Extending Local Search in Geometric Semantic Genetic Programming

Mauro Castelli (1), Luca Manzoni (2), Luca Mariot (2), Martina Saletta (1)

1. NOVA Information Management School (NOVA IMS)Universidade Nova de LisboaLisbonPortugal
2. Dipartimento di Informatica Sistemistica e Comunicazione (DISCo)Università degli Studi di Milano BicoccaMilanItaly

This is the Author Peer Reviewed version of the following conference paper published by Springer:

Castelli, M., Manzoni, L., Mariot, L., & Saletta, M. (2019). Extending local search in geometric semantic genetic programming. In P. Moura Oliveira, P. Novais, & L. P. Reis (Eds.), Progress in Artificial Intelligence : 19th EPIA Conference on Artificial Intelligence, EPIA 2019, Proceedings (pp. 775-787). (Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics); Vol. 11804 LNAI). Springer Verlag. https://doi.org/10.1007/978-3-030-30241-2_64, *which has been published in final form at* https://doi.org/10.1007/978-3-030-30241-2_64

Extending Local Search in Geometric Semantic Genetic Programming

No Author Given

No Institute Given

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 general new 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) [**?**], in particular in the standard tree-based representation, has been proved to be a powerful method to automatically build symbolic expressions and programs to solve 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) [**?**] allowed to solve problems more efficiently and to produce better solutions. Still, GSGP had the ability to be improved further by combining it with a local search operation, replacing the GSO of mutation [**?**] with it. The application of local search operators allows the search to rapidly improve in the first few generations, thus increasing the speed of convergence.

The effect of local search and the best way to employ it in conjunction with GSGP are, however, still not well understood. In fact, among the many possible ways of combining GSGP with local search, only one, called GSGP-LS, has been defined. Here we introduce a new method to perform this local search operation, in particular we allow the section of a set of functions to be employed during the local search phase that can, potentially, allow to easily define custom local search operators suited for the problem at hand.

We compare our newly proposed method, which we call GSGP-reg, with GSGP and GSGP-LS, showing that most of the time we outperform GSGP and our results are on-par with GSGP-LS, but 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: in Section 2 we recall the definition of GSGP, then, in Section 3 we perform a short survey of the existing works linking local search with GP. We then define GSGP-reg in Section 4. The settings of the experiments performed are then introduced in Section 5 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 [**?**], with the main idea of defining mutation and crossover operators for which the effects on the semantics of the individuals were 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 produces an exponential increase in the size of the individuals with respect to the number of generations. A different representation of the individual, that logically remain trees, but are represented in memory as graphs, was introduced shortly after, in 2013 [**?**]. This new way of implementing GSGP allowed to obtain better performances than classical tree-based GP with shorter execution times.

A disadvantage of this implementation is that the trees, being represented in a succinct way, once "expanded" are too large to be interpreted by a practitioner, thus reducing one of the advantages that classical GP has. Namely, that it produces readable solutions.

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 taken 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 [?], a symbolic regression problem can be formally defined as

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

where $\mathbb{G}$ is the solution or syntactic space defined by the primitive set $\mathbb{P}$ (functions and terminals), f is the fitness function based on the distance between a program's output $K(x_i, \boldsymbol{\theta})$ and the expected output y_i, like for instance the root mean square error (RMSE) as in this work, and $\boldsymbol{\theta}$ is a particular parametrization of the symbolic expression K, assuming m real-valued parameters.

While in standard GP parameter optimization is usually not performed explicitly (i.e., genetic operators focus on the syntax), GP was able to successfully solve problems in different domains [?]. Despite that, the impossibility to optimize the parameters of the model translates into significant limitations, such as search stagnation, bloat [?] and solutions that are poorly understandable [?,?]. This is mostly due to the fact that GP performs a highly-explorative search, characterized by large fitness changes when a modest syntactic modifications occurs and viceversa [?].

Different works were proposed to include a local search strategy into evolutionary algorithms [?,?]. 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 a LS on the syntax or apply it on numerical parameters of the program [?,?].

With respect to the first approach, Azad and Ryan [?] 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 is easy to understand and cheap to implement [?]. 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 [?], authors studied the effectiveness of gradient search optimization of numeric leaf values for GP. The results reported by the authors showed that local learning produced an improved approximation accuracy, even if they only optimize the value of the terminal nodes of the trees.

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

In [?] the authors investigated a Lamarckian memetic GP, that incorporates a LS strategy to refine GP individuals expressed as syntax trees. Authors tested different heuristic methods to determine which individuals should be subject to a LS showing that better results can be obtained by applying LS to all the individuals in the population or to a subset of the best individuals. All in all, the

results demonstrated that including a LS strategy is beneficial in terms of both convergence and performance, and it also limit the code growth.

The use of LS in GP for symbolic regression was also proposed in [**?**], where authors incorporated a LS optimizer as an additional search operator. 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 is the one proposed in [**?**], where 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:

$$T_M = \alpha_0 + \alpha_1 \cdot T + \alpha_2 \cdot (R_1 - R_2)$$

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

As reported in [**?**], 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 posed by the GSM operator. When compared against the original GSM operator, GSM-LS was able to improve the convergence speed of the search process, thus reducing the size of the resulting solution [**?**,**?**,**?**,**?**].

In this paper we build on the top of this idea, by integrating the LS into the GSGP search process and, differently with respect to the work described in [**?**], by applying LS to all the individuals and not limiting the use of LS within the mutation operator.

4 A New Way to Perform Local Search

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 root mean square error.

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 have fitness which is no worse than that of the original individual T over the training set.

The above observation can be generalized to allow more complex transformations of a GP tree as follows. 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 a non-zero constant function and the identity function both belong to the set $\mathcal{F}$. 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. In fact, it would not be useful to have two linearly dependent functions in $\mathcal{F}$, since one can be expressed in term of the other.

Our modified version of GSGP uses the linear regression problem defined by Equations (1) and (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 a local search mutation operator introduced in [**?**], and GSGP-reg the regression-based method introduced in this paper.

The set of functional symbols employed in the experiments was $\{+, -, \times, \div\}$, where $\div$ is division, protected returning 1 when the denominator is sufficiently close to 0. The set terminal symbols is the set of all input variables, no fixed constant was used in it. Crossover was employed 60% of the time and mutation the remaining 40%, with a mutation step randomly selected in $[0, 1]$ at each mutation event. The population size was set at 250 individuals, generated with a ramped half-and-half method with a maximum initial depth of 6. Survival of the best individual in the population was assured by employing elitism. The random trees used in the semantic mutation and crossover operations were randomly generated with a depth of 6 and their values constrained between 0 and 1 by using a logistic function.

For both GSGP-LS and GSGP-reg, the local search was performed for the first 10 generations, then the algorithm switched to GSGP. As shown in [**?**], this allows to limit the overfitting introduced by the local search procedure.

As for GSGP-reg, we performed some preliminary explorations with different sets of functions for the regression step. This shown that a good trade-off between performances on the training set and the avoidance of overfitting was given by the four function $f_1(x) = 1$ (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 problems considered, a 70/30 random split between train and test was used. To ensure the statistical validity of the results, we performed 100 runs of 500 generations on each test problem, each time with different training and test sets. In all problems the fitness was the *root mean square error* (RMSE), which was minimized.

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. Contrarily to the average and the standard deviation, the median and the MAD are more resistant to outliers. The results of the different tested methods on the test set at the last generation were also compared with the other methods using the Mann-Whitney U-test with the alternative hypothesis that the fitness obtained by the first method (either GSGP or GSGP-LS) were greater, and thus worse, then the ones produced by the second method (either GSGP-LS or GSGP-reg).

6 Regression Problems Used for Testing

We performed our experiments over five regression problems. In particular, the first three comes 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 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 [**?**,**?**] for the pharmacokinetics problems and to [**?**,**?**] for the concrete problems. In our experiments we adopted the same datasets used in those works.

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. In this case, 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 [**?**], depending on the specific test organism and the administration route, one can have several different toxicity parameters. The particular toxicity parameter considered in our experiments is the used in [**?**,**?**], and it concerns mice as test organisms and oral supplying as an administration route. The dataset is composed of 234 molecules instances which, analogously to 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 to loads that compress it. The dataset used in 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.

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.

Table 1 summarizes the sizes of the five considered datasets. The column "features" includes both the input features and the output value of the parameter.

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.

Table 1. Sizes of the considered datasets.

	%F	%PPB	TOX	COMP	SLUMP
#Instances	260	131	234	1030	102
#Features	241	627	627	8	9

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 dataset.

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 [**?**] the local search step 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 an hybrid version both of GSGP-LS and GSGP-reg, where after 10 generations the local search step is not applied anymore.

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 former case, our generalization of local search is the worse performer among the three algorithms, with the median fitness values of GSGP and GSGP-LS lower than GSGP-reg and closer 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 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 statistical test.

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.

Table 2 shows the median and the median absolute deviation (MAD) of the fitness obtained by the best individual of the population. The MAD of a dataset $X = \{x_1, \ldots, x_n\}$ is defined as the median of the absolute deviations from the median of X. Formally, denoting by $\tilde{X}$ the median of X, $MAD(X)$ is defined as the median of $|x_i - \tilde{X}|$ for i ranging across all training samples.

Table 3 shows a statistical comparison of the three method tested. We perform the Mann-Whitney U test over the three possible couples of methods. We consider

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 2. Median and MAD of the fitness obtained by the best individual of the population for each dataset.

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

the null hypothesis that the median of the fitness obtained by applying the first method is less than the median obtained by applying the second one, while the alternative hypothesis is that the median of the fitness of the first method is greater than the median of the fitness of the second one, i.e. the first method gives worse results than the second method.

Considering a level of significance $\alpha = 0.05$ we can make the following considerations:

- GSGP is worse than GSGP-LS in 3 datasets over 5. In fact, only for %PPB and TOX datasets we are not able to refuse the null hypothesis;
- considering COMP and SLUMP datasets, we can refuse the null hypothesis that GSGP-reg does not perform better than GSGP. As a consequence, GSGP is worse than GSGP-reg for these two datasets;
- last column of the table shows that we can refuse the null hypothesis for the TOX dataset. Thus, GSGP-LS is worse than GSGP-reg for this dataset.

Table 3. Mann-Whitney U test for the comparison of the results obtained by GSGP, GSGP-LS and GSGP-reg. The alternative hypothesis considered is that the results obtained by the first method are worse than the results obtained by the second one.

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

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 GSGP and with the current best performing version of GSGP, called GSGP-LS, which is augmented with a local search operator, as defined in [**?**]. We are able in most cases to outperform classical GSGP and the 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 undertand what causes similar performances in most cases and remarkably different generalization behaviour in others.

There are multiple interesting research directions still open. First of all, an in-depth study of which families of functions $\mathcal{F}$ are the best to be employed in the regression should be performed. The regression method can also be tuned in multiple ways: for example by generating multiple regression models on different subsets of the training data and selecting 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 methods can be employed by using logistic regression instead of the traditional linear regression.