

Genetic Learning of Computer Programs Represented by Straight Line Code

César L. Alonso^{1*} José Luis Montaña² Cruz Enrique Borges^{2**}

¹ Centro de Inteligencia Artificial, Universidad de Oviedo
Campus de Viesques, 33271 Gijón, Spain
calonso@uniovi.es

² Departamento de Matemáticas, Estadística y Computación
Universidad de Cantabria, 39005 Santander, Spain
{jose Luis.montana,cruzenrique.borges}@unican.es

Abstract. To successfully apply evolutionary algorithms to the solution of increasingly complex problems we must develop effective techniques for evolving solutions in the form of interacting coadapted subcomponents. In this paper we present an architecture which involves cooperative coevolution of two subcomponents: a genetic program and an evolution strategy. As main difference with work previously done, our genetic program evolves straight line programs representing functional expressions, instead of tree structures. The evolution strategy searches for good values for the numerical terminal symbols used by those expressions. Experimentation has been performed over symbolic regression problem instances and the obtained results have been compared with those obtained by means of Genetic Programming strategies without coevolution. The results show that our coevolutionary architecture with straight line programs performs considerably better than traditional genetic programming.

1 Introduction

Coevolutionary strategies can be considered as an interesting extension of the traditional evolutionary algorithms. Basically, coevolution involves two or more concurrently performed evolutionary processes with interactive performance. Initial ideas on modelling coevolutionary processes were formulated in [10], [2] or [7]. When a coevolutionary strategy is applied, usually separate populations are evolved using their own evolutionary parameters (i.e. genotype of the individuals, recombination operators, ...) but there is a main difference with respect to evolutionary strategies. This difference resides in the computation of the fitness of the individuals, because the evaluation process for an individual from a population is based on interactions with other individuals from the rest of populations. Two basic classes of coevolutionary algorithms have been developed: competitive algorithms and cooperative algorithms. In the first class, the fitness of an individual is determined by a series of competitions with other individuals. Competition takes place between the partial evolutionary processes coevolving and the success

* The First two authors are supported by spanish grant TIN2007-67466-C02-02

** Supported by FPU program and MTM2004-01167

of one implies the failure of the other (see, for example, [12]). On the other hand, in the second class the fitness of an individual is determined by a series of collaborations with other individuals.

The standard approach of cooperative coevolution is based on the decomposition of the problem into several partial components. The structure of each component is assigned to a different population. Then the populations are evolved in isolation from one another but in order to compute the fitness of an individual from a population, a set of collaborators are selected from the other populations. Finally a solution of the problem is constructed by means of the combination of partial solutions obtained from the different populations. Some examples of application of cooperative coevolution strategies for solving problems can be found in [17] and [4].

This paper focuses on the design and the study of several coevolutionary strategies between Genetic Programming (GP) and Evolutionary Algorithms (EA). Although in the cooperative coevolution systems the different coevolving populations usually are homogeneous (i.e. with similar genotype representations), in this case we deal with two heterogeneous populations: one composed by elements of a structure named *straight line program* (slp) that represents programs and the other one composed by vectors of real constants. The coevolution of GP and EA was recently applied for the first time with promising results (see [3]). In that case a population of trees and another one of fixed length strings were used.

We have tested the performance of our strategies on symbolic regression problem instances. The problem of symbolic regression consists of finding in symbolic form a function that fits a given finite sample set of data points. More formally, we consider an input space $X = \mathbb{R}^n$ and an output space $Y = \mathbb{R}$. We are given a set of m pairs sample $z = (x_i, y_i)_{1 \leq i \leq m}$. These examples are drawn according to an unknown probability measure ρ on the product space $Z = X \times Y$ and they are independent identically distributed (i.i.d.). The goal is to construct a function $f : X \rightarrow Y$ which predicts the value $y \in Y$ from a given $x \in X$. The criterion to choose function f is a low probability of error. The empirical error of a function f w.r.t. z is:

$$\varepsilon_z(f) = \frac{1}{m} \sum_{i=1}^m (f(x_i) - y_i)^2 \quad (1)$$

which is known as the mean square error (MSE).

In our coevolutionary processes for finding the function f , the GP will try to guess the shape of the function whereas the EA will try to adjust the coefficients of the function.

The paper is organized as follows: section 2 provides the definition of the structure that will represent the programs and also includes the details of the designed GP algorithm. In section 3 we describe the components that constitute the EA for obtaining good values for the constants. Section 4 presents the cooperative coevolutionary architecture used for solving symbolic regression problem instances. In section 5 a experimental comparative study of the performance of our coevolutionary strategies is done. Finally, section 6 draws some conclusions and addresses future research directions.

2 Genetic Programming with Straight Line Programs

In the GP paradigm, the evolved computer programs are usually represented by LISP S-expressions or by directed trees with ordered branches (see [9]). Recently, as a variant of Linear Genetic Programming, a new structure for representing programs has appeared. This structure is known as *straight line program* (slp) and it was introduced for the first time into the GP setting in [1]. A slp consists of a finite sequence of computational assignments where each assignment is obtained by applying some functional to a set of arguments that can be variables, constants or pre-computed results. The slp structure can describe complex computable functions using less amount of computational resources than GP-trees, as they can reuse previously computed results during the evaluation process. Also, a slp can describe multivariate functions by selecting a number of assignments as the output set. Now follows the formal definition of this structure, taken from [1].

Definition 1. Let $F = \{f_1, \dots, f_n\}$ be a set of functions, where each f_i has arity a_i , $1 \leq i \leq n$, and let $T = \{t_1, \dots, t_m\}$ be a set of terminals. A straight line program (slp) over F and T is a finite sequence of computational instructions $\Gamma = \{I_1, \dots, I_l\}$, where for each $k \in \{1, \dots, l\}$,

$$I_k \equiv u_k := f_{j_k}(\alpha_1, \dots, \alpha_{a_{j_k}}); \text{ with } f_{j_k} \in F, \\ \alpha_i \in T \text{ for all } i \text{ if } k = 1 \text{ and } \alpha_i \in T \cup \{u_1, \dots, u_{k-1}\} \text{ for } 1 < k \leq l.$$

The set of terminals T satisfies $T = V \cup C$ where $V = \{x_1, \dots, x_p\}$ is a finite set of variables and $C = \{c_1, \dots, c_q\}$ is a finite set of constants. The number of instructions l is the length of Γ .

Observe that a slp $\Gamma = \{I_1, \dots, I_l\}$ is identified with the set of variables u_i that are introduced by means of the instructions I_i . Thus the slp Γ can be denoted by $\Gamma = \{u_1, \dots, u_l\}$. Each of the non-terminal variables u_i can be considered as an expression over the set of terminals T constructed by a sequence of recursive compositions from the set of functions F .

Example 2. Let F be the set given by the three binary standard arithmetic operations $F = \{+, -, *\}$ and let $T = \{1, x_1, x_2, \dots, x_n\}$ be the set of terminals. Any slp Γ over F and T represents a n -variate polynomial with integer coefficients.

An output set of a slp $\Gamma = \{u_1, \dots, u_l\}$ is any set of non-terminal variables of Γ , that is $O(\Gamma) = \{u_{i_1}, \dots, u_{i_t}\}$. Provided that $V = \{x_1, \dots, x_p\} \subset T$ is the set of terminal variables, the function computed by Γ , denoted by $\Phi_\Gamma : I^p \rightarrow O^t$, is defined recursively in the natural way and satisfies $\Phi_\Gamma(a_1, \dots, a_p) = (b_1, \dots, b_t)$, where b_j stands for the value of the expression over V of the non-terminal variable u_{i_j} when we substitute the variable x_k by a_k ; $1 \leq k \leq p$.

In our case, for solving symbolic regression problem instances, the individuals will be slp's over a set F of functions and a set T of terminals. The elements of T that are constants, i.e. $C = \{c_1, \dots, c_q\}$, they are not in principle fixed numeric values but they are references to numeric values. Hence, specializing each c_i to a fixed value we obtain

a specific slp whose corresponding semantic function is a candidate solution for the problem instance.

For the construction of each individual Γ over F and T of the initial population, we adopt the following process: for each instruction $u_k \in \Gamma$ first an element $f \in F$ is random selected and then the function arguments of f are also randomly chosen in $T \cup \{u_1, \dots, u_{k-1}\}$ if $k > 1$ and in T if $k = 1$. We will keep homogeneous populations of equal length individuals. In this sense, note that given a slp $\Gamma = \{u_1, \dots, u_l\}$ and $L \geq l$, we can construct the slp $\Gamma' = \{u_1, \dots, u_{l-1}, u'_l, \dots, u'_{L-1}, u'_L\}$, where $u'_L = u_l$ and u'_k , for $k = l$ to $L - 1$, is any instruction satisfying the conditions in the slp's definition. If we consider the same output set for Γ and Γ' is easy to see that they are equivalent because they represent the same computable function, i.e. $\Phi_\Gamma \equiv \Phi_{\Gamma'}$.

Consider now a symbolic regression problem instance represented by a sample set $z = (x_i, y_i) \in \mathbb{R}^n \times \mathbb{R}$, $1 \leq i \leq m$, and let Γ be a specific slp over F and T obtained by giving values to the constant references $C = \{c_1, \dots, c_q\}$. In this situation, the fitness of Γ is defined by the following expression:

$$\mathcal{F}_z(\Gamma) = \varepsilon_z(\Phi_\Gamma) = \frac{1}{m} \sum_{i=1}^m (\Phi_\Gamma(x_i) - y_i)^2 \quad (2)$$

That is, the fitness is the empirical error of the function computed by Γ , w.r.t. the sample set of data points z .

The recombination operators are those described in [1] and reproduced below followed by an example.

Definition 3. (*slp-crossover*) Let $\Gamma = \{u_1, \dots, u_L\}$ and $\Gamma' = \{u'_1, \dots, u'_L\}$ be two slp's over F and T . First, a position k in Γ is randomly selected; $1 \leq k \leq L$. Let $S_{u_k} = \{u_{j_1}, \dots, u_{j_m}\}$ the set of instructions of Γ involved in the evaluation of u_k with the assumption that $j_1 < \dots < j_m$. Next we randomly select a position t in Γ' with $m \leq t \leq L$ and we modify Γ' by making the substitution of the subset of instructions $\{u'_{t-m+1}, \dots, u'_t\}$ in Γ' , by the instructions of Γ in S_{u_k} suitably renamed. The renaming function \mathcal{R} applied over the elements of S_{u_k} is defined as $\mathcal{R}(u_{j_i}) = u'_{t-m+i}$, for all $i \in \{1, \dots, m\}$. With this process we obtain the first offspring from Γ and Γ' . For the second offspring we symmetrically repeat this strategy, but now we begin by randomly selecting a position k' in Γ' .

Example 4. Let us consider the following slp's:

$$\Gamma \equiv \begin{cases} \mathbf{u}_1 := \mathbf{x} + \mathbf{y} \\ u_2 := u_1 * u_1 \\ \mathbf{u}_3 := \mathbf{u}_1 * \mathbf{x} \\ u_4 := u_3 + u_2 \\ u_5 := u_3 * u_2 \end{cases} \quad \Gamma' \equiv \begin{cases} \mathbf{u}_1 := \mathbf{x} * \mathbf{x} \\ \mathbf{u}_2 := \mathbf{u}_1 + \mathbf{y} \\ u_3 := u_1 + x \\ \mathbf{u}_4 := \mathbf{u}_2 * \mathbf{x} \\ u_5 := u_1 + u_4 \end{cases}$$

If $k = 3$ then $S_{u_3} = \{u_1, u_3\}$, and t must be selected in $\{2, \dots, 5\}$. Assumed that $t = 3$, the first offspring will be:

$$\Gamma_1 \equiv \begin{cases} u_1 := x * x \\ \mathbf{u}_2 := \mathbf{x} + \mathbf{y} \\ \mathbf{u}_3 := \mathbf{u}_2 * \mathbf{x} \\ u_4 := u_2 * x \\ u_5 := u_1 + u_4 \end{cases}$$

For the second offspring, if the selected position in Γ' is $k' = 4$, then $S_{u_4} = \{u_1, u_2, u_4\}$. Now if $t = 5$, the offspring will be:

$$\Gamma_2 \equiv \begin{cases} u_1 := x + y \\ u_2 := u_1 * u_1 \\ \mathbf{u}_3 := \mathbf{x} * \mathbf{x} \\ \mathbf{u}_4 := \mathbf{u}_3 + \mathbf{y} \\ \mathbf{u}_5 := \mathbf{u}_4 * \mathbf{x} \end{cases}$$

When mutation is applied to a slp Γ , the first step consists of selecting an instruction $u_i \in \Gamma$ at random. Then a new random selection is made within the arguments of the function $f \in F$ that constitutes the instruction u_i . Finally the selected argument is substituted by another one in $T \cup \{u_1, \dots, u_{i-1}\}$ randomly chosen.

There is not a specific selection process for selecting the individuals to be recombined. In fact, we consider the current population randomly reordered as a mating pool, instead of using the more usual fitness-based selection methods. We use generational replacement between populations, but in the construction process of the new population the offsprings generated do not necessarily replace their parents. After a crossover we have four individuals: two parents and two offsprings. We rank them by their fitness values and we pick one individual from each of the two first levels of the ranking. If, for example, three of the individuals have equal fitness value, we only select one of them and the one selected in the second place is in this case the worst of the four individuals. This strategy prevents premature convergence and maintains diversity in the population.

At a high level language, the genetic programming algorithm with slp's that we have implemented is as follows:

```

generate a random initial population
evaluate the individuals
while (not termination condition) do
    randomly mate the individuals in the current population
    for i= 1 to population_pairs do
        Op:= random value in [0,1]
        if (Op < Probab_cross)
            then do crossover
        else
            if (Op < Probab_cross + Probab_mut)
                then do mutation
            else
                if (Op < Probab_cross + Probab_mut
                    + Probab_repr)
                    then do reproduction

```

```

    evaluate the new individuals
    insert in New_pop
    update population with New_pop

```

Reproduction operation consists of directly copying the individuals from the current population to the new population.

3 An Evolutionary Algorithm to Adjust the Constants

In this section we describe an EA that provides good values for the numeric terminal symbols $C = \{c_1, \dots, c_q\}$ used by the population of slp's that evolves during the GP process. In this sense we assume a population $P = \{I_1, \dots, I_N\}$ constituted by N slp's over F and $T = V \cup C$. Let $[a, b] \subset \mathbb{R}$ be the search space for the constants c_i , $1 \leq i \leq q$. In this situation, each individual \bar{c} is represented by a vector of floating point numbers in $[a, b]^q$.

There are several ways of defining the fitness of a vector of constants \bar{c} , but in all of them the current population P of slp's that evolves in the GP process is needed. So, considering a sample set $z = (x_i, y_i) \in \mathbb{R}^n \times \mathbb{R}$, $1 \leq i \leq m$, defining a symbolic regression instance, and given a vector of values for the constants $\bar{c} = (c_1, \dots, c_q)$, we could define the fitness of \bar{c} with the following expression:

$$\mathcal{F}_z^{EA}(\bar{c}) = \min\{\mathcal{F}_z(I_i^{\bar{c}}); 1 \leq i \leq N\} \quad (3)$$

where $\mathcal{F}_z(I_i^{\bar{c}})$ is computed by equation 2 and represents the fitness of the slp I_i after making the substitution of the references to the constants in C , by the numeric values of \bar{c} .

Observe that when the fitness of \bar{c} is computed by means of the above expression, the GP fitness values of a whole population of slp's are also computed. This could cause too much computational effort when the size of both populations increases. In order to prevent the above situation new fitness functions for \bar{c} can be introduced, where only a subset of the population P of the slp's is evaluated. Previous work in cooperative coevolutionary architectures suggests two basic methods for selecting the subset of collaborators (see [17]): The first one in our case consists in the selection of the best slp of the current population P , considering the GP fitness obtained from the last evaluation of P . The second one selects two individuals from P : the best one and a random slp. Then evaluates both slp's with the references to constants specialized to \bar{c} and assigns the best value to the fitness of \bar{c} . In general, there are three aspects to consider when selecting the subset of collaborators. These aspects are: The degree of greediness of choosing a collaborator (*collaborator selection pressure*); the number of collaborators (*collaboration pool size*) and the method of assigning a fitness value given the multiple partial results from the collaborators (*collaboration credit assignment*).

We will use arithmetic crossover (c.f. [13]). Thus, the crossover of two individuals of our EA: \bar{c}_1 and $\bar{c}_2 \in [a, b]^q$, produce two offsprings \bar{c}'_1 and \bar{c}'_2 which are linear combinations of their parents, i.e.

$$\bar{c}'_1 = \lambda \cdot \bar{c}_1 + (1 - \lambda) \cdot \bar{c}_2; \quad \bar{c}'_2 = \lambda \cdot \bar{c}_2 + (1 - \lambda) \cdot \bar{c}_1 \quad (4)$$

In our implementation we randomly choose λ for each crossover operation.

We define a non-uniform mutation operator adapted to our search space $[a, b]^q$ which is convex ([11]). The following expressions define our mutation operator, taking for p the value 0.5.

$$\bar{c}_k^{t+1} = \bar{c}_k^t + \Delta(t, b - \bar{c}_k^t), \text{ with probability } p \quad (5)$$

and

$$\bar{c}_k^{t+1} = \bar{c}_k^t - \Delta(t, \bar{c}_k^t - a), \text{ with probability } 1 - p \quad (6)$$

for $k = 1, \dots, q$ and t being the generation number. The function Δ is defined as $\Delta(t, y) := y \cdot r \cdot (1 - \frac{t}{T})$ where r is a random number in $[0,1]$ and T represents the maximum number of generations. Note that function $\Delta(t, y)$ returns a value in $[0, y]$ such that the probability of $\Delta(t, y)$ being close to 0 increases as t increases. This property lets this operator search the space uniformly initially (when t is small), and very locally at later stages. In experiments reported by Michalewicz et al. ([11]), the following function was used $\Delta(t, y) := y \cdot r \cdot (1 - \frac{t}{T})^b$ where b is a parameter determining the degree of non-uniformity. In both cases mutation is applied according to a given probability p_m .

In our EA we will use q -tournament as the selection procedure for the choice of the individuals to be recombined. In this procedure, first q individuals of the population will be selected at random and then the best one of these q individuals will be picked.

4 Cooperative Coevolutionary Architecture

When applying standard coevolution strategies to a particular problem, the several involved populations usually evolve independently, except for evaluation. Since any individual from a population represents a subcomponent of the solution of the problem instance, some collaborators will need to be selected from the other populations in order to assess the fitness. In our case the EA for the adaptation of the constants is subordinated to the main GP process with the slp's. Hence, several collaborators are used during the computation of the fitness of a vector of constants \bar{c} , whereas only the best vector of constants is used for computing the fitness of a population of slp's.

A basic cooperative coevolutionary strategy begins by initializing both populations. First the initial fitness of the members of the slp's population are computed by using a random selected vector of constants. Then alternative generations of both cooperative algorithms in a round-robin fashion are performed. This strategy can be generalized in a natural way by the execution of alternative turns of both algorithms. We will consider a *turn* as the isolated and uninterrupted evolution of one population for a fixed number of generations. Note that when a turn consists of only one generation we obtain the basic strategy. We display below the algorithm describing this cooperative coevolutionary architecture:

```

begin
  Pop_slp := initialize GP-slp population
  Pop_const := initialize EA-constants population
  Const_collabor := random(Pop_const)
  evaluate(Pop_slp, Const_collabor)
  While not termination condition do
    Pop_slp := one_turn_GP(Pop_slp, Const_collabor)
    Collaborator_set_slp := {best(Pop_slp), random(Pop_slp)}
    Pop_const := one_turn_EA(Pop_const, Collaborator_set_slp)
    Const_collabor := best(Pop_const)
  end
end

```

As we have mentioned above, a turn is the execution of a fixed number of generations where each generation in the corresponding algorithm consists of the construction of a new population from the current population by means of the selection, recombination operators and fitness evaluation considering the corresponding collaborators.

5 Experimentation

5.1 Experimental Settings

For the experimentation we consider some instances of Symbolic Regression. The Symbolic Regression problem has been approached by Genetic Programming in several contexts. Usually, in this paradigm a population of tree-like structures encoding expressions is evolved. Recently the straight line programs have been adopted as the structure that evolves within the process. The performed experiments suggested that the use of slp's as data structure in GP seems to be more effective than the standard tree data structure (see [1]). Now we have executed several cooperative coevolutionary strategies considering the following group of four target functions and we have compared the performance of these strategies. Also we have compared the obtained results with those obtained with standard GP without coevolution.

$$f_1(x, y, z) = (x + y + z)^2 + 1 \quad (7)$$

$$f_2(x, y, z) = \frac{1}{2}x + \frac{1}{3}y + \frac{2}{3}z \quad (8)$$

$$f_3(x, y, z, w) = \frac{1}{2}x + \frac{1}{4}y + \frac{1}{6}z + \frac{1}{8}w \quad (9)$$

$$f_4(x) = e x^2 + \pi x \quad (10)$$

For every execution the sample set is constituted by 30 points. In the case of executions with f_1 , f_2 and f_3 as target functions, the sample points are randomly generated in the range $[-100, 100]$; the terminal set include six constants: $\{c_1, \dots, c_6\}$, that take values in $[0, 1]$; and the function set is $F = \{+, -, *, /\}$. When we consider the univariate function f_4 as target function, the sample points are in the range $[-\pi, \pi]$; two

constants c_1 , c_2 are considered and the function set F is incremented with the operations \sin and \cos . The operation in F denoted as $//$ indicates the protected division i.e. $x//y$ returns x/y if $y \neq 0$ and 1 otherwise.

The particular settings for the parameters of the GP with slp's are the following: population size: 200, crossover rate: 0.9, mutation rate: 0.05, reproduction rate: 0.05 and maximum length of the slp's: 32. In the case of the EA to adjust the constants, the population size will be of 100 vector of constants, crossover rate: 0.9, mutation rate: 0.1 and 2-tournament as selection procedure. For all the coevolutionary strategies, the computation of the fitness of an slp will use as collaborator the best vector of constants, whereas in order to compute the fitness of a vector of constants a collaborator set containing the best slp and another one randomly selected is used.

We have tested three coevolutionary strategies designed from the general architecture described above by means of a pseudocode. The first one, named Basic GP-EA (BGPEA), consists in the alternative execution of one generation of each cooperative algorithm. The second strategy, named Turns GP-EA (TGPEA), generalizes BGPEA by means of the execution of alternative turns of each algorithm. Finally the third strategy executes first a large turn of the GP algorithm with the slp's and then follows the execution of the EA related with the adjustment of the constants until termination condition was reached. This strategy is named Separated GP-EA (SGPEA). In the case of TGPEA we have considered the GP turn as the evolution of the population of slp's during 20 generations. On the other hand, the EA turn consists of 5 generations of evolution for the population related to the constants. SGPEA strategy divides the computational effort between the two algorithms: 50% for each one. We define the computational effort (CE) as the total number of basic operations that have been computed up to that moment. We have performed 100 executions of the algorithm for each instance and strategy and for all the executions evolution finished after 10^7 basic operations have been computed.

5.2 Experimental Results

A useful tool to compare performances of evolutionary strategies is the average over all runs of the best fitness values at termination. This measure is known as the *mean best fitness* (MBF). As it was mentioned above, the objective of the algorithm is to learn the target function. For this purpose a sample set of 30 points of the function is given and after the execution, the best individual is presented as the proposed hypothesis to the target function. The quality of the individuals is measured by means of the sample set. Hence the evolutionary process could be considered as a training process and the sample set would be the training set. Then it makes sense to consider another set of points, named the *validation set*, in order to give a more appropriate indicator about the quality of the hypothesis. The computation of a new fitness of the best individual after the training process, but now using the validation set, is known as the *validation fitness* (VF). In the following tables we compare the performance of the proposed coevolutionary strategies and we also include the traditional GP strategy with slp's without coevolution.

In table 1 we show the MBF for the studied strategies. In terms of MBF, we can see that coevolutionary strategies are not much better than standard GP. Nevertheless it seems that coevolutionary strategies TGPEA and specially SGPEA are the best ones.

Table 1. MBF calculated over 100 independent runs of each strategy applied to each target function.

MBF				
Strategy	f_1	f_2	f_3	f_4
BGPEA	1,20	18,5	27	0,08
TGPEA	1,59	7,81	15,9	0,07
SGPEA	2,17	5,68	4,60	0,03
GP	2,13	7,94	12,4	0,11

After these results we can state that considering only the training set, some of the co-evolutionary strategies slightly outperform traditional GP.

Table 2. VF for the best individual of the best execution for each strategy, considering a set of 200 points as validation set

VF				
Strategy	f_1	f_2	f_3	f_4
BGPEA	$2,81 \cdot 10^{-24}$	3,12	30	$4,49 \cdot 10^{-30}$
TGPEA	$2,75 \cdot 10^{-24}$	$1,19 \cdot 10^{-11}$	3,9	$2,14 \cdot 10^{-14}$
SGPEA	0,00	$1,82 \cdot 10^{-16}$	$5,22 \cdot 10^{-3}$	$4,41 \cdot 10^{-25}$
GP	$1,01 \cdot 10^{-32}$	$2,27 \cdot 10^{-1}$	1,03	$1,08 \cdot 10^{-3}$

Table 2 presents results about a more appropriate quality measure for the learning processes in order to predict values of the target function over points not belonging to the training set. For this purpose we consider a validation set that includes 200 new points randomly generated for each function. The results of table 2 represent the VF or MSE of the best individual from the best execution of each strategy, over the validation set. From these results we can state that for function f_1 all the strategies perform quite well. It seems that the evolutive algorithm to adjust the constants does not give any benefit in the case of target function f_1 , and the constants are easily determined with a standard GP procedure. On the other hand, for function f_4 the coevolutionary methods clearly outperform the GP strategy. In this case the cooperative coevolutionary architectures are fundamental for finding the coefficients of the function. If we compare the three coevolutionary methods, we can see that TGPEA and SGPEA performs better than BGPEA, but BGPEA obtains the best result with function f_4 .

The above mentioned facts are more visible in table 3 where we have reproduced the results of table 2 but considering as zero every VF lower than 10^{-15} . Now is more clear that for all problem instances the best strategy is SGPEA, which first evolves the population of slp's obtaining the shape of the target function and then adjusts the constants by means of an evolutionary algorithm. The method TGPEA that execute

Table 3. VF from table 2, rounding to zero those results lower than 10^{-15}

Strategy	VF			
	f_1	f_2	f_3	f_4
BGPEA	0,00	3,12	30	0,00
TGPEA	0,00	$1,19 \cdot 10^{-11}$	3,9	$2,14 \cdot 10^{-14}$
SGPEA	0,00	0,00	$5,22 \cdot 10^{-3}$	0,00
GP	0,00	$2,27 \cdot 10^{-1}$	1,03	$1,08 \cdot 10^{-3}$

alternative turns of each cooperative algorithm also performs considerably better than traditional GP.

6 Conclusive Remarks and Future Research

We have designed several cooperative coevolutionary strategies between a genetic program and an evolutionary algorithm. The genetic program evolves straight line programs that represent functional expressions whereas the evolutionary algorithm optimizes the values of the constants used by those expressions. The straight line programs have been introduced in the GP approach in a previous work ([1]) and they maintain good performance in solving the symbolic regression problem. This problem can be thought of a supervised learning problem (see [14]) where the hypothesis class is the search space described by the genotypes of the evolving structures. Experimentation has been performed on a group of four target functions and we have compared the performance between the studied strategies and the standard slp-GP approach without coevolution. Experimental results have showed that the coevolutionary architectures clearly outperform the traditional GP algorithm. This fact is more clear when we consider a validation set of points randomly generated instead of the training set used for the evolution process. Hence, we can conclude that coevolution seems to be a good tool for learning programs represented by straight line code.

Future work can be addressed to several research directions. We could include some local search procedure into the EA that adjusts the constants. Also the fitness function could be modified, including several penalty terms to perform model regularization such as complexity regularization using the length of the slp structure or structural risk minimization based on Vapnik-Chervonenkis dimension or Covering Numbers (see [16], [5] and [6]). Another goal could be to study the behavior of our coevolutionary model without assuming previous knowledge of the length of the slp structure. To this end new recombination operators must be designed since the crossover procedure employed in this paper only applies to populations having fixed length chromosomes. Finally we might optimize the constants of our slp's by means of classical local optimization techniques such as gradient descent or linear scaling, as it was done for tree-based GP (see [8] and [15]), and compare the results with those obtained by the computational model described in this paper.

References

1. Alonso, C.L., Montaña, J.L., Puente, J.: Straight line programs: a new Linear Genetic Programming Approach. Proc. 20th IEEE International Conference on Tools with Artificial Intelligence (ICTAI), 517–524 (2008)
2. Axelrod, R.: The evolution of cooperation. New York: Basic Books (1984)
3. Vanneschi L., Mauri G., Valsecchi A., Cagnoni S.: Heterogeneous Cooperative Coevolution: Strategies of Integration between GP and GA. Genetic and Evolutionary Computation Conference (GECCO 2006), 361–368 (2006)
4. Casillas, J., Cordon, O., Herrera, F., Merelo, J.: Cooperative coevolution for learning fuzzy rule-based systems. Proc. of the Fifth Conference on Artificial Evolution (AE 2001), 311–322 (2001)
5. V. Cherkassky, X. Shao, F. Mulier, V. Vapnik: Model complexity control for regression using VC generalization bounds. IEEE Transactions on Neural Networks 10(5), 1075-1089 (1999)
6. Cucker, F., Smale, S.: On the mathematical foundations of learning. Bulletin of the American Mathematical Society 39 (1), 1–49 (2002)
7. Hillis, D.: Co-evolving parasites improve simulated evolution as an optimization procedure. Artificial Life II, SFI Studies in the Sciences Complexity 10, 313–324 (1991)
8. Keijzer, M.: Improving symbolic regression with interval arithmetic and linear scaling. Proc. of EuroGP 2003, 71–83 (2003)
9. Koza, J. R.: Genetic Programming: On the Programming of Computers by Means of Natural Selection. The MIT Press, Cambridge, MA (1992)
10. Maynard, S.: Evolution and the theory of games. Cambridge University Press (1982)
11. Michalewicz, Z., Logan, T., Swaminathan, S.: Evolutionary operators for continuous convex parameter spaces. Proceedings of the 3rd Annual Conference on Evolutionary Programming. World Scientific, 84–97 (1994)
12. Rosin, C., Belew, R.: New methods for competitive coevolution. Evolutionary Computation 5 (1), 1–29 (1996)
13. Schwefel, H.-P.: Numerical Optimization of Computer Models. New-York: John Wiley and Sons, (1981)
14. Teytaud, O., Gelly, S., Bredeche, N., Schoenauer, M. A.: Statistical Learning Theory Approach of Bloat. Proceedings of the 2005 conference on Genetic and Evolutionary Computation (GECCO), 1783-1784 (2005)
15. Topchy, A., Punch, W.F.: Faster genetic programming based on local gradient search of numeric leaf values. Proc. of GECCO 2001, 155–162 (2001)
16. V. Vapnik: Statistical learning theory. John Wiley & Sons, (1998)
17. Wiegand, R.P., Liles, W.C., De Jong, K.A.: An Empirical Analysis of Collaboration Methods in Cooperative Coevolutionary Algorithms. Proc. of GECCO 2001, Morgan Kaufmann, 1235–1242 (2001)