Models of Performance of Evolutionary Program Induction Algorithms Based on Indicators of Problem Difficulty

Mario Graff  
mgraffg@dep.fie.umich.mx  
Facultad de Ingeniería Electrónica, Universidad Michoacana de San Nicolás de Hidalgo, Mexico

Riccardo Poli  
rpoli@essex.ac.uk  
School of Computer Science and Electronic Engineering, University of Essex, UK

Juan J. Flores  
juanf@umich.mx  
Facultad de Ingeniería Electrónica, Universidad Michoacana de San Nicolás de Hidalgo, Mexico

doi:10.1162/EVCO_a_00096

Abstract

Modeling the behavior of algorithms is the realm of evolutionary algorithm theory. From a practitioner’s point of view, theory must provide some guidelines regarding which algorithm/parameters to use in order to solve a particular problem. Unfortunately, most theoretical models of evolutionary algorithms are difficult to apply to realistic situations. However, in recent work (Graff and Poli, 2008, 2010), where we developed a method to practically estimate the performance of evolutionary program-induction algorithms (EPAs), we started addressing this issue. The method was quite general; however, it suffered from some limitations: it required the identification of a set of reference problems, it required hand picking a distance measure in each particular domain, and the resulting models were opaque, typically being linear combinations of 100 features or more. In this paper, we propose a significant improvement of this technique that overcomes the three limitations of our previous method. We achieve this through the use of a novel set of features for assessing problem difficulty for EPAs which are very general, essentially based on the notion of finite difference. To show the capabilities of our technique and to compare it with our previous performance models, we create models for the same two important classes of problems—symbolic regression on rational functions and Boolean function induction—used in our previous work. We model a variety of EPAs. The comparison showed that for the majority of the algorithms and problem classes, the new method produced much simpler and more accurate models than before. To further illustrate the practicality of the technique and its generality (beyond EPAs), we have also used it to predict the performance of both autoregressive models and EPAs on the problem of wind speed forecasting, obtaining simpler and more accurate models that outperform in all cases our previous performance models.

Keywords

Evolutionary program-induction algorithms, performance forecasting, hardness measures, genetic programming, wind speed forecasting.

1 Introduction

Evolutionary Program-induction Algorithms (EPAs) are evolutionary search techniques that aim at the automatic evolution of computer programs. Under this rubric, we find
genetic programming (GP; Koza, 1992; Langdon and Poli, 2002; Poli et al., 2008), Cartesian GP (CGP; Miller and Thomson, 2000), grammatical evolution (O’Neill and Ryan, 2001), and gene expression programming (GEP; Ferreira, 2001), among others.

Given the proven effectiveness of EPAs (see, e.g., Koza et al., 2003; Poli et al., 2008; Koza, 2010), there has been an enormous increase in interest in EPAs over the last two decades, which has resulted in the appearance of a plethora of new generic operations, new representations, and, of course, a variety of new problems.

In principle, one would expect theory to be able to shed light on old and new EPA techniques and problems. However, this expectation has only partially been met by reality. Developing EPA theory is objectively a very hard and slow process where the precise details of the algorithm and the problem matter a great deal (see Langdon and Poli, 2002; Poli and McPhee, 2003). In practice, exact theoretical EPA models are hard to apply to realistic situations.

This has important consequences from a practitioner’s point of view. For instance, one is rarely able to estimate the performance of an algorithm on a particular problem without running the algorithms on that problem. Thus, we cannot a priori (i.e., without experimentation) anticipate which is the most suitable algorithm to solve a particular problem out of a set of different algorithms, nor can we say which set of parameters would work best for one particular algorithm. The current lack of theoretically sound guidelines forces practitioners to hand-tune algorithms, parameters, and operators by trial and error.

In recent work, Graff and Poli (2008, 2010) started filling this gap, developing a method to practically estimate the performance of EPAs. The method was quite general, being based on the notion of using the similarities between a set of reference problems and the problem at hand as features which were then linearly combined to produce a performance estimate. The method was able to accurately model different EPAs as well as an artificial neural network and two bin packing heuristics on different classes of problems. Nonetheless, in each particular domain it required the identification of the reference problems, the coefficient associated with each reference problem, and also a distance measure to compute similarities. Also, when tested on two significant classes of problems—symbolic regression on rational functions and Boolean function induction—we found that models including some on the order of 100 features were required for our performance predictors to work well. This meant that, while models were accurate, it was hard to infer what makes a particular problem easy or hard for an EPA.

In this paper, we propose a significant improvement of this technique that does not require the identification of reference problems; that does not use a large number of features to produce good predictions; and that does not require choosing a distance measure, thereby overcoming the three limitations of our previous method.

The proposed method is effectively an integration of the approach mentioned above and of the models used for performance prediction in the literature on the algorithm selection problem—the problem of deciding which algorithm to use to solve a particular problem from a set of available algorithms. Models used for algorithm selection are typically based on problem-domain specific features that experts in the problem domain consider to be useful to measure the hardness of a problem, while our previous models were based on the general notion of distance between problems. In the paper, we will

---

1 Commonly, GP is also used to refer to many EPAs (see e.g., Poli et al., 2008); however, in this paper the term GP refers only to tree-based GP.
take the approach used in the algorithm selection literature and propose a set of features for assessing problem difficulty for program-induction algorithms. However, as in our previous models, the features we propose are very general, essentially based on the notion of finite difference. Also, as before, linear functions of such features will be used as performance estimators and a form of regression will be used for learning (setting the coefficients of a model). It is worth noting that a different methodology such as artificial neural networks could be used to combine these features; however, we decided to use linear functions, since they worked well in our previous approach.

To show the capabilities of our technique and compare it with our previous performance models, we create models for two important classes of problems—symbolic regression on rational functions and Boolean function induction—using two performance measures—expected end-of-run fitness and the success rate—as we did in our previous work. We modeled a variety of EPAs including: different versions of GP, GEP, stochastic iterated hill climbing (SIHC) in program space and one version of CGP. The comparison showed that for the majority of the algorithms and problem classes, the new method produced much simpler models which predicted performance 26% better than before.

Furthermore, to start transferring our techniques from the class of benchmark problems to more practical domains and to illustrate the generality of the method, in this contribution we show that the models proposed here can be used to predict the performance of different EPAs on the problem of wind speed forecasting. This problem has been a very active area of research in the field of renewable energy (see, e.g., a recent review by Foley et al., 2012). This problem allows us to demonstrate a further feature of our modeling technique. The technique is not restricted to model EPAs; it can also be applied to other types of systems. We illustrate this by modeling the performance of different auto-regressive models applied to the wind speed forecasting problem.

The rest of the paper is organized as follows. In Section 2, we review related work. Section 3 presents our modeling technique and the process to apply it. The problems and algorithms used to illustrate the capabilities of our modeling technique are described in Sections 4 and 5, respectively. Section 6 provides our experimental results. Some conclusions and possible directions for future work are given in Section 7.

2 Related Work

In this section, we will review previous work relevant to the EPA performance-modeling technique proposed in this paper. We first look at the long tradition of research in the area of problem difficulty in Section 2.1, then we consider models developed in the algorithm selection literature in Section 2.2, and, finally, Section 2.3 reviews our previous EPA performance-modeling technique.

2.1 Problem Hardness in Evolutionary Computation

Our work is related to the problem of understanding what makes a problem easy or hard for EAs. Problem-difficulty studies in EAs focused initially on the building-block hypothesis for GAs (Goldberg, 1989) and the related notion of deception (Deb and Goldberg, 1993). Later approaches used the notion of fitness landscape (originally proposed by Wright, 1932) to study the influence of multimodality on problem difficulty (Horn and Goldberg, 1995). The fitness landscape metaphor can be helpful in understanding the difficulty of a problem. It is straightforward, for instance, that search in a smooth landscape will be relatively easy for most algorithms, while a rugged landscape with many local optima may be more problematic. Unfortunately, given the size of typical
search spaces, it is generally impossible to have a graphical representation of a fitness landscape. Thus, one really needs to condense useful information of fitness landscapes into one or a few numeric descriptors.

Jones and Forrest (1995) introduced a heuristic called fitness distance correlation ($f_{dc}$), as an algebraic indicator of problem difficulty for GAs. The study of $f_{dc}$ has been extended to GP (e.g., in Clergue et al., 2002; Vanneschi, Tomassini, Collard, et al., 2003; Vanneschi and Tomassini, 2003; Vanneschi, Tomassini, Clergue, et al., 2003; Vanneschi, 2004; Tomassini et al., 2005). These studies show that $f_{dc}$ is often quite a reliable indicator of problem hardness. However, it has some weaknesses, the most severe of which is that the computation of $f_{dc}$ requires that an optimal solution (or solutions) are known beforehand. This is obviously unrealistic and prevents one from using $f_{dc}$ to estimate how hard a problem is in practical applications. A measure that does not require knowledge of optimal solution(s) is the negative slope coefficient ($n_{sc}$). It was introduced and tested in Vanneschi et al. (2004); Vanneschi (2004); and Vanneschi et al. (2005, 2006). The $n_{sc}$ is based in the concept of a fitness cloud (i.e., a scatter plot of parent/offspring fitness pairs). The $n_{sc}$ has been proven to be a reliable measure for a number of different benchmark problems in GP, including the multiplexer, the intertwined spiral problem, six instances of the royal tree problem, the artificial ant on the Santa Fe trail, the even-parity problem, one particular instance of symbolic regression, and a trap function (Vanneschi et al., 2004; Vanneschi, 2004; Vanneschi et al., 2005, 2006). A slightly modified version of $n_{sc}$ called fitness proportional $n_{sc}$ has also given good results in GAs, for the class of invertible functions of unitation (Poli and Vanneschi, 2007).

Another difficulty indicator that has been used lately to assess problem hardness in GP is locality as defined by Hutchison et al. (2006) and Galván-López et al. (2011). Locality is a measure of the agreement between the genotypic and phenotypic neighborhood structures. For example, a smooth landscape has a high locality, while a rough one has a low locality.

While the $f_{dc}$, $n_{sc}$, and locality have shown some success at providing insights on how hard or easy a problem is, they do not really provide an estimation of the performance of an algorithm. For instance, they are unable to predict the success rate of a particular evolutionary algorithm or the fitness of the best solution found in a run of the algorithm.

### 2.2 Algorithm Selection

Moving away from the evolutionary computation literature, the models developed in this paper are related to the algorithm selection problem (Rice, 1976) and algorithm portfolios (Leyton-Brown et al., 2009, 2006; Hutter et al., 2006; Nudelman et al., 2004; Leyton-Brown et al., 2003a, 2003b, 2002; Xu et al., 2008, 2007a, 2007b). The algorithm selection problem is the problem of deciding which algorithm to use to solve a particular problem from a set of available algorithms. An algorithm portfolio is a particular way of solving the algorithm selection problem based on choosing a collection of algorithms that are run in parallel or in sequence to solve a particular problem. The connection between these and the models presented in this paper is that in order to decide which algorithm to use (i.e., solve the algorithm selection problem) and/or to build an algorithm portfolio, one needs to first instantiate and then use a model of performance for each algorithm involved in the selection process or in the portfolio. Generally, these models are built using machine learning techniques. The models then are used to predict which algorithm from the collection will have the best performance, starting from a description of the problem to be solved.
In general, in order to obtain good portfolios, one needs (Smith-Miles, 2008): (1) a large collection of problem instances of variable complexity; (2) a diverse set of algorithms, each having the best performance on some such instances; (3) a set of features that describe the properties of the problems in the collection; (4) a good machine learning technique. It is interesting to note that a machine learning technique is used to model an algorithm which might be another machine learning technique. The difference between these two is that the model is built using only features that characterize the problems being solved, while a machine learning technique uses the characteristics of the objects (that composed the problem) to perform a classification or regression task.

Creating a diverse set of algorithms is not an easy task. For example, Xu et al. (2010) developed an algorithm to create portfolios called Hydra. Hydra optimizes the parameters of the algorithms included in the portfolio to produce different performances in these algorithms. For the cases tested in Xu et al., this diversity was obtained by using different algorithms with different parameter settings. On the other hand, for the case of stochastic algorithms, one needs also to take into consideration that the performance is typically measured doing multiple independent runs and taking the average (or some other statistical measure of tendency, such as the median). As expected, this introduces noise in the algorithm selection process and so it must be kept under control. Here, we decided to perform 100 independent runs to have a robust estimation of the performance.

The machine learning techniques used have taken a variety of forms including: linear equations (Boyan and Moore, 1998; Telelis and Stamatopoulos, 2001; Nudelman et al., 2004; Leyton-Brown et al., 2009, 2006, 2002, 2003a, 2003b), linear regression and a mixture of experts (Xu et al., 2008, 2007a, 2007b), Markov decision processes (Lagoudakis and Littman, 2001, 2000), Bayesian models (Horvitz et al., 2001), and case-based reasoning (Gebruers et al., 2005, 2004; Gebruers and Guerri, 2004), among others.

There have been a number of different problems where the algorithm selection problem has been tackled successfully. These include: matrix multiplication (Vuduc et al., 2001; Thomas et al., 2005), sorting (Brewer, 1995; Vuduc et al., 2001; Thomas et al., 2005), solving partial differential equations (Brewer, 1995), signal processing (Singer and Veloso, 2000), SAT (e.g., Xu et al., 2008, 2007a, 2007b), auction winner determination (e.g., Leyton-Brown et al., 2009, 2006, 2002, 2003a, 2003b), and constraint programming (e.g., Gebruers et al., 2005, 2004; and Gebruers and Guerri, 2004), among others (see also Smith-Miles (2008) for a detailed discussion of how the algorithm selection problem was tackled in different fields).

We should also note that a form of metalearning—predicting how each of a set of machine learning methods would perform on a learning task and then choosing the most appropriate method for the data—is a special case of the algorithm selection problem, in which the problem to be solved is a learning problem, such as classification or forecasting. This is an area in which there has been significant progress in recent years (see, e.g., the reviews in Vilalta and Drissi, 2002; and Schaul and Schmidhuber, 2010).

Relevant for this paper is the algorithm-selection/metalearning work on predicting which forecasting algorithm will do best in time series forecasting, since we will apply EPAs and autoregressive models to a wind speed forecasting problem later on in the paper. Again, solving these problems requires some form of learning to construct models of the algorithms available for the task. These models do not need to be numeric. For example, Wang et al. (2009) used a rule-based system as their model and used a rule-induction algorithm to “train” it. Features used are normally those found by experts.
to help in the forecasting task. These include: trend, seasonality, periodicity, serial correlation, kurtosis, nonlinearity, self-similarity, and chaos.

We should note that, despite these successes, before our own work (described below), no attempt to tackle the problem of selecting EPAs had been reported in the literature.

2.3 EPA Performance Models Based on Problem Distances

Recently, the construction of accurate models of EPAs and their use in the algorithm portfolios composed by a number of different EPAs was described in Graff and Poli (2008, 2010). The approach was successfully tested on two classes of problems, namely symbolic regression of rational functions\(^2\) and Boolean induction problems. The key behind the model used there was to predict the performance, \(P(t)\), of an EPA based on the similarity between the problem, \(t\), to be faced by the algorithm and a set of reference problems, \(S\), previously selected. In formula representation, this is

\[
P(t) \approx a_0 + \sum_{p \in S} a_p \cdot d(p, t),
\]

where \(d\) is a similarity measure and \(a_p\) are coefficients. The reference problems and the coefficients were determined using a training set of pairs \((f, P(f))\), where \(f\) represents a problem and \(P(f)\) is the performance of the EPA on \(f\), while the similarity (or distance) measure \(d\) was determined by trial and error. The number of terms, the reference problems, and the coefficients associated with each reference problem were determined via an algorithm that included multiple cycles of least angle regression (LARS; Efron et al., 2004) embedded within a cross-validation loop.

We assessed model performance via the generalization error measured using the relative square error (RSE; Alpaydin, 2004), defined as

\[
\text{RSE} = \frac{\sum_{i} (\bar{P} - \tilde{P}_i)^2}{\sum_{i} (\bar{P}_i - \bar{P})^2},
\]

where \(i\) ranges over either a training set or a validation set, \(\bar{P}_i\) is the average performance recorded for problem \(i\), \(\tilde{P}_i\) is the performance predicted by the model, and \(\bar{P}\) is the average performance over all problems in a set. Values of RSE less than 1 indicate that a model can predict better than the mean.

The performance prediction method worked well. The average RSE over 20 EPAs assessed on an independent validation set was 0.34 for the prediction of the best of run fitness in symbolic regression tasks originating from sampling random rational functions. Also, the average RSE over 20 EPAs assessed on an independent validation set was 0.42 for the prediction of the success rate achieved with random Boolean induction problems.

However, the method suffered from some limitations: it required the identification of a set of reference problems, it required hand picking a distance measure for problems in each particular domain, and the resulting models were opaque, since they were typically linear combinations of many features. In particular, over the 20 EPAs assessed, on average approximately 117 reference problems were required in the case of symbolic regression of rational functions and approximately 118 were required on average for Boolean induction problems.

We believe that this last issue is probably the result of using very general features (the similarity between the given problems and a set of reference problems), each

\(^2\)We decided to use rational functions, instead of polynomial functions, because of the greater diversity of functions that can be generated with low-degree rational functions.
perhaps not as discriminative as the typical difficulty indicators used in EAs and in other domains (e.g., like SAT). It then stands to reason that significant improvements of accuracy of EPA performance models and reductions in the number of features they require would result from following a more traditional algorithm-selection route and base model predictions on difficulty indicators. The question, naturally, is: what difficulty indicators would work well for quantitatively predicting the performance of program-induction algorithms?

3 Modeling EPA Performance using Difficulty Indicators

As we have mentioned previously, the majority of models used for the algorithm selection problem are based on problem-specific features that experts in the problem domain consider to be useful to measure the hardness of a problem. As we mentioned above, the $f_{dc}$, $nsc$, and locality have all been proposed as indicators of the hardness of problems in GP. Based on this, it is reasonable to ask whether these difficulty indicators are suitable to form performance models. This issue is addressed in Section 3.1. Section 3.2 describes two difficulty indicators for Boolean functions that are suitable for creating models of performance. In Section 3.3, we generalize these difficulty indicators from the Boolean domain to the domain of continuous functions. Section 3.4 presents our modeling technique.

3.1 Can NSC, FDC, and Locality Be Used to Quantitatively Model EPA Performance?

The first candidates we considered as features to be used for the creation of models of performance are the three indicators of problem difficulty proposed for EPAs, namely $f_{dc}$, $nsc$, and locality, which we reviewed in Section 2.1. Unfortunately, we found that all three have some features that limit their applicability in the creation of performance models.

The main limitation of the $f_{dc}$, as mentioned previously, is that it requires the knowledge of the solutions of a problem. Therefore, we cannot use it to create practical models of performance for realistic problems (where the optima are not known).

The $nsc$ and locality do not present this limitation. However, all indicators (including $f_{dc}$) are computed through a stochastic procedure which involves sampling the search space. As a consequence, they inject noise both in the learning process (more specifically, in the construction of training and validation sets) and, later, in the process of estimating performance (effectively acting as noisy inputs to the predictor). Of course these errors can be kept under control. The approach typically followed by researchers is, indeed, performing extensive sampling; but this leads us to a further problem: the computational cost required to estimate these difficulty indicators is very heavy in realistic scenarios.

There is one final issue. All three difficulty indicators, as original proposed, provide a single measurement for each problem tested. This is exactly what one would want for a qualitative analysis of problem difficulty. However, based on the literature on the algorithm selection problem, the use of only two difficulty indicators (remember, we could only use $nsc$ and locality due to $f_{dc}$'s first limitation) as independent variables in a quantitative model of performance would be unlikely to result in good predictions, particularly considering the complexity of predicting the performance of EPAs.

Thus, the identification of a number of reliable problem difficulty indicators is a prerequisite for the creation of a good performance model of EPAs, irrespective of
whether we used nsc and locality for this purpose. Below we propose a class of indicators that are general, efficient, practical, and reliable.

### 3.2 Difficulty Indicators for Boolean Functions

Our EPA difficulty indicators took inspiration from the work of Franco (2006), who recently proposed an indicator for the difficulty of learning Boolean functions (in terms of the number of training epochs required) for feed-forward artificial neural networks.

Franco’s indicator is based on the idea that the more a function’s output varies as its inputs are varied, the more difficult it is to learn. He, thus, estimated difficulty by evaluating the Boolean function to be learned for different pairs of inputs and calculating the proportion of such pairs for which the function produced different outputs. This process was applied to every pair of input patterns that are at Hamming distance of 1 and 2. Franco’s indicator is defined as:

\[
\tilde{\vartheta}(f) = \vartheta_1(f) + \vartheta_2(f),
\]

where \(\vartheta_i(f)\) is the proportion of pairs of outputs that are different when input patterns having a Hamming distance of \(i\) are used. That is,

\[
\vartheta_i(f) = \frac{1}{2^N \times \binom{N}{i}} \sum_{j \in I} \sum_{\{k \in I : H(j,k) = i\}} |f(j) - f(k)|,
\]

where \(f(j)\) is the output of Boolean function \(f\) for input \(j\), \(N\) is the number of variables, \(I\) contains all the possible input patterns (e.g., for three variables \(I\) contains eight different input patterns), \(H(j,k)\) is the Hamming distance between inputs \(j\) and \(k\), and the term \(1/(2^N \times \binom{N}{i})\) is a normalization factor.

Let us illustrate the definition of \(\vartheta_i\) with an example. Let \(f\) be the Boolean function defined in Table 1. For a function of three variables, only \(\vartheta_1\), \(\vartheta_2\), and \(\vartheta_3\) can be computed (there cannot be any input pattern that is at a Hamming distance of more than three from another). Let us for example compute \(\vartheta_3\). This is proportional to the sum over the search space \(\{0,1\}^3\) of the sum over all points that are at a Hamming distance of three from a given point. Of course with three variables, there is only one such point. So (ignoring scaling factors)

\[
\vartheta_3 \propto 2|f_1 - f_8| + 2|f_2 - f_7| + 2|f_3 - f_6| + 2|f_4 - f_5| \\
= 2|1 - 1| + 2|1 - 0| + 2|0 - 0| + 2|1 - 0| = 4.
\]
In Franco (2006) the indicator $\tilde{\vartheta}(f)$ was found to correlate well with the time needed (i.e., number of epochs) by a learning algorithm to train a feed-forward neural network. Based on this, we decide to explore whether Franco’s idea could also be used to assess the hardness of Boolean induction problems that are also in relation to EPAs. Since the active ingredients in Franco’s difficulty indicator are the terms $\vartheta_i(f)$, we decided to include these terms in our set of features in the creation of models of performance for EPAs.

We should note that difficulty indicator $\vartheta_1$ is related to the concept of differences (or derivative) of Boolean functions (Akers, 1959). The (first order) difference of a Boolean function $f(x)$, where $x = (x_1, \ldots, x_N)$, with respect to the variable $x_j$ is defined as

$$\triangle f_j(x) = f(x) \oplus f(x \oplus e_j),$$

where $e_j$ is a unit vector whose $j$th component is 1, and $\oplus$ is the exclusive-or operator. Then it is easy to see that we can rewrite Equation (3) as

$$\vartheta_1(f) = \frac{1}{2^N} \sum_{x \in I} \sum_{j=1}^{N} \triangle f_j(x).$$

In other words, $\vartheta_1$ can be interpreted as the average (across the variables $x_1, \ldots, x_N$) of the average (across all possible inputs) of the first order derivatives of a Boolean function.

Having seen that $\vartheta_1$ is proportional to the sum of differences of the Boolean function at hand (Equation (4)), it is reasonable to investigate whether sums of higher-order differences of Boolean functions are also suitable features to create models of performance. A second order difference of a Boolean function is defined as the difference of the difference of the function. That is,

$$\triangle f_{i,x_j}(x) = \triangle(\triangle f_j)(x)$$

$$= \triangle f_j(x) \oplus \triangle f_j(x \oplus e_j)$$

$$= (f(x) \oplus f(x \oplus e_j)) \oplus (f(x \oplus e_j) \oplus f((x \oplus e_j) \oplus e_j))$$

$$= f(x) \oplus f(x \oplus e_j) \oplus f(x \oplus e_j) \oplus f(x \oplus e_j \oplus e_j)$$

Higher order differences are similarly defined. We define our new features for the modeling performance of EPAs as follows:

$$\varphi_3(f) = \sum_{x \in I} \sum_{a \in C(i)} \triangle f_a(x),$$

where $C(i)$ is the set of all multisets of cardinality $i$ generated from $N$ Boolean variables. For example, for $N = 4$, $C(1)$ is defined as $\{x_1, \ldots, x_4\}$, $C(2) = \{x_1x_1, x_1x_2, \ldots, x_4x_4\}$, and so on.\(^3\)

Let us again illustrate the calculations involved in these new features with the sample function in Table 1. Let us compute $\varphi_3$. In this case, one needs to derive function $f$ three times with respect to all possible combinations of variables and then sum over all possible inputs. However, of course, only the derivative with respect to $a$, $b$, and $c$ (in any order) can be different from 0. Table 2(a) shows $f_{ab}$, that is, the discrete derivative of $f$ with respect to the variable $a$. Table 2(b) shows the derivative of $f_{ab}$ with respect to the variable $b$, that is, $f_{abc}$. Finally, the derivative of $f_{abc}$ with respect to $c$ is given in Table 2(c). Since $f_{abc} \equiv \perp$, we then have that $\varphi_3 = 0$.

\(^3\)Note that since $\triangle f_{i,x_j}(x) = 0$ if $i \in \{1, \ldots, N\}$, many terms in Equation (5) are 0. So $C(i)$ could more effectively be defined as the set of all sets of cardinality $i$ generated from $N$ Boolean variables.
Table 2: Higher-order differences of the function in Table 1: (a) \( f_a = f(x) \oplus f(x \oplus e_1) \) where \( e_1 = (1, 0, 0) \), (b) \( f_{ab} = f_a(x) \oplus f_a(x \oplus e_2) \) where \( e_2 = (0, 1, 0) \), and (c) \( f_{abc} = f_{ab}(x) \oplus f_{ab}(x \oplus e_3) \) where \( e_3 = (0, 0, 1) \).

<p>| | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>x</td>
<td>f(x)</td>
<td>x \oplus e_1</td>
<td>f(x \oplus e_1)</td>
<td>f_a</td>
</tr>
<tr>
<td>000</td>
<td>1</td>
<td>100</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>001</td>
<td>1</td>
<td>101</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>010</td>
<td>0</td>
<td>110</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>011</td>
<td>1</td>
<td>111</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>100</td>
<td>0</td>
<td>000</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>101</td>
<td>0</td>
<td>001</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>110</td>
<td>0</td>
<td>010</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>111</td>
<td>1</td>
<td>011</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

<p>| | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>x</td>
<td>f_a(x)</td>
<td>x \oplus e_2</td>
<td>f_a(x \oplus e_2)</td>
<td>f_{ab}</td>
</tr>
<tr>
<td>000</td>
<td>1</td>
<td>010</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>001</td>
<td>1</td>
<td>011</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>010</td>
<td>0</td>
<td>000</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>011</td>
<td>0</td>
<td>001</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>100</td>
<td>1</td>
<td>110</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>101</td>
<td>1</td>
<td>111</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>110</td>
<td>0</td>
<td>100</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>111</td>
<td>0</td>
<td>101</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

<p>| | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>x</td>
<td>f_{ab}(x)</td>
<td>x \oplus e_3</td>
<td>f_{ab}(x \oplus e_3)</td>
<td>f_{abc}</td>
</tr>
<tr>
<td>000</td>
<td>1</td>
<td>001</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>001</td>
<td>1</td>
<td>000</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>010</td>
<td>1</td>
<td>011</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>011</td>
<td>1</td>
<td>010</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>100</td>
<td>1</td>
<td>101</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>101</td>
<td>1</td>
<td>100</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>110</td>
<td>1</td>
<td>111</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>111</td>
<td>1</td>
<td>110</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

### 3.3 Difficulty Indicators for Continuous Functions

Clearly, the aforementioned difficulty indicators can only be used for Boolean functions (we cannot meaningfully define the Hamming distance and the exclusive-or in continuous spaces). However, the ideas behind such difficulty indicators can be extended to continuous functions. In other words, we can derive similarly-inspired difficulty indicators for the problem of inducing continuous functions from a training set of input-output pairs: a problem normally termed *symbolic regression* in GP.
Let us start from generalizing the notion of Boolean difference. A natural counterpart for it in the continuous domain is the discrete derivative. The discrete derivative of function $f$ w.r.t. a variable $x_i$ can be written as

$$\Delta_h f_{x_i} = \frac{f(x + he_i) - f(x)}{h}, \quad (6)$$

where $x$ is the independent variable, $h$ is the step, and $e_i$ is a unit vector whose $i$th component is 1 (and all other components are 0).

Based on this equivalence, let us now see if we can extend the $\vartheta_i$ difficulty indicators. Note that the term $f(x + he_i) - f(x)$ in Equation (6) matches the term $f(j) - f(k)$ of Equation (3). This suggests that one might be able to generalize Equation (3) to continuous spaces if we grouped the training set inputs according to their distance, $h_i$.

To achieve this, we create groups of pairs of inputs (from the training set) having distances $h_1, h_2, \ldots$, where $h_i < h_{i+1}$. This process continues until the maximum distance between pairs of input patterns is reached. Then we define our first difficulty indicator\(^4\) for continuous functions as:

$$\vartheta_i(f) = \sum_{x \in I} \sum_j |\Delta_{h_i} f_{x_j}(x)|, \quad (7)$$

where $I$ contains all the input patterns and $N$ is the number of independent variables.

Let us illustrate the definition of $\vartheta_i(f)$ with an example. Let $f$ be the symbolic regression problem defined in Table 3. In this case, the number of variables, $N$, is 2. Since variable $x_1$ takes values from the set $\{-0.5, 0, 0.5\}$ and variable $x_2$ takes values from the set $\{-0.5, 0, 0.5\}$, the set $I$ is composed of nine pairs of values obtained by the Cartesian product of such sets. That is, $I = \{(-0.5, -0.5), (0, -0.5), \ldots, (0.5, 0.5)\}$. Let us now term $x_{ik}$ the values that variable $x_i$ can take. The distances $h_i$ are all the different values greater than zero of $\{|x_{il} - x_{lk}| : \forall l, k\}$. For this case, the distances can only take

---

\(^4\)More specifically, our difficulty indicator measures the ruggedness of the function. Indirectly, it is also correlated with the difficulty of the problem; however, this is not the case for all functions.
two values: \( h_1 = 0.5 \) and \( h_2 = 1.0 \). Then \( \varphi_1(f) \) is computed as follows:

\[
\varphi_1(f) = \sum_{x \in I} |\Delta h_1 f_x(x)| + |\Delta h_1 f_y(x)|
\]

\[
= \sum_{x \in I} \left| \frac{f(x + (0.5, 0)) - f(x)}{0.5} \right| + \left| \frac{f(x + (0.5, 0)) - f(x)}{0.5} \right|
\]

\[
= \frac{1}{0.5} \sum_{x \in I} |f(x + (0.5, 0)) - f(x)| + |f(x + (0.5, 0)) - f(x)|.
\]

For the purpose of illustration, let us compute the value of \( |f(x + (0.5, 0)) - f(x)| \) when \( x \) is \((-0.5, -0.5)\) and \((0.5, 0.5)\). For \( x = (0.5, 0) \) we obtain \(|f((-0.5, -0.5) + (0.5, 0)) - f(-0.5, -0.5)| = |f(0, 0.5) - f(-0.5, -0.5)| = |0.5 - 0.5| = 0.5 \). For \( x = (0.5, 0.5) \), instead we obtain \(|f((0.5, 0.5) + (0.5, 0)) - f(0.5, 0.5)| = |f(1.0, 0.5) - f(0.5, 0.5)|\).\(^5\)

We can also generalize our second difficulty indicator for Boolean functions defined in Equation (5) for continuous functions. Naturally, instead of using the Boolean difference, we use the discrete derivative (Equation (6)). However, given that discrete derivatives can be positive or negative, while Boolean differences cannot, in order to keep the two families of indicators as similar as possible, we decided to take the absolute value of the derivative as our “microscopic” measure of difficulty, thereby considering only the magnitude of changes, not their direction. Our new indicators are defined as follows:

\[
\zeta_i(f) = \sum_{x \in I} \sum_{a \in C(i)} |\Delta h f_a(x)|,
\]

where \( h \) is the minimum distance obtained as done for Equation (7) and \( C(i) \) is defined as for Equation (5). For example, for two variables, \( x_1 \) and \( x_2 \), \( C(2) \) is \( \{x_1 x_1, x_1 x_2, x_2 x_2\} \).

To sum up, in this section and Section 3.2 we have adopted from a different field and reinterpreted (\( \vartheta_i \) for Boolean induction problems) or defined for the first time (\( \varphi_1 \) for Boolean induction problems and \( \varphi_i \) and \( \zeta_i \) for the case of symbolic regression on continuous functions) four families of general difficulty indicators for program induction. In the next section, we will employ them to create models of performance for EPAs.

### 3.4 Difficulty Indicators Models and their Instantiation

On the basis of our previous experience in creating performance models and also taking inspiration from the modeling techniques used to solve the algorithm selection problem, we decided in the first instance to use a linear combination of the difficulty indicators presented in the previous two sections to model EPA performance. That is, we use the following performance model for EPAs applied to Boolean induction problems:

\[
P(f) \approx a_0 + \sum_{i=1}^{N} a_i \cdot \vartheta_i(f) + \sum_{j=2}^{N} b_j \cdot \varphi_j(f),
\]

\(^5\)It is important to note that the value of \( f(1.0, 0.5) \) is not included in Table 3. As a result, one is unable to determine the value of \( |f(1.0, 0.5) - f(0.5, 0.5)| \) unless an explicit expression for function \( f \) is available. Unfortunately, this is not possible because the problem, in the first place, is to find an expression for \( f \). So by convention, for any terms that cannot be computed, we set them to zero, that is, \( |f(1.0, 0.5) - f(0.5, 0.5)| = 0 \).
where \( j \) starts at 2 because \( \vartheta_1(f) = \varphi_1(f) \), and \( a_i \) and \( b_i \) are coefficients that need to be identified. Similarly, for the case of continuous function induction we use the following difficulty indicator model:

\[
P(f) \approx a_0 + \sum_{i} a_i \cdot \vartheta_i(f) + \sum_{j=2}^{N} b_j \cdot \varsigma_j(f),
\]

where \( i \) ranges from the minimum distance between all pairs of input patterns to the maximum distance (in steps equal to the minimum distance) and \( a_i \) and \( b_i \) are coefficients.

To instantiate these models, one needs a training set of problems \( T \) and a validation set \( V \). The set \( T \) is used to identify the coefficients \( a_i \) and \( b_j \) in order to obtain a good fit between predicted and actual performance. Set \( V \) is used to test the generality of the model. \( T \) and \( V \) are composed of pairs \((f, P(f))\), where \( f \) is a problem and \( P(f) \) is the corresponding performance of the algorithm under study on problem \( f \). \( P(f) \) is obtained by running the algorithm on \( f \) and estimating its performance. As mentioned earlier, this is done by performing multiple independent runs and taking some statistical measure of tendency.

Given a sufficiently representative training set \( T \) including more elements than the number of indicators to be identified, one could apply the standard linear regression method to determine the coefficients \( a_i \) and \( b_i \). Thus, in order to identify \( a_i \) and \( b_i \) of Equation (9), one would need to solve the linear system

\[
Wa = p,
\]

where \( a = (a_0, a_1, \ldots, a_N, b_2, \ldots, b_N)^T \), \( p = (P(f_1), \ldots, P(f_1))^T \) is a vector that contains the measure performance of the algorithm for every problem in training set \( T \), and \( W \) is a matrix formed by a column of ones and the result of computing either \( \vartheta_i \) and \( \varphi_i \), or \( \varrho_i \) and \( \varsigma_i \), depending on whether the problems are Boolean or continuous, respectively. For example, for the case of Boolean functions

\[
W = \begin{pmatrix}
1 & \vartheta_1(f_1) & \ldots & \vartheta_N(f_1) & \varphi_2(f_1) & \ldots & \varphi_N(f_1) \\
\vdots & \vdots & \ddots & \vdots & \vdots & \ddots & \vdots \\
1 & \vartheta_1(f_T) & \ldots & \vartheta_N(f_T) & \varphi_2(f_T) & \ldots & \varphi_N(f_T)
\end{pmatrix}
\]

However, while this procedure identifies the values of all the coefficients, some of the difficulty indicators may not be strongly correlated with the performance of the algorithm and, therefore, their inclusion in the model is likely to be detrimental for its accuracy.

In order to identify and retain only those difficulty indicators that contribute the most to the model, we used LARS (Efron et al., 2004) and a cross-validation technique. At this point, we provide a brief outline of how LARS works. LARS starts by setting all the coefficients to zero and finds the predictor (i.e., the difficulty indicator) most correlated with the response (i.e., the performance). Then it takes the largest possible step in the direction of this predictor until some other predictor has as much correlation with the residual. At this point, LARS proceeds in the direction of the equiangular between the two predictors until a third variable has as much correlation with the residual. The process continues until the last predictor is incorporated into the model.

Informally, we can think of LARS as a procedure that sorts the difficulty indicators according to their importance given a particular algorithm instance and a class of
problems. As a result, one can decide to create a model using only the \( n \) best indicators. But what value of \( n \) should one choose? In order to free the user from this difficult choice, we used a cross-validation technique on the training set. This works as follows. The training set is split into five sets of equal size. Four sets are joined together and are used to produce a model (i.e., these sets are now the training set), while the remaining set is used to assess its generalization. The process is repeated five times, each time leaving out a different fifth of the training set \( T \). At the end of this process, we have a prediction of the performance of an algorithm for all the problems in \( T \). Since these predictions are obtained via cross-validation they give us an idea of the generalization capability of our model.

This cross-validation procedure is iteratively applied to the models produced by LARS for \( n = 1, 2, \ldots, m \) with the aim of identifying the value of \( n \) which provides the best generalization.\(^6\) The process ends when the best generalization on the training set has been reached. As we mentioned previously, we measured the overall generalization using the RSE.

4 Test Problems and Performance Measures

To test the scope and accuracy of the difficulty indicator models described above, we considered different classes of problems and performance measures. In particular, we tested our approach on Boolean induction problems and continuous symbolic regression on rational functions, modeling different versions of GP, GEP, CGP, and stochastic iterated hill climbers (SIHC). Also, we modeled different versions of GP, SIHC, and auto-regressive models (AR; more on this in Section 5.5) on the problem of wind speed forecasting.

We present the problems in more detail in the rest of this section, and we describe the algorithms in Section 5.

4.1 Boolean Induction Problems

Our first benchmark is the class of four input Boolean induction problems. This class includes 65,536 different functions, each of which can be fully represented by a bit-vector of length 16 (a truth table), that is formed by the functions' output for each possible combination of the four inputs. We randomly selected 1,100 different Boolean functions from this set and for each counted the number of times an EPA found a program that encoded the target functionality in 100 independent runs. We took as our performance measure for this class of problems the success rate, that is, the fraction of successful runs out of the total number of runs.

These randomly generated problems were divided into a training set \( T \) composed of 500 problems, and a validation set \( V \) composed of the remaining 600 problems. In this case \( I = \{(0, 0, 0, 0), (0, 0, 0, 1), \ldots, (1, 1, 1, 1)\} \).

4.2 Symbolic Regression on Rational Functions

The second benchmark is the class of continuous symbolic regression problems obtained by first randomly generating and then sampling rational functions. We created 1,100 different rational functions of the form \( f(x) = \frac{W(x)}{Q(x)} \), where \( W(x) \) and \( Q(x) \) are two

\(^6\)For the case of Boolean induction problems, \( m = 2 \cdot N - 1 \). In continuous functions this limit is imposed by the user and has a maximum value equal to the number of points minus one. For the case of the rational functions the value of \( m \) is 20, and for the case of wind speed time series the value of \( m \) is 80. In either case, this corresponds to the number of points minus one.
Models of Performance

random polynomials. More specifically, $W(x)$ and $Q(x)$ were built by randomly choosing the degree of each, in the range 2 to 8, and then choosing random real coefficients in the interval $[-10, 10]$ for the powers of $x$ up to the chosen degree. Each of the rational functions in the set was then sampled at 21 points uniformly distributed in the interval $[-1, 1]$ to generate 1,100 continuous symbolic regression problems. Note in this case $I = \{-1, -0.9, \ldots, 0.9, 1\}$.

Again, these randomly generated problems were divided into a training set $T$ composed of 500 problems, and a validation set $V$ composed of the remaining 600 problems.

For each problem, we performed 100 independent runs recording the normalized best of run fitness (NBRF). This is obtained by first standardizing the target outputs for a problem and the corresponding outputs produced by the best-of-run individual and then summing over the absolute differences between the corresponding standardized target and actual outputs. The standardized target outputs, $f(x)$ for $x \in I$, are as follows: $\frac{f(x) - \mu}{\sigma}$ where $\mu$ and $\sigma$ are the mean and standard deviation of $f(x)$, respectively. Similarly, the standardized program outputs, $p(x)$, are defined as $\frac{p(x) - \mu}{\sigma}$.

4.3 Wind Speed Forecasting

Our third benchmark is a real-world problem: predicting the wind speed in existing meteorological stations. We used data from the National Oceanic and Atmospheric Administration (2012) of the US National Climatic Data Center. More specifically, we collected the measurements from all the stations of Mexico and the USA in the period from January 2006 to December 2011.

In each station a number of variables related to the weather conditions are recorded. From these, we selected the six variables that were common to all stations: wind speed, temperature, dew point, visibility, maximum wind speed of the day, and maximum temperature of the day. Since there are many missing values in the recordings of some stations (in the worst cases more than half of the measurements were not available), we removed from our dataset all those stations where it was impossible to train a predictor (i.e., GP, SIHC, or AR). This left us with 634 stations. As before, we split them into two sets: a training set $T$ of 300 stations and a validation set $V$ composed of the remaining 334 stations.

In order to measure the performance of GP and SIHC, we performed 10 independent runs on each of the 634 problems. In each run, we collected the NBRF (see Section 4.2) and took the median of these values as our performance measure of an algorithm on a problem. In the case of AR models, we also used NBRF as a performance measure but we only performed one run because AR models were instantiated using a deterministic algorithm.

The wind speed forecasting problem presents a number of characteristics that make it different from the previous two benchmarks. Firstly, rather naturally, the problem being continuous, it differs significantly from the Boolean induction problems defined in Section 4.1. Secondly, the problem is a forecasting problem and not a regression. So algorithms can only use previous measurements of exogenous variables and past wind speed measurements to perform a prediction. Also, the problem is multi-variate, but not all the variables involved are expected to correlate with the wind speed. Finally, there are many factors that affect wind speed that are not captured by the measurements. So the data here are inherently noisy, unlike for the previous two problems.

Given these differences, we need to explain how the difficulty indicators $\rho$ and $\zeta$ are computed. First, we treat this problem as if it were a symbolic regression problem, so
we are able to build a table similar to Table 3. That is, let $x$ be a vector formed by the time (e.g., 1st of January 2006) and the values of temperature, dew point, visibility, maximum wind speed of the day, maximum temperature of the day on that particular day, and let $f(x)$ be the wind speed on that particular day. Clearly, under these circumstances, we set $I = \{x_1, x_2, \ldots, x_n\}$, where $n$ is the number of points in the time series. Now we are in the position to compute the values of $h$. Let start by $x_1$ (i.e., the time). Here the range of $h$ is $1, 2, \ldots, 2190$ (where 2,190 is the number of points in the time series minus one). For the other variables (temperature, dew point, etc.) there are something of the order of $O(n^2)$ different $h$s. This is because the difference between any two consecutive $h$s is not a constant, given that the values of these variables are measurements of weather conditions. Finally, looking at the different combinations of $x$, $e$, and $h$ to compute $f(x + he)$, the majority of defined points occur when $i = 1$ and $h = 1, 2, \ldots, n - 1$. As consequence, our multivariate wind speed forecasting problem can be seen as a single variable problem when computing $\varrho$ and $\varsigma$. That is, we treat the wind speed $f$ as a function that only depends on time, that is, $f(t)$. Under these assumptions, $I = \{1, 2, \ldots, n - 1\}$ and, thus, the procedure used to determine $\varrho$ and $\varsigma$ is equivalent to the one used for the rational symbolic regression problems.

5 Systems and Parameter Settings

In this section, we describe the algorithms used to test our modeling technique. We used different versions of EPAs, namely GP (Section 5.1), GEP (Section 5.2), CGP (Section 5.3), and SIHC (Section 5.4), and auto-regressive models (Section 5.5).

5.1 GP Systems

We used two different implementations of GP, both tree-based and using both subtree crossover and subtree mutation. One system is essentially identical to the one used by Koza (1992), the only significant difference being that we select crossover and mutation points uniformly at random, while Koza used a nonuniform distribution. This modification has as a consequence that most of the time crossover swaps the leaves of the parents while mutation replaces a leaf with a random subtree. The other system is TinyGP (Poli, 2004) with the modifications presented in Poli et al. (2008) to allow the evolution of constants, which is required in the case of continuous problems. The main difference between the two systems is that Koza’s system is generational while TinyGP uses a steady state strategy. A second difference is that Koza’s system uses roulette wheel selection while TinyGP uses tournament selection.

Table 4 shows the parameters and primitives for the GP systems.\(^7\) To avoid division by zero, we used the standard protected division function in the continuous problems. Specifically, the protected division returns the numerator when the absolute value of the denominator is lower than 0.001. Naturally, only combinations of crossover and mutation rates such that $p_{xo} + p_{m} \leq 100\%$ could be used, since crossover and mutation are mutually exclusive operators in GP. The terminal set used in the wind speed forecasting problems is actually not only composed of six variables, but also includes the values that these variables took in the previous 30 observations. That is, in order to forecast the current wind speed, forecasting models will use the past 30 measurements of each of the variables previously mentioned (this procedure will be

\(^7\)It is worth mentioning that we did not try to fine tune the parameters of any of the algorithms studied. The objective here is modeling their performance, not to get the best possible performance.
Table 4: Parameters used in the GP experiments.

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Values</th>
</tr>
</thead>
<tbody>
<tr>
<td>Function set (rational problems)</td>
<td>{+, −, *, / (protected)}</td>
</tr>
<tr>
<td>Function set (wind speed forecasting)</td>
<td>{+, −, *, / (protected), sin, cos}</td>
</tr>
<tr>
<td>Function set (Boolean problems)</td>
<td>{AND, OR, NAND, NOR}</td>
</tr>
<tr>
<td>Terminal set (rational problems)</td>
<td>{x, (\mathbb{R})}</td>
</tr>
<tr>
<td>Terminal set (wind speed forecasting)</td>
<td>{wind speed, temperature, dew point, visibility, max. wind speed, max. temperature, and (\mathbb{R})}</td>
</tr>
<tr>
<td>Random constants (i.e., (\mathbb{R}))</td>
<td>100 constants in ([-10, 10])</td>
</tr>
<tr>
<td>Terminal set (Boolean problems)</td>
<td>{x_1, x_2, x_3, x_4}</td>
</tr>
<tr>
<td>Crossover rate (p_{xo})</td>
<td>100%, 90%, 50%, and 0%</td>
</tr>
<tr>
<td>Mutation rate (p_{m})</td>
<td>100%, 50%, and 0%</td>
</tr>
<tr>
<td>Maximum tree depth used in mutation</td>
<td>4</td>
</tr>
<tr>
<td>Selection mechanism</td>
<td>Tournament (size 2) and roulette wheel</td>
</tr>
<tr>
<td>Population size</td>
<td>1,000</td>
</tr>
<tr>
<td>Number of generations</td>
<td>50</td>
</tr>
<tr>
<td>Number of independent runs</td>
<td>100</td>
</tr>
</tbody>
</table>

clarified in Section 5.5). So in total, for the case of wind speed forecasting, the terminal set has 180 elements plus the random constants represented by the terminal \(\mathbb{R}\).

For the first GP system (i.e., Koza’s system), besides the traditional roulette wheel selection (which gives parents a probability of being chosen for reproduction proportional to their fitness), we also used tournament selection with a tournament size of 2.

Thus, in total we tested three variants of GP: generational with tournament selection, generational with roulette wheel selection, and steady state with tournament selection.

5.2 GEP Systems

We used a modified version of gene expression programming (GEP; Ferreira, 2001), where we replaced the standard GEP initialization method with a technique equivalent to the ramped half-and-half method in GP (Koza, 1992).\(^8\)

We used three different versions of GEP. The first two systems are identical except for the selection mechanism used: roulette wheel selection in the first and tournament selection with tournaments of size 2 in the second. Both versions are generational, as is standard in GEP. The third version, instead, used a steady state reproduction strategy with tournament selection (size 2). The parameters and primitives used in our GEP runs are shown in Table 5.

5.3 CGP Systems

Instead of coding our CGP system, we decided to use the program implemented by Miller (2009), which is a generational system with the particular characteristic that new generations are created entirely by mutation.

\(^8\)In preliminary runs, GEP was considerably worse than the performance of the worst GP system. However, we found that using ramped half-and-half initialization considerably improved performance. We therefore, retained this modification in our experimentation.
Table 5: Parameters used in the GEP experiments.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Function set (rational problems)</td>
<td>{+, −, *, /protected}</td>
</tr>
<tr>
<td>Function set (Boolean problems)</td>
<td>{AND, OR, NAND, NOR}</td>
</tr>
<tr>
<td>Terminal set (rational problems)</td>
<td>{x, R}</td>
</tr>
<tr>
<td>Random constants (i.e., R)</td>
<td>100 constants drawn from the interval [−10, 10]</td>
</tr>
<tr>
<td>Terminal set (Boolean problems)</td>
<td>{x₁, x₂, x₃, x₄}</td>
</tr>
<tr>
<td>Head length</td>
<td>63</td>
</tr>
<tr>
<td>Number of genes</td>
<td>3</td>
</tr>
<tr>
<td>Mutation rate pₘ</td>
<td>5%</td>
</tr>
<tr>
<td>1 point recombination rate</td>
<td>20%</td>
</tr>
<tr>
<td>2 point recombination rate</td>
<td>50%</td>
</tr>
<tr>
<td>Gene recombination rate</td>
<td>10%</td>
</tr>
<tr>
<td>IS transposition rate</td>
<td>10%</td>
</tr>
<tr>
<td>IS elements length</td>
<td>1, 2, 3</td>
</tr>
<tr>
<td>RIS transposition rate</td>
<td>10%</td>
</tr>
<tr>
<td>RIS elements length</td>
<td>1, 2, 3</td>
</tr>
<tr>
<td>Gene transposition rate</td>
<td>10%</td>
</tr>
<tr>
<td>Selection mechanism</td>
<td>Tournament (size 2) and roulette wheel</td>
</tr>
<tr>
<td>Population size</td>
<td>1,000</td>
</tr>
<tr>
<td>Number of generations</td>
<td>50</td>
</tr>
<tr>
<td>Number of independent runs</td>
<td>100</td>
</tr>
</tbody>
</table>

Table 6: Parameters used in the CGP experiments.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Function set (rational problems)</td>
<td>{+, −, *, /protected}</td>
</tr>
<tr>
<td>Function set (Boolean problems)</td>
<td>{AND, OR, NAND, NOR}</td>
</tr>
<tr>
<td>Terminal set (rational problems)</td>
<td>{x, R}</td>
</tr>
<tr>
<td>Terminal set (Boolean problems)</td>
<td>{x₁, x₂, x₃, x₄}</td>
</tr>
<tr>
<td>Random constants (i.e., R)</td>
<td>5 real value constants in the interval [−10, 10]</td>
</tr>
<tr>
<td>Mutation rate</td>
<td>1%</td>
</tr>
<tr>
<td>Number of rows</td>
<td>1</td>
</tr>
<tr>
<td>Number of columns</td>
<td>2,000</td>
</tr>
<tr>
<td>Population size</td>
<td>5</td>
</tr>
<tr>
<td>Number of generations</td>
<td>12,500</td>
</tr>
<tr>
<td>Number of independent runs</td>
<td>100</td>
</tr>
</tbody>
</table>

Table 6 shows the parameters used in the CGP runs (which are effectively those suggested by Miller). The table highlights some differences between CGP and the other EPA systems used (see Section 5.1 and Section 5.2). First, CGP uses very small population sizes. Second, it evolves for a large number of generations. So, for fairness, in our comparisons, we gave all systems the same number of fitness evaluations. Third, the only genetic operator used by CGP is mutation. Fourth, in CGP each new population is created by mutating the best individual found in the previous generation. As a result, the mutation rate refers to the percentage of mutation done to the best individual in order to obtain an offspring.
Table 7: Parameters used in the SIHC experiments.

<table>
<thead>
<tr>
<th>Mutation</th>
<th>Maximum number of mutations</th>
<th>Maximum number of fitness evaluations</th>
</tr>
</thead>
<tbody>
<tr>
<td>Subtree</td>
<td>50, 500, 1,000, and 25,000</td>
<td>50,000</td>
</tr>
<tr>
<td>Uniform</td>
<td>25,000</td>
<td>50,000</td>
</tr>
</tbody>
</table>

5.4 SIHC Systems

The SIHC we used is similar to the one presented by O’Reilly and Oppacher (1994a, 1994b) although here we used different mutation operators. We used two types of mutation: subtree mutation, which is the same type of mutation used in the GP runs, and a mutation operator similar to the one described by O’Reilly and Oppacher (1994a, 1994b), which we will call uniform mutation since all the elements of a tree may be modified. Uniform mutation works as follows. Each node in the tree is mutated with a certain probability. A node selected for mutation can be modified with one of three different operations (each having the same probability). First, the node can be replaced with another node of the same type, that is, a terminal is replaced with another terminal, while a function is replaced with another function of the same arity. The second option is to replace it with a randomly generated subtree. This operation also involves replacing one of the children of the random subtree with the tree originally rooted at the mutated node. The third option is to replace the node with one of its children. The child chosen is the one having the most nodes.

The SIHC algorithm starts by creating a random program tree. To create the tree, we adopted the same procedure as for the GP and GEP experiments and the same terminal and function sets as for the GP systems (Table 4). SIHC then mutates this tree until a better program is found (i.e., when the fitness of an offspring is better than the parent’s fitness). This then becomes the parent and the process is repeated. When a maximum number of allowed mutations is reached, the current individual is set aside, a new random individual is created, and the process is repeated. Finally, the algorithm terminates when a solution has been found or when a maximum number of fitness evaluations is reached. The output of the algorithm is the program which scored the highest fitness throughout the search process.

Table 7 shows the parameters used for the SIHC experiments.

5.5 Auto-Regressive Models (AR)

An AR model (Makridakis et al., 1992) is a statistical model for modeling and forecasting the behavior of time series. These models have the following form:

\[
\hat{y}_t = \sum_{k=1}^{w} a_k y_{t-k} + \sum_{x} \sum_{k=1}^{w} b_{xk} x_k
\]

where \(y_t\) represents the measurement of variable \(y\) at time \(t\) in the time series, \(x\) are the exogenous variables, \(\hat{y}_t\) is the forecast produced by the AR equation, \(w\) is the window width, and \(a_k\) and \(b_{xk}\) are coefficients that determine how the history of measurement affects the prediction. The window width, \(w\), represents how far behind in time a measurement is considered an important input for the AR model. Outside the window, observations are not taken into account.

For the wind speed forecast task considered here, in Equation (12) we decided to set \(w = 30\), that is, we used a month of observations to make the prediction, while \(y\) stands...
for the wind speed and $x$ ranges over the temperature, dew point, visibility, maximum wind speed, and maximum temperature. The coefficients $a$ and $b$ were identified using LARS.

As mentioned previously, LARS is an iterative process that can be stopped at any step. In each step an additional variable is included in the AR model. This gives the flexibility of deciding among the available variables how many of them to include in the AR. Of course, this is a delicate task and one should take care to include only those variables that contribute the most to the accuracy of the model. Here, we decided to create three different AR models. The first two models are obtained by stopping LARS after five or 10 variables have been included, respectively. In the third model, instead, an equivalent procedure to the one described in Section 3.4 was used to decide the number of variables. That is, we used a fivefold cross-validation to identify how many variables to include in the AR model.

6 Results

In this section, we analyze the accuracy of the difficulty indicator models proposed in this paper across different algorithms, different benchmark problems, and different performance measures.

6.1 Symbolic Regression and Boolean Induction

Let us start by analyzing the RSE obtained by different systems for the benchmark classes of Boolean induction problems and symbolic regression of rational functions. We will compare the quality of our new difficulty indicator models in Equations (9) and (10) against the modeling accuracy achieved by our previous models of EPA performance (Graff and Poli, 2010) in Equation (1).

Table 8 presents these comparisons. For each class, the table shows the accuracy of the difficulty-indicator models (first column of each column block) and the RSE values of our previous models of performance. As can be seen from the table, all the models have RSE values well below one, indicating that they are predicting better than the mean. Let us remember that an RSE value of one indicates that the model is predicting exactly as the mean, that is, the average performance of the algorithm being modeled. For the case of Boolean functions, we can see that our new difficulty indicator models obtained the lowest RSE values in all the cases except for the GEP systems. In the case of symbolic regression on rational functions, the new difficulty indicator models obtained the lowest RSE values in all the cases. For the case of Boolean induction, the average percentage of improvement is of 17.96%. For the case of rational functions, the average percentage of improvement is 31.6%. Furthermore, the one-tailed Wilcoxon Rank sign test for paired data shows with a confidence of 95% and for both classes of problems that the RSE values of our difficulty indicator models are lower than the RSE values of our previous models of performance.

So it is clear that our new difficulty indicator models provide significantly more accurate predictions than our earlier performance models. In addition, they are simpler to understand because they have fewer degrees of freedom. For instance, in the case of rational functions, the difficulty indicator models require on average 27 coefficients, while our earlier performance models require more than 100. The situation is even more extreme in the case of Boolean induction problems, where the new difficulty indicator models have at most eight coefficients, while our earlier performance models require, again, more than 100.
While RSE values provide an objective assessment of the quality of the models, it might be difficult for the reader to appreciate the accuracy of our models from only such figures. To visually illustrate the quality of our models, Figure 1 shows scatter plots of the actual performance versus the performance estimated with our new models on the validation set for both Boolean induction and symbolic regression problems.\footnote{The data in the figure refers to the GP system with 90\% crossover rate, no mutation, and roulette wheel selection, but other parameter settings and systems provide qualitatively similar results.} The solid diagonal line in each plot represents the behavior of a perfect model. As can be seen, the points form tight clouds around the perfect models, which is a clear qualitative indication of the accuracy of the predictions of our models of performance.
M. Graff, R. Poli, and J. J. Flores

Figure 1: Scatter plots of the performance measured versus the performance obtained with the model of (a) continuous regression problems and (b) Boolean induction problems on the validation set. The data refer to a GP system with 90% crossover rate, no mutation, and roulette wheel selection.

Table 9: Quality (RSE) of the difficulty indicator models and the previous performance models (Equation (1)) on the wind speed forecasting problem.

<table>
<thead>
<tr>
<th>Forecasting method</th>
<th>Difficulty indicator models</th>
<th>Previous performance models</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Xval. RSE</td>
<td>T RSE</td>
</tr>
<tr>
<td>AR-CV</td>
<td>0.5319</td>
<td>0.3805</td>
</tr>
<tr>
<td>AR-5</td>
<td>0.5756</td>
<td>0.4481</td>
</tr>
<tr>
<td>AR-10</td>
<td>0.5398</td>
<td>0.3563</td>
</tr>
<tr>
<td>GP with $p_{co} = 0%$</td>
<td>0.6291</td>
<td>0.3865</td>
</tr>
<tr>
<td>SIHC</td>
<td>0.6372</td>
<td>0.4214</td>
</tr>
</tbody>
</table>

6.2 Models of Wind Speed Forecasting

Table 9 shows the accuracy (RSE values) of the difficulty indicator models in the wind speed forecasting problem. The table also presents the accuracy of our previous modeling technique in this problem. For each technique and modeling approach, the table presents the $k$-fold cross-validation RSE, training set RSE, and the validation set RSE. As can be seen, the RSE value of the difficulty indicator models are in all cases better than the corresponding values obtained with our previous performance models and these differences are statistically significant. Indeed, there is an improvement on average of 35.69%. Furthermore, the RSE values are comparable to the ones obtained in the previous two benchmark problems.

We should note that, while the new performance models work very well, the RSE values recorded on the validation set by our previous performance models on the wind speed benchmark are higher than for other benchmarks. We suspect this may be due to the effect known as the curse of the dimensionality (Chávez et al., 2001). Informally, this manifests itself as the characteristic that the distance between any pair of vectors is almost a constant when the vectors have a high dimensionality, as is the case of this
class of problems where each vector (i.e., the wind speed problem being forecast) has more than 2,000 dimensions given that it is composed of the wind speed measurements of each day. This has as the consequence that the features representing each problem lack diversity and, thus, one is unable to produce a good model.

The algorithm selection problem in the domain of time series forecasting presents another challenge that it is not exhibited by the wind speed time series presented here. The problem is that, in general, each time series has a different length. As a consequence, one needs to deal with this diversity in order to apply either the difficulty indicator models or our previous performance models. Let us discuss this problem in more detail. In the performance models one needs to define a distance \(d\) between the two problems, \(a\) and \(b\), that is, \(d(a, b)\). If \(a\) and \(b\) have different lengths, \(d\) must deal with this characteristic. A simple approach would be to pad the shortest time series with 0s until both time series have the same length. However, in the case of the difficulty indicator models, this problem is less severe. Let us remember that one needs to decide the number of components \(\varrho_i\) and \(\varsigma_i\) to use in the model. Clearly, this number is restricted by the length of the time series; a simple approach is to select as many components as the maximum number of components that can be computed for the shortest time series. This should work as our experiments show that high order \(\varrho_i\)s and \(\varsigma_i\)s do not provide as much information as the lower order ones.

7 Conclusions

We presented a set of techniques to build efficient and accurate models of performance of different program induction algorithms and auto-regressive models. We modeled three versions of GP with multiple parameter settings, three versions of GEP, two versions of SIHC (one with multiple parameters settings), one version of CGP, and three different AR models. These algorithms were applied to Boolean induction problems, symbolic regression of rational functions problems, and wind speed forecasting problems. This corresponds to a total of 47 modeling conditions (21 for Boolean induction, 21 for symbolic regression, and five for wind speed forecasting).

We compared the modeling technique presented here against the only alternative so far, that is, our previous approach to model the performance of EPAs (Graff and Poli, 2008, 2010). For 44 out of 47 conditions, our new difficulty indicator models provided significantly better accuracy than the corresponding models obtained with our previous technique. Furthermore, our difficulty indicator models have fewer coefficients than previous models. This has a positive effect on the generalization ability of the models, since more parsimonious models are less likely to learn the noise in the training set and overfit.

Many applications are possible for our difficulty indicator models. Our models could be used, for example, to analyze the effects that the parameters have in the performance of EPAs (see Graff and Poli, 2010), thereby enabling practitioners to wisely optimize the parameters for a new problem before they start production runs with an algorithm. Also, as we did in Graff and Poli (2010) it would be possible to construct taxonomies of the performance of algorithms. It would also be possible to use our models to create algorithm portfolios of EPAs, thereby solving the algorithm selection problem for induction problems. As illustrated with the wind speed forecasting problems, it would also be possible to apply our models in real world applications. For example, algorithm portfolios guided by our models could select the most appropriate modeling technique for each meteorological station without the need to test them all. Also, we
can easily imagine extending the approach to the domain of financial forecasting and many others. We will explore these avenues of research in future work.

We have presented models that can accurately predict the performance of EPAs and other algorithms; however, there are still problems that limit the applicability of these techniques. One particular problem common to all the procedures used to solve the algorithm selection problem is that these methodologies need a training set to instantiate their models. There are many open questions regarding the construction of this training set. For instance, it is not clear how many elements this set must have in order to capture all the characteristics of the phenomena being modeled. This has important consequences: having an insufficient training set would produce models that do not generalize. On the other hand, having a very large training set would impact the time required to build a model. In future research we will explore this problem.

Acknowledgments

The authors would like to thank Leonardo Vanneschi and the anonymous reviewers for their helpful comments. The first author acknowledges support by PROMEP grant UMSNH-PTC-333.

References


Models of Performance


