Straight Line Programs: A new Linear Genetic Programming Approach

César L. Alonso
Jorge Puente
Centro de Inteligencia Artificial
Universidad de Oviedo
Campus de Viesques 33271 Gijón
calonso@uniovi.es
puente@uniovi.es

José Luis Montaña
Dpto. de Matemáticas Estadística y Computación
Universidad de Cantabria
Avda de los Castros s.n.
joseluis.montana@unican.es

Abstract

Tree encodings of programs are well known for their representative power and are used very often in Genetic Programming. In this paper we experiment with a new data structure, named straight line program (slp), to represent computer programs. The main features of this structure are described and new recombination operators for GP related to slp's are introduced. Experiments have been performed on symbolic regression problems. Results are encouraging and suggest that the GP approach based on slp's consistently outperforms conventional GP based on tree structured representations.

1. Introduction

Genetic Programming (GP) can be seen as any direct evolution method of computer programs with the purpose of inductive learning. This general definition makes GP independent of the data structures used for the representation of the evolved programs. The size, the shape and the contents of these computer programs can dynamically change during the evolution process. Usually, these programs are represented either by LISP S-expressions or by directed trees with ordered branches (see [14]). Nevertheless other variants of Genetic Programming have emerged in recent years. Besides the traditional tree representation of programs, several representation models, as linear or graph representation have been developed ([3]). Linear Genetic Programming (LGP) is a GP variant that evolves sequences of instructions from an imperative programming language or from a machine language. The term *linear*, in this case, refers to the data structure used in the program representation, constituted by sequences of assignments of operations over constants or variables to another variables. With this simple representation non-linear expressions can be generated. One of the first applications of linear bit sequences in GP appears in [7]. Other recent contributions are those in [2], where a general linear approach was introduced, and also [18], where the first GP approach that operates directly on an imperative representation was presented. For a complete overview on LGP the reader is referred to [5].

This paper focuses on the study of the performance of a new data structure for representing programs in the linear GP paradigm: straight line programs (slp). In the present work straight line programs are considered as linear representations of programs, but they could also be considered as graph representations (see section 2 below). For this linear representation we develop *ad-hoc* recombination operators which seem to be more suited for symbolic regression tasks than the straightforward generalizations given by one point crossover, k point crossover and uniform crossover, commonly used in most linear GP existing approaches.

A particular class of straight line programs, known in the literature as arithmetic circuits, have a large history and constitute the underling computation model in the field of Algebraic Complexity Theory (see [6] for an overview on this subject). Arithmetic circuits with the standard arithmetic operations $\{+, -, *, /\}$ are the natural model of computation to study the computational complexity of algorithms solving problems which have an algebraic flavor. They have been used in linear algebra problems ([4] and [17]), in quantifier elimination ([9], [13]) and in algebraic geometry ([10], [11] and [12]). In [15] non-trivial lower bounds for the complexity of straight line programs and arithmetic networks solving decisional problems are exhibited. A recent theoretical study of the capacity of straight line programs as classifiers in Machine Learning can be found in [16].

We present experimental results obtained in testing our linear GP approach, based on slp's, on symbolic regression problems and compare them to results obtained on the same problems by similar approaches which use tree encoding of programs. We envision our development as the simplest possible implementation of a general scheme for evolving slp's driven by a fitness function that reflects their ability to solve the considered problem. In this sense, the results described in this paper are just a basic step towards a GP scenario in which the slp structure is used to solve real world problems. The paper is organized as follows: in section 2 we define the data structure *straight line program* as well as some properties and related concepts. Section 3 describes the slp-GP approach for solving symbolic regression problem instances. In section 4 we present some experimental results of the execution of our implemented algorithm considering several classes of target functions. Finally, section 5 draws some conclusions and addresses future research directions.

2. The data structure *straight line program*

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

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

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

Note that if we consider the *slp* Γ as the code of a program, at each instruction I_i a new variable u_i is introduced. So the number of variables that do not belong to the terminal set T, coincides with the number of instructions and also with the length of Γ. Thus, in the following we will denote a slp $\Gamma = \{I_1, \ldots, I_l\}$ by $\Gamma = \{u_1, \ldots, u_l\}$. 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. We will denote by $SLP(F, T)$ the set of all slp's over F and T.

Example Let F be a set of three binary arithmetic operations $F = \{+, -, *\}$ and let $T = \{1, x_1, x_2\}$ be the set of terminals. In this situation any slp $\Gamma \in SLP(F, T)$ is a sequence of polynomials in two variables with integer coefficients. If we consider the following slp of length 5:

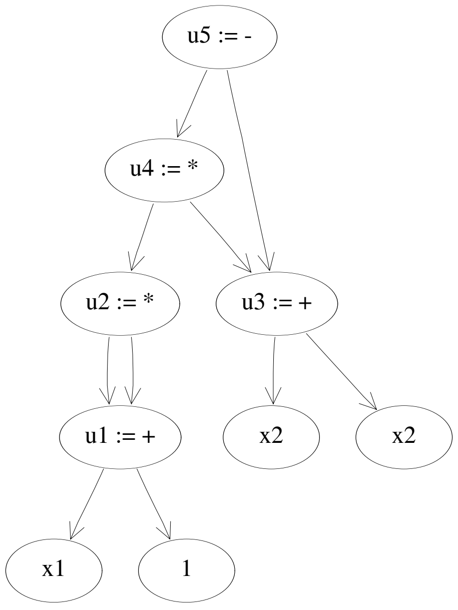
$$\Gamma \equiv \begin{cases} u_1 := x_1 + 1 \\ u_2 := u_1 * u_1 \\ u_3 := x_2 + x_2 \\ u_4 := u_2 * u_3 \\ u_5 := u_4 - u_3 \end{cases} \tag{1}$$

the term computed in u_5 is the polynomial

$$2x_2(x_1 + 1)^2 - 2x_2$$

Remark Every slp $\Gamma = \{u_1, \ldots, u_l\}$ over F and T can be represented by a directed graph $G_\Gamma = (V, E)$. The set of vertices is $V = T' \cup \{u_1, \ldots, u_l\}$, where T' contains all terminals involved in the computation. The set of edges E is constructed as follows: for every k, $1 \leq k \leq l$, we draw an edge (u_k, α_i) for each $i \in \{1, \ldots, a_{j_k}\}$. Note that T' is the set of leaves of G_Γ and is a subset of the set T of terminals . Figure 1 is a directed graph representing the slp described in equation 1

Figure 1. Directed graph representing a slp



To define a semantic function associated to a slp we consider an output space as follows. Let $\Gamma = \{u_1, \ldots, u_l\}$ be a slp over F and T. An output set of Γ, $O(\Gamma) = \{u_{i_1}, \ldots, u_{i_t}\}$, is any set of non-terminal variables of Γ. Provided that $V = \{x_1, \ldots, x_p\} \subset T$ is the set of terminal variables, the semantic function of Γ, denoted as $\Phi_\Gamma : I^p \rightarrow O^t$, satisfies $\Phi_\Gamma(a_1, \ldots, a_p) = (b_1, \ldots, b_t)$, where b_j stands for the value of the expression over V of the non terminal variable u_{i_j} when we substitute variable x_k by a_k; $1 \leq k \leq p$. Throughout this paper the output set $O(\Gamma)$ will always be constituted by only one variable, hence our slp's will compute multivariate functions with values in $\mathbb{R}$.

Given two slp's Γ_1 and Γ_2 over F and T; they will be said equivalent if they have the same semantic functions; i.e. $\Phi_{\Gamma_1} \equiv \Phi_{\Gamma_2}$.

Let $\Gamma = \{u_1, \ldots, u_l\}$ be a slp over F and T with output set $O(\Gamma) = \{u_{i_0}\}$, $1 \leq i_0 \leq l$. Then is easy to see that the slp $\Gamma' = \{u_1, \ldots, u_{i_0}\}$ is equivalent to Γ. Note that for the computation of the semantic function Φ_Γ, at most $u_1, \ldots, u_{i_0}$ are necessary. Hence $\Phi_\Gamma \equiv \Phi_{\Gamma'}$. From now on we will assume without loss of generality that the output set of Γ is $O(\Gamma) = \{u_l\}$.

2.1. Effective and non-effective code in slp's

Let us consider the following slp over $F = \{+, *, -\}$ and $T = \{1, x, y\}$,

$$\Gamma \equiv \begin{cases} u_1 := x * 1 \\ u_2 := u_1 + y \\ u_3 := u_2 * u_2 \\ u_4 := u_1 * y \end{cases}$$

Let $O(\Gamma) = \{u_4\}$ be the output of Γ. Note that if we want to compute the value of the semantic function of Γ for an input $(a_1, a_2) \in \mathbb{R}^2$, i.e. $\Phi_\Gamma(a_1, a_2) \in \mathbb{R}$, it is not necessary to compute the intermediate values of u_2 and u_3. In this case the assignments u_2 and u_3 in Γ could be considered as non-effective code and should be removed. After eliminating u_2 and u_3 and renaming the remainder assignments in Γ, a new slp Γ' is obtained:

$$\Gamma' \equiv \begin{cases} u_1 := x * 1 \\ u_2 := u_1 * y \end{cases}$$

The slp Γ' is equivalent to Γ since they have the same semantic function $\Phi : \mathbb{R}^2 \to \mathbb{R}$, $\Phi(x, y) = x * y$, if we consider for Γ' the output set $O(\Gamma') = \{u_2\}$.

In general, for computing the effective code of a slp $\Gamma = \{u_1, \ldots, u_l\}$ we first need to establish a relation in the set of the non-terminal variables. In this sense, we will say that $u_i R u_k$ if u_i appears in the functional expression assigned to u_k. Formally:

Let $\Gamma = \{u_1 \ldots, u_l\}$ be a slp over F and T. We define the following relation in the set $\{u_1, ..., u_l\}$. Assume $u_i := f_{j_i}(\alpha_1, \ldots, \alpha_{a_{j_i}})$ and $u_k := f_{j_k}(\beta_1, \ldots, \beta_{a_{j_k}})$, with $i < k$. Then $u_i R u_k$ if and only if $u_i = \beta_s$, for some s, $1 \leq s \leq a_{j_k}$. If we consider $\bar{R}$, as the reflexive and transitive closure of R, then it constitutes an order relation over $\{u_1, \ldots, u_l\}$.

Provided that $O(\Gamma) = \{u_l\}$, the effective code of Γ is the set of non-terminal variables involved in the process of evaluation of u_l for an input value of the terminal variables. We shall denote this set by $S = \{u_i \in \Gamma \ / \ u_i \bar{R} u_l\} = \{u_{i_1}, \ldots, u_{i_m}\}$, assuming that $i_1 < \ldots < i_m$. For obtaining S we construct a non decreasing chain of sets $S_0 \subseteq S_1 \subseteq \cdots \subseteq S_t = S$, where $S_0 = \{u_l\}$ and in general $S_k = S_{k-1} \cup \{u_i \in \Gamma \ / \ \exists u_j \in S_{k-1};\ u_i R u_j\}$ being R the relation defined above. It is easy to see that the above chain of sets becomes stationary after finitely many steps. In the worst case S_t becomes Γ. Also it is clear that $u_{i_m} = u_l$ and that we can construct a new slp $\Gamma' = \{u'_1, \ldots, u'_m\}$ considering the assignment instructions in S and a renaming function $\mathcal{R}$ over S such that $\mathcal{R}(u_{i_k}) = u'_k$. Note that Γ' is equivalent to Γ as a direct consequence of the construction process and it satisfies:

$$u'_i \bar{R} u'_m \ \forall i \in \{1, \ldots, m\} \tag{2}$$

If $\Gamma = \Gamma'$ we will say that Γ is an effective slp.

2.2 Computation of the semantic function

Let F be a set of functions and let T be a set of terminals with set of variables $V = \{x_1, \ldots, x_n\}$. The strategy for computing the semantic function of a slp $\Gamma = \{u_1, \ldots, u_l\}$ over F and T that consists of evaluating the non-terminal variables following the declaration order in Γ, is not a good method when Γ is not an effective slp. In practice is better to obtain the effective slp equivalent to Γ and then evaluate this one. The following algorithm describes this method.

Algorithm for computing the semantic function
Input:A slp $\Gamma = \{u_1, \ldots, u_l\}$ over F and T, with output set $O(\Gamma) = \{u_l\}$; and a vector of values $(a_1, \ldots, a_n)$ where a_i is the value of variable x_i.
Output:$\Phi_\Gamma(a_1, \ldots, a_n)$

1. Computation of the above described set $S = \{u_{i_1}, \ldots, u_{i_m}\}$, $i_1 < \ldots < i_m$, by means of the partial sets S_k.
2. For $j = 1$ to m evaluate u_{i_j} replacing each occurrence of x_i by a_i and each occurrence of u_{i_k}, with $k < j$, by its value, which was previously computed.
3. Return the value of u_{i_m}

3. GP with slp's for solving symbolic regression problems

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 $f : X \to Y$ 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:

$$\varepsilon_z(f) = \frac{1}{m} \sum_{i=1}^{m} (f(x_i) - y_i)^2 \tag{3}$$

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

The symbolic regression problem has been approached by Genetic Programming in several contexts. Usually, in this paradigm a population of tree-like structures which encode expressions, is evolved following the Darwinian principle of survival and reproduction of the fittest. Throughout this paper we adopt straight line programs as the structures that evolve within the process. One of the advantages is that the slp structure allows the result of a subexpression to be reused multiple times during calculation. This permits to express more complex calculations with less amount of instructions and the resulting individuals are, in general, more compact in terms of size. The step size of variations may also be easier to control in a slp structure than in a tree structure. In fact, using non-effective code, we can force the same size for all slp's in the selected search space. Nevertheless, how much advantage evolution can take from these features strongly depends on the design of the recombination operators.

At a very high level language, the whole genetic programming algorithm that we have implemented is as follows:

```
generate a random initial population
evaluate the individuals
while (not termination condition) do
    for i= 1 to Population_size do
        Op:= random value in [0,1]
        if (Op < Prob_cross)
        then do crossover
        if (Op < Prob_cross + Prob_mut)
        then do mutation
        if (Op < Prob_cross + Prob_mut
        + Prob_repr)
        then do reproduction
        evaluate new individuals
        insert in New_pop
    update population with New_pop
```

3.1. The Initial population

Let $F = \{f_1, \dots, f_n\}$ be a set of functions, where f_i has arity a_i $i = 1 \dots n$, and let $T = \{t_1, \dots, t_m\}$ be a set of terminals, as they appear in section 2. The generation of each slp in the initial population is done as follows.

For the first instruction u_1 select $f_{j_1} \in F$ at random. Whenever this function is selected, for each argument $i \in \{1, \dots a_{j_1}\}$ of f_{j_1}, an element α_i from T is randomly chosen.

In general the construction of the instruction u_k, $k > 1$, also begins by a random selection of $f_{j_k} \in F$. Now, for $i = 1, \dots, a_{j_k}$, we randomly choose $\alpha_i \in T \cup \{u_1, \dots, u_{k-1}\}$.

In practice, an upper bound L for the length of the slp individuals involved in the GP process, is necessary. So, given this upper bound, the first step of the generation process for each slp could be the random selection of the length $l \in \{1, \dots, L\}$.

Note that the slp's generated with the above strategy could be non-effective. Nevertheless, our aim is to permit non-effectiveness during the evolution process. On the other hand we will maintain homogeneous populations of equal length individuals. In this sense the length will be a parameter of the algorithm. For this purpose, given a slp $\Gamma = \{u_1, \dots, u_l\}$ and $L \geq l$, we can construct $\Gamma' = \{u_1, \dots, u_{l-1}, u'_l, \dots, u'_{L-1}, u'_L\}$, 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. Considering $O(\Gamma') = O(\Gamma)$, is easy to see that Γ' is equivalent to Γ.

3.2. Fitness function

In GP, we measure fitness in some way and then use this measurement to simulate nature and to control the operations that modify the structures in our artificial population. The most common approach is to assign to each individual in the population a fitness value by means of some well defined explicit evaluative procedure. So a fitness function that operates over the search space is defined. Some general type of fitness in the GP context can be seen in [14]. In our case, the procedure to compute the fitness value of a slp individual will always involve the computation of the values of the corresponding semantic function over the given sample set of values for the terminal variables. So given $z = (x_i, y_i) \in \mathbb{R}^n \times \mathbb{R}$, $1 \leq i \leq m$, for any slp Γ over F and T we will define the fitness of Γ as follows:

$$\mathcal{F}_z(\Gamma) = \varepsilon_z(\Phi_\Gamma) = \frac{1}{m}\sum_{i=1}^{m}(\Phi_\Gamma(x_i) - y_i)^2 \qquad (4)$$

That is, the fitness is the empirical error of the semantic function of Γ w.r.t. the sample set of data points z. We will use the algorithm presented in 2.2 within the process to compute the fitness of Γ.

3.3. Recombination operators

Because our representation by means of slp's consists of a finite sequence of instructions and taking into account that all individuals have the same length, the well known crossover methods such as uniform crossover, one point crossover or two point crossover, can be adapted to our situation in a natural way. However experimental testing of these generalizations does not provide good results when dealing with slp structures. Due to this situation we have designed a new "ad-hoc" crossover operation that produces another type of information exchange between the two parents. The objective is to carry subexpressions from one parent to the other. A subexpression is represented by an instruction u_i and all the instructions that are used to evaluate u_i. This is just the effective piece of code of the slp that is needed to compute the expression, over the terminal variables, associated to u_i. Formally, this crossover operator is as follows.

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; $1 \le k \le L$. We consider again the defined relation $\overline{R}$, for the description of the set:

$$S_{u_k} = \{u_j \in \Gamma \;/\; u_j \overline{R} u_k\} = \{u_{j_1}, \dots, u_{j_m}\} \tag{5}$$

with the assumption that $j_1 < \ldots < j_m$. As it was mentioned above, the set S_{u_k} is the effective piece of the code of Γ related to the evaluation of u_k. Next we randomly select a position t in Γ' with $m \le t \le L$ 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 $\mathcal{R}$ over 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'. As example, let us consider the following slp's

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

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:

$$\Gamma_1 \equiv \begin{cases} u_1 := x * x \\ \mathbf{u_2 := x + y} \\ \mathbf{u_3 := u_2 * x} \\ u_4 := u_2 * x \\ u_5 := u_1 + u_4 \end{cases}$$

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:

$$\Gamma_2 \equiv \begin{cases} u_1 := x + y \\ u_2 := u_1 * u_1 \\ \mathbf{u_3 := x * x} \\ \mathbf{u_4 := u_3 + y} \\ \mathbf{u_5 := u_4 * x} \end{cases}$$

The mutation is asexual and acts on only one parent. This operation introduces random changes in the individual. Mutation can be beneficial in reintroducing diversity in a population that may be tending to converge prematurely to a local optimum. The first step when mutation is applied to a slp Γ 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. The final step is the substitution of the selected argument by another one in $T \cup \{u_1, \dots, u_{i-1}\}$ randomly chosen. The formal definition of the mutation operation is as follows.

Mutation Let $\Gamma = \{u_1, \dots, u_L\}$ be a slp over F and T. Let $u_i = f(\alpha_1, \dots, \alpha_n)$ be the selected mutation point, where $f \in F$, $\alpha_k \in T \cup \{u_1, \dots, u_{i-1}\}$. The mutation of Γ at point i yields:

$$\Gamma' = \{u_1, \dots, u_{i-1}, u'_i, u_{i+1}, \dots, u_L\}, \tag{6}$$

where $u'_i = f(\alpha_1, \dots, \alpha_{j-1}, \alpha'_j, \alpha_{j+1}, \dots, \alpha_n)$, $\alpha_j \neq \alpha'_j \in T \cup \{u_1, \dots, u_{i-1}\}$ with $j \in \{1, \dots, n\}$. j and α'_j are both randomly selected.

Reproduction consists of copying an individual from the current population to the new population.

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 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. Also because the above process, we obtain better results if the individuals involved in the recombination operators are randomly selected, instead of using the more usual fitness-based selection methods.

4. Experiments

4.1. Experimental setting

We have run our implemented algorithm based on GP with straight line programs considering two groups of target functions. The first group of functions includes the following three functions that were also used for experimentation in [19] and [20].

$$F(x, y, z) = (x + y + z)^2 + 1 \tag{7}$$

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

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

The second group of functions is constituted by five functions of several classes: trigonometric functions, polynomial functions and one exponential function. These functions are the following:

$$\begin{aligned}
&f_1(x) = x^4 + x^3 + x^2 + x \\
&f_2(x) = e^{-sin\ 3x+2x} \\
&f_3(x) = 2.718\ x^2 + 3.1416\ x \\
&f_4(x) = cos(2x) \\
&f_5(x) = min\{\tfrac{2}{x}, sin(x) + 1\}
\end{aligned} \tag{10}$$

The experimental results obtained with the first group of target functions are compared with those obtained using standard GP based on tree structures ([20] and [19]). The experimental settings for this first group are summarized in table 1. Function $//$ indicates the protected division i.e. $x//y$ returns x/y if $y \neq 0$ and 1 otherwise. The sample points are randomly generated in the range $[-100, 100]$. The constants $c_i, 1 \leq i \leq 6$, take random values in $[0, 1]$. For each target function, the constants are fixed before the beginning of the first run and conserve the assigned values along the 100 runs.

Table 1. Summary of experiment setup for runs with F, G and K as target functions

Number of sample points	30
Population size	200
Crossover rate	0.9
Mutation rate	0.05
Reproduction rate	0.05
Maximum slp's length L	12
Function set F	$\{+,\ -,\ *,\ //\}$
Variables V	$\{x,\ y,\ z,\ w\}$
Constants C	$\{c_1, \ldots, c_6\}$
Runs per function	100

The experimental settings for the second set of test functions are basically the same as described in table 1 with the following differences: in this case there are univariate functions, the set of constants is $\{0, 1, 2\}$ for the five functions, the basic set of functions $F = \{+, -, *, //\}$ is incremented with other operations, some of the functions have a particular interval range for the set of sample points. These last two aspects are described in table 2 for each of the five functions.

For all the executions, evolution finished after 10^7 basic operations have been computed. We define the computational effort (CE) as the total number of basic operations that have been computed up to that moment.

4.2. Experimental results

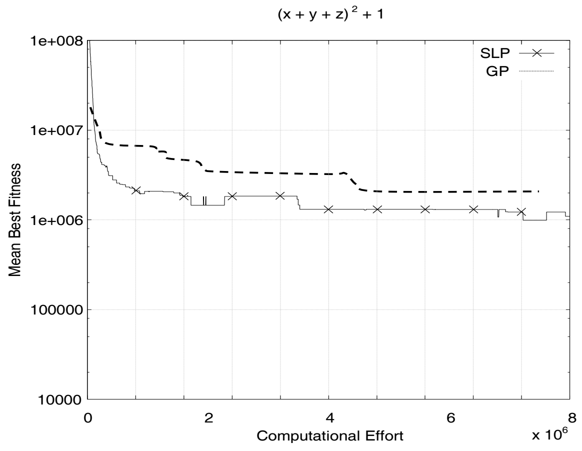
A useful tool for comparing performances of evolutionary strategies is the average over all runs of the best fitness values at termination. This measure is known as the *mean best fitness* (MBF). As in hard real-life optimization problems the solution is unknown, one common attitude is to measure performance after a specified amount of CE. For problems with known solutions, such as those considered in this work, the *success rate* (SR), defined as the ratio of successful runs with respect to the total number of runs which have been finished after reaching a specific CE, is a good indicator of algorithmic effectiveness (see [8], [1]).

Table 2. Interval ranges for sample points and function set for the second group of target functions.

Function	Range	Function set
f_1	$[-5, 5]$	$F \cup \{sqrt\}$
f_2	$[-\frac{\pi}{2}, \frac{\pi}{2}]$	$F \cup \{sqrt, sin, cos, exp\}$
f_3	$[-\pi, \pi]$	$F \cup \{sin, cos\}$
f_4	$[-\pi, \pi]$	$F \cup \{sqrt, sin\}$
f_5	$[0, 15]$	$F \cup \{sin, cos\}$

In this study we compare, for the three functions belonging to the first group, our plain GP strategy based on slp's with the GP strategy based on trees. For this comparative, the MBF and the SR are computed. Following [20], for these functions, a run will be considered successful if an individual with a fitness value lower than 30 has been evolved.

Figure 2. Best average fitness against CE over 100 independent runs for standard GP with trees and standard GP with slp's. Results on function F.



In Figures 2, 3 and 4 the mean best fitness is plotted

Figure 3. Results on function G.

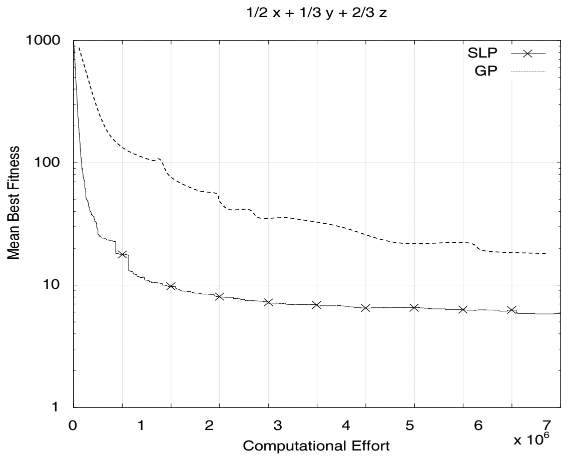
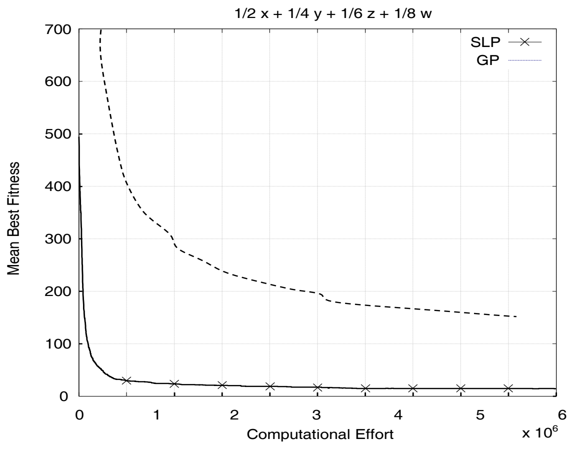


Figure 4. Results on function K.



against computational effort over the target functions F, G and K, for the two considered data structures (trees and slp's). As shown by the figures, GP with slp's outperforms standard GP with trees on every tested function.

Table 3 shows the success rate for 100 independent runs. In terms of SR the representation based on slp's is much better than the representation based on trees for the three tested functions. Considering next the successful runs and using the slp as data structure, we show in table 4 the mean best fitness and the absolute best obtained fitness (ABF) after the maximum computational effort of 10^7 basic operations was reached.

As conclusion, for the above studied target functions, the use of slp's as data structure in GP is more effective than the standard tree data structure.

The results of the execution of our algorithm over the second group of five functions are displayed in table 5.

Table 3. Success rate calculated over 100 independent runs for standard GP with trees and standard GP with slp's.

Function	Tree-GP	Slp-GP
F	55	90
G	88	100
K	72	84

Table 4. Mean best fitness and absolute best fitness calculated over the success runs for GP with slp's.

Function	MBF	ABF
F	$5{,}09 \cdot 10^{-1}$	$5 \cdot 10^{-4}$
G	3,52	$5{,}68 \cdot 10^{-3}$
K	5,26	$5{,}67 \cdot 10^{-2}$

There we present the success rate and also the MBF and the ABF for the successful runs. In this case, an execution will be considered successful if an individual of fitness near zero is found. We can observe that our GP approach based on slp's also performs quite well on this set of target functions: for all of them but f_2 the success rate is 100%.

Table 5. Success rate, mean best fitness and absolute best fitness for the GP approach based on slp's.

Function	SR	MBF	ABF
f_1	100	$3{,}40 \cdot 10^{-7}$	$2{,}15 \cdot 10^{-8}$
f_2	90	$3{,}28 \cdot 10^{-1}$	$2{,}03 \cdot 10^{-10}$
f_3	100	$9{,}04 \cdot 10^{-2}$	$2{,}13 \cdot 10^{-6}$
f_4	100	$1{,}15 \cdot 10^{-3}$	$1{,}03 \cdot 10^{-11}$
f_5	100	$8{,}40 \cdot 10^{-3}$	$6{,}94 \cdot 10^{-4}$

5. Conclusions and future research

We have experimented with a new data structure for representing computer programs inside the GP paradigm: straight line programs. This data structure allows to express complex expressions with less amount of instructions than the tree data structure. We have also designed appropriated recombination operators for slp's. Using this data structure a standard GP strategy has been implemented for solving instances of the symbolic regression problem. Experimentation has been performed on two sets of target functions. On the first set of functions our strategy based on slp's consistently outperforms standard GP: our slp encoding exhibited higher convergence rate and better quality solutions. On the second set of functions our slp encoding exhibited a success rate of 100% (except on more complicated test function f_2). From these experimental results we conclude that, inside the GP scenario, straight line programs constitute a promising data structure to represent programs.

As anticipating in the introduction, both the algorithm and the corresponding experimental results, despite offering interesting outcomes themselves, must be looked upon as the first basic step towards a more general long-term goal. The final, long term achievement that we would like to pursue is the definition of a GP scheme based on straight line encoding of programs capable of dealing with some real-world hard problems. Future work includes a more extensive experimentation over random target functions using several penalty functions to perform model regularization: complexity regularization using the length of the slp structure and structural risk minimization based on Vapnik-Chervonenkis dimension (see [22] and [21]). Another natural "next step" in our research is the combination of the plain GP approach developed here with other methods such as optimization by gradient descendent and with cooperative co-evolution as it is done in [20].

6. Acknowledgements

César Luis Alonso and José Luis Montaña are supported by spanish grant TIN2007-67466-C02-02. Jorge Puente is supported by Spanish grant TIN2007-67466-C02-01.

References

[1] T.Bäck. *Evolutionary Algorithms in Theory and Practice.* Oxford University Press, Oxford. 1996.

[2] W. Banzhaf. *Genetic Programming for Pedestrians.* S. Forrest (ed.) Proceedings of the Fifth International Conference on Genetic Algorithms (IGGA'93). pp.638–. Morgan Kaufmann, San Francisco, CA. 1993.

[3] W. Banzhaf, P. Nording, R. Keller, F. Francone. *Genetic Programming - An Introduction: On the Automatic Evolution of Computer Programs and its Applications.* Morgan Kaufmann Series in Artificial Intelligence. Morgan Kaufmann, Heilderberg - San Francisco. 1998.

[4] S. J. Berkowitz. *On comomputing the determinant in small parallel time using a small number of processors*. Information Processing Letters 18:147-150. 1984.

[5] M. Brameier; W. Banzhaf. *Linear Genetic Programming.* Springer 2007.

[6] P. Bürguisser, M. Clausen, M. A. Shokrollahi. *Algebraic Complexity Theory*. Comprehensive Studies in Mathematics. Springer. 1997.

[7] N.L. Cramer. *A Representation for the Adaptive Generation of Simple Sequential Programs.* J. Grefenstette (ed.) Procceedings of the First International Conference on Genetic Algoritms (IGA'85). pp. 183–187. 1985.

[8] A. Eiben, M. Jelasity. *A critical note on experimental research methodology in EC.* In Proc. of the Congress on Evolutionary Computation (CEC 2002). pp.582–587. 2002.

[9] N. Fitchas, A. galligo, J. Morgenstern. *Algorithmes rapides en séquentiel et parallèle pour lèlimination des quantificateurs en géométrie élémentaire*. In Delon, Dickman Gongard Seminar. Sélection déxposes 1986-1987. Vol I, (32) pp. 103-145. Publications Mathematiques de l'Université Paris 7.

[10] M. Giusti, J. Heintz. *Algorithms- disons rapide- pour la décomposition dúna varieté algébrique en composants irreductibles et équidimensionelles* Proc. MEGA'90. Algorithms in Algebraic Geometry and Applications, pp 169-194.

[11] M. Giusti, J. Heintz. *La détermination des points isolés et la dimension dúne varité algebrique peut se faire en temps polynomial.* Symposia Mathematica. pp. 216-256. Cambridge University Press.

[12] M. Giusti, J. Heintz, J. Morais, J. E. Morgentern, L. M. Pardo. Straight Line Programs in Geometric Elimination Theory. Journal of Pure and Applied Algebra (124) pp. 121-146. 1997.

[13] J. Heintz., M. F. Roy, P. Solerno. *Sur la complexité du principe de Tarski-Seidenberg*. Bulletin de la Societé Mathematique de France, 118 pp. 101-126.

[14] J. R. Koza. *Genetic Programming: On the Programming of Computers by Means of Natural Selection.* The MIT Press, Cambridge, MA 1992.

[15] J. L. Montaña; L. M. Pardo *Lower bounds for arithmetic networks.* Appl. Algebra Engrg. Comm. Comput. 4 (1993), no. 1, 1–24.

[16] J. L. Montaña. *VCD bounds for some GP genotypes* To appear in Proc. on the 18th European Conference on Artificial Intelligence. ECAI 2008.

[17] K. Mulmuley. *A parallel algorithm to compute the rank of a matrix over an arbitrary field.* Combinatorica (7) pp. 101-104. 1995.

[18] P. Nordin. *A Compiling Genetic Programming System that Directly Manipulates the Machine-Code.* K.E. Kinnear (ed.). Advances in Genetic Programming, ch. 14.pp. 311-331. MIT Press, Cambridge, MA. 1994.

[19] A. Topchy, W.F. Punch. *Faster genetic programming based on local gradient search of numeric leaf values.* In Proc. of Genetic and Evolutionary Computation Conference (GECCO 2001). pp. 155–162. 2001.

[20] L. Vanneschi, G. Mauri, A. Valsecchi, S. Cagnoni. *Heterogeneus Cooperative Coevolution: Strategies of Integration between GP and GA.* In Proc. of the 8th annual conference on Genetic and Evolutionary Computation. pp. 361–368. 2006

[21] 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)

[22] V. Vapnik: Statistical learning theory, John Willey & Sons. 1998.