Asociación Argentina
de Mecánica Computacional

Mecánica Computacional Vol XXIX, págs. 9107-9124 (artículo completo)
Eduardo Dvorkin, Marcela Goldschmit, Mario Storti (Eds.)
Buenos Aires, Argentina, 15-18 Noviembre 2010

COMPARING TWO WAYS OF INFERRING A DIFFERENTIAL EQUATION MODEL VIA GRAMMAR-BASED IMMUNE PROGRAMMING

Heder S. Bernardino[a] **and Helio J.C. Barbosa**[b]

Laboratório Nacional de Computação Científica,
Av. Getúlio Vargas, 333, 25.651-075, Quitandinha,
Petrópolis–RJ, Brazil

[a]*hedersb@gmail.com, http://www.lncc.br/~hedersb*

[b]*hcbm@lncc.br, http://www.lncc.br/~hcbm*

Keywords: artificial immune systems, grammatical evolution, grammar-based immune programming, symbolic regression, model inference

Abstract. An ordinary differential equation (ODE) is a mathematical form to describe physical or biological systems composed by time-derivatives of physical positions or chemical concentrations as a function of its current state. Given observed pairs, a relevant modeling problem is to find the symbolic expression of a differential equation which mathematically describes the concerned phenomenon.

The Grammar-based Immune Programming (GIP) is a method for evolving programs in an arbitrary language by immunological inspiration. A program can be a computer program, a numerical function in symbolic form, or a candidate design, such as an analog circuit. GIP can be used to solve symbolic regression problems in which the objective is to find an analytical expression of a function that better fits a given data set.

At least two ways are available to solve model inference problems in the case of ordinary differential equations by means of symbolic regression techniques. The first one consists in taking numerical derivatives from the given data obtaining a set of approximations. Then a symbolic regression technique can be applied to these approximations. Another way is to numerically integrate the ODE corresponding to the candidate solution and compare the results with the observed data.

Here, by means of numerical experiments, we compare the relative performance of these two ways to infer models using the GIP method.

Copyright © 2010 Asociación Argentina de Mecánica Computacional http://www.amcaonline.org.ar

1 INTRODUCTION

Metaheuristics have been applied with success to several optimization problems. For the automatic generation of programs, the Genetic Programming (GP) paradigm has become very popular. Smith (1980, 1983) report the first results of the GP technique. The method was also studied by other authors and greatly expanded by Koza (1992).

The tree structure proposed by Cramer (1985) is often used to represent a candidate solution. That data structure can represent a computer program, a numerical function in symbolic form, or a candidate design, such as an analog circuit. An objective function which translates a conveniently defined performance index of the candidate program, function, or design is then optimized. According to Koza (1992), “fitness begets structure", i.e., over a period of time (generations), fitness induces structure via natural selection and the effects of gene recombination and mutation. The reader can find a freely available survey about GP in Poli et al. (2008).

There are not many studies in the literature using artificial immune systems (AISs) as search mechanism for this class of problem. An AIS programming proposal inspired by the GP paradigm can be found in Johnson (2003). That method uses a tree-based representation and was applied to symbolic regression problems (to derive the analytical expression of a function that better fits a given data set). The Immune Programming (IP) technique was proposed by Musilek et al. (2006) and shown to outperform GP for the considered symbolic regression problems. IP was extended by Lau and Musilek (2009) which have obtained better results than previously presented in the literature when applied to evolve a model of disinfection of *Cryptosporidium parvum*. Also the “elitist Immune Programming" [Ciccazzo et al. (2008)], or simply eIP, is inspired by clonal selection and uses the GP machinery to generate the candidate solutions. In eIP, the best individual of the population and its hypermuted clone are always kept in the population. The technique was applied to circuit design optimization where it outperformed both the standard IP and Koza's GP [Koza et al. (2000)]. Finally, the Clonal Selection Programming (CSP) algorithm, proposed by Gan et al. (2009b,a), uses a symbolic string of fixed length. Fault detection system [Gan et al. (2009b)] as well symbolic regression [Gan et al. (2009a)] problems were used to evaluate and compare the algorithm with IP and gene expression programming (GEP) [Ferreira (2001)].

Grammatical Evolution (GE), proposed by Ryan et al. (1998), is an approach to evolve programs by means of a user defined grammar. GE uses a Backus-Naur Form (BNF) to express a grammar which defines the syntax of the language, and a variable length string to represent the candidate solutions. That work was extended in O'Neill and Ryan (2001) presenting good results for symbolic regression and integration, and Santa Fe Ant Trail problems. Also, different metaheuristics can be found in the literature as its search engine: genetic algorithms (GA) [O'Neill and Ryan (2001)], particle swarm optimization (PSO) [O'Neill et al. (2004)], differential evolution (DE) [O'Neill and Brabazon (2006)], and artificial immune systems (AISs) [Bernardino and Barbosa (2009b)].

In the AISs field, a co-evolutionary strategy was proposed by Amarteifio and O'Neill (2005) where a simple GA evolves with the help of an immune inspired method aiming at promoting diversity. The approach was applied to symbolic regression and multiplexer problems (a classical GP problem where the objective is to discover a boolean expression). Another immune approach using GE can be found in McKinney and Tian (2008) where the grammatical immunoglobulin hypermutation, an evolutionary operator in which non-terminals are mutated to another according to a specified grammar, was proposed. The technique was applied to the inference problem of a kinetic model for the oxidative metabolism of 17β-estradiol (E_2).

Bernardino and Barbosa (2009b, 2010) proposed the Grammar-based Immune Programming (GIP). According to this technique, the candidate solutions are represented as in O'Neill and Ryan (2001) but a new decoding process is performed. The decoding procedure creates a tree (based on the grammar definition) and repairs it (via repair derivation rules) when necessary, always generating valid programs in the evolutionary process. GIP was applied to symbolic regression and integration problems, inference of ordinary differential equations, and inference of iterated and discontinuous functions. The results found are better than previously found in the literature for most problems. Also the repair method shown improves the performance also when other search mechanisms were used, such as GA [Bernardino and Barbosa (2010)].

Differently from the symbolic regression problem, when the objective is to infer an ordinary differential equation the user must figure out a way to compare the candidate solutions with the given data set. At least two ways can be thought to solve this task by means of symbolic regression techniques, namely: to take the numerical derivatives from the given data or to numerically integrate the ODE corresponding to the candidate solutions. In this paper, GIP is analyzed with respect to these two different alternatives to identify a causal model (an ordinary differential equation) from an observed data set.

The remaining of this paper is structured as follows. The two alternative ways to infer ordinary differential equations by means of symbolic regression are presented in Section 2. Section 3 describes GE while the Grammar-based Immune Programming and the repair method are described in Section 4. Experiments, results, and comparisons can be shown in Section 5. Finally, Section 6 concludes the paper.

2 INFERENCE OF ORDINARY DIFFERENTIAL EQUATION

A set of ordinary differential equations (ODEs) is one form of a mathematical description of a physical or biological system. They can describe the time-derivatives of physical positions or chemical concentrations as a function of its current state [Schmidt and Lipson (2008)].

The problem is to find a symbolic form for $f(x, y)$ such that the solution of the differential equation $y' = f(x, y)$ matches the given data. That is to find a differential equation model which describes the behavior of the observed pairs (x_j, y_j), where $y = y(x)$, $j = 1, \ldots, m$, and m is the number of observed values.

There are at least two ways to solve this problem by using symbolic regression techniques. The first one consists in taking numerical derivatives from the given data obtaining a set of approximations $\bar{y}'_i \approx y'_i$. Then a symbolic regression technique can be applied in order to find $f(x, y)$ that minimizes the distance between $f(x, y)$ and $\bar{y}'_i$. The two- and five-point numerical differentiation are considered for our study and can be defined as

$$\begin{aligned}
\bar{y}'_i &= \frac{y_{i+1} - y_i}{h}\,, && \text{for two-point numerical differentiation}\\
\bar{y}'_i &= \frac{y_{i-2} - 8y_{i-1} + 8y_{i+1} - y_{i+2}}{12h}\,, && \text{for five-point numerical differentiation,}
\end{aligned}$$

where $h = x_{i+1} - x_i$ and here this is equal for all possible i.

Another way to solve this problem is to numerically integrate the ODE $y' = f(x, y)$ corresponding to the phenotype $f(x, y)$ of a given candidate solution and compare it to the observed pairs (x_i, y_i). Here the 4-th order Runge-Kutta method is used to obtain the numerical integra-

Copyright © 2010 Asociación Argentina de Mecánica Computacional http://www.amcaonline.org.ar

tion $\bar{y}$ of the candidate solution and can be calculated by

$$\begin{aligned}
\bar{y}_{i+1} &= \bar{y}_i + \frac{h}{6}\left(k_1 + 2k_2 + 2k_3 + k_4\right) \\
k_1 &= f\left(t_i, \bar{y}_i\right) \\
k_2 &= f\left(t_i + \frac{h}{2}, \bar{y}_i + \frac{h}{2}k_1\right) \\
k_3 &= f\left(t_i + \frac{h}{2}, \bar{y}_i + \frac{h}{2}k_2\right) \\
k_4 &= f\left(t_i + h, \bar{y}_i + hk_3\right),
\end{aligned}$$

where $\bar{y}_0 = y_0$ is the initial value.

This second approach tends to be more accurate since 4-th order Runge-Kutta method has an error of the order h^5 while the error using five-point numerical differentiation is of the order h^4. However, it is also much more computationally expensive because a numerical integration is required in order to evaluate the error of each candidate solution (its affinity). Besides we are interested in infer a model which better represents the observed phenomena and, as are shown in Section 5, the accuracy of the numerical method used by the search technique is not necessarily proportional to the quality of the solutions.

Although Bernardino and Barbosa (2010) have used the integration of the candidate solution rather than signal derivation, other works can be found considering this last way [Iba (2008)]. The objective of this work is evaluate what is the behavior of the search mechanism (GIP) over this class of problem and the influence of the alternative ways in its performance, analyzing not only the accuracy of the final solutions but also their generalization.

3 GRAMMATICAL EVOLUTION

Grammatical Evolution (GE) originally used a GA as the search engine in which the genotype is mapped onto terminals by means of a defined grammar. This technique was proposed by Ryan et al. (1998) and extended by O'Neill and Ryan (2001). According to Chomsky (2002), a formal grammar G can be

$$G = \{N, \Sigma, R, S\}, \tag{1}$$

where N is a finite set of non-terminals, Σ is a finite set of terminals or symbols which can appear in the language (such as x, $+$, and $\cos$), R is a finite set of rules, and $S \in N$ is the start symbol. Each rule (or production) from R is described in Backus-Naur Form (BNF) as

$$\left(\Sigma \cup N\right)^* N \left(\Sigma \cup N\right)^* ::= \left(\Sigma \cup N\right)^*, \tag{2}$$

where $\cup$ denotes set union and * is a unary operation widely used for regular expressions in which if N is a set of symbols or characters, then N^* is the set of all strings composed by symbols in N.

The context-free grammar, used by standard GE, can be defined as $N ::= \left(\Sigma \cup N\right)^*$. If the right side of this expression is composed by more than one sequence, the choices are delimited by the symbol "|". A non-terminal can be expanded into one or more terminals and non-terminals likewise may self-reference to specify recursion.

A context-free grammar to generate simple mathematical expressions in scheme language (one of the two main dialects of the programming language Lisp) is given by Expression (1)

Copyright © 2010 Asociación Argentina de Mecánica Computacional http://www.amcaonline.org.ar

where

$$\begin{aligned} N &= \{<expr>, <op>, <unit\_op>, <var>\} \\ \Sigma &= \{sin, cos, log, exp, sqrt, x, y, 1, 2, 3, +, -, *, div, pow, (,)\} \\ S &= <expr> \end{aligned}$$

and R is defined as

$$\begin{array}{rcll} <expr> & ::= & (<op><expr><expr>) & (0) \\ & & \mid (<unit\_op><expr>) & (1) \\ & & \mid\ <var> & (2) \\ <op> & ::= & + & (0) \\ & & \mid - & (1) \\ & & \mid * & (2) \\ & & \mid div & (3) \\ & & \mid pow & (4) \\ <unit\_op> & ::= & sin & (0) \\ & & \mid cos & (1) \\ & & \mid log & (2) \\ & & \mid exp & (3) \\ & & \mid sqrt & (4) \\ <var> & ::= & x & (0) \\ & & \mid y & (1) \\ & & \mid 1 & (2) \\ & & \mid 2 & (3) \\ & & \mid 3 & (4) \end{array}$$

Thus, $(+\ x\ (*\ x\ y))$ and $(+\ (div\ 1\ x)(sin\ y))$ represent the expressions $x + x * y$ and $1/x + \sin(y)$.

In GE, the genotype is a binary array representation of an integer array in which each value is encoded by 8 bits [O'Neill and Ryan (2001)]. The integer value is used to select a rule from the set of the grammar's productions via the following expression

$$rule = (int)mod(nr)$$

where int is an integer number and nr is the number of rules for the current non-terminal.

A genotype and its integer array representation are illustrated in Figure 1.

Table 1 presents the genotype-to-phenotype mapping process considering the grammar from Expression (1) and the candidate solution shown in Figure 1. The result of a given row is the input for the next one in Table 1.

According to Elseth and Baumgardner (1995), degeneracy of the genetic code can be observed, as in biological organisms. It means that it is not necessary to use all integer values from the array to create the phenotype.

Moreover, it is important to notice that during the genotype-to-phenotype mapping process it may happen that after using all integers in the array a complete expression is still not available.

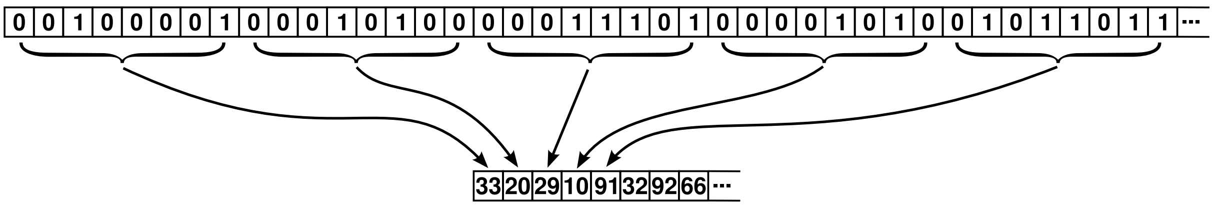

Figure 1: An example of genotype being mapped into an 8-bit integer array [Bernardino and Barbosa (2010)].

Input	Rule	Result
$< expr >$	$(33)\ mod\ (3) = 0$	$(< op >< expr >< expr >)$
$(< op >< expr >< expr >)$	$(20)\ mod\ (5) = 0$	$(+\ < expr >< expr >)$
$(+\ < expr >< expr >)$	$(29)\ mod\ (3) = 2$	$(+\ < var >< expr >)$
$(+\ < var >< expr >)$	$(10)\ mod\ (5) = 0$	$(+\ x\ < expr >)$
$(+\ x\ < expr >)$	$(91)\ mod\ (3) = 1$	$(+\ x\ (< unit\_op >< expr >))$
$(+\ x\ (< unit\_op >< expr >))$	$(32)\ mod\ (5) = 2$	$(+\ x\ (exp\ < expr >))$
$(+\ x\ (exp\ < expr >))$	$(92)\ mod\ (3) = 2$	$(+\ x\ (exp\ < var >))$
$(+\ x\ (exp\ < var >))$	$(66)\ mod\ (5) = 1$	$(+\ x\ (exp\ y))$

Table 1: Example of the mapping process [Bernardino and Barbosa (2010)].

In this case, the strategy from O'Neill and Ryan (2001) is to wrap the integer array and reuse its values (two or more times). Verify that this can not be a good solution since it is possible that the genotype-to-phenotype mapping process enters an infinite loop. An integer array composed by values so that $(int)mod(3) = 0$ causes an infinite loop when used with the grammar from Expression (1) and GE mapping-process will generate $(< op >< expr >< expr >)$, $(+(< op >< expr >< expr >) < expr >)$, $\ldots$, indefinitely. However, when this happens, it is suggested by O'Neill and Ryan (2001) that the fitness of this candidate solution be set to the worst possible value (GEVA, a well known implementation of GE, uses 10^8). It is assumed that the faulting solutions will be eliminated by the evolutionary process. However, good solutions can be eliminated and computational time is unnecessarily spent by using this policy. To solve this issue Bernardino and Barbosa (2009b, 2010) proposed the repair method which generates only valid solutions from genotype-to-phenotype mapping process.

4 GRAMMAR-BASED IMMUNE PROGRAMMING

Artificial Immune Systems are intelligent methodologies inspired by the biological immune system used to solve real world problems. Natural immunity is composed by innate and adaptive immunities. The first one is composed by cells and mechanisms that defend the host from attacks by other organisms in a non-specific manner. The later comprises highly specialized cells and processes that defend the organism from antigens. Besides, according to Dasgupta and Nino (2008), most immune inspired algorithms are based on the adaptive immunity behavior. de Castro and Timmis (2002) suggested that AISs can be applied to several classes of problem, such as: pattern recognition, scheduling, control, machine-learning, information systems security, and optimization. Bernardino and Barbosa (2009a) presents a survey focused in the application of immune inspired algorithms to solve optimization problems.

There is a selection mechanism which leads to the evolution of the immune system repertoire. Also, on binding with a suitable antigen, activation of lymphocytes occurs. Once activated, clones of the lymphocyte are produced expressing receptors identical to the original lympho-

Copyright © 2010 Asociación Argentina de Mecánica Computacional http://www.amcaonline.org.ar

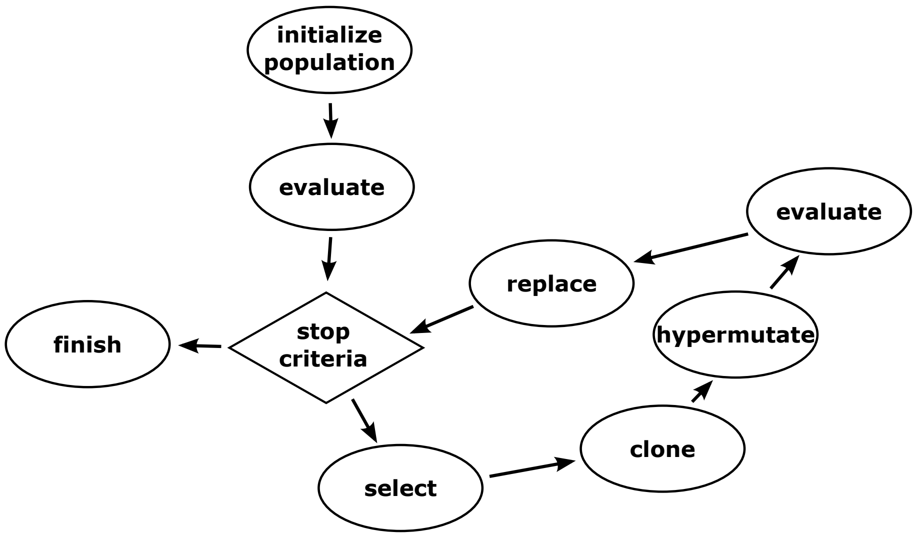

Figure 2: Diagram presenting CLONALG's execution steps.

cyte that encountered the antigen. Any lymphocyte that has receptors specific to molecules of the host organism are not cloned and die. Thus, only an antigen may cause this clonal expansion. These set of actions are called clonal expansion and clonal selection theory. The increase in the average of the affinity between the antibodies and antigens came from the somatic hypermutation suffered by the generated clones followed by the clonal selection.

A clonal selection algorithm (CLONALG) was proposed by de Castro and von Zuben (2002) and is the immune method used by the approach used here. The algorithm evolves the antibodies inspired by the concept of clonal selection where each individual is cloned, hypermutated, and those with higher affinity are selected. Normally, the mutation rate is inversely proportional to the affinity of the antibody with respect to the antigen. Moreover, inspired by the bone marrow behavior, new candidate solutions could be randomly generated to improve the exploration capability of the algorithm.

The technique used here uses a binary string to encode each candidate solution and CLONALG as the search engine. Figure 2 presents a diagram describing CLOANLG's execution flux where "initialize population" randomly generates the initial population, "evaluate" decodes and calculates the antibodies's affinity, "select" selects individuals to be cloned, "clone" clones the selected antibodies, "hypermutate" applies the somatic hypermutation in generated clones, and "replace" generates new candidate solutions and update the population of antibodies. Here an antibody is replaced by its best hypermuted clone if it is better than the original one [Cutello et al. (2005)].

The hypermutation operator used here is the defined by de Castro and von Zuben (2002) which defines the mutation rate as

$$m_r(i) = e^{-\rho f_n(i)}$$

where $m_r(i)$ is the mutation rate of the i-th cell, $f_n(i)$ is the normalized affinity (assuming unity for the best antibody) of the i-th candidate solution, and ρ is a parameter.

Also, to keep the diversity of the population: (i) every antibody in population is selected to be cloned and (ii) the number of clones is the same for all candidate solutions.

The procedure "evaluate" must be specially considered. This is where each antibody is transformed from a binary string into a tree (i.e. a symbolic representation) and evaluated with

Copyright © 2010 Asociación Argentina de Mecánica Computacional http://www.amcaonline.org.ar

respect to the objective function in order to define its affinity value. The process to decode an antibody into a tree is defined in Sections 3 and 4.1.

4.1 The repair method

Considering the CLONALG presented above, it is important to notice that all antibodies generate hypermutated clones. It means that a candidate solution which does not belongs to the language will try to evolve, mostly producing new invalid solutions. Graphics showing the maintenance of invalid solutions by this search mechanism over the evolutionary process can be found in Bernardino and Barbosa (2010). However, an “invalid" solution phenotype can be repaired in mapping procedure to always be recognized by the grammar.

In order to generate only valid candidate solutions, the repair method produces a tree in the genotype-to-phenotype mapping process and fixes it when the array of integers is exhausted and the program is not complete. The information about how the repair must operate over a invalid candidate solution is incorporated to the user defined grammar which now is defined by a 5-tuple

$$G = \{N, \Sigma, R, R_r, S\}, \tag{3}$$

where N, Σ, R, and S are previously defined in the Expression (1). R_r is a set of production rules, as defined in (2), to repair the generated tree. Each R_f rule is defined as

$$(\Sigma \cup N)^* \, N \, (\Sigma \cup N)^* \, N \, (\Sigma \cup N)^* ::= (\Sigma \cup N)^* \, \Sigma \, (\Sigma \cup N)^* , \tag{4}$$

This new set of rules must contain a backward step to create the tree when there are no more integers available. Two constraints must be highlighted: the new rule must have less subtrees than the original one and given $w \in R_f$ then the inverse transformation of w is in R, or at least can be reached by this one. It is important since a repaired program also must be recognized by the original grammar.

An example of R_f, applicable to the grammar given in Expression (1) and used in the computational experiments, can be found following

$$\begin{array}{rl} (< op >< expr >< expr >) ::= & < expr > \;|\; x \;|\; y \;|\; 1 \;|\; 2 \;|\; 3 \\ (< unit\_op >< expr >) ::= & x \;|\; y \;|\; 1 \;|\; 2 \;|\; 3 \end{array}$$

The repair method is performed unstacking the integer values used in the mapping process. Thus, the first integer in the sequence defines which rule from R_r (R_{ij}) will be the repair rule. However, not all possibilities from R_r (R_{ij}) can be considered. The options will be those with a number of non-terminals not larger than the number of non-null children of the invalid sub-tree. Also, the types of the children must be considered. Thus the children of the invalid sub-tree can be used by the new (valid) one.

Figure 3 shows how the repair method fixes the supposed “invalid" solution while it is constructed using the grammar from Expression (3). In Figure 3, the next integer in the sequence is 91 and $91 \; mod(5) = 1$. Thus the terminal y is chosen concluding the generation of the (now) valid tree.

5 COMPUTATIONAL EXPERIMENTS

In order to study the performance of GIP when solving the problem of finding the symbolic expression which better models the given observed data we apply the two alternatives presented

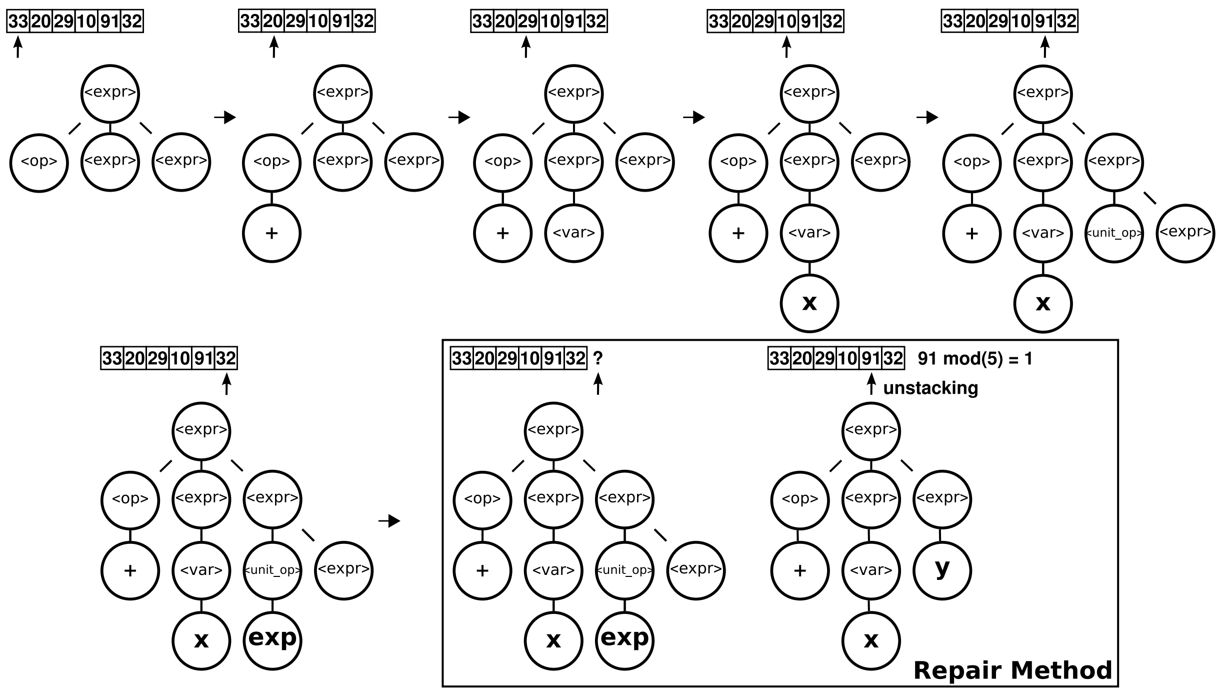

Figure 3: The repair method [Bernardino and Barbosa (2010)].

here against a suite composed by fifteen problems. Almost all problems are obtained from Ince (1956) and Bernardino and Barbosa (2010).

In all cases, 21 or 101 observed pairs (x_i, y_i) are assumed given, where x takes equally spaced values in the range of interest $[a, b]$, and y_i is the corresponding experimentally observed value. Here synthetical data are used: the “observed" values were substituted by the values $y(x_i)$ computed from the exact solution of the target ODE. The functions which define the observed data and their range of interest $[a, b]$ are presented in Table 2.

To evaluate a candidate solution by integration we used a 4-th order Runge-Kutta method. The step-sizes $h = \frac{b-a}{m}$ or $h = \frac{b-a}{10m}$ were considered; the second one corresponding to a more accurate but also more computationally expensive approach.

Considering the alternative of derivation of the observed data, we used two- or five-point numerical approximations for the derivative in order to assess how the number of points affects the solution quality.

GIP search mechanism is a clonal selection algorithm (CLONALG) with binary encoding using population size=50, length of the integer array=100, each integer encoded with 8 bits (as suggested by O’Neill and Ryan (2003)), number of clones=1, $\rho = 5$, and percentage of new randomly generated antibodies=10%. These parameters were also used in Bernardino and Barbosa (2010) and are the same for all problems with no attempt of fine tuning. Two GIP variants were considered according to the use (or not) of the repair method (indicated by R).

The error, to be minimized, is given by

$$error = \frac{1}{m}\sum_{i=1}^{m}\left(y_{i,exact} - y_{i,approximate}\right)^2 ,$$

where, m is the number of observed data-pairs. Two different values for m were used: 21 and 101.

For all problems the stop condition is 25,000 objective function evaluations, and the grammar defined by Expression (1) is applied together with the R_f presented in Section 4.1.

Copyright © 2010 Asociación Argentina de Mecánica Computacional http://www.amcaonline.org.ar

	Functions used to generate the observed values	**Range**
p_{01}	$y(x) = Ax^p + Bx^q$	$[0, 10]$
p_{02}	$y(x) = Ae^{px} + Be^{qx}$	$[0, 2]$
p_{03}	$y(x) = A\cos(qx) + B\sin(qx)$	$[0, 10]$
p_{04}	$y(x) = e^{px}\left(A\cos(qx) + B\sin(qx)\right)$	$[0, 2]$
p_{05}	$y(x) = A\cosh\left(\frac{x}{A}\right)$	$[0, 10]$
p_{06}	$y(x) = x^q\left(A + B\log(x)\right)$	$[0, 10]$
p_{07}	$y(x) = e^{px}\left(A + Bx\right)$	$[0, 2]$
p_{08}	$y(x) = (A + Bx)\cos(qx) + (C + Dx)\sin(qx)$	$[0, 10]$
p_{09}	$y(x) = e^{px}\left[(A + Bx)\cos(qx) + (C + Dx)\sin(qx)\right]$	$[0, 2]$
p_{10}	$y(x) = Ax\cos\left(\frac{q}{x} + B\right)$	$[0, 10]$
p_{11}	$y(x) = A + e^{-6x}$	$[0, 10]$
p_{12}	$y(x) = A\sin(Cx)$	$[0, 10]$
p_{13}	$y(x) = Ae^{-x^2}$	$[0, 10]$
p_{14}	$y(x) = \sqrt{A + Bx}$	$[0, 10]$
p_{15}	$y(x) = \log(A + x)(A + x) - x$	$[0, 10]$

Table 2: Functions used to generate the observed values where we have set $A = C = 1$, $B = D = p = 2$, and $q = 3$.

The comparisons are based on the average of the error. Also the two alternatives to solve the symbolic regression problem considered here are compared evaluating the final solutions achieved by the search process on all variations of each problem, that is, the solution found using the numerical derivative is also evaluated integrating its final solution. It is important not only to compare both approaches in the same way but also to verify the generalization power of the final solutions found.

Concluding the comments about the computational experiment's environment, JScheme[1] and Michael Thomas Flanagan's Java Scientific Library [2] are used by our implementation. JScheme is a implementation with a very simple interface of the Scheme language while the second library has a great collection of numerical methods, including the Runge-Kutta method considered here.

5.1 Performance profiles

The most common way of assessing the relative performance of a set V of variants $v_i, i \in \{1, \ldots, n_v\}$ is to define a set P of representative test-problems $f_j, j \in \{1, \ldots, n_p\}$ and then test all variants against all problems measuring the performance $t_{p,v}$ of variant $v \in V$ when applied to problem $p \in P$. The performance indicator to be maximized here is the inverse of the minimum objective function value found by variant v in test-problem p averaged over 30 runs.

Now a performance ratio can be defined as

$$r_{p,v} = \frac{t_{p,v}}{\min\{t_{p,v} : v \in V\}} \tag{5}$$

It is interesting to be able to assess the performance of the variants in V on a potentially large set of test-problems P in a compact graphical form. This can be attained following

[1]http://jscheme.sourceforge.net/jscheme/main.html
[2]http://www.ee.ucl.ac.uk/~mflanaga/java

Copyright © 2010 Asociación Argentina de Mecánica Computacional http://www.amcaonline.org.ar

Dolan and Moré (2002) and defining

$$\rho_v(\tau) = \frac{1}{n_p} |\{p \in P : r_{p,v} \leq \tau\}|$$

where $|.|$ denotes the cardinality of a set. Then $\rho_v(\tau)$ is the probability that the performance ratio $r_{p,v}$ of variant $v \in S$ is within a factor $\tau \geq 1$ of the best possible ratio. If the set P is large and representative of problems yet to be tackled then variants with larger $\rho_s(\tau)$ are to be preferred. The performance profiles thus defined have a number of useful properties [Dolan and Moré (2002); Barbosa et al. (2010)] such as (i) $\rho_v(1)$ is the probability that variant v will provide the best performance in P among all variants in V. If $\rho_{V1}(1) > \rho_{V2}(1)$ then variant $V1$ was the winner in a larger number of problems in P than variant $V2$, and (ii) a measure of the reliability of variant v is its performance ratio in the problem where it performed worst: $R_v = \sup\{\tau : \rho_v(\tau) < 1\}$. As a result, the most reliable variant is the one that minimizes R_v; that is, it presents the best worst performance in the set P:

$$v^* = \arg\min_{v \in V} R_v = \arg\min_{v \in V} \sup\{\tau : \rho_v(\tau) < 1\}$$

5.2 Evaluating the repair method

We first evaluated if the introduction of the repair method in the GIP technique leads to better results. GIP with and without repair method was applied to all problems presented in Table 2 considering the two alternative ways of solving them (by numerical integration or derivation). The variation of the number of observed values and internal parameters of the numerical algorithms were also considered. Since there are fifteen problems, two set of data points (21 and 101), and two classes of numerical algorithms (integrative and derivative) with two internal parameters values for each (two values for h and two number of points considered by the numerical derivative), then we have a total of $15 \times 2^3 = 120$ different tests to be performed.

Performance profiles (see Section 5.1) are used to simplify our comparison task and they are shown in Figure 4. It is easy to see that, as verified in Bernardino and Barbosa (2010), GIP with the repair method performs better than without it. Thus the results obtained by the GIP-R are used to evaluate the two alternative ways considered here to infer an ODE model.

5.3 Evaluating the two alternative ways inferring ODEs using GIP

As presented in Section 5.2, the GIP-R variant performs much better than the one without the repair method. Thus GIP-R is used to evaluate the two alternative ways to solve the symbolic regression problem for ODEs.

First we consider solving the inference problem using the numerical integration of the candidate solutions and comparing the predicted values with the observed ones. Figure 5 presents the performance profiles corresponding to the solutions found. $NI - 21 - 1$ and $NI - 101 - 1$ correspond to the use of numerical integration, $h = \frac{b-a}{m}$, and 21 or 101 observed values, respectively. The same pattern is used for the labels $NI - 21 - 10$ and $NI - 101 - 10$, when adopting $h = \frac{b-a}{10m}$. It is important to notice that it is assumed similarity between the errors what is not a strongly assumption since the errors are the averaged square error in each observed value. Figure 5 shows that for both values of h few observed values are preferred and results in best reliability. When $h = \frac{b-a}{m}$, the variant with more observed values is the most efficient. However this does not happen when $h = \frac{b-a}{10m}$.

To verify the generalization of the solutions found by the variants using numerical integration we compared their performance with respect to the other approach, that is, by numerical

Copyright © 2010 Asociación Argentina de Mecánica Computacional http://www.amcaonline.org.ar

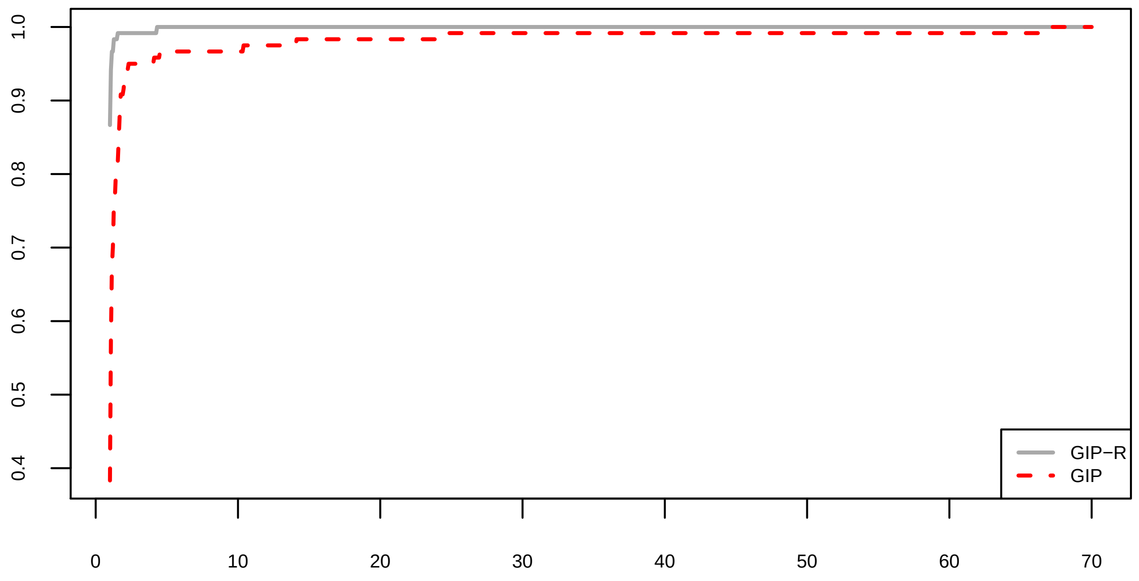

Figure 4: Performance profiles comparing the GIP technique with and without the repair method.

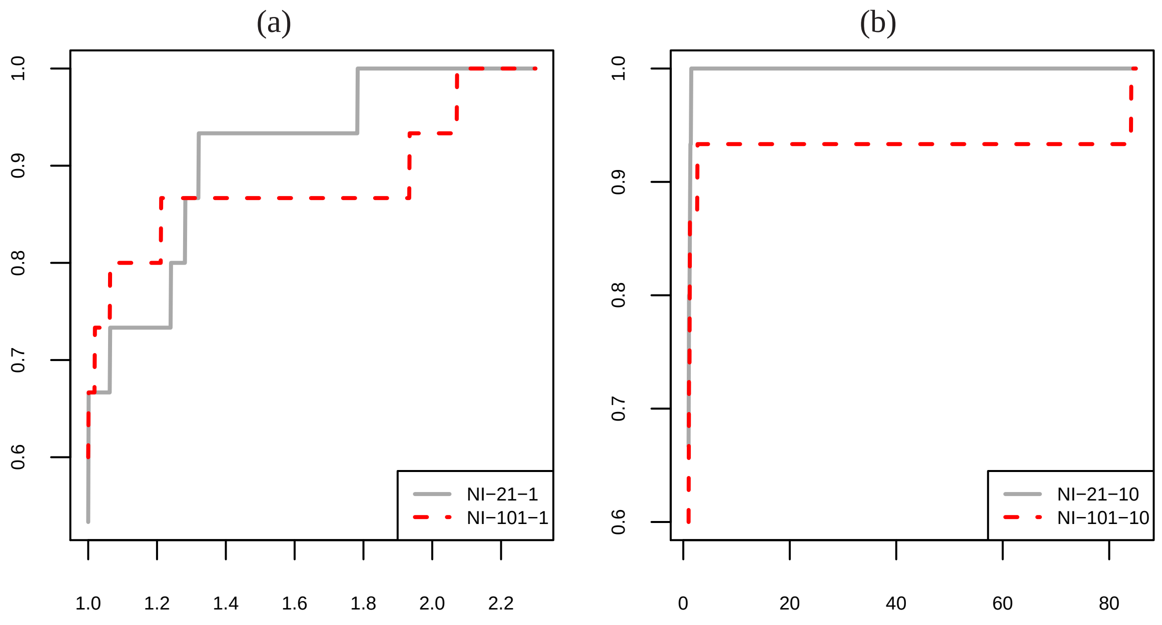

Figure 5: Performance profiles comparing the numerical integration approach with respect to h. $h = \frac{b-a}{m}$ and $h = \frac{b-a}{10m}$ are used in plots (a) and (b), respectively.

Copyright © 2010 Asociación Argentina de Mecánica Computacional http://www.amcaonline.org.ar

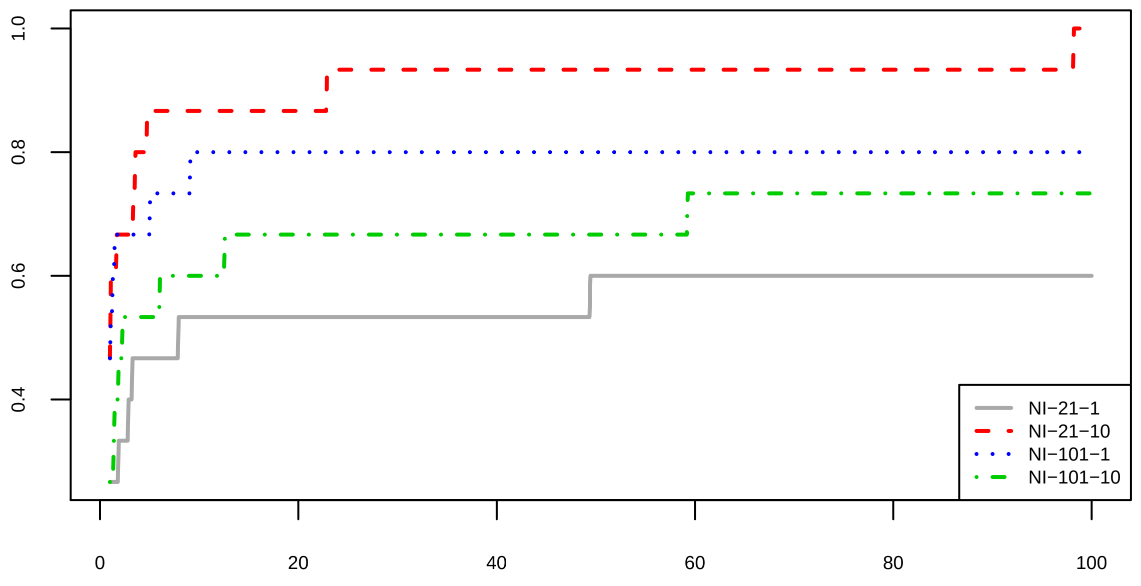

Figure 6: Performance profiles comparing the generalization of the solutions found by GIP-R using numerical integration. The errors are evaluated against 101 observed values and a five-point numerical derivative.

derivative. We used the set of user defined parameters which results in maximum accuracy: 101 observed values and five-point numerical derivative approximation. The performance profiles are presented in Figure 6 where the problem's variant $NI - 21 - 10$ achieved the best performance. This result is important because it shows that it is not necessary to have many observed values in order to obtain good results using numerical integration. Also the combination of a large amount of data and a smart h can result in overfitting.

Figure 7 shows the performance profiles corresponding to the solutions found using numerical derivative. $ND - 21 - 2$ and $ND - 101 - 2$ correspond to the use of two-point numerical derivative, and 21 or 101 observed values, respectively. The same pattern is used for the labels $ND - 21 - 5$ and $ND - 101 - 5$, where a five-point numerical derivative is adopted. As with numerical integration comparisons, here we also assumed similarity between the errors. The performance profiles show that the two-point numerical derivative approximation is better when few observed values are available while five-point are preferred otherwise. In both cases, the best efficiency is achieved with only 21 observed values.

Also, to verify the generalization of the solutions found by the variants using numerical derivative we compared their performance with respect to the other approach, that is, by the numerical integration. The variants are evaluated using 101 observed values and the five-point numerical derivative. The performance profiles are presented in Figure 8 where the problem's variant $ND-101-2$ achieved the best performance. Differently from the case using numerical integration, here the best result is achieved using 101 observed values and with the two-point numerical derivative approximation.

Since the $NI - 21 - 10$ and $ND - 101 - 2$ variants presented the best results in the generalization test, they are used in the following comparisons where the approaches are evaluated by both numerical derivative and integration. Figure 9 presents the performance profiles corresponding to the final solutions found by these variants evaluated by numerical integration with 101 observed values and $h = \frac{b-a}{10m}$ while Figure 10 shows the results evaluating the solutions with five-point numerical derivative approximation using 101 observed values. According to these plots variant $NI - 21 - 10$ performs better than $ND - 101 - 2$ for both cases. That

Copyright © 2010 Asociación Argentina de Mecánica Computacional http://www.amcaonline.org.ar

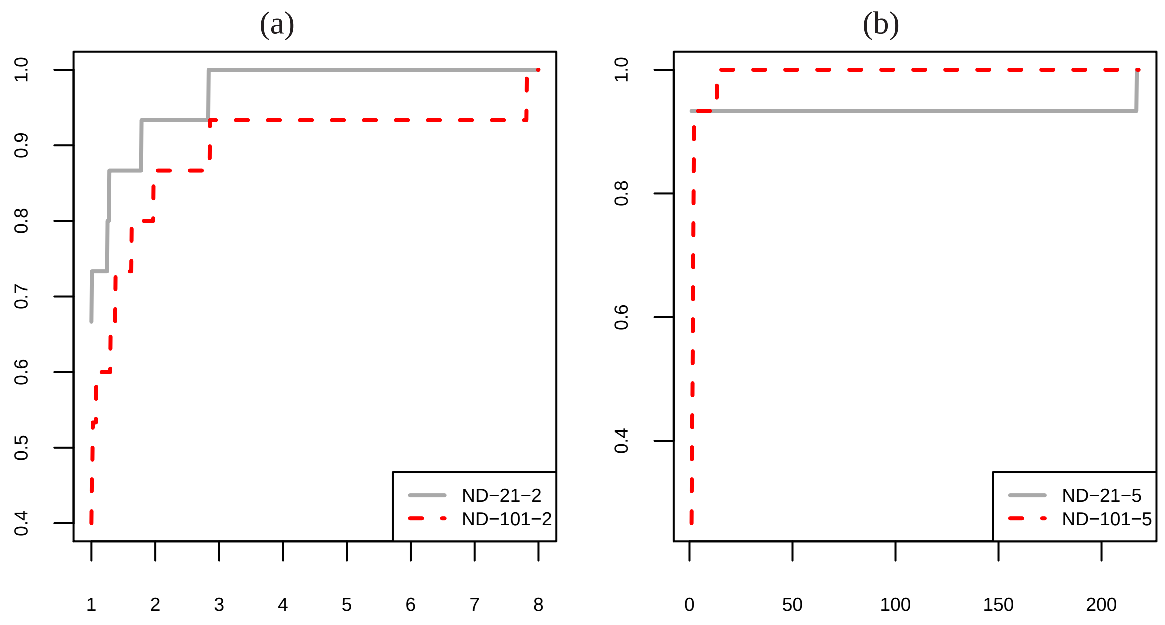

Figure 7: Performance profiles comparing the numerical derivative approach varying the number of points used in the numerical derivative approximation in each observed value. Two and five points are used in plots (a) and (b), respectively.

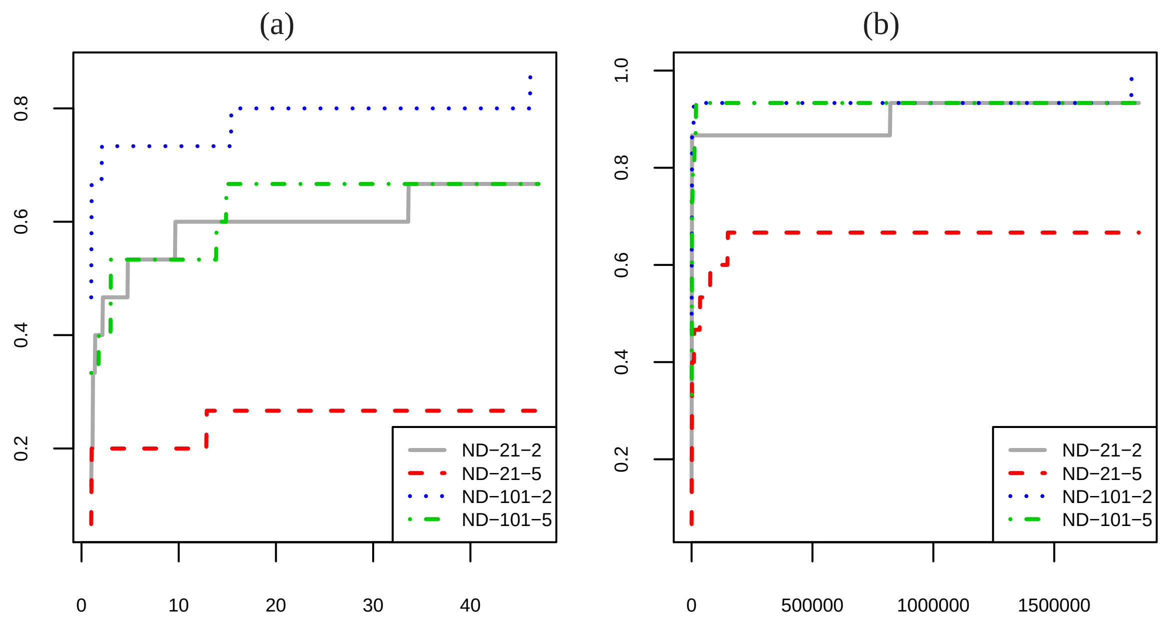

Figure 8: Performance profiles comparing the generalization of the solutions found by GIP-R using numerical derivative. The errors are evaluated with 101 observed values and a 4-th order Runge-Kutta method with $h = \frac{b-a}{10m}$. (a) presents the curves with $\tau \in [1; 47]$ while in (b) $\tau \in [1; 1.85e + 6]$.

Copyright © 2010 Asociación Argentina de Mecánica Computacional http://www.amcaonline.org.ar

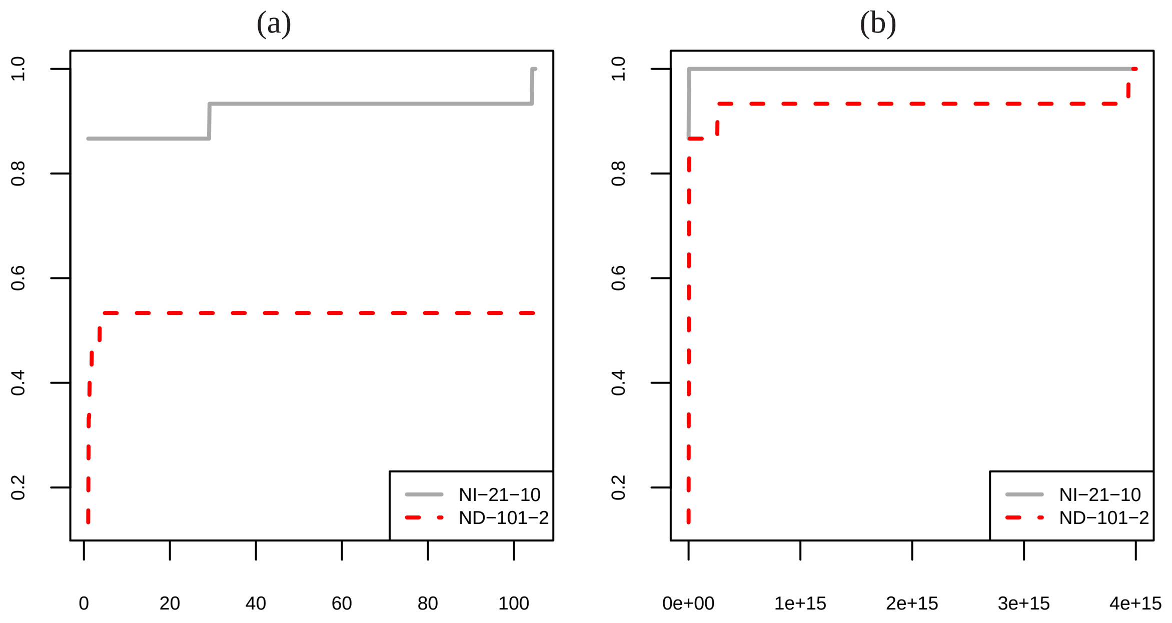

Figure 9: Performance profiles comparing the variants $NI-21-10$ and $ND-101-2$. The errors are evaluated with 101 observed values and a 4-th order Runge-Kutta method with $h = \frac{b-a}{10m}$. (a) presents the curves with $\tau \in [1; 105]$ while in (b) $\tau \in [1; 4e+15]$.

means that $NI-21-10$ not only is able to infer a solution which better matches the observed values by numerical integration but also finds solutions which generalize to the corresponding numerical derivative.

6 CONCLUSIONS

This paper presented two alternative ways to solve model inference problems in the case of ordinary differential equations (ODE) by means of symbolic regression techniques using Grammar-based Immune Programming (GIP). Also, the use of the repair method previously proposed was evaluated in this class of problems showing the efficiency of this approach.

The first way to solve the ODE inference problem consists in taking numerical derivatives from the given data obtaining a set of approximations $\bar{y}'_i \approx y'_i$ while another way is to numerically integrate the ODE $y' = f(x, y)$ corresponding to the candidate solution $f(x, y)$ and compare the results with the observed values.

Computational experiments verified that when using numerical integration few observed values together with a small integration step h achieve better generalized results. When the numerical derivative is used more observed values with a simpler numerical derivative approximation (fewer points used) is the best option.

Comparing the best variants with respect to each approach, that is, the best using numerical derivative and integration we could verify that, although more computationally expensive, the use of numerical integration leads to better results.

We believe that (i) investigating the determination of the numerical constants, (ii) guiding the search by minimizing not only the error but also the complexity of the solutions, and (iii) to apply the technique to systems of ODEs are important research avenues for future work.

Copyright © 2010 Asociación Argentina de Mecánica Computacional http://www.amcaonline.org.ar

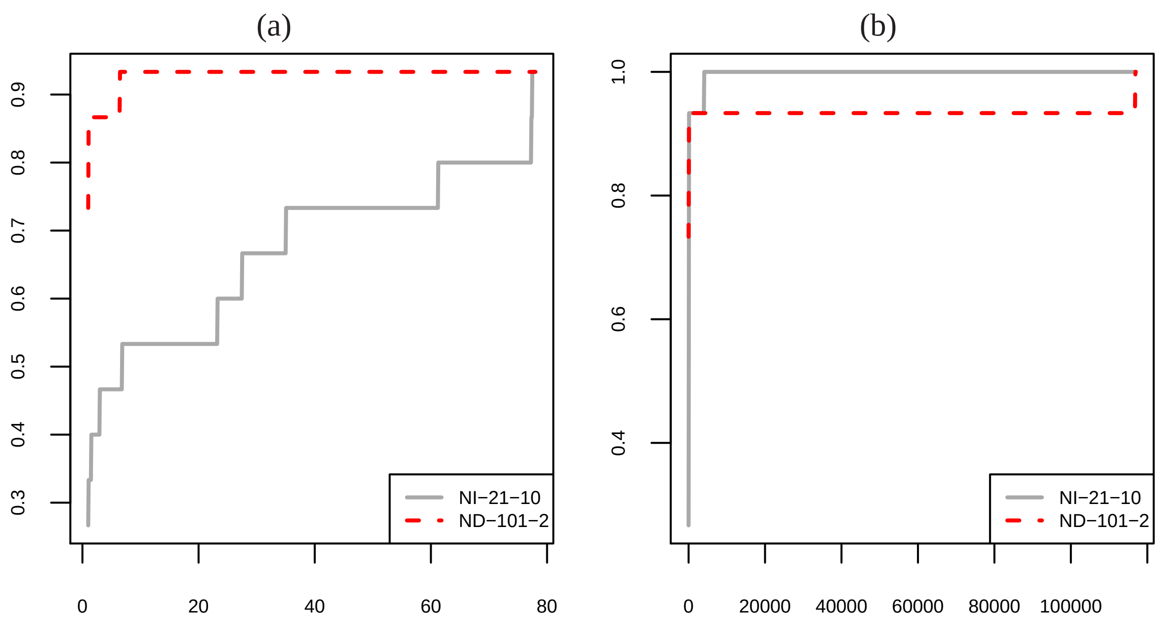

Figure 10: Performance profiles comparing the variants $NI-21-10$ and $ND-101-2$. The errors are evaluated using the five-point numerical derivative approximation with 101 observed values. (a) presents the curves with $\tau \in [1; 78]$ while in (b) $\tau \in [1; 1.17e + 5]$.

REFERENCES

Amarteifio S. and O'Neill M. Coevolving antibodies with a rich representation of grammatical evolution. In D. Corne, Z. Michalewicz, M. Dorigo, G. Eiben, D. Fogel, C. Fonseca, G. Greenwood, T.K. Chen, G. Raidl, A. Zalzala, S. Lucas, B. Paechter, J. Willies, J.J.M. Guervos, E. Eberbach, B. McKay, A. Channon, A. Tiwari, L.G. Volkert, D. Ashlock, and M. Schoenauer, editors, *Proceedings of the 2005 IEEE Congress on Evolutionary Computation*, volume 1, pages 904–911. IEEE Press, Edinburgh, UK, 2005. ISBN 0-7803-9363-5.

Barbosa H.J.C., Bernardino H.S., and Barreto A.M.S. Using performance profiles to analyze the results of the 2006 CEC constrained optimization competition. In *IEEE World Congress on Computational Intelligence*. Barcelona, Spain, 2010.

Bernardino H.S. and Barbosa H.J. *Nature-Inspired Algorithms for Optimisation*, chapter Artificial Immune Systems for Optimization, pages 389–411. Springer Berlin / Heidelberg, 2009a.

Bernardino H.S. and Barbosa H.J.C. Grammar-based immune programming for symbolic regression. In *Proceedings of the International Conference on Artificial Immune Systems - ICARIS'09*, Lecture Notes in Computer Science, pages 274–287. Springer, 2009b.

Bernardino H.S. and Barbosa H.J.C. Grammar-based immune programming (accepted). *Natural Computing*, 2010.

Chomsky N. *Syntactic Structures*. Mouton de Gruyter, 2002.

Ciccazzo A., Conca P., Nicosia G., and Stracquadanio G. An advanced clonal selection algorithm with ad-hoc network-based hypermutation operators for synthesis of topology and sizing of analog electrical circuits. In P.J. Bentley, D. Lee, and S. Jung, editors, *Proceedings of the International Conference on Artificial Immune Systems - ICARIS 2008*, volume 5132 of *LNCS*, pages 60–70. Springer, 2008.

Cramer N.L. A representation for the adaptive generation of simple sequential programs. In

Proceedings of the 1st International Conference on Genetic Algorithms, pages 183–187. L. Erlbaum Associates Inc., Hillsdale, NJ, USA, 1985.

Cutello V., Narzisi G., Nicosia G., and Pavone M. Clonal selection algorithms: A comparative case study using effective mutation potentials. In *Proceedings of the International Conference on Artificial Immune Systems - ICARIS 2005*, volume 3627 of *LNCS*, pages 13–28. Springer, 2005.

Dasgupta D. and Nino F. *Immunological Computation: Theory and Applications*. Auerbach Publications, Boston, MA, USA, 2008.

de Castro L.N. and Timmis J. An artificial immune network for multimodal function optimization. In *Proc. of the 2002 IEEE World Congress on Computational Intelligence*, volume I, pages 669–674. Honolulu, Hawaii, USA, 2002.

de Castro L.N. and von Zuben F.J. Learning and optimization using the clonal selection principle. *IEEE Trans. Evo. Comp.*, 6(3):239–251, 2002.

Dolan E. and Moré J.J. Benchmarking optimization software with performance profiles. *Math. Programming*, 91(2):201–213, 2002.

Elseth G.D. and Baumgardner K.D. *Principles of Modern Genetics*. Brooks Cole, 1995.

Ferreira C. Gene Expression Programming: a New Adaptive Algorithm for Solving Problems. *ArXiv Computer Science e-prints*, 2001.

Gan Z., Chow T.W., and Chau W. Clone selection programming and its application to symbolic regression. *Expert Systems with Applications*, 36(2):3996–4005, 2009a.

Gan Z., Zhao M.B., and Chow T.W. Induction machine fault detection using clone selection programming. *Expert Systems with Applications*, 36(4):8000 – 8012, 2009b.

Iba H. Inference of differential equation models by genetic programming. *Information Sciences*, 178(23):4453 – 4468, 2008.

Ince E.L. *Ordinary Differential Equations*. Dover Publications, 1956.

Johnson C.G. Artificial immune system programming for symbolic regression. In *Proceedings of the 6th European Conference on Genetic Programming - EuroGP 2003*, pages 345–353. 2003.

Koza J.R. *Genetic Programming: On the Programming of Computers by Means of Natural Selection (Complex Adaptive Systems)*. The MIT Press, 1992.

Koza J.R., Bennett III F.H., Andre D., and Keane M.A. Synthesis of topology and sizing of analog electrical circuits by means of genetic programming. *Computer Methods in Applied Mechanics and Engineering*, 186(2-4):459–482, 2000.

Lau A. and Musilek P. Immune programming models of cryptosporidium parvum inactivation by ozone and chlorine dioxide. *Information Sciences*, 179(10):1469–1482, 2009.

McKinney B. and Tian D. Grammatical immune system evolution for reverse engineering nonlinear dynamic bayesian models. *Cancer Informatics*, 6:433–447, 2008.

Musilek P., Lau A., Reformat M., and Wyard-Scott L. Immune programming. *Information Sciences*, 176(8):972–1002, 2006.

O'Neill M. and Brabazon A. Grammatical differential evolution. In *Proceedings of the 2006 International Conference on Artificial Intelligence - ICAI 2006*, pages 231–236. CSREA Press, Las Vegas, Nevada, USA, 2006.

O'Neill M., Brabazon A., and Adley C. The automatic generation of programs for classification problems with grammatical swarm. In *Congress on Evolutionary Computation - CEC 2004*, volume 1, pages 104–110. 2004.

O'Neill M. and Ryan C. Grammatical evolution. *IEEE Transactions on Evolutionary Computation*, 5(4):349–358, 2001.

Copyright © 2010 Asociación Argentina de Mecánica Computacional http://www.amcaonline.org.ar

O'Neill M. and Ryan C. *Grammatical Evolution: Evolutionary Automatic Programming in an Arbitrary Language*. Kluwer Academic Publishers, 2003.

Poli R., Langdon W.B., and McPhee N.F. *A Field Guide to Genetic Programming*. 2008.

Ryan C., Collins J., and Neill M.O. Grammatical evolution: Evolving programs for an arbitrary language. In *LNCS 1391, Proceedings of the First European Workshop on Genetic Programming*, pages 83–95. Springer-Verlag, 1998.

Schmidt M.D. and Lipson H. Data-mining dynamical systems: Automated symbolic system identification for exploratory analysis. In *Proceedings of the Biennial ASME Conference on Engineering Systems Design and Analysis - ESDA08*. Haifa, Israel, 2008.

Smith S.F. *A learning system based on genetic adaptive algorithms*. Ph.D. thesis, University of Pittsburgh, Pittsburgh, PA, USA, 1980.

Smith S.F. Flexible learning of problem solving heuristics through adaptive search. In *IJCAI'83: Proceedings of the Eighth international joint conference on Artificial intelligence*, pages 422–425. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1983.

Copyright © 2010 Asociación Argentina de Mecánica Computacional http://www.amcaonline.org.ar