GramGen: A Genetic Programming System Based on Context Free Grammar

Arnaud Quirin, Jerzy Korczak

Abstract— **In this paper, a new genetic programming system, called GramGen, is described. The system combines context free grammar (CFG) with genetic programming and uses an extended operator set. The objective of the grammar is to limit the size of the search space by allowing the user to define constraints related to the structure or the simplicity of the discovered formulas. These constraints are taken into account by the use of specific genetic operators. Our algorithm have been validated on image processing applications containing voluminous, noisy, and sometimes, not well registered data. The experiments shown that the proposed system allows users to discover new formulas as well as improve the performance of existing ones.**

Index Terms— **genetic programming, CFG grammar, classifiers, genetic representation, data mining.**

I. Introduction

A genetic programming algorithm (GP) is a kind of evolutionary algorithm in which the genetic individuals correspond to functions or programs [11]. GP offers several interesting advantages in terms of knowledge representation and knowledge discovery. The rule representation by syntactic trees is easy to understand and facilitates knowledge validation by a human expert. On the other hand, an evolutionary algorithm provides an increased resistance of discovery with regards to noisy data and local minima. The ability to customize genetic operators allows a deeper interaction with the human expert or the domain knowledge (such as specific convergence criteria, fitness function able to deal with understandability measures for the trees such as size, depth, balancing).

GP has been used for a long time for image processing, especially for shape detection. The work of Daida [4] or Harris [9] can be quoted, in which an individual is a program being executed on a 3∗3 or 5∗5 window of an image to detect the contrast difference between two pixels. Interesting results have been obtained in edge detection, by using the specification language EASEA [2], [3] to create artificial-animal detectors.

GP has been also applied for the processing of satellite images and their classification. For instance, a solution to solve the *inverse PAR problem* (Photosynthesis Available Radiation) has been proposed in [5], [14] or [18]. This approach tries to discover a function representing several characteristics of the domain knowledge. Here, the number of available photons for photosynthesis in a marine environment is modelled, using the signal perceived by the sensors. With a population of 5000 individuals, the operators ($+$, $-$, $*$, $/$) and a maximum depth of 10 for the trees, the authors have obtained a result correlated at 81% with those of an algorithm employed by NASA [14]. As shown in the paper, the algorithm is very sensitive to over-fitting and noisy data. In fact, the application of the algorithm to noisy data, as is often the case in remote sensing, remains relatively difficult.

To illustrate our approach, the symbolic regression on remote sensing images are examined in the case of supervised learning. Regression of remote sensing images is performed by assigning pixels of a remote sensing image to real numbers. These numbers indicate the proportion of a given class (i.e. water, vegetation, building, etc.) in this pixel. The process of regression is complex because of voluminous, noisy data, and existance of contradictions between the observed and the ground truth data.

An interesting application of GP is the determination of indices such as the Normalized Difference Vegetation Index[1] or the Brightness Index[2] [7]. The indices are formulas converting the values of a pixel (a spectral vector) into a proportion of a given class (expressed as a percentage of vegetation or ground). Since the indices may contain many operators, and since the images are voluminous (sometimes about 1000×1000 pixels), the main difficulty is caused by the very large size of the search space. Therefore for remote sensing geographers the opportunity of automatic discovery of such formulas is very interesting not only because new indices may be elaborated, but also one may improve the performance of the existing ones.

In this paper, a grammar-based GP system, called **GramGen**, is proposed to limit the size of the search space and allow to define constraints related to the structure and complexity of the formulas. In the system, the constraints are taken into account by applying specific genetic operators to simplify generated formulas, and in consequence, make them more comprehensive.

To introduce the problem of knowledge representation in GP systems, let us define a few basic terms, notably:

- **Genotypic tree**. A tree corresponding to the content of the genetic chromosomes. This tree is related to the grammar and is used to facilitate the generation of new individuals.
- **Phenotypic tree**. A tree corresponding to the interpreta-

LSIIT/CNRS Laboratory, Boulevard Sebastien Brant, 67400 Illkirch, France. {quirin,jjk}@lsiit.u-strasbg.fr

[1]NDVI = $\frac{XS3-XS2}{XS3+XS2}$
[2]BI = $\sqrt{XS2^2+XS3^2}$

tion of the genotypic tree. This tree is derived from the genotypic tree and encodes the function given to the user.

- **Non terminal symbol**. The non terminal symbols are nodes of the genotypic trees and they are only useful during the derivation step, in which the production rules of the grammar are applied.
- **Terminal symbol** or **terminal operator**. The terminal symbols are nodes existing both in a genotypic tree and in a phenotypic tree. In this paper, this term does not indicate the value of a node, but the corresponding keyword. We will use these keywords in the case study section. For instance, *opMUL* refers to the multiplication, *opSIN* to the sinus and *opCST* to an instantiated constant.
- **Node operator** or **functions**. Node operators correspond to functions with an arity greater than 0, such as *opMUL*, *opSIN*, etc. They are only used in the leafs in the genotypic trees (and not in the phenotypic trees).
- **Leaf terminal**. Leaf terminals are leafs in the phenotypic tree and correspond to functions of arity 0, such as *opCST* (instantiated constants) or *opARG* (instantiated arguments).
- **Grammatical disjunction**. Right part of a production rule containing one or several conjunctions. This part replace the left part during a derivation.
- **Grammatical conjunction**. Part of a disjunction corresponding to a terminal or a non terminal symbol.

Each one of these terms will be detailed later.

In the next section the **GramGen** system is presented. In section III, the formalism of the grammar is described. Section IV introduces the terminal operators used in the algorithm, the *opPUSH* operator and the *opPOP* operator followed by the initialisation, the crossover and the mutation operators are described. Finally, in the last section, four cases are discussed illustrating the ability of our algorithm to produce comprehensible rules.

II. Main algorithm

The general principle of rule discovery by GP follows the principle of evolution-based algorithms [8], except that here, the rules introduced by the algorithm correspond to tree structures including node operators and leaf terminals.

GramGen uses this principle, and in addition, implements a set of constraints on the tree representation using a grammar initially defined by the user. An other point is that the individuals of the population are ordered by their *fitness* values as an efficient way to find similar individuals. This measure is simply used as an indicator of a premature convergence of the population and is shown to the user.

Algorithm A1 describes the main algorithm of **GramGen**.

GramGen applies a grammar, parameterized by the user before starting the algorithm to discover the rules. The genetic operators are strongly based on this grammar. Thus, during the initialisation step, the crossover step, or the mutation step, the constraints defined by the grammar are always true *at the output* of the operators. These constraints are defined in terms of grammatical constraints *(« the numerator of a ratio must not have multiplicative signs »)* and probability constraints *(« the multiplicative and addition signs must have the same probability of appearance »)*. In particular, we have proposed two ways to guarantee that the genetic operators produce only correct offsprings. The *passive method* checks each produced individual and deletes the trees which do not satisfy the grammatical constraints. The *active method* ensures that the operators produce directly correct individuals in terms

```
ALGORITHM A1
GRAMGEN : MAIN ALGORITHM

⇝ Result - $I$, the function (tree) solving the given problem
⇝ $Prop_\chi$ is the proportion of crossovers in the population
⇝ $Prop_\rho$ is the proportion of reproductions in the population
⇝ $Prop_\mu$ is the proportion of mutations in the population

Initialize the algorithm (gen=0, pop={})
Randomly create an initial population $pop$
repeat
  Begin a generation (indiv=0, $pop_{new}$={})
  while indiv/size(pop) < $Prop_\chi$ do
    Choose two individuals from $pop$ based on the crossover selection strategy
    Carry out the crossover (see the section IV)
    Insert the two offsprings into $pop_{new}$ (indiv=indiv+2)
  end while
  while indiv/size(pop) < $Prop_\rho$ do
    Choose an individual from $pop$ based on the reproduction selection strategy
    Copy this individual into $pop_{new}$ (indiv=indiv+1)
  end while
  while indiv/size(pop) < $Prop_\mu$ do
    Choose an individual from $pop_{new}$ based on the mutation selection strategy
    Choose randomly a type of mutation among the three available (see the section IV)
    Carry out the mutation
    Insert the new individual into $pop_{new}$ (indiv=indiv+1)
  end while
  Compute the new population (replacement) :
  pop=Recycle(pop,$pop_{new}$)
  Evaluate the fitness of each individual in the population
  Order the individuals of the population by their fitness
  Compute the values of the termination criteria (number of generations, best fitness exceeding a threshold, average fitness exceeding a threshold)
  gen=gen+1
until One of the termination criteria are satisfied
Return the best individual $I$ according to its fitness
```

Algorithm 1: *Algorithm* **GramGen**

of grammatical constraints and predefined probabilities. These last operators are more complex to describe, but as they do not lose time by generating and deleting useless individuals, thus they have been selected in the final implementation of **GramGen**.

The fitness function of **GramGen** measures the relative quality of an individual by comparing it to other individuals in the population. The fitness function is the ratio of the wealth of an individual and the sum of the wealth of all individuals of the whole population. In our case, this evaluation is computed for each individual using the following formula:

$$\begin{aligned} E(i) &= C_{fun}E_{fun}(i) + C_{size}E_{size}(i) \\ &\quad + C_{cst}E_{cst}(i) + C_{arg}E_{arg}(i) \end{aligned} \tag{1}$$

where $E_{fun}(i)$, $E_{size}(i)$, $E_{cst}(i)$ and $E_{arg}(i)$ are the evaluation of the individual i according to specific constraints, and C_{fun}, C_{size}, C_{cst} and C_{arg} are the weights set by the user according to his appreciation for each constraint. The values of each specific evaluation function are between 0 and 1.

$E_{fun}(i)$ computes the accuracy between the expected value of a training sample and the value obtained using the formula encoded in the individual i. The average of the sum of the error squared obtained on the training set is used as the result of $E_{fun}(i)$. It should be noted that in regression problems, the error corresponds to $e^{-|D|}$ where D is the difference between the obtained and the expected value. In classification problems, for an expected class, the error is 0 for a positive value and 1 for a negative value. And, for an unexpected class, the error is equal to 1 for a positive value and 0 for a negative value.

$E_{size}(i)$ computes the score of an individual according to an expected size, which corresponds to the number of nodes in the tree. This enables the user to specify a constraint on the size of the tree, which influences to the understandability of the obtained formula. In general, the shorter a formula is, the more understandable it will be, but the accuracy will also decrease. The user inputs an ideal number of nodes, X, and a maximal number of nodes, M. Let a given formula be i with s nodes. If s is between 0 and X, the score is computed proportionally between 0 and 1. If s is between X and M, the score is computed proportionally between 1 and 0. And, if s is above M, the score is set to 0.

$E_{cst}(i)$ and $E_{arg}(i)$ compute the score of an individual according to the number of constants and to the number of arguments used in the formulas. When a formula contains more constants, the accuracy is higher, but it will be more difficult and specific to the training set. Comparatively, when a formula uses more arguments (attributes of a training sample), this formula will be more complex and more difficult to understand. The user can input an upper number of constants and/or arguments, and the score is computed as before.

Consequently, the evaluation function used in **GramGen** is a tradeoff between the accuracy and the constraints defined by the user in terms of understandability and readability of the obtained formulas.

III. Formalism of the grammar

Regression systems based on GP usually amplify the size of the trees during the search of a reliable solution. For instance, a research work presented by Ross [15] shows a tree requiring about fifty nodes to be effective in a classification problem for a real-world application. The produced tree does not deliver a clear and intuitive explanation to users. Type-based genetic programming is often difficult to understand, especially for users without extensive GP experience who need to design grammars. However, in these kind of problems, a rigorous interpretation of the generated functions by a human expert is required for the validation of these functions. To simplify the trees, some constraints have often been proposed, as for instance those described by Montana [13]. In his project, the nodes are associated with data types and only the authorized grammatical constructions are admitted. However, in many regression or classification problems, the data have often the same format and this kind of type assignment is not required.

In our approach, the well-known Context Free Grammar (CFG) has been applied [6], [10]. This grammar is simple, general and can be put in the normal form to be fast and effective for the parsing operators.

Formally, a CFG can be defined as a quadruplet $G = (V_t, V_n, P, S)$, where:

- V_t is a finite set of terminals,
- V_n is a finite set of non-terminals,
- P is a finite set of production rules,
- S is an element of V_n and correspond to the *start symbol*.

The elements of P are represented by $V_n \rightarrow (V_t \cup V_n)^*$. Below, an example of a CFG grammar is given:

- $V_t = \{opADD, opARG, opCST\}$
- $V_n = \{S\}$
- $P = \{S \rightarrow opADD\ S\ S,\ S \rightarrow opCST,\ S \rightarrow opARG\}$
- $S = S$

Figure 1 illustrates this CFG. The operator *opADD* corresponds to the addition between two numbers and the operator *opARG* corresponds to an argument of the function (the real index of the argument is instantiated in the phenotypic tree). The operator *opCST* corresponds to a constant (the real value of the constant is instantiated in the phenotypic tree). Grammars like this are able to derive trees of variable sizes.

In **GramGen**, the *start symbol* is derived using the production rules until the obtained set contains only terminal symbols. The trace of this derivation can be reproduced in the form of a tree, called in the paper a *derivation tree* or a *genotypic tree*. Thus, the trace of the rule $A \rightarrow B$ is a tree with a root node A connected to a child node B. A rule $A \rightarrow BC$ is a tree with a root node A connected to two child nodes B and C. At a given time, a non terminal symbol has to be derived,

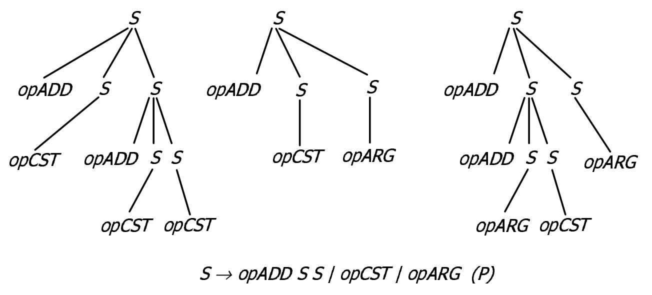


Fig. 1. Several genotypic trees produced using a derivation rule and an instance of a production rule.

in one or more symbols, using only one of the production rules. The symbol « | » is used to separate several production rules (disjunctions). The creation of the genetic individuals is carried out in two steps (Figure 2): the grammar is firstly used to define a genotypic tree A_G, then this tree is converted into a phenotypic tree A_P corresponding to the function that the algorithm is looking for. The tree A_P is obtained by replacing any genotypic subtree A'_G containing a root node X and $N+1$ edges by a phenotypic subtree A'_P where the root corresponds to the first edge of A'_G (corresponding to a function of arity N), and where the N edges correspond to the number of edges 2 to $N+1$ of the tree A'_G. The complete phenotypic tree is obtained by performing all the replacements but the grammar is not required during this process. The Polish notation is applied: the node pointed by the first edge of each node encodes the function and the following nodes its arguments.

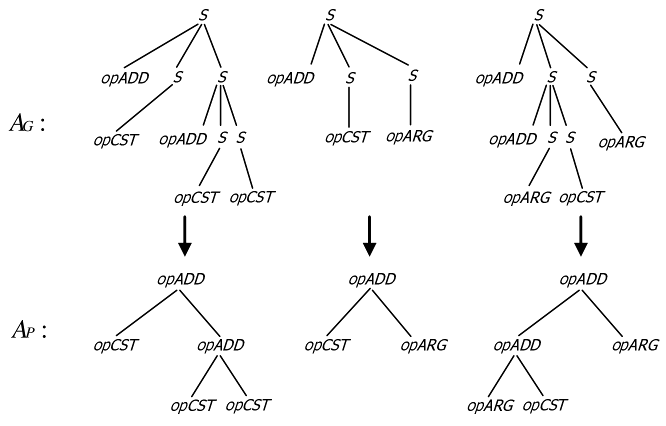


Fig. 2. Conversion of genotypic trees into phenotypic trees.

IV. The Gramgen operators

1) Terminal operators: In the next five sections the basic operators will be detailed: the terminal operators, the queue operators, the initialisation operators, the crossover and mutation operators. The terminal operators correspond to atomic units for the phenotypic trees produced by GP. Two kinds of terminals can be distinguished: the leaf terminals (arguments or constants), shown in Table I, and functions, shown in Table II. In the experiments all the terminals and the majority of the operators introduced in the two tables have been used, in particular the mathematical operators, the operators of arity 3 and the operators of arity N. In addition to the implementation of common functions, a few useful functions for regression problems have been proposed such as the functions of arity N with an unspecified number of parameters in the input, in particular *opSOMM* (sum of the arguments) or *opAVG* (average of the arguments).

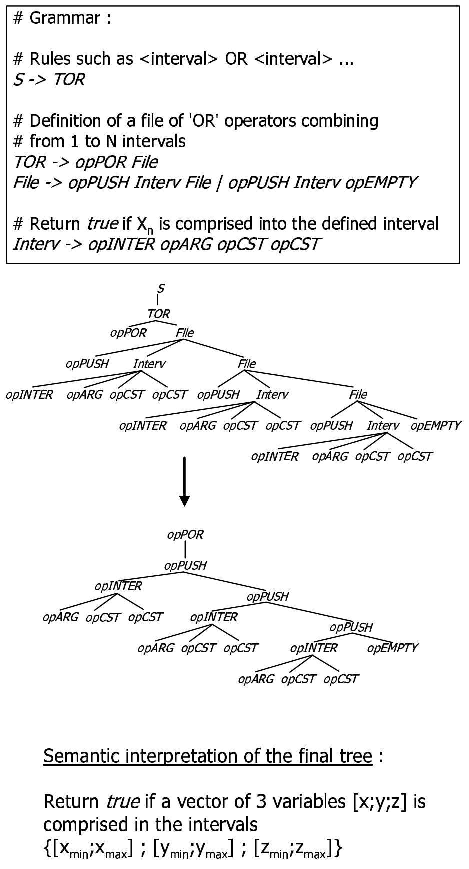
TABLE I
List of the leaf terminals available to the user for the representation of the trees

Symbol	Description
opFIXED	Constant (integer or real) directly specified in the grammar (for instance, 1.03E-06).
opRANGE	Range of integer or real constants (for instance, [4;7]).
opCST	Constant randomly instantiated during the creation of the tree or the node. Its value will not change until the next genetic mutation.
opARG	Argument of the function. Each sample of the training set is represented by a vector of a known size, so each argument is instantiated by one of the values of this vector during the creation of the node. This instantiation is then modified during a genetic mutation.
opTRUE, opFALSE, opPI, opE, opPINF ($+\infty$)*, opMINF* ($-\infty$)*, cstDIVIZERO* (represents the value of a division by zero), *cstOVERFLOW* (represents an overflow such as $log(0)$)	Miscellaneous constants, mainly used for the condition part of a test.

TABLE II
List of the operators available to the user for the nodes of the trees

Description	Symbol
Mathematical operators of arity 1	*opOPP, opINV, opCOS, opSIN, opTAN, opLOGE* (napierian logarithm), *opEXP, opSQRT, opABS, opCEIL, opFLOOR, opACOS, opASIN, opATAN, opCOSH, opSINH, opTANH, opACOSH, opASINH, opATANH, opLOG10, opSIGN, opFACT* ($x!$), *opSQ* (x^2), *opCUB* (x^3), *opLOG2, opP10* (10^x), *opCURT* ($\sqrt[3]{x}$), *opP2* (2^x)
Boolean operators of arity 1	*opNOT, opCOMP1* (complement to 1), *opCOMP2* (complement to 2), *opSHL* (left *shift*), *opSHR* (right *shift*), *opROTL* (left rotation), *opROTR* (right rotation)
Miscellaneous operators of arity 1	*opIP* (integer part), *opFP* (floating part), *opRND* (round to the closest integer), *opARGN* (select the argument N in the current function)
Operators of various arities for the management of a FIFO file allowing to deal with tables of various size (see Fig. 3)	*opEMPTY* (empty file), *opPOP* (returns the first element), *opPUSH* (piles up the first argument empile and returns the number of piled objects), *opPSUM* (sum of all the objects in the file), *opPAVG* (average), *opPMED* (median), *opPAND* (boolean AND), *opPOR* (boolean OR), *opPEQUI* (returns *true* if all the objects are identical (all positives or all equal to *null*), *false* in the other case
Mathematical operators of arity 2	*opADD, opSUB, opMUL, opDIV, opPOW, opINF* ($<$), *opSUP* ($>$), *opINFE* ($\leq$), *opSUPE* ($\geq$), *opEGAL, opDIFF, opPRCT* ($x * \frac{y}{100}$), *opPRCTA* ($x * (1 + \frac{y}{100})$), *opPRCTS* ($x * (1 - \frac{y}{100})$), *opCOMB* (C_y^x), *opPERM* (P_y^x), *opAPRX* ($\|x - y\| < 1E - 4$), *opMOD* (modulo), *opQUOT* (integer division), *opXRT* ($\sqrt[y]{x}$)
Boolean operators of arity 2	*opAND, opOR, opXOR, opXSHL* (x is left-*shifted* of y bits), *opXSHR* (x is right-*shifted* of y bits), *opXROTL* (left rotation of x of y bits), *opXROTR* (right rotation of x of y bits), *opIMPL* (boolean implication), *opEQUI* (boolean equivalent)
Operators of arity 3	*opITE* (if $x > 0$ then y else z), *opLERP* (linear interpolation between y and z: $x * (z - y) + y$), *opINTER* (inclusion in an interval: *true* if $x \in [y; z]$, *false* in the other case)
Operators of arity N. The behavior of these operators depends on the number of edges that are associated with.	*opSOMM* (sum of all the arguments of the operator), *opPROD* (product), *opAVG* (average), *opMED* (median), *opMIN, opMAX, opETYP* (standard-deviation), *opVAR* (variance), *opSQSOM* (sum of the squares), *opSQAVG* (average of the squares), *opMAND* (boolean AND), *opMOR* (boolean OR), *opMEQUI* (*true* if the equivalence of the arguments is verified), *opSELECT* (selection of the value of the argument N)

The user may select the terminals to use in the genetic programs, either implicitly in the form of probabilities in the generated trees, or directly by the definition of the production rules. If needed, the setting up of the appearance probability of a terminal symbol (or a non terminal symbol) can be defined by repeating the occurrence of this symbol several times in the production rule. For instance, the rule *S* → *opADD opADD opADD opSUB* defines the appearance probability of the addition operator as 75 %.



```
# Grammar :

# Rules such as <interval> OR <interval> ...
S -> TOR

# Definition of a file of 'OR' operators combining
# from 1 to N intervals
TOR -> opPOR File
File -> opPUSH Interv File | opPUSH Interv opEMPTY

# Return true if X_n is comprised into the defined interval
Interv -> opINTER opARG opCST opCST
```

S
TOR
opPOR File
opPUSH Interv File
opINTER opARG opCST opCST opPUSH Interv File
opINTER opARG opCST opCST opPUSH Interv opEMPTY
opINTER opARG opCST opCST
opPOR
opPUSH
opINTER opPUSH
opARG opCST opCST opINTER opPUSH
opARG opCST opCST opINTER opEMPTY
opARG opCST opCST

Semantic interpretation of the final tree :

Return *true* if a vector of 3 variables [x;y;z] is comprised in the intervals $\{[x_{min};x_{max}] ; [y_{min};y_{max}] ; [z_{min};z_{max}]\}$

Fig. 3. Grammar using the *opPUSH* operator.

2) Queue operators: To solve some problems, operators to facilitate the management of *First In, First Out* (FIFO) queues are needed. FIFO queues allow a function to build tables where its size is known only during their execution. So their size can dynamically increase or decrease depending on the input attributes of the symbolic expression. Two queue operators are designed: the *opPUSH* and *opPOP* operators are these special operators. As a tree is interpreted by depth-first search (the child nodes of a given node are computed before its parent), it is consequently possible for a node to carry out computations on a file encoded in one of its child nodes. The operators allowing this kind of computation are introduced in Table II. Figure 3 shows a grammar construct using such operators, a genotypic tree built using this grammar, the related phenotypic tree and its semantic interpretation.

If needed, some inter-type conversions may occur in the nodes during the assignment of a value. For instance, a strictly positive constant returned by a mathematical operator is converted into a boolean constant equal to *true*. A null or negative value is converted into *false*, and the values *true* and *false* are respectively converted into 1 and 0. Lastly, a parameter is integrated into each one of these operators to define if the operator is commutative or not. This parameter is applied during the structural comparison of two trees, and that is used as a diversity criterion in the termination operator.

3) Initialisation operator: In **GramGen**, the initialisation operator for genetic individuals has to select the grammatical rules as well as the order of their application regarding a given number of constraints. This operator is applied to create the first genetic population or new edges claimed by the mutation operator. If several possibilities arise, the choice of the production rule is done according to several criteria, based on the size of the trees and the desired depth. The two principal steps devoted to the construction of the new individuals are as follows:

- determination of the height $H(X)$ or the smallest number of terminals $T(X)$ that it is possible to generate in the best case for each symbol X,
- during the construction of a subtree or a complete tree, determination of the symbol to use according to a given random probability related to $H(X)$ or $T(X)$.

The first step is performed only once, at the time of the grammar parameterization. This step is described in Algorithm A2. Figure 4 shows a parametrized grammar. The parametrization concerns the height and the minimal number of symbols that can be obtained in a genotypic tree derived from a given symbol. The computing of the maximum values are not very interesting for most grammars because they are infinite. The main interest of this algorithm is that it converges even in the case of *full-recursive* grammar rules (for instance, $A \rightarrow A$). In this case, these rules are automatically ignored.

$S \rightarrow E$
$E \rightarrow OEE \mid V$
$O \rightarrow opADD \mid opMUL$
$V \rightarrow opARG \mid opCST \mid 5.34$

Symbol	H(Symbol)	T(Symbol)
S	3	1
E	2	1
O	1	1
V	1	1
opADD	0	1
opMUL	0	1
opARG	0	1
opCST	0	1
5.34	0	1

Fig. 4. Example of a grammar and its parametrization.

The second step is performed during the creation of the individuals or each time a genetic operator requires the creation of a subtree. This step is presented in algorithm A3. The parameters of the algorithm are the following: a grammar, a non-terminal symbol (either the *start symbol S*, or another) and the constraints defining tree size and tree height. The result of the algorithm is a complete genotypic tree or a subtree which can then be included in a larger tree, which will be converted into a phenotypic tree before the evaluation of the individual. The algorithm performs its computing in an exact constraint environment, that is, the size specified by the user in terms of number of nodes is always respected. For instance, if the user selects an even size N, and that grammar can only produce trees with odd sizes, then there is a probability of 0.5 that an individual of size $N-1$ is produced, and 0.5 for an individual of size $N+1$. If the user specifies a null or a negative value for the size, the produced tree will be as small as possible. If some variable sizes are required, the user has to choose a range of acceptable values, for instance, trees containing from 3 to 15 nodes. Then the algorithm randomly selects a value in this range and uses it as

ALGORITHM A2
INITIALISATION OF THE SYMBOLS

```
⇝ Height(X) - Function computing the minimal height of a symbol X
⇝ Comment - $H(X)$ is an attribute of the symbol X
⇝ Result - The height of the symbol X

let $h := \infty$
for each disjunction $D$ of the rule $X$ do
    let $hc := 0$
    for each conjunction $C$ of $D$ do
        $hc := Max(hc, H(C))$
    end for
    $h := Min(h, hc)$
end for
Height(X) := $h + 1$
```

```
⇝ Size(X) - Function computing the minimal size of a tree generated by the derivation of a symbol X
⇝ Comment - $T(X)$ is an attribute of the symbol X
⇝ Result - The size of the symbol X
let $s = \infty$
for each disjunction $D$ of the rule $X$ do
    let $sc = 0$
    for each conjunction $C$ of $D$ do
        $sc = sc + T(C)$
    end for
    $s = Min(s, sc)$
end for
Size(X) := $s$
```

```
⇝ Parameter - The list of the symbols $L = V_t \cup V_n$ of a grammar $G$
⇝ Results - A parameterized grammar: the height and the minimal size of each symbols of $L$
for each symbol $X$ of $L$ do
    if $X$ is terminal then
        $H(X) = 0$
        $T(X) = 1$
    else
        $H(X) = \infty$
        $T(X) = \infty$
    end if
end for
repeat
    for each non terminal symbol $X$ of $L$ do
        $H(X)$ := Height(X)
        $T(X)$ := Size(X)
    end for
until the attributes of the symbols in $L$ have converged
Returns the parameterized grammar
```

Algorithm 2: *Function InitSymbols*

ALGORITHM A3
ALGORITHM FOR THE CREATION OF GENOTYPIC TREES FROM A GRAMMAR

```
⇝ Parameters - A parameterized grammar G, a non terminal symbol $A$, the requested size or height $f_{ask}$ for the generated tree
⇝ Result - The created individual
⇝ $f(X)$ is the criterion to optimize. Either $H(X)$ (the height of the subtree generated by the symbol $X$), or $T(X)$, the size of this subtree
⇝ Choice(L) is a function which selects in a uniformly random way an element of the set L
⇝ $\psi(n)$ is a function that gives the selection probability of a symbol if the subtree generated by the derivation of this symbol adds in the final tree more than $n$ symbols compared to the size expected by the user

let $R$ a tree with a root node $A$
while $R$ contains a leaf which is a non terminal symbol do
    let $L$ the list of the leafs in the tree $R$
    let $f_{min}$ := 0
    for each symbol $X$ of $L$ do
        $f_{min}$ := $f_{min} + f(X)$
    end for
    let $L_{nt}$ the list of the non terminal symbols of $L$
    let $T$ := Choice($L_{nt}$)
    for each disjunction $D_i$ of the right part of the rule $T$ do
        let $f_{add}$ := $f(D_i)$
        let $f_{sub}$ := $f(T)$
        $D_{i,suppl}$ := $f_{min} + f_{add} - f_{sub} - f_{ask}$
        $D_{i,proba}$ := $\psi(D_{i,suppl})$
    end for
    Choose a disjunction $D \in D_1, \ldots, D_n$ of the rule $T$ using the selection probabilities $D_{proba}$
    let $R_{ins}$ a tree with a root node $T$ and where each edges is one of the conjunctions of $J$
    Replace in the tree $R$ the symbol $T$ by the subtree $R_{ins}$
end while
Returns R, a tree with a root node $A$ where the leafs are terminal symbols
```

Algorithm 3: *Function TreeCreation*

parameter. The algorithm is called as many times as required to constitute a complete population. Thus, it is possible to obtain a population including trees where sizes can be specified by various probability functions: uniform, linear, Gaussian, etc. In our case, a linear function is used.

The determination of the non terminal symbol to derive, when the current derivation tree contains several symbols (known the derivation style), is not performed from the left (*leftmost derivation*), nor from the right (*rightmost derivation*). In fact, the best results have been obtained each time by randomly choosing the non terminal symbol in the tree currently in derivation. Moreover, that guarantees some diversity in the pool, even if the grammar is badly written.

When a disjunction needs to be derivated, a choice has to be made to select the best symbol B. This choice is related to the number of symbols needed to complete the current tree and the estimation of the number of symbols that can be generated by a derivation of B. Given a grammar G, a non terminal symbol B and a criterion F (F could be the expected size or height of a subtree generated by B), the algorithm uses a function $\psi(n)$ returning the selection probability of B if the subtree generated by the derivation of B adds in the final tree more than n symbols compared to the size expected by the user. It is clear that this probability should be low for high values of n. Several probabilistic functions for $\psi(n)$ (see algorithm A3) have been selected, among them:

$$\psi_1(n) = \frac{1}{a + |n|} \tag{2}$$

$$\psi_2(n) = e^{-\frac{(a*n^2)}{b}} \tag{3}$$

where a and b are constants.

The function $\psi_2(n)$ was found to have good selection qualities using the constant values of $a = 2$ and $b = 5.2$.

4) Crossover operator: The proposed operator for crossing over two genetic trees is based on the following principles:

- the generated individuals must remain coherent at the output of the operator. In particular, the grammar rules must always be verified as well as the arities of the nodes (a terminal operator should never change its arity),
- the selection probability of each node must be identical. Even if the genetic recombinations actually implement a very complex chromosomal system, it is preferable to keep the same probabilities of mixing, as in the case of a more simpler representation. For instance, a root node is not selected more frequently than a leaf node, so that the modifications are smoother,
- the algorithm must bring a guarantee that the resulting offsprings are different from the parents.

In conclusion, only subtrees coming from the same grammatical symbol are safe to be exchanged. The crossover operator, Algorithm A4, only deals with the genotypic description of the trees. The use of the genotypic trees guarantees the three points mentioned above. For instance, two subtrees, even deriving from the same terminal operator, will not be exchanged if they are not in the same grammatical *context*, i.e. if they derive from two distinct grammar rules. The resulting trees are converted into phenotypic trees and inserted in the new population.

Two remarks can be stated concerning Algorithm A4. First, the only case in which the offsprings would be identical to the parents is the case of a filiform genotypic tree. That corresponds to a grammar containing only one terminal symbol, consequently the corresponding phenotypic tree consists of only one symbol. In this case, the crossover cannot do better than exchange this symbol with one of the corresponding symbols of the other parent respecting the grammar. Second, the algorithm uses a parameter constraining the size of the generated trees. After the crossover, it is possible that this criterion is not respected any more. So, this criterion is verified *a posteriori* in the evaluation function.

```
ALGORITHM A4
CROSSOVER ALGORITHM IN GRAMGEN

⇝ Parameters - $A_1$ et $A_2$ are the two genotypic trees to cross
⇝ Results - $A_1'$ et $A_2'$ are the two resulting genotypic trees
⇝ $S(A)$ is a function which returns the list of the left-defined symbols of the grammar G defined in the tree A
⇝ Deriv(A,X) is a function which returns the list of the nodes in the tree A containing the symbol X
⇝ Choice(L) is a function which choose in a uniformly random way an element of the set L
⇝ GetChild(A,n) is a function which returns the child number n of the node A
⇝ X is a symbol of the grammar
⇝ $n_1$ and $n_2$ are nodes from genotypic trees

for each tree $A \in \{A_1, A_2\}$ do
   while CountChild(A) = 1 do
      A := GetChild(A, 1)
   end while
end for
let $L := S(A_1) \cap S(A_2)$
let $X :=$ Choice$(L)$
let $N_1 :=$ Deriv$(A_1, X)$
let $N_2 :=$ Deriv$(A_2, X)$
let $n_1 :=$ Choice$(N_1)$
let $n_2 :=$ Choice$(N_2)$
- Exchange $n_1$ and $n_2$ in the trees $A_1$ and $A_2$
- Returns the resulting trees $A_1'$ and $A_2'$
```

Algorithm 4: *Function Crossover*

The crossover parameters set by the user (besides the grammar) are the following:

- the percentage Q of individuals to be crossed. Most of the literature [8], [16] considers that 80% is an acceptable value, but we obtained good results with a comprised value between 70% and 95%,
- the crossover selection type (roulette wheel, tournament, etc).

5) Mutation operator: The mutation operator contains two significant characteristics; it imposes fewer parametrization by the user and it preserves almost all the material from the parent. This has led us to define three different and complementary sub-operators applied in a uniformly random way. Each one of these operators takes as input a genotypic tree and returns the modified tree. The conversion into a phenotypic tree is necessary before the insertion into the new population for the calculation of the evaluation function. In all cases, the constraints (which have been explained earlier) for the crossover operator have to be also respected (coherence, probability of selection, production of new offsprings).

The three sub-operators are as follows:

- Mutation of a node: performed by removing one of the nodes in the tree and replacing it by an equivalent one generated by the grammar.
- Mutation of a terminal: performed by changing one of the values of a numerical terminal (a constant or an argument).
- Mutation by self-crossing: performed by crossing the tree with itself. This sub-operator is complementary to the other ones because it can, for instance, reverse the numerator and the denominator of a ratio, which is not possible with the other sub-operators.

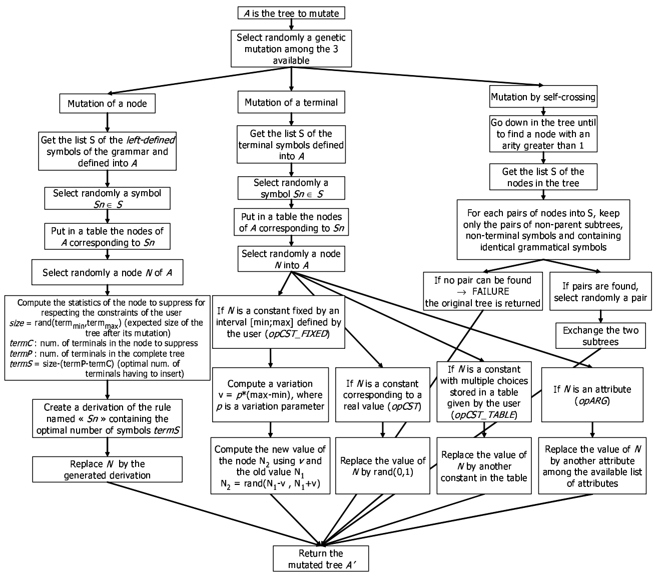

A is the tree to mutate
Select randomly a genetic mutation among the 3 available
Mutation of a node
Mutation of a terminal
Mutation by self-crossing
Get the list S of the left-defined symbols of the grammar and defined into A
Select randomly a symbol Sn ∈ S
Put in a table the nodes of A corresponding to Sn
Select randomly a node N of A
Compute the statistics of the node to suppress for respecting the constraints of the user
size = rand(term$_{min}$,term$_{max}$) (expected size of the tree after its mutation)
termC: num. of terminals in the node to suppress
termP: num. of terminals in the complete tree
termS = size-(termP-termC) (optimal num. of terminals having to insert)
Create a derivation of the rule named « Sn » containing the optimal number of symbols termS
Replace N by the generated derivation
Get the list S of the terminal symbols defined into A
Select randomly a symbol Sn ∈ S
Put in a table the nodes of A corresponding to Sn
Select randomly a node N into A
If N is a constant fixed by an interval [min;max] defined by the user (opCST_FIXED)
Compute a variation v = p*(max-min), where p is a variation parameter
Compute the new value of the node N$_2$ using v and the old value N$_1$
N$_2$ = rand(N$_1$-v , N$_1$+v)
If N is a constant corresponding to a real value (opCST)
Replace the value of N by rand(0,1)
If N is a constant with multiple choices stored in a table given by the user (opCST_TABLE)
Replace the value of N by another constant in the table
If N is an attribute (opARG)
Replace the value of N by another attribute among the available list of attributes
Go down in the tree until to find a node with an arity greater than 1
Get the list S of the nodes in the tree
For each pairs of nodes into S, keep only the pairs of non-parent subtrees, non-terminal symbols and containing identical grammatical symbols
If no pair can be found → FAILURE the original tree is returned
If pairs are found, select randomly a pair
Exchange the two subtrees
Return the mutated tree A′

Fig. 5. Mutation operators in **GramGen**. Note that rand(a,b) is a function returning a random value between a and b.

For the mutation operator (Figure 5), the user has to define the following parameters:

- the percentage Q of individuals to be mutated (the considered percentages have been set between 5% and 45%,
- the mutation type,
- and in the case of the terminal mutation operator, the new value is selected in an interval $[x-v; x+v]$ where x is the previous value and v is a parameter of variation related to the size of the authorized range for this value (so, this operator is data-scale independent).

To summarize, the described operators guarantee that the numerical constraints imposed by the user are respected (for instance, the size of a tree) as well as the user defined grammar. Some specific checks can be implemented to deal with the case of badly conceived grammars. In the case of the last mutation sub-operator (mutation by self-crossing), the exchange of a node with one of the progenitors of this node should never be permitted. However, this can occur with a grammar generating filiform trees. In this situation, the case is detected and the non-modified tree is returned.

V. Case study

A. First experiment (COSLOG)

The goal of the first experiment is to test our algorithm on a very simple regression problem, called the COSLOG problem. The algorithm has to interpolate a set of points given by the function $f(x) = cos(log(x))$. This function is very complex compared to the terminal operators available for this algorithm. The set of available terminals contains ($+$, $-$, $*$, $/$, $|x|$). The chosen grammar allows generation of trees with any number of nodes, but the *ideal number of nodes* parameter has been set to 10. In this case, trees of any sizes can be generated, but the fitness function will penalize trees that are too small or too large. This will give clues about the ability of the algorithm to model complex real world functions with only a limited set of operators. The experiment has been made more complex by limiting the number of available points: the training set contained only 20 pairs in the form $(x, f(x))$.

The learning has been carried out using the set of parameters shown in Table III.

TABLE III
Used parameters for the COSLOG problem.

Parameter	Value
Sampling	20 instances
Population size	|P| = 50 individuals
Size of the trees	5 to 10 nodes
Termination criterion	50 generations
Operator set	{+, -, *, /, absolute value (*opABS*)}
Terminals	{constants (*opCST*), variable X (*opARG*)}
P_{mut}	0.40
P_{cross}	0.70
Selection operator	Direct ranking
Replacing operator	Direct ranking
Number of offsprings per generation	|P|
Elitism	1% (high)
Duration	2 min (2.5 GHz CPU) for 20 instances

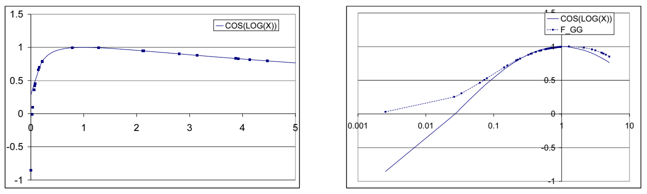


Fig. 6. The COSLOG problem. Left part: function $f(x) = cos(log(x))$ and the training set. Right part: the function $f(x)$ (dashed line) and the function found by the algorithm (dotted line).

Figure 6 shows the COSLOG problem. The left part shows the function $f(x)$ for a training set of 20 points. More points have been sampled in the interval $(0; 0.2]$ because of the shape of the function. The right part shows the function $f(x)$

(dashed line) and the function found by **GramGen** (dotted line). A horizontal logarithmic scale has been used in the second figure to facilitate visualization.

Equation 4 represents the formula which was finally generated and Equation 5 is its simplified form.

$$\begin{aligned} f_{GG}(x) &= \frac{(0.316 + 0.833) * |x|}{0.833 - 0.741 + x} \\ &\quad -(0.316 * 0.316 * 0.741 * x * 0.741) \end{aligned} \tag{4}$$

$$f_{GG}(x) = \frac{1.149 * x}{0.091 + x} - 0.055 * x \tag{5}$$

Discussion: A high correlation has been observed between the function interpolated and the obtained formula. The genetic evaluation of the individual corresponding to $f_{GG}(x)$ is 0.946 and the correlation coefficient between $f(x)$ and $f_{GG}(x)$ is 0.926. According to Figure 6, this correlation is also relatively high for small ($\sim$ 0.1) and high ($\sim$ 5) values of x, despite the relative simplicity of the obtained formula. Consequently, **GramGen** achieves a good performance for the small data set and for the complex function shown in this experiment.

B. The Multispectral Vegetation Index

This case study concerns the discovery of formulas indicating vegetation pixels on remote sensing image. In these problems, the learned classes have to be modeled in the form of functions assigning a real value to a pixel. In general, the assigned value indicates the proportion of a given ground cover in a given pixel. Thus, the problem can be viewed as a symbolic regression problem. The adequate representations for these rules are trees in which the operator located in the top node is able to produce continuous values. In **GramGen**, this value is always produced by a mathematical operator located at the top of the tree.

The accuracy of the obtained function is considered the most important parameter. Accordingly, the following values have been used: $C_{fun} = 0.95$, $C_{size} = 0.05$, $C_{cst} = 0$ and $C_{arg} = 0$. The ideal and the maximal number of nodes of a tree is between 10 and 30 but this size depend on each problem.

Two indices concerning the composition of a *mixel* in vegetation have been analyzed. The first index, well-known in remote sensing, is the *NDVI* index (*Normalized Difference Vegetation Index*):

$$\mathit{NDVI} = \frac{C_1 - C_2}{C_1 + C_2} \tag{6}$$

where C_1 and C_2 are two variables depending on the remote sensor.

This index allows to estimate the composition in terms of vegetation of the spectral samples. This equation uses the fact that the solar radiation is more strongly reflected by a vegetation in full growth in the near infrared than in shorter wavelengths located in the visible part of the spectrum [12]. Generally, with the *VEGETATION* sensor of the satellite SPOT, XS3 is used for the variable C_1 and XS2 for the variable C_2. With the hyperspectral images, the channels have to be chosen on a case-by-case basis, according to the wavelengths. In our experiment, the index was analyzed on multispectral data (SPOT satellite, 1100x900 pixel instances, 3 bands) and hyperspectral data (CASI airborne sensor, 329 instances, 288 bands).

The second index, called I_{Lim}, has not been yet expressed, to our knowledge, using a standard formulation. Its purpose is to determine the proportion in the ground of a plant known as *Limonium Narbonense*, more commonly called *Lavender of Sea*, usually found in the marshes and the clay soils. This vegetation class is interesting within the framework of studies for safeguarding the coastal environment. For this study, the MIVIS images (airborne sensor, 397x171 instances, 20 bands) were used. More information about the CASI and the MIVIS sensors can be found in [17]. In this experiment, a generic grammar has been proposed to produce any tree containing the operators specified in Table IV. The goal of the algorithm is to discover the formula of the index according to a large set of operators. It is up to the algorithm to respect this constraint (or not) according to the evaluation of the generated trees. The algorithm has been also tuned to find the set of parameters bringing the best results, shown in the Table IV. The division operator, just like in the next studies, performs a protected division. The terminal operator set has been selected because it corresponds to the operators frequently applied by geographers.

TABLE IV
USED PARAMETERS FOR SPOT.

Parameter	Value
Sampling	100 instances (< 1% of the data)
Population size	|P| = 200 individuals
Size of the trees	5 to 10 nodes
Termination criterion	200 generations
Operator set	{+, -, *, /, sum of arity N (*opSOMM*), absolute value (*opABS*)}
Terminals	{constants (*opCST*), spectral channels (*opARG*)}
P_{mut}	0.15
P_{cross}	0.70
Selection operator	Direct ranking
Replacing operator	Direct ranking
Number of offsprings per generation	|P|
Elitism	1% (high)
Duration	7 min (2.5 GHz CPU) for 5000 instances

The discovered formula is presented in Equation 7.

$$f_{SPOT} = \frac{1.065 + x_3 - |1.065 + x_2|}{|x_3 + x_2|} \tag{7}$$

It is easy to notice that the formula can be reduced to the

NDVI index:

$$f_{SPOT} = \frac{x_3 - x_2}{x_3 + x_2} \tag{8}$$

Discussion: The original index has been directly discovered by the algorithm from the image without using preliminary knowledge. The used grammar is generic and uses a few parameters and typical values of crossover and mutation probabilities. The discovered formula, imitating exactly the target index, is the one that obtained the maximal performance both in terms of prediction of the values of the test data but also in terms of the number of nodes desired by the user.

C. Hyperspectral NDVI *Index*

The same algorithm as above has been applied on images captured by the CASI satellite which has a higher number of sensors, where several of them were highly correlated. The algorithm has to correctly choose the sensors C_1 and C_2 in Equation 6. The used parameters are presented in Table V.

TABLE V
USED PARAMETERS FOR CASI.

Parameter	**Value**
Sampling	9000 samples (1% of the data)
Population size	|P| = 200 individuals
Size of the trees	from 10 to 20 nodes
Termination criterion	700 generations
Operator set	{+, -, *, /, sum of arity N (*opSOMM*), absolute value (*opABS*)}
Terminals	{constants (*opCST*), spectral channels (*opARG*)}
p_{mut}	0.15
p_{cross}	0.70
Selection operator	Direct ranking
Replacing operator	Direct ranking
Number of offsprings per generation	|P|
Elitism	1% (high)
Duration	40 min (2.5 GHz CPU) for 160 instances

The increasing number of spectral channels led us to increase the number of generations and the desired size of the trees. The discovered formula is as follows:

$$f_{CASI} = 2.253 \cdot \frac{2.253 \cdot x_{11} - 2.253 \cdot x_8 - x_4}{6.498 \cdot x_{12} + 6.498 \cdot x_8 - 6.289} \tag{9}$$

Discussion: It is difficult to evaluate the value of this formula by direct comparison with *NDVI*. Therefore the results of the two indices have been compared using the correlation ratio. Figure 7 shows an extract of a CASI image computed by these two indices.

Note that the two images are very similar: the correlation between the *NDVI* index and f_{CASI} is very high (C = 0.986). However, this formula contains four different attributes rather than two. The index used to generate the data was as follows:

$$NDVI_{CASI} = \frac{x_{11} - x_3}{x_{11} + x_3} \tag{10}$$

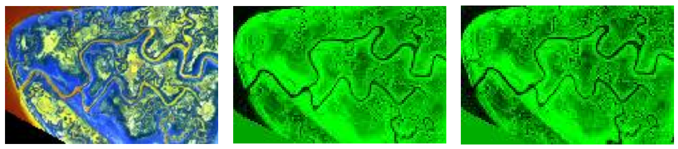


Fig. 7. On the left: a part of a CASI image. In the middle: percentage of vegetation according to *NDVI*. On the right: percentage of vegetation according to f_{CASI}.

In the formulas, a confusion between x_{11} and x_{12} on one hand, and x_3 and x_8 on the other hand can be explained by the correlations between these sensors are very strong (0.868 and 0.997 respectively). Summing up, the formulas determined for SPOT and CASI are rather short and reliable to ground truthing. The formulas discovered by the genetic programming algorithm, even if they are not exact, can always be substituted with the *NDVI* index. For example, in the case of searching sensors other than the standard ones to obtain the same result (i.e. in order to avoid noisy channels).

D. Multispectral I_{Lim} index

In this case study a problem of symbolic regression of mixed pixels has been addressed. In general, the values of pixels on remote sensing image result of spectral signatures of diffent ground classes. In the project TIDE [17], the images of Venise lagoon were classified taking into consideration the vegatation classes. The composition values in *Limonium* of the samples (*mixels*) were much more varied in the MIVIS data than in the CASI or the SPOT data. Figure 8 presents the statistical distribution of the samples according to their composition in *Limonium* or of another class for the counter-examples. In the experiment, 975 samples have been used, including 50% samples containing a dominant proportion of *Limonium* and 60% containing at least 5% of *Limonium*. The counter-examples are mainly water and ground pixels but have been selected from all the available classes. The MIVIS image includes 20 spectral channels and each *mixel* has a resolution of 2.6 m^2. 50% of the instances were used for the training set, and the remaining for the testing set.

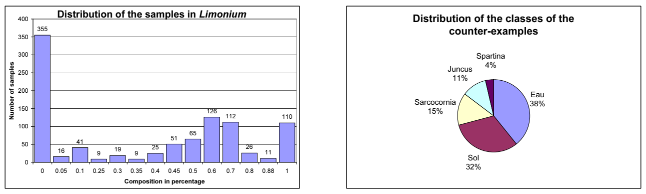


Fig. 8. Data set for the I_{Lim} index. Left: distribution of the samples according to the quantity of *Limonium* in the samples. Right: distribution of the classes of the counter-examples.

The training set was suitable to validate the I_{Lim} index with various proportions of *Limonium* in the samples. It should be noted that such a database is rather rare since each sample had to be hand-validated for agreement with the spectrometric data on the ground and the visual analysis of the pictures acquired during the ground-truthing.

It should be noted that the discovery of such an index was not trivial. The use of a grammar construct was very useful to reduce the search space. Several experiments with various grammars were carried out using the parameters presented in Table VI. The grammars were composed of operators and spectral sensors but not constants.

TABLE VI
USED PARAMETERS FOR MIVIS.

Parameter	Value
Sampling	100 samples (10% of the data)
Population size	|P| = 250 individuals
Size of the trees	from 10 to 30 nodes
Termination criterion	500 generations
Operator set	{+, -, *, /, absolute value (*opABS*), *opINV*, *opSQ*, *opINF*, *opSUP*, *If* ... *Then* ... *Else*, *opINTER*, *opRND*, *opEMPTY*, *opPOP*, *opPUSH*, *opSOMM*}
Terminals	{constants (*opCST*), spectral channels (*opARG*)}
P_{mut}	0.15
P_{cross}	0.60
Selection operator	Direct ranking
Replacing operator	Direct ranking
Number of offsprings per generation	|P|
Elitism	1% (high)
Duration	10 to 50 min (2.5 GHz CPU) for 500 instances

The experiments have been conducted with four grammars and generated trees of various sizes. The first three grammars have produced trees having depths located respectively in the ranges $[2;3]$, $[2;4]$ and $[3;5]$. In fact, the use of constants often tends to let the algorithm over-fit the data, and more importantly, the discovered formulas are less independent than those without constants.

To illustrate our approach, the fourth grammar will be examinated, in which the algorithm can generate random constants. The discription of the grammar is as follows:

```
# Start symbol
S → EQ24

# Definition of a tree with a depth of 2 to 4, with 3 to 15 nodes
EQ24 → EQ23 | OP EQ23 EQ23

# Definition of a tree with a depth of 2 to 3, with 3 to 7 nodes
EQ23 → OP EQ12 EQ12

# Definition of a simple equation (depth: 1 to 2, nodes: 1 to 3)
EQ12 → Arg | OP Arg Arg

# Definition of the operators and the attributes
Arg → opARG | opCST
OP → opADD | opSUB | opMUL | opDIV | opABS
```

The symbols on the left in the production rules are the non terminal symbols of the genotypic trees, and those on the right are terminal symbols or symbols used for the derivation. The terminal symbols corresponding to functions (node operators) are always followed by the symbols that represent their arguments. To simplify the notation, without loss of generality, there can be only one node operator by grammatical disjunction. For instance, the non ambiguous term « *opMUL opARG opCST* » indicates a multiplication between one of a sample attributes (*opARG*) and an instantiated constant (*opCST*).

The evaluation of the discovered formulas are shown in Table VII, with the performance evaluation on the training set and the evaluation of the tree size, their accuracy on the testing set (computed by the correlation compared to the expected compositions) and the number of generations required to produce the final formulas.

TABLE VII
FORMULAS FOR THE I_{Lim} INDEX, WITH SOME CHARACTERISTIC PARAMETERS.

	Formula	Evaluation	Correlation	Generation
1	$\frac{b_{19}-b_{14}}{b_{17}}$	0.859	0.802	14
2	$\frac{b_{15}\cdot(b_{19}-b_{16})}{b_1\cdot b_6}$	0.904	0.876	224
3	$\frac{b_{19}\cdot b_{15}}{(b_2-b_9)+(b_8\cdot b_{16})}-\frac{(b_2-b_9)+(b_8\cdot b_{16})}{b_{19}+(b_8\cdot b_{16})}$	0.668	0.814	463
4	$\frac{b_{17}-b_{15}}{b_4\cdot 0.985}-\frac{0.938}{b_{17}-b_{18}}$	0.694	0.858	94

Discussion: One can notice than the sensors b_{16} (740 nm) and b_{19} (800 nm) are important for I_{Lim} (they appear eight times in the formulas). In spite of their simplicity, a very high correlation is observed for the first two formulas. The first formula has been obtained very quickly, in spite of the low number of training samples. **GramGen** was thus able to discover short formula that were relatively expressive for the expert (because they were similar to the *NDVI* index) in a relatively short time. The learning time was short compared to the size of the search space. For instance, for a tree containing five arguments on the leaf level and 20 attributes, as in the formula 2 in Table VII, it was necessary to evaluate 20^5 (more than 3 million) combinations, without counting the operators. Note that only basic operators have been tested and rather simple grammars. The use of operators specific to the field of study would undoubtedly improve the results.

VI. CONCLUSION

In this paper, a new approach to discover rules able to solve symbolic regression problems by grammar-based GP is proposed. In general, the trees are a powerful representation, but are sometimes difficult to understand. This representation requires the redefinition of the genetic operators in a such way that the generated individuals are coherent, according to the grammar. In the paper, specific attention has been given to the legibility and the complexity of the trees by integrating thresholds defined by the user. The thresholds control the number of nodes, the height of the trees and the expected accuracy using a small number of parameters.

Various tests have been carried out with this new approach using many grammars. Generally, in terms of accuracy on the testing set, the obtained results have been acceptable, however the most comprehensible trees have been obtained with precise grammars. Concerning the comprehensibility, the question of knowing why a tree is more comprehensible only because it is generated by a grammar written by an expert remains an interesting research perspective. In many fields, experts are still accustomed to precise schemata, and it is advisable to respect this predisposition.

In spite of the complexity of the mutation operator, the bulk of the computating is led in an automatic way to use the grammar which discharges the user from parameter setting. Nevertheless, this operator can be improved. For instance, the principle of the self-adapting mutation [1] has not been tested yet within the framework of **GramGen**.

Real-world problems such as remote sensing image regression impose a large search space upon **GramGen** which it must be capable of searching efficiently. The presence of constants involves a considerable increase of the size of this search space. Techniques in which some constants values are freezed and locally optimized have been described in the literature, but they have not been yet implemented in **GramGen**. During the experiments, the amount of use of the *opCST* operator was limited, which restrains the effect of over-fitting and returns formulas which are slightly more adapted to the new data. Another interesting point concerns the anticipated algebraic simplification of the obtained formulas during their evolution, either to reduce the search space, or to deliver more understandable formulas. Resarch work in these areas is needed and they will undoubtedly be considered as future research directions.

References

[1] T. Back, F. Hoffmeister, and H. Schwefel. A survey of evolution strategies. In *Proceedings of the 4th International Conference on Genetic Algorithms*, pages 2–9, San Diego, 1991.

[2] E. Bolis, C. Zerbi, P. Collet, J. Louchet, and E. Lutton. A GP artificial ant for image processing: Preliminary experiments with EASEA. In *EUROGP 2001*, pages 246–255, Como, 2001.

[3] P. Collet, E. Lutton, M. Schoenauer, and J. Louchet. Take it EASEA. In *Proceedings of PPSN VI, Springer, LNCS 1917*, pages 891–901, Paris, 2000.

[4] J. M. Daida, T. F. Bersano-Begey, S. J. Ross, and J. F. Vesecky. Evolving feature-extraction algorithms: Adapting genetic programming for image analysis in geoscience and remote sensing. In *Proceedings of the International Geoscience and Remote Sensing Symposium: Remote Sensing for a Sustainable Future*, Washington, 1996.

[5] C. Fonlupt and D. Robilliard. Genetic programming with dynamic fitness for a remote sensing application. In *Proceedings of Parallel Problem Solving from Nature (PPSN'2000)*, pages 191–200, Paris, 2000.

[6] J. J. Freeman. A linear representation for GP using context free grammars. In *Genetic Programming 1998: Proceedings of the Third Annual Conference*, pages 72–77, University of Wisconsin, Madison, 1998.

[7] M. D. Gates. *Biophysical Ecology*. Springer-Verlag, New York, 1980.

[8] D. E. Goldberg. *Genetic Algorithms in Search, Optimization, and Machine Learning*. Addison-Wesley, Reading, Mass, 1989.

[9] C. Harris and B. Buxton. Evolving edge detectors with genetic programming. In John R. Koza, David E. Goldberg, David B. Fogel, and Rick L. Riolo, editors, *Genetic Programming 1996: Proceedings of the First Annual Conference*, pages 309–315, Stanford University, 1996. MIT Press.

[10] F. Javed, B. Bryant, M. Crepinsek, M. Mernik, and A. Sprague. Context-free grammar induction using genetic programming. In *Proceedings of the 42nd Annual ACM Southeast Conference*, pages 404–405, Huntsville, 2004.

[11] J. R. Koza. *Genetic Programming*. Cambridge: The MIT Press/Bradford Books, 1992.

[12] T. Mohr, 1999. Répertoire CGMS des Applications Météorologiques Satellitales (*in french*), EUMETSAT, document n°EUM BR 08, online version on http://www.eumetsat.int.

[13] D. J. Montana. Strongly typed genetic programming. Technical Report #7866, Bolt Beranek and Newman, Inc., Cambridge, 1994.

[14] D. Robilliard and C. Fonlupt. Backwarding: An overfitting control for genetic programming in a remote sensing application. In *Proceedings of Artificial Evolution, LNCS 2310*, pages 245–254, 2001.

[15] B. J. Ross, A. G. Gualtieri, F. Fueten, and P. Budkewitsch. Hyperspectral image analysis using genetic programming. In *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO 2002)*, pages 1196–1203, 2002.

[16] M. Schoenauer and Z. Michalewicz. Evolutionary computation, control and cybernetics. *Special Issue on Evolutionary Computation*, 26(3):307–338, 1997.

[17] TIDE, 2005. Tidal Inlets Dynamics and Environment, Research Project Supported by the European Commission under the Fifth Framework Programme, contract n° EVK3-CT-2001-00064, document available at http://www.istitutoveneto.it/tide.

[18] G. Valigiani, C. Fonlupt, and P. Collet. Analysis of GP improvement techniques over the real-world inverse problem of ocean colour. In *EUROGP'04*, pages 174–186, Coimbra, 2004.