

Stephan WINKLER*, Michael AFFENZELLER*, Stefan WAGNER*

NEW METHODS FOR THE IDENTIFICATION OF NONLINEAR MODEL STRUCTURES BASED UPON GENETIC PROGRAMMING TECHNIQUES

Identifying nonlinear model structures as a part of analyzing a physical system means trying to generate an algebraic expression as a part of an equation that describes the physical representation of a dynamic system. Many existing system identification methods are based on parameter identification. In this paper we describe a method using genetic programming to evolve an algebraic representation of measured input-output response data. The main advantage of the presented approach is that unlike many other identification methods, it does not restrict the set of models that can be identified but can be applied to any kind of data sets representing a system's observed or simulated input and output signals.

This paper describes research that was done for the project "Specification, Design and Implementation of a Genetic Programming Approach for Identifying Nonlinear Models of Mechatronic Systems". The goal of the project is to find models for mechatronic systems; our task was to examine, whether the methods of Genetic Programming are suitable for determining the structures of physical systems by analyzing a system's measured behaviour or not.

1. INTRODUCTION

The problem of discovering the mathematical relationship between empirically observed variables measuring a system is an important problem in technical fields such as mechatronics, but also in economics and other areas of science (see e.g. [8]). In practice, the observed data may be noisy and there may be no known way to express the relationships involved in a precise way. Problems of this type are sometimes called symbolic system identification problems, black box problems, data mining problems, or modeling problems. When the model that is discovered is used in predicting future values of the state variables of the system, the problem is called a forecasting problem [6].

We have designed a genetic programming (GP) model including appropriate crossover and mutation operators for this problem. This GP approach, described in Chapter 3, has also

* Institute of Systems Theory and Simulation, Johannes Kepler University Linz, Austria
{winkler,ma,sw}@cast.uni-linz.ac.at

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.

been implemented as a part of the already existing “HeuristicLab”, a framework for prototyping and analyzing optimization techniques (developed by Stefan Wagner and Michael Affenzeller, described in [11]) 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. We have also tested our approach intensively. Moreover, in addition to standard GP implementations, new generic concepts, based on evolutionary algorithms and developed to increase the quality of the produced solutions, were used and compared to the classical GP approach. A summary of the results of our test series is given in Chapter 4.

2. GENETIC ALGORITHMS, GENETIC PROGRAMMING

Evolutionary computing is the collective name for heuristic problem-solving techniques based on the principles of biological evolution, which are natural selection and genetic inheritance. One of the greatest advantages of these techniques is that they can be applied to a variety of problems, ranging from leading-edge scientific research to practical applications in industry and commerce; by now, evolutionary algorithms are in use in various disciplines like optimization, artificial intelligence, machine learning, simulation of economic processes, computer games or even sociology.

The forms of evolutionary computation, that are relevant for the work depicted in this paper, are the genetic algorithm (GA) and genetic programming (GP). The fundamental principles of GAs were first presented by Holland ([4]), overviews about GAs and their implementation in various fields were for example given by Goldberg [2] and Michalewicz [9]. A GA works with a set of candidate solutions (also known as individuals) called population. During the execution of the algorithm each individual has to be evaluated, which means that a value indicating the "fitness" or "goodness" is returned by a fitness function. New individuals are created on the one hand by combining the genetic make-up of two solution candidates (this procedure is called "crossover" or "recombination"), producing a new "child" out of two "parents", and on the other hand by mutating some individuals, which means that randomly chosen parts of genetic information are changed (normally 3-5% of the algorithm's population is mutated in each generation). Beside crossover and mutation, the third decisive aspect of genetic algorithms is selection, a mechanism in analogy to biology also called "survival of the fittest". As already mentioned, each individual is associated with a fitness value. The individual's probability to inherit its genetic information to the next generation is proportional to its fitness; the better a solution candidate's fitness value, the higher the probability, that its genetic information will be included in the next generation's population. This procedure of crossover, mutation and selection is repeated many times (over many generations) until some termination criterion is fulfilled.

Genetic programming was first explored in depth in 1992 by John R. Koza, a computer scientist at Stanford University, CA, USA. In his famous book "Genetic Programming: On the Programming of Computers by Means of Natural Selection" [5] he pointed out that virtually all problems in artificial intelligence, machine learning, adaptive systems, and

automated learning can be recast as a search for a computer program, and that genetic programming provides a way to successfully conduct the search for a computer program in the space of computer programs.

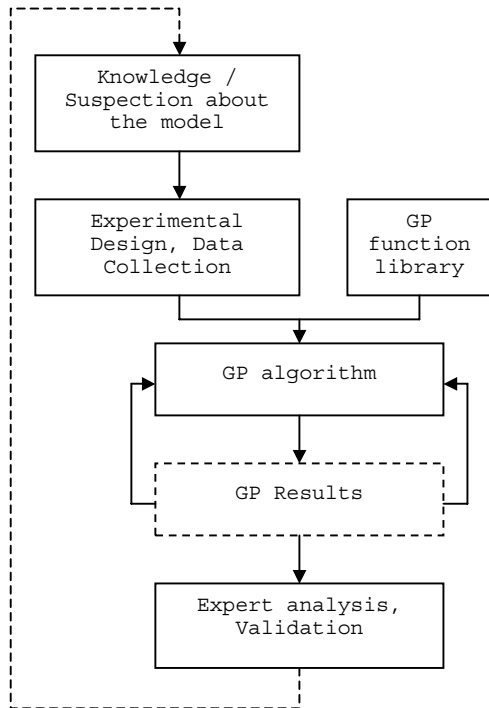


Fig. 1. The GP process

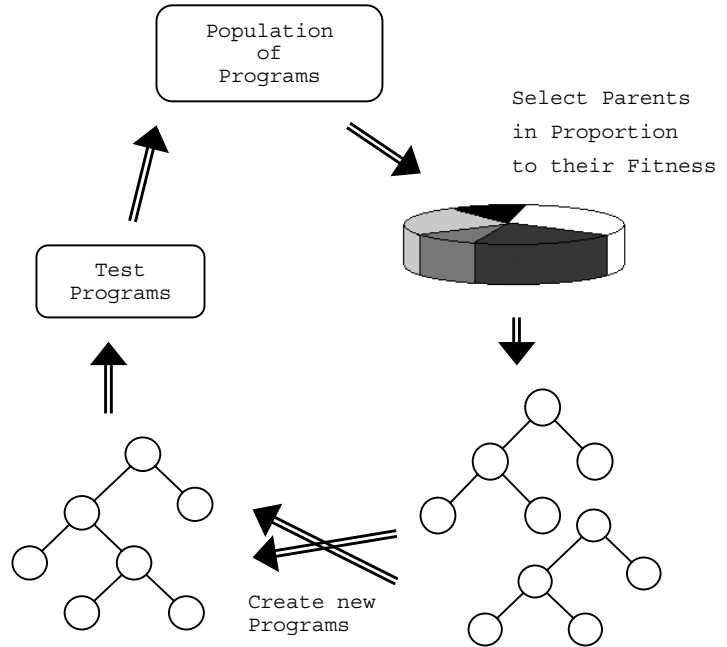


Fig. 2. The GP cycle [7]

Similar to GAs, GP works by imitating aspects of natural evolution to generate a solution that maximizes (or minimizes) some fitness function [5]. A population of solution candidates evolves through many generations towards a solution using certain evolutionary operators and a 'survival-of-the-fittest' selection scheme. The main difference is, that whereas GAs are intended to find an array of characters respectively integers that represents the solution of a given problem, the goal of a GP process is to produce a computer program (or, as in our case, a formula) that is the solution of the optimization problem at hand. Typically the population of a GP algorithm contains a few hundred individuals and evolves through the action of operators known as crossover, mutation and selection. A flowchart of the GP process is displayed in Fig. 1, and Fig. 2 shows, how the GP cycle works: Like in every evolutionary process, new individuals (in GP's case, new programs) are created. They are tested, and the fitter ones in the population succeed in creating children of their own. Unfit ones die and are removed from the population [7].

3. A GP BASED APPROACH FOR MODELING NONLINEAR STRUCTURES

As already mentioned, the goal of a GP system identification process is to produce an algebraic expression from a database storing the measured results of experiments that are to

be analyzed. Thus, in the GP approach we have designed and implemented for this project, the GP algorithm works with solution candidates that are 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. The approach described here was first explained in [12].

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 modeling process [3] 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.

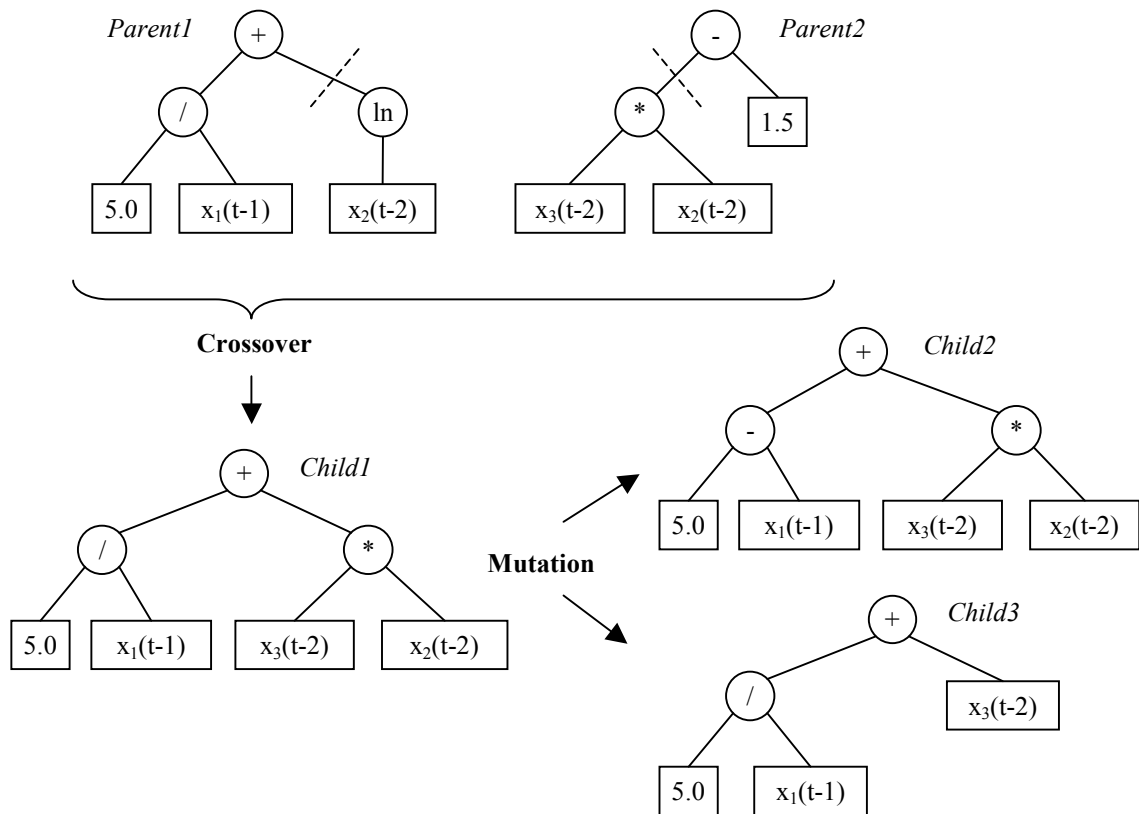


Fig. 3. Example tree structures and genetic operations. The crossover of parent1 and parent2 yields child1, child2 and child3 are possible mutants of child1.

Since the tree structures 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 (in this context 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. 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.

Since the GP algorithm maximizes or minimizes some objective fitness function (better model structures evolve as the GP algorithm minimizes the fitness function), every solution candidate has to be evaluated. 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

$$J = \sum_{i=0}^N e_i^2. \quad (1)$$

This function is very widely used and was applied in most of the test series we have carried out for this project (also in those mentioned exemplarily in the next chapter). Still, there are many other fitness functions that could be used instead and have also been tested as possible fitness functions for GP system identification processes.

4. EXPERIMENTAL TEST RESULTS

Empirical studies with different problem instances are the most effective way to analyze the potential of heuristic optimization searches like evolutionary algorithms. The results presented in this section were achieved using Pentium 4 PCs under Windows XP. The programs are, as already mentioned, written in the C# programming language using the Microsoft .NET framework 1.1.

In addition to implementations of the standard genetic algorithm (SGA), we have also intensively tested a new hybrid algorithm that was developed at the Institute of Systems Theory and Simulation, Johannes Kepler University, Linz. This algorithm, the so-called SASEGASA (Self Adaptive SEgregative Genetic Algorithm with Simulated Annealing aspects), is a new generic Evolutionary Algorithm for retarding the unwanted effects of premature convergence by combining concepts of GAs, Evolution Strategies [10] and Simulated Annealing. As summarized in [1], this algorithm introduces a new generic selection method for Genetic Algorithms. The main difference of this selection principle in contrast to conventional selection models is that it considers not only the fitness of an individual compared to the fitness of the total population in order to determine the possibility of being selected. Additionally, in a second selection step, the fitness of an offspring is compared to the fitness of its own parents. Thus, the evolutionary process is continued mainly with offspring that have been created by advantageous combination of

their parents' attributes. A self-adaptive feature of this approach is realized in that way that it depends on the actual stadium of the evolutionary process how many individuals have to be created in order to produce a sufficient amount of "successful" offspring.

In the following we will show the test results obtained using SGA and SASEGASA for identifying 2 example problem instances, *INS1* and *INS2*. In both cases the system's output data was distorted (the noise is approximately 5% of each systems' output signal). *INS1* was simulated using the following formula:

$$Y_{(t)} = X4_{(t-1)} * X3_{(t-2)} + \frac{X1_{(t-1)}}{X2_{(t-1)}} \quad (2)$$

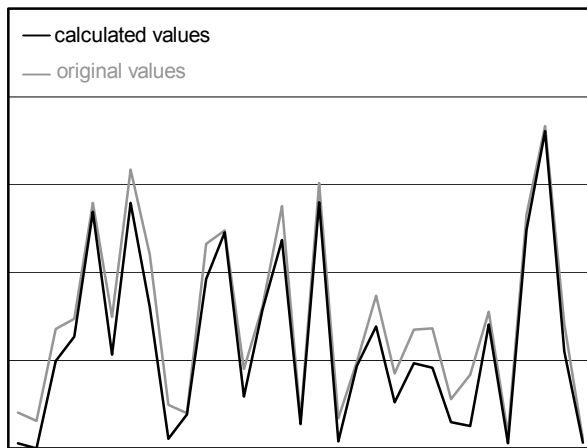


Fig. 4. Representation (30 sample points) of the result for *INS1*, obtained by using the SGA.

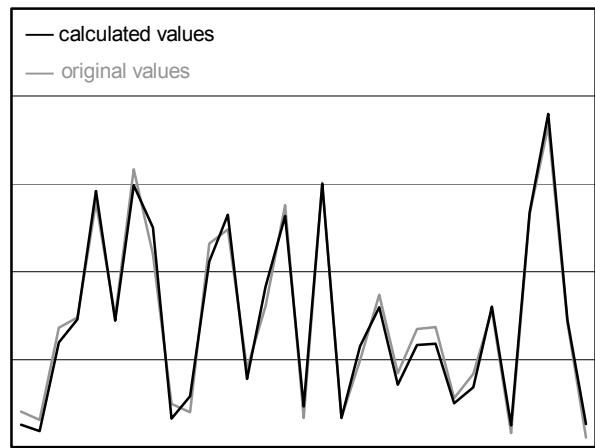


Fig. 5. Representation (30 sample points) of the result for *INS1*, obtained by using the SASEGASA.

<i>Algorithm</i>	SGA	SASEGASA
<i>Population size</i>	1000	100
<i>Generations</i>	2000	30
<i>Mutation rate</i>	5%	5%
<i>Computing time</i>	1 h 7 min	37 sec
<i>Mean square error</i>	739.16	8.79

Tab. 1. Parameter settings and solution quality of the GP algorithms that yielded the results shown in Fig. 4 respectively Fig. 5.

In this case, the SGA was not able to identify the formula absolutely correctly (only the multiplication $X4 * X3$ was found). The SASEGASA, however, found exactly the formula that had been used to generate the output signal in not even a minute (the reason why the error is not 0 is just that the data was distorted).

INS2 was simulated using a formula we do not know. Since in such a case one can not simply compare the algorithm's result to the formula that generated the simulated output values, we have restricted the data set available for the GP algorithm and checked, whether

the forecast values produced using the formula match the original ones. The idea behind this was, that a good approximation of the data at hand does not guarantee good prediction; only if the structure of a system is really identified, a forecast of its behaviour is possible.

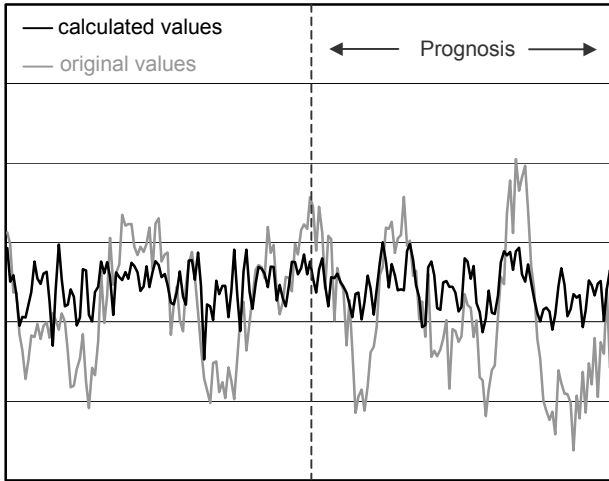


Fig. 6. Representation (200 sample points) of the result for INS2, obtained by using the SGA

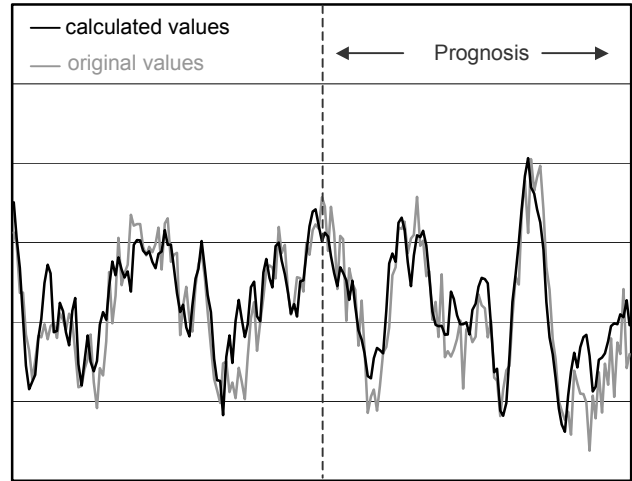


Fig.7. Representation (200 sample points) of the result for INS2, obtained by using the SASEGASA

<i>Algorithm</i>	SGA	SASEGASA
<i>Population size</i>	2000	1000
<i>Generations</i>	2000	27
<i>Mutation rate</i>	5%	5%
<i>Computing time</i>	7 h 35 min	3 h 10 min
<i>Mean square error</i>	179.23	57.04

Tab. 2. Parameter settings and solution quality of the GP algorithms that yielded the results shown in Fig. 6 respectively Fig. 7.

As one can exemplarily see in the Fig. 6 and Fig. 7, the SASEGASA obviously was able to find quite a good approximation of the formula that generated the output values of *INS2*, whereas the tests using the SGA were not successful at all. In fact, this phenomenon did not only occur for these two examples, but for almost all of our problem instances. Whereas system identification using GP does not work very good with the SGA, the SASEGASA can successfully be applied to do so, even if the data is rather noisy. Eminently good results were obtained using the following SASEGASA specific settings: One subpopulation, a very high maximum selection pressure (e.g., 300 or 500) and the selection operators Random and Roulette. Especially the use of these two selection methods has proved itself valuable, most probably because it has the effect that for each crossover procedure on the one hand a rather good formula and another, randomly chosen one are combined to produce a new formula tree structure.

5. CONCLUSION, OUTLOOK

In this paper an approach for identifying model structures of nonlinear systems by using genetic programming has been presented plus some examples that show, how this procedure works using on the one hand the standard genetic algorithm and on the other hand new, enhanced GA variants. By presenting a method that can obviously identify simulated, nonlinear systems, we have shown that it is definitively possible to use GP techniques for identifying models of mechatronical systems. Right now, we are busy with analyzing data collections of real-world industrial systems (amongst others a motor engine and a pneumatic positioning system), improving the genetic operators and parameter settings and finally working together with experts, who know more about the systems we are analyzing, in order to find out whether the obtained results respectively formulae are suitable or not.

REFERENCES

- [1] AFFENZELLER M. and WAGNER S., *SASEGASA: A New Generic Parallel Evolutionary Algorithm for Achieving Highest Quality Results*, Journal of Heuristics, Special Issue on New Advances on Parallel Meta-Heuristics for Complex Problems, Vol. 10 (2004), Kluwer Academic Publishers, pp. 239-263.
- [2] GOLDBERG D., *Genetic Algorithms in Search, Optimization and Machine Learning*, Addison Wesley Longman, 1989.
- [3] GRAY G. et al., *Nonlinear Model Structure Identification Using Genetic Programming*, Control Engineering Practice 6, 1998.
- [4] HOLLAND J., *Adaption in Natural and Artificial Systems*, University of Michigan Press, 1975.
- [5] KOZA J., *Genetic Programming: On the Programming of Computers by means of Natural Selection*, The MIT Press, Cambridge, Mass. 1992.
- [6] KOZA J., *Genetic Programming for Econometric Modeling*, Intelligent Systems for Finance and Business (1995), pp. 251-269.
- [7] LANGDON W. and POLI R., *Foundations of Genetic Programming*, Springer Verlag, Berlin Heidelberg New York 2002.
- [8] LANGLEY P. et al., *Scientific Discovery: Computational Explorations of the Creative Process*, The MIT Press, Cambridge, Mass. 1987.
- [9] MICHALEWICZ Z., *Genetic Algorithms + Data Structures = Evolution Programs (3rd edition)*, Springer-Verlag, Berlin Heidelberg New York 1996.
- [10] SCHWEFEL H., *Numerische Optimierung von Computer-Modellen mittels Evolutionsstrategie*, Birkhäuser Verlag, Basel 1994.
- [11] WAGNER S. and AFFENZELLER M., *HeuristicLab - A Generic and Extensible Optimization Environment*, to be published in: Proceedings of IBERAMIA 2004.
- [12] WINKLER S. et al., *Identifying Nonlinear Model Structures Using Genetic Programming Techniques*, Cybernetics and Systems 2004, pp. 689-694.