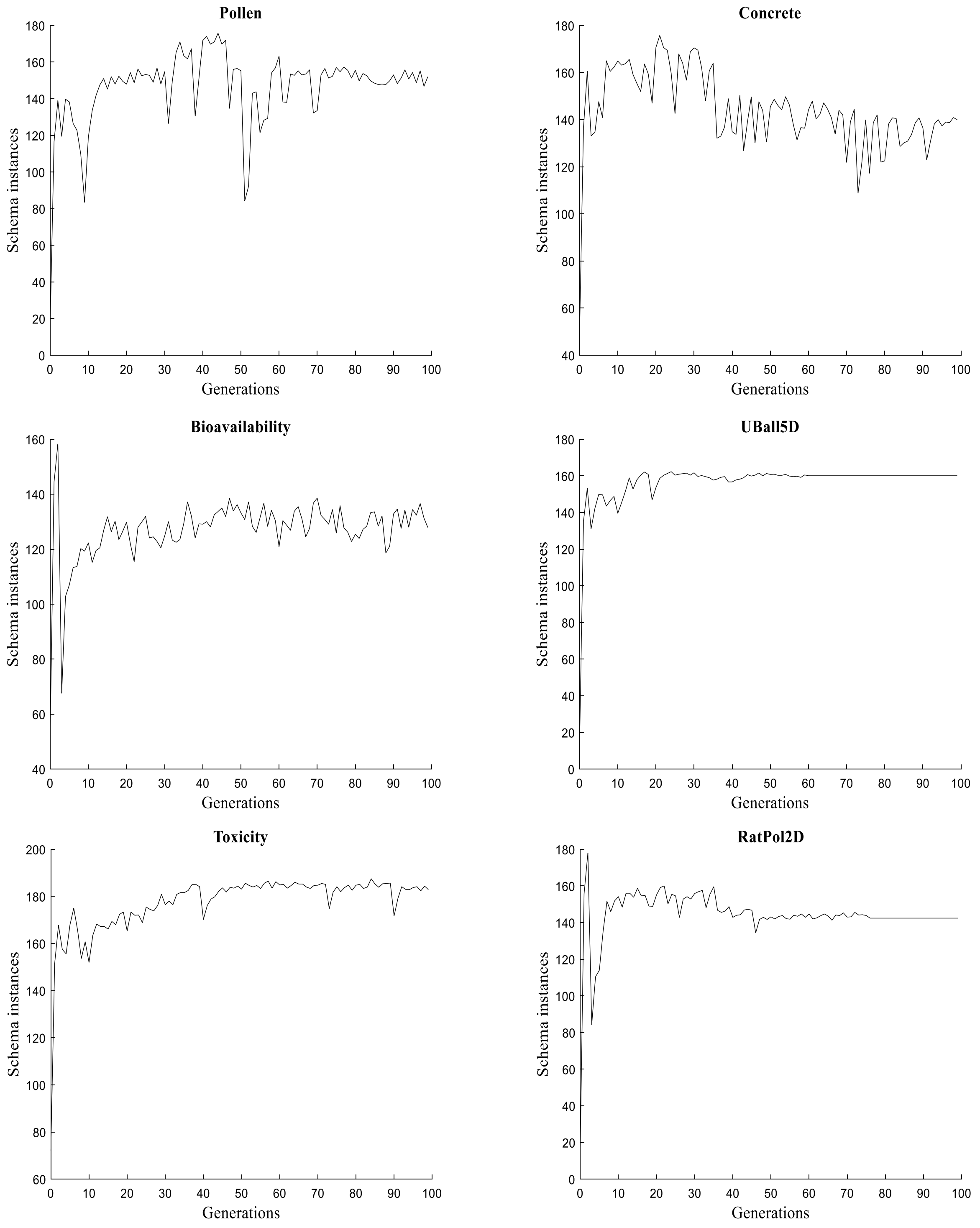

Fig. 14. The number of significant schema samples during the evolution.

Table 3 compares the accuracy of semantic genetic programming methods in terms of the best individual's error along with the average tree size for each algorithm and each problem. The error of the best found individual is measured upon training and testing datasets of each benchmarks, respectively and reported under training and test error columns. Regarding the repetition of running each algorithm, the average and standard deviation of obtained results in different runs are specified.

Form the training error viewpoint, SBGP outperforms other methods in all benchmarks except the RatPol2D. The average improvement rates for Pollen, Concrete, Bioavailability, Toxicity and UBall5D over other algorithms are 34%, 40%, 22%, 27% and 32%, respectively. In RatPol2D benchmark, SPGP has obtained the lowest training error after RDO. Since, gradual evolution gives the SBGP algorithm a chance for local search in semantic space and manages the evolution beyond the semantic schemas, it is capable of minimizing the training error better than most of semantic GPs. In terms of test error, again, SBGP has achieved the best performance in four out of six benchmarks. More precisely, the average improvement rates for Pollen, Bioavailability, Toxicity and RatPol2D over other algorithms are 33%, 15%, 7% and 70%, respectively. In both Concrete and UBall5D, SPGP is the second

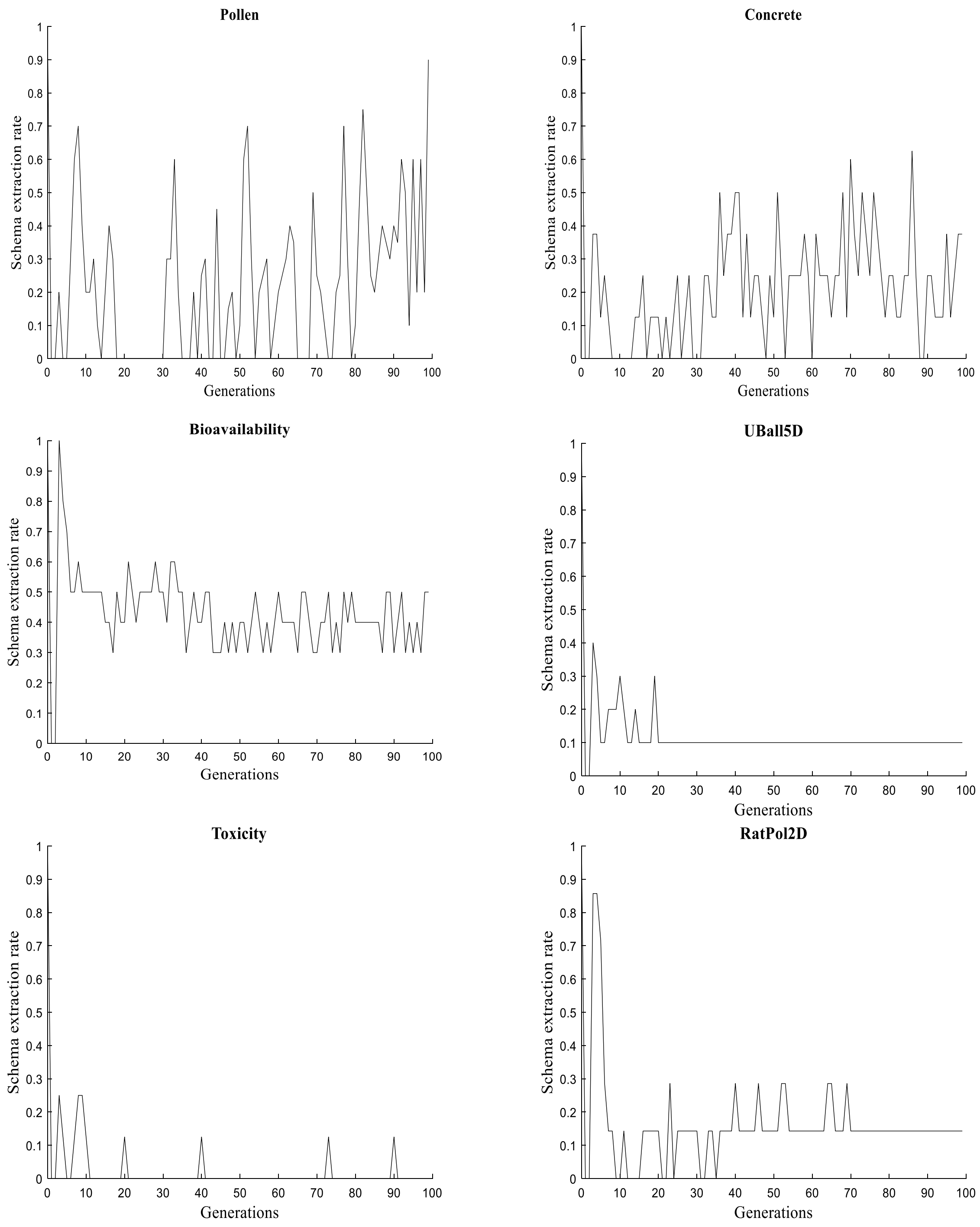

Fig. 15. Schema extraction rate in different generations.

best algorithm after RDO in the former and SSGX in the latter. Since in SBGP, the evolution is biased towards the samples of a specific schema, genetic programming cannot freely fit the model to training data. This constraint decreases the overfitting and increases generalization. For example, although RDO has the best performance in training error in RatPol2D dataset, it cannot generalize well. Because RatPol2D is a very challenging benchmark from the generalization aspect. However, in spite of higher training error, SPGP has superior performance in generalization of this benchmark. To sum up, it can be revealed from the table that SBGP has had less training error in 29 comparison cases out of 30 and less test error denoting the better generalization in 28 comparison cases out of 30.

The performance of the proposed method is comparable with other methods in terms of average tree size. SSGX has had the least tree size in most benchmarks since this method first estimates each tree with one of its subtrees and then applies recombination.

5.13. Comparison with data-driven layered learning genetic programming methods

Data-driven layered learning genetic programming includes a set of genetic programming methods that attempt to make the evolution gradual through data layering and provide high generalization. Here, we compare the proposed algorithm in terms of

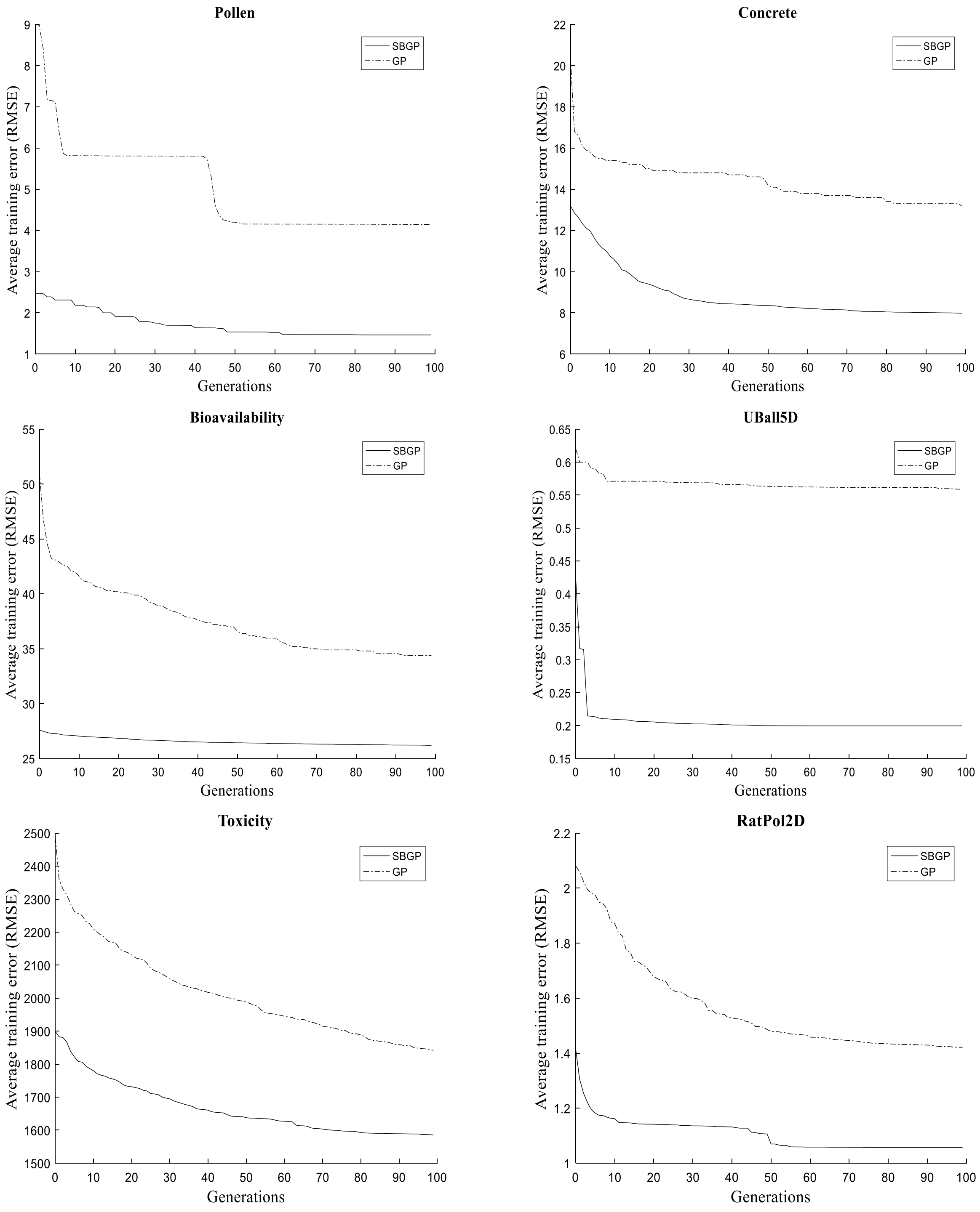

Fig. 16. Comparing SBGP convergence with standard genetic programming.

training and test error and trees' size with the mentioned algorithms described in Section 3.4. Among these methods, GPC [66], RST [65], and Stage-Based Method [67] have been selected as well-known methods in this area, and VBLL-GP [68] and SGP [69, 78], are chosen as statistical variants of GP, along with RCGP [70] that is presented recently.

Table 4 compares the accuracy of data-driven layered learning based GP methods in terms of the best individual's error along with the average tree size for each algorithm and each problem.

The error of the best individual of the evolution is measured over the training and testing datasets of each benchmarks and reported under corresponding columns. The average and standard deviation of obtained results in different runs are specified, due to execution repetition.

The results indicate that SBGP had less training error than other methods in 45 comparisons out of 48. Actually, it has outperformed other methods in Pollen, Concrete and Toxicity datasets and has gained the second place in UBall5D and Bioavailability datasets with a slight difference to RCGP. The average improvement rates for Pollen, Concrete and Toxicity over other algorithms are 59%, 23% and 10%, respectively. Since, SBGP is equipped with early termination of the evolution based on the validation error in order to prevent the individuals' function from overfitting to the training data, we do not expect it to have

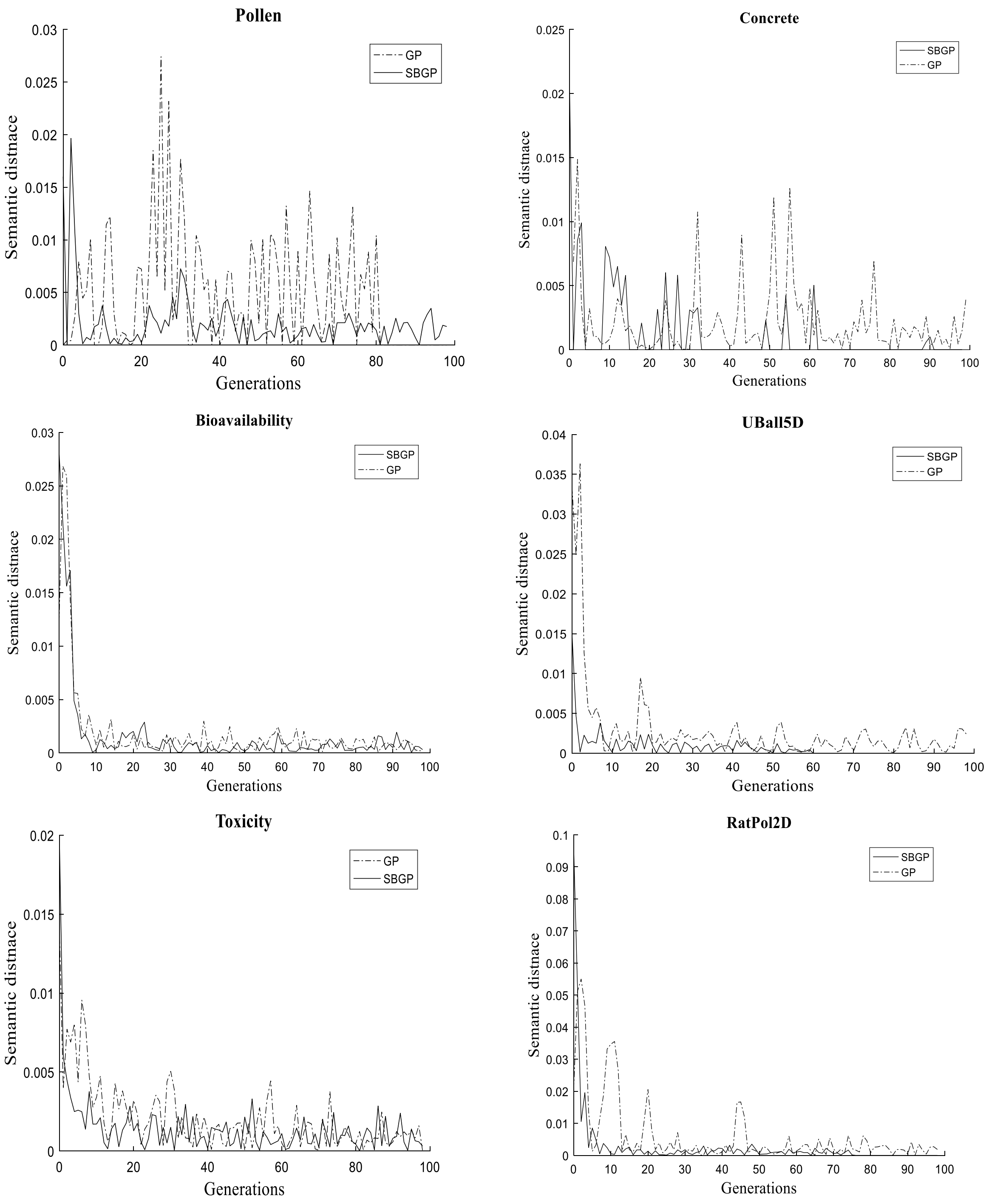

Fig. 17. Comparing the semantic space of SBGP and standard genetic programming.

the best performance from the training error perspective. Rather SBGP is designed to generalize well over the benchmarks. In the case of test error, SBGP has gained the best performance in four out of six benchmarks. More precisely, the average improvement rates for Pollen, Concrete, Bioavailability and UBall5D over other algorithms are 65%, 36%, 10% and 61%, respectively. In both Toxicity and RatPol2D benchmarks, SPGP is the second best algorithm after RCGP. Although the selected methods often utilize strategies for increasing generalization to improve their efficiency, SBGP has gained better generalization in 46 out of 48 records of the table. In addition to early termination of the evolution, biasing the population towards the instances of the significant schema, prevents SBGP from overfitting to the training dataset and increases generalization.

The average size of the trees in SBGP is also far less than other methods except RCGP. Since the smallest form of the building blocks in local operators is used to produce the offsprings, the trees' size is controlled to some extent. Additionally, using the validation set causes the algorithm to terminate before overfitting, resulting in simpler and shorter solutions. The main objective of RCGP is reducing the tree size; therefore, this measure is extremely low in this method. The average tree size in SBGP is less than all methods except RCGP in almost all benchmarks.

Table 3
Error comparison with semantic genetic programming.

Benchmark function	Algorithm	Training error	Test error	Tree Size Average	p value (Training error)	p value (Test error)
Pollen	SBGP	**1.46 ± 0.02**	**1.57 ± 0.24**	154.6 ± 88.7	–	–
	GP	4.14 ± 0.37	6.78 ± 0.93	5349.8 ± 407.2	3.74E−18	4.06E−17
	AGX	1.83 ± 0.47	1.85 ± 0.48	304.6 ± 94.6	2.29E−03	2.71E−02
	RDO	1.61 ± 0.40	1.62 ± 41.0	194.6 ± 67.5	**1.10E−01**	**9.96E−01**
	SGX	3.14 ± 0.09	3.18 ± 0.11	**65.25 ± 10.8**	1.28E−27	5.32E−21
	SSGX	1.97 ± 0.28	1.98 ± 0.28	100.45 ± 22.1	1.24E−07	1.53E−05
Concrete	SBGP	**7.98 ± 0.51**	8.89 ± 0.46	270.0 ± 58.1	–	–
	GP	13.13 ± 1.62	17.49 ± 1.89	5113.8 ± 411.3	2.23E−12	3.67E−15
	AGX	11.51 ± 1.12	12.48 ± 1.48	271.1 ± 31.4	6.84E−13	4.65E−10
	RDO	11.79 ± 16.40	**8.02 ± 1.64**	390.5 ± 47.7	**3.12E−01**	**3.24E−02**
	SGX	26.51 ± 3.43	27.38 ± 3.28	**53.2 ± 14.2**	4.29E−16	2.08E−16
	SSGX	11.81 ± 1.42	12.80 ± 1.73	114.9 ± 31.0	4.29E−11	2.14E−09
Bioavailability	SBGP	**26.20 ± 1.50**	**31.23 ± 1.26**	302.5 ± 160.6	–	–
	GP	34.85 ± 1.87	36.34 ± 2.80	4114.5 ± 361.1	3.87E−18	6.06E−08
	AGX	30.97 ± 3.64	33.70 ± 2.78	**21.5 ± 4.6**	1.22E−05	1.23E−03
	RDO	29.97 ± 0.87	33.97 ± 0.47	188.5 ± 16.6	7.43E−11	2.73E−09
	SGX	48.71 ± 2.12	46.57 ± 2.09	114.8 ± 56.8	7.42E−30	1.00E−23
	SSGX	30.41 ± 2.46	36.28 ± 13.34	90.3 ± 15.6	2.54E−07	**1.08E−01**
Toxicity	SBGP	**1585.4 ± 97.3**	**1902.10 ± 52.37**	301.89 ± 79.03	–	–
	GP	1859.3 ± 65.2	2125.5 ± 101.1	9081.1 ± 434.7	4.85E−12	1.36E−09
	AGX	2271.2 ± 111.1	1931.2 ± 28.13	207.6 ± 22.85	4.12E−22	3.67E−02
	RDO	2176.2 ± 119.67	1978.1 ± 198.4	310.8 ± 28.6	4.98E−19	**1.12E−01**
	SGX	2530.6 ± 125.01	2243.7 ± 417.37	340.1 ± 53.6	3.07E−25	1.71E−03
	SSGX	2141.03 ± 65.3	2017.36 ± 312.51	**109.44 ± 25.23**	6.93E−21	**1.19E−01**
UBall5D	SBGP	**0.19 ± 0.01**	0.21 ± 0.01	246.8 ± 100.01	–	–
	GP	0.551 ± 0.044	0.690 ± 0.088	181.435 ± 119.968	2.83E−20	5.07E−16
	AGX	0.23 ± 0.01	0.23 ± 0.07	**142.3 ± 144.6**	3.41E−15	**2.21E−01**
	RDO	0.22 ± 0.02	0.45 ± 0.093	167.4 ± 53.47	1.85E−06	4.19E−10
	SGX	0.47 ± 0.07	0.48 ± 0.07	640.10 ± 9.45	1.35E−13	2.66E−13
	SSGX	0.20 ± 0.01	**0.19 ± 0.01**	622.05 ± 38.91	3.07E−03	2.04E−07
RatPol2D	SBGP	1.05 ± 0.21	**2.18 ± 0.31**	189.430 ± 46.198	–	–
	GP	1.43 ± 0.42	5.81 ± 4.83	239.586 ± 228.848	1.16E−03	3.30E−03
	AGX	1.15 ± 0.62	6.05 ± 2.01	414.624 ± 171.515	**5.01E−01**	4.60E−08
	RDO	**0.46 ± 0.19**	17.01 ± 13.11	509.5 ± 135.79	2.58E−11	6.97E−05
	SGX	2.71 ± 1.07	5.36 ± 4.81	197.93 ± 51.29	1.13E−06	8.16E−03
	SSGX	1.15 ± 0.39	9.53 ± 5.29	**111.05 ± 25.36**	**3.21E−01**	5.67E−06

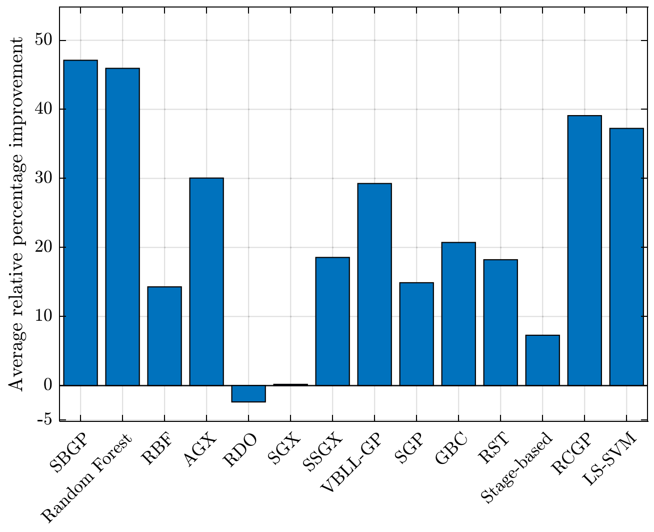

Fig. 18. Comparison of average relative percentage improvement between different methods.

5.14. Comparison with other function approximation methods

Here, we compare the training and test error of the proposed method with some of the most popular machine learning methods to determine SBGP position between non-evolutionary methods. These methods include the linear and binomial regression, called linear model and Quadratic Model in the tables, as well as, Least Squares Support Vector Machine (LS-SVM) [84], one of the best methods to approximate the function and RBF (Radial Basis Function) Neural Network, along with Random Forrest, a widely used and famous decision tree algorithm. For the last three methods, training, test, and validation sets have been used similar to SBGP. The validation set has been employed to find the main parameters of each method. Moreover, a random model is developed as a baseline which generates target from uniform random distribution in the range of training samples. The obtained results can be seen in Table 5, wherein the standard deviation of the proposed method has been reported, given the determinism of other methods except Random Forest and Random model, the standard deviation is zero in their different runs. Based on the results, it can be said that SBGP's performance and generalization are considerably better than linear regression and quadratic model. Moreover, this method's generalization in four datasets has been better than RBF and Random Forrest methods and in three cases out of six datasets, better than LS-SVM. In addition, it can be inferred from the table that the proposed method performs better than all or all but one method in five datasets in terms of generalization.

5.15. Comparison summary

To compare the overall performance of SBGP with other methods, the results of Standard GP were selected as a baseline, and relative improvement percentage of each method was computed. The obtained results for each method were averaged over six benchmark functions. Fig. 18 shows the average relative improvement percentage of 14 methods. As can be seen, the proposed method obtained the highest average (i.e., 47.11%), followed by Random Forest and RCGP that obtained 45.94% and 39.09% improvement, respectively.

Table 4
Comparison with data-driven layered learning genetic programming.

Benchmark function	Algorithm	Training error	Test error	Trees' Size Average	p value (Training error)	p value (Test error)
Pollen	SBGP	**1.46** ± **0.02**	**1.57** ± **0.24**	154.68 ± 88.7	–	–
	GP	4.14 ± 0.37	6.78 ± 0.93	5349.8 ± 407.2	3.74E−18	4.06E−17
	VBLL-GP	3.74 ± 0.25	3.76 ± 0.53	4021.0 ± 532.3	3.98E−20	1.15E−15
	SGP	3.01 ± 0.29	5.22 ± 0.64	1107.6 ± 165.5	1.01E−15	2.35E−18
	GBC	3.98 ± 0.20	4.42 ± 0.46	3649.2 ± 366.1	6.49E−23	8.71E−21
	RST	3.55 ± 0.44	4.30 ± 0.87	4190.3 ± 426.9	9.89E−15	4.20E−12
	Stage-based	3.57 ± 0.16	6.54 ± 0.72	4651.0 ± 372.8	1.81E−23	8.24E−20
	RCGP	3.16 ± 0.34	3.18 ± 0.31	**104.5** ± **31.4**	3.66E−15	8.62E−20
Concrete	SBGP	**7.98** ± **0.51**	**8.89** ± **0.46**	280.01 ± 58.1	–	–
	GP	13.13 ± 1.62	17.49 ± 1.89	5113.8 ± 411.3	2.23E−12	3.67E−15
	VBLL-GP	9.09 ± 1.21	11.39 ± 1.35	6043.1 ± 468.8	8.45E−04	5.46E−08
	SGP	8.11 ± 1.48	15.73 ± 1.54	1563.2 ± 220.9	**7.14E−01**	2.63E−15
	GBC	12.81 ± 1.38	13.65 ± 1.51	4331.8 ± 366.5	1.62E−13	2.92E−12
	RST	12.30 ± 1.57	15.13 ± 2.04	5763.4 ± 548.8	3.71E−11	1.06E−11
	Stage-based	11.67 ± 1.49	15.10 ± 0.83	5300.6 ± 372.1	2.61E−10	1.94E−23
	RCGP	8.55 ± 0.94	11.61 ± 1.31	**51.7** ± **17.4**	2.38E−02	6.98E−09
Bioavailability	SBGP	26.20 ± 1.50	**31.23** ± **1.26**	302.58 ± 160.6	–	–
	GP	34.85 ± 1.87	36.34 ± 2.80	4114.5 ± 361.1	3.87E−18	6.06E−08
	VBLL-GP	31.48 ± 1.05	31.79 ± 1.16	3076.6 ± 282.2	1.19E−14	**1.52E−01**
	SGP	27.56 ± 1.26	34.97 ± 1.21	875.3 ± 103.6	3.65E−03	1.15E−09
	GBC	33.75 ± 1.92	35.47 ± 2.01	2036.5 ± 281.4	5.59E−16	4.06E−09
	RST	30.30 ± 2.57	34.87 ± 2.30	3842.1 ± 340.6	8.18E−07	8.45E−07
	Stage-based	30.83 ± 2.98	36.22 ± 2.93	3522.0 ± 265.1	1.04E−06	2.07E−07
	RCGP	**25.10** ± **0.27**	32.91 ± 1.13	**47.2** ± **11.38**	4.18E−03	7.66E−05
Toxicity	SBGP	**1585.4** ± **97.3**	1902.10 ± 52.37	301.89 ± 79.03	–	–
	GP	1859.3 ± 65.2	2125.5 ± 101.1	9081.1 ± 434.7	4.85E−12	1.36E−09
	VBLL-GP	1839.9 ± 45.3	1954.0 ± 73.2	5033.2 ± 562.1	4.21E−11	1.44E−02
	SGP	1703.2 ± 88.90	2053.4 ± 67.6	1026.3 ± 105.2	2.88E−04	2.27E−09
	GBC	1863.2 ± 71.1	2065.1 ± 66.4	5642.1 ± 382.3	4.07E−12	2.78E−10
	RST	1817.7 ± 151.3	2092.1 ± 185.7	7680.3 ± 547.2	1.99E−06	2.25E−04
	Stage-based	1714.5 ± 80.3	2198.3 ± 216.0	7996.2 ± 321.6	5.26E−05	6.19E−06
	RCGP	1617.1 ± 50.8	**1788.5** ± **293.3**	**48.3** ± **10.4**	**2.07E−01**	**1.03E−01**
UBall5D	SBGP	0.19 ± 0.01	**0.21** ± **0.01**	264.8 ± 100.01	–	–
	GP	0.551 ± 0.044	0.690 ± 0.088	1651.2 ± 239.9	2.83E−20	5.07E−16
	VBLL-GP	0.495 ± 0.021	0.589 ± 0.057	806.7 ± 176.0	3.58E−30	5.28E−18
	SGP	0.372 ± 0.045	0.647 ± 0.051	263.0 ± 54.9	5.03E−14	2.30E−20
	GBC	0.503 ± 0.027	0.627 ± 0.051	1002.8 ± 184.5	1.39E−25	5.92E−20
	RST	0.491 ± 0.061	0.631 ± 0.076	1456.8 ± 357.3	2.07E−15	3.19E−16
	Stage-based	0.520 ± 0.073	0.656 ± 0.093	1574.1 ± 254.9	1.43E−14	5.98E−15
	RCGP	**0.16** ± **0.03**	0.29 ± 0.20	**44.2** ± **9.8**	3.03E−04	**8.98E−02**
RatPol2D	SBGP	1.05 ± 0.21	2.18 ± 0.31	189.430 ± 46.198	–	–
	GP	1.43 ± 0.42	5.81 ± 4.83	239.586 ± 228.848	1.16E−03	3.30E−03
	VBLL-GP	1.27 ± 0.14	2.27 ± 0.52	2138.1 ± 197.6	4.47E−04	**5.11E−01**
	SGP	0.42 ± 0.08	3.32 ± 0.56	548.9 ± 88.0	3.96E−12	7.47E−09
	GBC	1.37 ± 0.39	2.72 ± 0.70	1765.5 ± 201.4	3.05E−03	4.01E−03
	RST	1.24 ± 0.27	3.19 ± 0.66	2682.7 ± 249.8	1.78E−02	1.27E−06
	Stage-based	1.10 ± 1.52	4.38 ± 1.03	2891.9 ± 286.0	**8.86E−01**	5.04E−09
	RCGP	**0.68** ± **0.13**	**2.06** ± **0.84**	**47.8** ± **15.1**	1.53E−07	**5.55E−01**

5.16. Computational cost comparison

To further evaluate the performance of SBGP, some important GP based methods' computational costs were compared. Table 6 presents the average running time and standard deviation of each method in each benchmark function. As can be seen, the running time of SBGP is close to AGX and RDO results. The time efficiency of GP is better and RCGP is far better than semantic methods because it uses some mechanisms for controlling the size of trees. To sum up, the computational cost of SBGP is comparable with semantic GP methods. Due to semantic calculation and semantic space manipulation (alone or along with syntactic space) these have more computational cost in comparison with traditional GP methods, however in most of symbolic regression applications, error minimization is more important than computational cost.

6. Conclusion and future work

This paper introduced the Schema-Based Genetic Programming (SBGP) algorithm. It makes the searching process of genetic programming gradual, somehow combining local and global searches to improve the algorithm's performance. Utilizing local operators, which bias the offsprings towards the schema, improves genetic programming search power. After introducing local operators and the SBGP algorithm in this paper, we analyzed this method's performance over six dataset including real world and synthesized benchmarks. Comparing SBGB with the most popular semantic genetic programming versions on the one hand and data-driven layered methods on the other hand. Results show that in terms of generalization measured by RMSE, the proposed method outperforms both genetic programming sets of algorithm in four out of six benchmarks and obtain near to best results in others. The generalization improvement of SBGP in semantic genetic programming comparisons is up to 87% and in layered learning comparison experiments up to 76%.

As the future work, we intend to expand the notion of semantics and the algorithm SBGP for classification domain. It is also beneficial to modify the algorithm in order to evolve trees with lower sizes and replace trees with their semantically equal subtrees as performed in some of the previous works.

Table 5
Comparison with other function approximation methods.

Benchmark function	Algorithm	Training error	Test error	p value (Training error)	p value (Test error)
Pollen	SBGP	**1.46** ± **0.02**	**1.57** ± **0.24**	–	–
	Linear model	8.26	8.29	8.85E−50	3.50E−29
	Quadratic model	8.23	8.26	9.63E−50	3.81E−29
	LS-SVM	1.78	3.79	1.42E−24	4.42E−20
	RBF	2.82	8.01	1.68E−36	7.86E−29
	Random forest	2.08 ± 0.24	3.18 ± 0.04	4.41E−10	5.07E−18
	Random model	21.75 ± 0.55	22.46 ± 0.74	1.61E−31	1.25E−33
Concrete	SBGP	7.98 ± 0.51	8.89 ± 0.46	–	–
	Linear model	99.76	102.75	1.57E−44	1.45E−45
	Quadratic model	48.57	49.19	8.48E−38	1.37E−38
	LS-SVM	**1.35**	13.15	7.22E−23	4.32E−20
	RBF	12.80	19.74	2.95E−20	9.07E−28
	Random forest	3.22 ± 0.50	**6.26** ± **0.84**	5.92E−28	4.09E−13
	Random model	29 ± 0.53	30 ± 0.46	1.32E−51	9.22E−54
Bioavailability	SBGP	26.20 ± 1.50	31.23 ± 1.26	–	–
	Linear model	203.65	1463.88	4.55E−41	9.67E−60
	Quadratic model	147.81	736.12	5.97E−38	6.89E−54
	LS-SVM	**15.65**	**29.85**	7.49E−18	9.99E−05
	RBF	18.58	32.04	3.10E−15	9.70E−03
	Random forest	19.67 ± 0.97	31.79 ± 1.04	3.01E−17	**1.34E−01**
	Random model	46.6 ± 1.92	42.2 ± 1.06	2.29E−30	2.01E−27
Toxicity	SBGP	1585.4 ± 97.3	1902.10 ± 52.37	–	–
	Linear model	332691.43	332718.92	8.71E−69	6.85E−74
	Quadratic model	52577.51	52214.66	2.38E−53	2.38E−58
	LS-SVM	1014.90	**1893.76**	2.20E−16	**4.85E−01**
	RBF	**278.32**	2275.27	3.88E−23	5.87E−18
	Random forest	1250.33 ± 138	2058.63 ± 19.02	2.18E−10	5.02E−12
	Random model	2378 ± 12	2462 ± 14	2.14E−19	3.64E−23
UBall5D	SBGP	0.19 ± 0.01	**0.21** ± **0.01**	–	–
	Linear model	0.739	0.836	9.82E−35	8.12E−36
	Quadratic model	0.525	0.531	1.16E−30	2.62E−30
	LS-SVM	0.190	0.472	**1.00E+00**	1.24E−28
	RBF	0.182	0.541	2.00E−03	1.46E−30
	Random forest	**0.141** ± **0.02**	0.272 ± 0.03	1.53E−10	8.07E−09
	Random model	0.34 ± 0.01	0.35 ± 0.007	1.99E−35	8.38E−34
RatPlo2D	SBGP	1.05 ± 0.21	2.18 ± 0.31	–	–
	Linear model	3.67	4.58	1.57E−22	1.25E−18
	Quadratic model	3.32	3.42	2.35E−21	2.40E−13
	LS-SVM	0.073	**0.34**	1.55E−14	1.76E−16
	RBF	**0.024**	0.56	6.34E−15	1.84E−15
	Random forest	0.33 ± 0.06	1.04 ± 0.07	6.53E−13	3.10E−13
	Random model	2.48 ± 0.14	3.16 ± 0.18	2.98E−23	2.73E−13

Table 6
Error comparison with semantic genetic programming.

Benchmark function	Running Time (s)				
	SBGP	GP	AGX	RDO	RCGP
Pollen	923 ± 98.2	271.2 ± 71.2	845.3 ± 165.4	977.4 ± 151.7	168.6 ± 49.3
Concrete	1127.5 ± 390.5	539.0 ± 53.9	1147.0 ± 103.2	1401.2 ± 125.5	281.43 ± 37.9
Bioavailability	717.4 ± 54.0	260.1 ± 65.8	645.0 ± 75.1	688.9 ± 69.0	171.9 ± 46.1
Toxicity	1593 ± 118.5	521.4 ± 96.2	1105.4 ± 117.0	1301.3 ± 123.5	351.6 ± 56.5
UBall5D	229 ± 20.3	151.5 ± 30.2	621.3 ± 76.3	521.1 ± 80.9	134.9 ± 29.2
RatPol2D	779 ± 54.1	351.5 ± 51.9	822.9 ± 81.2	893.0 ± 77.9	271.1 ± 39.8

CRediT authorship contribution statement

Zahra Zojaji: Conceptualization, Methodology, Software, Data curation. **Mohammad Mehdi Ebadzadeh:** Supervision, Conceptualization. **Hamid Nasiri:** Writing – reviewing and editing, Visualization, Validation.

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] M. Gandomi, A.R. Kashani, A. Farhadi, M. Akhani, A.H. Gandomi, Spectral acceleration prediction using genetic programming based approaches, Appl. Soft Comput. 106 (2021) 107326.

[2] A. Fathi, R. Sadeghi, A genetic programming method for feature mapping to improve prediction of HIV-1 protease cleavage site, Appl. Soft Comput. 72 (2018) 56–64.

[3] Y. Bi, B. Xue, M. Zhang, Multi-objective genetic programming for feature learning in face recognition, Appl. Soft Comput. 103 (2021) 107152.

[4] A.R. Azarhoosh, Z. Zojaji, F. Moghadas Nejad, Nonlinear genetic-base models for prediction of fatigue life of modified asphalt mixtures by precipitated calcium carbonate, Road Mater. Pavement Des. 21 (2020) 850–866.

[5] D. Quammen, Was Darwin Wrong? National Geographic Society, 2004.
[6] C. Darwin, On the Origin of Species by Means of Natural Selection, or, The Preservation of Favoured Races in the Struggle for Life, J. Murray, 1859.
[7] R.M. Downing, Neutrality and gradualism: Encouraging exploration and exploitation simultaneously with binary decision diagrams, in: 2006 IEEE Int. Conf. Evol. Comput., 2006, pp. 615–622.
[8] E. Galvan-Lopez, M. O'Neill, On the effects of locality in a permutation problem: The sudoku puzzle, in: 2009 IEEE Symp. Comput. Intell. Games, 2009, pp. 80–87.
[9] E. Galván-López, J. McDermott, M. O'Neill, A. Brabazon, Towards an understanding of locality in genetic programming, in: Proc. 12th Annu. Conf. Genet. Evol. Comput., 2010, pp. 901–908.
[10] J. Gottlieb, G.R. Raidl, The effects of locality on the dynamics of decoder-based evolutionary search, in: Proc. 2nd Annu. Conf. Genet. Evol. Comput., 2000, pp. 283–290.
[11] F. Rothlauf, M. Oetzel, On the locality of grammatical evolution, in: Eur. Conf. Genet. Program., 2006, pp. 320–330.
[12] F. Rothlauf, D.E. Goldberg, Redundant representations in evolutionary computation, Evol. Comput. 11 (2003) 381–415.
[13] E. Galván-López, J. McDermott, M. O'Neill, A. Brabazon, Defining locality as a problem difficulty measure in genetic programming, Genet. Program. Evol. Mach. 12 (2011) 365–401.
[14] T. Seaton, J.F. Miller, T. Clarke, An ecological approach to measuring locality in linear genotype to phenotype maps, in: Eur. Conf. Genet. Program., 2012, pp. 170–181.
[15] J.P. Rosca, D.H. Ballard, Causality in Genetic Programming, in: Proc. 6th Int. Conf. Genet. Algorithms, 1995, pp. 256–263.
[16] W.-C. Lee, Genetic programming decision tree for bankruptcy prediction, in: 9th Jt. Int. Conf. Inf. Sci., 2006, pp. 33–36.
[17] J. Rosca, An analysis of hierarchical genetic programming, 1995.
[18] M. Črepinšek, S.-H. Liu, M. Mernik, Exploration and exploitation in evolutionary algorithms: A survey, ACM Comput. Surv. 45 (2013) 1–33.
[19] C. Soza, R.L. Becerra, M.C. Riff, C.A.C. Coello, Solving timetabling problems using a cultural algorithm, Appl. Soft Comput. 11 (2011) 337–344.
[20] W. Fu, B. Xue, X. Gao, M. Zhang, Transductive transfer learning based genetic programming for balanced and unbalanced document classification using different types of features, Appl. Soft Comput. 103 (2021) 107172.
[21] J. Ma, G. Teng, A hybrid multiple feature construction approach for classification using genetic programming, Appl. Soft Comput. 80 (2019) 687–699.
[22] G. Campobello, D. Dell'Aquila, M. Russo, A. Segreto, Neuro-genetic programming for multigenre classification of music content, Appl. Soft Comput. 94 (2020) 106488.
[23] J.H. Holland, et al., Adaptation in Natural and Artificial Systems: An Introductory Analysis with Applications to Biology, Control, and Artificial Intelligence, MIT Press, 1992.
[24] J.R. Koza, J.R. Koza, Genetic Programming: On the Programming of Computers by Means of Natural Selection, MIT Press, 1992.
[25] D.E. Golberg, Genetic Algorithms in Search, Optimization, and Machine Learning, Addion Wesley, 1989, p. 36.
[26] Z. Zojaji, M.M. Ebadzadeh, Semantic schema theory for genetic programming, Appl. Intell. 44 (2016) 67–87.
[27] K. Krawiec, T. Pawlak, Locally geometric semantic crossover: A study on the roles of semantics and homology in recombination operators, Genet. Program. Evol. Mach. 14 (2013) 31–63.
[28] L. Vanneschi, M. Castelli, S. Silva, A survey of semantic methods in genetic programming, Genet. Program. Evol. Mach. 15 (2014) 195–214.
[29] N.F. McPhee, B. Ohs, T. Hutchison, Semantic building blocks in genetic programming, in: Eur. Conf. Genet. Program., 2008, pp. 134–145.
[30] N.Q. Uy, N.X. Hoai, M. O'Neill, R.I. McKay, E. Galván-López, Semantically-based crossover in genetic programming: Application to real-valued symbolic regression, Genet. Program. Evol. Mach. 12 (2011) 91–119.
[31] A. Moraglio, K. Krawiec, C.G. Johnson, Geometric semantic genetic programming, in: Int. Conf. Parallel Probl. Solving from Nat., 2012, pp. 21–31.
[32] L. Beadle, C.G. Johnson, Semantically driven crossover in genetic programming, in: 2008 IEEE Congr. Evol. Comput. (IEEE World Congr. Comput. Intell.), 2008, pp. 111–116.
[33] K. Krawiec, P. Lichocki, Approximating geometric crossover in semantic space, in: Proc. 11th Annu. Conf. Genet. Evol. Comput., 2009, pp. 987–994.
[34] K. Krawiec, The framework of behavioral program synthesis, in: Behav. Progr. Synth. with Genet. Program, Springer, 2016, pp. 35–41.
[35] K. Krawiec, U.-M. O'Reilly, Behavioral programming: A broader and more detailed take on semantic GP, in: Proc. 2014 Annu. Conf. Genet. Evol. Comput., 2014, pp. 935–942.
[36] K. Krawiec, J. Swan, Pattern-guided genetic programming, in: Proc. 15th Annu. Conf. Genet. Evol. Comput., 2013, pp. 949–956.
[37] Z. Zojaji, M.M. Ebadzadeh, An improved semantic schema modeling for genetic programming, Soft Comput. 22 (2018) 3237–3260.
[38] Z. Zojaji, M.M. Ebadzadeh, Semantic schema modeling for genetic programming using clustering of building blocks, Appl. Intell. 48 (2018) 1442–1460.
[39] D. Jackson, Phenotypic diversity in initial genetic programming populations, in: Eur. Conf. Genet. Program., 2010, pp. 98–109.
[40] D. Jackson, Promoting phenotypic diversity in genetic programming, in: Int. Conf. Parallel Probl. Solving from Nat., 2010, pp. 472–481.
[41] Q. Chen, B. Xue, M. Zhang, Preserving population diversity based on transformed semantics in genetic programming for symbolic regression, IEEE Trans. Evol. Comput. (2020).
[42] L. Beadle, C.G. Johnson, Semantic analysis of program initialisation in genetic programming, Genet. Program. Evol. Mach. 10 (2009) 307–337.
[43] T.P. Pawlak, K. Krawiec, Semantic geometric initialization, in: Eur. Conf. Genet. Program., 2016, pp. 261–277.
[44] E. Galvan-Lopez, B. Cody-Kenny, L. Trujillo, A. Kattan, Using semantics in the selection mechanism in genetic programming: A simple method for promoting semantic diversity, in: 2013 IEEE Congr. Evol. Comput., 2013, pp. 2972–2979.
[45] S. Ruberto, V. Terragni, J.H. Moore, SGP-DT: Semantic genetic programming based on dynamic targets, in: Eur. Conf. Genet. Program. (Part EvoStar), 2020, pp. 167–183.
[46] Q.U. Nguyen, T.H. Chu, Semantic approximation for reducing code bloat in genetic programming, Swarm Evol. Comput. 58 (2020) 100729.
[47] L.F. Miranda, V.B.O.L. Otavio, B.S.M.J. Francisco, G.L. Pappa, Instance selection for geometric semantic genetic programming, in: 2020 IEEE Congr. Evol. Comput., 2020, pp. 1–8.
[48] L. Beadle, C.G. Johnson, Semantically driven mutation in genetic programming, in: 2009 IEEE Congr. Evol. Comput., 2009, pp. 1336–1342.
[49] N.Q. Uy, N.X. Hoai, M. O'Neill, B. McKay, Semantics based crossover for boolean problems, in: Proc. 12th Annu. Conf. Genet. Evol. Comput., 2010, pp. 869–876.
[50] Q.U. Nguyen, X.H. Nguyen, M. O'Neill, Semantic aware crossover for genetic programming: The case for real-valued function regression, in: Eur. Conf. Genet. Program., 2009, pp. 292–302.
[51] N.Q. Uy, M. O'Neill, N.X. Hoai, Predicting the tide with genetic programming and semantic-based crossovers, in: 2010 Second Int. Conf. Knowl. Syst. Eng., 2010, pp. 89–95.
[52] T.A. Pham, Q.U. Nguyen, X.H. Nguyen, M. O'Neill, Examining the diversity property of semantic similarity based crossover, in: Eur. Conf. Genet. Program., 2013, pp. 265–276.
[53] N.Q. Uy, N.X. Hoai, M. O'Neill, R.I. McKay, D.N. Phong, On the roles of semantic locality of crossover in genetic programming, Inf. Sci. 235 (2013) 195–213.
[54] K. Krawiec, Medial crossovers for genetic programming, in: Eur. Conf. Genet. Program., 2012, pp. 61–72.
[55] K. Krawiec, T. Pawlak, Approximating geometric crossover by semantic backpropagation, in: Proc. 15th Annu. Conf. Genet. Evol. Comput., 2013, pp. 941–948.
[56] T.P. Pawlak, B. Wieloch, K. Krawiec, Semantic backpropagation for designing search operators in genetic programming, IEEE Trans. Evol. Comput. 19 (2014) 326–340.
[57] Z. Zhu, A.K. Nandi, M.W. Aslam, Adapted geometric semantic genetic programming for diabetes and breast cancer classification, in: 2013 IEEE Int. Work. Mach. Learn. Signal Process., 2013, pp. 1–5.
[58] M. Castelli, S. Silva, L. Vanneschi, A C++ framework for geometric semantic genetic programming, Genet. Program. Evol. Mach. 16 (2015) 73–81.
[59] M. Castelli, L. Vanneschi, S. Silva, Prediction of the unified Parkinson's disease rating scale assessment using a genetic programming system with geometric semantic genetic operators, Expert Syst. Appl. 41 (2014) 4608–4616.
[60] L. Vanneschi, S. Silva, M. Castelli, L. Manzoni, Geometric semantic genetic programming for real life applications, in: Genet. Program. Theory Pract. Xi, Springer, 2014, pp. 191–209.
[61] A. Moraglio, A. Mambrini, Runtime analysis of mutation-based geometric semantic genetic programming for basis functions regression, in: Proc. 15th Annu. Conf. Genet. Evol. Comput., 2013, pp. 989–996.
[62] Q.U. Nguyen, T.A. Pham, X.H. Nguyen, J. McDermott, Subtree semantic geometric crossover for genetic programming, Genet. Program. Evol. Mach. 17 (2016) 25–53, http://dx.doi.org/10.1007/s10710-015-9253-5.
[63] 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.
[64] M. Castelli, L. Manzoni, L. Vanneschi, S. Silva, A. Popovič, Self-tuning geometric semantic genetic programming, Genet. Program. Evol. Mach. 17 (2016) 55–74.
[65] I. Gonçalves, S. Silva, J.B. Melo, J.M.B. Carreiras, in: A. Moraglio, S. Silva, K. Krawiec, P. Machado, C. Cotta (Eds.), Random Sampling Technique for Overfitting Control in Genetic Programming BT - Genetic Programming, Springer, Berlin, Heidelberg, 2012, pp. 218–229.

[66] M. Castelli, L. Manzoni, S. Silva, L. Vanneschi, A quantitative study of learning and generalization in genetic programming, in: Eur. Conf. Genet. Program., 2011, pp. 25–36.

[67] N.T. Hien, N.X. Hoai, B. McKay, A study on genetic programming with layered learning and incremental sampling, in: 2011 IEEE Congr. Evol. Comput., 2011, pp. 1179–1185.

[68] M.A. Haeri, M.M. Ebadzadeh, G. Folino, Improving GP generalization: A variance-based layered learning approach, Genet. Program. Evol. Mach. 16 (2015) 27–55.

[69] M.A. Haeri, M.M. Ebadzadeh, G. Folino, Statistical genetic programming: The role of diversity, in: Soft Comput. Ind. Appl., Springer, 2014, pp. 37–48.

[70] S.M.H.H. Amini, M. Abdollahi, M.A. Haeri, Rule-centred genetic programming (RCGP): An imperialist competitive approach, Appl. Intell. (2020) 1–21.

[71] G.B. Thompson, II—A brief survey of the species of Mallophaga described from (3) Procellariiformes and (4) Pelecaniformes, Ann. Mag. Nat. Hist. 1 (1938) 23–25.

[72] N. Eldredge, Punctuated equilibria: An alternative to phyletic gradualism, in: Time Fram., Princeton University Press, 2014, pp. 193–224.

[73] T.H. Nguyen, X.H. Nguyen, Learning in stages: A layered learning approach for genetic programming, in: 2012 IEEE RIVF Int. Conf. Comput. Commun. Technol. Res. Innov. Vis. Futur, 2012, pp. 1–4, http://dx.doi.org/10.1109/rivf.2012.6169838.

[74] R.M.A. Azad, C. Ryan, Variance based selection to improve test set performance in genetic programming, in: Proc. 13th Annu. Conf. Genet. Evol. Comput., 2011, pp. 1315–1322.

[75] J. Žegklitz, P. Pošík, Benchmarking state-of-the-art symbolic regression algorithms, Genet. Program. Evol. Mach. 22 (2020) 5–33, http://dx.doi.org/10.1007/S10710-020-09387-0.

[76] W. La Cava, J.H. Moore, Learning feature spaces for regression with genetic programming, Genet. Program. Evol. Mach. 21 (2020) 433–467, http://dx.doi.org/10.1007/S10710-020-09383-4.

[77] K. Yang, X. You, S. Liu, H. Pan, A novel ant colony optimization based on game for traveling salesman problem, Appl. Intell. 50 (2020) http://dx.doi.org/10.1007/s10489-020-01799-w.

[78] M. Amir Haeri, M.M. Ebadzadeh, G. Folino, Statistical genetic programming for symbolic regression, Appl. Soft Comput. 60 (2017) http://dx.doi.org/10.1016/j.asoc.2017.06.050.

[79] M. Nicolau, A. Agapitos, M. O'Neill, A. Brabazon, Guidelines for defining benchmark problems in genetic programming, in: 2015 IEEE Congr. Evol. Comput., CEC 2015 - Proc., 2015, pp. 1152–1159.

[80] 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 (2009) 333–349, http://dx.doi.org/10.1109/TEVC.2008.926486.

[81] E.D. Gribkova, B.A. Ibrahim, D.A. Llano, A novel mutual information estimator to measure spike train correlations in a model thalamocortical network, J. Neurophysiol. 120 (2018) 2730–2744, http://dx.doi.org/10.1152/JN.00012.2018/ASSET/IMAGES/LARGE/Z9K0111848360009.JPEG.

[82] F. Mosteller, J. Tukey, Data analysis and regression: A second course in statistics, 1977, http://www.sidalc.net/cgi-bin/wxis.exe/?IsisScript=AGRIUAN.xis&method=post&formato=2&cantidad=1&expresion=mfn=023366. (Accessed 9 November 2021).

[83] C.J. Cellucci, A.M. Albano, P.E. Rapp, Statistical validation of mutual information calculations: Comparison of alternative numerical algorithms, Phys. Rev. E (3) 71 (2005) http://dx.doi.org/10.1103/PHYSREVE.71.066208.

[84] J.A.K. Suykens, J. Vandewalle, Least squares support vector machine classifiers, Neural Process. Lett. 9 (1999) 293–300.