

# Identifying Nonlinear Model Structures Using Genetic Programming Techniques \*

Stephan Winkler, Michael Affenzeller, Stefan Wagner

Institute of Systems Theory and Simulation

Johannes Kepler University

Altenbergerstrasse 69

A-4040 Linz, Austria

email: stephanwinkler@gmx.at, {ma,sw}@cast.uni-linz.ac.at

## Abstract

This paper points out how combined Genetic Programming techniques can be applied to the identification of nonlinear structures in experimental data. In many cases it is essential to generate an algebraic expression as a part of an equation that describes the physical representation of a dynamic system. Genetic Programming is an optimization procedure that can be used to identify such nonlinear structures, and evolution strategies as a second step can be applied to the tuning of constant parameters and delays.

In this paper we present a new approach for identifying nonlinear models of mechatronic systems in which the Evolutionary Computation concepts of both Genetic Programming and Evolution Strategies are used.

## 1 Introduction

This paper depicts the research done for the project "Specification, Design and Implementation of a Genetic Programming Approach for Identifying Nonlinear Models of Mechatronic Systems" which is a part of a bigger strategical project led by Prof. Del Re at the Institute of Automatic Control and Electrical Drives, Johannes Kepler University Linz, Austria.

The goal of the project is to find models for mechatronic systems. It has to be examined, whether the methods of Genetic Programming (GP) (see for instance [Koza, 1992]) are suitable for determining an 'optimal' configuration of operations and their logical connections (determining the structure) or not.

The methods of Genetic Programming, based on the theory of genetic algorithms, introduce new kinds of encoding so that an individual of a population can be interpreted as a structure, as a formula or even more generally as a program.

It is always a challenge for the designers of Genetic Algorithms to find proper crossover operators for a

given problem; this challenge is even bigger when it comes to Genetic Programming because the crossing of the genetic make-up of two individuals should create a new valid solution candidate consisting of the genes of its parents. In the context of identifying structures this means combining substructures, components of formulas respectively parts of programs without violating constraints. Thus, it is very important to design carefully the encoding of the problems and the operators belonging to them.

The following tasks have to be carried out:

- Specification of the problem and how it can be encoded; design of the appropriate operators for crossover and mutation.
- Implementation: The designed GP model has to be implemented as part of the already existing 'HeuristicLab', a framework for prototyping and analyzing optimization techniques (developed by Stefan Wagner and Michael Affenzeller) for which both generic concepts of Evolutionary Algorithms and many functions to evaluate and analyze them are available. The programming language chosen for this project (and the 'HeuristicLab') is C# using the Microsoft .NET Framework 1.1.
- Evaluation: Concrete training- and testing data are available to interpret the quality of the generated solutions. In addition to that, new generic concepts, based on Evolutionary Algorithms and developed to increase the quality of the produced solutions, should be used and compared to the classical GP approach. Especially the self-adaptive steering of selection-pressure [Affenzeller and Wagner, 2003] should produce a massive increase of the quality of the solutions found by the program because it has been designed to have a stabilizing effect with regard to critical results of crossovers. The introduction of a second optimization stage, using Evolution Strategies (ES) to optimize the parameters of solution candidates, should be implemented as well and will also be presented later in this paper.

The identification of nonlinear physical models is a quite difficult task since both the structure and the parameters of the physical model must be determined. In contrast to many existing system identification

---

\*This work has been performed under the research grant 4.3 of the LCM Center of Competence in Mechatronics in Linz, which is sponsored by means of the federal Austrian budget and the province of Upper Austria.

methods based on parametric identification [Gray *et al.*, 1998], structure determination often uses a trial and error approach to test candidate model structures. Possible structures are deduced from engineering knowledge of the system and the parameters of these models are estimated; the resulting model performance is compared with experimental data from available measured quantities or with other model structures.

Table 1 shows a part of the test data provided by members of the Institute of Automatic Control and Electrical Drives, JKU Linz.

$t$	$x$	$p1$	$p2$	$pk$
2.150	546.417	4.1875	3.3232	5.8375
2.151	546.875	4.1838	3.3269	5.8430
2.152	546.997	4.1911	3.3342	5.8414
2.153	547.088	4.1838	3.3452	5.8496
2.154	547.393	4.1948	3.3378	5.8567
2.155	547.760	4.1801	3.3159	5.8288
2.156	548.126	4.1691	3.3122	5.8329
2.157	548.431	4.1728	3.3085	5.8363
2.158	548.645	4.1801	3.3195	5.8320
2.159	549.011	4.1728	3.3122	5.8352
2.160	549.011	4.1728	3.3159	5.8340
2.161	549.652	4.1875	3.3269	5.8420
2.162	549.804	4.1838	3.3269	5.8361
2.163	550.079	4.1948	3.3269	5.8411
2.164	550.262	4.1984	3.3452	5.8496

Table 1: Exemplary test data

The data partially shown in Table 1 are the results of experiments with a pneumatic positioning system. In the context of identifying mathematical structures we are not interested in the way these data have been obtained or what they mean at all; for us it is just important to know which variables are potential input (in this case the columns  $p1$ ,  $p2$  and  $pk$  representing certain pressure values) and which ones potential output (in this case the column  $x$  representing the way of some cylinder).

This paper presents a method which creates new formula structures using methods of genetic Genetic Programming and Genetic Algorithms, and also tries to optimize the parameter settings of each candidate model structure by using the concept of evolution strategies.

## 2 Genetic Programming, Genetic Algorithms and Evolution Strategies

Genetic Programming works by imitating aspects of natural evolution to generate a solution that maximizes (or minimizes) some fitness function [Koza, 1992]. A population of solution candidates evolves through many generations towards a solution using certain evolutionary operators and a 'survival-of-the-fittest' selection scheme.

Typically the population of a Genetic Programming process contains a few hundred individuals and evolves through the action of operators known as crossover, mutation and selection. The initial population is typically created at random.

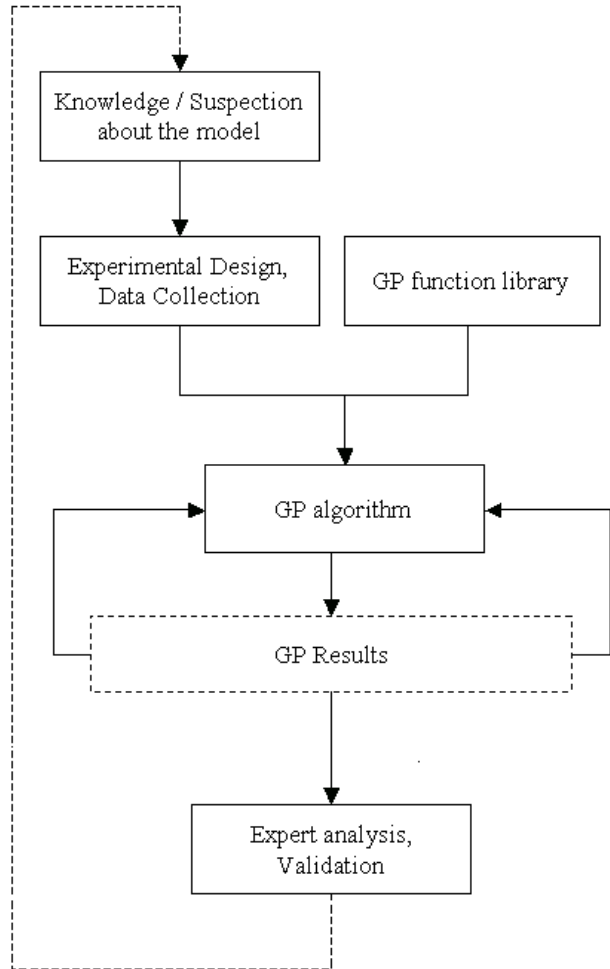


Figure 1: The Genetic Programming Process

Selection involves evaluating the fitness of each population member and choosing the fittest to continue to the next generation; there are various selection strategies which can be used to determine which of the population members will survive to the next generation [Koza, 1992]. The fitness function does not have to be differentiable or even continuous; crossover, mutation and selection improve the general fitness of the population from one generation to the next and the algorithm repeats through many generations until some convergence criterion is satisfied.

Elitism can also be applied during the selection phase. Elitism ensures that the best member of each generation will be copied into the succeeding generation (as mentioned for instance in [Adamopoulos *et al.*, 1997]). Thus, by increasing the speed of domination of a super individual, the elitist strategy appears to improve GA performance [Davis, 1991].

In the context of identifying nonlinear model struc-

tures the algorithm should create a near-optimal model that can then be used for nonlinear simulation of the system or for further investigation of the physical system, or to validate the structure of an existing model developed in some other way [Gray *et al.*, 1998]. As summarized e.g. in [Affenzeller, 2003], there are two basic heuristic optimization approaches in computer science that copy evolutionary mechanisms: Evolution Strategies (ES) [Rechenberg, 1973; Schwefel, 1994] and Genetic Algorithms (GA) [Holland, 1975]. The major difference between these two concepts of Evolutionary Computation lies in the form of the genotype and in the operators (mutation, recombination and selection). While mutation is the primary operator of Evolution Strategies, it is only used to avoid stagnation in genetic algorithms. Furthermore selection in case of ES is absolutely deterministic which is not the case in the context of GA or in nature.

### 3 Genetic Programming Based Method for Modelling Nonlinear Structures

#### 3.1 GP Representation of an Algebraic Expression

As already mentioned, Genetic Programming can be used to evolve algebraic expressions from a database representing the measured results of experiments that are to be analyzed. It allows optimization of tree structure representations of symbolic expressions. These tree representations consist of nodes and are of variable length. The nodes can either be nonterminal signifying functions performing some action on one or more signals within the structure to produce an output signal, or terminal representing an input variable or a constant. An exemplary tree structure can be seen in Figure 2 representing the algebraic expression  $5.0/x_1(t-1) + \ln(x_2(t-2))$ . In the case of this system there is one output and the terminal nodes are the constant 5.0 and the system inputs  $x_1(t-1)$  and  $x_2(t-2)$ .

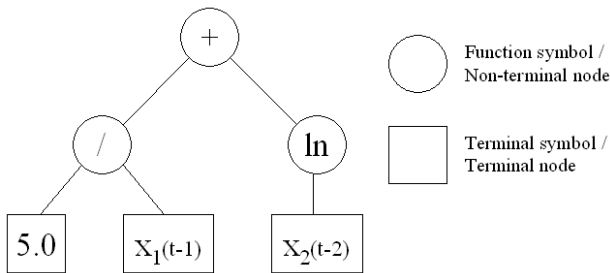


Figure 2: An example structure

The nonterminal nodes have to be selected from a library of possible functions, a pool of potential nonlinear model structural components; the selection of the library functions is an important part of the GP modelling process [Gray *et al.*, 1998] because

this library should be able to represent a wide range of systems. The trees are built by combining nodes according to grammar rules defining the number of inputs for each node type. Any prior knowledge of the physical system should be included in an initial model and in the function library.

When the genetic algorithm is executed, each individual of the population represents one structure tree; usually the structures are limited by a predefined maximum tree size.

#### 3.2 Crossover Operator

Since the trees have to be usable by the Genetic Algorithm, mutation and crossover operators for the tree structures have to be designed. Both crossover and mutation processes are applied to randomly chosen branches (usually a branch is the part of a structure lying below a given point in the tree). Crossing two trees means randomly choosing a branch in each parent tree and replacing the branch of the tree, that will serve as the root of the new child (randomly chosen, too), by the branch of the other tree. Examples of this procedure are shown in Figure 3: The formulas  $5.0/x_1(t-1) + \ln(x_2(t-2))$  and  $x_3(t-2) * x_2(t-2) - 1.5$  can for example be recombined to  $5.0/(x_1(t-1)) + x_3(t-2) * (x_2(t-2))$  or  $5.0/(x_1(t-1)) - 1.5$ .

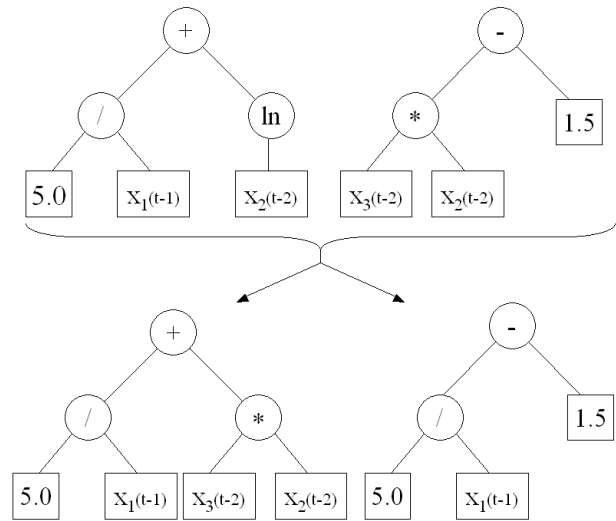


Figure 3: Example GP structure recombinations

This crossover procedure ensures that characteristics of both parents survive to the next generation but are combined in a different way, sometimes producing offspring that yield to a better fitness value than either parent.

#### 3.3 Mutation Operator

As already mentioned, mutation in the context of genetic algorithms means modifying a solution candidate randomly and so creating a new individual. In

the case of identifying structures, mutation works by choosing a node and changing it: A function symbol could become another function symbol or be deleted, the value of a constant node or the time offset of a variable could be modified. This procedure is less likely to improve a specific structure but it can help the optimization algorithm to reintroduce genetic diversity in order to restimulate genetic search. Examples for this procedure are shown in Figure 4: Mutating the formula  $5.0/x_1(t-1) + \ln(x_2(t-2))$  could yield to  $5.0 + x_1(t-1) + \ln(x_2(t-2))$  or  $5.0/x_1(t-1) + x_2(t-2)$ .

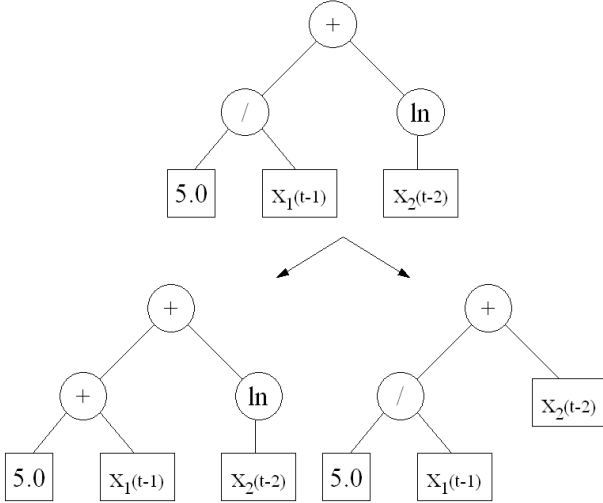


Figure 4: Example GP structure mutations of the structure shown in Figure 2

### 3.4 Evaluation Operator

Every solution candidate has to be evaluated because the GP algorithm maximizes or minimizes some objective function. In the context of structure identification this function should be an appropriate measure of the level of agreement between the model and system response. One example is the sum of squared error function

$$J = \sum_{i=1}^N e_i^2$$

where  $e_i$  is the error between experimental data and structure output for each of  $N$  data points. Of course many other fitness functions could be used instead. Better model structures evolve as the GP algorithm minimizes the fitness function.

Another possible fitness function could be

$$J = 5000 * \left( 8 - \log_{10} \left( \sum_{i=1}^N e_i^2 \right) \right)$$

This function was presented in [Gray *et al.*, 1998]; it was successfully used to calculate the fitness of solution candidates for identifying a coupled water tank system using Genetic Programming. New variants of Genetic Algorithms are going to

be used, e.g. those mentioned in [Affenzeller and Wagner, 2003] (algorithms that should be able to produce quite good crossover results using crossover operators that do not have to be appropriate for the specific problem by applying a higher selection pressure).

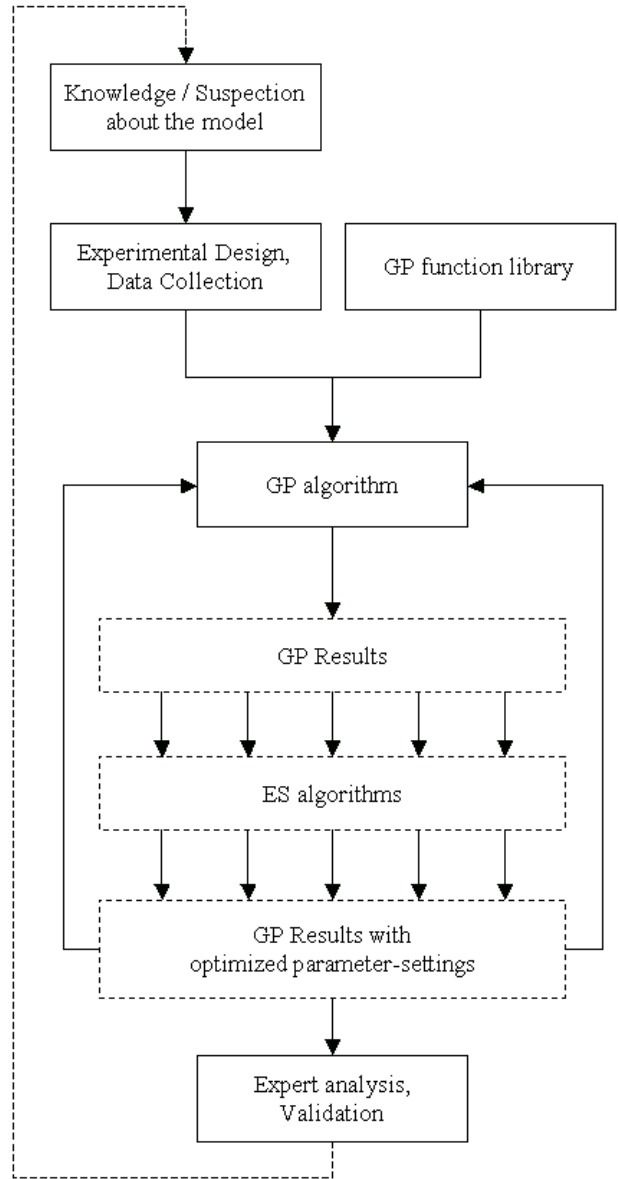


Figure 5: Extended Genetic Programming Process including a second optimization step

## 4 Introducing a Second Optimization Step

Analyzing a structure might show the problem that the structure of a formula can be very useful for solving the problem, but still produces a bad fitness value just because of bad settings of system inputs (constants, variables or time offsets). So we decided that it would be useful to optimize the parameter of every

new solution candidate created (either by initialization, mutation or recombination) before it is put into the population of the GA.

This means that every time a initialization, crossover or mutation procedure is finished, a new Evolution Strategy algorithm is started to optimize the parameter settings of the new model structure. When the ES algorithm terminates, the structure is given back to the GA. Figure 5 illustrates this combined concept. The introduction of this second optimization phase means that in every generation of the GA many new ES algorithms are started, and that the GA has to wait until all ES algorithms have finished their calculations. Obviously this consumes a lot of runtime. To cope with this, we have designed the automatic generation of ES problems so that they can be processed completely separately from each other in an distributed computing environment.

## 5 Distributed GP

Since the ES processes, that optimize the new individuals of the GA before they are inserted into the population, can be executed separately, we are planning to use a grid architecture for Genetic Programming problems developed by Stefan Wagner and Michael Affenzeller.

The idea of this grid architecture is that optimization problems are inserted into a database and assigned to participating computers. In the context of identifying structures using two optimization steps (as described in the previous chapter) this means that all solution candidates generated by the GA can be inserted into the database as problems that have to be solved. So all the additional optimization problems of the second optimization phase can be executed in parallel. When those second step problems are executed and their results written back into the database, the GA can fetch those results (the improved solution candidates) and go on generating its next generation.

## 6 Open Issues

### 6.1 Distribute the Workload Using a Database

Currently we are busy writing procedures to insert the problems of the second optimization phase into the database automatically. Furthermore those problems of the second optimization phase have to be fetched by the grid clients and the results have to be calculated and written back into the database automatically, too. One of the main difficulties in this context is that the grid clients are not programmed to do anything except to fetch an unsolved problem, start a new algorithm to solve the problem, execute the algorithm and finally write the value of the best solution back into the database. So we will have to implement a new Evolution Strategy algorithm that is able to perform some actions that are only needed in the context of identifying nonlinear structures. This extended ES algorithm will have to:

- fetch the string representation of a formula from the database plus the corresponding timeseries (the data of the original optimization problem),
- optimize the solution like a 'normal' ES algorithm
- and finally save the optimized string representation of the formula in the database so the GA can fetch it from there and go on with its calculations.

### 6.2 Using the Execution-Time Code Generation Concept

Another possibility to speed up the calculation of the fitness values of the solution candidates is using the Execution-Time Code Generation concept. In the .NET Common Language Runtime it is possible to load an assembly from disk and to create instances of classes from that assembly. (See for instance [Gunnerson, 2001].) The code of such an assembly can also be generated and manipulated at runtime; so we can generate the code to evaluate a structure for given data at runtime. That may help us save a lot of calculation time.

### 6.3 Tuning

Moreover we will surely have to optimize the general settings of the problems such as the range for constants and time offsets or the maximum height of the structure trees. The evaluation of generated formulas will have to be optimized as well (the sum of square errors is a fine, but maybe not the best evaluation function for such problems). The operators for mutation and crossover will have to be improved as well; finally the function library will be checked whether the selected functions are suitable for identifying mathematical structures in mechatronical systems or not.

## 7 Summary and Outlook

As already mentioned the goal of this project is to investigate whether the application of Genetic Programming techniques can be used to generate algebraic expressions as parts of equations that describe the physical representations of dynamic systems. Extracting structural characteristics is quite a difficult task, especially in the case of complex and nonlinear systems. Of course it can happen that the final result of this investigation will be that this approach is not suitable because it takes too much time to generate a solution or an appropriate solution cannot be found at all.

We are hoping to be able to present the results of our project by spring 2004.

## References

- [Adamopoulos *et al.*, 1997] A. V. Adamopoulos, S. D. Likothanassis, and E. F. Georgopoulos. Extracting Structural Characteristics of a Nonlinear Time-series Using Genetic Algorithms. *Proceedings of the 1997 IASTED International Conference on Intelligent Information Systems (IIS '97)*, 1997.

- [Affenzeller, 2003] Michael Affenzeller. *New Hybrid Variants of Genetic Algorithms: Theoretical and Practical Aspects*. Trauner Verlag Linz, 2003.
- [Affenzeller and Wagner, 2003] Michael Affenzeller and Stefan Wagner. A Self-Adaptive Model for Selective Pressure Handling within the Theory of Genetic Algorithms. *Computer Aided Systems Theory: EUROCAST 2003, Lecture Notes in Computer Science*, 2003.
- [Davis, 1991] L. Davis. *Handbook of Genetic Algorithms*. Van Nostrand Reinhold, 1991.
- [Gray et al., 1998] G. J. Gray, D. J. Murray-Smith, Y. Li, K. C. Sharman, and T. Weinbrenner. Nonlinear model structure identification using genetic programming. *Control Engineering Practice* 6, 1998.
- [Gunnerson, 2001] E. Gunnerson. *A Programmer's Introduction to C# (Second Edition)*. APress, 2001.
- [Holland, 1975] J. H. Holland. *Adaption in Natural and Artificial Systems*. University of Michigan Press, 1975.
- [Koza, 1992] J. Koza. *Genetic Programming: On the Programming of Computers by means of Natural Selection*. The MIT Press, Cambridge, Mass., 1992.
- [Rechenberg, 1973] I. Rechenberg. *Evolutionsstrategie*. Friedrich Frommann Verlag, 1973.
- [Schwefel, 1994] H. P. Schwefel. *Numerische Optimierung von Computer-Modellen mittels Evolutionsstrategie*. Birkhuser Verlag. Basel, 1994.