Extending Local Search in Geometric Semantic Genetic Programming

Mauro Castelli[1], Luca Manzoni[2], Luca Mariot[2], and Martina Saletta[1]

[1] NOVA Information Management School (NOVA IMS)
Universidade Nova de Lisboa
Campus de Campolide, 1070-312 Lisboa, Portugal
{mcastelli,msaletta}@novaims.unl.pt
[2] Dipartimento di Informatica, Sistemistica e Comunicazione (DISCo)
Università degli Studi di Milano Bicocca
Viale Sarca 336, 20126 Milano, Italy.
{luca.manzoni,luca.mariot}@unimib.it

Abstract. In this paper we continue the investigation of the effect of local search in *geometric semantic genetic programming* (GSGP), with the introduction of a new general local search operator that can be easily customized. We show that it is able to obtain results on par with the current best-performing GSGP with local search and, in most cases, better than standard GSGP.

1 Introduction

Genetic programming (GP) [12], in particular in the standard tree-based representation, has proved to be a powerful method to automatically build symbolic expressions and programs for solving problems in a wide variety of domains. Recently, the introduction of Geometric Semantic Genetic Programming (GSGP) and the new ideas related to the definition of Geometric Semantic Operators (GSO) [15] allowed to solve problems more efficiently and to produce better solutions [10,5,3]. Recently, GSGP has been improved by replacing the mutation GSO with a local search operator: the application of this local search operators allows the search to rapidly improve in the first few generations, thus increasing the speed of convergence [6]. However, the effect of local search and the best way to employ it in conjunction with GSGP are still not well understood. In fact, among the several possible ways of combining GSGP with local search, only one, called GSGP-LS, has been investigated in [6]. Here, we introduce a new method to perform this local search operation, namely by defining a generic set of functions to locally modify a candidate GP individual. In particular, the possibility of modifying the set of functions allows one to easily customize the local search operator, by making it suited for the problem at hand.

We compare our newly proposed method, which we call GSGP-reg, with GSGP and GSGP-LS, showing that in most cases GSGP-reg outperforms GSGP and achieves results similar to those of GSGP-LS, although with a different "fitness profile". That is, the problems in which GSGP-LS and GSGP-reg produce

overfitting solutions are not the same, showing that the selection of the best local search operators for GSGP is still an open (and interesting) problem.

The paper is organized as follows: Section 2 recalls the basic concepts of GSGP, while Section 3 provides a short survey of the existing works linking local search with GP. Then, Section 4 defines our proposed integration of local search in GSGP, namely GSGP-reg. In Section 5 the settings of the experiments performed are then introduced and the datasets used are described in Section 6. The results of the experiments and their discussion are the topics of Section 7. Finally, some directions for future research are highlighted in Section 8.

2 Geometric Semantic Genetic Programming

GSGP was originally defined by Moraglio and coworkers in 2012 [15]. The main idea is that mutation and crossover operators can be defined in such a way that the effects on the semantics of the individuals are predictable, differently from the usual syntactic crossover and mutation. They were successful in defining those operators, and proved that GSO induce a unimodal fitness landscape, in which the unique global optimum is known and the fitness is derived from the distance from this global optimum.

In particular, the geometric semantic crossover between two trees T_1 and T_2 is defined as

$$R \cdot T_1 + (1 - R) \cdot T_2$$

where R is a randomly generated tree with outputs in $[0, 1]$. The geometric semantic mutation of a tree T is defined as:

$$T + ms \cdot (R_1 - R_2)$$

where ms is a positive constant (called the *mutation step*) and both R_1 and R_2 are randomly generated trees with outputs in $[0, 1]$.

While GSGP produces a "nice" fitness landscape, in its original formulation the crossover operator induces an exponential increase in the size of the individuals with respect to the number of generations, as already remarked when GSGP was introduced [15]. A different representation of the individual was introduced shortly after in [18], where the individuals are still trees at the logical level, but they are represented in memory as directed graphs. This new way of implementing GSGP allowed to obtain better performances than classical tree-based GP with shorter execution times.

3 Related work

This section reports some of the most important works related to the method described in the rest of this paper. Most of the existing methods were specifically designed for standard syntax-based GP and taking into account symbolic regression problems. Thus, it is fundamental to frame the context in which the existing techniques were developed.

The main objective in addressing a symbolic regression problem is to search for the symbolic expression $K^O : \mathbb{R}^p \to \mathbb{R}$ that best fits a particular training set $\mathbb{T} = \{(x_1, y_1), \ldots, (x_n, y_n)\}$ of n input/output pairs with $x_i \in \mathbb{R}^p$ and $y_i \in \mathbb{R}$.

Then, following the same formulation proposed by Castelli and coauthors [6], a symbolic regression problem can be formally defined as

$$(K^O, \boldsymbol{\theta^O}) \leftarrow \underset{K \in \mathbb{G}; \theta \in \mathbb{R}^m}{arg\ min}\ f(K(x_i, \theta), y_i) \text{ with } i = 1, \ldots, p\ ,$$

where $\mathbb{G}$ is the solution or syntactic space defined by the primitive set $\mathbb{P}$ (functions and terminals), while f is the fitness function based on the distance between a program's output $K(x_i, \theta)$ and the expected output y_i (such as the root mean square error–RMSE), and $\theta \in \mathbb{R}^m$ is a particular parametrization of the symbolic expression K.

Standard GP operators work at the syntax level, without taking into account the effects on the semantics. Nonetheless, GP was able to successfully solve problems in different domains [13]. Despite that, the impossibility to optimize the parameters of the model translates into significant limitations, such as search stagnation, bloat [19] and solutions that are poorly understandable [14,6]. This is mostly due to the fact that GP performs a highly-exploratory search, characterized by large fitness changes when a modest syntactic modification occurs and vice-versa [6].

Different works were proposed to include a local search strategy into evolutionary algorithms [9,16]. The common idea shared by these methods consists of defining an operator that, given a candidate solution, is able to exploit the local region around that solution to search for the best neighbor.

Considering the particular case of GP, it is possible to distinguish two main methods for applying a local search (LS) strategy: apply LS either on the syntax or on the numerical parameters of the program [14,6].

With respect to the first approach, Azad and Ryan [2] proposed the use of local search to change the fitness of individuals during their lifetime. While the proposed system was not the first attempt to include LS in GP, it was easy to understand and cheap to implement [2]. Their results show that GP with LS outperforms standard GP over different symbolic regression problems. Moreover, they show that the system uses the available genetic material more efficiently than standard GP, and that the training process produces smaller individuals.

With respect to the second approach, several works are worth to be mentioned. In [17], authors studied the effectiveness of gradient search optimization of numeric leaf values in GP individuals. The results reported by the authors showed that local learning yielded an improved approximation accuracy, even if they optimized only the value of the terminal nodes of the trees.

A similar approach was proposed in the work of Zhang and Smart [22], where a LS algorithm was integrated into the GP search process to optimize the value of the terminal nodes.

In [21] the authors investigated a Lamarckian memetic GP, incorporating a LS strategy to refine GP individuals expressed as syntax trees. The authors tested different heuristic methods to determine which individuals should be subject to

LS, showing that better results can be obtained by applying LS to all individuals in the population or to a subset of the best individuals. All in all, the results demonstrated that including a LS strategy in GP is beneficial both in terms of convergence and performance, as well as limiting code growth.

The use of LS in GP for symbolic regression was also proposed in [14], where the authors integrated a LS optimizer as an additional search operator. The results showed that the use of the LS operator helps improving the convergence and performance of tree-based GP, while reducing the size (i.e., the number of nodes) of the trees.

With respect to GSGP, to the best of our knowledge, the only work published in this line of research is the one proposed in [6], where the authors modified the original geometric semantic mutation (GSM) operator to integrate a greedy LS optimizer. Given an individual T, the resulting operator (called GSM-LS) was defined as follows:

$$T' = \alpha_0 + \alpha_1 \cdot T + \alpha_2 \cdot (R_1 - R_2)$$

where R_1 and R_2 are random trees with output in $[0, 1]$, while $\alpha_i \in \mathbb{R}$. In particular, α_2 replaces the mutation step parameter ms that characterizes the geometric semantic mutation operator.

As reported in [6], the GSM-LS operator tries to determine the best linear combination of the parent tree and the random trees used to perturb it, and it is local in the sense of the linear problem defined by the GSM operator. When compared against the original GSM operator, GSM-LS was able to improve the convergence speed of the search process, and reduced the size of the resulting solution [6,8,11,4].

4 A New Way to Perform Local Search

In this paper we build on top of the GSGP-LS idea, whereby differently from the work described in [6], we apply LS to all the individuals during a separate step after mutation and crossover.

Let $T : \mathbb{R}^p \to \mathbb{R}$ be a GP individual encoded by a tree which is defined over a set of primitives $\mathbb{P}$, and let $s(T) = (T(x_1), T(x_2), \ldots, T(x_n))$ be its semantic vector computed on the inputs $X = (x_1, \ldots, x_n)$, where $x_i \in \mathbb{R}^p$ and $T(x_i) \in \mathbb{R}$ for all $i \in \{1, \cdots, p\}$. Further, let $Y = (y_1, \ldots, y_n) \in \mathbb{R}^n$ be the vector of target values associated to X. In particular, the *training set* $\mathbb{T} = \{(x_1, y_1), \cdots, (x_n, y_n)\}$ is the diagonal of the Cartesian product $X \times Y$. In what follows, we assume that the fitness f of the individual T is the *mean squared error* (MSE) of T in predicting Y from X, i.e.

$$f(T) = \frac{1}{n} \sum_{i=1}^{n} (T(x_i) - y_i)^2 \ .$$

Remark that, however, our local search method can be defined with any order-preserving transformation of the MSE as the underlying fitness function, such as the *RMSE*.

A first idea to introduce a local search step in GSGP is to define a regression problem that aims at minimizing the error between the values in the vector $s(T)$ and Y as follows: find two coefficients $\alpha^*, \beta^* \in \mathbb{R}$ that minimize the sum of the squares of the differences between $\alpha T(x_i) + \beta$ and y_i. Formally, for all $i \in \{1, \cdots, n\}$ we have

$$T'(x_i) = \alpha^* T(x_i) + \beta^* \ , \text{ where } (\alpha^*, \beta^*) = \underset{\alpha,\beta \in \mathbb{R}}{arg\ min} \{f(\alpha T(x_i) + \beta)\} \ .$$

In other words, the original tree T is replaced by an affine transformation T', which is then encoded as a new GP individual. Since $\alpha = 1$ and $\beta = 0$ is a valid solution, the resulting individual T' will not have a worse fitness than that of the original individual T over the training set.

The idea of replacing a tree with an affine transformation of it can be generalized to allow more complex transformations of a GP tree. In particular, let $\mathcal{F} = \{f_1, \ldots, f_k : \mathbb{R} \to \mathbb{R}\}$ be a collection of k real functions. We can thus define the semantic vectors for all $1 \leq i \leq k$ as

$$s(f_i \circ T) = (f_i(T(x_1)), f_i(T(x_2)), \cdots, f_i(T(x_n))) \ .$$

Similarly to the affine transformation case, one can define a regression problem using the components of the vectors $s(f_i \circ T)$ in the following way: find k coefficients $\alpha_1^*, \cdots, \alpha_k^* \in \mathbb{R}$ that minimize the sum of the squares of the differences between y_i and $T'(x_i)$, where

$$T'(x_i) = \sum_{j=1}^{k} \alpha_j f_j(T(x_i)) = \alpha_1^* f_1(T(x_i)) + \alpha_2^* f_2(T(x_i)) + \cdots + \alpha_k^* f_k(T(x_i)) \quad (1)$$

for all $1 \leq i \leq n$. Therefore, the formulation of the regression problem becomes

$$(\alpha_1^*, \cdots, \alpha_k^*) = \underset{\alpha_1, \cdots \alpha_k \in \mathbb{R}}{arg\ min} \left\{ f\left(\sum_{j=1}^{k} \alpha_j f_j(T(x_i)) \right) \right\} \ . \quad (2)$$

Clearly, one obtains a direct generalization of the previous regression problem if set $\mathcal{F}$ includes both a non-zero constant function and the identity function. The goodness of the solutions obtained by solving this linear regression problem depends on the particular set $\mathcal{F}$, which can include non-linear functions as well.

Our modified version of GSGP uses the linear regression problem defined by Equations (1), with coefficient given by Equation (2), as an additional local search step that tries to exploit the structure of the candidate solutions. In particular, this step is applied at each generation over all individuals in the current population after having applied semantic crossover and mutation, and before the insertion in the new population.

5 Experimental Settings

In the following, GSGP denotes standard Geometric Semantic GP, GSGP-LS the variant of GSGP with the local search mutation operator introduced in [6], and GSGP-reg the regression-based method described in Section 4.

The set of functional symbols employed in the experiments was $\{+, -, \times, \div\}$, where $\div$ is division, protected by returning 1 when the denominator is sufficiently close to 0. The set of terminal symbols was the set of all input variables over which the symbolic expression encoded by a GP individual is defined. In particular, no fixed constant was used in the terminal set. All GSGP variants investigated in our experiments adopted a generational evolutionary strategy, using tournament selection with tournament size $t = 4$. Each offspring individual was created by applying either semantic crossover or mutation, with a probability respectively of 0.6 and 0.4. The population size was set at 250 individuals, generated through a ramped half-and-half method with a maximum initial depth of 6. Survival of the best candidate solution in the population was assured by employing elitism with replacement of a random individual. The trees used in the semantic mutation and crossover operators were randomly generated with a maximum depth of 6, and their values constrained between 0 and 1 by using a logistic function.

Both GSGP-LS and GSGP-reg performed the local search step only for the first 10 generations; then, the algorithm switched to GSGP. As shown in [6], this limits the overfitting introduced by the local search procedure. Further possibilities, like the application every k generations, will be the aim of future investigations. As for GSGP-reg, we performed some preliminary explorations with different sets of functions for the regression step. This showed that a good trade-off between performances on the training set and the avoidance of overfitting was given by the four functions $f_1(x) = 1$ (constant), $f_2(x) = x$ (identity), $f_3(x) = \max(0, x)$ (positive part), and $f_4(x) = \min(0, x)$ (negative part).

In all considered problems, a 70/30 random split between train and test sets was used. To ensure the statistical validity of the results, we performed 100 runs of 500 generations on each test problem, using a different random split of training and test sets in each run. We used the RMSE as the fitness function to minimize in all problems.

For all test problems we recorded the median and the median absolute deviation (MAD) of the fitness obtained by the best individual of the population, since they are more resistant to outliers than the average and the standard deviation. The results obtained by the three methods on the test sets were also compared among themselves using the Mann-Whitney U-test, adopting the alternative hypothesis that the fitness achieved by the first method (either GSGP or GSGP-LS) was greater, and thus worse, than the fitness achieved by the second method (either GSGP-LS or GSGP-reg). The significance level adopted for the tests was $\alpha = 0.05$. The Mann-Whitney U-test was used since it makes no assumption on the distribution underlying the samples.

6 Regression Problems Used for Testing

We performed our experiments over five regression problems. In particular, the first three come from the domain of pharmacokinetics, and concern the prediction of three different parameters featured by a set of chemical compounds that are considered for potential drug development, and which are represented by their

Table 1. Sizes of the considered datasets.

	%F	%PPB	TOX	COMP	SLUMP
#Instances	260	131	234	1030	102
#Features	242	627	627	8	9

molecular structure. On the other hand, the last two problems pertain civil engineering, and specifically consist in predicting two parameters of concrete based on the mix of ingredients used to produce it. We briefly describe each of the considered problems, and summarize in a table the dimensions of the respective datasets at the end of this section. For further information, the reader may refer to [1,6] for the pharmacokinetics problems and to [7,20] for the concrete problems. In our experiments we adopted the same datasets used in those works. Table 1 summarizes the sizes of the five considered datasets. The "#Features" row includes both the input features and the output value of the parameter.

Human Oral Bioavailability (%F). Human Oral Bioavailability (shortened as %F) is a pharmacokinetic parameter which measures the quantity of an orally-administered drug that actually reaches blood circulation after being processed by the liver. The dataset adopted in our experiments is composed of 260 molecules instances, each of them represented by 241 molecular descriptors and the corresponding value of %F.

Plasma Protein Binding (%PPB). Plasma Protein Binding (indicated as %PPB in what follows) is a parameter more specific than %F, since it measures the quantity of drug that reaches circulation and further attaches to plasma proteins in the blood. The dataset is composed of 131 molecules instances, where each instance is described by 626 features and the associated value of %PPB.

Median Lethal Dose (TOX). Median Lethal Dose (informally referred to as *toxicity*, and abbreviated as TOX) measures the quantity of drug which is necessary to kill half of the test organisms. As noted in [1], one can have different toxicity parameters depending on the specific test organism and administration route. The parameter considered in our experiments is the one used in [1,6], which concerns mice as test organisms and oral supplying as an administration route. The dataset is composed of 234 molecules instances which, as in the %PPB dataset, are described by 626 features and the corresponding value of TOX.

Concrete Compressive Strength (COMP). Concrete Compressive Strength (abbreviated as COMP in the following) is a parameter that measures how much a particular mix of concrete can resist compression forces. The dataset of our experiments is composed of 1030 instances of concrete mix, each described by 7 features (i.e. the ingredients composing the mix) and the corresponding value of COMP.

Table 2. Median and MAD of the fitness obtained by the best individual.

Dataset			GSGP	GSGP-LS	GSGP-reg
%F	Training set	Median	31.5945	**22.8211**	23.4196
		MAD	0.7531	0.3585	0.3733
	Test set	Median	33.2053	**30.7070**	30.7311
		MAD	1.2162	2.0702	3.2942
%PPB	Training set	Median	20.5467	**4.8129**	5.2265
		MAD	0.7768	0.7783	1.0574
	Test set	Median	**36.5051**	38.2085	56.3562
		MAD	5.0257	4.0538	25.0935
TOX	Training set	Median	2159.6356	**1650.8873**	1768.6019
		MAD	68.2308	48.0973	52.6530
	Test set	Median	2223.4332	2290.4879	**2209.9571**
		MAD	157.9263	328.4134	210.1360
COMP	Training set	Median	8.3835	**5.5940**	5.8519
		MAD	0.4053	0.1002	0.1618
	Test set	Median	8.9826	**6.3625**	6.6204
		MAD	0.6422	0.2255	0.2179
SLUMP	Training set	Median	1.6911	**0.8124**	0.8956
		MAD	0.2336	0.0757	0.0746
	Test set	Median	4.4309	**2.8535**	2.9441
		MAD	0.8338	0.4305	0.4482

Concrete Slump (SLUMP). Concrete Slump (indicated as SLUMP) is a parameter that measures the consistency of fresh concrete. The dataset employed for our tests is composed of 102 instances described by 8 input features, plus the corresponding SLUMP value.

7 Experimental Results

Figures 1 to 5 report the plots of the median fitness for the three compared methods (GSGP, GSGP-LS and GSGP-reg) over the five considered datasets. In particular, the left part (respectively, right part) of each figure refers to the median fitness achieved by each method in 500 generations over 100 experimental runs on the training set (respectively, test set) of the relevant problem. The median fitness is also reported in Table 2 for each problem, along with the associated median absolute deviation (MAD). In general, one can see from the plots of the training sets that both GSGP-LS and GSGP-reg perform better than pure GSGP over all considered datasets. This is an expected outcome, since as observed in [6] local search tends to overfit the datasets when applied to each generation of the GSGP algorithm. As discussed in Section 5, this is the reason why we investigated a hybrid version of both GSGP-LS and GSGP-reg, where the local search step is not applied after 10 generations. On the other hand, one can still observe on the test sets that GSGP-LS and GSGP-reg generally fare better than pure GSGP, except over the %PPB and TOX datasets. In the

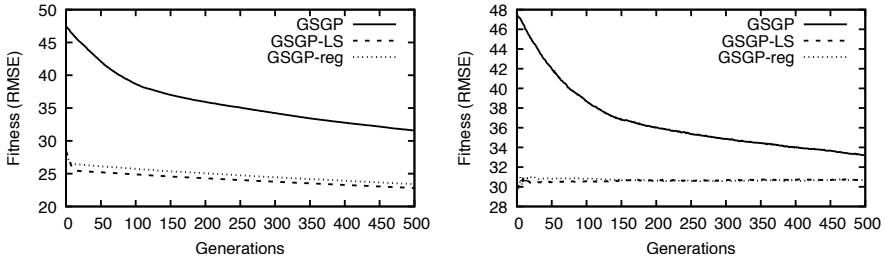

Fig. 1. Median fitness on the training (left) and test (right) sets for the %F dataset.

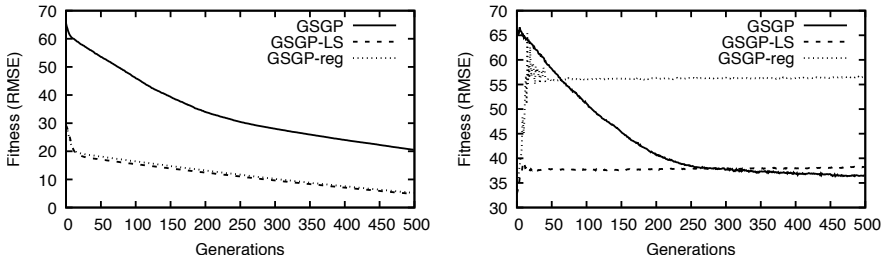

Fig. 2. Median fitness on the training (left) and test (right) sets for the %PPB dataset.

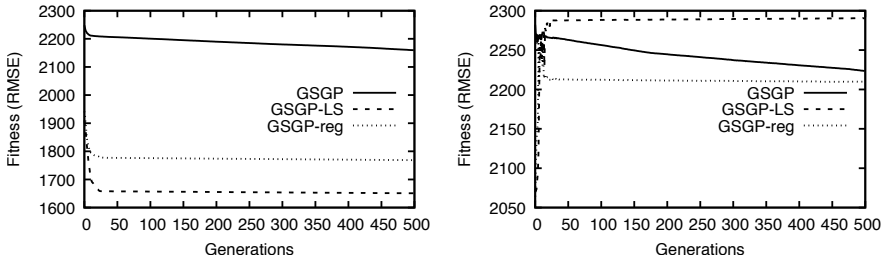

Fig. 3. Median fitness on the training (left) and test (right) sets for the TOX dataset.

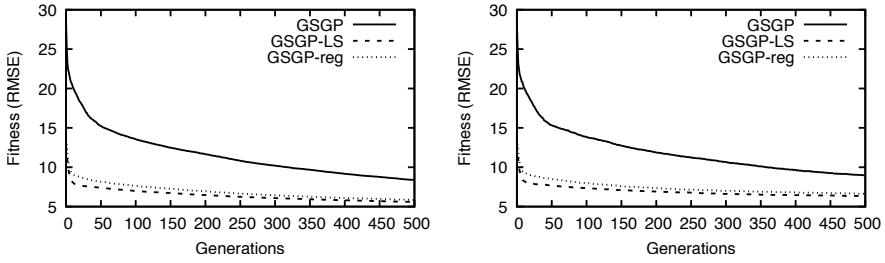

Fig. 4. Median fitness on the training (left) and test (right) sets for the COMP dataset.

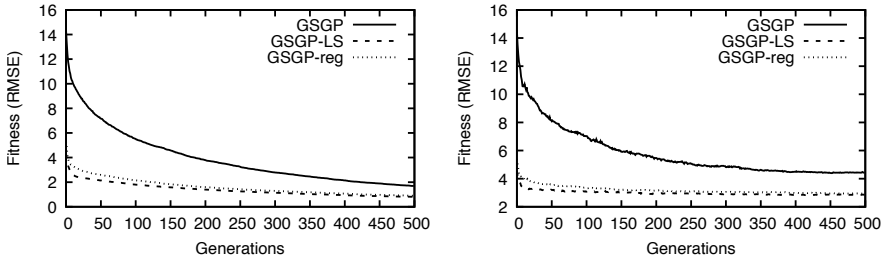

Fig. 5. Median fitness on the training (left) and test (right) sets for the SLUMP dataset.

Table 3. p-values obtained with the Mann-Whitney U test.

Dataset	GSGP vs GSGP-LS	GSGP vs GSGP-reg	GSGP-LS vs GSGP-reg
%F	3.0863E−7	0.0933	0.7386
%PPB	0.7159	0.9999	0.9999
TOX	0.9952	0.8231	0.0316
COMP	2.5092E−33	2.0975E−28	0.9999
SLUMP	1.1896E−16	2.4460E−20	0.7425

former case, our generalization of local search is the worse performer among the three algorithms, the median fitness values of GSGP and GSGP-LS being lower than GSGP-reg and close to each other. In the latter, GSGP-reg actually scores the best performance, while GSGP-LS achieves the highest median fitness after 500 generations. Regarding the comparison of the two versions of GSGP with local search, one cannot rely on the plots alone, except for the %PPB and TOX datasets. In fact, as it can be seen in the right parts of Figures 1, 4 and 5, the plots of the median fitness values of GSGP-LS and GSGP-reg are almost superimposed. For this reason, we performed a more thorough comparison using the Mann-Whitney U-test. Table 3 reports the p-values of all three comparisons (GSGP vs. GSGP-LS, GSGP vs. GSGP-reg and GSGP-LS vs. GSGP-reg) over the test sets of the five considered problems. However, by looking at the fourth column of Table 3, it can be seen that under the considered significance level ($\alpha = 0.05$) one cannot reject the null hypothesis, i.e. that the performance of GSGP-LS is not worse than that of GSGP-reg over %F, COMP and SLUMP. The p-value corresponding to the %PPB dataset actually shows that GSGP-LS is not worse than GSGP-reg as well. On the other hand, the null hypothesis is rejected for the TOX dataset ($p = 0.0316$), thereby confirming our observation above regarding the median fitness plot on the test set of Figure 3. As a final note, for both GSGP-LS and GSGP-reg, the additional computational resources needed did not increase significantly the time required to perform the evolution.

8 Conclusion

In this paper we have defined a new local search operator for GSGP called GSGP-reg, and compared it with both the classical version of GSGP and the current best performing combination of GSGP with local search, called GSGP-LS, as defined in [6]. We were able to outperform classical GSGP in most cases, and our proposed method is on par with the performances of GSGP-LS. However, the problems in which GSGP-reg and GSGP-LS show overfitting are different. Therefore, it would be interesting to understand what causes similar performances in certain datasets and remarkably different generalization behavior in others.

There are multiple interesting research directions still open. First of all, an in-depth study of the best functions families $\mathcal{F}$ to be employed for regression should be performed, thereby expanding the preliminary exploration presented here, and analyzing how each of these families influences the learning process and the ability to generalize. The regression method can also be tuned in multiple ways. For example, one could generate multiple regression models on different subsets of the training data and select one with the best generalization on the part of the training set that was not used for the generation of the linear regression model. Finally, for classification problems a similar method can be employed by using logistic regression instead of the traditional linear regression.

Acknowledgments. This work was partially supported by national funds through FCT (Fundação para a Ciência e a Tecnologia) under project DSAIPA/DS/0022/2018 (GADgET).

References

1. Archetti, F., Lanzeni, S., Messina, E., Vanneschi, L.: Genetic programming for computational pharmacokinetics in drug discovery and development. Genetic Programming and Evolvable Machines **8**(4), 413–432 (2007)
2. Azad, R.M.A., Ryan, C.: A simple approach to lifetime learning in genetic programming-based symbolic regression. Evolutionary Computation **22**(2), 287–317 (2014)
3. Castelli, M., Manzoni, L., Vanneschi, L., Silva, S., Popovič, A.: Self-tuning geometric semantic genetic programming. Genetic Programming and Evolvable Machines **17**(1), 55–74 (2016)
4. Castelli, M., Trujillo, L., Vanneschi, L.: Energy consumption forecasting using semantic-based genetic programming with local search optimizer. Computational Intelligence and Neuroscience **2015**, 57 (2015)
5. Castelli, M., Trujillo, L., Vanneschi, L., Popovič, A.: Prediction of relative position of ct slices using a computational intelligence system. Applied Soft Computing **46**, 537 – 542 (2016)
6. Castelli, M., Trujillo, L., Vanneschi, L., Silva, S., et al.: Geometric semantic genetic programming with local search. In: Proceedings of the 2015 Annual Conference on Genetic and Evolutionary Computation. pp. 999–1006. ACM (2015)
7. Castelli, M., Vanneschi, L., Silva, S.: Prediction of high performance concrete strength using genetic programming with geometric semantic genetic operators. Expert Syst. Appl. **40**(17), 6856–6862 (2013)

8. Castelli, M., Vanneschi, L., Trujillo, L., Popovič, A.: Stock index return forecasting: semantics-based genetic programming with local search optimiser. International Journal of Bio-Inspired Computation **10**(3), 159–171 (2017)
9. Chen, X., Ong, Y.S., Lim, M.H., Tan, K.C.: A multi-facet survey on memetic computation. Transactions on Evolutionary Computation **15**(5), 591–607 (2011)
10. Enríquez-Zárate, J., Trujillo, L., de Lara, S., Castelli, M., Emigdio, Z., Muñoz, L., Popovič, A., et al.: Automatic modeling of a gas turbine using genetic programming: An experimental study. Applied Soft Computing **50**, 212–222 (2017)
11. Hajek, P., Henriques, R., Castelli, M., Vanneschi, L.: Forecasting performance of regional innovation systems using semantic-based genetic programming with local search optimizer. Computers & Operations Research **106**, 179 – 190 (2019)
12. Koza, J.R.: Genetic programming: on the programming of computers by means of natural selection. MIT press (1992)
13. Koza, J.R.: Human-competitive results produced by genetic programming. Genetic Programming and Evolvable Machines **11**(3-4), 251–284 (2010)
14. Leonardo, T., Z-Flores, E., Juarez Smith, P.S., Legrand, P., Silva, S., Castelli, M., Vanneschi, L., Schütze, O., Munoz, L.: Local Search is Underused in Genetic Programming. In: Arbor, A. (ed.) Genetic Programming Theory and Practice XIV. Springer (2017)
15. Moraglio, A., Krawiec, K., Johnson, C.G.: Geometric semantic genetic programming. In: International Conference on Parallel Problem Solving from Nature. pp. 21–31. Springer (2012)
16. Neri, F., Cotta, C., Moscato, P.: Handbook of memetic algorithms, vol. 379. Springer (2012)
17. Topchy, A., Punch, W.F.: Faster genetic programming based on local gradient search of numeric leaf values. In: Proceedings of the 3rd Annual Conference on Genetic and Evolutionary Computation. pp. 155–162. GECCO'01, Morgan Kaufmann Publishers Inc., San Francisco, CA, USA (2001)
18. Vanneschi, L., Castelli, M., Manzoni, L., Silva, S.: A new implementation of geometric semantic GP and its application to problems in pharmacokinetics. In: European Conference on Genetic Programming. pp. 205–216. Springer (2013)
19. Vanneschi, L., Castelli, M., Silva, S.: Measuring bloat, overfitting and functional complexity in genetic programming. In: Proceedings of the 12th Annual Conference on Genetic and Evolutionary Computation. pp. 877–884. ACM (2010)
20. Yeh, I.C.: Modeling of strength of high-performance concrete using artificial neural networks. Cement and Concrete Research **28**(12), 1797 – 1808 (1998)
21. Z-Flores, E., Trujillo, L., Schütze, O., Legrand, P.: Evaluating the effects of local search in genetic programming. In: Tantar, A.A., Tantar, E., Sun, J.Q., Zhang, W., Ding, Q., Schütze, O., Emmerich, M., Legrand, P., Del Moral, P., Coello Coello, C.A. (eds.) EVOLVE - A Bridge between Probability, Set Oriented Numerics, and Evolutionary Computation V. pp. 213–228. Springer International Publishing, Cham (2014)
22. Zhang, M., Smart, W.: Genetic programming with gradient descent search for multiclass object classification. In: Keijzer, M., O'Reilly, U.M., Lucas, S., Costa, E., Soule, T. (eds.) Genetic Programming. pp. 399–408. Springer Berlin Heidelberg, Berlin, Heidelberg (2004)