Soft Computing (2020) 24:14757–14770
https://doi.org/10.1007/s00500-020-04829-4

METHODOLOGIES AND APPLICATION

On the analysis of hyper-parameter space for a genetic programming system with iterated F-Race

Leonardo Trujillo[1] · Ernesto Álvarez González[1] · Edgar Galván[2] · Juan J. Tapia[3] · Antonin Ponsich[4]

Published online: 17 March 2020

Abstract
Evolutionary algorithms (EAs) have been with us for several decades and are highly popular given that they have proved competitive in the face of challenging problems' features such as deceptiveness, multiple local optima, among other characteristics. However, it is necessary to define multiple hyper-parameter values to have a working EA, which is a drawback for many practitioners. In the case of genetic programming (GP), an EA for the evolution of models and programs, hyper-parameter optimization has been extensively studied only recently. This work builds on recent findings and explores the hyper-parameter space of a specific GP system called neat-GP that controls model size. This is conducted using two large sets of symbolic regression benchmark problems to evaluate system performance, while hyper-parameter optimization is carried out using three variants of the iterated F-Race algorithm, for the first time applied to GP. From all the automatic parametrizations produced by optimization process, several findings are drawn. Automatic parametrizations do not outperform the manual configuration in many cases, and overall, the differences are not substantial in terms of testing error. Moreover, finding parametrizations that produce highly accurate models that are also compact is not trivially done, at least if the hyper-parameter optimization process (F-Race) is only guided by predictive error. This work is intended to foster more research and scrutiny of hyper-parameters in EAs, in general, and GP, in particular.

Keywords Hyper-parameter optimization · Iterated F-Race · Genetic programming

1 Introduction

Hyper-parameters are configuration variables that regulate or control the behavior of an optimization or learning algorithm (Birattari 2009; Sipper et al. 2018). For evolutionary algorithms (EAs), hyper-parameters[1] include population size, mating rates, number of generations and others. In all meta-heuristic search techniques, the proper setting of hyper-parameter values can be a difficult and tedious endeavor (Birattari 2009). This is also true for EAs that often lack a theoretical foundation to derive their optimal parametrization values analytically, with few exceptions (Hansen and Ostermeier 2001). This work focuses on hyper-parameter tuning or optimization (Birattari 2009; Neumüller et al. 2012), also referred to as meta-optimization, which is performed off-line, in contrast to hyper-parameter control where an EA, or

[1] They are also referred to as parameters, but the distinction between parameters and hyper-parameters is important, particularly when the EA is performing a learning process, searching for models that might also include their own parameters.

Communicated by V. Loia.

✉ Leonardo Trujillo
leonardo.trujillo@tectijuana.edu.mx

Ernesto Álvarez González
emmalv365@gmail.com

Edgar Galván
edgar.galvan@mu.ie

Juan J. Tapia
jtapiaa@ipn.mx

Antonin Ponsich
aspo@azc.uam.mx

[1] Leonardo Trujillo · Ernesto Álvarez González
Tecnológico Nacional de México/IT de Tijuana, Tijuana, BC, Mexico

[2] Department of Computer Science, Maynooth University, Maynooth, Ireland

[3] Instituto Politécnico Nacional - CITEDI, Av. Instituto Politécnico Nacional No. 1310 Colonia Nueva Tijuana, C.P. 22435 Tijuana, BC, Mexico

[4] UAM (Azcapotzalco) - Universidad Autónoma Metropolitana, Av. San Pablo No. 180, Col. Reynosa Tamaulipas, C.P. 02200 Reynosa, CDMX, Mexico

any other meta-heuristic algorithm, dynamically modifies the hyper-parameter values during online execution (Karafotias et al. 2015).

This topic has been studied extensively for EAs, but has received less attention in genetic programming (GP) (Koza 1992; Langdon and Poli 2010). In GP, an EA is used to search for models, functions or small programs that perform a particular computational process, usually following a supervised learning problem formulation, such as regression (Vanneschi et al. 2019) or automatic improvement of code (López-López et al. 2018). Probably, the most extensive experimental study regarding hyper-parameter optimization for GP was recently published by Sipper et al. (2018). These authors also present an up to date survey on this topic.

Sipper et al. (2018) evaluate two different GP systems using several standard GP benchmark problems and supervised learning on classification benchmark problems. Two hyper-parameter optimization procedures are tested: an EA (meta-evolution) and a random search. They consider common hyper-parameters for EAs and GP, such as population size, number of generations, crossover rate, mutation rate and tournament size. The authors performed an extensive number of GP runs to gather their results, allowing them to draw interesting insights and suggestions. Surprisingly, random search seems to be as good as an EA for this task, and at least two conclusions reported by the authors are noteworthy and useful for practitioners and applied researchers. First, it seems that *good* parametrizations (specific combination of hyper-parameter values) can be found all over hyper-parameter space; i.e., good parameter settings can be quite diverse, and they are not clustered in specific regions of hyper-parameter space. Second, finding general hyper-parameter values (that are applicable to several different problems) is more difficult compared to finding good parameter values for a single problem instance, since the latter can be efficiently done through random search.

It is fair to say, however, that there are some aspects that were not considered by Sipper et al. (2018). First, the authors do not account for the fact that most problems that are solved with GP are supervised learning problems. In such cases, fitness is proportional to training performance, which is the measure used to evaluate different hyper-parameter configurations. However, of more interest to practitioners is performance on an unseen test set of fitness cases. Test performance is not considered (Sipper et al. 2018). Second, the study focuses on very general hyper-parameters, and it does not consider system-specific parameters, even for the more sophisticated GP system studied (Cava et al. 2019). Third, the work does not explore the effect that different parametrizations have on solution size, an important aspect in most GP-based systems (Trujillo et al. 2016). Finally, there are other state-of-the-art methods for hyper-parameter optimization, namely racing algorithms (Birattari et al. 2002; López-Ibáñez et al. 2016), that can be used for this task besides a basic EA and random search, particularly since racing algorithms have been widely used for hyper-parameter optimization of search and optimization algorithms.

Therefore, this paper aims to address the aforementioned limitations, with the following contributions:.

1. Hyper-parameter space is explored for a bloat-free GP system called neat-GP (Trujillo et al. 2016). Two common EA hyper-parameters are considered (crossover rate and mutation rate), as well as three system-specific hyper-parameters of neat-GP.
2. Three variants of the iterated F-Race algorithm are evaluated, using two extensive sets of 15 *easy* and 16 *hard* benchmark problems.
3. Results are evaluated considering testing performance, solution size and best parameter values found, all relative to the manual configuration suggested in Trujillo et al. (2016).
4. Results are contrasted with those reported in Sipper et al. (2018), showing agreement in some results while diverging in other relevant aspects.

The remainder of this paper is organized as follows. Section 2 provides a brief overview of required background, discussing the general principles of hyper-parameter optimization and the basics of racing algorithms. Section 3 presents a summary of the neat-GP system and discusses the hyper-parameters considered in this study. Afterward, Sect. 4 presents the experimental work and detailed results. Section 5 contains a discussion of the results. Concluding comments are presented in Sect. 6.

2 Background

The goal in this section is to provide a brief introduction to the problem of hyper-parameter optimization and to the racing algorithms used in this work, namely the iterated F-Race algorithm. For a more thorough account of these subjects, the interested reader is referred to (Birattari 2009; Neumüller et al. 2012; Sipper et al. 2018) for the former and (Birattari et al. 2002; López-Ibáñez et al. 2016) for the latter.

2.1 Hyper-parameter optimization

The process of hyper-parameter optimization is also known as meta-optimization, automatic parameter tuning and, in the case of an EA, meta-evolution (Sipper et al. 2018). It consists in using an optimization algorithm to automatically tune the hyper-parameters of another algorithm, such that the expected performance on a given (set of) problem(s) can be maximized (or minimized as the case may be). In other

words, an optimization algorithm, which can be referred to as the meta-optimizer *MO*, searches for an optimal set (or vector) of hyper-parameter values θ that represent a particular parametrization for an optimization or learning algorithm A, such that A achieves the best possible performance on a set of problems $\{P_i\}$, with $i = 1, \ldots, n$.

To perform hyper-parameter optimization, the following must be defined:

1. The algorithm A to be tuned.
2. The hyper-parameters of A that are considered in θ and their valid ranges.
3. The meta-optimizer *MO* to be used.
4. The set(s) of problems $\{P_i\}$ that are used to evaluate the quality of different parametrizations θ; these can be thought of as meta-fitness cases in a GP sense.

Some comments of how previous works relate to the four points mentioned above are relevant. Hyper-parameter optimization is now widely known, but often not used due to the associated computational cost. It has been applied to a wide variety of optimization algorithms, meta-heuristic search techniques and machine learning algorithms. However, one class of algorithms has received relatively lesser attention, namely GP. Probably the most notable study concerning GP is Sipper et al. (2018), as discussed above.

Regarding the hyper-parameters to be optimized, these can be numerical, ordinal or categorical parameters. In the case of an EA or GP, numerical parameters can be population size or number of total generations, categorical parameters can refer to the type of crossover that is used, while ordinal parameters are similar to categorical, but with an implicit ordering. This work focuses on numerical hyper-parameters, as done in Sipper et al. (2018).

With regard to the type of meta-optimizer, there is a variety of methods in the current literature, most of which are gradient-free heuristics or meta-heuristics, experimental design techniques or statistical modeling approaches. Probably the simplest approach is to perform a random search (Bergstra and Bengio 2012) or a grid search (Olson et al. 2017), which are easy to implement, tend to achieve good performance improvements compared to manual tuning and are widely used.

Another common strategy is to use EAs, as in Sipper et al. (2018). However, one class of algorithms that have proved to be particularly successful are racing approaches (Birattari et al. 2002; López-Ibáñez et al. 2016), since they deal with one of the main issues in hyper-parameter optimization, which is the computational cost. Performing hyper-parameter optimization is often avoided because it requires multiple executions (using different parameter values) of the algorithm to be tuned. Moreover, meta-heuristics are often stochastic search techniques, meaning that their performance level for solving one problem instance has to be evaluated in terms of some statistical measure (for instance, mean or median value) computed over several executions of the algorithm. This increases the computational burden of performing hyper-parameter optimization. This is particularly problematic for EAs, in general, and GP, in particular, which are notoriously plagued by long run times. Racing algorithms perform a *race* between different candidate parametrizations, and discard those parametrizations that show the weakest performance over a subset of the problems used for evaluation; these methods are further discussed in the following subsection.

Finally, racing algorithms and, more precisely, iterated versions of the algorithm, have not been previously used to tune a GP system. This paper presents the first extensive study that considers this combination, using a diverse set of widely accepted benchmarks for symbolic regression using GP (McDermott et al. 2012).

2.2 F-Race and iterated F-Race

In racing algorithms for hyper-parameter optimization, a set of candidate parametrizations $\{\theta_j\}$, with $j = 1, \ldots, m$, are generated using a given distribution, with $\theta_j = [x_1, \ldots x_p]$, x_u being the *uth* hyper-parameter and p the total number of hyper-parameters. For instance, if we define X as the space of valid parametrizations considered in a given study, then $\{\theta_j\}$ can be constructed by uniform random sampling of X, using an independent distribution for each hyper-parameter. Then, each parametrization is sequentially evaluated on each problem from set $\{P_i\}$. For each parametrization θ_j, a performance or cost vector is constructed of the form $C(\theta_j, \{P_i\}) = [C_j(P_1), C_j(P_2), \ldots]$, where $C_j(P_i)$ represents the performance of parametrization θ_j applied to problem P_i. After a certain number of problems $1 \leq s \leq n$ have been evaluated, and after each subsequently tested problem, the parametrizations are compared according to their accumulated performance vector $[C_j(P_1), \ldots, C_j(P_s)]$.

Depending on the racing algorithm used, a specific criterion is applied to eliminate the worst performing parametrizations based on their accumulated performance vectors. In the case of F-Race, the criterion for removal from the race is based on Friedman's nonparametric two-way analysis of variance by ranks. In particular, if a parametrization θ_j is performing statistically worse than at least one other parametrization in the race, then θ_j is deleted. The parameter s is the minimum number of problems that have to be evaluated such that the Friedman test has sufficient evidence within the corresponding performance vectors to determine whether a statistically significant difference exists between parametrizations and is normally set to 3. At the end of the run, F-Race returns the subset $\Theta^O \subset \{\theta_j\}$ of parametrizations that contains all of the statistically supe-

rior and equivalent parametrizations, also referred to as elite parametrizations.

A common extension to the F-Race algorithm consists in performing the racing process multiple times sequentially, which is referred to as the iterated F-Race or iF-Race (López-Ibáñez et al. 2016). The algorithm returns the set $\{\bigcup \Theta_l^O\}$ with $l = 1, \ldots, s$ and Θ_l^O being the optimal set of parametrizations returned after iteration l, with the process executed a total of s iterations. The iF-Race algorithm is a direct extension of the F-Race algorithm, where the crucial step is in how the set $\{\theta_j\}$ is generated at the beginning of each iteration. In particular, three variants are considered in the present paper, following (López-Ibáñez et al. 2016):

Original iF-Race. This variant creates a new set $\{\theta_j\}$ for each iteration of iF-Race using a uniform distribution over X, executing several independent F-Race executions. A useful feature is that each F-Race execution can be run in parallel. However, there is an obvious loss of information between runs that might be useful to make the racing process more efficient. This observation justifies the strategies in the following two variants.

iF-Race with Elitism. In this case, the set Θ_l^O is used to seed the set $\{\theta_j\}$ for iteration $l + 1$. If $|\Theta_l^O| < m$, then the remaining $m - |\Theta_l^O|$ parametrizations are randomly generated using a uniform distribution like in the Original iF-Race.

iF-Race with Gaussian update. This strategy is based on (López-Ibáñez et al. 2016), with minor changes to simplify the process. As with iF-Race with Elitism, the set Θ_l^O is used to seed the set $\{\theta_j\}$ for iteration $l + 1$. However, new parametrizations are generated as perturbations from the parametrizations returned in Θ_l^O. First, with equal probability for each elite configuration, a single elite parametrization θ_e is chosen from Θ_l^O. A new parametrization is generated by sampling a normal distribution for each hyper-parameter x_u, using the value of x_u in θ_e as the mean and setting the standard deviation $\sigma_u^l = (x_{\max} - x_{\min})/2$, and decreasing this value for each subsequent iteration l by $\sigma_u^l = \sigma_u^{l-1}(m - |\Theta_{l-1}^O|)^{(-1/p)}$ with p the total number of hyper-parameters to be tuned.

The aforementioned three variants of iF-Race are used in this work, which we refer to as Original, Elitism and Gaussian, respectively. If we consider iF-Race as a search process, then the methods range from an explorative process (Original), to a more greedy exploitative approach (Gaussian). In the first one, the process repeats the F-Race algorithm for s iterations. The Elitism strategy is a more greedy approach since the set of parametrizations used by F-Race at each iteration is seeded by the best parametrizations found in previous iterations. This allows the algorithm to be more efficient, by quickly discarding bad parametrizations as the iF-Race process progresses. The Gaussian approach increases the level of exploitation in the search, by using the best parametrizations from previous iterations and by generating new parametrizations as *tweaked* versions of these elite parametrizations, similar to what is done in a random walk with Gaussian perturbation.

The former observations regarding exploration/intensification behaviors provide some hints with respect to the convergence rates of the three tackled variants. Clearly, the highly greedy aspect of the Gaussian version is likely to involve higher (i.e., faster) convergence rates than the two other variants, although presenting the risk of premature convergence toward the first good configurations found and reducing the diversity of the candidate configurations produced. Conversely, the original version should be the slowest to achieve convergence toward a set of statistically satisfactory configurations, since a wider exploration of the hyper-parameter space is allowed.

As reported by López-Ibáñez et al. (2016), iterated racing procedures such as iF-Race present some drawbacks, the first one being the unavoidable computational burden due to the intensive experiments performed on many distinct instances. On the other hand, iF-Race processes also have a certain number of parameters to be tuned, which may have some influence on the final configurations obtained. Finally, an undesired behavior has been observed in some practical cases, for which the order of instances chosen for each race (iteration) can bias the results. For instance, a good quality configuration might be prematurely discarded after the first round, while it might provide good solutions for other instance classes. Despite the above-mentioned cons, the recognized pros of iF-Race make it one of the most relevant options for meta-optimization. First, because the selection of the best configurations produced is based on specific statistical tests (which is generally not done with manual tuning). Further, the experimental working mode allows not to test all configurations for all instances, since unsuitable configurations are progressively discarded during the iterated racing process. Also, parallelization is considered in most practical implementations, so that the final computational burden can be mitigated w.r.t. manual configuration. Finally, the automation and randomization of the whole experimental process allow for determining robust configuration's without any human effort.

3 neat-GP: Bloat-free genetic programming

A standard or canonical GP system, as proposed by Koza (Koza 1992; Langdon and Poli 2010), uses a variable-size tree-based representation for candidate solutions, which are syntax trees that encode a particular model or more generally an individual program that is a solution candidate. Trees are

composed by elements from a finite set of terminal elements (terminal set) that can be used as leaves in individual trees and that represent problem inputs, while internal nodes contain elements from a finite set of functions (function set) that represent operators that can be used by the GP system to construct individual models.

Bloat is the tendency of GP to generate unnecessarily large solutions (large syntax trees), even when larger solutions do not provide a proportional improvement in fitness. In general, bloat is seen as a problem that needs to be overcome in most GP systems, since it produces solutions that are more complex and difficult to interpret, while also slowing down the search process since larger solutions require more computational resources to be evaluated, without necessarily producing an improvement over time. In general, neat-GP is a bloat control GP system that is based on the neuroevolution of augmenting topologies algorithm (NEAT) (Stanley and Miikkulainen 2002), achieving strong results in regression, classification and computer vision problems (Trujillo et al. 2016; Hernández-Beltran et al. 2019).

The main features of neat-GP are the following: The initial population only contains shallow GP trees (three levels, with the root at level 1), while most GP algorithms initialize the search with small and medium sized trees (depth between three and six levels). The NEAT approach is to start with simple (small) solutions and to progressively build complexity (add size) as the search progresses. The population is divided into species, such that each species contains individuals of similar size and shape; this process is called speciation, which protects innovation during search. The algorithm uses fitness sharing, whereby the fitness of individuals is penalized proportionally to the size of the species to which it belongs. This allows the search to maintain an heterogeneous population of individuals based on their size. The only exceptions are the best individuals from each species, and these are not penalized to allow the search to maintain the best candidate solutions for the problem. This is the last element of neat-GP, a strong selection pressure that uses elitism frequently.

3.1 Speciation, tree dissimilarity and fitness sharing

In neat-GP, individuals are grouped together into species based on their size and shape. For a tree T, let n_T represent the size of the tree (number of nodes) and d_T represent its depth (number of levels). Moreover, let $S_{i,j}$ represent the shared structure between two trees T_i and T_j, 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 $\beta \in [0, 1]$.

This measure is used to group similar individuals into species. Note that this measure is not a metric, but non-metric dissimilarity functions can be used to create groups or clusters efficiently (Ackermann et al. 2008). Each time a new individual T_i is generated, the individual is compared with a randomly chosen individual T_j from each species, and this is done sequentially using a random shuffling of all species. If $\delta_T(T_i, T_j) < h$, with threshold h being an algorithm parameter, then T_i is grouped into the same species as T_j and no other comparisons are made. If T_i is not assigned to an existing species, then a new species is created. Fitness sharing is then applied at the species level; in this way, each individual receives a penalization that is proportional to the size of the species.

The above penalization is not applied to the best solution of each species. Moreover, the penalization is only considered during parent selection, which is done deterministically by ordering the individuals of the population based on their adjusted fitness. In this way, individuals that were penalized by a large amount might not be able to produce offspring. While GP is prone to generate a large number of isomorphic trees, controlling this can be computationally prohibitive (Burke et al. 2004). However, the speciation process of neat-GP is a simple heuristic approach to indirectly control the amount of isomorphic trees.

3.2 Selection, genetic operators and survival

Selection is elitist, in each species the $p_{\text{worst}}\%$ of the worst individuals are removed, and these are then replaced by the same number of offspring. This is done by setting a maximum number of offspring for each individual, determined in a proportional manner relative to their raw fitness. Individuals are ranked based on their adjusted fitness, they are then chosen in that order and the genetic operator applied (crossover or mutation) is determined randomly. When a parent is selected, the number of expected offspring is reduced accordingly, and individuals cannot be chosen as parents when this value reaches zero.

3.3 Hyper-parameter optimization for neat-GP

In the present study, we choose to focus on the following hyper-parameters of neat-GP:

1. Crossover rate (C_p) and mutation rate (M_p). Since new individuals are only created through these two genetic operators, and since the operators are mutually exclusive

Table 1 Neat-GP hyper-parameters, showing the manual configuration reported in (Trujillo et al. 2016; Hernández-Beltran et al. 2019) and the range of values considered by the iF-Race optimization

Parameter	Manual setting	Optimization range
Crossover probability C_p	0.7	$[0-1]$
Mutation probability M_p	0.3	$1-C_p$
Species criterion β	0.5	$[0-1]$
Species threshold h	0.15	$[0.05-0.40]$
Elitism $p_{\text{worst}}\%$	0.5	$[0.2-0.8]$

in neat-GP, then both hyper-parameters can be defined with one numerical value given that $M_p = 1 - C_p$.

2. Species criterion β. This is the short name for the β hyper-parameter in Eq. 1, which defines the relative importance in the similarity measure used to consider both the size and depth of trees when they are compared in the speciation process.
3. Species threshold h. This hyper-parameter controls the creation of new species during the search and can be thought of as a niche radius in an EA framework.
4. Elitism $p_{\text{worst}}\%$. The hyper-parameter determines the amount of elitism used by the search in each individual species.

Table 1 summarizes the above-mentioned neat-GP hyper-parameters and also shows the manual values defined for these parameters in earlier works (Trujillo et al. 2016; Hernández-Beltran et al. 2019) in the second column from left to right. Moreover, the third column shows the range of values considered by the iF-Race algorithms during the hyper-parameter optimization reported in the next section. Note that for h and $p_{\text{worst}}\%$, in theory both hyper-parameters could have values between 0 and 1. However, smaller ranges are used in this study for the following reasons. In a series of preliminary tests, when these hyper-parameters took values outside the range specified in Table 1, the search process produced very poor results or bloated solutions. When $p_{\text{worst}}\%$ is large (above 0.8), selective pressure is too high and the algorithm prematurely converges to a local optima, and if it is too low (below 0.2), then the algorithm converges very slowly to a high fitness solution. Regarding h, when this value was set to high (above 0.4) then almost all individuals were grouped into a single large species, thus eliminating the speciation process. Therefore, by limiting the ranges of these hyper-parameters iF-Race focuses on more promising regions of hyper-parameter space.

4 Experiments

The goal of the experimental work is to characterize the hyper-parameter space of the neat-GP algorithm. Three variants of iF-Race are considered, namely the Original formulation, the Elitism and Gaussian approaches described in Sect. 2. The test problems are defined in the following subsection, along with the experimental setup.

4.1 Problems and experimental setup

Two sets of benchmark problems suggested in (McDermott et al. 2012) are used; these are shown in Tables 2 and 3. All of the problems define symbolic regression tasks, the most common application domain of GP, i.e., learning problems with one or several real-valued inputs and a single real-valued output, defined by a training set used to compute fitness and a test set to validate the performance of the best model found. The first set of problems, which consist of the 15 Koza and Nguyen problems shown in Table 2, are considered to be easy problems. The second set of problems, the 16 Pagie and Korns problems shown in Table 3, present more difficult regressions tasks.

For all problems, the goal is to optimize the hyper-parameters shown in Table 1, and the remaining hyper-parameters were set as follows. In all runs, neat-GP used

Table 2 First set of *easy* benchmark problems: Koza and Nguyen

ID	Name	Inputs	Function
P1	Koza-1	1	$x^4 + x^3 + x^2 + x$
P2	Koza-2	1	$x^5 - 2x^3 + x$
P3	Koza-3	1	$x^6 - 2x^4 + x^2$
P4	Nguyen-1	1	$x^3 + x^2 + x$
P5	Nguyen-2	1	$x^4 + x^3 + x^2 + x + 1$
P6	Nguyen-3	1	$x^5 + x^4 + x^3 + x^2 + x$
P7	Nguyen-4	1	$x^6 + x^5 + x^4 + x^3 + x^2 + x$
P8	Nguyen-5	1	$\sin(x^2)\cos(x) - 1$
P9	Nguyen-6	1	$\sin(x) + \sin(x + x^2)$
P10	Nguyen-7	1	$\ln(x + 1) + \ln(x^2 + 1)$
P11	Nguyen-8	1	$\sqrt{x}$
P12	Nguyen-9	2	$\sin(x) + \sin(y^2)$
P13	Nguyen-10	2	$2\sin(x) + \sin(y^2)$
P14	Nguyen-11	2	x^y
P15	Nguyen-12	2	$x^4 - x^3 + \frac{y^2}{2} - y$

Table 3 Second set of *hard* benchmark problems: Pagie and Korns

ID	Name	Inputs	Function
P1	Pagie-1	2	$\frac{1}{1+x^4} + \frac{1}{1+y^4}$
P2	Korns-1	2	$1.57 + (24.3v)$
P3	Korns-2	5	$0.23 + 14.2\frac{v+y}{3w}$
P4	Korns-3	5	$-5.41 + 4.9\frac{v-x+\frac{y}{w}}{3w}$
P5	Korns-4	5	$-2.3 + 0.13\sin(z)$
P6	Korns-5	5	$3 + 2.13\ln(w)$
P7	Korns-6	5	$1.3 + 0.13\sqrt{x}$
P8	Korns-7	5	$213.80940889(1 - e^{-0.54723748542x})$
P9	Korns-8	5	$6.87 + 11\sqrt{7.23xvw}$
P10	Korns-9	5	$\frac{\sqrt{x}}{\ln(y)}e^{\frac{z}{v^2}}$
P11	Korns-10	5	$0.81 + 24.3\frac{2y+3z^2}{4v^3+5w^4}$
P12	Korns-11	5	$6.87 + 11\cos(7.23x^3)$
P13	Korns-12	5	$2 - 2.1\cos(9.8x)\sin(1.3w)$
P14	Korns-13	5	$32 - 3\frac{\tan(x)}{\tan(y)}\frac{\tan(z)}{\tan(v)}$
P15	Korns-14	5	$22 - 4.2(\cos(x) - \tan(y))\frac{\tanh(z)}{\sin(v)}$
P16	Korns-15	5	$12 - 6\frac{\tan(x)}{e^y}(\ln(z) - \tan(v))$

a population of 100 individuals and 100 generations. Fitness was computed based on the training set of fitness cases using the root mean squared error (RMSE) (thus defining a minimization problem). The terminals for all GP trees were the input variables of the problems and random ephemeral constants, and the function set was defined as $\{+, -, \times, \div, \sin, \cos, \log, \text{sqrt}, \tan, \tanh\}$.[2] In all cases, the training and testing sets were sampled following (McDermott et al. 2012).

Given that three variants of iF-Race are considered (Original, Elitism and Gaussian) and given two sets of problems (Koza and Nguyen and Pagie and Korns), six different sets of experiments are carried out. In all cases, training RMSE is used to compare solutions during racing, but testing performance is also recorded and reported below. For all iF-Race variants, a total of $s = 10$ iterations is performed, and a set of 200 random parametrizations are used to initialize each race.

The number of total neat-GP runs executed during a single iF-Race experiment is large. Take, for instance, the most extreme case, applying the Original iF-Race on the 16 Pagie and Korns problems. For each iteration of the race, 200 candidate parametrizations are considered initially. If at the end of an iteration there are, for example, at least 30 elite parametrizations, then the total number of runs is greater than $30 \times 16 \times 10 = 4,800$ runs for that experiment

[2] The division $\div$ is protected, returning the numerator when the denominator is zero

Table 4 Total elite parametrizations produced in each experiment

Problem set	iF-Race variant	Total parametrizations
Koza and Nguyen	Original	31
Koza and Nguyen	Elitism	61
Koza and Nguyen	Gaussian	42
Pagie and Korns	Original	612
Pagie and Korns	Elitism	227
Pagie and Korns	Gaussian	45

alone. Given this high computational cost, the entire system was implemented using the DEAP (De Rainville et al. 2012) implementation of neat-GP (Juárez-Smith and Trujillo 2016),[3] using Python 2.7 and parallelized using Message Passing Interface (MPI) mpi4py Python package running over Ubuntu 16.04, on a cluster with five nodes, where each node had a four core Intel i5-4590 CPUs @3.30 GHz and 8 GB RAM. The server node functioned as the administrator in charge of distributing independent threads of single neat-GP executions and also collects all the results from the client nodes.

4.2 Results

This section provides a detailed presentation of the main results. Table 4 presents a summary of the total number of elite parametrizations found in each experiment (each iF-Race variant and each problem set). Notice two trends. First, the number of parametrizations produced in the easier set of problems, Koza and Nguyen, is smaller than on the more difficult set when using the Original and Elitism iF-Race variants. This suggests that there are some elite parametrizations that quickly dominate the racing process for these problems. On the other hand, for the more difficult problems, Pagie and Korns, it seems to be more challenging to find parametrizations that dominate the iF-Race process; i.e., it is more difficult to find parametrizations that are good for all the problems. Such a result is not surprising, since it is known that in order to achieve the best possible results on difficult problem instances it is often necessary to tune an algorithm for that specific scenario (Michalewicz 1996).

In the case of iF-Race with Gaussian update, the number of elite parametrizations is similar for both problem sets. This result, however, may be better explained by the greedy nature of this optimization process, compared to the behavior of the Original and Elitism variants. The more explorative Original iF-Race is able to randomly find many parametrizations that cannot be discarded during the racing process. This is aligned with the logic of the algorithm and is corroborated by some

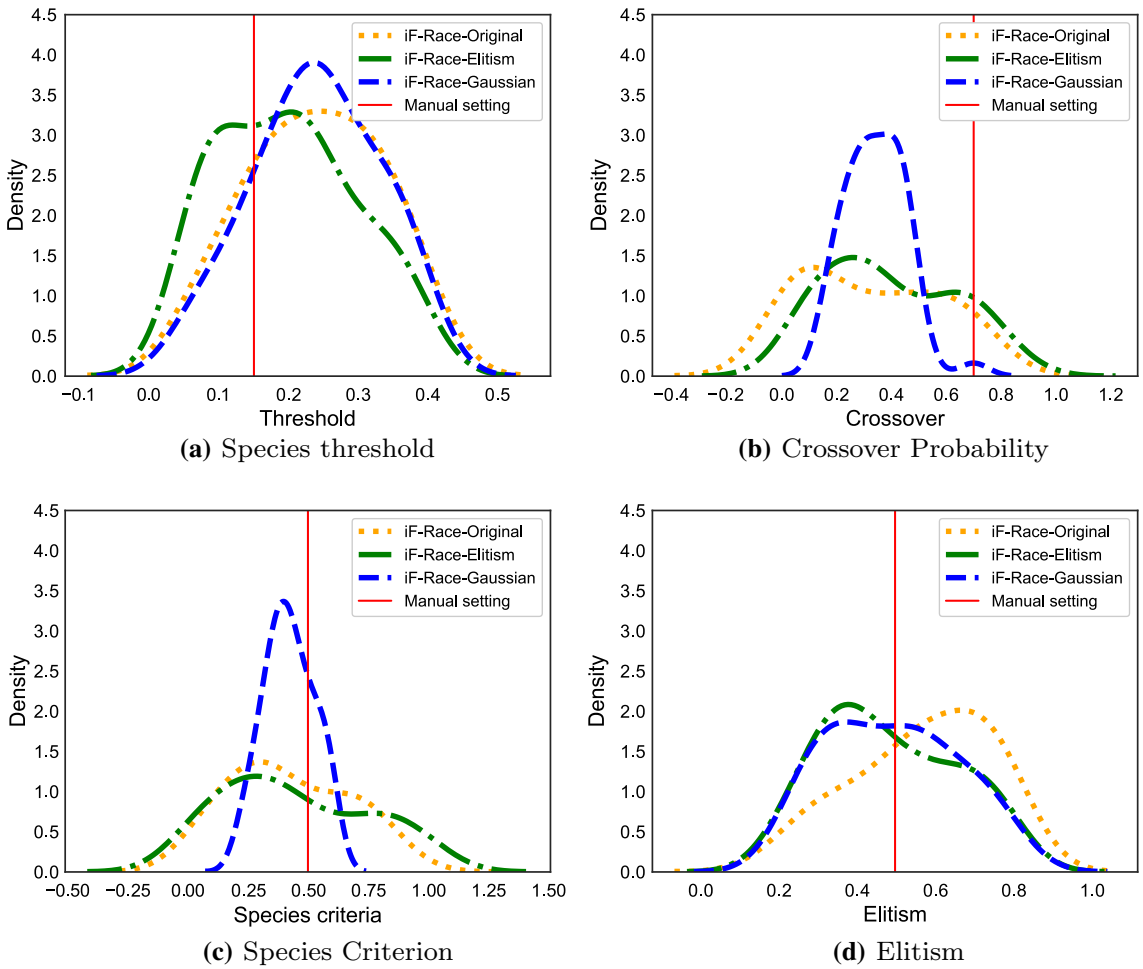
[3] https://github.com/saarahy/NGP-LS

of the results presented by Sipper et al. (2018), where the authors concluded that for difficult problems it is feasible to find random parametrizations that produce good results, but most of these parametrizations will not generalize to other problems. This highlights the need to conduct more research on hyper-parameter optimization in difficult real-world scenarios.

4.2.1 Sampling Density of Hyper-Parameter Space

Figures 1 and 2 present density plots for each hyper-parameter produced by each experiment. The plots show how the elite parametrizations produced by each iF-Race variant are clustered in hyper-parameter space. For reference, the plots also show the manual setting (as a solid red vertical line), as suggested in the original neat-GP paper (Trujillo et al. 2016). Several trends are noticeable in these plots. First, the Gaussian variant, denoted by a broken blue line, produces density plots where values are more tightly clustered, but this is more apparent for two hyper-parameters in particular: crossover probability and species criterion, as shown in Figs. 1 and 2, in subplots (b) and (c), respectively. This applies for both problem groups. Second, the manual configuration shows up very close to the maximum of the density curves for two hyper-parameters: species criterion and elitism percentage. This suggests good agreement between the iF-Race optimization and the manual configuration for these hyper-parameters. However crossover probability and, to a lesser extent, species threshold show larger deviations between the maxima of the iF-Race density and the manual configuration.

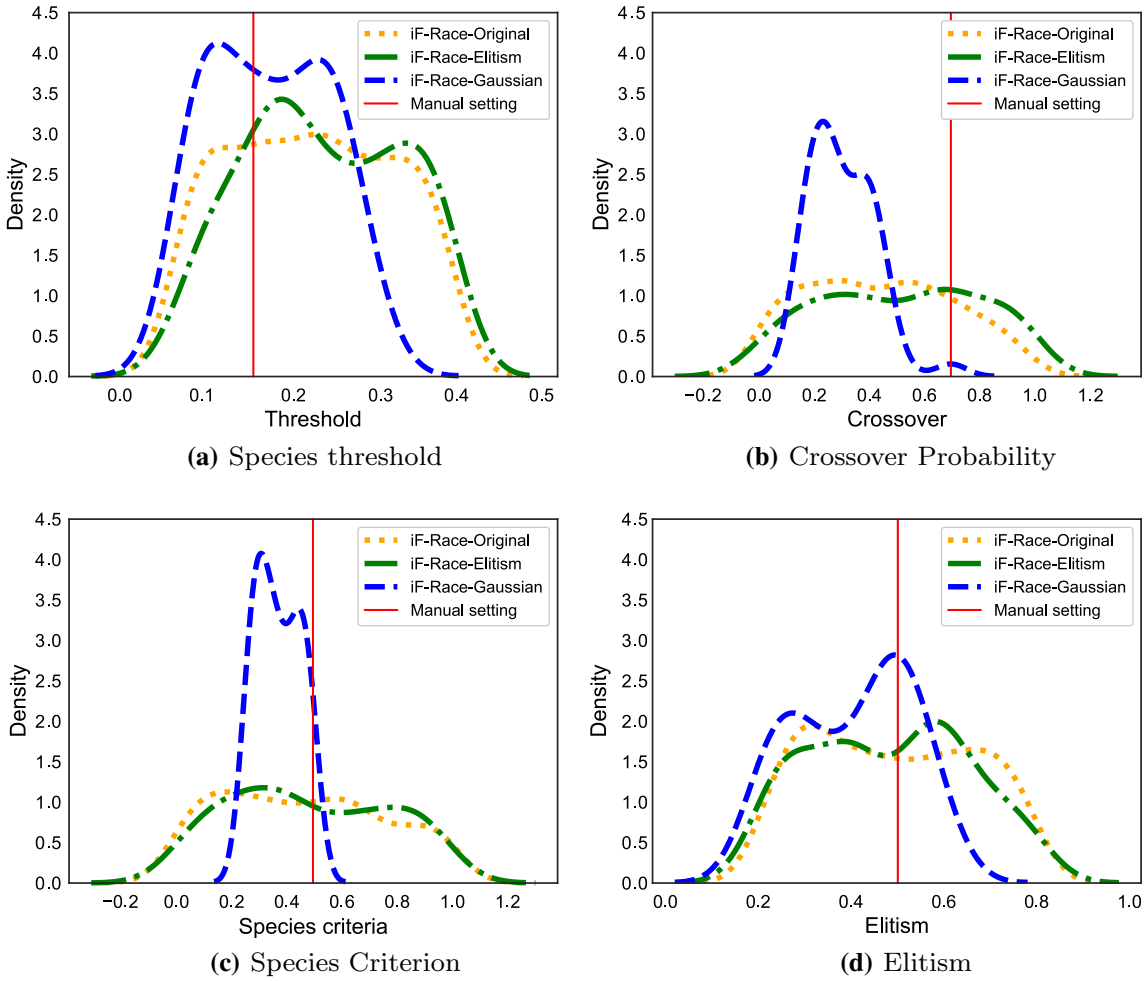
A more detailed view of the elite hyper-parameters found by each iF-Race variant is shown in the radar plots depicted in Fig. 3 (easy Koza and Nguyen problems in Table 2) and Fig. 4 (hard Pagie and Korns problems in Table 3). The plots highlight all of the elite parametrizations produced by the racing methods, namely (a) Original, (b) Elitism and (c) Gaussian.



(a) Species threshold **(b)** Crossover Probability

(c) Species Criterion **(d)** Elitism

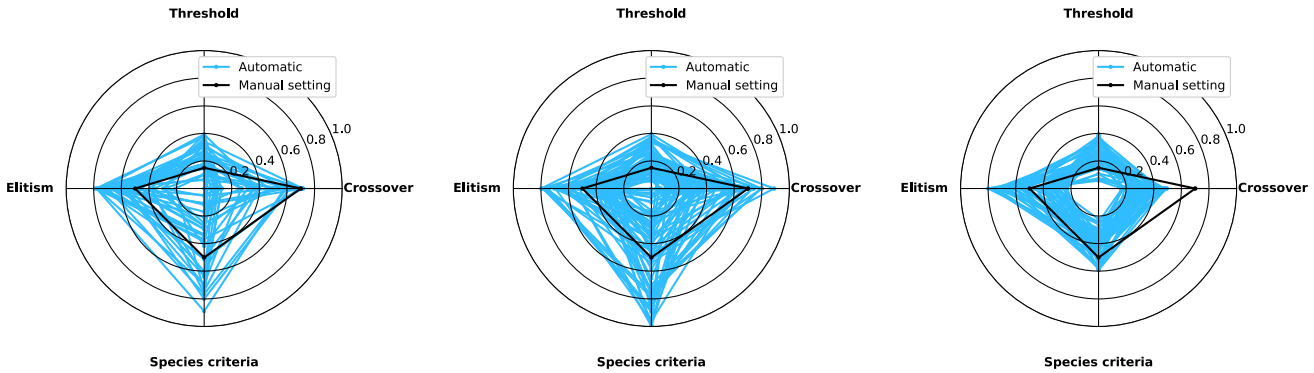
Fig. 1 Density plots for each hyper-parameter on the Koza and Nguyen problems: **a** species threshold; **b** crossover probability; **c** species criterion; and **d** Elitism. Each plot shows curves for the three iF-Race variants (Original, Elitism and Gaussian) and a vertical line for the manual configuration from (Trujillo et al. 2016)



(a) Species threshold **(b)** Crossover Probability

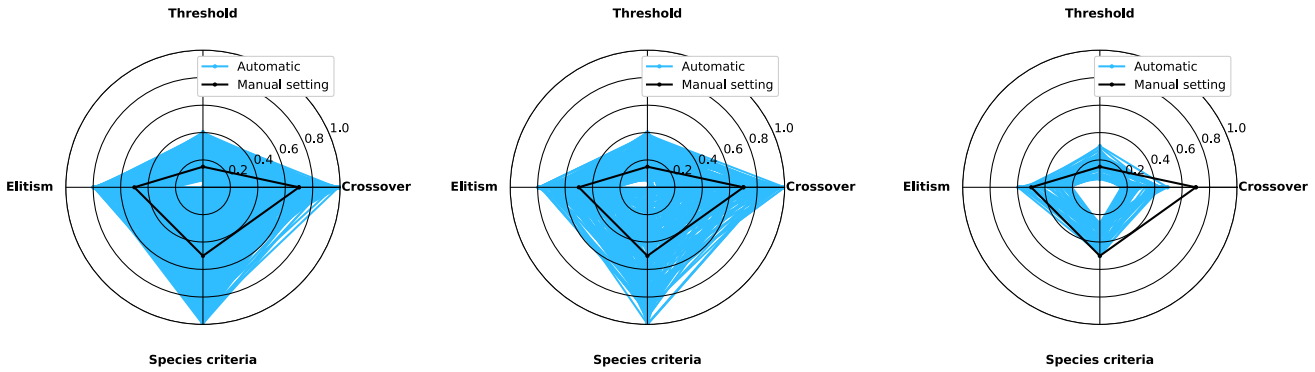
(c) Species Criterion **(d)** Elitism

Fig. 2 Density plots for each hyper-parameter on the Pagie and Korns problems: **a** species threshold; **b** crossover probability; **c** species criterion; and **d** Elitism. Each plot shows curves for the three iF-Race variants (Original, Elitism and Gaussian) and a vertical line for the manual configuration from (Trujillo et al. 2016)



(a) Original **(b)** Elitism **(c)** Gaussian

Fig. 3 Radar plots for each hyper-parameter on the Koza and Nguyen problems: **a** Original; **b** Elitism; and **c** Gaussian. Each plot shows curves (darker line) for the manual configuration from (Trujillo et al. 2016)



(a) Original **(b)** Elitism **(c)** Gaussian

Fig. 4 Radar plots for each hyper-parameter on the Pagie and Korns problems: **a** Original; **b** Elitism; and **c** Gaussian. Each plot shows curves (darker line) for the manual configuration from (Trujillo et al. 2016)

We included the manual configuration values as a reference point, denoted by a black solid line in the plots. Notice how the elite parametrizations are similar to the manual configuration for the crossover parameter on the easy problems, regardless of the method used (right-hand side of each plot in Fig. 3). The same can be observed for the species criteria, but only for the Gaussian method. For the rest of the parameters in the easy problems, the situation is less clear.

If we now focus our attention on the difficult problems, as shown in Fig. 4, we can see that the Gaussian method is able to find solutions in a more compact region of parameter space for all the considered hyper-parameters. In this case, the manual parametrization is close to the upper limit of the range of values produced by the Gaussian method for all hyper-parameters, except for the species threshold. On the other hand, the Original and Elitism iF-Race variants generate elite parametrizations that basically cover the entire space of allowed values, the manual configuration lying basically in the middle.

4.2.2 Test RMSE and Solution Size

It is also relevant to analyze the performance of the elite parametrizations produced by iF-Race, based on performance and size of the models generated by neat-GP. Performance is measured by the testing RMSE of the best model found (remember that the racing elimination process was based on training RMSE), while solution size is computed as the average number of nodes of all individuals in the final population of the neat-GP evolution. Figure 5 presents the testing RMSE achieved on each of the 15 problems in the Koza and Nguyen set, using the enumeration in Table 2. Figure 6 shows the same information for the Pagie and Korns set of 16 problems. The results are shown as a parallel coordinate plot, where each curve represents one elite parametrization and the vertical axis represents each benchmark problem. Moreover, two curves are shown with special lines: first, in black the average performance, over 30 runs, of the manual configuration as specified in (Trujillo et al. 2016) and second, with a dashed red line, a single randomly chosen elite configuration produced by the iF-Race variant, referred to as the single automatic solution (SAS). This second curve is used to provide a visual comparison with the manual configuration, and it is chosen randomly since the iF-Race does not provide a mechanism to prefer a particular elite parametrization over the rest. In all of these plots, lower is better since the goal is to minimize error.

Notice that while the automatic parametrizations do outperform the manual parametrization on some problems, this is not always the case. In fact, on most problems, the performance of the manual configuration and the elite parametrizations produced by iF-Race are more or less equivalent. While this may seem counter intuitive, it is important to remember that this comparison is based on testing RMSE and that iF-Race is based on training RMSE. Hence, while iF-Race pushes the search for optimal hyper-parameters toward better performing parametrizations, these might not translate to substantial (if any) gains in testing error due to the possibility of overfitting.

A similar analysis for solution size is presented in Figs. 7 and 8, for the Koza and Nguyen and Pagie and Korns problem sets, respectively. The plots show the average size of all individuals in the final population (end of the run) of each GP search process. These results show that the manual configuration evolves smaller trees, and this is true for both sets of problems. In fact, this difference is much more pronounced on the more difficult problem set, Pagie and Korns, as shown in Fig. 8. On the one hand, results such as these seem reasonable, since all of the iF-Race are exclusively focusing on predictive accuracy of the GP trees as measured by RMSE. The hyper-parameter optimization does not consider solution size, so there is no reason to expect

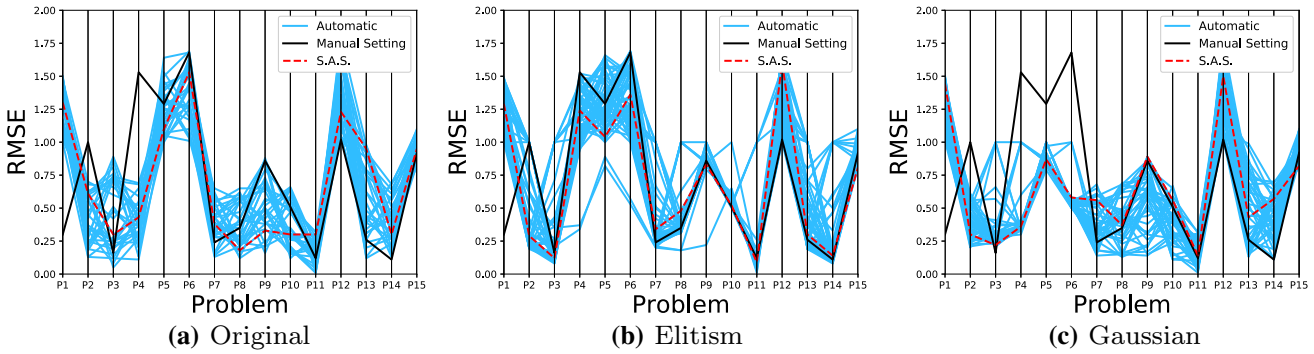


Fig. 5 Parallel coordinates plots for the test error (RMSE) on the Koza and Nguyen problems: **a** Original; **b** Elitism; and **c** Gaussian. Each plot shows curves (darker line) for the manual configuration from (Trujillo et al. 2016) and a dashed red line for the randomly chosen single automatic solution (SAS)

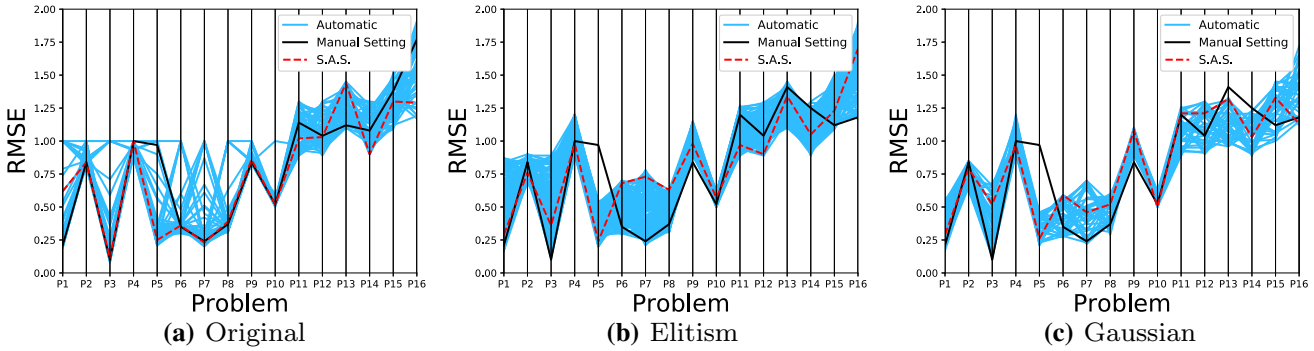


Fig. 6 Parallel coordinates plots for the test error (RMSE) on the Pagie and Korns problems: **a** Original; **b** Elitism; and **c** Gaussian. Each plot shows curves (darker line) for the manual configuration from (Trujillo et al. 2016) and a dashed red line for the randomly chosen single automatic solution (SAS)

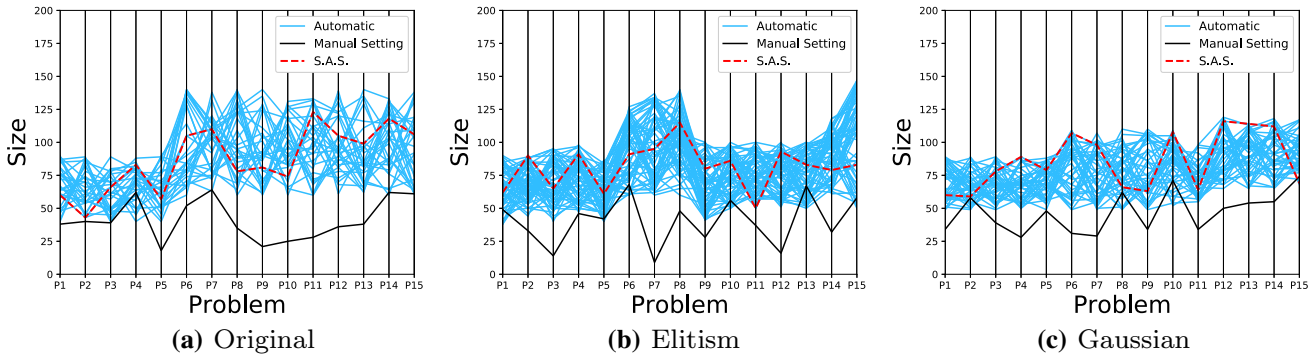


Fig. 7 Parallel coordinates plots for the average size (number of nodes) of all individuals in the final population for the Koza and Nguyen problems: **a** Original; **b** Elitism; and **c** Gaussian. Each plot shows curves (darker line) for the manual configuration from (Trujillo et al. 2016) and a dashed red line for the randomly chosen single automatic solution (SAS)

that the optimized parametrizations should produce compact solutions or eliminate bloat. On the other hand, such a wide gap between the automatic and manual parametrizations was unexpected. Moreover, the difference in solution size is consistent over all problems, probably the most consistent result in all of our experiments. Finally, it is worth noting that other measures could have been adopted besides solution size to analyze solution complexity, such as the order of nonlinearity (Vladislavleva et al. 2009). However, in the study presented in this work this measure of complexity would be similar among the solutions yielded by the methods because the use of inner functions was not controlled in any way as done in (Vladislavleva et al. 2009).

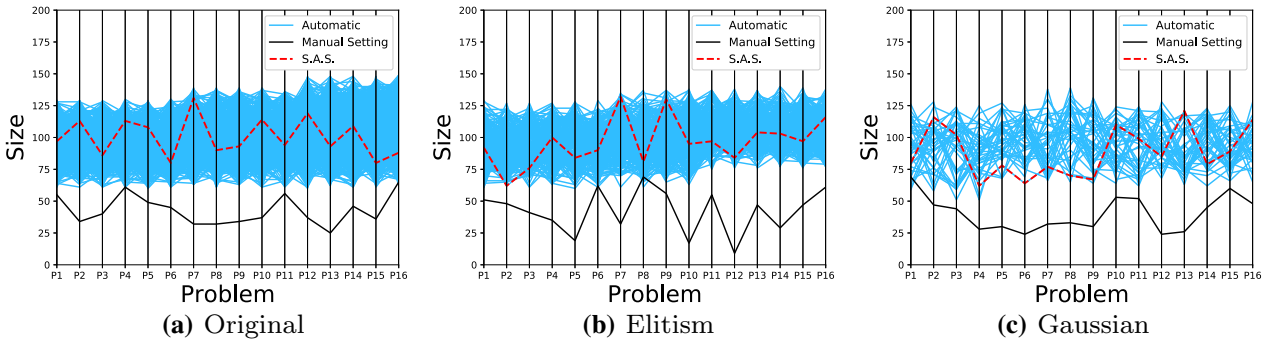


Fig. 8 Parallel coordinates plots for the average size (number of nodes) of all individuals in the final population for the Pagie and Korns problems: **a** Original; **b** Elitism; and **c** Gaussian. Each plot shows curves (darker line) for the manual configuration from (Trujillo et al. 2016) and a dashed red line for the randomly chosen single automatic solution (SAS)

5 Discussion

The most useful way to analyze the results presented so far is to contrast them with the results presented by Sipper et al. (2018). To do so, we will focus on some of the main findings by Sipper et al. (2018) and discuss how our results can be viewed from that perspective. Before doing so, it is important to keep in mind three differences between both studies. First, Sipper et al. (2018) considered common GP hyper-parameters, such as population size, while the present work focused on more specialized parameters of the neat-GP algorithm. Second, Sipper et al. (2018) used fitness (training performance) to evaluate the hyper-parameter optimization process, while the present work uses fitness to guide the hyper-parameter optimization process, but also considers test performance and solution size in the subsequent analysis. Finally, Sipper et al. (2018) used random search and an EA to perform hyper-parameter optimization, while this work uses iF-Race.

Sipper et al. (2018) suggest that hyper-parameter space is rich with parametrizations that can achieve high performance levels. In other words, determining *optimal* values for a single hyper-parameter is not a realistic goal, since there are strong dependencies between different parameters (for instance between population size and number of generations), such that many different values might lead to good performance when properly combined. The results presented in this paper, particularly those summarized in Figs. 1, 2, 3 and 4, seem to validate this assertion since at least two of the iF-Race variants (Original and Elitism) tend to cover almost the entire range of allowable parametrizations considered in our experiments.

Another conclusion from Sipper et al. (2018) is that finding a single parametrization that works well for a single problem is relatively easy, but finding a parametrization that works well across multiple problems is a more difficult task. In general, the results presented here cannot be said to contradict this conclusion, but do suggest that, at the very least, a better hyper-parametrization process (such as iF-Race) can consistently find parametrizations that actually generalize well across multiple problems.

On the other hand, some of the aspects not considered in the study by Sipper et al. (2018) reveal the following useful insights. First, that the manual configuration proposed in Trujillo et al. (2016) for neat-GP is actually comparable to the elite parametrizations produced by iF-Race. It seems clear that the automatic method can generate hyper-parameter settings that outperform the manual configuration in several problems, if not most, particularly when considering the predictive accuracy of the models. However, this comes at a non-negligible computational cost, which can be important in a real-world scenario. Conversely, the manual configuration is actually quite competetive and even outperforms the iF-Race parametrization in several cases.

Second, the elite iF-Race parametrizations produce bloated GP runs. Notice that the manual parametrization relies on single parameter values that are similar to many of the elite configurations produced by iF-Race, as clearly shown in the radar plots of Figs. 3 and 4. One possible explanation might be to assume that the elite parametrizations produced by iF-Race overfit the problem data. However, this is not the case. Figures 9 and 10 show comparisons of training RMSE between the elite parametrizations and the manual configuration. Clearly, the manual configuration shows, in general, lower training errors, particularly for the second set of problems. This suggests that the lack of bloat control by the elite parametrizations is not due to overfitting. In fact, this also reveals a good property of the elite parametrizations produced by iF-Race relative to the manual configuration, since the former produces less overfitting than the latter.

This suggests that the manual configuration is, in fact, unique, a *just right* setting that allows neat-GP to produce good solutions while also limiting the effect of bloat. If this is true, it could imply that the algorithm might be fragile,

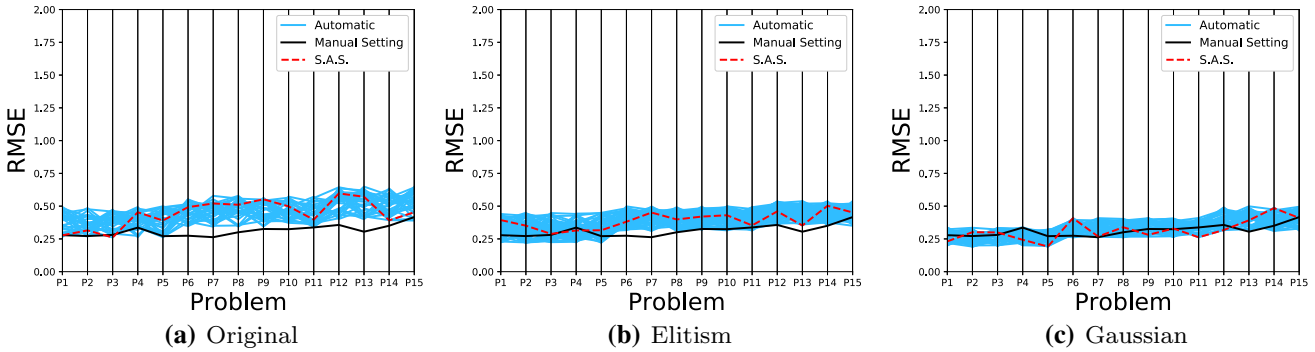


Fig. 9 Parallel coordinates plots for the training error (RMSE) on the Koza and Nguyen problems: **a** Original; **b** Elitism; and **c** Gaussian. Each plot shows curves (darker line) for the manual configuration from (Trujillo et al. 2016) and a dashed red line for the randomly chosen single automatic solution (SAS)

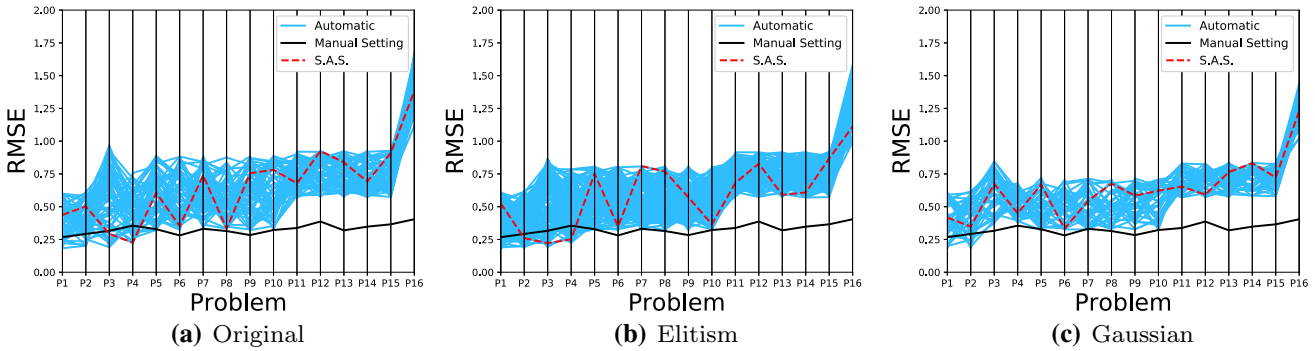


Fig. 10 Parallel coordinates plots for the training error (RMSE) on the Pagie and Korns problems: **a** Original; **b** Elitism; and **c** Gaussian. Each plot shows curves (darker line) for the manual configuration from (Trujillo et al. 2016) and a dashed red line for the randomly chosen single automatic solution (SAS)

in the sense that even slight hyper-parameter changes can have drastic effects on its ability to control bloat. Therefore, future work will focus on using a hyper-parameter optimization procedure that accounts for both performance measures (solution size and RMSE) concurrently, since iF-Race did not consider it in our experiments, while the manual configuration was tuned to account for this phenomenon.

6 Concluding remarks

This paper presents an experimental characterization of the hyper-parameter space for neat-GP. This work builds upon recent findings (Sipper et al. 2018), considering system-specific parameters and other important aspects to account for in a GP system, namely test performance and size.

There are several conclusions to be drawn from this paper. GP systems, and particularly neat-GP, can produce good test error using a wide range of different hyper-parameter values. The iF-Race algorithm can consistently find parametrizations that compare favorably with a manual configuration in terms of testing RMSE. However, most of these parametrizations have a negative effect on the ability of neat-GP to control bloat. This points to the need to perform multi-objective hyper-parameter optimization, which would be a non-trivial task, and to the best of our knowledge has not been carried out in the GP community, opening up a great possibility for future work. The results shown also lead to the conclusion that the manual configuration reported in (Trujillo et al. 2016) is, in fact, fragile and unique. The manual configuration shows lower training errors and basically equal testing error compared to the automatic parametrizations. The individual parameter values do not seem unique, and they are well within the ranges explored by iF-Race, but the specific combination used is clearly beneficial for bloat control.

From a more general perspective, this paper suggests that hyper-parameter optimization in GP is a difficult task to perform. This is true because GP can, and for many practitioners should, produce human readable models, and the ability to do so is an advantage the method has over other learning methods. However, having balance between both solution quality and solution size makes hyper-parameter optimization a more difficult computational task. This clearly highlights the need for more research on this topic.

Funding This research was funded by CONACYT (Mexico) Fronteras de la Ciencia 2015-2 Project No. FC-2015-2:944, TecNM project

6826.18-P, and the second author was supported by CONACYT graduate scholarship for his masters thesis.

Compliance with ethical standards

Conflict of interest The authors declare that they have no conflict of interest.

Ethical approval This article does not contain any studies with human participants or animals performed by any of the authors.

References

Ackermann MR, Blömer J, Sohler C (2008) Clustering for metric and non-metric distance measures. In: Proceedings of the nineteenth annual ACM-SIAM symposium on discrete algorithms, Philadelphia, PA, USA, SODA '08, pp 799–808

Bergstra J, Bengio Y (2012) Random search for hyper-parameter optimization. J Mach Learn Res 13:281–305

Birattari M (2009) Tuning metaheuristics: a machine learning perspective, 1st edn. Springer Publishing Company, Incorporated, Berlin

Birattari M, Stützle T, Paquete L, Varrentrapp K (2002) A racing algorithm for configuring metaheuristics. In: Proceedings of the 4th annual conference on genetic and evolutionary computation, Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, GECCO'02, pp 11–18

Burke EK, Gustafson S, Kendall G (2004) Diversity in genetic programming: an analysis of measures and correlation with fitness. IEEE Trans Evol Comput 8(1):47–62

Cava WL, Silva S, Danai K, Spector L, Vanneschi L, Moore JH (2019) Multidimensional genetic programming for multiclass classification. Swarm Evol Comput 44:260–272

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, New York, NY, USA, GECCO '12, pp 85–92

Hansen N, Ostermeier A (2001) Completely derandomized self-adaptation in evolution strategies. Evol Comput 9(2):159–195

Hernández-Beltran JE, Díaz-Ramírez VH, Trujillo L, Legrand P (2019) Design of estimators for restoration of images degraded by haze using genetic programming. Swarm Evol Comput 44:49–63

Juárez-Smith P, Trujillo L (2016) Integrating local search within neat-gp. In: Proceedings of the 2016 on genetic and evolutionary computation conference companion, ACM, New York, NY, USA, GECCO '16 Companion, pp 993–996

Karafotias G, Hoogendoorn M, Eiben AE (2015) Parameter control in evolutionary algorithms: trends and challenges. IEEE Trans Evol Comput 19(2):167–187

Koza JR (1992) Genetic programming: on the programming of computers by means of natural selection. MIT Press, Cambridge

Langdon WB, Poli R (2010) Foundations of genetic programming, 1st edn. Springer Publishing Company, Incorporated, Berlin

López-Ibáñez M, Dubois-Lacoste J, Pérez Cáceres L, Birattari M Stützle (2016) The irace packag: iterated racing for automatic algorithm configuration. Oper Res Perspect 3:43–58

López-López VR, Trujillo L, Legrand P (2018) Applying genetic improvement to a genetic programming library in c++. Soft Comput 23:11593–11609

McDermott J, White DR, Luke S, Manzoni L, Castelli M, Vanneschi L, Jaskowski W, Krawiec K, Harper R, De Jong K, O'Reilly UM (2012) Genetic programming needs better benchmarks. In: Proceedings of the 14th annual conference on genetic and evolutionary computation, ACM, New York, NY, USA, GECCO '12, pp 791–798

Michalewicz Z (1996) Genetic algorithms + data structures = evolution programs, 3rd edn. Springer, Berlin

Neumüller C, Wagner S, Kronberger G, Affenzeller M (2012) Parameter meta-optimization of metaheuristic optimization algorithms. In: Proceedings of the 13th international conference on computer aided systems theory–volume part I, Springer, Berlin, EUROCAST'11, pp 367–374

Olson RS, La Cava W, Orzechowski P, Urbanowicz RJ, Moore JH (2017) Pmlb: a large benchmark suite for machine learning evaluation and comparison. BioData Min 10(1):36

Sipper M, Fu W, Ahuja K, Moore JH (2018) Investigating the parameter space of evolutionary algorithms. BioData Min 11(1):2

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

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

Vanneschi L, Castelli M, Scott K, Trujillo L (2019) Alignment-based genetic programming for real life applications. Swarm Evol Comput 44:840–851

Vladislavleva EJ, Smits GF, den Hertog D (2009) Order of nonlinearity as a complexity measure for models generated by symbolic regression via pareto genetic programming. IEEE Trans Evol Comput 13(2):333–349