Local Search is Underused in Genetic Programming

Leonardo Trujillo, Emigdio Z-Flores, Perla S. Juárez-Smith

Tecnológico Nacional de México/I.T. Tijuana, Tijuana, B.C., Mexico

Pierrick Legrand

Universitè de Bordeaux, Institut de Mathèmatiques de Bordeaux, UMR CNRS 5251, Bordeaux, France

CQFD Team, Inria Bordeaux Sud-Ouest, Talence, France

Sara Silva

BioISI Biosystems and Integrative Sciences Institute, Faculty of Sciences, University of Lisbon, Lisbon, Portugal

Mauro Castelli, Leonardo Vanneschi

NOVA IMS, Universidade Nova de Lisboa, Lisbon, Portugal

Oliver Schütze

Computer Science Department, CINVESTAV-IPN, Mexico City, Mexico

Luis Muñoz

Tecnológico Nacional de México/I.T. Tijuana, Tijuana, B.C., Mexico

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

Trujillo, L., Z-Flores, E., Juárez-Smith, P. S., Legrand, P., Silva, S., Castelli, M., ... Muñoz, L. (2018). Local Search is Underused in Genetic Programming. In R. Riolo, B. Worzel, B. Goldman, & B. Tozier (Eds.), Genetic Programming Theory and Practice XIV (pp. 119-137). [8] (Genetic and Evolutionary Computation). Springer. https://doi.org/10.1007/978-3-319-97088-2_8

Local Search is Underused in Genetic Programming

Leonardo Trujillo, Emigdio Z-Flores, Perla S. Juárez Smith, Pierrick Legrand, Sara Silva, Mauro Castelli, Leonardo Vanneschi, Oliver Schütze and Luis Muñoz

Abstract There are two important limitations of standard tree-based genetic programming (GP). First, GP tends to evolve unnecessarily large programs, what is referred to as bloat. Second, GP uses inefficient search operators that focus on modifying program syntax. The first problem has been studied in many works, with many bloat control proposals. Regarding the second problem, one approach is to use al-

Leonardo Trujillo
Tree-Lab, Posgrado en Ciencias de la Ingeniería, Instituto Tecnológico de Tijuana, Blvd. Industrial y Av. ITR Tijuana S/N, Mesa Otay C.P. 22500, Tijuana B.C., México

Emigdio Z-Flores
Tree-Lab, Posgrado en Ciencias de la Ingeniería, Instituto Tecnológico de Tijuana, Blvd. Industrial y Av. ITR Tijuana S/N, Mesa Otay C.P. 22500, Tijuana B.C., México

Perla S. Juárez Smith
Tree-Lab, Posgrado en Ciencias de la Ingeniería, Instituto Tecnológico de Tijuana, Blvd. Industrial y Av. ITR Tijuana S/N, Mesa Otay C.P. 22500, Tijuana B.C., México

Pierrick Legrand
Universitè de Bordeaux, Institut de Mathèmatiques de Bordeaux, UMR CNRS 5251, CQFD Team, Inria Bordeaux Sud-Ouest, France

Sara Silva
BioISI Biosystems & Integrative Sciences Institute, Faculty of Sciences, University of Lisbon, Portugal.

Mauro Castelli
NOVA IMS, Universidade Nova de Lisboa, 1070-312 Lisbon, Portugal

Leonardo Vanneschi
NOVA IMS, Universidade Nova de Lisboa, 1070-312 Lisbon, Portugal

Oliver Schütze
Computer Science Department, CINVESTAV-IPN, Av. IPN 2508, Col. San Pedro Zacatenco, 07360, Mexico City, México

Luis Muñoz
Tree-Lab, Posgrado en Ciencias de la Ingeniería, Instituto Tecnológico de Tijuana, Blvd. Industrial y Av. ITR Tijuana S/N, Mesa Otay C.P. 22500, Tijuana B.C., México

ternative search operators, for instance geometric semantic operators, to improve convergence. In this work, our goal is to experimentally show that both problems can be effectively addressed by incorporating a local search optimizer as an additional search operator. Using real-world problems, we show that this rather simple strategy can improve the convergence and performance of tree-based GP, while reducing program size. Given these results, a question arises: why are local search strategies so uncommon in GP? A small survey of popular GP libraries suggests to us that local search is underused in GP systems. We conclude by outlining plausible answers for this question and highlighting future work.

Key words: Genetic Programming, Local Search, Bloat, NEAT

1.1 Introduction

Genetic programming (GP) is one of the most competitive approaches towards automatic program induction and automatic programming in artificial intelligence, machine learning and soft computing (Koza, 2010). In particular, even the earliest version of GP, proposed by Koza in the 1990's and commonly referred to as tree-based GP or standard GP[1] (Koza, 1992), continues to produce strong results in applied domains over 20 years later (Olague and Trujillo, 2011; Trujillo et al, 2012). However, while tree-based GP is supported by sound theoretical insights (Langdon and Poli, 2002; Poli and McPhee, 2003a,b, 2008), these formalisms have not allowed researchers to completely overcome some of GP's weaknesses.

In this work, we focus on two specific shortcomings of standard GP. The first drawback is bloat, the tendency of GP to evolve unnecessarily large solutions. In bloated runs the size (number of nodes) of the best solution and/or the average size of all the individuals increases even when the quality of the solutions stagnates. Bloat has been the subject of much research in GP, comprehensively surveyed in (Silva and Costa, 2009). The most successful bloat control, or size control, strategies have basically modified the manner in which fitness is assigned (Dignum and Poli, 2008; Poli and McPhee, 2008; Silva, 2011; Silva et al, 2012; Silva and Vanneschi, 2011), focusing the search towards specific regions of solution space.

A second problem of standard GP is the nature of the search operators. Subtree crossover and mutation operate on syntax, but are blind to the effect that these changes will have on the output of the programs, what is referred to as semantics (Moraglio et al, 2012). This has lead researches to use the geometric properties of semantic space (Moraglio et al, 2012) and define search operators that operate at the syntax level but have a known and bounded effect on semantics, what is known as Geometric Semantic GP (GSGP). While GSGP has achieved impressive results in several domains (Vanneschi et al, 2014), it suffers from an intrinsic shortcoming that is difficult to overstate. In particular, the sizes of the evolved solutions grow

[1] We will use the terms standard GP and tree-based GP interchangeably in this work, referring to the basic GP algorithm that relies on a tree representation and subtree genetic operators.

exponentially with the number of generations (Moraglio et al, 2012). Since program growth is not an epiphenomenon in GSGP, as it is in standard GP, it does not seem correct to call it bloat, it is just the way that the GSGP search operates. Nonetheless, this practically eliminates one of the possible advantages of GP compared to other machine learning techniques, that the evolved solutions might be amenable to human interpretation (Koza, 1992, 2010; Olague and Trujillo, 2011).

The goal of this work is twofold. First, we intend to experimentally show that the effect of these problems can be substantially mitigated, if not practically eliminated, by integrating a powerful local search (LS) algorithm as an additional search operator. Our work analyzes the effects of LS on several variants of GP, including standard GP, a bloat free GP algorithm called neat-GP (Trujillo et al, 2016), and GSGP. In all cases, we will show that LS has at least one, if not several, of the following consequences: improved convergence, improved performance and reduction in program size. Moreover, we will argue that the greedy LS strategy does not increase overfitting or computational cost, two common objections towards using such approaches in meta-heuristic optimization. The second goal of this work is to pose the following question: why are LS strategies seldom used, if at all, in GP algorithms? While we do not claim that no previous works have integrated a local optimizer into a GP algorithm, the fact remains that most works with GP do not do so, with most works on the subject presenting specific application papers. This is particularly notable when we consider how ubiquitous hybrid evolutionary-LS algorithms have become, what are commonly referred to as memetic algorithms (Chen et al, 2011; Neri et al, 2012; Lara et al, 2010). We will attempt to give plausible answers to this question, and to highlight important future research on the subject.

This chapter proceeds as follows. Section 1.2 discusses related work. Section 1.2.1 describes our proposal to apply LS in GP for symbolic regression with an experimental example. Section 1.3 shows how LS combined with a bloat-free GP can substantially reduce code growth. Afterward, Section 1.4 discusses recent works that apply LS with GSGP, improving convergence and performance in real-world domains. Based on the previous sections, Section 1.5 argues that LS strategies are underused in GP search. Finally, Section 1.6 presents our conclusions and future perspectives.

1.2 Local Search in Genetic Programming

Many works have studied how to combine evolutionary algorithms with LS (Chen et al, 2011; Neri et al, 2012). The basic idea is to include an additional operator that takes an individual as an initial point and searches for its optimal neighbor. Such a strategy can help guarantee that the local region around each individual is fully exploited. These algorithms, often called memetic algorithms, have produced impressive results in a variety of domains (Chen et al, 2011; Neri et al, 2012).

When applying a LS strategy to GP, there are basically two broad approaches to follow: (1) apply a LS on the syntax; or (2) apply it on numerical parameters of

the program. Regarding the former, (Azad and Ryan, 2014) presents an interesting recent example. The authors apply a greedy search on a randomly chosen GP node, attempting to determine the best function to use in that node among all the possibilities in the function set. To reduce computational overhead the authors apply a heuristic decision rule to decide which trees are subject to the LS, preferring smaller trees to bias the search towards smaller solutions.

Regarding the optimization of numerical parameters within the tree, the following works are of note. In (Topchy and Punch, 2001) gradient descent is used to optimize numerical for symbolic regression problems. However, the work only optimizes the value of the terminal elements (tree leaves), it does not consider parameters within internal nodes. Similarly, in (Zhang and Smart, 2004) and (Graff et al, 2013) a LS algorithm is used to optimize the value of constant terminal elements. In (Zhang and Smart, 2004) gradient descent is used and tested on classification problems, applying the LS process on every individual of the population. Another recent example is (Graff et al, 2013), where Resilient Backpropagation (RPROP) is used, in this case applying the LS operator to the best individual of each generation.

From these examples, an important question for memetic algorithms is to determine when to apply the LS. For instance, (Zhang and Smart, 2004) applies it to all the population, while (Graff et al, 2013) does so only for the best solution of each generation, and (Azad and Ryan, 2014) uses a heuristic criterion. In the case of GP for symbolic regression, this question is studied in (Z-Flores et al, 2014), concluding that the best strategies might be to apply LS to all the individuals in the population or a subset of the best individuals. However, that work focused on synthetic benchmarks and did not consider specialized heuristics (Azad and Ryan, 2014). Nonetheless, (Z-Flores et al, 2014) does show that in general, including a LS strategy improves convergence and performance, while reducing code growth.

Other works have increased the role of the local optimizer, changing the basic GP strategy. Fast Function Extraction (FFX) (McConaghy, 2011), for instance, poses the symbolic regression problem as the search for the best linear combination of candidate basis functions. Thus, FFX builds linear models, and optimizes these model using a modified version of the elastic net regression technique, eliminating the evolutionary process altogether. A similar approach can be seen in the prioritized grammar enumeration (PGE) technique (Worm and Chiu, 2013), where dynamic programming replaces the the basic search operators of traditional GP, and numerical parameters are optimized using the non-linear Levenberg-Marquardt algorithm.

1.2.1 Local Search in Symbolic Regression with Standard GP

In this work we focus on symbolic regression, however we believe that some of the conclusions might be more general. For now, this section describes our proposal to integrate a LS operator in GP in this domain, which we originally presented in (Z-Flores et al, 2014, 2015). For symbolic regression, the goal is to search for the

symbolic expression $K^O(\theta^O) : \mathbb{R}^p \rightarrow \mathbb{R}$ that best fits a particular training set $\mathbb{T} = \{(\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_n, y_n)\}$ of n input/output pairs with $\mathbf{x}_i \in \mathbb{R}^p$ and $y_i \in \mathbb{R}$, stated as

$$(K^O, \theta^O) \leftarrow \underset{K \in \mathbb{G}; \theta \in \mathbb{R}^m}{arg\,min} \; f(K(\mathbf{x}_i, \theta), y_i) \; with \; i = 1, \ldots, n \; , \tag{1.1}$$

where $\mathbb{G}$ is the solution or syntactic space defined by the primitive set $\mathbb{P}$ of functions and terminals, f is the fitness function which is based on the difference between a program's output $K(\mathbf{x}_i, \theta)$ and the desired output y_i, and θ is a particular parametrization of the symbolic expression K, assuming m real-valued parameters. This dual problem of simultaneously optimizing syntax (structure) and parametrization can be addressed following two general approaches (Lohmann, 1991; Emmerich et al, 2001). The first group is *hierarchical structure evolution* (HSE), when θ has a strong influence on fitness, and thus a LS is required at each iteration of the global (syntactic) search as a nested process. The second group is called *simultaneous structure evolution* (SSE), when θ has a marginal effect on fitness, in such cases a single evolutionary loop could simultaneously optimize both syntax and parameters. These are abstract categories, but it is reasonable to state that standard GP, for instance, falls in the SSE group. On the other hand, memetic algorithms, such as the GP version we proposed in (Z-Flores et al, 2014, 2015), fall in the HSE group.

1.2.2 Proposal

First, as suggested in (Kommenda et al, 2013), for each individual K in the population we add a small linear uppertree above the root node, such that

$$K' = \theta_2 + \theta_1(K) \; , \tag{1.2}$$

K' represents the new program output, while $\alpha_1, \alpha_2 \in \mathbb{R}$ are the first two parameters from θ. Second, for all the other nodes n_k in the tree K we add a weight coefficient $\theta_k \in \mathbb{R}$, such that each node is now defined by

$$n'_k = \theta_k n_k \; , \tag{1.3}$$

where n'_k is the new modified node, $k \in \{1, ..., r\}$ and r is the size of tree K. Notice that each node has an unique parameter that can be modified to help meet the overall optimization criteria of the non-linear expression. At the beginning of the GP run each parameter is initialized by $\theta_i = 1$. During the GP syntax search, subtrees belonging to different individuals are swapped, added or removed together with their corresponding parameters, often called Lamarckian inheritance (Z-Flores et al, 2014, 2015). We consider each tree as a non-linear expression and the local search operator must now find the best fit parameters of the model K'. The problem could be solved using a variety of techniques, but following (Z-Flores et al, 2014, 2015) we use a trust region algorithm.

Table 1.1 GP parameters

Parameter	Value
Runs	30
Population	100
Function evaluations	2'500,000
Training set	70% of complete data
Testing set	30% of complete data
Crossover operator	Standard subtree crossover, 0.9 prob.
Mutation operator	Mutation probability per node 0.05
Tree initialization	Full, max. depth 6
Function set	+,-,×,÷,exp,sin,cos,log,sqrt,tan,tanh
Terminal set	Input features, constants
Selection for reproduction	Tournament selection of size 7
Elitism	Best individual survives
Maximum tree depth	17

Finally, it is important to consider that the LS could increase the computational cost of the search, particularly when individual trees are very large. While applying the LS strategy to all trees might produce good results (Z-Flores et al, 2014, 2015), it is preferable to reduce the amount of trees to which it is applied. Therefore, we use the heuristic proposed in (Azad and Ryan, 2014), where the LS is applied stochastically based on a probability $p(s)$ determined by the tree size s and the average size of the population $\bar{s}$ (details in (Azad and Ryan, 2014; Z-Flores et al, 2015)). In this way, smaller trees are more likely to be optimized than larger trees, which reduces the computational cost and improves the convergence of the optimizer by keeping the parameter vectors relatively small. We refer to this version of GP as GP-LS.

1.2.3 Experiments and Results

We evaluate the this proposal on a real-world symbolic regression task, the Yacht problem that has 6 features and 308 input/output samples (Ortigosa et al, 2007). The experiments are carried out using a modified version of the Matlab GP toolbox GPLab (Silva and Almeida, 2005). The GP parameters used are given in Table 1.1. In what follows, we will present results based on the median performance over all runs. The fitness function used is the RMSE, and the stopping criterion is the total number of fitness function evaluations. Function evaluations are used to account for the computational cost of the trust region optimizer, which in this case is allowed to run for 100 iterations. Results are compared with a standard GP.

Figure 1.1 summarizes the main results. The convergence plots of GP and GP-LS are shown in Figure 1.1(a), showing the median training and testing performance. The figure clearly shows that GP-LS converges faster to a lower error, and at the end of the run it substantially outperforms standard GP, consistent with (Z-Flores et al, 2014, 2015). Figure 1.1(b) presents a scatter plot (each point is one individual) of all individuals generated in all runs. The individuals are plotted based on function evaluations and size. Each individual is color coded using a heat map based on test

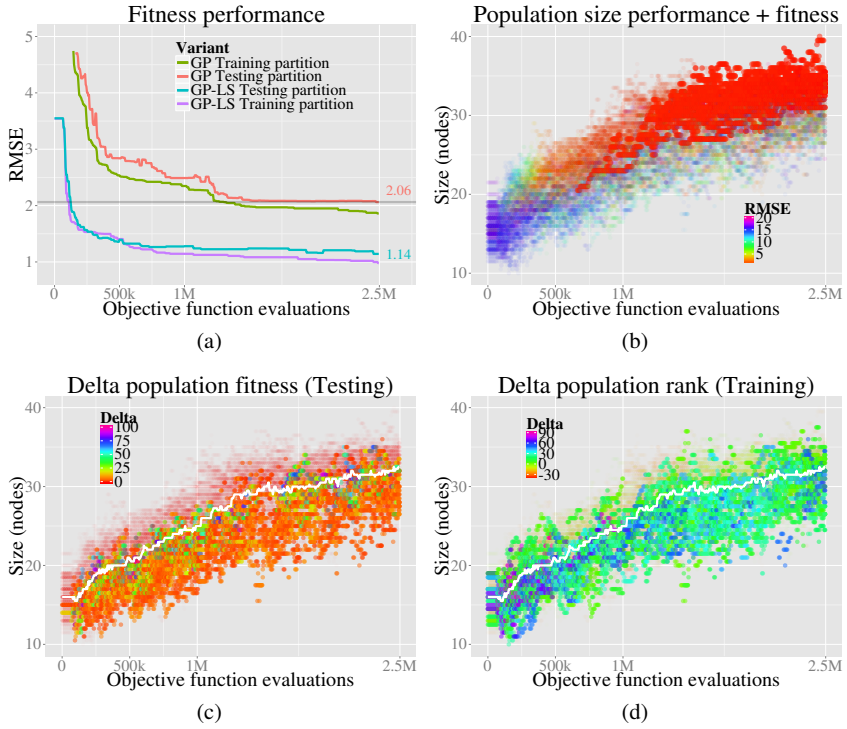

Fig. 1.1 Experimental results for GP-LS on the Yacht problem.

performance, with the best individuals (lowest error) in red. Figure 1.1(b) shows that the best performance is achieved by the largest individuals.

However, our strategy is to apply the LS on the smallest individuals of the population. This is clearly validated in Figures 1.1(c) and 1.1(d). Figure 1.1(c) shows the raw improvement of test fitness for each individual before and after the LS. A similar plot is shown in Figure 1.1(d), however instead of showing the raw improvement, this figure plots the improvement in rank within the population. In both plots the average program size is plotted with a white line, and individuals that were not passed to the local optimizer have a zero value. These plots reveal that: (1) most individuals below the average size are improved by the LS; and (b) the largest improvement is exhibited by individuals that are only slightly smaller than the average program size. While the effect on program size by the LS process will be further discussed in Section 1.3, for now it is important to state that the median of the average program size produced by standard GP on this problem is 123.576, which is substantially higher than what is shown by GP-LS.

These results present three interesting and complimentary results. First, GP-LS clearly outperforms standard GP, in terms of convergence, solution quality and average solution size. Second, the LS is clearly improving the quality of the smallest

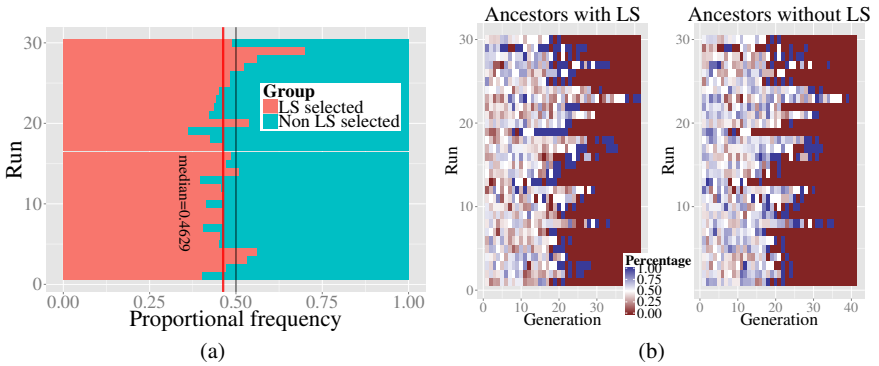

Fig. 1.2 Influence of the LS operator on the construction of the best solution found for the Yacht problem.

individuals in the population, in some cases substantially. On the other hand, and thirdly, the best solutions are still the largest trees in the population. This means that while the LS operator improves smaller solutions, the best solutions are not necessarily subjected to the LS process. This means that the LS process should be seen as an important complimentary operator. While many previous works have applied a LS process on the best solutions found, our results indicate that this is insufficient, the LS should be applied more broadly to achieve the best results.

Figure 1.2 summarizes how the LS operator influences the construction of the best solution. First, Figure 1.2(a) shows how many times the best solution in the population was chosen by the LS selection heuristic for each run. The plot indicates that the best solution was chosen about 50% of the time. Second, we track all of the ancestors of the best individual from each run, and Figure 1.2(b) plots the percentage of ancestors that were subjected to the LS. This plot also suggests that, on average, about half of the ancestors were subjected to the LS and half were not.

1.3 Bloat Control and Local search

The goal of this section is to analyze the effect that the LS has on program size. We use a recently proposed bloat-free GP algorithm called neat-GP (Trujillo et al, 2016), which is based on operator equalization (OE) family of methods (Dignum and Poli, 2008; Silva et al, 2012).

The OE approach is to control the distribution of program sizes, defining a specific shape for the distribution and then enforcing heuristic rules to fit the population to the goal distribution. Surprisingly, some of the best results are achieved by using a uniform or flat distribution; this method is called Flat-OE (Silva and Vanneschi, 2011). One of the main drawbacks of OE methods has been the difficulty of efficiently controlling the shape of the distribution without modifying the nature of the

search. Recently, neat-GP was proposed to approximate the behavior of Flat-OE in a simpler manner, exploiting well-known EC principles such as speciation, fitness sharing and elitism (Trujillo et al, 2016). As the name suggests, neat-GP is designed following the general principles of the NeuroEvolution of Augmenting Topologies (NEAT) algorithm (Stanley and Miikkulainen, 2002). While NEAT has been used in a variety of domains, its applicability for GP in general, and for bloat control in particular, was not fully exploited until recently (Trujillo et al, 2014, 2016).

The main features of neat-GP are the following. (a) The initial population contain trees of small depth (3 levels), the NEAT approach is to start with simple (small) solutions, and to progressively build complexity (increasing size) only if the problem requires it. (b) As the search progresses, the population is divided into subsets called species, such that each species contains individuals of similar size and shape; this process is called speciation, which protects innovation during the search. (c) The algorithm uses fitness sharing, such that individuals from very large species are penalized more than individuals that belong to smaller species. This allows the search to maintain an heterogeneous population of individuals based on their size, following Flat-OE. The only exception are the best individuals in each species, these are not penalized allowing the search to maintain the best solutions. (d) Crossover operations mostly take place between individuals from the same species, such that the offspring will have a very similar size and shape to their parents. For a full description of neat-GP the reader is referred to (Trujillo et al, 2016).

1.3.1 Experiments and Results

The proposal made in this work is straightforward, combine neat-GP with the GP-LS strategy; hereafter this hybrid method will be referred to as neat-GP-LS. The experimental work centers around comparing four GP variants: standard GP; neat-GP with subtree crossover; GP-LS and neat-GP-LS. Each variant was applied to the real-world problems summarized in Table1.2. This table specifies the number of input variables and the size of the data sets for the real-world problem and a description of the dataset is provided along with the appropriate reference publication. In this case, 10 runs of each algorithm were performed, suing the parameters specified in Table 1.3. For neat-GP the parameters were set according to (Trujillo et al, 2016). For all algorithms, fitness and performance are computed using the RMSE. The algorithms are implemented using the DEAP library for Python (De Rainville et al, 2012) and available for download[2].

Several changes were made to the GP-LS approach used in the previous section. First, the LS operator is applied randomly with a 0.5 probability to every individual in the population, based on the results shown in Figure 1.2. Second, the LS operator was only allowed to run for 40 iterations, to reduce the total computational cost. Third, the termination criterion was set to 100,000 function evaluations.

[2] http://www.tree-lab.org/index.php/resources-2/downloads/open-source-tools/item/145-neat-gp

Table 1.2 Real world regression problems used to compare all algorithms.

Problem	Features	Samples	Brief description
Housing(Quinlan, 1993)	14	506	Housing values in suburbs of Boston
Energy Cooling Load(Tsanas and Xifara, 2012)	8	768	Energy analysis using different building shapes simulated in Ecotectn

Table 1.3 Parameters used in all experiments.

Parameter	GP Std and GP-LS	neat-GP
Runs	10	10
Population	100	100
Function evaluations	100'000	100'000
Training set	70%	70%
Testing set	30%	30%
Operator probabilities Crossover (p_c), Mutation (p_m)	p_c=0.9, p_m=0.1	p_c=0.7,p_m=0.3
Tree initialization	Ramped Half-and-Half, with 6 levels of maximun depth	Full initialization, with 3 levels of maximun depth
Function set	+,-,x,sin,cos,log,sqrt,tan,tanh	
Terminal set	Input variables for each real world problem	
Selection for reproduction	Tournament selection of size 7	Eliminate the worst individuals of each specie
Elitism	Best individual survives	Don't penalize the best individual of each species
Maximum tree depth	17	17
Survival threshold	-	0.5
Specie threshold value	-	$h = 0.15$ with $\alpha = 0.5$
LS Optimizer probability	$P_s = 0.5$	$P_s = 0.5$

The algorithms are compared based on the following performance criteria: best training fitness, test fitness of the best solution, average size (number of nodes) of all individuals in the population, and size of the best solution. In particular we will plot and present in table form the median performance. These results will be summarized using convergence plots (performance relative to iterations).

Figure 1.3 summarize the results of the tested techniques on both problems, showing convergence plots for training and testing fitness, and the average program size, each one with respect to the number of fitness function evaluations, showing the median over 10 independent runs. Notice that in this case, performance is more or less equal for all algorithms. This might be due to the different GP implementations used (GPLab and DEAP) or to the different parametrizations of the LS strategy. Nevertheless, when we inspect program size we clearly see the benefits of the using the LS strategy. First, GP evolves substantially larger solutions for both problems, about one order of magnitude difference relative to the other methods. Second, surprisingly GP-LS is able to control code growth just as well as neat-GP. In other words, GP-LS has the same parsimonious effect on evolution as an algorithm explicitly designed for bloat control. Finally, when we combine both algorithms in neat-GP-LS, the reduction in solution size is drastically improved.

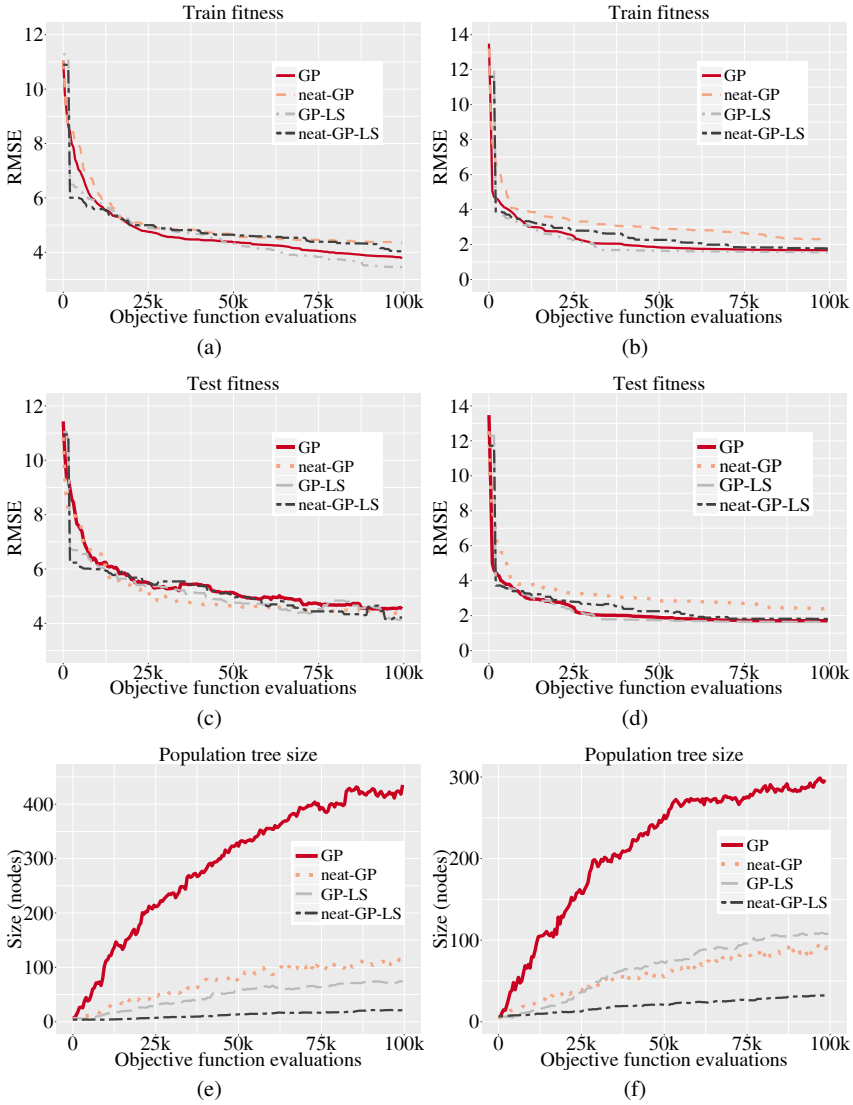

Fig. 1.3 Results for real world problems Housing (a,c,e) and Energy Cooling Load (b,d,f) plotted with respect to total function evaluations: (a, b) Fitness over train data; (c, d) Fitness over test data; and (e, f) Population size. Plots show median values over 10 independent runs.

1.4 Local Search in Geometric Semantic GP

In this section we briefly summarize our recent results of integrating a LS operator to GSGP. Our approach was originally presented in (Castelli et al, 2015f), which we briefly summarize first. In (Moraglio et al, 2012) two new genetic opera-

tors were proposed, Geometric Semantic Mutation (GSM) and Geometric Semantic Crossover (GSC). Both operators define syntax transformations, but their effect on program semantics is determined by the semantics of the parents within certain geometrical bounds. While other semantic approaches have been proposed (Vanneschi et al, 2014), the GSGP is probably the most promising given these nice properties. In (Castelli et al, 2015f), we extended the GSM operator to integrate a greedy LS optimizer, we call this operator GSM-LS.

The advantage of GSM-LS relative to GSM is that at every mutation event, the semantics of the offspring T_M is not restricted to lie within a ball around the parent T. Indeed, GSM sometimes produces offspring that are closer to the target semantics, but sometimes it does not. On the other hand, with GSM-LS the semantics of T_M is the closest we can get from the semantics of T to the target using the GSM construction, only limited by the particular random semantics of T_{R1} and T_{R2}.

The first effect of GSM-LS is that it inherently improves the convergence speed of the search process, which was experimentally confirmed in (Castelli et al, 2015f). In several test cases GSGP reaches the same performance as GSGP-LS, but requires much more iterations. This is an important difference since, as we stated above, code-growth in GSGP is an intrinsic property of applying the GSGP operators; i.e., the size of the offspring is always (substantially) larger than the size of the parents. This fact does not necessarily increase the computational cost of the search, for instance by using clever implementation that exploit the nature of the search operators (Castelli et al, 2015a). However, it does limit the possibility of extracting parsimonious solutions that might be amenable to further human analysis or interpretation. Therefore, by using GSM-LS in practice, it will be possible to reduce the number of iterations required by the algorithm to achieve the same level of performance. This means that the solutions can be vastly smaller (Castelli et al, 2015f). Moreover, real-world experimental work has shown that GSGP-LS also outperforms GSGP in overall performance on several noteworthy examples.

In (Castelli et al, 2015d), we applied GSGP-LS to the prediction of energy performance of residential buildings, predicting the heading load and cooling load for efficient power consumption. In this work, we used a hybrid algorithm, where GSM-LS is used at the beginning of the run and GSM is used during the remainder of the search, while also performing linear scaling of the input variables. Experimental results showed that the algorithms outperformed such methods as iteratively reweighted least squares and random forests. A similar application domain was addressed in (Castelli et al, 2015c), where GSGP-LS was used for energy consumption forecasting. Accurate forecasting can have many benefits for electric utilities, with errors increasing costs and reducing efficiency. In this domain, GSGP-LS outperformed GSGP and standard GP, with the former considered to be a state-of-the-art method in this domain.

Then, in (Castelli et al, 2015e) we applied GSGP-LS to predict the relative position of computerized tomography slices, an important medical application for machine learning methods. In this work, GSGP-LS was compared with GSGP, standard GP and state-of-the-art results reported on the same problem dataset. GSGP-LS outperformed all other methods, sometimes the difference was quite large, as much as

22% relative to other published results. One final example we present in Castelli et al (2015b), where GSGP-LS was used to predict the per capita violent crimes in urban areas. In this case GSGP-LS was compared with linear regression, radial basis function networks, isotonic regression, neural networks and support vector machines (SVM). The only conventional algorithm that achieved equivalent performance was SVM. These examples are meant to illustrate the benefits of integrating LS into GSGP, with clear real-world examples of the state-of-the-art performance that can be achieved.

1.5 Discussion

Based on the results presented and discussed in sections 1.2, 1.3.1 and 1.4, we make the following two major conclusions, limiting our discussion to the real-valued symbolic regression domain. First, integrating a numerical LS operator within a GP search brings about several benefits, including improving convergence, improving (or at least not reducing) performance, and substantially reducing code growth. We would stress the importance of the last point, the reduction in solution size is maybe the most important, if we consider the attention that bloat has received in GP literature and the potential of GP as a machine learning paradigm to generate human interpretable solutions. Moreover, program size is reduced even more when LS is combined with an explicit bloat control strategy. Second, the LS approach should be seen as an additional genetic operator, and not as a post-processing step. It seems that by subjecting the GP individuals to a numerical optimization process, the search is able to unlock the full potential of each individual. It is common to see that small individuals usually have a substantially lower fitness than larger ones, indeed this is understood as one of the reasons for bloat to appear (Dignum and Poli, 2008). Our results make this observation more nuanced, it is not small individuals but small individuals with sub-optimal parametrizations that will usually perform poorly. Therefore, the LS operator should be seen as a way of extracting the full potential of each GP expression, before it is kept or filtered by selection.

These conclusions seem to be supported by our experimental evidence, but we do not feel like we have hit upon an overly hidden truth or property of the GP search process. In fact, these observations seem to be relatively obvious and simple, particularly (as we have said) keeping to symbolic regression, the most common application domain for GP. Therefore, a question comes to mind: why are LS strategies so uncommon in GP systems? Take for instance some of the most popular GP platforms, including for instance lilGP (Punch and Zongker, 1998), TinyGP (Poli, 2004), DEAP (De Rainville et al, 2012), GPLab (Silva and Almeida, 2005), ECJ (White, 2012), Open Beagle (Gagne and Parizeau, 2002), Evolving Objects (Keijzer et al, 2002), JGAP (Chen et al, 2001) and HeuristicLab (Wagner and Affenzeller, 2005). None of these software systems (to the authors knowledge, and based on current descriptions on their respective websites) include an explicit mechanism for applying a memetic algorithmic framework as was discussed here, where

a greedy numerical optimizer performs parameter tuning for GP expressions. Some of these algorithms include numerical constants, and associated mutations for these constants to search for optimal values, or post-processing functions for solution simplification and/or optimization. But even these features are quite uncommon, and are not equivalent to the type of approach we describe here.

We speculate that several different reasons might be the causing this, some of these reasons are practical and some are conceptual. First, it might be that integrating this functionality might be overly complex based on specific implementation details. If this is the case, we highly recommend that future versions of these or other libraries should be made amenable to numerical LS operators. Second, it might be assumed that integrating a LS operator might make the algorithm converge to local optima or make the solutions overfit the training data. While this is a valid concern, our results, in this chapter and previous publications discussed above, suggest that this does not occur. Though we admit that further evidence should be obtained, it is reasonable to assume that a GP system should a the very least allow for a LS operator to be included. Third, it may be that some consider the LS to be a computational bottleneck, increasing the already long running time of GP. While we have not fully explored this, we find that the evidence might actually point in the opposite direction. Consider that when given the same amount of fitness function calls, GP-LS seems to outperform standard GP. If we factor in the size of the evolved solutions, it is obvious that integrating a LS operator allows GP to evaluate much smaller, and by definition, more efficient GP trees (when we do not include loops). Moreover, in the case of GSGP our results suggest that no extra cost is incurred by performing the LS process (Castelli et al, 2015f,e). Finally, we feel that it may be the case that LS might not be used because it is expected that the evolutionary process should find both the solution syntax and its optimal parametrization. While the first three reasons are practical concerns, the final one is conceptual, regarding the nature of what GP is expected to do. We believe that the design of GP algorithms should exploit all of the tools at our disposal to search for optimal models in learning problems, and that greedy LS operators, particularly those from the well-established mathematical optimization community, should be considered to be an integral part of GP search.

1.6 Conclusions and Future Work

The first major conclusion to draw from this chapter, including the experimental evidence and related literature analysis, is that integrating a numerical LS operator helps to substantially improve the performance of a GP symbolic regression, based on performance, convergence, and more notably program size. The second conclusion is that numerical LS and memetic search is seldom integrated in most GP systems. Numerical LS optimizers should be considered an important part of any GP-based search, allowing the search process to fully evaluate the usefulness of a particular GP tree before discarding it, since it very well may be that a low fitness value is due to a suboptimal parametrization of the solution.

Going forward, we identify several areas of opportunity for the GP community. The fact that memetic approaches have not been fully explored in GP literature, opens up several areas future lines of inquiry. For instance, to determine what is the best memetic strategy for GP, Lamarckian or Baldwinian, a choice that might be domain dependent. Another topic is to study the effect of using different LS optimization algorithms. Numerical optimizers are well suited for real-valued symbolic regression, but this approach might not generalize to other domains. Moreover, while we are assuming that the GP tree is always a non-linear expression, this may not always be the case. Therefore, other numerical optimization methods should be evaluated, based on the application domain.

The combination of both syntactic and numerical LS should also be the subject of future work, allowing us to fully exploit the local neighborhood around each solution. Moreover, while we believe that computational cost is not an issue relative to standard GP, it would still be advantageous to reduce any associated costs of the LS operator. One way, of course, is to use very efficient LS techniques or efficient implementations of these algorithms. Another possibility is to explore the development of surrogate models of LS optimization. The most important effect of the LS process is the change in relative rank for the individuals. It may be possible to derive predictive models that allow us to determine the expected effect that the LS process will have on a particular program.

Acknowledgements Funding for this work was provided by CONACYT Basic Science Research Project No. 178323, TecNM (México) Research Projects 5414.14-P and 5621.15-P, and by the FP7-Marie Curie-IRSES 2013 European Commission program through project ACoBSEC with contract No. 612689. Second, third and ninth author supported by CONACYT (México) scholarships No. 294213, No. 332554 and No. 302526.

References

Azad R, Ryan C (2014) A simple approach to lifetime learning in genetic programming-based symbolic regression. Evolutionary computation 22(2):287–317

Castelli M, Silva S, Vanneschi L (2015a) A c++ framework for geometric semantic genetic programming. Genetic Programming and Evolvable Machines 16(1):73–81

Castelli M, Sormani R, Trujillo L, Popovič A (2015b) Predicting per capita violent crimes in urban areas: an artificial intelligence approach. Journal of Ambient Intelligence and Humanized Computing pp 1–8

Castelli M, Trujillo L, Vanneschi L (2015c) Energy consumption forecasting using semantic-based genetic programming with local search optimizer. Intell Neuroscience 2015

Castelli M, Trujillo L, Vanneschi L, Popovič A (2015d) Prediction of energy performance of residential buildings: A genetic programming approach. Energy and

Buildings 102:67–74
Castelli M, Trujillo L, Vanneschi L, Popovič A (2015e) Prediction of relative position of CT slices using a computational intelligence system. Applied Soft Computing
Castelli M, Trujillo L, Vanneschi L, Silva S, Z-Flores E, Legrand P (2015f) Geometric semantic genetic programming with local search. In: Proceedings of the 2015 Annual Conference on Genetic and Evolutionary Computation, ACM, GECCO '15, pp 999–1006
Chen DY, Chuang TR, Tsai SC (2001) Jgap: A java-based graph algorithms platform. Softw Pract Exper 31(7):615–635
Chen X, Ong YS, Lim MH, Tan KC (2011) A multi-facet survey on memetic computation. Evolutionary Computation, IEEE Transactions on 15(5):591–607
De Rainville FM, Fortin FA, Gardner MA, Parizeau M, Gagné C (2012) Deap: A python framework for evolutionary algorithms. In: Proceedings of the 14th Annual Conference Companion on Genetic and Evolutionary Computation, ACM, GECCO '12, pp 85–92
Dignum S, Poli R (2008) Operator Equalisation and Bloat Free GP, Springer Berlin Heidelberg, chap Genetic Programming: 11th European Conference, EuroGP 2008. Proceedings, pp 110–121
Emmerich M, Grötzner M, Schütz M (2001) Design of graph-based evolutionary algorithms: A case study for chemical process networks. Evol Comput 9(3):329–354
Gagne C, Parizeau M (2002) Open beagle: A new versatile c++ framework for evolutionary computations. In: In: Late-Breaking Papers of the 2002 Genetic and Evolutionary Computation Conference (GECCO 2002, pp 161–168
Graff M, Pea R, Medina A (2013) Wind speed forecasting using genetic programming. In: 2013 IEEE Congress on Evolutionary Computation, pp 408–415
Keijzer M, Merelo JJ, Romero G, Schoenauer M (2002) Artificial Evolution: 5th International Conference, Evolution Artificielle, Springer Berlin Heidelberg, chap Evolving Objects: A General Purpose Evolutionary Computation Library, pp 231–242
Kommenda M, Kronberger G, Winkler S, Affenzeller M, Wagner S (2013) Effects of constant optimization by nonlinear least squares minimization in symbolic regression. In: Proceedings of the 15th Annual Conference Companion on Genetic and Evolutionary Computation, ACM, GECCO '13 Companion, pp 1121–1128
Koza JR (1992) Genetic Programming: On the Programming of Computers by Means of Natural Selection. MIT Press
Koza JR (2010) Human-competitive results produced by genetic programming. Genetic Programming and Evolvable Machines 11(3-4):251–284
Langdon WB, Poli R (2002) Foundations of Genetic Programming. Springer-Verlag
Lara A, Sanchez G, Coello CAC, Schütze O (2010) Hcs: A new local search strategy for memetic multiobjective evolutionary algorithms. IEEE Transactions on Evolutionary Computation 14(1):112–132

Lohmann R (1991) Application of evolution strategy in parallel populations. In: Proceedings of the 1st Workshop on Parallel Problem Solving from Nature, Springer-Verlag, pp 198–208

McConaghy T (2011) FFX: Fast, Scalable, Deterministic Symbolic Regression Technology, Springer New York, chap Genetic Programming Theory and Practice IX, pp 235–260

Moraglio A, Krawiec K, Johnson CG (2012) Geometric semantic genetic programming. In: Proceedings of the 12th international conference on Parallel Problem Solving from Nature - Volume Part I, Springer-Verlag, pp 21–31

Neri F, Cotta C, Moscato P (eds) (2012) Handbook of Memetic Algorithms, Studies in Computational Intelligence, vol 379. Springer

Olague G, Trujillo L (2011) Evolutionary-computer-assisted design of image operators that detect interest points using genetic programming. Image Vision Comput 29(7):484–498

Ortigosa I, López R, García J (2007) A neural networks approach to residuary resistance of sailing yachts prediction. Proceedings of the international conference on marine engineering MARINE 2007:250

Poli R (2004) TinyGP. see Genetic and Evolutionary Computation Conference (GECCO-2004) competition at http://cswww.essex.ac.uk/staff/sml/gecco/TinyGP.html

Poli R, McPhee NF (2003a) General schema theory for genetic programming with subtree-swapping crossover: Part i. Evol Comput 11(1):53–66

Poli R, McPhee NF (2003b) General schema theory for genetic programming with subtree-swapping crossover: Part ii. Evol Comput 11(2):169–206

Poli R, McPhee NF (2008) Parsimony pressure made easy. In: Proceedings of the 10th Annual Conference on Genetic and Evolutionary Computation, ACM, GECCO '08, pp 1267–1274

Punch B, Zongker D (1998) lil-gp 1.1. A genetic programming system. Available at http://garage.cse.msu.edu/software/lil-gp/

Quinlan JR (1993) Combining instance-based and model-based learning. Machine Learning 76:236–243

Silva S (2011) Reassembling operator equalisation: A secret revealed. In: Proceedings of the 13th Annual Conference on Genetic and Evolutionary Computation, ACM, GECCO '11, pp 1395–1402

Silva S, Almeida J (2005) Gplab-a genetic programming toolbox for matlab. In: In Proc. of the Nordic MATLAB Conference (NMC-2003), pp 273–278

Silva S, Costa E (2009) Dynamic limits for bloat control in genetic programming and a review of past and current bloat theories. Genetic Programming and Evolvable Machines 10(2):141–179

Silva S, Vanneschi L (2011) The Importance of Being Flat–Studying the Program Length Distributions of Operator Equalisation, Springer New York, chap Genetic Programming Theory and Practice IX, pp 211–233

Silva S, Dignum S, Vanneschi L (2012) Operator equalisation for bloat free genetic programming and a survey of bloat control methods. Genetic Programming and Evolvable Machines 13(2):197–238

Stanley KO, Miikkulainen R (2002) Evolving neural networks through augmenting topologies. Evolutionary Computation 10(2):99–127

Topchy A, Punch WF (2001) Faster genetic programming based on local gradient search of. In: Proceedings of the Genetic and Evolutionary Computation Conference GECCO'01, Morgan Kaufmann, pp 155–162

Trujillo L, Legrand P, Olague G, LéVy-VéHel J (2012) Evolving estimators of the pointwise hölder exponent with genetic programming. Inf Sci 209:61–79

Trujillo L, Muñoz L, Naredo E, Martínez Y (2014) NEAT, Theres No Bloat, Lecture Notes in Computer Science, vol 8599, Springer Berlin Heidelberg, pp 174–185

Trujillo L, Muñoz L, Galván-López E, Silva S (2016) neat genetic programming: Controlling bloat naturally. Information Sciences 333:21 – 43

Tsanas A, Xifara A (2012) Accurate quantitative estimation of energy performance of residential buildings using statistical machine learning tools. Energy and Buildings 49:560–567

Vanneschi L, Castelli M, Silva S (2014) A survey of semantic methods in genetic programming. Genetic Programming and Evolvable Machines 15(2):195–214

Wagner S, Affenzeller M (2005) Adaptive and Natural Computing Algorithms: Proceedings of the International Conference in Coimbra, Portugal, 2005, Springer Vienna, chap HeuristicLab: A Generic and Extensible Optimization Environment, pp 538–541

White DR (2012) Software review: the ecj toolkit. Genetic Programming and Evolvable Machines 13(1):65–67

Worm T, Chiu K (2013) Prioritized grammar enumeration: Symbolic regression by dynamic programming. In: Proceedings of the 15th Annual Conference on Genetic and Evolutionary Computation, ACM, GECCO '13, pp 1021–1028

Z-Flores E, Trujillo L, Schütze O, Legrand P (2014) Evaluating the Effects of Local Search in Genetic Programming, Springer International Publishing, chap EVOLVE - A Bridge between Probability, Set Oriented Numerics, and Evolutionary Computation V, pp 213–228

Z-Flores E, Trujillo L, Schütze O, Legrand P (2015) A local search approach to genetic programming for binary classification. In: Proceedings of the 2015 Annual Conference on Genetic and Evolutionary Computation, ACM, GECCO '15, pp 1151–1158

Zhang M, Smart W (2004) Genetic Programming with Gradient Descent Search for Multiclass Object Classification, Springer Berlin Heidelberg, chap Genetic Programming: 7th European Conference, EuroGP 2004, Coimbra, Portugal, April 5-7, 2004. Proceedings, pp 399–408