Performance Models for Evolutionary Program Induction based on Problem Difficulty Indicators

Mario Graff[1] and Riccardo Poli[2]

[1] Division de Estudios de Posgrado
Facultad de Ingenieria Electrica
Universidad Michoacana de San Nicolas de Hidalgo, Mexico
mgraffg@lsc.fie.umich.mx
[2] School of Computer Science and Electronic Engineering
University of Essex, UK
rpoli@essex.ac.uk

Abstract. Most theoretical models of evolutionary algorithms are difficult to apply to realistic situations. In this paper, two models of evolutionary program-induction algorithms (EPAs) are proposed which overcome this limitation. We test our approach with two important classes of problems — symbolic regression and Boolean function induction — and a variety of EPAs including: different versions of genetic programming, gene expression programing, stochastic iterated hill climbing in program space and one version of cartesian genetic programming. We compare the proposed models against a practical model of EPAs we previously developed and find that in most cases the new models are simpler and produce better predictions. A great deal can also be learnt about an EPA via a simple inspection of our new models. E.g., it is possible to infer which characteristics make a problem difficult or easy for the EPA.

Keywords: Evolutionary Program-induction Algorithms, Genetic Programming, Performance Prediction, Hardness Measures

1 Introduction

Evolutionary Program-induction Algorithms (EPAs) are search techniques for the automatic evolution of computer programs. Genetic Programming (GP) [10, 16], Cartesian GP (CGP) [12], Grammatical Evolution [14] and Gene Expression Programming (GEP) [3], among others, are members of this family.

There has been an enormous increase in interest in EPAs over the last two decades, resulting in the appearance of new generic operations, representations, and so on. Also, researchers and practitioners from a number of disciplines have used EPAs on a huge variety of new problems. There are, however, also many unanswered questions regarding EPAs.

In principle one would expect theory to be able to shed light on old and new EPA techniques and problems. However, this is not so. The key reason is that producing EPA theory is objectively a very hard and slow process where the

precise details of an algorithm matter a great deal [11, 17]. In practice, exact theoretical EPA models are almost impossible to apply to realistic situations.

This has important consequences from a practitioner's point of view. For instance, one is unable to estimate the performance of an algorithm on a particular problem without running the algorithms on that problem. Thus, we cannot *a priori* discern which is the most suitable algorithm to solve a particular problem out of a set of different algorithms and/or algorithms with different parameters. Also, the current lack of theoretically-sound guidelines forces practitioners to manually hand-tune algorithms, parameters and operators.

This paper proposes a solution to these problems. We introduce two practical models for the performance of EPAs based on *difficulty indicators*. With these we are able to precisely forecast the performance of a variety of EPAs for symbolic regression and Boolean induction problems and two performance measures — the expected end-of-run fitness and the success rate. These models also allow us to define procedures that can solve the algorithm selection problem [18] (i.e., the problem of deciding which algorithm to use from a set of available algorithms to solve a particular problem) via the creation of algorithm portfolios. Also, as we will see, our models are simple to analyse.

The rest of the paper is organised as follows. In Section 2, we review related work. Section 3 presents our new modelling technique and the process used to apply it. The problems and algorithms used to validate our approach are described in Section 4. Section 5 provides our experimental results. Section 6 shows how our models can be used to analyse algorithms. Some conclusions and possible directions for future work are given in Section 7.

2 Related work

Our work is related to the problem of understanding what makes a problem easy or hard for EAs. One of the earliest attempts in this direction was made by Jones [9] who introduced a heuristic called *fitness distance correlation* (*fdc*), as an algebraic indicator of problem difficulty for GAs. The study of *fdc* has been extended to GP (e.g., see [19]). This has shown that *fdc* is often a reliable indicator of problem hardness, but that it has also some flaws, the most severe one being that the computation of *fdc* requires the optimal solution(s) to be known beforehand. This is obviously unrealistic and prevents one from using the *fdc* 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* (*nsc*) [20] which has been shown to be fairly reliable in a number of different benchmark problems in GP.

While these (and other) measures have shown some success at providing insights on how hard or easy a problem is, they do not really provide a estimation of the performance of an algorithm. For instance, they are unable to predict the success rate of a particular algorithm or the fitness of the best solution found during the evolution process.

There are two other research areas where models capable of making such predictions have been proposed: (a) the algorithm selection problem and the related algorithm portfolios (e.g., see [8]) and (b) the models of performance of evolutionary algorithms proposed in [2, 6, 7]. These are described in more detail in the next sub-sections.

2.1   Algorithm Selection and Portfolios

The algorithm selection problem and algorithm portfolios (which are a collection of algorithms that are run in parallel or in sequence to solve a particular problem) require the creation of models to forecast the performance of every algorithm in a set before deciding which one to use to solve a problem [8].

Such models take a variety of forms including: linear equations, Markov decision processes, Bayesian models, and so on. However, generally, all models are based on a set of features that are related to the hardness of the problems being solved. For instance, in SAT problems the feature set might include the number of variables, the number of clauses, the ratio between variables and clauses, and so on. This characteristic was a source of inspiration for the work in this paper.

2.2   Models of Performance for Evolutionary Algorithms

[2] proposed to model the performance of GAs based on a re-representation of the fitness function and a liner combination of the degrees of freedom of such a representation. The idea was that any algorithm whose selection is based on comparing the fitness of different solutions (such as GAs with rank selection, truncation selection or tournament selection) can be re-represented using the outcome of all possible comparisons between pairs of solutions. This approach presents some interesting features, but it does not scale well with the size of the search space. As a result, it cannot be applied directly to large search spaces (such as the spaces explored by EPAs).

In [6, 7] we proposed an approach that has shown success in modelling the performance of EPAs. The key idea was to predict the performance, $P(\mathbf{t})$, of an EPA based on the similarity between the problem, $\mathbf{t}$, to be faced by the algorithm and a set of reference problems, $\mathcal{S}$, previously selected. In formulae:

$$P(\mathbf{t}) \approx a_0 + \sum_{\mathbf{p} \in \mathcal{S}} a_{\mathbf{p}} \cdot d(\mathbf{p}, \mathbf{t}), \tag{1}$$

where d is a similarity measure and a_i 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.

3   Modelling EPA's Performance

In this paper we want to combine the strengths of the approaches reviewed above. We want to use approximate linear models as in [2] and [6, 7], together

with sets of features that are related to the difficulty of problems, as is done in algorithm portfolios and the algorithm selection problem.

We start by asking if *fdc* and *nsc*, which have been proposed as difficulty indicators for EPAs' problems, can be used as features in our new performance models. Unfortunately, as we will see in the next subsection, the characteristics of these difficulty indicators prevent their use in this particular task. However, in Section 3.2, we propose a new set of difficulty indicators for Boolean functions that is suitable for creating performance models. We generalised these difficulty indicators to the domain of continuous functions in Section 3.3. Then in Section 3.4 we present our new model and the procedure used to apply it.

3.1 Can NSC and FDC be used in Performance Models?

As mentioned previously, the *fdc* requires the optimal solution(s) or the distance of each point from a global optimum to be known in advance. As a consequence, this indicator cannot be used for performance modelling with unseen/unexplored problems, which is really the main objective of performance modelling.

This limitation is not present in the *nsc*. However, the procedure used to compute the *nsc* is stochastic and involves a considerable computational load. This makes *nsc* less than ideal for the construction of performance models of EPAs.

3.2 Difficulty Indicators for Boolean Functions

Franco [4] has recently proposed an indicator for the difficulty of learning Boolean functions (in terms of the number of training epochs required) using feed-forward artificial neural networks. The indicator is based on evaluating a Boolean function with different inputs and counting in how many cases the function produces different outputs. This process is applied to every pair of input patterns that are at a specific distance. The Hamming distance was used to measure the distance between pairs of inputs and Franco limited his indicator to use only those inputs that are at a Hamming distance of 1 or 2.

In formulae, this indicator is defined as:

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

where $\vartheta_i(f)$ counts the number of pair of outputs that are different when the Hamming distance of their inputs is i. That is,

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

where $f(\mathbf{j})$ is the output of Boolean function f for input $\mathbf{j}$, N is the number of variables, I contains all the possible input patters (e.g., for 3 variables I contains 8 different input patterns), and $H(\mathbf{j}, \mathbf{k})$ is the Hamming distance between input $\mathbf{j}$ and $\mathbf{k}$. The term $\frac{1}{2^N \times \binom{N}{i}}$ is a normalisation factor that ensures the value of

$\tilde{\vartheta}_i(f)$ is in the interval $[0, 1]$. The second summation is over all inputs $\mathbf{k}$ that are at a Hamming distance i from input $\mathbf{j}$.

The indicator $\tilde{\vartheta}(f)$ was found to correlate well with the time needed by a learning algorithm to train a feed-forward neural network. Based on this success, it is reasonable to wonder whether this difficulty indicator could help in assessing the hardness of Boolean induction problems also in relation to EPAs.

Clearly, the active ingredients in Franco's difficulty indicator are the terms $\vartheta_i(f)$. Therefore, we will use these terms as our features in the creation of difficulty-indicator-based performance models of EPAs.

3.3 Difficulty Indicators for Continuous Functions

We should note that the Boolean difficulty indicator described above is related to the concept of differences of Boolean functions [1]. For example, in the case of a Hamming distance of 1, Eq. (2) can be rewritten as:

$$\frac{1}{2^N \times \binom{N}{i}} \sum_{\mathbf{x} \in I} \sum_i^N \triangle f_{x_i}(\mathbf{x}),$$

where $\triangle f_{x_i}(\mathbf{x})$ is defined as $f(\mathbf{x}) \oplus f(\mathbf{x} \oplus \mathbf{e}_i)$, $\mathbf{e}_i$ is a unit vector whose i-th component is 1 and $\oplus$ is the exclusive disjunction.

Naturally, the counterpart of the Boolean difference for continuous functions is the discrete derivative. Based on this equivalence, it is reasonable to ask whether the discrete derivative can be used to produce difficulty indicators for continuous functions. Let us see what this entails.

The discrete derivative of function f w.r.t. a variable x_i can be written as

$$\triangle_h f_{x_i} = \frac{f(\mathbf{x} + h\mathbf{e}_i) - f(\mathbf{x})}{h}, \tag{3}$$

where $\mathbf{x}$ is the independent variable, h is the step and $\mathbf{e}_i$ is a unit vector.

Note: the term $f(\mathbf{x} + h\mathbf{e}_i) - f(\mathbf{x})$ in Eq. (3) matches the term $f(\mathbf{j}) - f(\mathbf{k})$ of Eq. (2). This suggests that we might be able to generalise Eq. (2) to continuous spaces if we grouped inputs according to the distance h. To achieve this we create groups of input-pairs 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 difficulty indicator for continuous functions as:

$$\varrho_i(f) = \sum_{\mathbf{x} \in I} \sum_j^N |\triangle_i f_{x_j}(\mathbf{x})|, \tag{4}$$

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

To sum up, in this and the previous section we have introduced two sets of difficulty indicators: the set $\{\vartheta_i\}$ for the case of Boolean functions and the set $\{\varrho_i\}$ for the case of continuous functions. In the next section, we will employ them to create difficulty-indicator-based performance models.

3.4 Difficulty-indicators Models

Having introduced suitable difficulty indicators, we are now in a position to define our new models of performance. We model the performance of EPAs on Boolean functions using the equation:

$$P(f) \approx a_0 + \sum_{i=1}^{N} a_i \cdot \vartheta_i(f) \tag{5}$$

and the performance of EPAs on continuous functions using the equation:

$$P(f) \approx a_0 + \sum_{i}^{N} a_i \cdot \varrho_i(f), \tag{6}$$

respectively. In the Boolean case N is the number of inputs, while in the continuous case it is the number of different distance classes. In both cases a_i are coefficients that need to be identified and f is a problem.

3.5 Model Identification

To instantiate the equations introduced above, one needs a training set of problems, T, and a validation set, V. T is used to identify the coefficients a_i so as to obtain a good fit between predicted and actual performance. The set V is used to test the generality of the model. T and V are composed by pairs $(f, P(f))$ where $P(f)$ is obtained by running the algorithm being modelled on f and assessing its performance. This was done by performing 100 independent runs of the algorithms being modelled and averaging their performance (either the expected end-of-run best fitness or the success rate) obtained in such runs.

Given a training set T, one can apply an ordinary least square method to determine the coefficients a_i [13]. For example, in order to identify a_i of Eq. (5), one needs to solve the linear system $\mathbf{Wa} = \mathbf{p}$, where $\mathbf{a}$ is a vector representing the a_i coefficients (i.e., $\mathbf{a} = (a_0, a_1, \ldots, a_N)'$), $\mathbf{p}$ is a vector that contains the measure performance of the algorithm for every problem in the training set T (i.e., $\mathbf{p} = (P(f_1), \ldots, P(f_{|T|}))'$), and $\mathbf{W}$ is a matrix whose first column is all ones and the remaining columns are the result of computing ϑ_i (or ϱ_i) from 1 to N for every problem in T.

Although the procedure described above identifies $\mathbf{a}$, it does not identify and discard those difficulty indicators (i.e., ϑ_i or ϱ_i) that are not positively contributing to the accuracy of the model. Since their inclusion decreases the quality of the model, an alternative procedure is needed. In order to identify and use only the difficulty indicators that contribute to the model, we decided to use least-angle regression (LARS) and a cross-validation technique. LARS sorts the difficulty indicators based on their correlation with the performance of the algorithm while cross validation tells us how many of them to include in the model.

Finally, we measure the overall generalisation error using the Relative Square Error (RSE) which is defined as

$$\text{RSE} = \sum_i (P_i - \tilde{P}_i)^2 / \sum_i (P_i - \bar{P})^2,$$

where i ranges over T or V depending on whether we are measuring the training set or validation set error, respectively, 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.

4 Test Problems and Systems

We consider two radically different classes of problems, namely continuous symbolic regression of rational functions and Boolean function induction problems, and two typical performance measures: a normalised version of best of run fitness (more on this below) and the success rate.

A benchmark set was created for continuous symbolic regression by randomly generating and then sampling rational functions. We created 1,100 different rational functions using the following procedure. Two polynomials, $W(x)$ and $Q(x)$, were built by randomly choosing the degree of each in the range 2 to 8, and then choosing at random real coefficients in the interval $[-10, 10]$ for the powers of x up to the chosen degree. A rational function in our training set is then given by $f(x) = \frac{W(x)}{Q(x)}$. Each of the rational functions in the set was finally sampled at 21 points uniformly distributed in the interval $[-1, 1]$.

For each rational function f, we performed 100 independent runs recording the normalised best of run fitness (NBRF) which is computed by normalising the problem and the behaviour of the best individual and then summing the absolute differences between the components of these normalised functions.

The second benchmark used is the class of 4 input Boolean induction problems. We randomly selected 1,100 different Boolean functions from this set and for each we counted the number of times the algorithm found a program that encoded the target functionality in 100 independent runs. We took as our performance measure the success rate, i.e., the fraction of successful runs out of the total number of runs.

Each benchmark set was divided into two sets: a training set T composed of 500 elements, and a validation set V comprising the remaining 600 elements.

The algorithms modelled are different versions of GP, GEP, CGP, and a Stochastic Iterated Hill Climber (SIHC). Tab. 1 shows the parameters that are common to all the systems. Below, we briefly review the systems used:

GP We used two different implementations of GP. One system is essentially identical to the one used by Koza [10]. The other is TinyGP with the modifications presented in [16] to allow the evolution of constants. For the Koza-style GP system, besides the traditional roulette-wheel selection, we also used tournament selection with a tournament size of 2.

Table 1. Parameters of the EPAs used in our tests

Function set (rational problems)	$\{+, \ -, \ *, \ / \ \text{(protected)}\}$
Function set (Boolean problems)	$\{AND, \ OR, \ NAND, \ NOR\}$
Terminal set (rational problems)	$\{x, \ \mathbb{R}\}$
Random constants (i.e., $\mathbb{R}$)	100 constants drawn from the inteval $[-10, 10]$
Terminal set (Boolean problems)	$\{x_1, \ x_2, \ x_3, \ x_4\}$
Number of individuals evaluated	50000
Number of independent runs	100

GEP We used three different configurations of GEP [3]. The first is almost equivalent to the one described in [3] with the difference that we replaced GEP's initialisation method with the ramped-half-and-half method used in GP [10]. We used this modification in all configurations. The second GEP uses tournament selection (tournament size 2). The third is a steady state system with tournament selection.

CGP We used the implementation of CGP available at `https://sites.google.com/site/julianfrancismiller/` .

SIHC Our Stochastic Iterated Hill Climber is similar to the one in [15], but we modified the mutation operators. We used sub-tree mutation and a mutation operator similar to the one in [15], which we call *uniform mutation*. We also tested this system with different values for the maximum number of allowed mutations before the search restarts from a new random individual.

5 Results

We start this section by comparing the quality of the models proposed in terms of RSE values. As we will see, the new models do quite well in terms of making accurate predictions. Furthermore, in most cases they outperform the performance models we proposed in our earlier work [6, 7] (see Eq. (1)).

Tab. 2 shows an accuracy comparison (in terms of RSE) between difficulty-indicator models (i.e., Eqs. (5) and (6)) and performance models (i.e., Eq. (1)) in the validation set for rational symbolic regression and Boolean induction. All difficulty-indicator models have an RSE value well below 1 indicating that they are predicting better than the mean. For the case of rational functions, we see that difficulty-indicator models obtained lower RSE values in all the cases except for the GEP system with tournament selection. In Boolean induction, difficulty-indicator models obtain lower RSE values in 18 out of 21 cases.

Difficulty-indicator models clearly make 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 difficulty-indicator models use on average 14 coefficients whilst performance models use more than 100 coefficients. The situation is even more extreme in the case of Boolean induction problems where difficulty-indicator models have at most 4 coefficients whilst performance models require more than 100 coefficients.

Table 2. Quality (RSE) of the difficulty-indicators models (Eqs.. (6) and (5)) and the performance models (Eq. (1)) in the validation set, for different EPAs and parameter settings (crossover rate p_{xo} and mutation rate p_m).

Configuration				Rational Functions		Boolean Functions	
Type	Selection	p_{xo}	p_m	Eq. (6)	Eq. (1)	Eq. (5)	Eq. (1)
Generational	Roulette	1.00	0.00	**0.4511**	0.5246	**0.2306**	0.2877
		0.90	0.00	**0.4552**	0.5375	**0.2289**	0.2962
		0.50	0.50	**0.4206**	0.4999	**0.2274**	0.2833
		0.00	1.00	**0.4074**	0.4907	**0.2493**	0.3058
		GEP		**0.4926**	0.5212	0.5265	**0.3745**
Generational	Tournament	1.00	0.00	**0.3506**	0.4082	**0.3166**	0.4065
		0.90	0.00	**0.3525**	0.4257	**0.3182**	0.3941
		0.50	0.50	**0.3448**	0.4130	**0.3267**	0.4010
		0.00	1.00	**0.3545**	0.4291	**0.3643**	0.4686
		GEP		0.4794	**0.4477**	0.5570	**0.4501**
Steady State	Tournament	1.00	0.00	**0.4436**	0.5778	**0.4433**	0.5401
		0.90	0.00	**0.4399**	0.5634	**0.4628**	0.5820
		0.50	0.50	**0.4794**	0.5967	**0.4729**	0.6379
		0.00	1.00	**0.5137**	0.6367	**0.5208**	0.6336
		GEP		0.4347	**0.4243**	0.7216	**0.5512**
Sys.	Mut.	Max. Mut.		Eq. (6)	Eq. (1)	Eq. (5)	Eq. (1)
SIHC	Sub-Tree	50		**0.2859**	0.4349	**0.3226**	0.4045
		500		**0.2885**	0.4540	**0.3025**	0.3989
		1000		**0.2676**	0.4378	**0.2855**	0.3787
		25000		**0.2935**	0.4587	**0.2416**	0.3120
	Uni.	25000		**0.4002**	0.4890	**0.2792**	0.3641
System				Eq. (6)	Eq. (1)	Eq. (5)	Eq. (1)
CGP				**0.4348**	0.5271	**0.7400**	0.8355

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 such values alone. To provide a more visual indication of the quality of the models, Fig. 1 shows scatter plots of the actual performance *vs* the performance estimated with our new models on the validation set for both classes of problems. The data in Fig. 1 refer 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 behaviour 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.

6 Eliciting Knowledge from our Models

Our difficulty-indicator models can be used to analyse the behaviour of the systems being modelled. For instance, they could, for example, be used to create
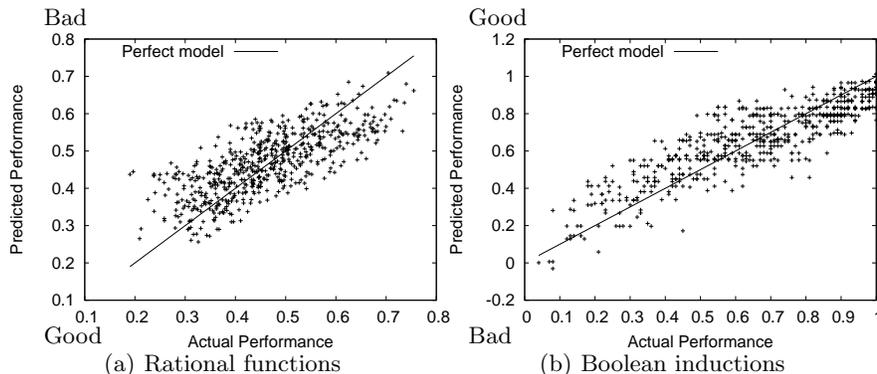
Fig. 1. Scatter plots of the predicted *vs.* actual performance in continuous regression problems (a) and Boolean induction problems (b) on the validation set for a GP system with 90% crossover, no mutation and roulette-wheel selection.

meaningful taxonomies of EPAs following the procedure presented in [5], by simply replacing the modelling technique used there with the difficulty-indicator models. Such taxonomies could reveal relationships, in terms of performance, of the different algorithms involved in the taxonomy.

However, here we want to focus on a simpler technique. By looking at the coefficients of the models, one can infer which characteristics of the problem make it difficult to solve for the algorithm under study. That is, the sign and the value of a coefficient indicate whether the corresponding characteristic makes a problem easier or harder for a particular EPA. For example, if $a_i > 0$ and the performance measure is the success rate, then a problem having a high value in factor ϑ_i (or ϱ_i) will be easier than a problem with a lower ϑ_i (or ϱ_i), and *vice versa* for $a_i < 0$.

Tab. 3 shows the a_i coefficients for different systems in the Boolean induction case. From the table one can see that the coefficients associated to ϑ_1 and ϑ_4 are negative indicating that a problem with a high value on those factors will be difficult. For example, the even-parity Boolean function, which is considered to be a hard problem, has indeed a high value of ϑ_1 (while $\vartheta_2 = 0$).

From the table, it may be also observed that the coefficients of the systems that only differ in the crossover and mutation rates are more similar than the ones that differ in the selection mechanism and type of evolution. Furthermore, one can see that the coefficients of the generational GP system with tournament selection are closer to the steady state GP system than to the generational system with roulette selection. This is an indication that the selection mechanism is playing a more important role than the type of evolution and the rates of the genetic operators in the EPAs under test. These conclusions are similar to those drawn in [5] from the analysis of algorithm taxonomies. However, here such conclusions were reached by the much simpler direct inspection of the coefficients of a model, further illustrating the practicality of our new models.

Table 3. Coefficients of the difficulty-indicators models, in the Boolean induction problems, corresponding to the GP systems with 100% crossover and 100% mutation.

Configuration			Coefficients			
Type of Evolution	Selection	p_{xo}	ϑ_1	ϑ_2	ϑ_3	ϑ_4
Generational	Roulette	1.00	-1.9744	1.2584	0.2382	-0.1529
Generational	Roulette	0.00	-1.8071	1.2427	0.2099	-0.2139
Generational	Tournament	1.00	-1.1809	1.1316	0.1316	-0.1258
Generational	Tournament	0.00	-0.9720	0.9342	0.1281	-0.1472
Steady State	Tournament	1.00	-0.7842	0.7720	0.2144	-0.1059
Steady State	Tournament	0.00	-0.6959	0.7386	0.1763	-0.1503

7 Conclusions

We have presented a new technique for building efficient and accurate models of the performance of EPAs. We modelled three versions of GP with multiple parameter settings, three versions of GEP, two versions of SIHC (one with multiple parameters settings) and one version of CGP. These algorithms were applied to two problem classes: symbolic regression of rational functions and Boolean induction. The new models are a hybrid between our previous models and the models based on difficulty measures used to solve the algorithm selection problem.

We compared the new models against our previous approach to model the performance of EPAs. In most of the cases the new difficulty-indicator models have better accuracy than the corresponding models obtained in our previous work. Furthermore, the new difficulty-indicator models require to identify fewer coefficients. This has a positive effect from the point of view of the generalisation ability of the models. Furthermore, it makes it easy to analyse our models and draw conclusions about what makes a problem hard or easy for a particular system.

Finally, we would like to briefly discuss possible future research avenues. Our approach is able to perform accurate predictions on the classes of problems tested here. However, our difficulty indicators are at present restricted to these two classes of problems. In future work, we will explore whether these indicators can be further generalised for use in other classes of problems.

References

1. S. B. Akers Jr. On a theory of boolean functions. *Journal of the Society for Industrial and Applied Mathematics*, 7(4):487–498, 1959.
2. Y. Borenstein and R. Poli. Information landscapes. In H.-G. Beyer and U.-M. O'Reilly, editors, *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO-2005)*, pages 1515–1522, Washington DC, USA, June 2005. ACM.

3. C. Ferreira. Gene expression programming: A new adaptive algorithm for solving problems. *Complex Systems*, 13(2):87–129, 2001.

4. L. Franco. Generalization ability of boolean functions implemented in feedforward neural networks. *Neurocomputing*, 70:351–361, 2006.

5. M. Graff and R. Poli. Automatic creation of taxonomies of genetic programming systems. In L. Vanneschi *et al.*, editor, *EUROGP-2009*, volume 5481 of *Lecture Notes in Computer Science*, pages 145–158, Tuebingen, Apr. 15-17 2009. Springer.

6. M. Graff and R. Poli. Practical performance models of algorithms in evolutionary program induction and other domains. *Artif. Intell.*, 174(15):1254–1276, 2010.

7. M. Graff and R. Poli. Practical model of genetic programming's performance on rational symbolic regression problems. In M. O'Neill *et al.*, editor, *Proceedings of the 11$^{\text{th}}$ European Conference on Genetic Programming (EuroGP-2008)*, volume 4971 of *Lecture Notes in Computer Science*, pages 122–132, Naples, Italy, March 2008. Springer.

8. F. Hutter, Y. Hamadi, H. H. Hoos, and K. Leyton-Brown. Performance prediction and automated tuning of randomized and parametric algorithms. In F. Benhamou, editor, *CP*, volume 4204 of *Lecture Notes in Computer Science*, pages 213–228. Springer, 2006.

9. T. Jones and S. Forrest. Fitness distance correlation as a measure of problem difficulty for genetic algorithms. In L. J. Eshelman, editor, *ICGA*, pages 184–192. Morgan Kaufmann, 1995.

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

11. W. B. Langdon and R. Poli. *Foundations of Genetic Programming*. Springer, 2002.

12. J. F. Miller and P. Thomson. Cartesian genetic programming. In R. Poli *et al.*, editor, *Proceedings of the 3$^{\text{rd}}$ European Conference on Genetic Programming (EuroGP-2000)*, volume 1802 of *LNCS*, pages 121–132, Edinburgh, Apr. 2000. Springer-Verlag.

13. P. J. Olver and C. Shakiban. *Applied Linear Algebra*. Prentice Hall, 2006.

14. M. O'Neill and C. Ryan. Grammatical evolution. *IEEE Transactions on Evolutionary Computation*, 5(4):349–358, Aug. 2001.

15. U.-M. O'Reilly and F. Oppacher. Program search with a hierarchical variable length representation: Genetic programming, simulated annealing and hill climbing. In Y. Davidor *et al.*, editor, *Proceedings of the Third International Conference on Parallel Problem Solving from Nature (PPSN VI)*, number 866 in Lecture Notes in Computer Science, pages 397–406, Jerusalem, 9-14 Oct. 1994. Springer-Verlag.

16. R. Poli, W. B. Langdon, and N. F. McPhee. *A field guide to genetic programming*. Published via `http://lulu.com` and freely available at `http://www.gp-field-guide.org.uk`, 2008. (With contributions by J. R. Koza).

17. R. Poli and N. F. McPhee. General schema theory for genetic programming with subtree-swapping crossover: II. *Evolutionary Computation*, 11(2):169–206, 2003.

18. J. R. Rice. The algorithm selection problem. *Advances in Computers*, 15:65–118, 1976.

19. M. Tomassini, L. Vanneschi, P. Collard, and M. Clergue. A study of fitness distance correlation as a difficulty measure in genetic programming. *Evolutionary Computation*, 13(2):213–239, Summer 2005.

20. L. Vanneschi, M. Clergue, P. Collard, M. Tomassini, and S. Vérel. Fitness clouds and problem hardness in genetic programming. In K. Deb *et al.*, editor, *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO-2004)*, volume 3103 of *Lecture Notes in Computer Science*, pages 690–701, Seattle, WA, USA, June 2004. Springer.