Integrating Local Search within neat-GP

Perla Juárez-Smith [*], Leonardo Trujillo [†]
Instituto Tecnológico de Tijuana
Tijuana, México

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, it uses inefficient search operators that operate at the syntax level. The first problem has been the subject of a fair amount of research over the years. Regarding the second problem, one approach is to use alternative search operators, for instance geometric semantic operators. However, another approach is to introduce greedy local search strategies, combining the syntactic search performed by standard GP with local search strategies for solution tuning, which is a simple strategy that has comparatively received much less attention. This work combines a recently proposed bloat-free GP called neat-GP with a local search strategy. One benefit of using a bloat-free GP is that it reduces the size of the parameter space confronted by the local searcher, offsetting some of the added computational cost. The algorithm is validated on a real-world problem with promising results.

Keywords

Genetic Programming, Bloat, NEAT, Local Search

1. INTRODUCTION

Genetic programming (GP) is one of the most competitive approaches towards automatic program induction and automatic programming in the fields of artificial intelligence, machine learning and soft computing [7]. 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 [6], continues to produce strong results in difficult domains over 20 years later [13]. However, while tree-based GP is supported by sound theoretical insights [8], these formalism's have not allowed researchers to overcome some of GPs most glaring weaknesses.

[*]pjuarez@tectijuana.edu.mx
[†]Corresponding author: leonardo.trujillo@tectijuana.edu.mx

GECCO'16 Companion, July 20–24, 2016, Denver, CO, USA.

ACM ISBN 978-1-4503-4323-7/16/07…$15.00.

DOI: http://dx.doi.org/10.1145/2908961.2931659

The first problem is bloat, the tendency of GP to evolve unnecessarily large solutions. In bloated GP 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 literature [11]. The most successful bloat control strategies have basically modified the manner in which fitness is assigned [3, 10], focusing the search towards specific regions of solution space.

A second issue with standard GP is related to the nature of the search operators. Subtree crossover and mutation operate at the level of program syntax, but they are blind to their effect on program output or semantics [9]. This has lead researches to exploit the geometric properties of semantic space [9] and define search operators that operate at the syntax level but have a known and bounded effect on semantics, in what is referred to Geometric Semantic GP (GSGP). While GSGP has recently achieved impressive results in several real-world domains [1] it suffers from an intrinsic shortcoming. In particular, that the size of the evolved solutions grows exponentially with the number of generations [9], which practically eliminates the possibility of interpreting the evolved solutions [6, 7]. In this work we assume that the true shortcoming of standard GP search does not lie in the nature of the search operators, but in the fact that they are used improperly. Standard GP is basically using syntactic operations to search for both the structure of a desired solution and it's optimal parametrization, whether this is explicitly stated or not [15, 16]. This has lead researchers to integrate numerical local search strategies within the basic GP process, an intuitive solution that has not received, in our opinion, sufficient literature attention [2]. Exclusively using syntax search operators at the very least contributes towards the inefficient nature of the search, as evidenced by the fact that GP algorithms that integrate a local search (LS) find shorter solutions on most problems using comparable computational effort [15, 16].

The goal of this work is to incorporate recent methodologies towards bloat control and local search optimization into a single GP methodology, otherwise based on standard GP principles. In particular, this work focuses on the symbolic regression task and the proposal builds on two recently published works. First, the neat-GP algorithm [14], which is based on the operator equalization (OE) [3] family of bloat control methods, in particular the Flat-OE algorithm [10], and the NeuroEvolution of Augmenting Topologies (NEAT) algorithm [12] that was originally proposed for the evolution of neural networks. Second, we apply the local search strategy proposed in [15, 16], which has exhibited strong performance in both symbolic regression and classification problems, we will refer to this method as GP-LS.

2. BACKGROUND

This section presents neat-GP and GP-LS.

2.1 neat-GP

Probably the most successful bloat control approaches are derived from the basic ideas of operator equalization. The OE approach is to control the distribution of program sizes at each generation, defining a specific shape for the distribution and then using heuristic rules that fit the population to the goal distribution. Surprisingly, some of the best results are achieved by using the simplest distribution, a uniform or flat distribution; this method is called Flat-OE [10]. One of the main drawbacks of OE methods has been the difficulty of efficiently controlling the shape of the distribution without severely modifying the nature of the search. Recently, a related method was proposed, this method approximates the behavior of Flat-OE in a much simpler manner, exploiting well-known EC principles such as speciation, fitness sharing and elitism [14]. The method is called neat-GP, and as its name suggests it is designed following the general principles of the NEAT algorithm [12]. While NEAT has been widely used in a variety of domains, its applicability for GP has been shown recently [14].

The main features of neat-GP are the following: (a) The initial population contain trees of small depth (3 levels), while most GP algorithms initialize the search with small and medium sized trees (depth between 3 and 6 levels), the NEAT approach is to start with simple (small) solutions, and to progressively build complexity (add 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, whereby the fitness of individuals is penalized proportionally to the size of the species to which it belongs. In this way, 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 candidate solutions for the problem. (d) Crossover operations mostly take place between individuals from the same species.

Tree dissimilarity and speciation.

Individuals are grouped together into species based on their size and shape. The goal is to maintain an heterogenous collections of program sizes. For a tree T let n_T represent the size of the tree (number of nodes) and d_T represents its depth (number of levels). Moreover, let $S_{i,j}$ represent the shared structure between both trees starting from the root node (upper region of the trees), which is also a tree. Then, the dissimilarity between two trees T_i and T_j is given by

$$\delta_T\left(T_i, T_j\right) = \beta \frac{N_{i,j} - 2n_{s_{i,j}}}{N_{i,j} - 2} + (1-\beta)\frac{D_{i,j} - 2d_{s_{i,j}}}{D_{i,j} - 2} \quad (1)$$

where $N_{i,j} = n_{T_i} + n_{T_j}$, $D_{i,j} = d_{T_i} + d_{T_j}$, and $\beta \in [0,1]$.

Using the above measure we can group individuals based on their size and depth. Each individual T_i that has not been assigned to a species is compared with each individual T_j in the current population that does belong to a species, one after another. When $\delta_T(T_i, T_j) < h$, with threshold h an algorithm parameter,then T_i is assigned to the species to which T_j belongs and no more comparisons are done. If no such T_j exists then a new species is created.

Fitness sharing is used in each species, such that individuals that belong to large species are penalized while individuals that belong to smaller ones are less so. For a minimization problem a simple penalization is:

$$f^{'}(T_i) = |S_u|\, f(T_i) \quad (2)$$

where $f(T_i)$ is the raw fitness of the tree, $f'(T_i)$ is the penalized or adjusted fitness, S_u is the species to which T_i belongs, and $|S_u|$ is the number of individuals in species S_u. The best individual of each species will be an exception, the fitness of this solution wont be penalized.

Parent selection uses an elitist strategy, such that the $p_{worst}\%$ individuals of each species are deleted, and the same number of offspring are generated to replace them. A maximum number of possible offspring for each individual is set proportionally to its raw fitness; the adjusted fitness is not used because some individuals were not penalized, if the adjusted fitness was used then the population would loose diversity. Afterward, individuals are ordered based on their adjusted fitness, and the algorithm iterates through the remaining population choosing parents. A random equiprobable decision is made each time, of either selecting the best individual or a random individual. Each time an individual is selected as parent its expected number of descendants is reduced accordingly, and when this value reaches zero the individual cannot be used again as a parent.

Two types of search operators are used, crossover and mutation, which are chosen randomly based on the probabilities p_m and p_c. For mutation, standard subtree mutation is used and the number of expected offspring is reduced by one. For crossover, after the first parent T_1 of species S is chosen the second parent T_2 is chosen based on the following heuristic: the best solution in S with $T_1 \neq T_2$; or a random individual if S does not contain other trees.

2.2 Genetic Programming with Local Search

The most common application of GP is to solve what are known as symbolic regression problems, where a model that best fits a dataset is sought. Unlike other forms of regression, the form of the model is not defined a priori, as is done in linear regression for example. The goal is to search for the symbolic expression $K^O : \mathbb{R}^p \rightarrow \mathbb{R}$ 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 $y_i \in \mathbb{R}$, states as

$$(K^O, \boldsymbol{\theta}^{\boldsymbol{O}}) \leftarrow \underset{K\in\mathbb{G};\boldsymbol{\theta}\in\mathbb{R}^m}{arg\ min}\ f(K(\mathbf{x}_i, \boldsymbol{\theta}), y_i)\ with\ i = 1, \ldots, p\ , \quad (3)$$

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 $K(\mathbf{x}_i, \boldsymbol{\theta})$ and the desired output y_i, and $\boldsymbol{\theta}$ is a particular parametrization of the symbolic expression K, assuming m real-valued parameters. In standard GP, only the symbolic expression is constructed and modified by the search process, while the underlying parametrization $\boldsymbol{\theta}$ is not considered. Recent works have shown that concurrently optimizing both the structure and parametrization of the evolved models can speed up convergence and improve performance. In particular we adopt the approach proposed in [15, 16] which has been applied to both symbolic regression and classification problems.

First, as suggested in [5], for each individual K in the population we add a small linear upper-tree above the root node, such that

$$K' = \theta_2 + \theta_1(K) \ , \tag{4}$$

K' represents the new program output, while θ_1 and θ_2 are the first two parameters from $\boldsymbol{\theta}$. 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:

$$n'_k = \theta_k n_k \ , \tag{5}$$

where n'_k is the new modified node, $k \in \{1, ..., r\}$, $r = |Q|$ and Q is the tree representation. 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 (following the standard crossover/mutation rules) together with its corresponding parameters, without affecting their values. This follows a memetic search process with Lamarckian inheritance [15, 16]. Therefore, 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 can be solved using a variety of techniques, but following [15, 16] we employ a trust region algorithm.

Finally, the LS mechanism could be applied in different ways, in this work this mechanism is applied with a random selection using a probability of $P_s = 0.5$ for each individual.

3. PROPOSAL

The proposal in this work is straightforward, combine neat-GP with the GP-LS strategy; hereafter this hybrid will be referred to as neat-GP-LS. This is done quite easily in fact, Figure 1 presents a flow diagram depicting each of the main stages in the neat-GP-LS process, combining the main elements of both algorithms described in Section 2, proceeding as follows: (a) the algorithm starts with a randomly generated population P of small full trees; (b) speciation is performed on the entire population P based on the size and depth of each solution; (c) each individual is then parameterized and (d) extended by the linear upper tree; (e) afterward, fitness evaluation is performed and (f) fitness sharing is applied, penalizing individuals from larger species; (g) if the stopping criterion is not met then (h) the set of parents Q is selected and (i) a set R of offspring are generated using subtree mutation, subtree crossover or neat-crossover; (j) the offspring are combined with the current population $\{P \cup R\}$ and speciation is applied to this set; (k) the fitness of the offspring in R is computed and (l) fitness sharing applied to them; (m) the best offspring replace the worst p_{worst}% individuals in P; (n) individuals are selected for local search; and (o) the local search is applied to them.

When genetic operators are applied care must be taken to consider the inheritance of parameter values, the proposal is a Lamarckian scheme. In case of subtree mutation, all the parameters in the new subtree are initialized to unity.

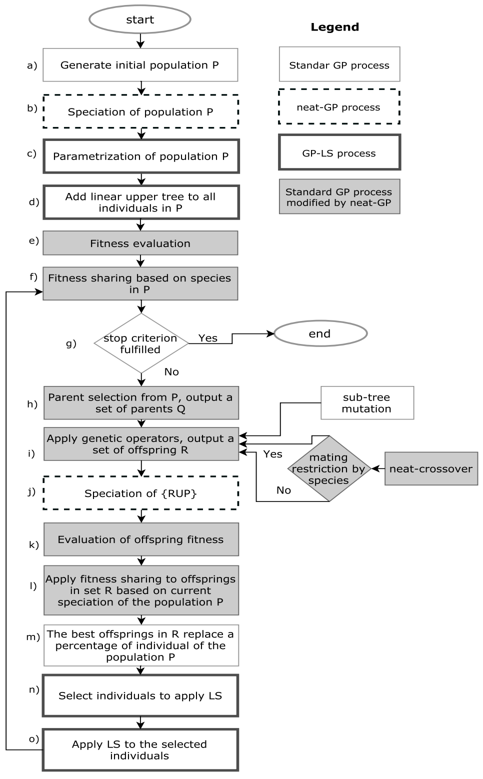
start
Legend
a) Generate initial population P
Standar GP process
b) Speciation of population P
neat-GP process
c) Parametrization of population P
GP-LS process
d) Add linear upper tree to all individuals in P
Standard GP process modified by neat-GP
e) Fitness evaluation
f) Fitness sharing based on species in P
g) stop criterion fulfilled
Yes
end
No
h) Parent selection from P, output a set of parents Q
sub-tree mutation
i) Apply genetic operators, output a set of offspring R
Yes
mating restriction by species
neat-crossover
No
j) Speciation of {RUP}
k) Evaluation of offspring fitness
l) Apply fitness sharing to offsprings in set R based on current speciation of the population P
m) The best offsprings in R replace a percentage of individual of the population P
n) Select individuals to apply LS
o) Apply LS to the selected individuals

Figure 1: General flow diagram of the neat-GP-LS algorithm.

For crossover, the swapped nodes and subtrees maintain the parameter values specified in the parent tree. In this case the upper linear trees are not considered during crossover or mutation.

4. EXPERIMENTS AND RESULTS

The experimental work centers around comparing these variants: standard GP; neat-GP with standard-crossover; GP-LS and neat-GP-LS. These variants are validated on the Boston Housing real world problem that concerns housing values in suburbs; it consists of 506 cases having 13 input variables. The aim is to predict the median house price given a number of demographic features.

In this case, 30 runs of each algorithm was performed with the parameters specified in Table 1. In the neat-GP variants the parameters were set according to [14]. The termination criterion is given by a maximum number of function evaluations (calls to the fitness function), considering that we cannot compare the algorithms based on generations since this would not account for the computational effort required to apply the LS heuristic. For all algorithms, fitness and performance are computed using RMSE.

The algorithms are compared based on the following performance criteria: best training fitness, test fitness of the best solution and average size (number of nodes) of all individuals in the population. Figure 2 presents the plot of the median performance over all runs, relative to the total function evaluations. Also presents rank statistics for each method over all runs using box plots.

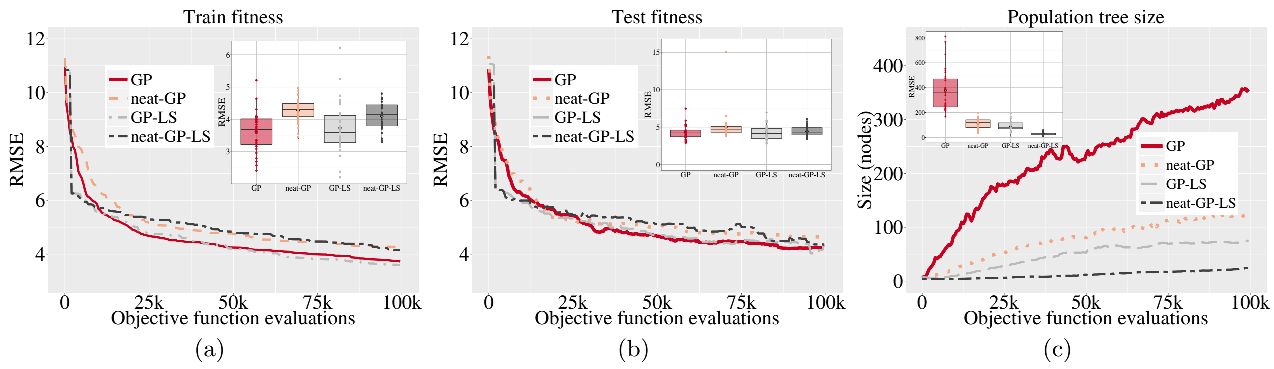

Figure 2: Results for Housing problem plotted with respect to total function evaluation: (a) train fitness; (b) test fitness; (c) average population size. Plots show median values over 30 independent runs. Box plots represent the performance distribution of each measure at the end of the 30 runs.

Table 1: Parameters used in the reported experimental work.

Parameter	Value GP Std and GP-LS	Value neat-GP
Runs	30	30
Population	100	100
Function evaluations	100’000	100’000
Training set	70%	
Testing set	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 real world problems.	
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	
Survival threshold		0.5
Specie threshold value		$h = 0.15$ with $\alpha = 0.5$
LS Optimizer probability	$P_s = 0.5$	$P_s = 0.5$

5. CONCLUSIONS

The exploration advantage provided by neat-GP through speciation provides good diversity of solutions and GP-LS helps to exploit these areas. This contrasts with approaches based on GSGP [9], that compromise possible interpretability to improve of performance.

Future work will explore other parametrization schemes and heuristics to apply the LS operator, but we will focus on reducing computational effort using the EvoSpace Model [4]. It is a framework for developing EAs that works with modules called EvoWorkers, in each EvoWorker there’s a partial evolutionary process which takes the initial population from a repository of individuals called EvoStore. neat-GP-LS organizes the population into species, allowing us to distribute the neat-GP-LS search using EvoWorkers for each species.

Acknowledgments

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

6. REFERENCES

[1] M. Castelli, L. Trujillo, L. Vanneschi, and A. Popovič. Prediction of energy performance of residential buildings: A genetic programming approach. *Energy and Buildings*, 102:67 – 74, 2015.

[2] X. Chen, Y.-S. Ong, M.-H. Lim, and K. C. Tan. A multi-facet survey on memetic computation. *Evolutionary Computation, IEEE Transactions on*, 15(5):591–607, Oct 2011.

[3] S. Dignum and R. Poli. *Genetic Programming: 11th European Conference, EuroGP 2008*, chapter Operator Equalisation and Bloat Free GP, pages 110–121. Springer Berlin Heidelberg, Berlin, Heidelberg, 2008.

[4] M. García-Valdez, L. Trujillo, J.-J. Merelo, F. Fernández de Vega, and G. Olague. The evospace model for pool-based evolutionary algorithms. *Journal of Grid Computing*, 13(3):329–349, 2014.

[5] M. Kommenda, G. Kronberger, S. M. Winkler, M. Affenzeller, and S. Wagner. Effects of constant optimization by nonlinear least squares minimization in symbolic regression. In *Genetic and Evolutionary Computation Conference, GECCO ’13, Amsterdam, The Netherlands, July 6-10, 2013, Companion Material Proceedings*, pages 1121–1128, 2013.

[6] J. R. Koza. *Genetic Programming: On the Programming of Computers by Means of Natural Selection.* MIT Press, Cambridge, MA, USA, 1992.

[7] J. R. Koza. Human-competitive results produced by genetic programming. *Genetic Programming and Evolvable Machines*, 11(3-4):251–284, Sept. 2010.

[8] P. R. Langdon W.B. *Foundations of Genetic Programming.* Springer-Verlag Berlin Heidelberg, 2002.

[9] A. Moraglio, K. Krawiec, and C. G. Johnson. Geometric semantic genetic programming. In *Proceedings of the 12th international conference on Parallel Problem Solving from Nature - Volume Part I*, PPSN’12, pages 21–31, Berlin, Heidelberg, 2012. Springer-Verlag.

[10] S. Silva. Reassembling operator equalisation: A secret revealed. In *Proceedings of the 13th Annual Conference on Genetic and Evolutionary Computation*, GECCO ’11, pages 1395–1402, New York, NY, USA, 2011. ACM.

[11] S. Silva and E. Costa. 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, 2009.

[12] K. O. Stanley and R. Miikkulainen. Evolving neural networks through augmenting topologies. *Evolutionary Computation*, 10(2):99–127, 2002.

[13] L. Trujillo, P. Legrand, G. Olague, and J. LéVy-VéHel. Evolving estimators of the pointwise hölder exponent with genetic programming. *Inf. Sci.*, 209:61–79, Nov. 2012.

[14] L. Trujillo, L. Muñoz, E. Galván-López, and S. Silva. neat genetic programming: Controlling bloat naturally. *Information Sciences*, 333:21 – 43, 2016.

[15] E. Z-Flores, L. Trujillo, O. Schütze, and P. Legrand. *EVOLVE - A Bridge between Probability, Set Oriented Numerics, and Evolutionary Computation V*, chapter Evaluating the Effects of Local Search in Genetic Programming, pages 213–228. Springer International Publishing, Cham, 2014.

[16] E. Z-Flores, L. Trujillo, O. Schütze, and P. Legrand. A local search approach to genetic programming for binary classification. In *Proceedings of the 2015 Annual Conference on Genetic and Evolutionary Computation*, GECCO ’15, pages 1151–1158, New York, NY, USA, 2015. ACM.