# MODEL SYNTHESIS USING A MULTI-AGENT LEARNING STRATEGY

**Sebastian Bohlmann[(a)], Arne Klauke[(b)], Volkhard Klinger[(c)], Helena Szczerbicka[(d)]**

[(a)(d)]Department of Simulation and Modeling, Leibniz University Hannover, 30167 Hannover, Germany
[(b)(c)]Department of Embedded Systems, FHDW Hannover, 30173 Hannover, Germany

[(a)]bohlmann@sim.uni-hannover.de, [(b)]arne.klauke@fhdw.de, [(c)]volkhard.klinger@fhdw.de, [(d)]hsz@sim.uni-hannover.de

**ABSTRACT**
In this paper we give an overview of our multi-agent based model identification framework. We are identifying functional relationships in process data. We do this by using multi-agent based heuristic algorithms. Moreover we give a proof of concept concerning the abilities and performance of our system.

Keywords: model synthesis, agent-based evolutionary computation

## 1. INTRODUCTION

Manufacturing systems are one of the largest application areas for modelling and simulation. In particular the pulp and paper industry is one instance of a large-scale production processes (Bohlmann and Klinger, 2007). We have brought up a framework for modelling and simulation of those process environments in (Bohlmann, Klinger and Szczerbicka, 2009), (Bohlmann, Klinger and Szczerbicka, 2010b) and (Bohlmann, Klinger and Szczerbicka, 2010c).

This paper focuses on the identification procedure. We consider time series extracted from process data. These time series are subdivided in input and output series. The problem treated here, is to find a functional relationship between the input and output series. At the beginning it is unknown which of the input series are actually used in that relationship. Our approach to solve this problem is a multi-agent based learning strategy (Bohlmann, Klinger and Szczerbicka, 2010b).

In figure 1 the system identification overview is shown. It consists of two basic steps, the preprocessing and the multi-agent based optimization. The process data input (PData) is used to generate an appropriate process model (Law and Kelton, 2000).

To verify this identification procedure we have to evaluate the different steps very carefully not only to its technically correct function but on its performance behaviour.

$$\begin{aligned} f_1((x_1)_t, \dots, (x_m)_t) &= (y_1)_t, t \in \mathbb{N} \\ &\vdots \\ f_j((x_1)_t, \dots, (x_m)_t) &= (y_j)_t, t \in \mathbb{N} \end{aligned} \quad (0)$$

The verification strategy is based on a set of data sequences $(x_1)_t, \dots, (x_m)_t, t \in \mathbb{N}$, called Input Sequences and Output Sequences $(y_1)_t, \dots, (y_j)_t, t \in \mathbb{N}$, which are related to the Input Sequences by a functional relationship $f: \mathbb{R}^m \longrightarrow \mathbb{R}^j$ (formula 0), illustrated in figure 2.

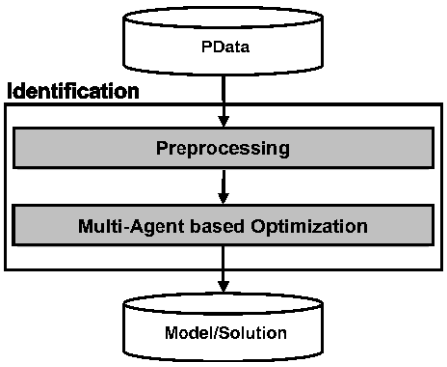

Figure 1: Function block view

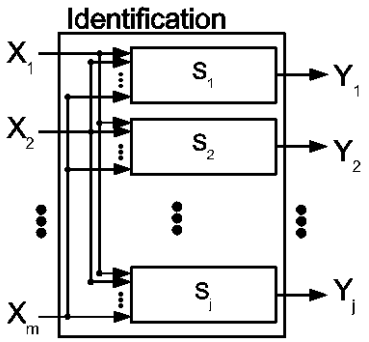

Figure 2: Input/ Output sequence

In figure 3 an example for m=10 and j=1 is shown. The problem we are solving is to identify this function f, only knowing values of $(x_1)_t, \dots, (x_m)_t$ (thin lines) and $(y)_t$ (thick line) for a limited set $T \subset \mathbb{N}$ of time indices, which may differ for each sequence. In this paper we are treating only problems with j=1.

Our approach for this challenge is formed by the identification framework used for process model identification and it uses the data management framework presented in (Bohlmann, Klinger and Szczerbicka, 2010c).

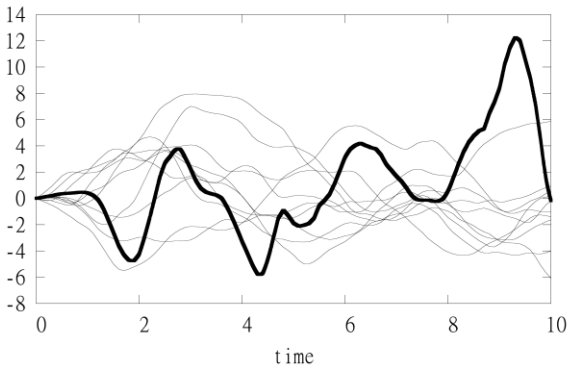

Figure 3: Input and output data series

The pre-processing is followed by a Multi-Agent-based Learning Strategy (section 3.1.2) using memetic evolutionary algorithm.

In the next sections the identification framework is explained in detail. It consists essentially of two parts (see Figure 1): A data pre-processing unit and our evolutionary algorithm. Finally some examples, proving the functioning of the framework, are discussed.

## 2. DATA PREPROCESSING

In this section the preprocessing of the raw data is explained in detail. figure 4 presents the basic function blocks.

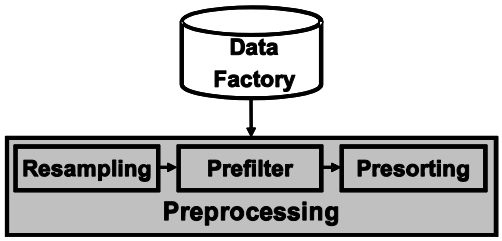

Figure 4: Data Proprocessing

### 2.1 Data Factory

The data used for the identification process is either extracted from various sensors in a real system or they may be synthetic. In the first case the data is usually collected over a long time period and saved in a data archive. This raw data then runs through different preprocessor units, explained below, to build usable data series for the identification framework. In the second case the raw data is produced synthetic using random data streams and is then passed to the preprocessing units.

### 2.2 Resampling the Data

In this initial unit the weaknesses of the data-recording by the sensors are remedied. The sensors distributed over the system are usually not working periodic or even synchronously. Moreover they might sometimes measure values, which are obvious wrong. These error values are simply removed from the data set. The actual task in this module is to produce a time series of equidistant data samples, each of which describes the state of the whole system at a moment.

To achieve this, at first the data from each sensor is linearly interpolated and then smoothed by a convolution. After that the new data sequences are equalized with the original values from the sensors. Finally equidistant values from each sensor are picked and combined to data samples, which describe the whole system, as desired.

### 2.3 Prefiltering the Data

The environment built for the evolutionary algorithm, explained in the next section, needs a predetermined number of data samples. In general the number of data samples delivered by the resampling unit is too large. Moreover it may contain redundant data samples, containing no information. This may happen if the state of the real system does not change for a time period. In this unit the samples, which contain the most information, i.e. these with the highest entropy, are chosen. This is implemented in different prefilter modules.

1. *No Prefilter*: The simplest way is to choose just the last 729 samples.
2. *Random Prefilter*: The samples to be passed onto the planets are chosen randomly.
3. *Weighted Random Prefilter*: The samples to pass on are chosen randomly, but with different probabilities. This probability corresponds to the angle between the input values of the current sample and its predecessor and successor.
4. *k-means Prefilter*: In this method we are using a cluster algorithm to choose the samples to be passed on (Kanungo, Mount, Netanyahu, Piatko, Silverman and Wu, 2002). The data set delivered by the resampling unit is subdivided in blocks of a fixed size. In each of these blocks we build a fixed number of clusters and only the centres of these clusters are passed to the planets. We have decided to use k-means clustering because one can choose the number of centres from start. Furthermore the clusters generated by this algorithm are formed spherical, what seems to be the most suitable form for our purpose.

### 2.4 Presorting the Data

After the samples are selected they need to be arranged on the planets in a useful way. The simplest method is to keep them in their current order. But there are concepts, which can improve the system behaviour.

One approach to order the samples in a more useful way is the so called TSP Filling (Travelling Salesman Problem). The samples are arranged in a way that approximately minimizes the sum of the distances of neighbouring samples, with respect to a chosen metric. This concept can be generalized by not just taking the direct neighbours into account, but the next *n* neighbours in both directions for each sample.

## 3. MULTI-AGENT-BASED OPTIMIZATION

The algorithm uses an evolutionary approach to find the functional relationship in the process data. We have created an environment which offers aliments to the creatures living in it. These creatures own a genotype, which they are trying to adapt to their location in the environment by building children with a changed genotype. A creature which is well fitted to its location has a better ability to absorb aliments from it. Aliments are used to perform various evolutionary operations to build a child. The genotype passed to the child can be mutated, crossed with the genotype of another creature, and enhanced in several ways. The creatures can move within their environment and interact with other creatures.

The environment is representing the data set, the local view is just a subset of it. The creatures are software agents and their genotype is a model function, approximating the functional relationship. We can rate how well an agent is fitted to its location by calculating the error of his model function using the local data and a search metric.

In the following section we will give an overview of the system architecture, depicted in figure 5.

### 3.1 System architecture overview

The architecture is organized in four stages, the data processing and local and global agent environment. Each is arranged in functional levels, for data management, agent behaviour control, supervision, execution and synchronization, regarding the overall management. The basic level is called MPU (Multi Processor Unit) and represents the system thread representation.

The optimization starts with the initialization of the preprocessed data, managed in the data processing stage. The splitter in the supervision level supplies the evaluation units in the other stages with data samples.

To illustrate the agent based algorithm, we give a detailed description of the environment, the individuals live in, of the individuals themselves and of their behaviour.

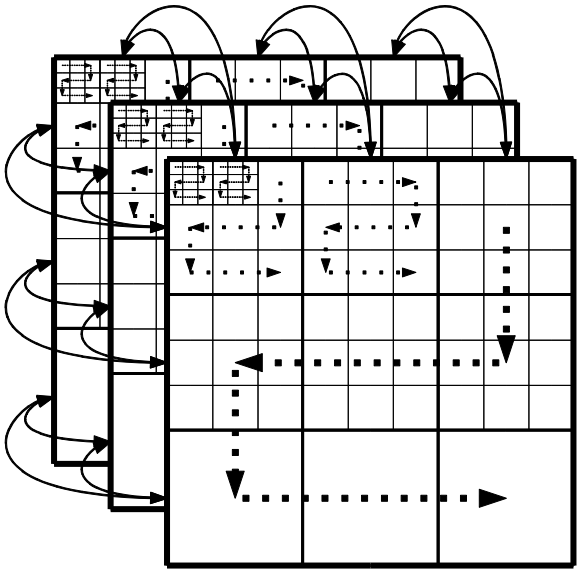

Figure 5: Planet Surface

#### 3.1.1 Agent Environment

The environment of the agents consists of areas. Each area contains one data sample and can hold one agent. Moreover areas can be linked to other areas, called neighbours. These neighbours serve for two purposes. At first the agent held by one area can be moved onto one of the areas neighbouring areas. Secondly the links between the areas are used to build the set of data samples for the local learning.

The areas are aggregated in planets. The surface of a planet is a torus, represented by a quadratic field of areas.

This surface is build in a recursive pattern of squares containing nine elements, filled meander like. This method leads to the planet size $9^3 = 729$ and is illustrated in figure 6. Each area on a planet is linked to its four neighbours to the left, right, top and bottom.

Finally the planets build the universe, which is controlled by the universe supervisor. All planets are controlled by so called planet supervisors. Some of the areas on each planet are marked as beam areas. When a certain number of iterations have passed, the planet supervisor sends copies of all agents, which are placed on a beam area to a randomly chosen area on a randomly chosen planet.

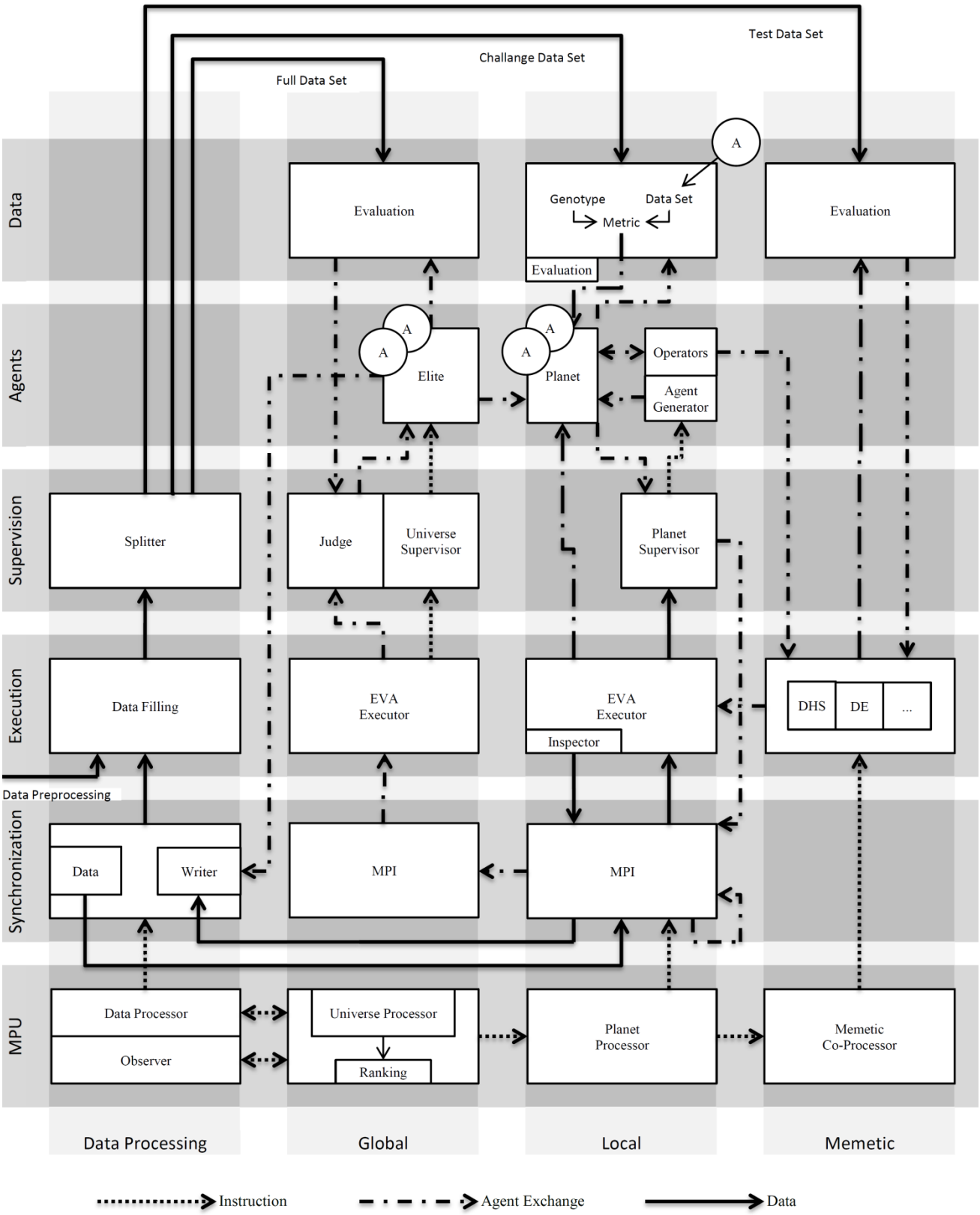


Figure 6: System Architecture

### 3.1.2 Software Agents

The software agents are operating in the environment described before. Each agent owns a model function and is placed onto an area. The model function is stored in a tree representation and is build using elementary operations like sin, +, *, / the variables $x_1,..,x_m$ and a set of parameters (Schmidt and Lipson, 2007). This concept is shown in figure 7. Moreover agents may build a child, to pass on their information. We have implemented in the following evolutionary operations:

*Replication*: The individual produces a copy of himself. This is the most expensive operation.

*Cross*: Two individuals interchange randomly chosen parts of their model functions.

*Mutate*: A part of the model function is replaced by a randomly build function.

*Enhance*: The structure of the model function is simplified, if possible.

*Short Learn*: The parameters of the model function are fitted to the local learn data using a simple, but fast algorithm.

*Learn*: A more sophisticated algorithm is used to calibrate the parameters of the model function. This operation is implemented in a memetic coprocessor.

All of these operations have an energy effort, which is subtracted from the agents energy, if the operation is performed. Furthermore the agents can measure the error of their model function using different kind of search metrics.

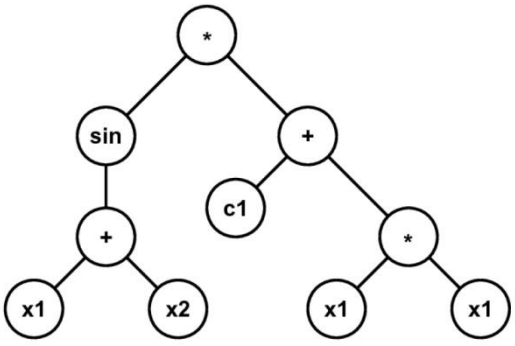


Figure 7: The tree representation of the model function

### 3.1.3 Agent Behaviour

Each iteration starts with an update of the agents properties. The age of the individual is increased. After that the energy level is recalculated. This happens as follows: The individual calculates the error of its model function using one of the search metrics described in section 3.4. If this error is too big the energy level is not increased. Else, the individual is allowed to absorb the energy offered by the area it is placed on and a value depending on the complexity of its model function is subtracted from its energy level. When these operations are done we decide whether the agent may live for one more iteration or not.

If the energy level is not positive the agent is removed. If the agent had a child, it is placed onto the area.

If the energy level is positive the agent tries to move to a randomly chosen neighbouring area. When the chosen area is empty, the agent just moves. If the area is already occupied by another agent, the procedure depends on the agent's energy level. If it is not high enough to perform a cross operation, the agent does not move. Else a cross operation is performed. If the cross operation yields a new agent it is saved as the child of the original agent and the agents do not move. After the move operation, the individual tries to build a child, if none is present. Depending on the agent's energy level, the agent performs a Replication, Mutate or Enhance operation to build his child.

In the next step the agent may perform a Learn operation, if he is not adult (without loss of energy), or if his energy is high enough it is decided randomly if the agent is allowed to learn or not.

Now the individual performs a short learn operation, if its age is appropriate.

Finally, if the agent has moved in this iteration and owns a child, he tries to place it on his last area.

## 3.2 Optimization stages

In the last section the agents have been introduced. Here we map this agent based algorithm on the system architecture. According to the planet setup, there is the global stage (universe) and the local stage (one planet). The fourth stage is formed by memetic coprocessors, assigned to the planets. These stages form separate execution loops, which run in parallel. The listing below gives an overview of these three loops.

1. *Global Stage*: This loop is used to manage the elite population. Agents nominated in the local loop are passed via the MPI (Message Passing Interface, synchronization level) to the EVA Executor (execution level) and then to the Judge (supervision level), which decides, if the agent is added to the elite population (agent level). The agents in the elite population are evaluated on the full data set (data level). If one of them fulfils the termination criterion the algorithm stops. The universe processor (MPU level) returns the ranking of the best individuals if the algorithm terminates.
2. *Local Stage*: In this stage the agents are generated and put onto the planets. Once they are placed on an area, the agents start their life cycle, described in subsection 3.1.3. In the data level the agents evaluate their model function, using a search metric and a subset of the challenge data set, called local learn data. In the MPI (synchronization level) agents are exchanged between the planets.

3. *Memetic Stage*: In this loop agents, which were passed from the Operators (agent level, local stage) to the memetic unit (execution level), are optimized by more expensive algorithms, like downhill simplex (DHS) (Nelder and Mead, 1965). To evaluate the error of the model function, a small test data set is used. The agents are then passed back to the EVA Executor in the execution level of the local loop.

### 3.3 The Elite Population

There are two opportunities for an individual to get nominated for the elite population. The first is when an individual is removed from the planet and has lived for at least 200 iterations. Furthermore an individual is nominated for the elite population after every 250 iterations. The universe supervisor chooses the best 25 individuals from all nominated agents.

The elite population is used to check the termination criterion: The model functions in the elite population are evaluated not just using the local learning set, but the data from the whole universe. If one of them has an error under a predefined border the algorithm terminates and returns this model function.

### 3.4 Search Metrics

The agents are able to use different search metrics to estimate the error of their model functions. For each data sample in the local learn set, which is build using the neighbouring areas, the agent calculates the difference between the original output and the output calculated by the model function. For this step the agent can use the euclidean or the absolute distance. In the next step these values are aggregated by building the mean value or the maximum over all samples. They might perform these steps using not all, but a randomly chosen subset of their local learn data, in these cases the metric is called partial.

## 4. PROOF OF CONCEPT

To proof the basic usability of our framework we executed 8 different experiments with different problem complexities to be solved by the agent system. Each experiment is repeated 30 times to reduce statistical deviation. For each run we take the runtime beginning with the first agent generation and ending at the first successful detection. A successful detection is defined by a mean absolute error of 0.001 over the whole dataset. Each run has a maximal runtime of 1800 seconds. If no valid solution is found in time, the execution is aborted and counted as an unsuccessful execution. To proof the parallelisation concept we repeated 8 times 30 experiments with 1 execution core (one Planet) and 8 cores.

### 4.1 Experimental Setup

This section specifies the used configurations and dependencies. All experiments are executed on a Dell PowerEdge R815 with in total 4 AMD Opteron 6174 processors (each providing 12 cores with respectively 128KByte L1-cache, 512 KByte L2-cache and common 12 MByte L3-cache) and an overall RAM configuration of 128 GByte. For the parallelization evaluation this platform provides a scalable hardware environment.

The challenge to be solved by the agent system is generated synthetically. Because randomness in the generation has huge impact on the detection system the same dataset is used in all 30 runs per experiment. We generated a five dimensional time series input stream

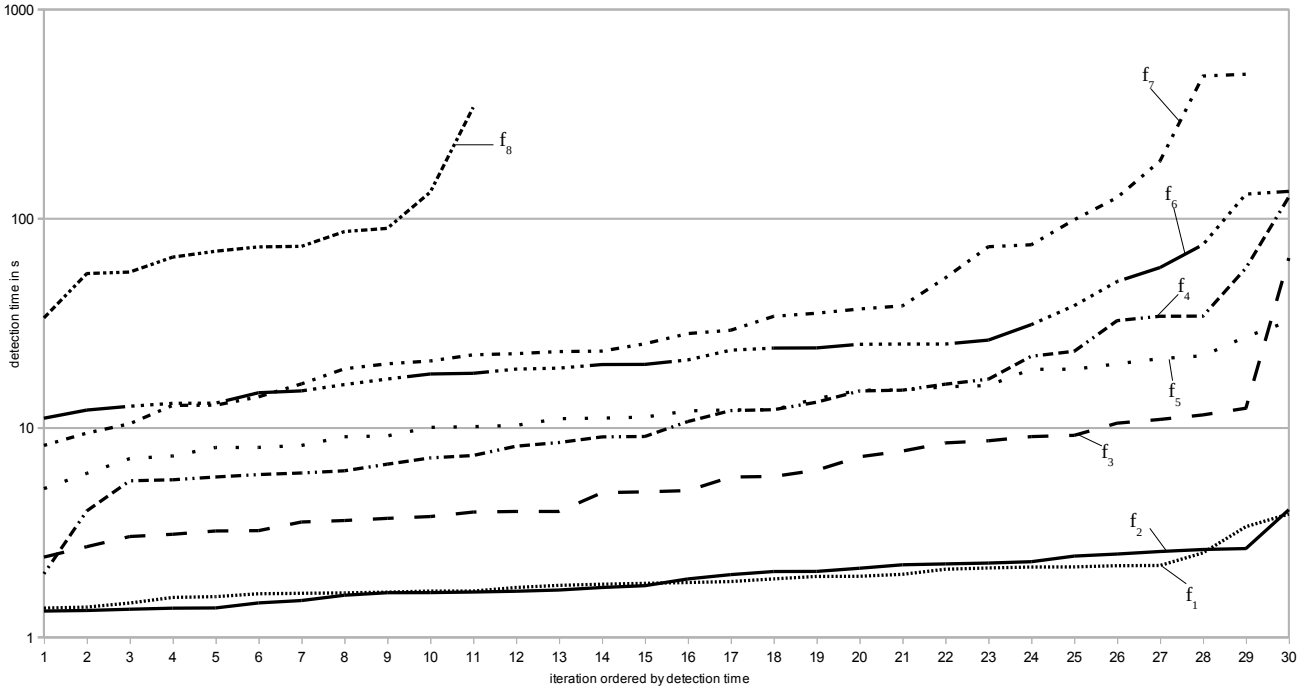

Figure 8: Measured ordered runtime

with $10^5$ samples each. Further a result stream with the same length is generated using one of the 8 functions in formula (1) to (8). To measure the quality of the result a metric calculating the mean absolute error is used.

$$f_1(x_1, \dots, x_5) = x_1 \cdot x_2 \quad (1)$$

$$f_2(x_1, \dots, x_5) = \sin\left(x_1 + \frac{\pi}{4}\right) \quad (2)$$

$$f_3(x_1, \dots, x_5) = 42 \cdot x_1 \cdot x_1 + 22 \cdot x_2 \quad (3)$$

$$f_4(x_1, \dots, x_5) = \sin(x_1 \cdot x_2 + 3) + 4 \quad (4)$$

$$f_5(x_1, \dots, x_5) = x_3 + x_4 + 4.3 \cdot \sin(x_1 + 7) + 1.65 \quad (5)$$

$$f_6(x_1, \dots, x_5) = x_1 \cdot x_1 + x_2 \cdot x_2 + x_3 \cdot x_3 + 22 \quad (6)$$

$$f_7(x_1, \dots, x_5) = -3.23 \cdot \sin(x_1) \cdot \sin(x_2) + 2.43 \cdot \sin\left(x_1 + \frac{\pi}{4}\right) \quad (7)$$

$$f_8(x_1, \dots, x_5) = -122 \cdot x_1 + 2.3 \cdot x_2 + 0.2 \cdot x_4 + x_3 \cdot \sin(0.1 \cdot x_4) \quad (8)$$

### 4.2 Results

Figure 8 illustrates the results for the parallelised agent system with 8 planets. Each line type represents the different successfully ended runs for one of the generator functions ordered by runtime. The y-scale is logarithmic. As expected the mean runtime is higher if the function is more complex. This value can also be found in table 1. Some of the data lines do not have 30 samples, because not all runs resulted in a valid solution. For function 1 to 6 the agents evolution always leads to the correct structure. Especially for the last two functions the highest runtimes increase strongly.

Table 1:Statistics for different functions

| | Mean | Std. Dev. | Detection Rate |
|---|---|---|---|
| $f_1$ | 1.8 s | 0.5 s | 100 % |
| $f_2$ | 1.8 s | 0.6 s | 100 % |
| $f_3$ | 5.0 s | 11.2 s | 100 % |
| $f_4$ | 9.9 s | 23.9 s | 100 % |
| $f_5$ | 11.7 s | 6.4 s | 100 % |
| $f_6$ | 20.6 s | 31.0 s | 100 % |
| $f_7$ | 25.3 s | 121.6 s | 97 % |
| $f_8$ | 73.4 s | 84.4 s | 37 % |

This effect caused a deadlocked evolution, if the overall diversity of the agents is low. When the overall number of agents and the separation by multiple planets is increased this probability decreases. This is indicated if we compare the total number of detections and the speedup as done in table 2. Here the functions five to eight are not listed because the detection rate is too low to acquire adequate measurements for a non-parallelized system. Speedup for complex challenges (3-5) is effective. As positive detection rates increase and detection runtime decreases the positive effect is higher than the expected factor 8. At the moment more planets do not lead to a better system performance on the used machine because the other system components (see figure 6) consume the remaining system resources. We conclude that also the agents only use heuristics for interaction and learning the combined execution is target-oriented.

Table 2: Statistics for parallelization

| | Mean x1 | Mean x8 | Speedup | Detection Improvement |
|---|---|---|---|---|
| $f_1$ | 5,6 s | 1.8 s | 309 % | 100 % |
| $f_2$ | 6,5 s | 1.8 s | 356 % | 100 % |
| $f_3$ | 28,0 s | 5.0 s | 561 % | 500 % |
| $f_4$ | 82,0 s | 9.9 s | 826 % | 230 % |
| $f_5$ | 59,0 s | 11.7 s | 505 % | 272 % |

## 5. SUMMARY

Modeling and simulation of non formalized system behavior still is a grand challenge for science and engineering. As we demonstrated it is possible to implement a machine learning system to help modeling specialists to gain knowledge form the data produced by the original system. In this scenario it is required to formulate the produced recommendations in a human comprehensible form. Differential equations (and simple equations, as in this concept paper) are one possible knowledge representation. And in difference, from knowledge e.g. learned by a neuronal net, knowledge is not encapsulated. Engineers have a huge tool kit to continue processing such a result. As done for a simulation system in (Bohlmann, Klinger and Szczerbicka, 2009) such a agent based modeling support system can easily connected to real word data sources and could be helpful to enhance or generate complex models for simulation environments (Zeigler, Praehofer and Kim, 2000). As a result the complexity to model a complex process is simplified by using the analytic strength of the modeling engineer and the knowledge compression strength of an agent based machine learning environment.

## 6. FURTHER WORK

The further work has two key aspects of activity: Increase the parallelization to be able to use more agents. As mentioned before the memetic co-processor cores use the majority of our machine resources. All used memetic algorithms are suited for SIMD coprocessors and would scale the system to about 40 planets. The second work to be done is to reduce the number of problem specific parameters by the help of control loops.

Finally the framework is written as generic as possible. There are only few dependencies e.g. the problem has to be dividable into local challenges. So we like to formulate solvers for different known problems in the area of modeling and machine learning.

## REFERENCES


Bohlmann, S. and Klinger, V. (2007)
Modellbildung für kontinuierliche Produktionsprozesse in der Papierindustrie. Forschungsberichte der FHDW Hannover (ISSN 1863-7043), 08:1–20.

Bohlmann, S., Klinger, V., and Szczerbicka, H. (2009)
HPNS - a Hybrid Process Net Simulation Environment Executing Online Dynamic Models of Industrial Manufacturing Systems. In Proceedings of the 2009 Winter Simulation Conference M. D. Rossetti, R. R. Hill, B. Johansson, A. Dunkin, and R. G. Ingalls, eds.

Bohlmann, S., Klinger, V., and Szczerbicka, H. (2010a)
System Identification with Multi-Agentbased Evolutionary Computation Using a Local Optimization Kernel. In Submitted to ICMLA 2010 (International Conference on Machine Learning and Applications).

Bohlmann, S., Klinger, V., and Szczerbicka, H. (2010b)
System identification with multi-agent-based evolutionary computation using a local optimization kernel. In The Ninth International Conference on Machine Learning and Applications, pages 840–845.

Bohlmann, S., Klinger, V., and Szczerbicka, H. (2010c)
Co-simulation in large scale environments using the HPNS framework. In Summer Simulation Multiconference, Grand Challenges in Modeling & Simulation. The Society for Modeling and Simulation.

Kanungo, T., Mount, D. M., Netanyahu, N. S., Piatko, C. D., Silverman, R., and Wu, A. Y. (2002)
An efficient k-means clustering algorithm: Analysis and implementation. IEEE Transactions on Pattern Analysis and Machine Intelligence, 24:881–892.

Law, A. M. and Kelton, W. D. (2000)
Simulation Modeling and Analysis. McGraw-Hill

Nelder, R. and Mead, J. (1965)
A simplex method for function minimization. Computer Journal, 7(4):308–313.

Schmidt, M. and Lipson, H. (2007)
Comparison of tree and graph encodings as function of problem complexity. In GECCO '07: Proceedings of the 9th annual conference on Genetic and evolutionary computation, pages 1674–1679, New York, NY, USA. ACM.

Zeigler, B. P., Praehofer, H., and Kim, T. G. (2000)
Theory of Modeling and Simulation: Integrating Discrete Event and Continuous Complex Dynamic Systems. Academic Press, San Diego, USA, 2 edition.


## AUTHORS BIOGRAPHY


SEBASTIAN BOHLMANN is a Ph.D. candidate at Department of Simulation and Modelling - Institute of Systems Engineering at the Leibniz Universit¨at Hannover. He received a Dipl.-Ing. (FH) degree in mechatronics engineering from FHDW university of applied sciences. His research interests are machine learning and heuristic optimization algorithms, complex dynamic systems, control system synthesis and grid computing. His email address is <bohlmann@sim.uni-hannover.de>.

ARNE KLAUKE is a researcher at the university of applied science FHDW in Hannover. He received a Dipl.-Math. from the Gottfried Wilhelm Leibniz Universit¨at Hannover. His email address is <arne.klauke@fhdw.de>.

VOLKHARD KLINGER has been a full time professor for embedded systems and computer science at the university of applied sciences FHDW in Hannover and Celle since 2002. After his academic studies at the RWTH Aachen he received his Ph.D. in Electrical Engineering from Technische Universität Hamburg-Harburg. He teaches courses in computer science, embedded systems, electrical engineering and ASIC/system design. His email address is <Volkhard.Klinger@fhdw.de>.

HELENA SZCZERBICKA is head of the Department of Simulation and Modelling - Institute of Systems Engineering at the Leibniz Universit¨at Hannover. She received her Ph.D. in Engineering and her M.S in Applied Mathematics from the Warsaw University of Technology, Poland. She teaches courses in discrete-event simulation, modeling methodology, queuing theory, stochastic Petri Nets, distributed simulation, computer organization and computer architecture. Her email address is <hsz@sim.uni-hannover.de>.