Plastic Fitness Predictors Coevolved with Cartesian Programs

Michal Wiglasz(✉) and Michaela Drahosova

Faculty of Information Technology, Brno University of Technology,
Božetěchova 2, 612 66 Brno, Czech Republic
{iwiglasz,idrahosova}@fit.vutbr.cz

Abstract. Coevolution of fitness predictors, which are a small sample of all training data for a particular task, was successfully used to reduce the computational cost of the design performed by cartesian genetic programming. However, it is necessary to specify the most advantageous number of fitness cases in predictors, which differs from task to task. This paper introduces a new type of directly encoded fitness predictors inspired by the principles of phenotypic plasticity. The size of the coevolved fitness predictor is adapted in response to the learning phase that the program evolution goes through. It is shown in 5 symbolic regression tasks that the proposed algorithm is able to adapt the number of fitness cases in predictors in response to the solved task and the program evolution flow.

Keywords: Fitness predictors · Cartesian genetic programming · Coevolution · Phenotypic plasticity

1 Introduction

Cartesian genetic programming (CGP) is a specific form of genetic programming (GP) and has been successfully applied to a number of challenging real-world problem domains [7]. In CGP, as well as in GP, every evolved program must be executed to find out what it does. Each program in the population is assigned a fitness value, representing the degree to which it solves the problem of interest. Often, but not always, the fitness is calculated over a set of *fitness cases*. A fitness case consists of potential program inputs and target values expected from a perfect solution as a response to these program inputs. The outputs of the evolved program are then compared with the desired outputs for given inputs. The choice of how many fitness cases (and which ones) to use is often a crucial decision since whether or not the evolved program will generalize over the entire domain depends on this choice.

In the case of digital circuit evolution, which is a typical task for CGP, it is necessary to verify whether a candidate n-input circuit generates correct responses for all possible input combinations (i.e., 2^n assignments). It was shown that testing just a subset of 2^n fitness cases does not lead to correctly working circuits [5].

In the *symbolic regression* tasks, the goal of GP system design and GP parameters' tuning is to obtain a solution with predefined accuracy and robustness.

M. Heywood et al. (Eds.): EuroGP 2016, LNCS 9594, pp. 164–179, 2016.
DOI: 10.1007/978-3-319-30668-1_11

In this case, k fitness cases are evaluated during one fitness function call, where k typically goes from hundreds to ten thousands. The time needed for evaluating a single fitness case depends on a particular application. Usually, in order to find a robust and acceptable solution a large number of fitness evaluations has to be performed. In order to reduce the evaluation time, *fitness approximation* techniques have been employed, e.g. fitness modeling [6].

Closely related concept to the fitness modeling is a *fitness prediction*, which is a low cost adaptive procedure utilized to replace the fitness evaluation. A framework for reducing the computation requirements of symbolic regression using fitness predictors has been introduced for standard genetic programming by Schmidt and Lipson [9]. The method utilizes a coevolutionary algorithm which exploits the fact that one individual can influence the relative fitness ranking between two other individuals in the same or a separate population [4]. The state of the art of coevolutionary principles has recently been summarized in the chapter of Handbook of Natural Computing [8].

Inspired by [9], we have introduced coevolving fitness predictors to CGP and have shown that by using them, the execution time of symbolic regression can significantly be reduced [12]. Fitness predictors have been represented as a constant-size array of pointers to elements in the fitness case set and operated using a simple genetic algorithm. The same coevolutionary CGP and Hillis' *competitive coevolution* approach [4] adapted for CGP have been used in the evolutionary image filter design [11]. Although the time of evolution has also been reduced, a large number of experiments had to be accomplished in order to find the most advantageous size of the fitness predictor (the number of fitness cases in predictor) for this particular task.

To solve this problem, we have introduced a new type of indirectly encoded fitness predictors which can automatically adapt the number of fitness cases used to evaluate the candidate programs [10]. However, during the evolution of fitness predictors, also large fitness predictors have to be evaluated (and then refused for a larger size), and thus plenty of fitness case evaluations have been wasted.

In this paper, we integrate *phenotypic plasticity* principles into coevolution. The phenotypic plasticity is the ability of an individual to learn how to utilize its genotype in order to adapt to the environment [1]. It was shown that a proper rate of environmental change may reduce the learning cost while evolving the solution [2,3]. Inspired by these principles, we introduce a new type of fitness predictors, operated using a simple genetic algorithm (GA), using the phenotypic plasticity in order to adapt the number of fitness cases for candidate solution evaluations and thus regulate the rate of environmental change. In the case of fitness prediction, a stable environment contains a complete fitness cases set, a highly changing environment only a few of them.

The paper is organized as follows. Section 2 introduces cartesian genetic programming and coevolution of fitness predictors. In Sect. 3, a new approach to fitness predictor encoding is presented. The proposed approach is evaluated using 5 symbolic regression benchmarks. Experimental results are discussed in Sect. 4. Finally, conclusions are given in Sect. 5.

2 Fitness Prediction in CGP

In standard CGP, candidate programs are represented in the form of directed acyclic graph, which is modeled as a matrix of $n_c \times n_r$ programmable elements (nodes). Each node is programmed to perform one of n_a-input functions defined in the set Γ. The number of primary inputs, n_i, and outputs, n_o, of the program is defined for a particular task. Each node input can be connected either to the output of a node placed in previous l columns or to one of the program inputs. Feedback is not allowed. The search is usually performed using a simple $(1 + \lambda)$ evolutionary algorithm, where usually $\lambda = 4$. Every new population consists of the best individual of the previous population and its λ offspring created using a mutation operator which modifies up to h genes of the chromosome. The state of the art of CGP has recently been summarized in a monograph [7].

In the case of *symbolic regression*, the set of fitness cases is usually constructed from experimentally obtained data. Then each of k fitness cases from the set is used to evaluate each candidate program (see Fig. 1). The fitness function of candidate program is often defined as the relative number of hits. Formally,

$$\mathrm{f}(s) = \frac{1}{k} \sum_{j=1}^{k} \mathrm{g}(y(j)), \text{ where} \tag{1}$$

$$\mathrm{g}(y(j)) = \begin{cases} 0 \text{ if } |y(j) - t(j)| \geq \varepsilon \\ 1 \text{ if } |y(j) - t(j)| < \varepsilon \end{cases} \tag{2}$$

and y is a candidate program response, t is a target response and ε is a user-defined acceptable error. The fitness evaluation is the most time consuming part in standard CGP (as well as tree-based GP).

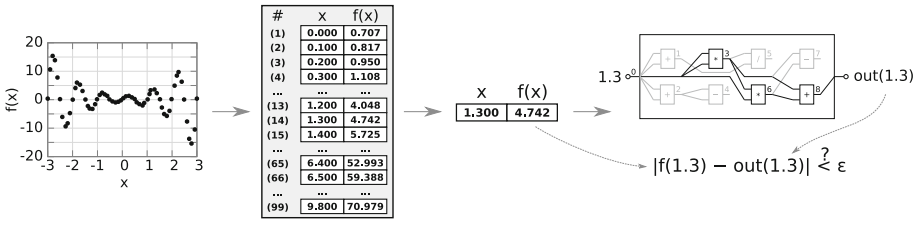


Fig. 1. Fitness evaluation of a candidate cartesian program.

2.1 Fitness Predictor

In order to reduce the total number of evaluations during each one fitness function call, fitness predictor in the form of small subset of the fitness case set have been introduced to CGP [12]. An optimal fitness predictor is sought using a simple genetic algorithm (GA) which operates with a population of fitness

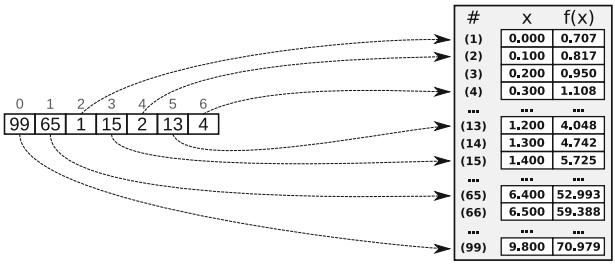


Fig. 2. Fitness predictor representation.

predictors. Every predictor is encoded as a constant-size array of pointers to elements in the training data (see Fig. 2). In addition to one-point crossover and mutation, a randomly selected predictor replacing the worst-scored predictor in each generation has been introduced as a new genetic operator of GA. The goal of the evolution of predictors is to minimize the relative error of fitness prediction and the expensive exact fitness evaluation.

2.2 Coevolution of Cartesian Programs and Fitness Predictors

The aim of coevolving fitness predictors and programs is to allow both solutions (programs) and fitness predictors to enhance each other automatically until a satisfactory problem solution is found. There are two concurrently working populations: (1) candidate programs (syntactic expressions) evolving using CGP and (2) fitness predictors evolving using GA. The overall scheme of the coevolutionary algorithm is shown in Fig. 3.

Evolution of candidate programs is based on principles of CGP. The fitness function for CGP is defined as the relative number of hits. There are, in fact, two fitness functions for candidate program s. While the exact fitness function $f_{exact}(s)$ utilizes the complete set of fitness cases, the predicted fitness function $f_{predicted}(s)$ employs only selected fitness cases. Formally,

$$\mathrm{f}_{\mathrm{exact}}(s) = \frac{1}{k}\sum_{j=1}^{k} g\left(y\left(j\right)\right) \tag{3}$$

$$\mathrm{f}_{\mathrm{predicted}}(s) = \frac{1}{m}\sum_{j=1}^{m} g\left(y\left(j\right)\right) \tag{4}$$

where k is the number of fitness cases in the set of fitness cases and m is the number of fitness cases in the fitness predictor. The $f_{predicted}$ is used to evaluate the candidate programs in the population. The f_{exact} is used during the predictor training.

The predictor training is accomplished as follows. The *archive of trainers* is generated and updated in response to the candidate program evolution. It consists of candidate programs with evaluated f_{exact} and is divided into two

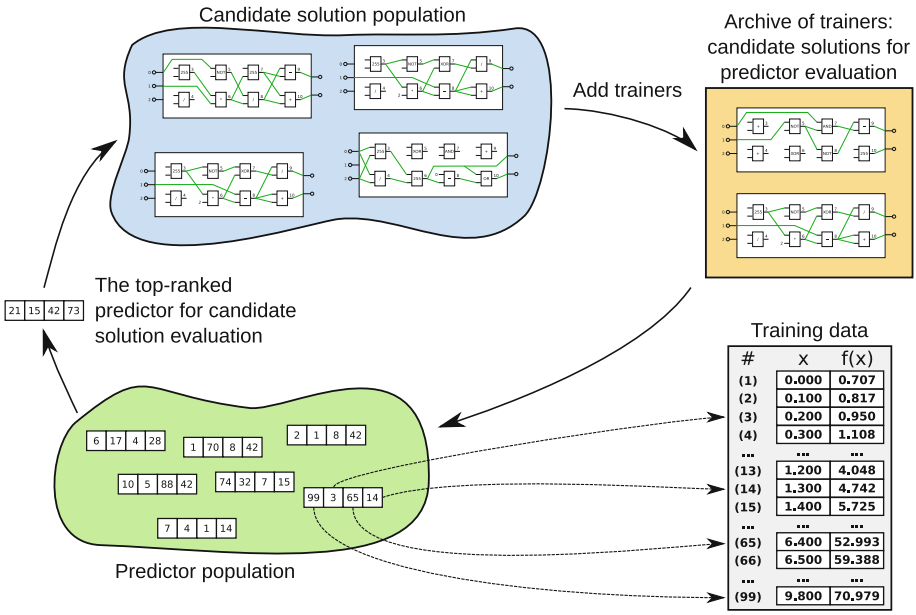


Fig. 3. Coevolution of candidate solutions and fitness predictors.

parts: The first part contains copies of top-ranked programs (with different fitness) obtained during the program evolution and the second part is periodically updated with randomly generated programs to ensure genetic diversity of the archive. The size of the archive is kept constant during the coevolution and each new trainer replaces the oldest one in the corresponding part of the archive.

The fitness value of predictor p is calculated using the *mean absolute error* of the exact and predicted fitness values of programs in the archive of trainers:

$$\mathrm{f}(p) = \frac{1}{u} \sum_{i=1}^{u} |\mathrm{f}_{exact}(i) - \mathrm{f}_{predicted}(i)| \tag{5}$$

where u is the number of candidate programs in the archive of trainers. The predictor with the best fitness value is used to predict the fitness of candidate programs in the population of candidate programs [9].

3 Proposed Method

In this paper, we propose a new approach to fitness predictor encoding. The number of fitness cases required to obtain a satisfactory solution varies from benchmark to benchmark. In order to apply coevolutionary CGP to different tasks, it is required to perform numerous experiments to find the most advantageous number of fitness cases in fitness predictors.

It can be observed that the population of solutions goes through various phases as the population's ability to adapt to the problem changes over the time [2]. A *lower fitness* phase needs less stimuli to improve solutions, but the same amount of stimuli does not lead to converge during the *higher fitness* phase. This property is discussed by Ellefsen [2], in order to reduce the learning cost. In this paper, the number of fitness cases in predictors is changed according to the latest development in the population of the candidate programs.

3.1 Plastic Directly Encoded Predictor

We propose directly encoded fitness predictors with an adaptive number of fitness cases for candidate solution evaluations. To be able to modify their size, we employ the principles of phenotypic plasticity. This allows the individual to produce different phenotypes from the same genotype, depending on the environmental conditions [2]. In the plastic fitness predictors, the phenotype is constructed by including only selected subset of genes.

The predictor genotype is a constant-size circular array of pointers to elements in the training data. Its size is equal to the total number of fitness cases. In order to produce the phenotype, the genes are read sequentially from specified position (*offset*). The genotype may contain duplicate gene values. Therefore, the gene with a value, which is already included in the phenotype, is skipped in order to prevent duplicate fitness case in predictor. The reading stops after it has processed the number of genes specified by the *readLength* variable. The *readLength* value is determined by the flow of the candidate program evolution.

The *offset* is determined by an extra gene included in the genotype, evolved by a special mutation operator, which adds a small Gaussian random number to the current value. Figure 4 shows an example of phenotype construction when 6 out of 10 available genes are used.

The evolution of predictors is directed by the genetic algorithm (GA). The crossover operator is modified so the split point is always selected within the active part of the genotype, which increases phenotype diversity.

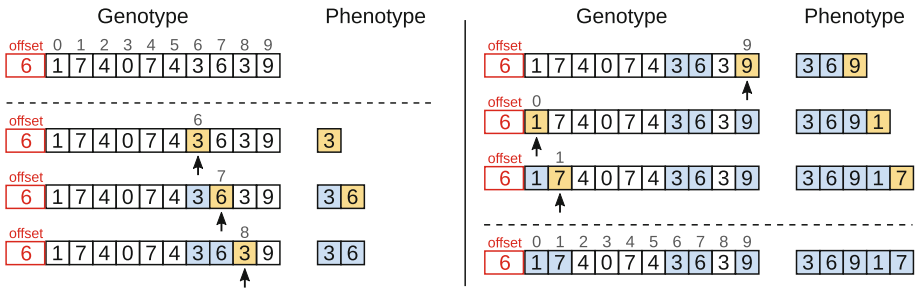


Fig. 4. Predictor phenotype construction with $offset = 6$ and $readLength = 6$.

3.2 Predictor Size Adaptation

The predictor size is adapted through the *readLength* variable. Its value is changed according to the latest development in the population of the candidate programs. It can be observed that the population goes through various phases as the population's ability to adapt to the problem changes over the time. If the ability is higher, the overall fitness increases towards better solutions, if it is lower, the fitness remains almost constant. In this case the evolution probably reached some local optimum.

The phase of evolution can be described in terms of the evolution speed which we express as follows:

$$v = \frac{\Delta f_{exact}}{\Delta G}, \tag{6}$$

where ΔG is the number of generations between two last fitness changes of CGP population parent (top-ranked programs) and Δf_{exact} is the difference of exact fitness values of these parents. Although the evolution of programs is guided by the predicted fitness, the speed can be negative, because it is calculated from the exact fitness.

It is necessary to set the lower boundary of the predictor size. If the prediction is based on only a few fitness cases (in extreme cases on only one fitness case), over-fitting of predictors occurs. The prediction inaccuracy can be expressed as the absolute difference between predicted and exact fitness:

$$I = |f_{predicted} - f_{exact}|, \tag{7}$$

In the case the prediction inaccuracy exceeds given threshold I_{thr}, the number of fitness cases should be increased.

The *readLength* value is updated each time a new solution with better predicted fitness than parent individual is found. It can be also updated after a user-specified number of generations during which a new solution is not found. The evolution speed and prediction inaccuracy is updated and a corresponding rule is selected. The rules are based on the following assumptions:

1. If the inaccuracy exceeds the threshold ($I > I_{thr}$), the size is increased.
2. If the fitness remains unchanged ($v \approx 0$), the predictor size is decreased, which should help the evolution to leave a local optimum.
3. If the fitness decreases ($v < 0$), the evolution is probably leaving a local optimum and decreasing the size can accelerate this process.
4. If the fitness increases ($v > 0$), the predictor size is increased to make the prediction more accurate.

The purpose of these rules is to find the lowest possible predictor size while the evolution still converges. The new *readLength* value is obtained by multiplication of the previous value and a coefficient, which is selected using described rules. Experimentally obtained values of the coefficients are specified in Sect. 4.2.

4 Results

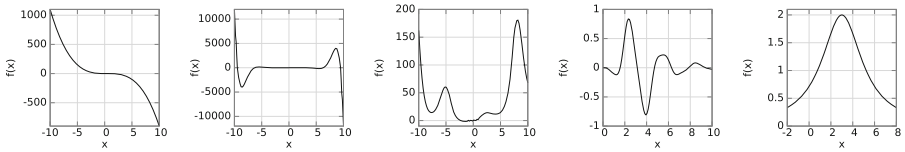
In this section, 5 symbolic regression benchmarks are introduced. Next, we present experimental results, in particular the proposed predictor behaviour and the comparisons of the proposed approach with the previously presented approaches to coevolutionary and standard CGP.

4.1 Benchmark Problems

Five symbolic regression benchmark functions (F1 – F5, see Fig. 5) were selected as training data sources for evaluation of the proposed method:

$$\begin{aligned}
&\text{F1}: f(x) = x^2 - x^3, && x = [-10 : 0.1 : 10]\\
&\text{F2}: f(x) = e^{|x|}\sin(x), && x = [-10 : 0.1 : 10]\\
&\text{F3}: f(x) = x^2 e^{\sin(x)} + x + \sin\left(\frac{\pi}{x^3}\right), && x = [-10 : 0.1 : 10]\\
&\text{F4}: f(x) = e^{-x}x^3 \sin(x)\cos(x)\left(\sin^2(x)\cos(x) - 1\right), && x = [0 : 0.05 : 10]\\
&\text{F5}: f(x) = \frac{10}{(x-3)^2 + 5}, && x = [-2 : 0.05 : 8]\,.
\end{aligned}$$

To form the training data, 200 equidistant distributed samples were taken from each function. Functions F1 – F5 are taken from [12] and all functions F1 – F5 were used in order to evaluate coevolution of CGP and both directly and indirectly encoded predictors [10,12].



(a) F1. (b) F2. (c) F3. (d) F4. (e) F5.

Fig. 5. Symbolic regression benchmark functions used for evaluation.

4.2 Experimental Setup

The setup of the program evolution is used according to literature [12], i.e. $\lambda = 12$, $n_i = 1$, $n_o = 1$, $n_c = 32$, $n_r = 1$, $l = 32$, every node has two inputs (i_1, i_2), $\Gamma = \{i_1 + i_2,\ i_1 - i_2,\ i_1 \cdot i_2,\ \frac{i_1}{i_2},\ \sin(i_1),\ \cos(i_1),\ e^{i_1},\ \log(i_1)\}$ and the maximum number of mutations per individual is $h = 8$. The program fitness function is defined as the relative number of hits (see Eqs. 3 and 4). For the benchmarks, the user-defined acceptable errors ε are as follows: F1, F2: 0.5; F3: 1.5; F4, F5: 0.025. The acceptable number of hits is 96 %.

Table 1. Rules used to adapt the *readLength* value.

Priority	Condition	Coefficient
1	$I > I_{thr}$	1.2
2	$\|v\| \leq 0.001$	0.9
3	$v < 0$	0.96
4	$0 < v \leq 0.1$	1.07
5	$v > 0.1$	1

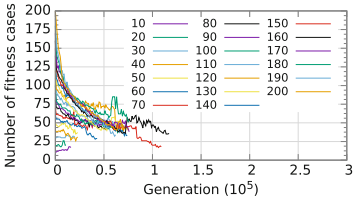
The predictor size is adapted as follows: The *readLength* value is initialized with 5 genes (the influence of the initial value is discussed in Sect. 4.3), its minimum is limited to 5 and the maximum is the total number of fitness cases. The value is updated after a new top-ranked program is found, or after 5000 generations since last update. The new *readLength* value is given as $readLength \cdot coefficient$. Experimentally obtained coefficient values are shown in Table 1. The threshold $I_{thr} = 15$ is chosen. Conditions are set according to assumptions in Sect. 3.2. If more conditions are fulfilled at the same time, the value is updated according to the priority (see Table 1).

4.3 Ability to Adapt the Number of Fitness Cases

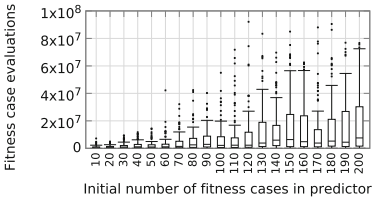
In order to confirm that the proposed algorithm is able to adapt the predictor size on a given task, we plot the progress of the average number (out of 100 independent runs) of fitness cases in top-ranked predictor during the evolution flow with respect to the initial predictor sizes. It can be seen in Fig. 6 that the size converges to the similar value independently of an initial size and the final predictor size differs for each benchmark.

The success rate is the same for each initial size setting. In the case of benchmarks F1 – F3, a larger initial size leads to more fitness case evaluations required to find an acceptable solution, see Fig. 6. This does not hold for benchmarks F4 and F5, where all settings lead to a comparable number of evaluations. The reason is that the predictor size converges in approximately 10^5 generations, while it takes much more time (approx. $3.7 \cdot 10^6$ generations) to find a satisfactory solution (see Table 2), so the effect of different predictor size in the beginning of the evolution is negligible. Note that a satisfactory solution for the benchmark F1 is found in less generations than it is necessary for the predictor size to converge.

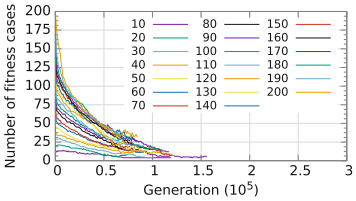
In general, it is advantageous to begin with a lower number of fitness cases in predictor, which in some cases leads to a lower number of evaluations and thus the design process acceleration. On the other hand, if the initial size is too low to find an acceptable solution, it will be automatically increased without a significant impact on the run time.



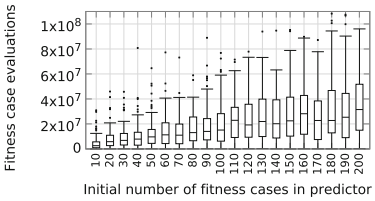
(a) Benchmark F1.



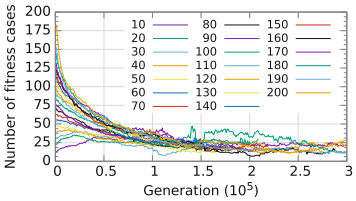
(b) Benchmark F1.



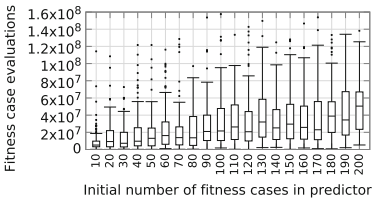
(c) Benchmark F2.



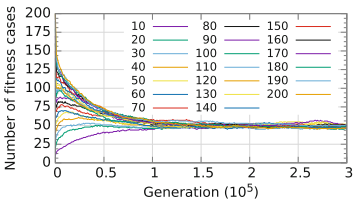
(d) Benchmark F2.



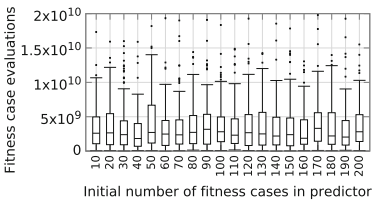
(e) Benchmark F3.



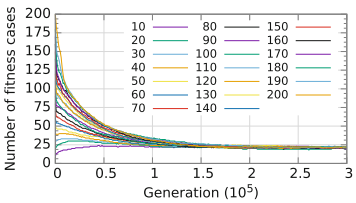
(f) Benchmark F3.



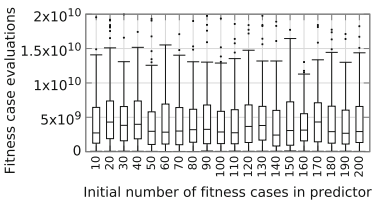
(g) Benchmark F4.



(h) Benchmark F4.



(i) Benchmark F5.

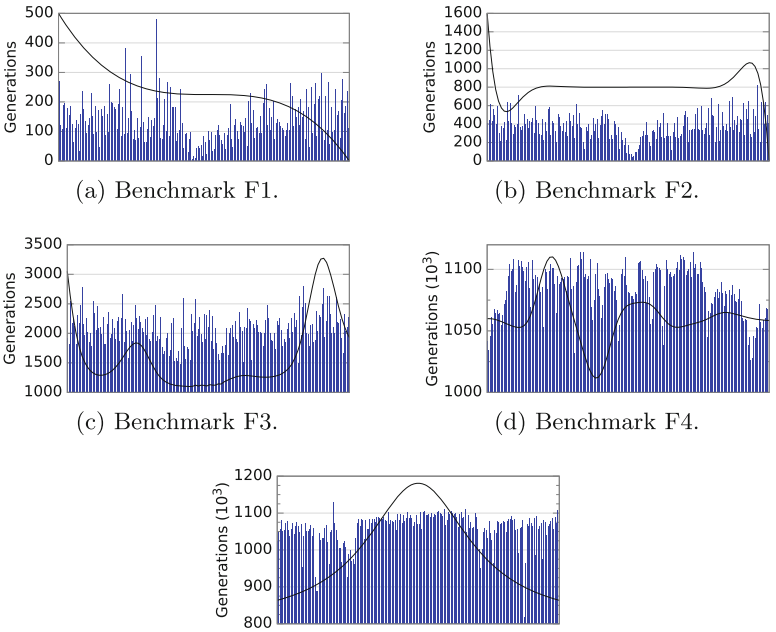


(j) Benchmark F5.

Fig. 6. Different initial predictor sizes: The average number of fitness cases in predictors and the number of fitness case evaluations necessary to find an acceptable solution.

4.4 Predictor Behaviour

In this section, we discuss how a predictor selects a subset of training data capable of guiding the evolution towards the satisfactory solution. We plot the distribution of fitness cases selected by predictors during the whole coevolutionary process out of 100 independent runs. Figure 7 show the frequency of fitness cases addressed by the top-ranked predictors during the coevolution flow. It can be seen that for benchmarks F1 and F2 predictors focus more on peaks and valleys than on flexes. On the other hand, in the case of F3 – F5, the samples are well distributed over the data set. Considering all fitness cases addressed by the predictor focused on the interesting regions (peaks and valleys) of the training data, the predictor would represent the maximum error. Note that this characteristic is desired in the Hillis' competitive coevolutionary approach [4], but is improper while requiring the predicted fitness corresponding to the exact fitness. Furthermore, fitness cases addressed by the fitness predictors are variable in response to the program evolution flow. The program evolution forces the predictors to contain two types of fitness cases, some of them are easy, others difficult, for a particular program.



(a) Benchmark F1.

(b) Benchmark F2.

(c) Benchmark F3.

(d) Benchmark F4.

(e) Benchmark F5.

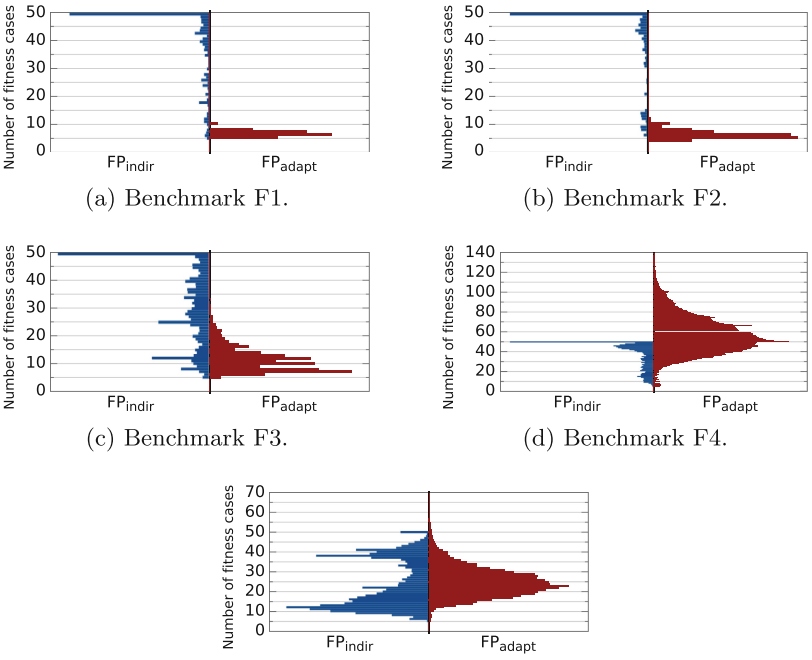
Fig. 7. Frequency of fitness cases in predictors used for programs evaluations.

4.5 Comparison of the Predictor Size

Indirectly encoded fitness predictors based on the principles of CGP (below FP_{indir}) were proposed in order to overcome the problem with selection of the most advantageous number of fitness cases used for fitness evaluation. In FP_{indir}, the predictor size parameter is included in the fitness function. Most of the sizable predictors are then rejected, but evaluated during the predictor training, which results into wasted evaluations. In order to reduce the computational cost of predictor fitness evaluations during the training, a limit of predictor size was introduced. Then, the maximum size of fitness predictor evolved using FP_{indir} was 50 fitness cases.

The size of the proposed adaptive directly encoded predictors (FP_{adapt}) varies only a little in the following generations, depending on coefficients (see Table 1). The number of fitness cases in the active predictor is thus changed only in small steps and no limit of the predictor size is necessary.

Figure 8 shows the number of fitness cases in the top-ranked predictor during the coevolution flow (left part of figures shows FP_{indir}, right part FP_{adapt}). In general, the preferred number of fitness cases differs from benchmark to benchmark. It can be seen that for the benchmarks F1 and F2 (in which only some of the first predictors are used) the preferred size of fitness predictor is the maximum value (50 fitness cases) for FP_{indir} and near to the initializing value for FP_{adapt} approach (6 – 7 fitness cases). For the benchmark F3, the maximum



(a) Benchmark F1. (b) Benchmark F2.

(c) Benchmark F3. (d) Benchmark F4.

(e) Benchmark F5.

Fig. 8. Number of fitness cases in predictors used for program evaluations.

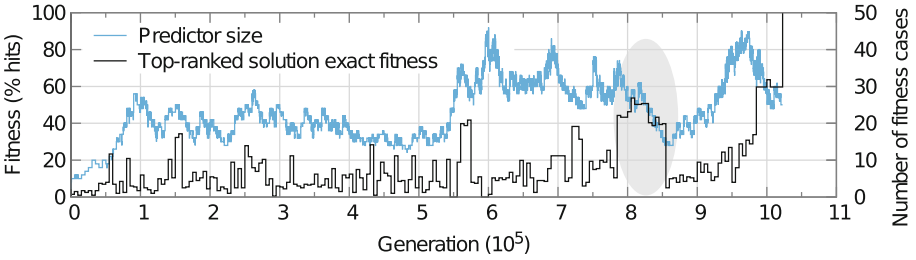


Fig. 9. Relation between the exact fitnesses of top-ranked candidate program and the size of predictor during a typical run for the F5 benchmark.

value is also preferred in FP_{indir} approach (because the evolution of predictors does not have enough time to adapt), but in FP_{adapt} the preferred value is between 7 and 12 fitness cases. Benchmark F4 is an example of how the limit of the predictor size in FP_{indir} could be restrictive. FP_{adapt} predictor size is distributed around 52 fitness cases, whereas FP_{indir} leads to predictors using 45 to 50 fitness cases and must not exceed 50. For benchmark F5, we can observe two peaks (in 12 and 38 fitness cases) for FP_{indir} predictors, but only one peak distributed around 25 fitness cases for FP_{adapt}. Note that FP_{indir} evolution allows fast changes of predictor size in contiguous generations and thus cause skips between distant values of predictor size in response to the program evolution flow. Conversely, FP_{adapt} evolution provides only small changes of predictor size in contiguous generations. The FP_{adapt} preferred predictor size, for benchmark F5, lies between preferred predictor sizes of the FP_{indir} approach, in the middle.

Although the average preferred size of predictor (out of 100 runs) in FP_{adapt} approach converges to the single value for a particular task, this trend is not so obvious while analyzing a single run. During a single run, the predictor size changes in response to the current development in the program population. Figure 9 shows the exact fitness of top-ranked program and the number of fitness cases in predictor used for program evaluation during a typical coevolutionary run for the F5 benchmark. It can be seen that the predictor size is first increased towards the preferred value and then it reacts on the development of candidate program. In this example the evolution seems to have reached a local optimum after approximately $8 \cdot 10^5$ generations which leads to decreasing of the predictor size. Around generation $8.5 \cdot 10^5$ the fitness of top-ranked program drops significantly as the evolution left the local optimum and the number of fitness cases starts to increase again, to increase accuracy of the fitness prediction.

4.6 Comparisons of Various Approaches to Fitness Prediction in CGP

The proposed coevolution employing adaptive directly encoded fitness predictors (FP_{adapt}) is compared with the original fixed-size directly encoded predictors

Table 2. Comparison of standard CGP (CGP_{STD}), coevolutionary CGP with directly encoded constant-size (FP_{const}) and adaptive predictors (FP_{adapt}) and coevolutionary CGP with indirectly encoded CGP-based predictors (FP_{indir}). For each benchmark, the best result is marked in bold font.

	Algorithm	F1	F2	F3	F4	F5
Success rate	CGP_{STD}	100 %	100 %	91 %	5 %	27 %
	FP_{const}	100 %	100 %	100 %	33 %	43 %
	FP_{adapt}	100 %	100 %	100 %	99 %	87 %
	FP_{indir}	100 %	100 %	100 %	100 %	90 %
Generations to converge (median)	CGP_{STD}	$8.66 \cdot 10^3$	$3.09 \cdot 10^4$	$1.17 \cdot 10^5$	$4.13 \cdot 10^6$	$3.25 \cdot 10^6$
	FP_{const}	$2.08 \cdot 10^3$	$1.07 \cdot 10^4$	$\mathbf{2.60 \cdot 10^4}$	$1.13 \cdot 10^7$	$7.32 \cdot 10^6$
	FP_{adapt}	$3.06 \cdot 10^3$	$1.24 \cdot 10^4$	$4.10 \cdot 10^4$	$2.42 \cdot 10^6$	$5.00 \cdot 10^6$
	FP_{indir}	$\mathbf{1.00 \cdot 10^3}$	$\mathbf{2.25 \cdot 10^3}$	$4.11 \cdot 10^4$	$\mathbf{1.47 \cdot 10^6}$	$\mathbf{1.74 \cdot 10^6}$
Fitness case evaluations to converge (median)	CGP_{STD}	$2.09 \cdot 10^7$	$7.45 \cdot 10^7$	$2.82 \cdot 10^8$	$9.96 \cdot 10^9$	$7.84 \cdot 10^9$
	FP_{const}	$\mathbf{4.41 \cdot 10^5}$	$3.09 \cdot 10^6$	$7.40 \cdot 10^6$	$2.31 \cdot 10^9$	$2.18 \cdot 10^9$
	FP_{adapt}	$6.27 \cdot 10^5$	$\mathbf{1.26 \cdot 10^6}$	$\mathbf{4.60 \cdot 10^6}$	$1.47 \cdot 10^9$	$1.53 \cdot 10^9$
	FP_{indir}	$7.43 \cdot 10^5$	$1.60 \cdot 10^6$	$1.90 \cdot 10^7$	$\mathbf{8.05 \cdot 10^8}$	$\mathbf{8.78 \cdot 10^8}$

(FP_{const}), indirectly encoded CGP-based predictors (FP_{indir}) and standard CGP without coevolution (CGP_{STD}).

FP_{const} is used according to literature [12], i.e. 12 fitness cases in chromosome, 32 individuals in predictor population, 2-tournament selection, a single-point crossover and the mutation probability 0.2. The same setup is used for FP_{adapt}, except the number of fitness cases, which is variable.

The algorithms are compared in terms of the success rate (the number of runs, giving a solution with predefined quality), the number of generations and the number of fitness case evaluations to converge (in order to compare the computational cost). Table 2 gives the median values calculated of 100 independent runs for each benchmark F1 – F5.

It can be seen in the Table 2 that both adaptive approaches, FP_{adapt} and FP_{indir}, have the highest success rate in all benchmarks. The difference is in the number of generations and fitness case evaluations required to converge. As described in Sect. 4.4, the FP_{adapt} uses fewer fitness cases than FP_{indir} for benchmarks F1 – F3. For benchmarks F1 and F2, this leads to a larger number of generations to converge using FP_{adapt} compared to FP_{indir}, fewer fitness case evaluations have to be performed using FP_{adapt}. This does not hold for the benchmark F3, where the number of generations is similar for both approaches. Nevertheless, for benchmarks F4 and F5, FP_{indir} needs fewer fitness case evaluations to converge, but still comparable in the order of magnitude. The size of fitness predictors in the FP_{indir} approach is limited to 50 fitness cases, to reduce larger predictor evaluations. However, FP_{adapt} approach prefers, for benchmark F4, more fitness cases in the predictor for this particular task (see Fig. 8), therefore the cost-reducing limit in FP_{indir} approach might be restrictive for more complex tasks.

In comparison with both FP_{adapt} and FP_{indir} approaches, FP_{const} required the lowest number of evaluations for the benchmark F1. In this case the satisfactory solution is found before the predictor size can adapt. However, let's note that many experiments have to be performed to find the most advantageous size of the predictor using FP_{const} approach for these benchmark tasks, while the FP_{adapt} and FP_{indir} adjust the size of the predictor during each single run in response to a particular task.

Finally, all three coevolutionary approaches beats CGP_{STD} in terms of number of fitness case evaluations to converge and thus accelerate the design process performed by CGP.

5 Conclusions

We have introduced the use of coevolution of cartesian programs with a new type of directly encoded predictors with the adaptive number of fitness cases. The proposed fitness predictors employ phenotypic plasticity and are able to modify the number of fitness cases used for program evaluation in dependence on the phase of program evolution.

Applied to the 5 symbolic regression tasks, we have found the proposed approach to outperform the original constant-size predictors, which use only 12 fitness cases for program evaluation, in terms of success rate and computational cost, expressed as the number of fitness case evaluations required to converge. We have shown that the proposed algorithm is able to adapt the predictor size on the solved problem in response to the development in candidate program evolution. As a result, it is possible to use coevolutionary CGP on a new task without the time-consuming experiments aimed at finding the most advantageous predictor size for the particular task.

Compared to coevolutionary CGP with indirectly encoded fitness predictors, the proposed predictor evolution does not produce predictors with larger predictor sizes than necessary. This reduces the number of necessary fitness case evaluations, while maintaining comparable program accuracy and robustness.

While symbolic regression is good to investigate the system behaviour, our future work will be devoted to applying the proposed approach to more complex problems, such as image filter design, and let the proposed approach and the approach employing indirectly encoded fitness predictors compete in the field in which the behaviour of the system is not so obvious.

The CGP has been applied to many different problem domains, predominantly in evolutionary design and optimization of logic networks. Hence the proposed approach will also be useful for evolvable hardware purposes and in real-world applications.

Acknowledgements. This work was supported by the Czech Science Foundation project 14-04197S. The authors thank the IT4Innovations Centre of Excellence for enabling these experiments.

References

1. Baldwin, J.M.: A new factor in evolution. Am. Nat. **30**(354), 441–451 (1896)
2. Ellefsen, K.O.: Balancing the costs and benefits of learning ability. In: Advances in Artificial Life, ECAL 2013, vol. 12, pp. 292–299. MIT Press (2013)
3. Ellefsen, K.O.: Evolved sensitive periods in learning. In: Advances in Artificial Life, ECAL 2013, vol. 12, pp. 409–416. MIT Press (2013)
4. Hillis, W.D.: Co-evolving parasites improve simulated evolution as an optimization procedure. Physica D **42**(1), 228–234 (1990)
5. Imamura, K., Foster, J.A., Krings, A.W.: The test vector problem and limitations to evolving digital circuits. In: Proceedings of the 2nd NASA/DoD Workshop on Evolvable Hardware, pp. 75–79. IEEE Computer Society (2000)
6. Jin, Y.: A comprehensive survey of fitness approximation in evolutionary computation. Soft Comput. J. **9**(1), 3–12 (2005)
7. Miller, J.F.: Cartesian Genetic Programming. Springer, Berlin (2011)
8. Popovici, E., Bucci, A., Wiegand, R.P., de Jong, E.D.: Coevolutionary principles. In: Rozenberg, G., Bäck, T., Kok, J.N. (eds.) Handbook of Natural Computing, pp. 988–1028. Springer, New York (2011)
9. Schmidt, M.D., Lipson, H.: Coevolution of fitness predictors. IEEE Trans. Evol. Comput. **12**(6), 736–749 (2008)
10. Sikulova, M., Hulva, J., Sekanina, L.: Indirectly encoded fitness predictors coevolved with cartesian programs. In: Machado, P., Heywood, M.I., McDermott, J., Castelli, M., García-Sánchez, P., Burelli, P., Risi, S., Sim, K. (eds.) Genetic Programming. LNCS, vol. 9025. Springer, Heidelberg (2015)
11. Sikulova, M., Sekanina, L.: Acceleration of evolutionary image filter design using coevolution in cartesian GP. In: Coello, C.A.C., Cutello, V., Deb, K., Forrest, S., Nicosia, G., Pavone, M. (eds.) PPSN 2012, Part I. LNCS, vol. 7491, pp. 163–172. Springer, Heidelberg (2012)
12. Šikulová, M., Sekanina, L.: Coevolution in cartesian genetic programming. In: Moraglio, A., Silva, S., Krawiec, K., Machado, P., Cotta, C. (eds.) EuroGP 2012. LNCS, vol. 7244, pp. 182–193. Springer, Heidelberg (2012)