Symbolic Regression via Genetic Programming

Douglas A. Augusto and Helio J.C. Barbosa
Laboratório Nacional de Computação Científica
C.P. 95113 CEP: 25651-070
Petrópolis - RJ, Brasil
e-mail: hcbm@lncc.br

Abstract

In this work we present an implementation of symbolic regression which is based on genetic programming (GP). Unfortunately, standard implementations of GP in compiled languages are not usually the most efficient ones. The present approach employs a simple representation for tree-like structures by making use of Read's linear code, leading to more simplicity and better performance when compared with traditional GP implementations. Creation, crossover and mutation of individuals are formalized. An extension allowing for the creation of random coefficients is presented. The efficiency of the proposed implementation was confirmed in computational experiments which are summarized in this paper.

1. Introduction

Given a class of functions and a set of experimental observations, the problem of searching for the element in this class that best fits the given data is known as regression. In its more usual form, the structure of the function is predefined by the analyst and one is left with the problem of determining certain coefficients of that function. A certain measure of the distance between the response predicted by the function/model and the data available is then minimized by a suitable optimization procedure. The case where the structure of the function is not "a priori" defined and thus must be found together with all its parameters is known as symbolic regression.

This is a problem that can be approached by genetic programming (GP)[2] which is a specialized genetic algorithm[1] where a population of strings which encode algebraic expressions is evolved mimicking the biological evolutionary process.

In this paper, the algebraic expressions are written as rooted trees which are then encoded using a specialized scheme which employs an adaptation of Read's linear code for genetic programming[5]

The remainder of the paper is organized as follows. Section 2 describes Read's linear code adapted for GP, Section 3 presents the proposed implementation for symbolic regression, Section 4 summarizes some numerical experiments and Section 5 draws some conclusions, outlines the research in progress and suggests future works.

2. Read's linear code for GP

Given a tree $T = T(V, E)$, formed by a set of vertices V and a set of edges E (see Figure 1), Read's linear code is an alternative way of representing it by a vector of non-negative integers $(\alpha_1, \alpha_2, \ldots, \alpha_p)$ which is efficiently manipulated by compiled languages.

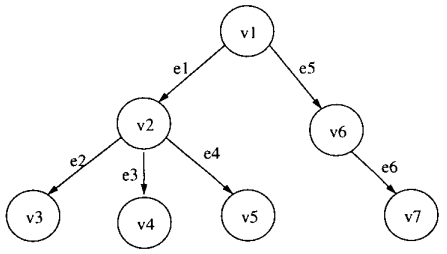


Figure 1. An example of a tree $T = T(V, E)$.

The code is generated by recursively traversing the tree beginning in the root and then visiting the left child followed by the right child, as exemplified in Figure 2, in what is known as pre-order.

Finally, Figure 3 exemplifies how a simple tree is coded beginning with the initial configuration (diagram A), assigning to each vertex a value corresponding to the number of childs (diagram B) and ending with a vector of integers at root (diagram C) which represents the whole tree.

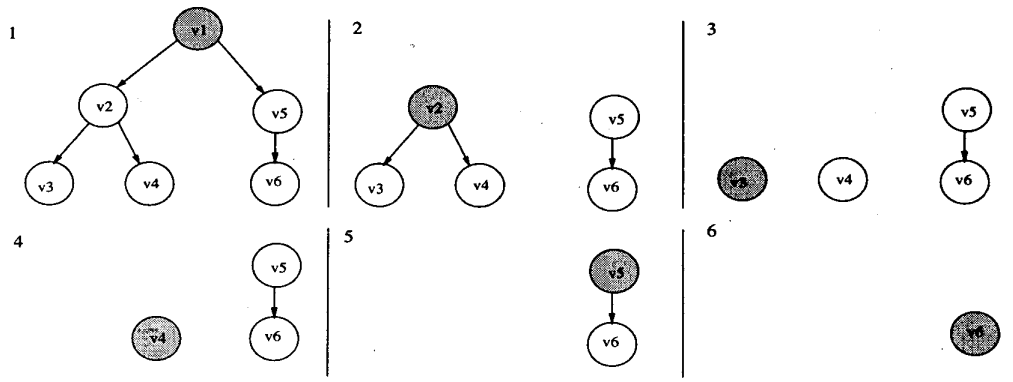


Figure 2. Traversing a tree.

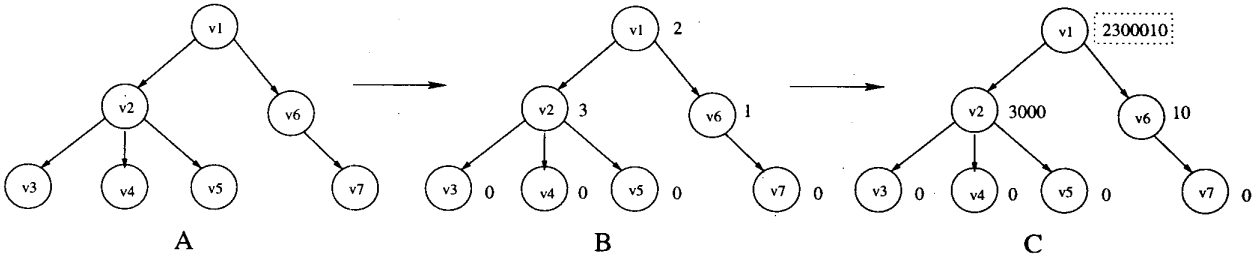


Figure 3. Linear code representation of a tree.

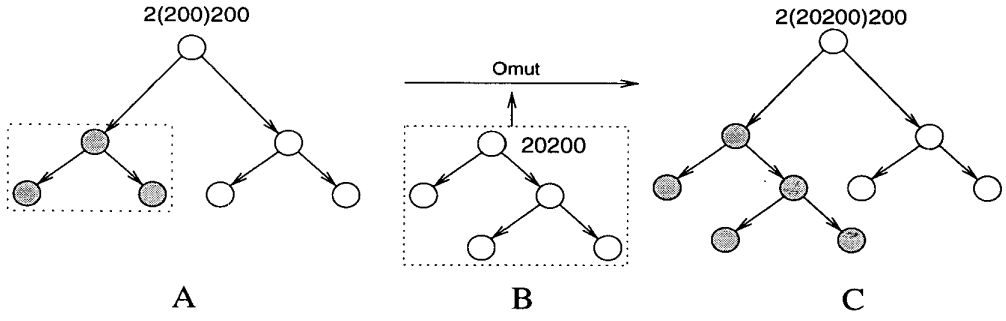


Figure 4. An example of mutation.

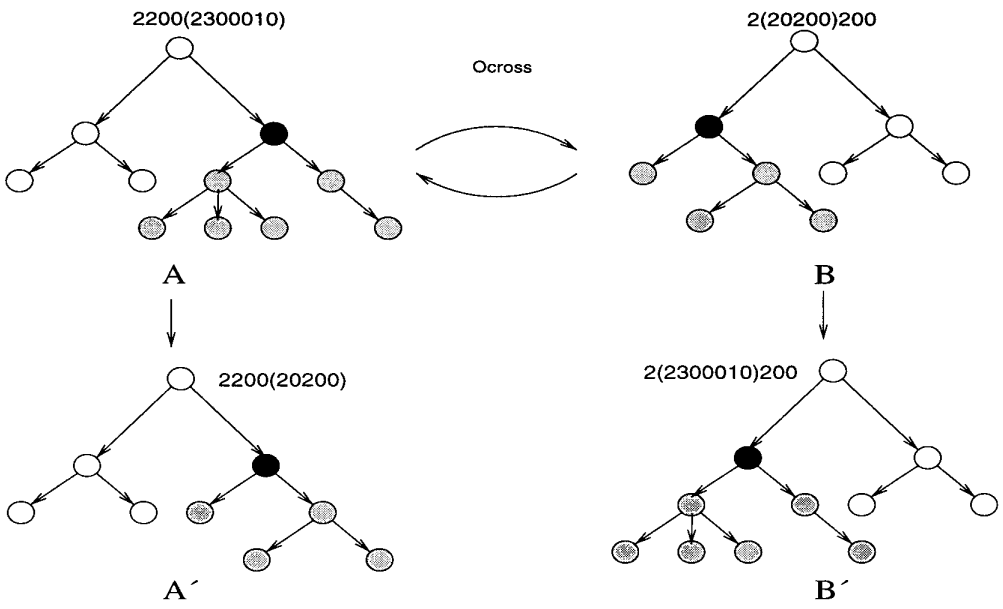


Figure 5. An example of the crossover operation.

Using the linear code, a mutation operation can be defined by randomly choosing a point (sub-tree) and substituting it for a new sub-tree randomly generated as exemplified in Figure 4.

A crossover operator can be defined by exchanging two randomly selected sub-trees belonging to each parent, generating two new individuals as exemplified in Figure 5.

It is important to note that when using the original linear code for symbolic regression[5], real coefficients must be evolved from elements of the terminal set operated upon by elements of the function set. In this paper an extension of the linear code is proposed in order to allow for the inclusion of real multiplicative coefficients.

3. The proposed implementation

In the proposed GP implementation for symbolic regression the candidate solutions can be represented in one of two ways: (i) with the finite set of functions and terminals without allowing for randomly generated coefficients or (ii) in a new extended form of the linear code capable of accommodating real numbers as multiplicative coefficients applied to the corresponding function or terminal. Figure 6 exemplifies the extended chromosome (diagram A), the original tree (diagram B) and the final represented tree (diagram C) for a situation where only terminals and one-argument functions have a corresponding coefficient.

The crossover operator for the extended linear code can be defined analogously to the case with no coefficients, as exemplified in Figure 7.

The mutation operation previously defined extends itself trivially to the case where coefficients are present. Additionaly, a mutation operator which acts only upon the coefficients is introduced as given by Michalewicz[4]. The basic idea consists in allowing larger perturbations in the initial generations and gradually reducing them along the evolutionary process.

The proposed GP implementation for symbolic regression uses a generational scheme with rank-based selection and elitism (a user-prescribed number of individuals of the elite of the population in a given generation is copied into the next one without modifications).

4. Computational experiments

This section presents some results obtained with the proposed algorithm which was coded in C, compiled with `gcc` under Linux and run on an Intel Pentium MMX 200 MHz personal computer.

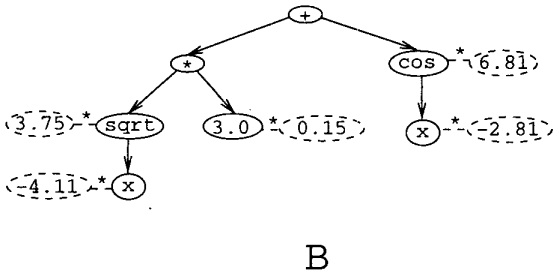
In all examples the crossover probability was set to 0.96 and the elitism was applied to the individuals ranked in the top $1/100$th of the population.

4.1. Example 1

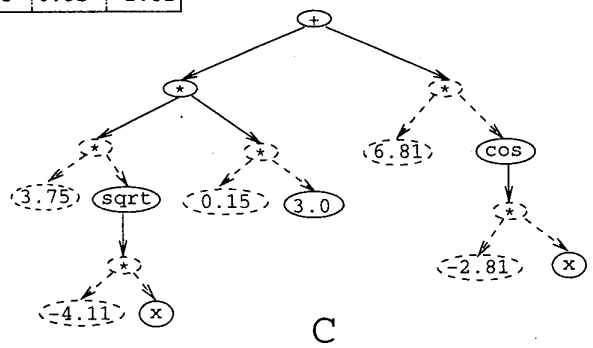
The first example[7] consists in finding the polynomial $f(x) = x^4 + x^3 + x^2 + x$ from 17 pairs (x_i, y_i) where

A

2	2	1	0	0	1	0
+	*	sqrt	x	3	cos	x
		3.75	-4.11	0.15	6.81	-2.81



B



C

Figure 6. Representation with coefficients.

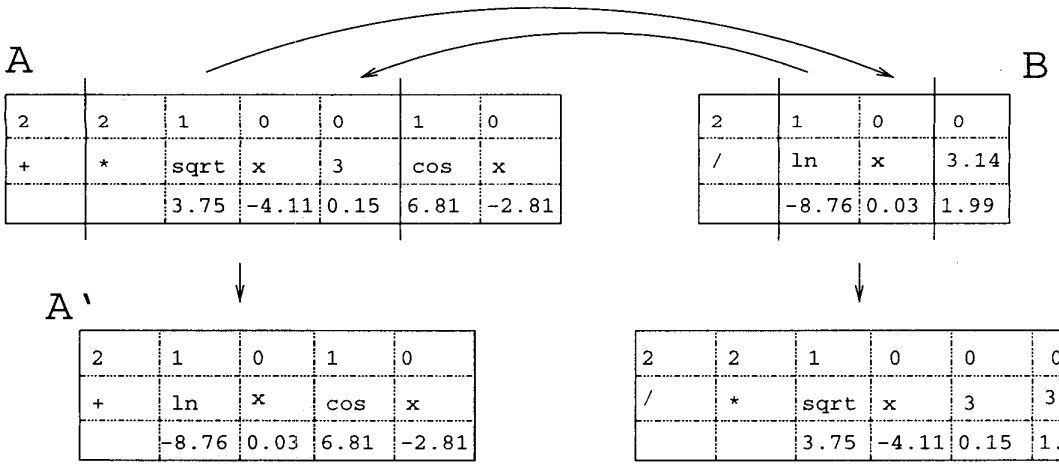


Figure 7. The crossover operator.

$y_i = f(x_i)$ and the x_i are uniformly distributed in $[-5, 5]$. The function set and the terminal set are given respectively by $F = \{+, -, *, /, \sqrt{}\}$ and $T = \{x, 1.0, 2.0, 3.0\}$

A population of 600 individuals reached the exact solution in the generation 21 after 4 seconds of execution. The maximum number of nodes in the trees was set to 25 and the mutation probability was set to 0.25. The best individual found is equivalent to the polynomial sought:

```
*(X,*(X,*(+(-(+(1.0,X),-(X,/(*(1.0,
1.0),X))),*(+(1.0,X),X)),1.0)))
```

4.2. Example 2

In the second example[6], the function $f(x) = e^{-(\sin 3x + 2x)}$ was used to generate 17 pairs (x_i, y_i) uniformly distributed in $[-\pi/2, \pi/2]$. For this problem the function set was enlarged to $F = \{+, -, *, /, \sqrt{}, \sin, \cos, \exp\}$ and the terminal set remained the same: $T = \{x, 1.0, 2.0, 3.0\}$. The maximum number of nodes in the trees was set to 20 and the mutation probability was set to 0.20.

A population of 1000 individuals evolved the solution

```
exp(-(-(cos(+(X,+(+(exp(cos(2.0)),
+(sin(2.0),X)),X))),X),X))
```

after 116 generations and 40 seconds of execution. The expression corresponds to

$$g(x) = e^{[\cos\left(e^{\cos 2} + 3x + \sin 2\right) - 2x]}$$

and can be simplified to

$$g(x) = e^{[\cos(3x + 1.5688) - 2x]}$$

which is a very good approximation to $f(x)$ since $\cos(3x + 1.5688) \approx \cos(3x + \pi/2) = -\sin(3x)$.

4.3. Example 3

The third example[3] consists in finding the symbolic expression for the data generated by the function $f(x) = 2.718x^2 + 3.1416x$ in $[-\pi, \pi]$. Seventeen pairs (x_i, y_i) were used (with the x_i uniformly distributed) and a population size of 600 was adopted. The function set is $F = \{+, -, *, /, \sin, \cos\}$ and the terminal set is $T = \{x, 1.0, 2.0, 3.0\}$. In this example real multiplicative coefficients are allowed to evolve by using the extended linear code. The maximum number of nodes in the trees was set to 20 and the mutation probability was set to 0.25. Also, with probability 0.20, Michalewicz's mutation operator is applied to the extended part of the linear code and the fitness of the mutated individual is computed. If no improvement is observed the genetic code is restored to its pre-mutation state.

After 300 generations (and 39.58 seconds of execution) the best solution was

```
+(*(*(2.250208,X),*(1.208496,X)),
*(*(2.250208,X),*(1.396278,1.0)))
```

which can be simplified to $2.7193x^2 + 3.1419x$.

When plotted, both curves are undistinguishable from each other and are thus not displayed here.

4.4. Example 4

Here[7] we will try to perform the symbolic regression of the data generated by the function $f(x) = \cos(2x)$ in the interval $[-\pi, \pi]$ excluding the function cosine from the function set: $F = \{+, -, *, /, \sqrt{}, \sin\}$. The terminal set will be $T = \{x, 1.0, 2.0, 3.0\}$ and 17 points, uniformly distributed are used. The maximum number of nodes in the trees was set to 25 and the mutation probability was set to 0.25. The population size was 500 and, after 15 generations, the following individual was found:

```
sin(+(*(2.0,X),sqrt(sqrt(*(3.0,2.0)))))
```

which can be simplified to $\sin(2x + \sqrt{\sqrt{6}})$, where $\sqrt{\sqrt{6}} \approx 1.5651$ is an approximation for $\pi/2$ leading to an excellent fitting (the curves are undistinguishable within plotting accuracy) to the original function

$$f(x) = \cos(2x) = \sin(2x + \pi/2).$$

4.5. Example 5

The last example[6] consists in finding the symbolic expression that best describes the relationship given by 33 pairs (x_i, y_i) uniformly distributed in $(0, 15)$ and defined by $y = f(x) = \min\{2/x, \sin(x) + 1\}$. The function set is $F = \{+, -, *, /, \sin, \cos\}$ and the terminal set is $T = \{x, 1.0, 2.0, 3.0\}$.

The maximum number of nodes in the trees was set to 65 and the mutation probability was set to 0.15. The population size was set to 600 and the best evolved individual $g(x)$ was:

```
/(+(X,2.0),+(+(-(2.0,*(X,/(cos(2.0),/(*
(X,sin(sin(sin(2.0)))),-(-(-(*(X,X),/
(3.0,+(+(cos(*(-(-(*(sin(sin(sin(2.0)))
,2.0),X),2.0),2.0)),2.0),cos(cos(3.0)))
)),2.0),-(2.0,/(2.0,+(cos(/(sin(+(X,X))
,2.0)),sin(X)))))))),1.0),sin(2.0)))
```

and was found after 3060 generations which consumed 50 minutes. Figure 8 displays the obtained solution together with the original function.

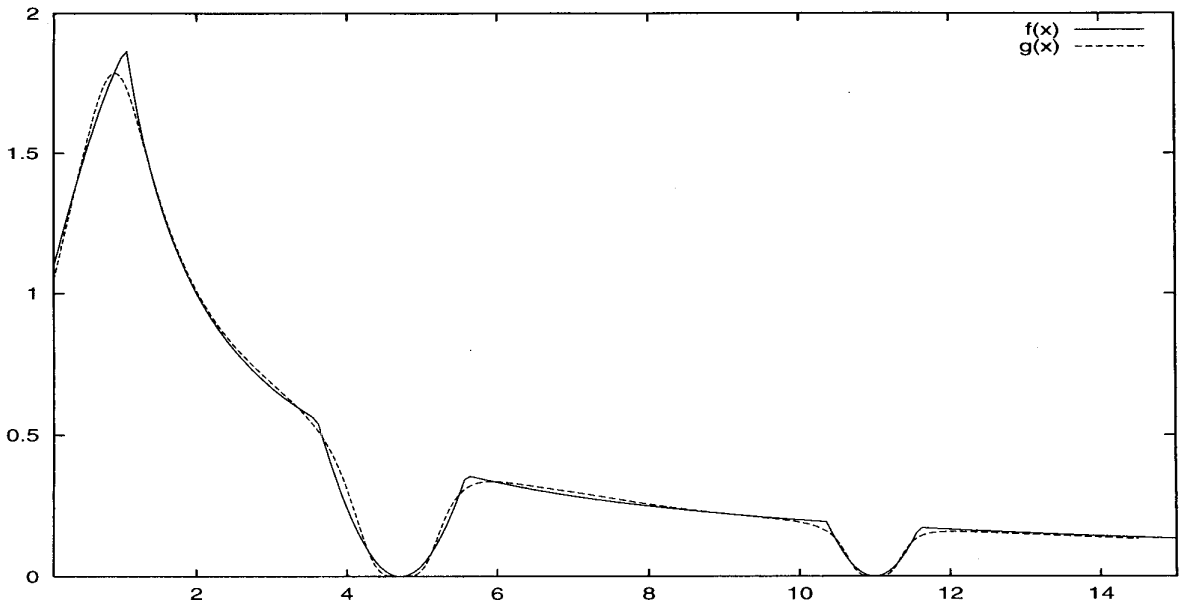


Figure 8. Original function $y = f(x) = \min\{2/x, \sin(x) + 1\}$ and the solution $g(x)$ found.

5. Conclusions and future work

In the present work we described an implementation for solving symbolic regression problems within the genetic programming paradigm. Moreover, we explained how to represent a chromosome (rooted tree) under a linear code extended here in order to accommodate real coefficients. The results are positive and show substantial gains over using unsophisticated coding of trees based on pointers pointing to successive nodes (standard implementation). The next steps in our research involve: (i) better tunning of the coefficients, (ii) move to a full Object Oriented paradigm, (iii) add new operators such as *Edit mutation*, *Permutation mutation*, and *Shrink mutation*, (iv) add an intelligent crossover in order to preserve context and (v) add a cost to each function and/or terminal, assigning greater chance of reproduction to individuals with smaller cost.

References

[1] D. Goldberg. *Genetic algorithms in search, optimization, and machine learning.* Addison-Wesley Reading, MA, 1989.

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

[3] J. R. Koza. Genetic programming: a paradigm for genetically breeding populations of computer programs to solve problems. Technical Report CS-TR-90-1314, Stanford University, Department of Computer Science, June 1990.

[4] Z. Michalewicz. *Genetic Algorithms + Data Structures = Evolution Programs.* Springer-Verlag, 1992.

[5] M. Pelikan, V. Kvasnicka, and J. Pospichal. Read's linear codes and genetic programming. In J. R. Koza, K. Deb, M. Dorigo, D. B. Fogel, M. Garzon, H. Iba, and R. L. Riolo, editors, *Genetic Programming 1997: Proceedings of the Second Annual Conference*, page 268, Stanford University, CA, USA, 13-16 July 1997. Morgan Kaufmann.

[6] H. Pöyhönen and D. Savic. Symbolic regression using object-oriented genetic programming (in C++). Technical Report 96/04, *Centre For Systems And Control Engineering* School of Engineering, University of Exeter, Exeter, United Kingdom, 1996.

[7] A. Salhi, H. Glaser, and D. De Roure. Parallel implementation of a genetic-programming based tool for symbolic regression. *Information Processing Letters*, 66(6):299–307, June 1998.