
A Genetic Programming Based Tool for Supporting Bioinformatical Classification Problems

Stephan Winkler^{a,b}, Michael Affenzeller^a, Stefan Wagner^a

^aInstitute for Formal Models and Verification,
Johannes Kepler University Linz, Altenbergerstraße 69, A-4040 Linz, AUSTRIA

^bInstitute for Design and Control of Mechatronical Systems,
Johannes Kepler University Linz, Altenbergerstraße 69, A-4040 Linz, AUSTRIA

ABSTRACT

Motivation: Classification problems are one of the major topics not only in computer science in general, but also especially in bioinformatics. Whereas many existing methods require additional information about the underlying system or user interaction, we here present a fully automatic, self-adaptive and problem instance independent data based classification tool based on Genetic Programming.

Results: For testing the presented approach we have used two data sets, namely the *Wisconsin Breast Cancer* and the *Melanoma* data sets. The results achieved for these two problems instances are documented in this paper.

Availability: The HeuristicLab, the optimization framework used for the research presented here, as well as an implementation of the Standard Genetic Algorithm are available from our website www.heuristiclab.com. The classification problem plug-in as well as more sophisticated genetic algorithms are still subject to ongoing research and therefore not yet fully published; the release of the first official versions is scheduled for the oncoming months.

Contact:

{stephan, michael, stefan}@heuristiclab.com

1. INTRODUCTION

Classification is understood as the act of placing an object into a set of categories, based on the object's properties. Objects are classified

according to an (in most cases hierarchical) classification scheme also called taxonomy. Amongst many other possible applications, examples of taxonomic classification are biological classification (the act of categorizing and grouping living species of organisms), medical classification and security classification (where it is often necessary to classify objects or persons for deciding whether a problem might arise from the present situation or not). A statistical classification algorithm is supposed to take feature representations of objects and map them to a special, predefined classification label. Such classification algorithms are designed to learn (i.e. to approximate the behavior of) a function which maps a vector of object features into one of several classes; this is done by analyzing a set of input-output examples ("training samples") of the function. The basic steps of this procedure are shown in Figure 1: After training the algorithm using a set of training samples, the algorithm should be able to classify a new, unknown object into one of the predefined classes by analyzing its measured features.

Since statistical classification algorithms are supposed to "learn" such functions, we are dealing with a specific subarea of *Machine Learning* and, more generally, *Artificial Intelligence*. There are several approaches that are nowadays used for solving data mining and, more specifically, classification problems. The

most common ones are (as for example described in [Mitchell, 2000]) decision tree learning, instance-based learning, inductive logic programming (such as Prolog, e.g.) and reinforcement learning.

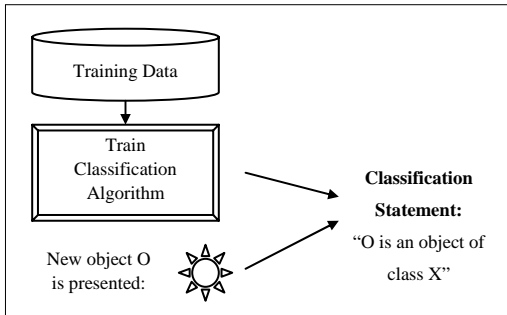


Fig. 1. The Basic Steps of Classification.

Unlike these methods, the approach we have designed is a Genetic Programming (GP) method including appropriate crossover and mutation operators for this problem. This GP approach has been implemented as a part of the “HeuristicLab”, a framework for prototyping and analyzing optimization techniques for which both generic concepts of evolutionary algorithms and many functions to evaluate and analyze them are available.

2. EVOLUTIONARY COMPUTATION: GENETIC ALGORITHMS AND 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 relevant for the work described in

this paper are *Genetic Algorithms (GAs)* and *Genetic Programming*.

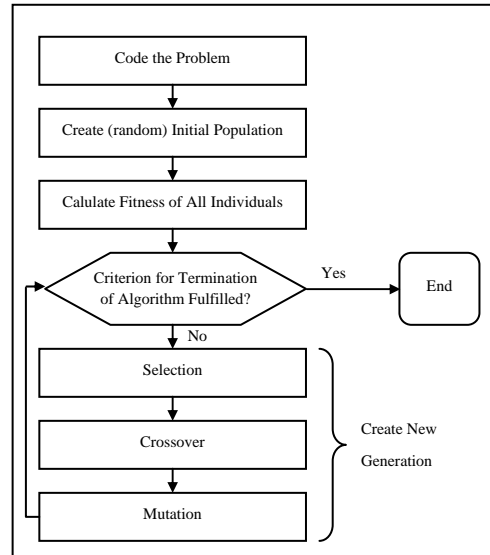


Fig. 2. Execution of a Genetic Algorithm.

The fundamental principles of the *Genetic Algorithm* were first presented by Holland [Holland, 1975], overviews about GAs and their implementation in various fields were given for instance in [Goldberg, 1989], [Michalewicz, 1996] and [Affenzeller, 2003].

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 “parent” solution candidates (this procedure is called “crossover”) producing a new “child”, and on the other hand by mutating some individuals, i.e. changing randomly chosen parts of genetic information (normally a minor ratio of the algorithm’s population is mutated in each generation).

Beside crossover and mutation, the third decisive aspect of genetic algorithms is selection. In analogy to biology this is a mechanism also called “survival of the fittest”. Usually, the

individual's probability to propagate 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 over many generations until some termination criterion is fulfilled.

Genetic programming was first explored in depth in 1992 by John R. Koza who 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 [Koza, 1992].

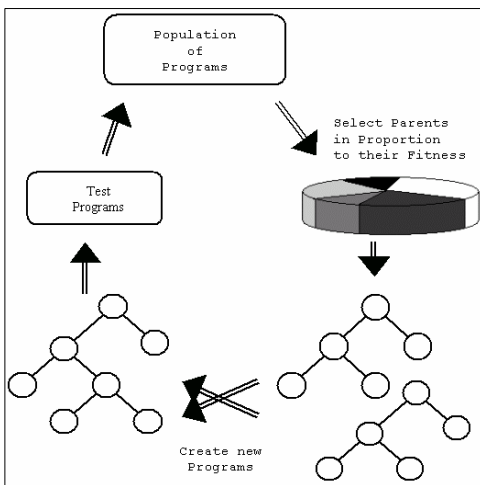


Fig. 3. The GP Cycle [Langdon, 2002].

Similar to GAs, GP works by imitating aspects of natural evolution to generate a solution that maximizes (or minimizes) some fitness function; a population of solution candidates evolves through many generations towards a solution using 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 or integers representing 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) solving 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. Fig. 3 visualizes how the GP cycle works: As 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 [Langdon, 2002].

3. GP-BASED CLASSIFICATION

On the basis of preliminary work on the identification of nonlinear structures in mechatronical systems ([Winkler et al., 2004 (a)], [Winkler et al., 2004 (b)], [Winkler, 2004]) we have developed a GP-based approach for a statistical classification algorithm. This algorithm works on a set of training examples with known properties ($X_1...X_n$). One of these properties (X_i) has to represent the membership information with respect to the underlying taxonomy. On the basis of the training examples, the algorithm tries to evolve (or, as one could also say, to "learn") a solution, i.e. a formula, that represents the function that maps a vector of object features into one of the given classes. In other words, each presented instance of the classification problem is interpreted as an instance of an optimization problem; a solution is found by a heuristic optimization algorithm. Further details about the operators used are given for example in [Winkler et al, 2005 (a)].

The goal of the implemented GP classification process is to produce an algebraic expression from a database containing the measured results of the experiments to be analyzed. Thus, 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 or terminal ones:

- A nonterminal node represents a function performing some action on one or more

property values within the structure to produce the values of the target property (which of course should be the property which indicates which class the objects belong to),

- a terminal node represents an input variable (i.e., a pointer to one of the objects' properties) or a constant.

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 any GP modeling process [Gray, 1998] because this library should be able to represent a wide range of systems. When the evolutionary algorithm is executed, each individual of the population represents one structure tree.

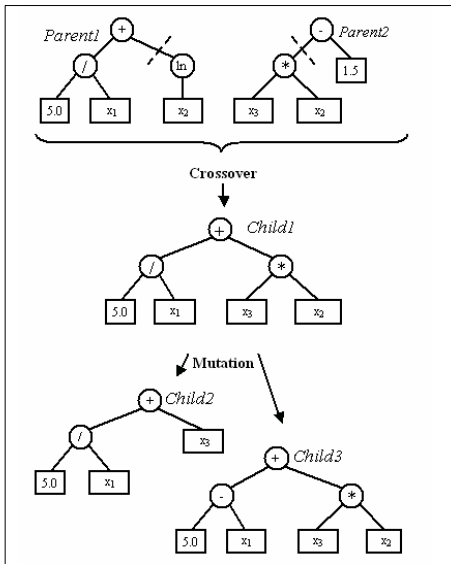


Fig. 4. Genetic Operations on Tree Structures.

Since the tree structures have to be usable by the evolutionary 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 index 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 re-stimulate genetic search. Examples of genetic operations on tree structures are shown in Figure 4: The crossover of parent1 and parent2 yields child1, child2 and child3 are possible mutants of child1.

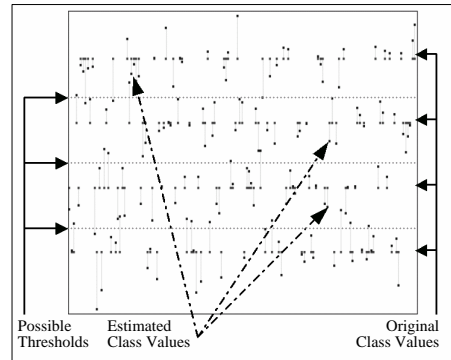


Fig. 5. Evaluation of a Classification Model.

Within the approach we present here one also has to deal with the problem that a mathematical formula, which is the output of the classification algorithm, cannot assign a certain class to a given test object. What is done instead is that a value is calculated and, on the basis of this number, one can determine how an object is classified. Furthermore one then has to determine optimal thresholds between the original class values so that as many objects as possible are classified into the correct class (cf. statistical approaches such as logistic regression). This is schematically shown in

Figure 5: An exemplary classification model is evaluated; each test sample object is evaluated and its estimated class value compared to the original one, possible thresholds between the given classes are drawn, too.

Since the GP algorithm tries to maximize or minimize 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 classification this function should be an appropriate measure of the level of agreement between the measured objects' or experiments' calculated class values and their original ones (i.e., the underlying classification hypothesis and the real world). Calculating the sum of squared errors J between original class values o_i and calculated class values c_i is a good measurement of the quality of the formula at hand:

$$J = \sum_{i=1}^N (o_i - c_i)^2 \quad (1)$$

4. USAGE OF THE PROPOSED GP-CLASSIFICATION TOOL

As already mentioned, the presented GP classification approach has been implemented as a part of the "HeuristicLab" (HL) [Wagner & Affenzeller, 2005 (a)], a framework for prototyping and analyzing optimization techniques 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. 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.

In fact, the usage of this software is really quite simple. A standard GA implementation as well as a problem plug-in for the Traveling Salesman Problem (TSP) is distributed as a part of the basic package which is available for download from www.heuristiclab.com; any kind of

new algorithm or problem-definition can easily be incorporated in the HL by simply including the respective library file. The *INS* plugin is used for data based structure identification (for time series analysis as well as for treating classification problems).

After importing the data representing a specific classification problem instance (typically, from a text file), a new structure identification problem instance is created. In fact, there are several parameters to be set; amongst others, the user has to select the target signal (e.g., that signal which represents the class values), select training data and test data, set the maximum size of the produced formula structure trees, select the node types which are available for building the formula structures, pick an evaluation operator and many more.

After defining an identification problem instance as an optimization problem in the HL, any given GA implementation can be used for solving it, i.e. identifying the structure that describes the underlying system as well as possible. The algorithm we have used for our investigations is the SASEGASA [Affenzeller & Wagner, 2004], a new generic evolutionary algorithm for retarding the unwanted effects of premature convergence by combining several evolutionary computation concepts. After starting a new instance of the chosen algorithm, the problem file has to be loaded as the problem to be solved, and after setting some basic parameters (as, e.g., the population size or the selection of crossover and mutation operators), the algorithm can be started. During its execution, the user is always able to monitor the algorithm's behavior by observing graphical displays of the actual best solution produced as well as the progress of the solution quality curve and the selection pressure. After the identification algorithm has finished, the produced solution can be exported and investigated in detail using the *INSSolutionAnalyzer*. This program can be used for analyzing solutions found for structure identification problems. For example, solutions can be pruned or manipulated, and ROC curves can be produced.

Receiver Operating Characteristic (ROC) curves have long been used in signal detection theory and are able to visualize the quality of a classification hypothesis; good explanations of ROC curves and how the area under them have to be interpreted are for instance given in [Hanley and McNeil, 1982] and [Bradley, 1997]. ROC curves are often used to judge the discrimination ability of classifiers or classification methods; typically, they are calculated as follows: For each possible threshold value discriminating two given classes (e.g., ‘true’ and ‘false’ or ‘normal’ and ‘diseased’), the numbers of true and false classifications for one of the classes are calculated. For example, if the two classes ‘true’ and ‘false’ are to be discriminated using a given classifier, each possible threshold is tested and the true positives (TP) and the false positives (FP) are counted. Each pair of TP and FP values produces a point of the ROC curve; if two classes can be ideally separated, the ROC curve goes through the upper left corner. Examples are given in the following section.

5. RESULTS

Empirical studies with various problem instances are possibly the most effective way to analyze the potential of heuristic optimization searches like evolutionary algorithms. For testing the classification tool presented here we have picked two real-world data sets: The *Wisconsin Breast Cancer* and the *Melanoma* data set.

The *Wisconsin* data set is a part of the UCI Machine Learning Repositoryⁱ; in short, it represents medical measurements which were recorded while investigating patients potentially suffering from breast cancer. The number of features recorded is 9, the file version we have used contains 683 recorded examples (by now, 699 examples are already available since the data base is updated regularly). A detailed description of the problem can be found on the KEEL homepageⁱⁱ; the best results published using various classifiers (which achieve up to

98.7% accuracy on the training data and 98.5% on the test data) can be found there, too.

The *Melanoma* data set represents medical measurements which were recorded while investigating patients potentially suffering from skin cancer. It contains 1311 examples for which 30 features have been recorded; it has been provided to us by Prof. Michael Binder from the Department of Dermatology at the Medical University Vienna, Austria.

Both data sets were investigated via 10-fold cross-validation. This means that each original data set was divided into 10 disjoint sets of (approximately) equal size. Thus, 10 different pairs of training (90% of the data) and test data sets (10% of the data) can be formed and used for testing the classification algorithm. The results obtained for the two mentioned problems are given in Table 1: For each test case, the percentages of correct classifications on the training data and on the test data are given.

Table 1. Test Results for *Wisconsin* and *Melanoma*.

<i>Test Case</i>	<i>Wisconsin</i>		<i>Melanoma</i>	
	<i>train</i>	<i>test</i>	<i>train</i>	<i>test</i>
0	98.37	94.12	97.29	95.42
1	98.37	97.06	95.51	98.47
2	99.02	98.53	96.36	96.18
3	98.54	92.65	96.36	93.13
4	98.70	89.71	96.27	91.60
5	98.37	98.53	96.36	96.95
6	98.21	95.59	97.12	93.13
7	98.05	100.00	96.19	96.18
8	97.56	95.59	95.59	96.95
9	98.21	97.10	96.19	96.18
<i>Avg.</i>	98.34	95.86	96.24	95.42

Furthermore, in Table 2 we give the so-called confusion matrix for partition 0 of the *Melanoma* problem stating for both classes the numbers of correct and incorrect classifications on the training as well as on the test data. The classification model produced for this partition is graphically shown in Figure 6. Finally, the ROC curves for the classifier produced for partition 9

of the *Wisconsin* problem evaluated on the training data as well as on the test data are shown in Figure 7.

Table 2. Confusion Matrices for *Melanoma* (part. 0).

<i>Identification Data</i>			
+	-	← <i>Correct Classification</i>	
96	6	+	<i>Predicted Classification</i>
26	1052	-	

<i>Test Data</i>			
+	-	← <i>Correct Classification</i>	
3	6	+	<i>Predicted Classification</i>
0	122	-	

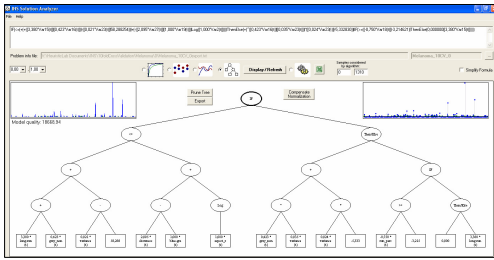


Fig. 6. Classification Model produced for *Melanoma*, Partition 0 (Screenshot of the INSSolutionAnalyzer).

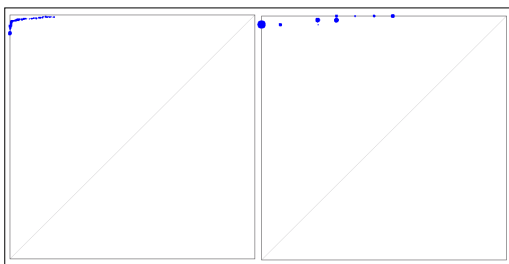


Fig. 7. ROC Curves Visualizing the Quality of the Best Result for *Wisconsin* (part. 9), Evaluated on Training Data (left) and Test Data (right).

6. WHY DOES IT WORK SO WELL?

There are several facts that make results of this quality possible.

On the one hand, the design of the implemented solution candidates as structure trees

representing mathematical formulae seems to be appropriate for time series analysis as well as for attacking classification problems.

On the other hand, the design of appropriate problem specific genetic operators (responsible for crossover and mutation) surely also contributes to the success or the failure of a GP based identification approach. In fact, we have used several different operators realizing crossover and mutation in parallel. If more than one crossover operator is available for a certain problem representation, it is beneficial and recommendable to apply all contempable crossover operators in parallel with the same probability of coming into operation. In this way all operators (even the worse) propagate only their advantageous crossover results into the next generation and as different crossover mechanisms tend to generate different offspring, this strategy increases the broadness of genetic search without counteracting the goal directedness, which also helps to retard premature convergence. [Affenzeller, 2005]

Furthermore, in addition to offspring selection incorporated in the SASEGASA (for a more detailed analysis see [Affenzeller, Wagner and Winkler, 2005]), the SASEGASA requires two selection operators (whereas standard GA implementations use only one). For a detailed explanation of this Gender-Specific Selection see [Wagner and Affenzeller, 2005 (b)]. It seems to be the best idea to choose the roulette-wheel selection in combination with the random selection operator. The reason for this is that apparently merging the genetic information of rather good individuals (formulae) with randomly chosen ones is the best strategy when using the SASEGASA for solving identification problems. This observation is in fact similar to our experience collected during the tests on the identification of mechatronical system.

7. CONCLUSION, OUTLOOK

A GP based classification algorithm has been explained in this paper. Furthermore, we have documented its applicability especially to

bioinformatical classification problems by analysing the results obtained for two wide spread benchmark data sets.

Of course, the software presented is still subject to ongoing research. At the moment we are testing an on-line version of this structure identification approach [Winkler, Affenzeller and Wagner, 2005 (b)]. Furthermore, the use of several populations in parallel and the combination of the respective classification statements via voting is planned to be implemented, too.

8. REFERENCES

- Affenzeller, M. (2003) *New Hybrid Variants of Genetic Algorithms - Theoretical and Practical Aspects*. Universitätsverlag Rudolf Trauner, Linz, Austria.
- Affenzeller, M. (2005) *Population Genetics and Evolutionary Computation - Theoretical and Practical Aspects*. Universitätsverlag Rudolf Trauner, Linz, Austria.
- Affenzeller, M. and Wagner, S. (2004) 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*, **10**, 239-263. Kluwer Academic Publishers.
- Affenzeller, M., Wagner, S. and Winkler, S. (2005) Goal-Oriented Preservation of Essential Genetic Information by Offspring Selection. *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO) 2005*.
- Bradley, A. (1997) The Use of the Area under the ROC curve in the Evaluation of Machine Learning Algorithms. *Pattern Recognition*, **30**, 1145-1159.
- Goldberg, D.E. (1989) *Genetic Algorithms in Search, Optimization and Machine Learning*. Addison Wesley Longman.
- Gray, G. et al. (1998) Nonlinear Model Structure Identification Using Genetic Programming. *Control Engineering Practice*, **6**.
- Hanley, J. and McNeil, B. (1982) The Meaning and Use of the Area under a Receiver Operating Characteristic (ROC) Curve. *Radiology*, **143**.
- Holland, J.H. (1975) *Adaption in Natural and Artificial Systems*. 1st edn. The MIT Press.
- Koza, J. (1992) *Genetic Programming: On the Programming of Computers by means of Natural Selection*. The MIT Press.
- Langdon, W. and Poli, R. (2002) *Foundations of Genetic Programming*. Springer Verlag, Berlin Heidelberg New York.
- Michalewicz, Z. (1996) *Genetic Algorithms + Data Structures = Evolution Programs*. 3rd edn. Springer-Verlag, Berlin Heidelberg New York.
- Mitchell, T.M. (2000) *Machine Learning*. McGraw-Hill, New York.
- Wagner, S. and Affenzeller, M. (2005) HeuristicLab: A Generic and Extensible Optimization Environment. *Adaptive and Natural Computing Algorithms*. Springer Computer Science, 538-541.
- Wagner, S. and Affenzeller, M. (2005) SexualGA: Gender-Specific Selection for Genetic Algorithms. *Proceedings of the 9th World Multi-Conference on Systemics, Cybernetics and Informatics (WMSCI) 2005*.
- Winkler, S., Affenzeller, M. and Wagner, S. (2004) Identifying Nonlinear Model Structures Using Genetic Programming Techniques. *Cybernetics and Systems 2004*, pp. 689-694. Austrian Society for Cybernetic Studies.
- Winkler, S., Affenzeller, M. and Wagner, S. (2004) New Methods for the Identification of Nonlinear Model Structures Based Upon Genetic Programming Techniques. *Proceedings of the 15th International Conference on Systems Science*, **1**, pp. 386-393. Oficyna Wydawnicza Politechniki Wroclawskiej.
- Winkler, S. (2004) *Identifying Nonlinear Model Structures Using Genetic Programming*. Diploma Thesis, Institut für Systemtheorie und Simulation, Johannes Kepler Universität Linz, Austria.
- Winkler, S., Affenzeller, M. and Wagner, S. (2005) Solving Multiclass Classification Problems by Genetic Programming. *Proceedings of the 9th World Multi-Conference on Systemics, Cybernetics and Informatics (WMSCI) 2005*.
- Winkler, S., Affenzeller, M. and Wagner, S. (2005) Genetic Programming Based Model Structure Identification Using On-Line System Data. Accepted to be published in: *Proceedings of Conceptual Modeling and Simulation Conference - CMS 2005*.

ⁱ <http://www.ics.uci.edu/~mllearn/MLRepository.html>

ⁱⁱ <http://sci2s.ugr.es/keel-dataset/>
