International Journal on Artificial Intelligence Tools

A NEW LINEAR GENETIC PROGRAMMING APPROACH BASED ON STRAIGHT LINE PROGRAMS: SOME THEORETICAL AND EXPERIMENTAL ASPECTS

CÉSAR L. ALONSO

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

JOSÉ LUIS MONTAÑA

Dpto. de Matemáticas Estadística y Computación, Universidad de Cantabria
Avda de los Castros Santander, Spain
joseluis.montana@unican.es

JORGE PUENTE

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

CRUZ ENRIQUE BORGES

Dpto. de Matemáticas Estadística y Computación, Universidad de Cantabria
Avda de los Castros Santander, Spain
cruzenrique.borges@unican.es

Received (Day Month Year)
Revised (Day Month Year)
Accepted (Day Month Year)

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, new recombination operators for GP related to slp's are introduced and a study of the Vapnik-Chervonenkis dimension of families of slp's is done. 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.

Keywords: Genetic programming; slp; Vapnik-Chervonenkis dimension.

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 pro-

grams. 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.[1] 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.[2] 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. In 1985 one of the first applications of linear bit sequences in GP has appeared.[3] Other recent contributions are those in[4]: where a general linear approach was introduced, and also[5]: where the first GP approach that operates directly on an imperative representation was presented. For a complete overview on LGP,[6] is a good reference.

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.[7] 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[8,9]; in quantifier elimination[10,11]; and in algebraic geometry.[12,13,14] Also non-trivial lower bounds for the complexity of straight line programs and arithmetic networks solving decisional problems are exhibited.[15]

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.

As an important theoretical aspect, we also present a study of the classification complexity of slp's. In this sense we give an upper bound of the Vapnik-Chervonenkis dimension of a family of slp's that is polynomial in the length of the slp's of the family. For this purpose we construct a universal slp with a set of parameters, that represents all the elements of the considered family. 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. In section 5 we analyze the Vapnik-Chervonenkis dimension of our structure to represent computer programs. Finally, section 6 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 2.1. 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:

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

With the following result we give an order of magnitude of the number of slp's of a given length l. This number depends on the sets F and T and also on the arity of the functions belonging to F.

Proposition 2.1. *Let $F = \{f_1, \ldots, f_n\}$ be a set of functions where f_i has arity a_i, $i = 1 \ldots n$, and let $T = \{t_1, \ldots, t_m\}$ be a set of terminals. Then the number of elements $\Gamma = \{u_1, \ldots, u_l\} \in SLP(F,T)$ of length l is*

$$\prod_{j=1}^{l}(\sum_{i=1}^{n}(m+j-1)^{a_i}) \tag{2}$$

Proof. For the first non-terminal variable u_1 there exist $\sum_{i=1}^{n} m^{a_i}$ possibilities. For u_2 there are $\sum_{i=1}^{n}(m+1)^{a_i}$ options because u_1 can appears as an argument of the function u_2. In general, there are $\sum_{i=1}^{n}(m+k-1)^{a_i}$ possibilities for the non-terminal variable u_k, $1 \leq k \leq l$. Hence the number of possible slp's over F and T of length l is:

$$\prod_{j=1}^{l}(\sum_{i=1}^{n}(m+j-1)^{a_i})$$

□

Remark 2.1. 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 it is a subset of the set T of terminals. Figure 1 is a directed graph representing the slp described in equation 1

Fig. 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; $1 \leq k \leq p$.

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 $O(\Gamma) = \{u_{i_1}, \ldots, u_{i_t}\}$. Assume that $1 \leq i_1 < \ldots < i_t \leq l$. It is easy to see that the slp $\Gamma' = \{u_1, \ldots, u_{i_t}\}$ is equivalent to Γ. Note that for the computation of the semantic function Φ_Γ, at most $u_1, \ldots, u_{i_t}$ are necessary. Hence $\Phi_\Gamma \equiv \Phi_{\Gamma'}$. From now on we will assume without

loss of generality that the output set of a slp always includes the last assignment. i.e. $u_l \in O(\Gamma)$.

2.1. *Straight line programs and trees*

In this section we will present some natural relations between straight line programs and GP-trees.

Definition 2.1. Let $\Gamma = \{u_1, \ldots, u_l\}$ be a slp over F and T. For each $k \in \{1, \ldots, l\}$ we define the directed tree $\mathcal{T}_k$ associated to u_k as follows.

(i) the root of $\mathcal{T}_k$ is labeled by f_{j_k}
(ii) the subtrees of $\mathcal{T}_k$, from left to right, are $\mathcal{T}_{\alpha_1}, \ldots, \mathcal{T}_{\alpha_{a_{j_k}}}$
(iii) If α_i is an element of the terminal set T, then $\mathcal{T}_{\alpha_i}$ has only one node labeled by α_i

The method for constructing $\mathcal{T}_k$ requires a recursive process that computes previously its subtrees. This constitutes a bottom-up process that generates the subtrees from the leaves to the root. Given a slp $\Gamma = \{u_1, \ldots, u_l\}$ we define a relation in the set of the non-terminal variables $\{u_1, \ldots, u_l\}$. We will say that $u_i R u_k$ if u_i appears in the functional expression assigned to u_k. Formally:

Definition 2.2. 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 $\{u_1, \ldots, u_l\}$.

The following definition introduces the forest of trees associated to a slp.

Definition 2.3. Let $\Gamma = \{u_1, \ldots, u_l\}$ be a slp over F and T as usually. We define the forest associated to Γ, as the set of trees $\mathcal{F}(\Gamma) = \{\mathcal{T}_{i_1}, \ldots \mathcal{T}_{i_s}\}$ where $M = \{u_{i_1}, \ldots, u_{i_s}\}$ is the set of maximal elements of Γ, considering the relation $\bar{R}$.

Example 2.2. Let $\Gamma = \{u_1, \ldots, u_5\}$ be the following slp:

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

where $F = \{*, +\}$ and $T = \{x_1, x_2\}$. then $M = \{u_2, u_4, u_5\}$ and the forest associated to Γ, $\mathcal{F}(\Gamma)$ is depicted in figure 2

Remark 2.2. The above definition associates to each slp a set of trees. We can also establish the opposite association as follows. Let $\mathcal{T}$ be an ordered tree with

Fig. 2. Forest associated to the slp Γ of example 2.2

internal nodes in F and leaves nodes in T; F and T as usually. Then there exists a slp $\Gamma(\mathcal{T}) = \{u_1, \ldots, u_l\}$ associated to $\mathcal{T}$ that is unique modulo bijections from $\{u_1, \ldots, u_l\}$ to itself. $\Gamma(\mathcal{T})$ can be constructed with a strategy that consists in defining the corresponding instructions u_i, $1 \leq i \leq l$, from the leaves of $\mathcal{T}$ to the root. Following this process, is easy to see that the length l of $\Gamma(\mathcal{T})$ is greater than or equal to the height of $\mathcal{T}$. On the other hand, the equality $\mathcal{F}(\Gamma(\mathcal{T})) = \{\mathcal{T}\}$ can be deduced from the construction method of $\Gamma(\mathcal{T})$. For example, the following slp has been constructed from the tree $\mathcal{T}_5$ in figure 2.

$$\Gamma \equiv \begin{cases} u_1 := x_1 * x_2 \\ u_2 := u_1 * u_1 \\ u_3 := u_1 * u_2 \end{cases}$$

2.2. *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 $O(\Gamma) = \{u_2, u_4\}$ be the output set 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. $\Phi_\Gamma(a_1, a_2) \in \mathbb{R}^2$, it is not necessary to compute the intermediate value of u_3. In this case the assignment u_3 in Γ could be considered as non-effective code and should be removed. After eliminating u_3 and renaming the remainder assignments in Γ, a new slp Γ' is obtained:

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

The slp Γ' is equivalent to Γ since they have the same semantic function $\Phi : \mathbb{R}^2 \rightarrow \mathbb{R}^2$, $\Phi(x, y) = (x + y, x * y)$ if we consider for Γ' the output set $O(\Gamma') = \{u_2, u_3\}$.

In the case of the same slp Γ but now with output set $O(\Gamma) = \{u_4\}$, the assignments u_2 and u_3 would be non-effective code and the new slp Γ' equivalent to Γ is the following:

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

where $O(\Gamma') = \{u_2\}$.

In general, the effective code of a slp $\Gamma = \{u_1, \ldots, u_l\}$ with output set $O(\Gamma) = \{u_{i_1}, \ldots, u_{i_t} = u_l\}$ is the set of the non-terminal variables involved in the process of computing Φ_Γ. We shall denote this set by:

$$S = \{u_j \in \Gamma \ / \ \exists\ u_{i_k} \in O(\Gamma)\ s.t.\ u_j \bar{R} u_{i_k}\} = \{u_{j_1}, \ldots, u_{j_m}\} \tag{3}$$

where $\bar{R}$ is the order relation described in definition 2.2. We can assume that $j_1 < \ldots < j_m$. For obtaining S we construct a non decreasing chain of sets $S_0 \subseteq S_1 \subseteq \cdots \subseteq S_p = S$, where $S_0 = O(\Gamma)$ 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 introduced in definition 2.2. It is easy to see that the above chain of sets becomes stationary after finitely many steps. In the worst case S_p becomes Γ. Also it is clear that $u_{j_m} = u_l$ 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 $\mathcal{R}(u_{j_k}) = u'_k$. Note that Γ' is equivalent to Γ if we consider $O(\Gamma') = \{\mathcal{R}(u_{i_1}), \ldots, \mathcal{R}(u_{i_t})\}$. Also Γ' satisfies:

$$\forall i \in \{1, \ldots, m\}\ \exists u'_j \in O(\Gamma')\ s.t.\ u'_i \bar{R} u'_j \tag{4}$$

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

2.3. *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 $O(\Gamma) = \{u_{i_1}, \ldots, u_{i_t}\}$; and a vector of values $(a_1, \ldots, a_n)$ where a_i is the value of variable x_i.
Output: $\Phi_\Gamma(a_1, \ldots, a_n) = (b_1, \ldots, b_t)$

(i) Computation of the above described set $S = \{u_{j_1}, \ldots, u_{j_m}\}$, $j_1 < \ldots < j_m$, by means of the partial sets S_k. Note that $\Gamma' = \{\mathcal{R}(u_{j_1}), \ldots, \mathcal{R}(u_{j_m})\}$ is effective and equivalent to Γ.
(ii) For $k = 1$ to m evaluate $\mathcal{R}(u_{j_k}) = u'_k$ replacing each occurrence of x_i by a_i and each occurrence of u'_j, with $j < k$, by its value, which was previously computed.
(iii) Return the vector of values $(\mathcal{R}(u_{i_1}), \ldots, \mathcal{R}(u_{i_t}))$.

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:

$$\varepsilon_z(f) = \frac{1}{m} \sum_{i=1}^{m} (f(x_i) - y_i)^2 \tag{5}$$

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
```

Throughout this section, the output set of a slp $\Gamma = \{u_1, \ldots, u_l\}$ will always be constituted by only one variable, i.e. $O(\Gamma) = \{u_l\}$. Hence our slp's will compute multivariate functions with values in $\mathbb{R}$.

3.1. *The initial population*

Let $F = \{f_1, \ldots, f_n\}$ be a set of functions, where f_i has arity a_i $i = 1 \ldots n$, and let $T = \{t_1, \ldots, t_m\}$ 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 $i \in \{1, \ldots a_{j_1}\}$ 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 $i = 1, \ldots, a_{j_k}$, we randomly choose $\alpha_i \in T \cup \{u_1, \ldots, u_{k-1}\}$.

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 $l \in \{1, \ldots, L\}$.

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 $\Gamma = \{u_1, \ldots, u_l\}$ and $L \geq l$, we can construct $\Gamma' = \{u_1, \ldots, u_{l-1}, u'_l, \ldots, u'_{L-1}, u'_L\}$, 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. Note that this type of homogeneity is not possible with populations of tree structures.

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 [1]. 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:

$$\mathcal{F}it_z(\Gamma) = \varepsilon_z(\Phi_\Gamma) = \frac{1}{m}\sum_{i=1}^{m}(\Phi_\Gamma(x_i) - y_i)^2 \tag{6}$$

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 2.3 within the process to compute the fitness of Γ.

3.3. *Recombination operators*

3.3.1. *Crossover*

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. The following definition summarizes these adaptations.

Definition 3.1. Let $\Gamma_1 = \{u_1, \ldots, u_L\}$ and $\Gamma_2 = \{u'_1, \ldots, u'_L\}$ be two equal length slp's over F and T.

(i) The one-point crossover of Γ_1 and Γ_2 with breakpoint $i \in \{1, \ldots, L\}$ produces the following two slp's:

$$\Gamma'_1 = \{u_1, \ldots, u_i, u'_{i+1}, \ldots, u'_L\};\ \ \Gamma'_2 = \{u'_1, \ldots, u'_i, u_{i+1}, \ldots, u_L\}$$

(ii) The two-point crossover of Γ_1 and Γ_2 with breakpoints $i, j \in \{1, \ldots, L\}$, $i < j$ produces the following two slp's:

$$\Gamma'_1 = \{u_1, \ldots, u_i, u'_{i+1}, \ldots, u'_j, u_{j+1}, \ldots, u_L\}$$

$$\Gamma'_2 = \{u'_1, \ldots, u'_i, u_{i+1}, \ldots, u_j, u'_{j+1} \ldots, u'_L\}$$

(iii) The uniform crossover of Γ_1 and Γ_2 produces two slp's computed with the aid of a random binary vector $\alpha = (\alpha_1, \ldots, \alpha_L) \in \{0, 1\}^L$ in the following form:

$$\Gamma'_1 = \{v_1 \ldots, v_L\};\ \ \Gamma'_2 = \{v'_1, \ldots, v'_L\}$$

where $v_i = u_i$ if $\alpha_i = 0$; $v_i = u'_i$ if $\alpha_i = 1$ and $v'_i = u'_i$ if $\alpha_i = 0$; $v'_i = u_i$ if $\alpha_i = 1$; $i \in \{1, \ldots, L\}$

Example 3.1. Consider $F = \{*, +\}$, $L = 5$ and $T = \{x, y\}$. Let Γ_1 and Γ_2 be the following two slp's:

$$\Gamma_1 \equiv \begin{cases} u_1 := x + y \\ u_2 := u_1 * u_1 \\ u_3 := u_1 * x \\ u_4 := u_3 + u_2 \\ u_5 := u_3 * u_2 \end{cases} \qquad \Gamma_2 \equiv \begin{cases} u_1 := x * x \\ u_2 := u_1 + y \\ u_3 := u_1 + x \\ u_4 := u_2 * x \\ u_5 := u_1 + u_4 \end{cases}$$

One-point crossover of Γ_1 and Γ_2 with breakpoint $i = 2$ produces:

$$\Gamma'_1 \equiv \begin{cases} u_1 := x + y \\ u_2 := u_1 * u_1 \\ u_3 := u_1 + x \\ u_4 := u_2 * x \\ u_5 := u_1 + u_4 \end{cases} \qquad \Gamma'_2 \equiv \begin{cases} u_1 := x * x \\ u_2 := u_1 + y \\ u_3 := u_1 * x \\ u_4 := u_3 + u_2 \\ u_5 := u_3 * u_2 \end{cases}$$

Two-point crossover of Γ_1 and Γ_2 with breakpoints $i = 1$, $j = 3$ produces:

$$\Gamma_1' \equiv \begin{cases} u_1 := x + y \\ u_2 := u_1 + y \\ u_3 := u_1 + x \\ u_4 := u_3 + u_2 \\ u_5 := u_3 * u_2 \end{cases} \qquad \Gamma_2' \equiv \begin{cases} u_1 := x * x \\ u_2 := u_1 * u_1 \\ u_3 := u_1 * x \\ u_4 := u_2 * x \\ u_5 := u_1 + u_4 \end{cases}$$

Finally, uniform crossover of Γ_1 and Γ_2 with random binary vector $\alpha = (10110)$ produces:

$$\Gamma_1' \equiv \begin{cases} u_1 := x * x \\ u_2 := u_1 * u_1 \\ u_3 := u_1 + x \\ u_4 := u_2 * x \\ u_5 := u_3 * u_2 \end{cases} \qquad \Gamma_2' \equiv \begin{cases} u_1 := x + y \\ u_2 := u_1 + y \\ u_3 := u_1 * x \\ u_4 := u_3 + u_2 \\ u_5 := u_1 + u_4 \end{cases}$$

Besides the above recombination operations, we have also 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.

Definition 3.2. (slp-crossover) Let $\Gamma_1 = \{u_1, \ldots, u_L\}$ and $\Gamma_2 = \{u_1', \ldots, u_L'\}$ be two slp's over F and T. First, a position k in Γ_1 is randomly selected; $1 \leq k \leq L$. We consider again the defined relation $\overline{R}$, for the description of the set:

$$S_{u_k} = \{u_j \in \Gamma \ / \ u_j \overline{R} u_k\} = \{u_{j_1}, \ldots, u_{j_m}\} \tag{7}$$

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_1 related to the evaluation of u_k. Next we randomly select a position t in Γ_2 with $m \leq t \leq L$ and we modify Γ_2 by making the substitution of the subset of instructions $\{u_{t-m+1}', \ldots, u_t'\}$ in Γ_2, by the instructions of Γ_1 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 $i \in \{1, \ldots, m\}$. With this process we obtain the first offspring from Γ_1 and Γ_2. For the second offspring we symmetrically repeat this strategy, but now we begin by randomly selecting a position k' in Γ_2.

Example 3.2. Let us consider the slp's Γ_1 and Γ_2 from the example 3.1

If $k = 3$ then $S_{u_3} = \{u_1, u_3\}$, and t must be selected in $\{2, \ldots, 5\}$. 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_2 is $k' = 4$, then $S_{u_4} = \{u_1, u_2, u_4\}$. Now if $t = 5$, the offspring will be:

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

3.3.2. *Mutation*

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 $T \cup \{u_1, \ldots, u_{i-1}\}$ randomly chosen. The formal definition of the mutation operation is as follows:

Definition 3.3. Let $\Gamma = \{u_1, \ldots, u_L\}$ be a slp over F and T. Let $u_i = f(\alpha_1, \ldots, \alpha_n)$ be the selected mutation point, where $f \in F$, $\alpha_k \in T \cup \{u_1, \ldots, u_{i-1}\}$. The mutation of Γ at point i yields:

$$\Gamma' = \{u_1, \ldots, u_{i-1}, u_i', u_{i+1}, \ldots, u_L\}, \tag{8}$$

where $u_i' = f(\alpha_1, \ldots, \alpha_{j-1}, \alpha_j', \alpha_{j+1}, \ldots, \alpha_n)$, $\alpha_j \neq \alpha_j' \in T \cup \{u_1, \ldots, u_{i-1}\}$ with $j \in \{1, \ldots, n\}$. 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 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. 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 previous works.[16,17]

$$F(x, y, z) = (x + y + z)^2 + 1 \tag{9}$$

$$G(x, y, z) = \frac{1}{2}\, x + \frac{1}{3}\, y + \frac{2}{3}\, z \tag{10}$$

$$K(x, y, z, w) = \frac{1}{2}\, x + \frac{1}{4}\, y + \frac{1}{6}\, z + \frac{1}{8}\, w \tag{11}$$

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:

$$\begin{aligned} f_1(x) &= x^4 + x^3 + x^2 + x \\ f_2(x) &= e^{-sin\ 3x+2x} \\ f_3(x) &= 2.718\ x^2 + 3.1416\ x \\ f_4(x) &= cos(2x) \\ f_5(x) &= min\{\tfrac{2}{x}, sin(x) + 1\} \end{aligned} \tag{12}$$

The experimental results obtained with the first group of target functions are compared with those obtained using standard GP based on tree structures.[16,17] 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.

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 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.

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.

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	$[0, 15]$	$F \cup \{sin, cos\}$

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.[18,19]

4.2.1. *Experiment 1*

First of all we have compared the four defined crossover methods: uniform, one-point crossover, two-point crossover and slp-crossover. For this purpose we have picked the function F from the first group and the functions f_1 and f_2 from the second group. We have performed 100 executions of our algorithm for each instance and crossover operator. In the following table we show the corresponding SR and MBF over the successful runs, for each crossover method and target function. In

this case one run will be considered successful if an individual with a fitness value lower than 1 is found. We can see clearly that slp-crossover is the best recombination operator for the studied target functions. Hence, for the rest of the experiments the slp-crossover will be the selected recombination operator.

Table 3. Success rate an mean best fitness calculated over the success runs, for each crossover operator.

	F	f_1	f_2
		SR	
uniform	48	82	0
one-point	46	96	0
two-point	46	88	6
slp	52	100	90
		MBF	
uniform	$8{,}78 \cdot 10^{-1}$	$5{,}36 \cdot 10^{-4}$	–
one-point	$8{,}25 \cdot 10^{-1}$	$2{,}03 \cdot 10^{-2}$	–
two-point	$8{,}45 \cdot 10^{-1}$	$3{,}76 \cdot 10^{-2}$	$5{,}46 \cdot 10^{-1}$
slp	$9{,}14 \cdot 10^{-1}$	$3{,}40 \cdot 10^{-7}$	$3{,}28 \cdot 10^{-1}$

4.2.2. *Experiment 2*

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 [17]: for these functions, a run will be considered successful if an individual with a fitness value lower than 30 has been evolved.

Fig. 3. Best average fitness against CE over 100 independent runs for standard GP with trees and standard GP with slp's. Results on function F.

Fig. 4. Results on function G.

Fig. 5. Results on function K.

In Figures 3, 4 and 5 the mean best fitness is plotted against computational effort over the target functions F, G and K, for the two considered data structures

Table 4. 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

(trees and slp's). As shown by the figures, GP with slp's outperforms standard GP with trees on every tested function.

Table 4 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 5 the mean best fitness and the absolute best obtained fitness (ABF) after the maximum computational effort of 10^7 basic operations was reached.

Table 5. Mean best fitness and absolute best fitness calculated over the success runs for GP with slp's.

Function	MBF	ABF
F	5,63	$2{,}28 \cdot 10^{-1}$
G	3,52	$5{,}68 \cdot 10^{-3}$
K	5,26	$5{,}67 \cdot 10^{-2}$

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.

4.2.3. *Experiment 3*

The results of the execution of our algorithm over the second group of five functions are displayed in table 6. 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%.

5. Vapnik-Chervonenkis Dimension of Families of *slp*'s

In the last years GP has been applied to a range of complex learning problems including that of classification and symbolic regression in a variety of fields like quantum computing, electronic design, sorting, searching, game playing, etc. A common

Table 6. 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	$8{,}40 \cdot 10^{-3}$	$6{,}94 \cdot 10^{-4}$

feature of both tasks is that they can be thought as a supervised learning problem where the hypothesis class $\mathcal{C}$ is the search space described by the genotypes of the evolving structures. In the seventies the work by Vapnik and Chervonenkis provided a remarkable family of bounds relating the performance of a learning machine.[20,21] More recently modern presentations of the theory have appeared.[22,23] The Vapnik-Chervonenkis dimension (VCD) is a measure of the capacity of a family of functions (or learning machines) as classifiers. The VCD depends on the class of classifiers. Hence, it does not make sense to calculate VCD for GP in general, however it make sense if we choose a particular class of computer programs as classifiers (i.e. a particular genotype). Our aim in this section is to go deep into the study of formal properties of GP algorithms focusing the analysis of the classification complexity (VCD) of straight line programs. In previous work and considering computer programs, polynomial bounds in the size of the computer programs are given for VCD.[24] In the case of GP-trees, a study of their capacity of classification has been done, but considering parallel complexity more than sequential time complexity.[25] In this case and if the GP-tree internal nodes consist of infinitely differentiable algebraic functionals, sign tests and conditional statements, then the VCD depends polynomially on the height of the tree. In the case of sequential time complexity for GP-trees the upper bound is polynomial in the number of the nodes.[26] Our bounds for the VCD of slp's as classifiers, considering sequential time complexity, are of polynomial order in the length of the slp. The following definition of VC dimension is standard.[22]

Definition 5.1. Let $\mathcal{C}$ be a class of subsets of a set X. We say that $\mathcal{C}$ shatters a set $A \subset X$ if for every subset $E \subset A$ there exists $S \in \mathcal{C}$ such that $E = S \cap A$. The VC dimension of $\mathcal{C}$ is the cardinality of the largest set that is shattered by $\mathcal{C}$.

Along this section we deal with concept classes $\mathcal{C}_{k,n}$ such that concepts are represented by k real numbers, $w = (w_1, \ldots, w_k)$; instances are represented by n real numbers, $x = (x_1, \ldots, x_n)$; and the membership test to the family $\mathcal{F}_{k,n}$ is expressed by a formula $\Phi_{k,n}(w, x)$ taking as inputs the pair concept/instance (w, x) and returning the value 1 if "x belongs to the concept represented by w" and 0 otherwise.

We can think of $\Phi_{k,n}$ as a function from $\mathbb{R}^{k+n}$ to $\{0,1\}$. So for each concept w, define:

$$C_w := \{x \in \mathbb{R}^n : \Phi_{k,n}(w,x) = 1\}, \tag{13}$$

The objective is to obtain an upper bound on the VCD of the collection of sets:

$$\mathcal{C}_{k,n} = \{C_w : w \in \mathbb{R}^k\} \tag{14}$$

For boolean combinations of polynomial equalities and inequalities the following seminal result by Golberg and Jerrum is known.[24]

Theorem 5.1. Suppose $\mathcal{C}_{k,n}$ is a class of concepts whose membership test can be expressed by a boolean formula $\Phi_{k,n}$ involving a total of s polynomial equalities and inequalities, where each polynomial has degree no larger than d. Then the VC dimension V of $\mathcal{C}_{k,n}$ satisfies

$$V \leq 2k\ log_2(4eds) \tag{15}$$

Now follows our main result about the VCD of a collection of subsets classified or accepted by a family of slp's. We will say that a subset $C \subset \mathbb{R}^n$ is accepted by a slp Γ if the semantic function of Γ express the membership test to C.

Theorem 5.2. (**Main Theorem**) Let $\mathcal{G} = G_{n,l,q,D,A}$ be the collection of subsets of $\mathbb{R}^n$ that can be accepted by some *slp* Γ over F and T of length l, with $T = \{t_1, \ldots, t_n\}$, $F = \{f_1, \ldots, f_q, sign\}$; f_i rational functions of degree bounded by D and arity a_i bounded by A; *sign* is a function that outputs 1 if its input is greater or equal than 0 and 0 otherwise. Then the VCD of $\mathcal{G}$ satisfies:

$$VCD(\mathcal{G}) \in O(t\ l\ (log_2\ q + log_2\ D)\ +\ t\ log_2\ s) \tag{16}$$

where $t \in O(lq + qA(l^2 + nl))$ and $s \in O(3^l)$.

The first step in order to prove the above theorem is to construct a universal slp Γ_U over F_U an T_U such that it represents all the elements of the family $\{\Gamma\}_{n,l,q,D,A}$. The main idea in the construction process of Γ_U, is to introduce a set of parameters which take values in $\{0,1\}$, such that for each specialization of the set of parameters we obtain a particular slp in $\{\Gamma\}_{n,l,q,D,A}$. This is the purpose of the following lemma.

Lemma 5.1. *For any natural numbers n, l, q, D, A, let $\{\Gamma\}_{n,l,q,D,A}$ be the family of the slp's over F and T of length l, with $T = \{t_1, \ldots, t_n\}$, $F = \{f_1, \ldots, f_q, sign\}$; f_i rational functions of degree bounded by D and arity bounded by A. There exists a universal slp Γ_U over F_U an T_U such that represents all the elements of $\{\Gamma\}_{n,l,q,D,A}$ and verifies the following properties:*

(i) Γ_U *has length* $3l$
(ii) $|T_U| \in O(n + lq + qA(l^2 + nl))$
(iii) F_U *contains* $2l$ *rational functions whose degrees belong to* $O(qD)$ *and the arities belong to* $O(q(n+l)A)$

Proof. The first instruction u_1 of a slp Γ is constructed by picking a function f_i from F and then selecting a_i arguments for f_i from T. The following expression includes all the possibilities by giving values in $\{0,1\}$ to the variables x_i^1, z_j^1, and $y_{i,1}^k$; $i \in \{1,\ldots,q+1\}$, $j \in \{1,\ldots,n\}$, $k \in \{1,\ldots,a_i\}$:

$$u_1 := x_1^1 g_1^1 + x_2^1 g_2^1 + \cdots + x_q^1 g_q^1 + x_{q+1}^1 sign(z_1^1 t_1 + \cdots + z_n^1 t_n) \tag{17}$$

where g_i^1 are the following rational functions:

$$g_i^1 = f_i(y_{i,1}^1 t_1 + \cdots + y_{i,n}^1 t_n, \ldots, y_{i,1}^{a_i} t_1 + \cdots + y_{i,n}^{a_i} t_n) \tag{18}$$

For the instruction u_1 of Γ, in order to separate the *sign* function from the rational functions g_i^1, we add two new instructions in the universal *slp* Γ_U in the following form:

$U_1 := z_1^1 t_1 + \cdots + z_n^1 t_n$
$U_2 := sign(U_1)$
$U_3 := x_1^1 g_1^1 + x_2^1 g_2^1 + \cdots + x_q^1 g_q^1 + x_{q+1}^1 U_2$

Observe that the number N_1 of terminal variables involved in the three instructions verifies:

$$N_1 \leq n + n + (q+1) + qnA \tag{19}$$

Note also that the degree of the rational functions corresponding to the instructions U_1 and U_3 is bounded by $2qD+1$ and arities are bounded by $max\{2n, (q+2)+q2nA\} = (q+2)+q2nA$

Following this strategy in the construction of Γ_U, the instruction $u_k \in \Gamma$ produces the following three instructions included in Γ_U :

$U_{3k-2} := z_1^k t_1 + \cdots + z_n^k t_n + z_{n+1}^k U_3 + \cdots + z_{n+k-1}^k U_{3(k-1)}$
$U_{3k-1} := sign(U_{3k-2})$
$U_{3k} := x_1^k g_1^k + x_2^k g_2^k + \cdots + x_q^k g_q^k + x_{q+1}^k U_{3k-1}$

Where now, for each $i \in \{1,\ldots,q\}$, the rational function g_i^k is the following:

$$g_i^k = f_i(p_{i,1}^k, \ldots, p_{i,a_i}^k) \tag{20}$$

and for each $j \in \{1,\ldots,a_i\}$

$$p_{i,j}^k = y_{i,1}^j t_1 + \cdots + y_{i,n}^j t_n + y_{i,n+1}^j U_3 + \cdots + y_{i,n+k-1}^j U_{3(k-1)} \tag{21}$$

In this general case the number N_k of terminal variables verifies:

$$N_k \leq n + (n+k-1) + (q+1) + q(n+k-1)A \tag{22}$$

The degrees are bounded by $2qD+1$ and the arities are bounded by $max\{2(n+k-1), (q+2)+q2(n+k-1)A\} = (q+2)+q2(n+k-1)A$

As $\Gamma = \{u_1, \ldots, u_l\}$ is of length l, the universal *slp* Γ_U will be of length $3l$. Note that the terminal variables $t_1, \ldots, t_n$ appear in all instructions U_{3k-2}. Then, the number N of total variables of Γ_U verifies:

$$N \leq n + l(q+1) + \sum_{i=0}^{l-1}(n+i) + qA\sum_{i=0}^{l-1}(n+i) \tag{23}$$

Operating in equation 23 we obtain:

$$N \leq n + l(q+1) + (qA+1)(nl + \frac{l(l-1)}{2}) \tag{24}$$

Hence

$$N \in O(n + lq + qA(l^2 + nl)) \tag{25}$$

Finally, Γ_U has $2l$ rational functions, corresponding to the instructions U_{3k-2}, U_{3k}; $k \in \{1, \ldots, l\}$, which degrees are bounded by $2qD + 1$ and the arities are bounded by $(q+2)+q2(n+l-1)A$. So we obtain that the degrees belong to $O(qD)$ and the arities belong to $O(q(n+l)A)$ □

Remark 5.1. Note that each element of the family $\{\Gamma\}_{n,l,q,D,A}$ is obtained giving values from $\{0,1\}$ to the parameters **z, x, y**. As $\{\Gamma\}_{n,l,q,D,A} \subset SLP(F,T)$ with $T = \{t_1, \ldots, t_n\}$, it follows that Γ_U includes t parameters with $t \in O(lq+qA(l^2+nl))$

Once the universal slp Γ_U has been constructed, the second step is to prove the existence of a universal boolean formula $\Phi_{t,n}(x,w)$ equivalent to Γ_U in the sense described with the following lemma:

Lemma 5.2. *For any natural numbers n, l, q, D, A, as in lemma 5.1, there exists a universal boolean formula $\Phi_{t,n}(x,w)$ such that for any slp over F and T, in the family $\{\Gamma\}_{n,l,q,D,A}$ and for any $x \in \mathbb{R}^n$ the following holds: x is accepted by a slp within the family if and only if there is $w \in \mathbb{R}^t$ such that $\Phi_{t,n}(x,w)$ is satisfied. Moreover, the formula $\Phi_{t,n}(x,w)$ has the following properties:*

(i) $t \in O(lq + qA(l^2 + nl))$
(ii) $\Phi_{t,n}(x,w)$ *contains* s *polynomial equations where* $s \in O(3^l)$
(iii) The equations of $\Phi_{t,n}(x,w)$ *have degree at most* $O((qD)^l)$

Proof. Considering the universal *slp* Γ_U of lemma 5.1 and the above remark, is easy to see that the number t of parameters is in $O(lq + qA(l^2 + nl))$.

To analyze the number of equations and their degrees, observe that Γ_U has l sign instructions, corresponding to U_{3k-1}, $1 \leq k \leq l$. For each sign instruction, let h_k be the function of $(x_1, \ldots, x_n, w_1, \ldots, w_t)$ that U_{3k-1} receives as input. It easily follows by induction that h_k is a piecewise rational function of $(x_1, \ldots, x_n, w_1, \ldots, w_t)$ of formal degree bounded by $(2qD+1)^{k-1} + 1$

Now, for each sign assignment $\epsilon = (\epsilon_k) \in \{>, =, <\}^l$ let Φ_ϵ be the formula:

$$\Phi_\epsilon = \bigwedge_{1 \leq k \leq l} (h_k \epsilon_k 0) \tag{26}$$

Claim A For every $\epsilon \in \{>, =, <\}^l$ there are rational functions r_k of $(x_1, \ldots, x_n, w_1, \ldots, w_t)$ of degree bounded by $(2qD+1)^{k-1}+1$ such that the formula Φ_ϵ is equivalent to the formula

$$\bigwedge_{1 \leq k \leq l} (r_k \epsilon_k 0) \tag{27}$$

The proof of this claim is by finite induction on the number of conjunctions in equation 26: for the basic case $i = 1$, the instruction in Γ_U previous to U_2 is a rational function of degree 2. Assume now that $\Phi_\epsilon = \bigwedge_{1 \leq k \leq i-1} (h_k \epsilon_k 0)$ satisfies the required condition. In this case the result follows by noting that the role played by sign instructions previous to U_{3i-1} on inputs satisfying formula $\Phi_\epsilon = \bigwedge_{1 \leq k \leq i-1} (h_k \epsilon_k 0)$ is superfluous and U_{3i-1} is the sign of a rational function r_i of degree bounded by $(2qD+1)^{i-1}+1$.

In what follows formula in equation 27 will be also denoted by Φ_ϵ. Notice that the set of inputs $(x_1, \ldots, x_n, w_1, \ldots, w_t)$ accepted by Γ_U can be described by a disjunction of the formulae Φ_ϵ, $\epsilon \in \{>, =, <\}^l$. Then we have 3^l possibilities for ϵ and the involved rational functions have degree at most $(2qD+1)^{l-1}+1$ (i.e. they are in $O((qD)^l)$). Finally observe that we can also describe Φ_ϵ by $O(3^l)$ polynomials which degrees are in $O((qD)^l)$. □

Proof. (**of Main Theorem 5.2**) Just plug lemma 5.2 in Theorem 5.1 and it will result the equation 16. □

Finally, if we substitute in the equation 16, t and s by their respective values and after making the suitable simplifications, we obtain the following corollary:

Corollary 5.1. *Let $\mathcal{G} = G_{n,l,q,D,A}$ be as in theorem 5.2. Then the VCD of $\mathcal{G}$ is polynomial in the length l of the slp's and satisfies:*

$$VCD(\mathcal{G}) \in O(qAl^2\ (l+n)\ (log_2\ q + log_2\ D)) \tag{28}$$

6. 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.

In order to go towards a GP scheme based on straight line encoding of programs, we have perform a study of the VCD of the slp structure. Our conclusion is that the classification capacity of a family of slp's is polynomial of degree three in the length of the elements of the family.

As anticipating in the introduction, the experimental and theoretical results of this work, despite offering interesting outcomes themselves, must be looked upon as the first step towards a more general long-term goal. The final long term achievement that we would like to pursue is the construction of a GP approach based on straight line 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 VCD. [22,27] 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.[17]

7. 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. Cruz Enrique Borges is supported by FPU program and MTM2004-01167.

References

1. J. R. Koza, *Genetic Programming: On the Programming of Computers by Means of Natural Selection* (The MIT Press, Cambridge MA, 1992).
2. W. Banzhaf, P. Nording, R. Keller and F. Francone, *Genetic Programming - An Introduction: On the Automatic Evolution of Computer Programs and its Applications* (Morgan Kaufmann, Heilderberg - San Francisco, 1998).
3. N.L. Cramer, A Representation for the Adaptive Generation of Simple Sequential Programs, in *Proccedings of the First International Conference on Genetic Algoritms (IGA'85)*, ed. J. Grefenstette (Lawrence Erlbaum Associates, 1985), pp.183–187.
4. W. Banzhaf, Genetic Programming for Pedestrians, in *Proceedings of the Fifth International Conference on Genetic Algorithms (IGGA'93))*, ed. S. Forrest (Morgan Kaufmann, San Francisco CA, 1993), pp.638–.
5. P. Nordin, A Compiling Genetic Programming System that Directly Manipulates the Machine-Code, in *Advances in Genetic Programming*, ed. K.E. Kinnear (MIT Press, Cambridge MA, 1994), pp.311–331.
6. M. Brameier and W. Banzhaf, *Linear Genetic Programming* (Springer, 2007).
7. P. Bürguisser, M. Clausen and M. A. Shokrollahi, *Algebraic Complexity Theory* (Springer, 1997).

8. S. J. Berkowitz, On computing the determinant in small parallel time using a small number of processors, in *Information Processing Letters*. **18**(1984) 147–150.
9. K. Mulmuley, A parallel algorithm to compute the rank of a matrix over an arbitrary field, in *Combinatorica*. **7**(1995) 101–104.
10. N. Fitchas, A. galligo and 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** (Publications Mathematiques de l'Université Paris 7, 1987), pp.103–145.
11. J. Heintz., M. F. Roy and P. Solerno, Sur la complexité du principe de Tarski-Seidenberg, in *Bulletin de la Societé Mathematique de France*. **118**(1990) 101–126.
12. M. Giusti and J. Heintz, Algorithms- disons rapide- pour la décomposition dúna varieté algébrique en composants irreductibles et équidimensionelles, in *Papers for the symposium: Effective Methods in Algebraic Geometry (MEGA'90)*, ed. T. Mora and C. Traverso (Birkhäuser, 1990), pp.169–194.
13. M. Giusti and J. Heintz, La détermination des points isolés et la dimension dúne varité algebrique peut se faire en temps polynomial, in *Computational Algebraic Geometry and Commutative Algebra, Symposia Matematica XXXIV*, ed. D. Eisenbud and L. Robbiano, (Cambridge University Press, 1993), pp.216–256.
14. M. Giusti, J. Heintz, J. Morais, J. E. Morgentern and L. M. Pardo, Straight Line Programs in Geometric Elimination Theory, in *Journal of Pure and Applied Algebra*. **124**(1997) 121–146.
15. J. L. Montaña and L. M. Pardo, Lower bounds for arithmetic networks, in *Appl. Algebra Engrg. Comm. Comput.* Vol. 4 **1**(1993) 1–24.
16. A. Topchy and 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)*, ed. L.Spector et al. (Morgan Kaufmann, 2001), pp.155–162.
17. L. Vanneschi, G. Mauri, A. Valsecchi and S. Cagnoni, Heterogeneus Cooperative Coevolution: Strategies of Integration between GP and GA, in *Proc. of the 8th annual conference on Genetic and Evolutionary Computation (GECCO 2006)*, ed. M. Keijzer (ACM, NY, 2006), pp.361–368.
18. A. Eiben and M. Jelasity, A critical note on experimental research methodology in EC, in *Proc. of the Congress on Evolutionary Computation (CEC 2002)*, (IEEE Computer Society, 2002), pp.582–587.
19. T.Bäck, *Evolutionary Algorithms in Theory and Practice* (Oxford University Press, Oxford, 1996).
20. V. Vapnik and A. Chervonenkis, Ordered risk minimization, in *Automation and Remote Control*. **34**(1974) 1226–1235.
21. V. Vapnik and A. Chervonenkis, On the uniform convergence of relative frequencies of events to their probabilities, in *Theory of Probability and its Applications*. **16**(1971) 264–280.
22. V. Vapnik, *Statistical learning theory* (John Willey & Sons, 1998).
23. G. Lugosi, *A pattern classification and learning theory* pp.5–62 (Springer, 2002).
24. P. Goldberg and M. Jerrum, Bounding the Vapnik-Chervonenkis dimension of concept classes parametrized by real numbers, in *Machine Learning*. **18**(1995) 131–148.
25. J. L. Montaña, VCD bounds for some GP genotypes, in *Proc. on the 18th European Conference on Artificial Intelligence (ECAI 2008)*, ed. M. Ghallab (IOS Press, 2008), pp.167–171.
26. C.L. Alonso and J.L. Montaña, Finiteness properties of some families of GP-trees, in *12th Conference of the Spanish Association for Artificial Intelligence (CAEPIA 2007)*, ed. D. Borrajo, M. Castillo and J.M. Corchado (Springer, 2007), pp.190–199.

27. V. Cherkassky, X. Shao, F. Mulier and V. Vapnik, Model complexity control for regression using VC generalization bounds, in *IEEE Transactions on Neural Networks*. Vol. 10 **5**(1999) 1075–1089.