Reducing the Number of Training Cases in Genetic Programming

Reducing the Number of Training Cases in Genetic Programming

Giacomo Zoppi[1*] Leonardo Vanneschi[2] Mario Giacobini[1]

[1]*Data Analysis and Modeling Unit, Department of Veterinary Sciences, University of Torino,* Turin, Italy

[2]*NOVA Information Management School (NOVA IMS), Universidade Nova de Lisboa, Campus de Campolide, 1070-312*, Lisboa, Portugal

[*]Corresponding Author, giacomo.zoppi@unito.it

Abstract

In the field of Machine Learning, one of the most common and discussed questions is how to choose an adequate number of data observations, in order to train our models satisfactorily. In other words, find what is the right amount of data needed to create a model, that is neither underfitted nor overfitted, but instead is able to achieve a reasonable generalization ability. The problem grows in importance when we consider Genetic Programming, where fitness evaluation is often rather slow. Therefore, finding the minimum amount of data that enables us to discover the solution to a given problem could bring significant benefits. Using the notion of entropy in a dataset, we seek to understand the information gain obtainable from each additional data point. We then look for the smallest percentage of data that corresponds to enough information to yield satisfactory results. We present, as a first step, an example derived from the state of art. Then, we question a relevant part of our procedure and introduce two case studies to experimentally validate our theoretical hypothesis.

1 Introduction

Evolutionary Algorithms are inspired by the dynamics at the basis of biological evolution. The individuals are the candidate solutions to a given problem, and a problem-specific fitness function determines the quality of the individuals by checking how well they solve the problem using the available data. This process is essential to provide possible improvements across different generations but has a significant impact in terms of computational time. Aiming to reduce the time needed for the evolution, parameters such as the number of individuals, total number of generations, or the size of the dataset on which we compute the fitness function, can be manipulated. Since varying these quantities can affect the quality of the obtained results, the ideal goal would be to shorten the needed time without affecting the generalization ability of the model. It may be thought that it is not necessary to consider all the available data, but, on the contrary, that the additional use of some of them does not bring any real benefit.

These concepts are the inspiration of this work, in which we try to achieve the above-mentioned goal by proposing methods for choosing the amount of data (specifically, the number of fitness cases) that allows the model to be trained adequately, and in a reasonable amount of time.

We start in Section 2 with a brief discussion of the state of the art in the area, and then we present our idea.
The focal point of our approach is to find a way to discover the usefulness of a *new* data point in the training of a model. In Section 3, we formalize this idea. More specifically, in Section 3.1 we consider some concepts borrowed from Information Theory to introduce a metric that allows us

to quantify the amount of information gained by adding an additional fitness case to the training set. The metric is derived from the concept of data *entropy* and we utilize it to find a rough lower bound for the number of fitness cases to be used. Then, we point out some limitations of this approach and suggest possible solutions. In Section 4, we present some examples to test our findings, develop our theories and adjust them to tackle some of the occurred issues. We start with a simple example, in Section 4.1, derived from the state of art. Then, we question a relevant part of our procedure, in Section 4.2. Here, we point out that the Evolutionary Algorithm might see all the available data after a little number of steps. So we introduce two possible solutions. The first one, which is to rerun our Algorithm but with a smaller number of generations, is tested in Section 4.3, where we check for any possible difference from the first method used. We then apply this solution to a classification problem, in Section 4.4, to test it in different contexts. In Sections 4.5 and 4.6, we introduce a more complex example, testing a further alternative solution to the above mentioned issue. This second approach consists of considering a fixed set of fitness cases instead of selecting them randomly at each generation. Finally, in Section 5, we conclude the work and propose some possible future developments, including extending the work to other Machine Learning methods or testing our hypothesis on real-life problems.

2 Literature Review

Given the importance that the choice of the training set has in all Machine Learning methods, a considerable amount of literature has been published so far, trying to address the issue. However, the same cannot be said for Genetic Programming (GP), for which the amount of published material is undoubtedly lower, compared to many other Machine Learning methods. A variant of Genetic Programming, called Semantic Genetic Programming [8], was developed, and received a significant amount of attention from the scientific community, mainly because it was able to improve GP's computational efficiency. Traditional GP explores the space of programs using operators that manipulate their *syntactic representation*, regardless of their actual output. The idea behind the work of Moraglio and colleagues is to utilize semantic-aware search operators. The reason why this approach can be more efficient than traditional GP is that the algorithm does not have to evaluate the new individuals that are generated during the evolution, since the effect of the operators on the output is known. However, the final model needs to be reconstructed by going backward in the evolution. So, the method is efficient in the evaluation part, at the expense of a more complex and time-consuming program reconstruction.

Other works approach the problem of time consumption from a purely computational point of view. Wong et al., for example [11], introduced a method called Subtree Caching, using a Hashing for Equivalence MEthod (SCHEME). The program subtrees are stored, while taking into consideration the possible algebraic equivalences between these programs. The idea is to reduce the number of node evaluations performed during a GP run, obtaining a faster training process. The results suggest that SCHEME significantly reduces the number of node evaluations performed during the GP runs, which can lead to a faster GP training process.

More similarly to our proposal, some contributions have been working on the optimization of the fitness cases set.

It is the case, for instance, of Xie *et al.* [12], who suggested the idea of clustering the whole population with a heuristic called fitness-case-equivalence. Then, they select a representative for each cluster, evaluate the fitness of the representative and assign the same value to other members in the same cluster.

Chen *et al.* [3], propose a new framework to identify those data sources that are more useful to the algorithm and similar to the target domain data.

In a different vein, Galván-López *et al.* [5] optimized the results when the number of available data is low. They allowed the GP algorithm to build the set of fitness cases dynamically, by

aggregation over time.

A different idea was advanced by Ekárt [4], who proposed a particular mutation operator, able to simplify the programs, while maintaining their quality unchanged.

Bi *et al.* [1], instead, use a multi-objective genetic programming (MOGP) that optimizes at the same time both accuracy and a measure intended to enhance generalization of learned features for face image classification. All these works have achieved interesting results, but, contrarily to our proposal, they do not consider the number of fitness cases as a pivotal element of the research.

Few contributions address the problem in a way similar to our proposal. One case is represented by the work of Cantú-Paz [2], who studied an adaptive sampling technique that varies the number of training observations, based on the uncertainty of deciding between two individuals. It allows eliminating the need to set the sample size *a priori*, increasing the quality of the solutions, especially in the presence of noisy data, but it often requires higher computational costs. Ziegler *et al.* [14] presented a method to decrease the necessary number of fitness evaluations. To do so, they introduced a new evolutionary object that is a classifier with confidence information used to replace evaluations during a tournament selection. The creation of a ML-based auxiliary function does not entirely solve the problem even if it could add some advantage. Instead, more similarly to our idea, Zhang *et al.* [13] proposed to use only a part of the available training data at each generation. However, contrarily to our proposal, the data used are increased during a run, and Zhang's algorithm always terminates the evolution using all the available data.

The existing contribution that is more similar to our proposal is probably represented by the work of Giacobini *et al.* [6]. In that work, the authors utilises the concept of entropy, as a metric to describe the amount of needed information in a dataset. Also, they derived a method to select *a priori* the number of needed fitness cases to reconstruct a reliable solution. However, contrarily to our proposal, the focus of the work of Giacobini *et al.* was not on the generalization ability of the final model. The fact that we have that focus here has led us to significant modifications compared to what was proposed by Giacobini *et al.*. Indeed, while maintaining the same starting point, we deviate by trying to attribute to entropy a meaning closer to the usefulness that every single new datum could bring to an Evolutionary Algorithm. In other words, the contribution that a new fitness case could have in the search for the solution of a problem. In our case studies, we analyze, among other things, how these data are used, trying to address some inaccuracies that might cause us to incorrectly estimate the importance of some fitness cases. Furthermore, the introduction of more complex case studies, compared to Giacobini's work, allows us to test our theoretical hypothesis in different situations. By addressing non-obvious problems, we analyze the impact of adding a small amount of data. Moreover, we consider a continuous function to highlight the limitations of this method and offer, at the same time, an approach to use it. In addition, we propose a method to choose the fitness cases, something that had not been done by Giacobini *et al.*.

3 Proposed solution

In this section, we introduce the concepts behind our approach, which will then be further explained and used in the examples that follow. Assume we have an objective function and we can randomly extract observations from it to form the set of fitness cases. The question we want to answer is whether it is worth trying to obtain new fitness cases or the ones we already have are sufficient to faithfully reconstruct the objective function. Linking to Information Theory and considering the randomness of the source, we can say that, if the new data occurs often, then it is reasonable that we do not gain a lot of *new* information by receiving it. Instead, if the data is linked with a low probability of occurrence, then it is probable that we can gain some new information. So, we are trying to find a metric to give *value* to a new data point.

More particularly, we look for a metric that attains high values when the data point considered occurs with low probability and low values when the data point considered occurs with high probability. A metric that satisfies these needs is entropy [9].

3.1 Why Entropy is Appropriate for our Needs

Given the set of GP fitness cases for a discrete target function, $S = \{x_1, .., x_N\}$, we can see the target function itself $g : \{x_1, .., x_N\} \to y_1, .., y_M \subset \mathbb{R}$ as a random variable. In fact, the target function is a mapping from the sample space S of a probability space to a *finite* subset of $\mathbb{R}$. This idea allows us to apply the concept of *entropy* of a random variable to a GP discrete target function. Let $p_j = \mathbb{P}(g(x) = y_j)$, for $j \in \{1, .., M\}$, then we can compute the entropy of the function $g(x)$:

$$H(g(x)) = -C \sum_{j=1}^{M} p_j \log_a(p_j) \tag{1}$$

Since we want this quantity to somehow represent the amount of information that is required to be able to reconstruct the target function, we can tune the parameters intending to limit the entropy value to the interval $[0, 1]$. For simplicity, we set in the above definition $a = e$ and therefore to obtain values in $[0, 1]$, we set $C = 1/\ln(N)$. So, in our hypothesis, it will be the entropy of the target function, that is the average amount of information that is brought by the function itself, to suggest the minimum number of fitness cases needed to reconstruct the function. In other words, in the following examples, we will compute:

$$H(g(x)) = -\frac{1}{\ln(N)} \sum_{j=1}^{M} p_j \ln(p_j) \in [0, 1] \tag{2}$$

and we will use the value of $H(g(x))$ as the lower bound of the required amount of data to reconstruct the function $g(x)$.

All the above works for discrete target functions. So a question comes naturally: how can it be extended to continuous functions? A possible answer can be found in [9], and it is called differential entropy. However, unlike its discrete counterpart, differential entropy has some properties that are undesired for our objective. For instance, it can assume negative values and, even most importantly, it is not invariant to the change of variables. Given this, an alternative approach is needed for continuous functions, and a simple workaround was introduced in 4.5, where the function is discretized, by binning similar points.

4 Case Studies

In this section, we will test our hypothesis with different examples, to check for possible errors, and try to uncover possible weaknesses. In the following sections, we will run different simulations, often providing a number of fitness cases lower than the available ones to our algorithm. A GP algorithm will compute the best individual in each generation, considering *only* the subset of data that we will have made available for that generation. The plotted and commented fitness functions are, instead, computed evaluating the best individual of each generation, selected on all the fitness cases, intending to check the generalization ability of the individuals.

The approach is similar to [6]. So, as a first step, we decided to use exactly the two examples used in [6]. As expected, we obtained similar results. We will quickly show the first example, to contextualize the rest of the work.

4.1 Boolean Function

The first example considers a Boolean dataset, with seven input variables and, as fitness cases, the 128 possible combinations of these input variables and the corresponding output variables. The target function is defined as:

$$g = ((x_1 \wedge x_2) \vee x_3) \tag{3}$$

For calculating the entropy, we have to consider the output values. It is easy to see that Equation (3) returns 1 (i.e., *True*) in 80 of the 128 possible combinations of the Boolean inputs (fitness cases), and 0 (i.e., *False*) on the remaining ones. Using Equation (2), we can then compute the entropy of the random variable $Y = g(x_1, .., x_7)$, given that $\mathbb{P}(Y = 1) = \frac{80}{128} = 0.625$, and $\mathbb{P}(Y = 0) = \frac{48}{128} = 0.375$:

$$H(Y) = -\frac{1}{\ln(128)}[0.625 \ln(0.625) + 0.375 \ln(0.375)] = 0.1363 = 13.63\% \tag{4}$$

Using this result, according to our hypothesis, the minimal number of fitness cases needed to reconstruct the function should roughly be equal to the 13.63% of 128, i.e., it should be equal to $s = 17.5$.

Now, we run the GP algorithm to predict the output for each one of the possible combinations of Boolean input values $x_1, .., x_7$. As fitness function, we use the percentage of errors of the model, calculated over all the fitness cases. The parameters used by the algorithm can be found in the leftmost column of Table 1. Since the idea is that there is a subset of the fitness

Table 1: Parameters of the GP. Boolean Function, Car Evaluation, and Benchmark Examples

	Boolean Function 4.1	Boolean Function 4.3	Car Evaluation	Benchmark
Type of GP	Classical	Classical	Classical	Classical
Number of generations	100	20	20	50
Number of individuals	200	1000	1000	2000
Function Set	$\{\wedge, \vee\}$	$\{\wedge, \vee\}$	$\{\wedge, \vee, \neg, \leq, \geq, +, -, *\}$	$\{+, -, \times, \%, x^2, e^x, e^{-x}, \sin, \cos\}$
Terminal Set	$\{x_1, .., x_7\}$	$\{x_1, .., x_7\}$	$\{x_{1_1}, x_{1_2}, .., x_{6_1}, x_{6_2}\}$	$\{x^i, x + i, xi\}, i \sim Uniform(-5, 5)$
Initialization	Ramped Half and Half	Ramped Half and Half	Ramped Half and Half	Ramped Half and Half
Selection	Tournament, size 10	Tournament, size 10	Tournament, size 10	Tournament, size 10
Probability of mutation	0.1	0.1	0.1	0.1
Probability of crossover	0.9	0.9	0.9	0.9

cases which is sufficient to solve the problem, but we do not know which elements are in that subset, at each generation a random subset of fitness cases is chosen at random, with uniform probability. This way of feeding the fitness cases will be discussed later.

In Figure 1, we can see the results of ten different GP runs with different seeds and different numbers of fitness cases, two lower than the entropy value and two higher, made available to the algorithm. Only ten of the hundred curves have been drawn to make the figure easier to understand. This approach gives each generation always the same amount of data, but those data are chosen randomly. So, even with the presence of elitism in the GP algorithm, this gives rise to numerous fluctuations in the values of the fitness function, since the same individuals could perform worse on a different set of cases. For this reason, an average of the fitness was not used as this peculiarity of the results would have been lost. In Table 2, it is possible to see the results of 100 different runs, divided between convergent (or with no evolution at all if it already starts at the global optimum), and oscillating (or not convergent).

From the analysis of both the table and the plots, we can see that when the number of fitness cases used by GP is greater than the entropy of the function, the results are better, and the behavior of the individuals is converging. Instead, when the number of fitness cases is smaller than the entropy of the function, we have worse results with many oscillating behaviors. This simple example seems to confirm our hypothesis, but we must point out some caveats. The first is relative to how the fitness cases are chosen. Indeed, the entropy of the fitness functions gives

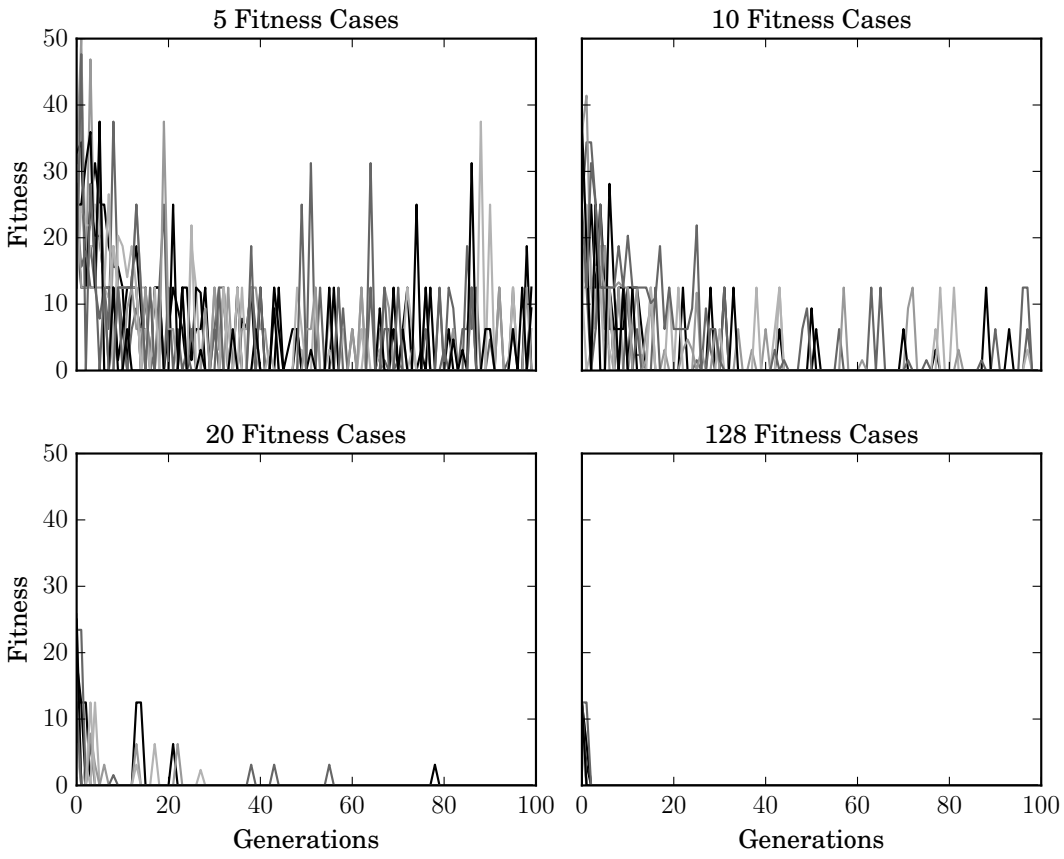

Figure 1: Ten sample GP runs for the boolean function with 5, 10, 20, and 128 fitness cases. 200 individuals.

Table 2: Comparison between runs that have found the solution to the boolean problem for different numbers of fitness cases

Behavior \ Fitness Cases	128	100	90	80	70	60	50	40	30	20	10	5
Convergent	100	100	100	100	100	99	96	97	98	97	43	3
Oscillating	0	0	0	0	0	1	4	3	2	3	57	97

an order of magnitude of the number of fitness cases, but, in our experiments, those were chosen randomly at each generation. This could allow the algorithm to see, after a little number of steps, all the available data. The second is that the case studies that confirm our hypothesis could be too simple. We will now discuss the first issue and next, we will consider more complex examples.

4.2 How Many Unique Fitness Cases Are Used?

The question that we would like to answer is how many unique fitness cases are seen by the algorithm if we use k random fitness cases at each generation.

Let X_t, $X_t \in [0, k]$, be the number of fitness cases used from the generation t, $t \in \mathbb{N}$, that were not used by any of the generations before. Now, consider $S_t = \sum_{j=1}^{t} X_j$, which is the total number of fitness cases used until generation t. The expected result of the first generation is deterministic and is equal to $\mathbb{E}[X_1] = k$.

Let us now consider the random variable X_t. Let us assume that we are at the t^{th} generation, and so we know the past until generation $t - 1$. We can then link the random variable to a

known random distribution in the following way:

$$X_t|X_{t-1},..,X_1 \sim HyperGeom(N,k,N-S_{t-1}) \tag{5}$$

Where $N - S_{t-1}$ describes the number of fitness cases not used until generation $t-1$ and *HyperGeom* indicates the hypergeometric distribution. Using to the Law of Total Expectation, we can now remove the conditioning on X_{t-1}, and we obtain:

$$\mathbb{E}[X_{t+1}] = \frac{k(N-k)^t}{N^t} \tag{6}$$

Then, using the fact that $q = \frac{N-k}{N} < 1$ since $k < N$, we have:

$$\begin{aligned}\mathbb{E}[S_{t+1}] = \mathbb{E}\left[\sum_{j=1}^{t+1} X_j\right] = \sum_{j=1}^{t+1}\mathbb{E}[X_j] = \sum_{j=1}^{t+1} k\frac{(N-k)^{j-1}}{N^{j-1}} = k\sum_{j=1}^{t+1}\frac{(N-k)^{j-1}}{N^{j-1}} = \\ = k\sum_{j=1}^{t+1}\left(\frac{N-k}{N}\right)^{j-1} = k\sum_{j=1}^{t+1} q^{j-1} = k\sum_{j=0}^{t} q^j = k\frac{1-q^{t+1}}{1-q}\end{aligned} \tag{7}$$

To sum up, what we can conclude is that even if we use only 5 out of 128 fitness cases at each available generation, after 40 iterations the GP algorithm will probably have seen approximately the 80% of all the fitness cases.

Actually, the expected percentage of seen cases is:

$$\frac{1}{n}\mathbb{E}[S_{t+1}] = \frac{1}{n}\cdot\frac{k(1-q^{t+1})}{1-q} = \frac{5}{128}\cdot\frac{1-\left(\frac{128-5}{128}\right)^{40}}{1-\frac{128-5}{128}} \approx \frac{101.99}{128} \approx 79.68\% \tag{8}$$

This outcome could affect or invalidate our hypothesis, and so we will now introduce two possible solutions.

The first one is to reduce the number of generations to decrease the percentage of fitness cases that can be used while keeping the same computational effort. In other words, instead of letting 200 individuals evolve for 100 generations, we will use a population of 1000 individuals and let them evolve for only 20 generations. This will be discussed in Section 4.3. The second option, discussed in more detail in Section 4.6, is to provide a fixed subset of fitness cases for all the generations. This one seems a simpler solution, but it actually carries a non-trivial choice: how to choose the particular used fitness cases.

4.3 Using a Smaller Number of Generations

The idea is to repeat the simulations of the previous example, and see if our outcomes are still valid. In Table 3, the expected percentages of fitness cases are presented, relative to 4.1, after 10, 20, and 100 generations. The setting of the GP algorithms is the same as the one used before, but this time with 1000 individuals and 20 generations. Further details on the parameters used in these experiments can be found in Table 1. We actually let the evolutionary process continue until generation 30, just to have a better view of the trends.

The results relative to the first example are presented in Figure 2, where we plotted the values for the first 30 generations. Similar to the case in which more generations were used, the results appear to confirm our hypothesis. Indeed, the plot relative to the runs with 10 fitness cases shows some oscillating behavior even after 20 generations, when, on average, over 80% of the fitness cases have globally been used.

Table 3: Expected percentage of seen cases

k	$\mathbb{E}[S_{10}]$	$\mathbb{E}[S_{20}]$	$\mathbb{E}[S_{100}]$
5	35.5%	56.7%	98.2%
10	59.1%	81.9%	99.9%
20	84.6%	97.2%	99.9%
128	100%	100%	100%

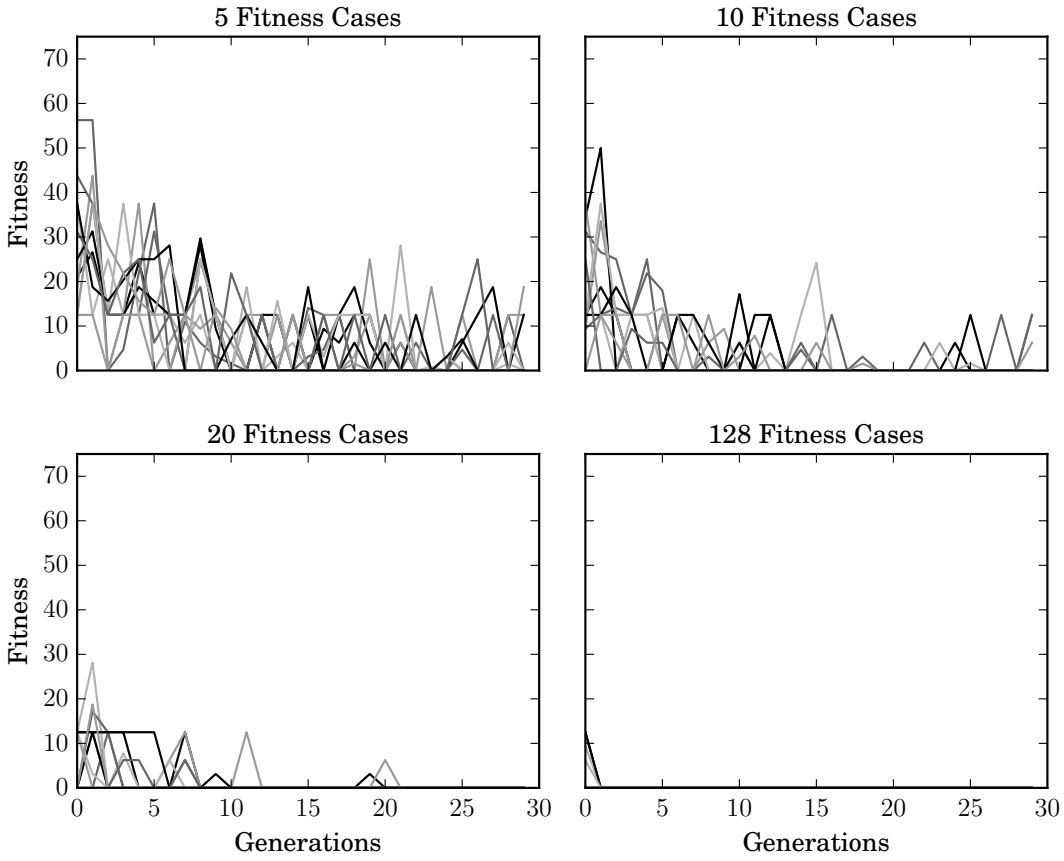

Figure 2: Ten sample GP runs for the boolean function with 5, 10, 20, and 128 fitness cases, respectively. 1000 individuals.

The plot relative to 20 fitness cases (remember that the limit value found from the entropy of the function is 17.5), instead, shows a converging behavior, for almost all the runs, even before reaching the 80% of seen fitness cases, which is near the 10^{th} generation. Now, with the aim of increasing the complexity of the problem, and testing the proposed second solution, we introduce a new case study.

4.4 Car Evaluation Data Set

In this example, we will consider a dataset called "Car Evaluation Data Set"[1]. The dataset is composed of 1728 data instances and 6 attributes. The attributes and class values that each instance can assume can be found in Table 4. As we can observe, there are four possible outcomes, related to the value of a car, namely, [acceptable, unacceptable, good, very good], and they appear in the dataset 1210, 384, 69, and 65 times respectively. For the entropy approach,

[1]https://archive.ics.uci.edu/ml/datasets/car+evaluation

Table 4: Values in the database

Class Values	Car acceptability: unacc, acc, good, vgood
Attributes	Buying price: vhigh, high, med, low.
	Price of Maintenance: vhigh, high, med, low.
	Number of doors: 2, 3, 4, 5 or more.
	Capacity in terms of persons: 2, 4, more.
	Size of luggage boot: small, med, big.
	Safety Estimated: low, med, high.

instead, we have to consider all possible output values of the function. Given the frequencies of each one of the outputs, we can compute the entropy of the target function. To use a similar notation as in Section 3.1, we are considering a function $Y = g(x_1, .., x_6)$, where x_j is the attribute j in the Car Evaluation Dataset. Now, we can compute the entropy of the function $Y = g(x_1, .., x_6)$:

$$H(Y) = -\frac{1}{\ln(1728)}\left[\frac{69}{1728}\ln\left(\frac{69}{1728}\right) + \frac{65}{1728}\ln\left(\frac{65}{1728}\right) + \frac{384}{1728}\ln\left(\frac{384}{1728}\right) + \frac{1210}{1728}\ln\left(\frac{1210}{1728}\right)\right] =$$
$$= 0.1121 = 11.21\%$$

According to our hypothesis, this value should give us an order of magnitude of the minimal number of fitness cases needed to reconstruct the function. It should approximately be equal to the 11.21% of 1728, which is equal to 193.72 fitness cases.

The goal is to predict the value of the car, given the other six attributes. As fitness function, that we want to minimize, we will use the amount of incorrect predictions of an evolving model. In Table 1, the parameters of the GP algorithm are shown. We did some transformations on the attributes. Those variables that represented qualitative characteristics were transformed into Boolean variables or pairs of them. For example the *buying price* values in the dataset were {vhigh, high, med, low} and they became {(1, 1), (1, 0), (0, 1), (0, 0)}. The quantitative variables were adjusted to represent a number, for instance, the possible values for *number of doors* were $\{2, 3, 4, 5, \text{more}\}$ and became $\{2, 3, 4, 5, 6\}$.

In Figure 3, we report the results of 15 independent GP runs. For this example, which is more complex than the previous one, the conclusions are not as straightforward as for the examples presented earlier. Indeed, even when the algorithm is able to see all the cases at each generation, there is not a rapid convergence of the loss towards zero, but more of a constant, slower, improvement over time. This leads us to draw different conclusions. In both the 200 fitness cases and 500 fitness cases plots, even if the number of fitness cases is bigger than the entropy of the function, we do not have a rapid convergence of the algorithm, and additionally, we also have numerous oscillations. However, it is also true that in the 150 fitness cases plot, the oscillations are much more frequent and with bigger magnitudes. It should also be noted that the difference in the behavior of the functions is much more visible when we pass from 150 to 200 fitness cases, than when we increase this number to 500. In other words, we have clearer benefits with an increase of 50 than with an increase of 300 fitness cases.

All in all, also in this example, it seems that the entropy of the target function could play a significant role in the decision of the number of fitness cases to use. It is indeed a good result, but it is also true the results we have obtained on this test case leave room for doubts.

In order to compute the entropy of a function, the only values needed are the outputs of the function itself.

The elements of the domain seem to have no impact on this computation. While for the previous test cases this was not really a problem, for this particular dataset this fact may be

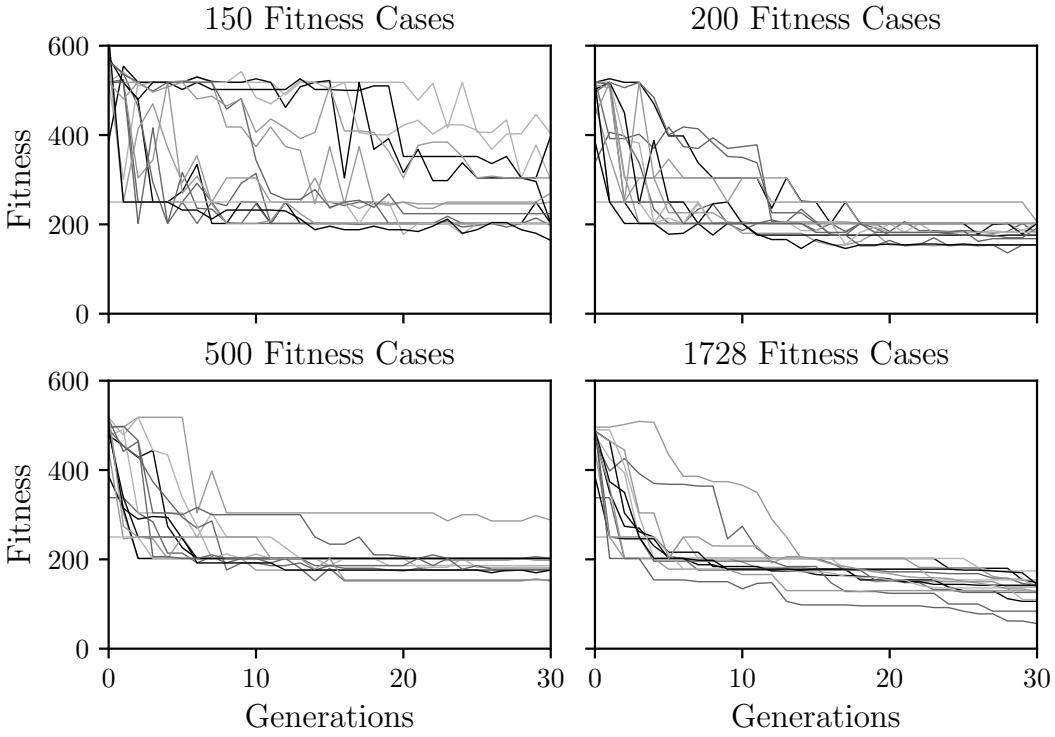

Figure 3: Fifteen sample GP runs for the dataset with 150, 200, 500, and 1728 fitness cases, respectively.

important. It actually seems reasonable that the distribution of the attributes in a dataset should also play an important role.

4.5 Benchmark Case Study

In this section, we use the following benchmark function taken from [7]:

$$e^{-x}x^3(\sin^2 x \cos x - 1)(\sin x \cos x) \tag{9}$$

This function is continuous and more complex than the previously studied ones. The fitness cases are the values of the function on 100 points equally distributed from 0.05 to 10, as proposed in [10], where the function was first used. This function, evaluated on a finite number of points, could be considered as a discrete one. Nonetheless, we cannot proceed as we did in the other examples since the function assumes 100 different values and that will give us an entropy equal to 100%. In an attempt to overcome this issue, as previously discussed in Section 3.1, we decided to group output values into bins, considering close values as if they were equal. By doing this, we could obtain a useful value for the entropy of the functions. In particular, we divided the graph of the function into bins of a size equal to 0.033, and we considered all the individuals belonging to the same bin as points with the same value, as can be seen in Figure 4. With this bin size, the entropy was 56.15%. We could then proceed to run the GP algorithm. In this example, we used as fitness function, to minimize, the mean absolute error between the true and the predicted values. As proposed in [10], the solutions are evaluated on a grid of points evenly spaced with an interval of 0.05, from -0.05 to 10.5 inclusive. Given the complexity of this problem, we decided to increase the computational effort compared to the previous experiments. So, we set the number of generations to 50 and the number of individuals in the population to 2000. The parameters we used can be found in Table 1.

In Figure 5, the results of ten different runs are reported. In the lower right plot, we can see how even with all the (100) available fitness cases, the algorithm did not always find the

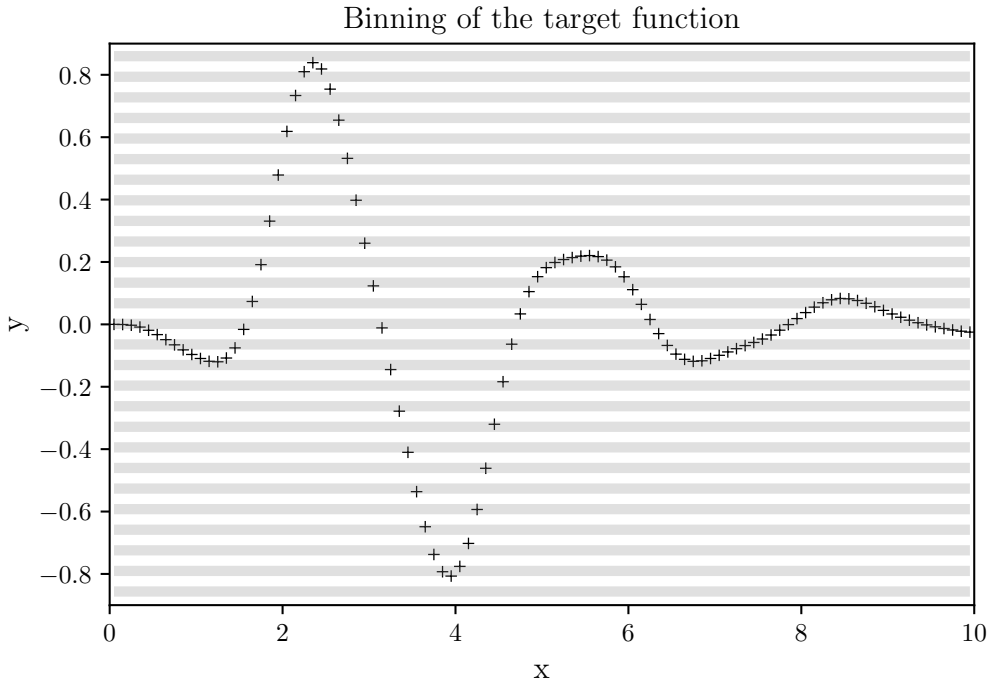

Figure 4: Discretization of the benchmark fitness cases with bins of size 0.033.

solution. For this reason, an error that is greater than zero with fewer fitness cases does not necessarily represent a negative result. However, while for the plot with 20 fitness cases the oscillations are visible (as are the converging behavior for the 100 fitness cases), the analysis of the other two plots is more complicated.

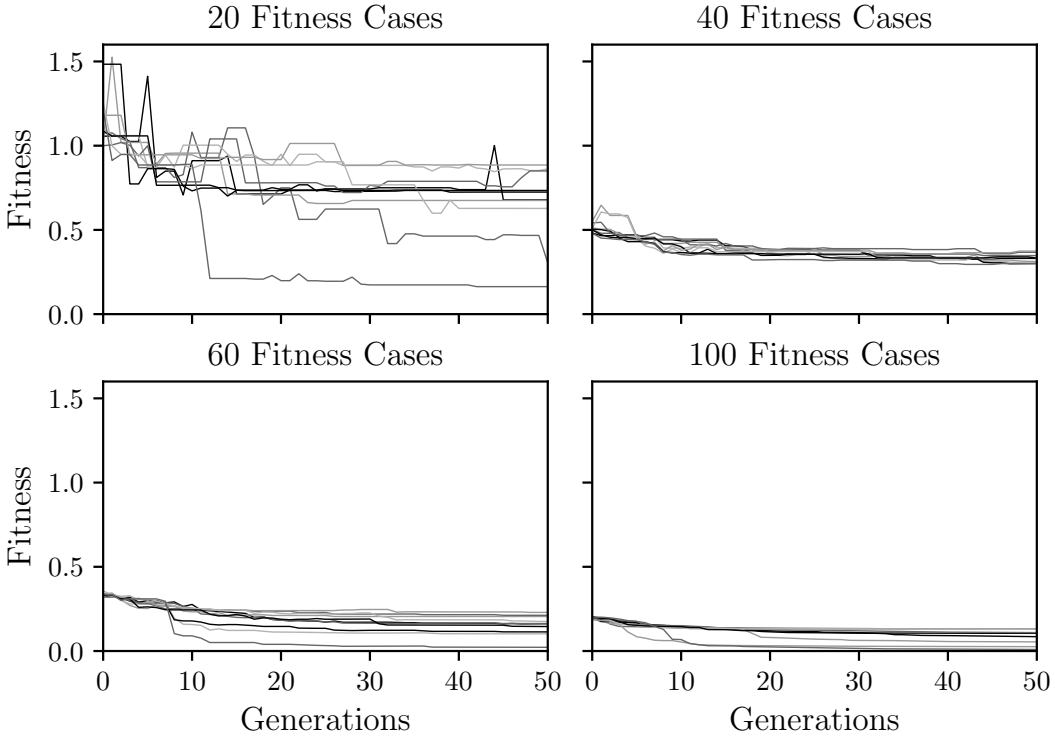

Figure 5: Ten sample GP runs for the benchmark function with 20, 40, 60, and 100 fitness cases, respectively. 2000 individuals.

In Figure 6, we can see a section of the plots analyzed before. Here, we are only considering generations from 5 to 20, with only 5 seeds to avoid graphical confusion (the results were similar for all the performed runs). We can observe that the 40 fitness cases plot shows some spikes and oscillations, while the 60 fitness cases plot has more of a converging behavior, apart from the different levels of errors that they are attaining. An increase of the error should indicate that the fitness cases that the algorithm is considering are not explanatory of the whole available information. Instead, an almost everywhere non-increasing trend could indicate that the size of the subset of fitness cases is sufficient to rebuild the whole function.

Since this example combines all the issues that arose until now, it seems a natural continuation to use it also to test the second proposed solution to the issues introduced in Section 4.2, and we will do it in Section 4.6.

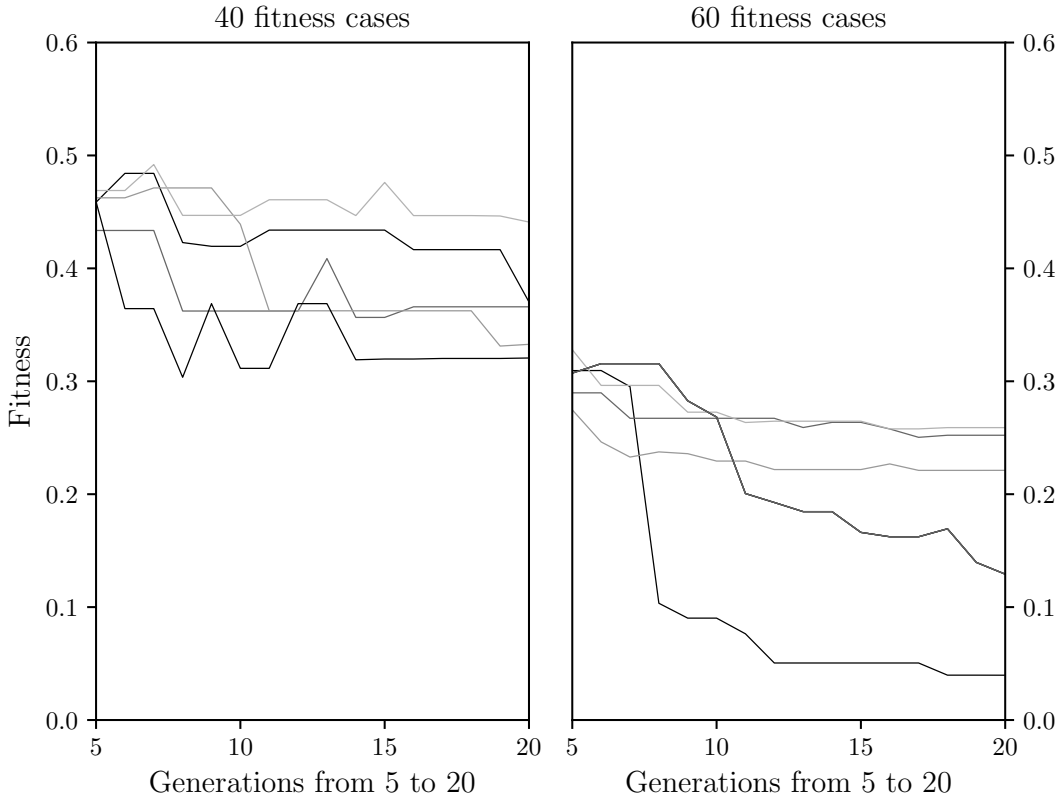

Figure 6: Zoom of figure 5. 5 seeds for the benchmark function with 40 and 60 fitness cases.

4.6 Choice of the Fitness Cases Subset

In this section, we choose a particular sample of the data every time that we need to use a subset of the available fitness cases. The idea is to take all the points corresponding to a change of trend, or concavity, of the function and add to these, if necessary, an additional number of points. The latter are evenly spaced points and the distance is chosen to achieve the desired number of fitness cases. The purpose is to provide the Evolutionary Algorithm with the critical points of the function, that can be considered as valuable data in reconstructing the function, intending to provide a common base of information to the runs with 20, 40, 60, or 100 fitness cases. In this way, we provide a method for the choice of the fitness cases while keeping the comparison between different settings meaningful. Since the four different sets of fitness cases share this group of critical points we do not favor any of the options considered. Thus, the differences obtained using sets of fitness cases of different sizes can be used to test, once again, our hypothesis.

In Figure 7, the results of the average fitness of the best individual in 10 different runs have been plotted. Remember that the best individual is found computing the fitness using the limited number of fitness cases (20, 40, or 60 fitness cases in this set). Then this best individual is tested on the whole dataset (100 fitness cases in this case) and the resulting value is reported. This time, we were able to use the average fitness since the set of fitness cases was fixed. In the figure, we can observe that the average performance of the algorithm with 60 fitness cases is closer to the one with 100 cases than to the one with 40 cases. That is, an addition of 20 data points, from 40 to 60, seems to be more beneficial than an addition of 40 data points, from 60 to 100. We conclude that our choice of the fitness cases is appropriate, since we gave all the

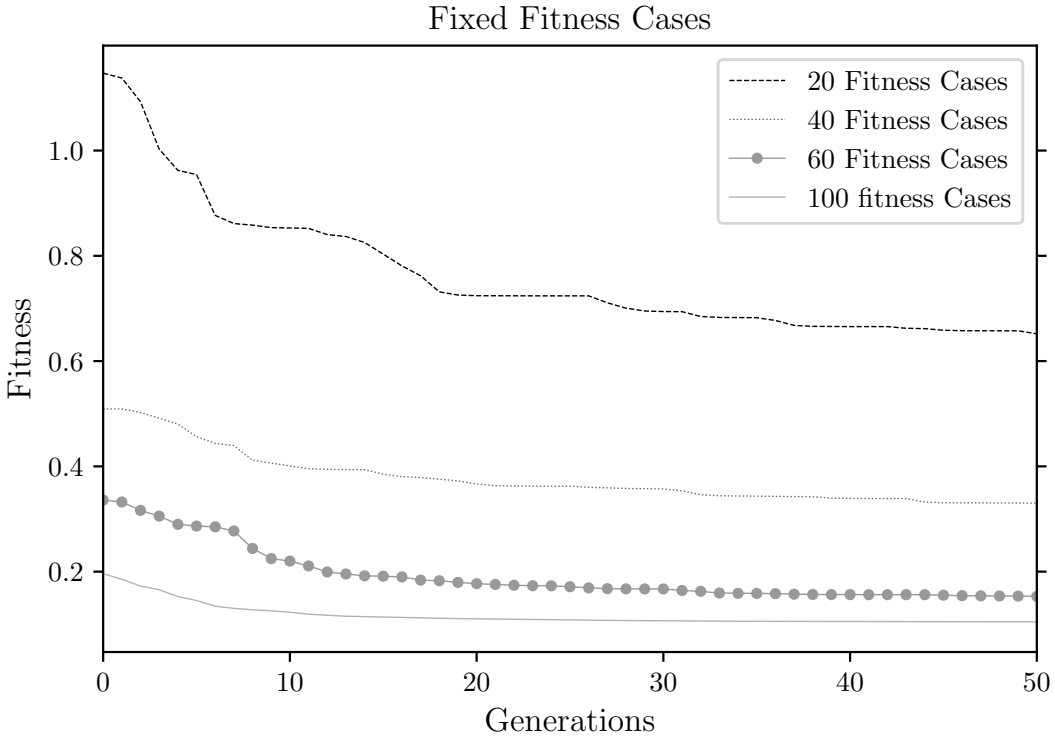

Figure 7: Average of ten runs with 20, 40, 60, and 100 fitness cases. 2000 individuals. The fitness cases here are fixed and not randomly chosen at each generation.

critical points of the function, and provided some points between them to aid the reconstruction of the function. With this in mind, we can say that the difference between the performance with 60 and 40 fitness cases could be caused by the fact that the available data between the critical points represent a relevant piece of information.

5 Conclusions

The objective of the presented work was to quantify the amount of training observations (fitness cases) that Genetic Programming (GP) needs to use, in order to generate models with a reasonable generalization ability. The presented idea was to treat the target function as a random variable and to compute its entropy. The hypothesis we wanted to test was that this entropy value would give us a faithful lower bound of the appropriate number of fitness cases.

As a first step, the hypothesis was tested on a simple Boolean function. Afterward, starting from a theoretical formulation, we studied how the fitness cases were used by the GP Algorithm. We then proposed a possible solution and applied it to the Boolean function already considered.

After that, we used two other test cases of increased complexity. The first one dealt with a Car Evaluation dataset. This allowed us to observe how considering various numbers of fitness

cases created differences even in non-trivial problems.

The last one targeted a function used in the literature as a benchmark test case. While we were dealing with this, we introduced and tested a method to adapt our hypothesis to continuous functions.

On all these problems, we obtained experimental results that corroborated our hypothesis, even though one important limitation emerged: for complex problems, the number of used fitness cases is not the only important piece of information. Another important step is the choice of the particular fitness cases to use. For the third test case studied, we proposed a method for the choice of the fitness cases that, although problem dependent, returned reasonable results.

In the future, we plan to develop a method to choose the appropriate fitness cases, that is problem independent. Furthermore, we plan to test our hypothesis on more real-life test problems, also extending the work to Machine Learning methods different from GP.

References

[1] Bi, Y., Xue, B., and Zhang, M. Using a small number of training instances in genetic programming for face image classification. *Information Sciences 593* (2022), 488–504.

[2] Cantú-Paz, E. Adaptive sampling for noisy problems. In *Genetic and Evolutionary Computation – GECCO 2004* (Berlin, Heidelberg, 2004), K. Deb, Ed., Springer Berlin Heidelberg, pp. 947–958.

[3] Chen, Q., Xue, B., and Zhang, M. Instance based transfer learning for genetic programming for symbolic regression. In *2019 IEEE Congress on Evolutionary Computation (CEC)* (2019), pp. 3006–3013.

[4] Ekárt, A. Shorter fitness preserving genetic programs. In *Artificial Evolution* (Berlin, Heidelberg, 2000), C. Fonlupt, J.-K. Hao, E. Lutton, M. Schoenauer, and E. Ronald, Eds., Springer Berlin Heidelberg, pp. 73–83.

[5] Galván-López, E., Vázquez-Mendoza, L., Schoenauer, M., and Trujillo, L. On the use of dynamic gp fitness cases in static and dynamic optimisation problems. In *Artificial Evolution* (Cham, 2018), E. Lutton, P. Legrand, P. Parrend, N. Monmarché, and M. Schoenauer, Eds., Springer International Publishing, pp. 72–87.

[6] Giacobini, M., Tomassini, M., and Vanneschi, L. Limiting the number of fitness cases in genetic programming using statistics. In *Parallel Problem Solving from Nature — PPSN VII* (Berlin, Heidelberg, 2002), J. J. M. Guervós, P. Adamidis, H.-G. Beyer, H.-P. Schwefel, and J.-L. Fernández-Villacañas, Eds., Springer Berlin Heidelberg, pp. 371–380.

[7] McDermott, J., White, D. R., Luke, S., Manzoni, L., Castelli, M., Vanneschi, L., Jaśkowski, W., Krawiec, K., Harper, R., De Jong, K., and O'Reilly, U. M. Genetic programming needs better benchmarks. In *GECCO'12 - Proceedings of the 14th International Conference on Genetic and Evolutionary Computation* (New York, New York, USA, 2012), ACM Press, pp. 791–798.

[8] Moraglio, A., Krawiec, K., and Johnson, C. G. Geometric semantic genetic programming. In *Parallel Problem Solving from Nature - PPSN XII* (Berlin, Heidelberg, 2012), C. A. C. Coello, V. Cutello, K. Deb, S. Forrest, G. Nicosia, and M. Pavone, Eds., Springer Berlin Heidelberg, pp. 21–31.

[9] Shannon, C. E. A mathematical theory of communication. *The Bell System Technical Journal 27*, 3 (1948), 379–423.

[10] Vladislavleva, E. J., Smits, G. F., and den Hertog, D. Order of nonlinearity as a complexity measure for models generated by symbolic regression via pareto genetic programming. *IEEE Transactions on Evolutionary Computation 13*, 2 (2009), 333–349.

[11] Wong, P., and Zhang, M. Scheme: Caching subtrees in genetic programming. In *2008 IEEE Congress on Evolutionary Computation (IEEE World Congress on Computational Intelligence)* (Hong Kong, China, 2008), IEEE, pp. 2678–2685.

[12] Xie, H., Zhang, M., and Andreae, P. Population clustering in genetic programming. In *Genetic Programming* (Berlin, Heidelberg, 2006), P. Collet, M. Tomassini, M. Ebner, S. Gustafson, and A. Ekárt, Eds., Springer Berlin Heidelberg, pp. 190–201.

[13] Zhang, B.-T., and Cho, D.-Y. Genetic programming with active data selection. In *Simulated Evolution and Learning* (Berlin, Heidelberg, 1999), B. McKay, X. Yao, C. S. Newton, J.-H. Kim, and T. Furuhashi, Eds., Springer Berlin Heidelberg, pp. 146–153.

[14] Ziegler, J., and Banzhaf, W. Decreasing the number of evaluations in evolutionary algorithms by using a meta-model of the fitness function. In *Genetic Programming* (Berlin, Heidelberg, 2003), C. Ryan, T. Soule, M. Keijzer, E. Tsang, R. Poli, and E. Costa, Eds., Springer Berlin Heidelberg, pp. 264–275.