Analysis of Different Search Metrics Used in Multi-Agent-based Identification Environment

Sebastian Bohlmann**, Arne Klauke**, Volkhard Klinger**, Helena Szczerbicka*
*Department of Simulation and Modelling Leibniz University Hannover
30167 Hannover, Germany
Email: {hsz}@sim.uni-hannover.de
**Department of Embedded Systems FHDW Hannover
30173 Hannover, Germany
Email: {sebastian.bohlmann,arne.klauke,volkhard.klinger}@fhdw.de

***Abstract*—Process data-based system identification is one great challenge in technical process modelling and simulation. In this paper we continue our former work presented in [1], [3] and concentrate on the analysis and evaluation of both the data preprocessing and different search metrics used in the multi-agent-based optimization algorithm. This analysis helps to review and benchmark the impact on different preprocessing steps as well as the influence of the selected search metrics. Based on this evaluation we are able to verify the correct and target-oriented identification procedure.**

***Keywords*-system identification, agent-based evolutionary computation, memetic optimization algorithms**

I. Introduction

Modelling and simulating complex process models of manufacturing systems are grand challenges for science and engineering. With regard to the number of influencing variables, like power efficiency, ecological boundary conditions or product quality, process understanding is the key feature for improving the manufacturing process itself. Due to the empiric character of process description, process modelling is one of the fundamental challenges for the profound knowledge of these technical or manufacturing processes. Based on such a model, described by the combination of physical equations and a graph structure, simulation allows the reconstruction of process behaviour and therefore the optimization of the entire process [1]. This identification frame is part of the Hybrid Process Net Simulator-framework (HPNS), described in subsection I-B [2], [4]. The objectives of this framework are to identify models for the simulation and optimization of processes, especially manufacturing and industrial processes. In the following subsections we present our reference process environment and the HPNS framework.

A. The Process Environment: Specific Problems In Industrial Paper Manufacturing

The pulp and paper industry as one application example is a qualified process with regard to large system requirements [9]. There are thousands of continuous and discrete signals describing the behaviour and states of the different subsystems. Hundreds of control loops and relations exist in between these subsystems. There are many backpropagations of state changes and there is almost no separation between the different process steps. For this process no detailed model exists, which could for example provide an evaluation of the process quality. Manufacturing steps have been developed empirically over decades. Therefore up to now techniques able to optimize the manufacturing process have scarcely been used. Two of these techniques are soft sensors and model predictive control [1].

B. HPNS framework including the process identification

Optimization problems are omnipresent in engineering and science. The technical background discussed in this paper, paper manufacturing, is one excellent instance. Instead of a lot of expertise regarding particular process parts, the overall knowledge about the process interrelationship is not sufficient. So, the data-based process model identification with physical equations seems to be the only solution for establishing a precise process model. Based on the constraints and requirements considered, the HPNS framework [1] provides certain functional frames to realize the process identification. The key issue here is the identification frame, represented in Figure 1.

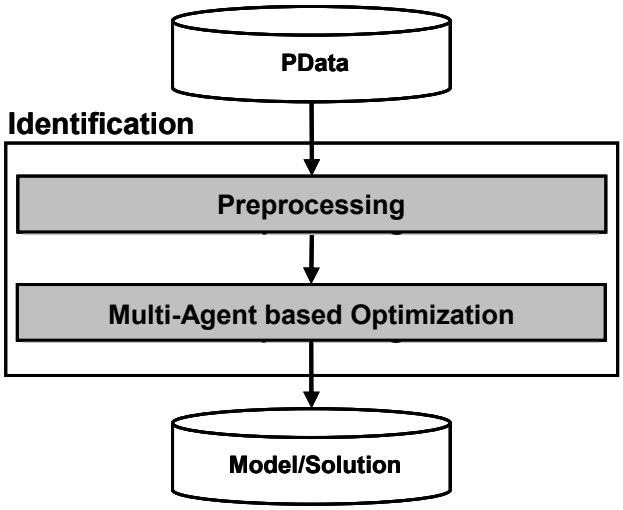

Figure 1. HPNS identification frame

The HPNS framework requires a process model as described above. The identification frame provides a model design flow, based on the historical process data archive

Proceedings 26th European Conference on Modelling and Simulation ©ECMS Klaus G. Troitzsch, Michael Möhring, Ulf Lotzmann (Editors)
ISBN: 978-0-9564944-4-3 / ISBN: 978-0-9564944-5-0 (CD)

on the running manufacturing process. While the interrelationship between the online data and the archive is not under focus here, in the following text we will call the data from all data sources process data (PData).

The data-based process identification is based on evolutionary computation and multi-strategy learning. Formerly, optimization techniques based on linear programming or branch and bound strategies have been used to solve such types of problems. In recent years agent-based computing providing evolutionary algorithms seem to be the most promising approach.

A process model is the basic requirement for successfully modelling and simulation. We are using the manufacturing process data cached by a stream database. It is transformed by an identification step to establish the model for the HPNS-framework [1], [3], [4]. It uses the symbiotic simulation approach to provide an online/offline process simulation to enable forward-looking properties like an online prediction mode.

In figure 1 we have seen the system identification overview. It consists of two basic steps, the preprocessing and the multi-agent based optimization. The PData input is used to generate an appropriate process model [6]. 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.

Here we focus on the evaluation of the data preprocessing and of different search metrics to compare their impact with regard to the quality and performance of the optimization phase. To be able to allow a meaningful evalution we are working in the following with synthetic data sets. The verification strategy is based on a set of these synthetic data sequences $(x_1)_t, \ldots, (x_m)_t$, $t \in \mathbb{N}$, called *Input Sequences* and sequences $(y_1)_t, \ldots, (y_j)_t$, $t \in \mathbb{N}$, called *Output Sequences*, which are related to the Input Sequences by functional relationships $f\colon \mathbb{R}^m \to \mathbb{R}^j$, illustrated in figure 2:

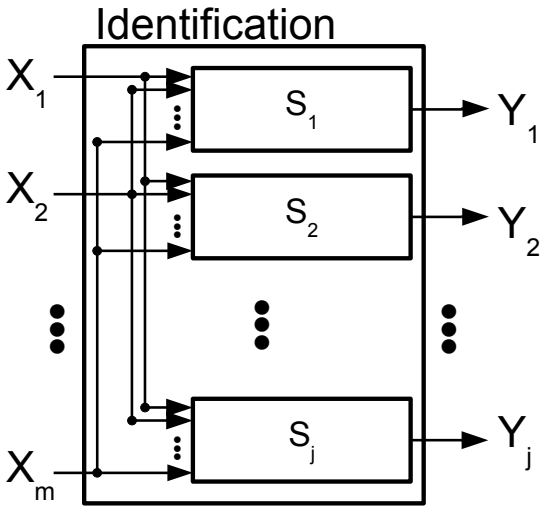

Figure 2. HPNS identification frame

$$f_1\left((x_1)_t, \ldots, (x_m)_t\right) = (y_1)_t,\ t \in \mathbb{N}$$
$$\vdots$$
$$f_j\left((x_1)_t, \ldots, (x_m)_t\right) = (y_j)_t,\ t \in \mathbb{N}$$

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 $(x_1)_t, \ldots, (x_m)_t$ (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 challange is formed by the identification framework used for process modell identification and it uses the data management framework presented in [4]. The preprocessing is followed by a Multi-Agent-based Learning Strategy (II-B) using evolutionary-memetic algorithms. In this paper we focus on the so-called *Search Metrics* (III) the agents use to measure the quality of their approximation for f. We will examine different types of metrics, to see which kind is the most suitable for our problem.

II. System Architecture

In this section we give a short overview of our System Architecture with regard to the system verification approach. It consists according to 1 of two main parts: The Data Preprocessing and a Multi-Agent-based Learning Environment.

A. Data Preprocessing

The agents operate on data fields, called planets, of the predetermined size $n = 9^3 = 729$. The objective of the Data Preprocessing is to fill these planets with data samples providing a high average information content, so called data entropy. It splits up in the following steps, shown in the upper half of figure 4.

1) Data Factory: First of all we have to produce the synthetic data sequences described in section I. To do so, we build random data streams for the Input Sequences and choose an appropriate generator function f to calculate the Output Sequence. Then we build the subsets $T \in \mathbb{N}$ for

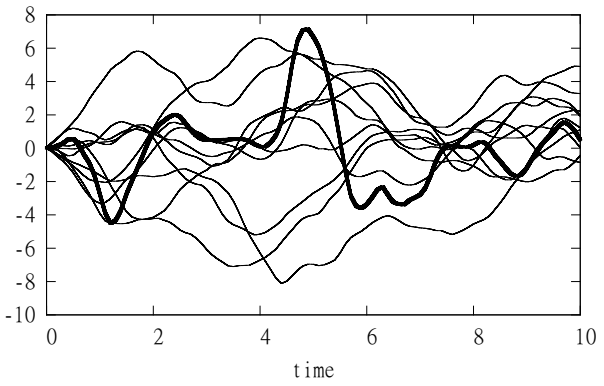

Figure 3. Input and output data series

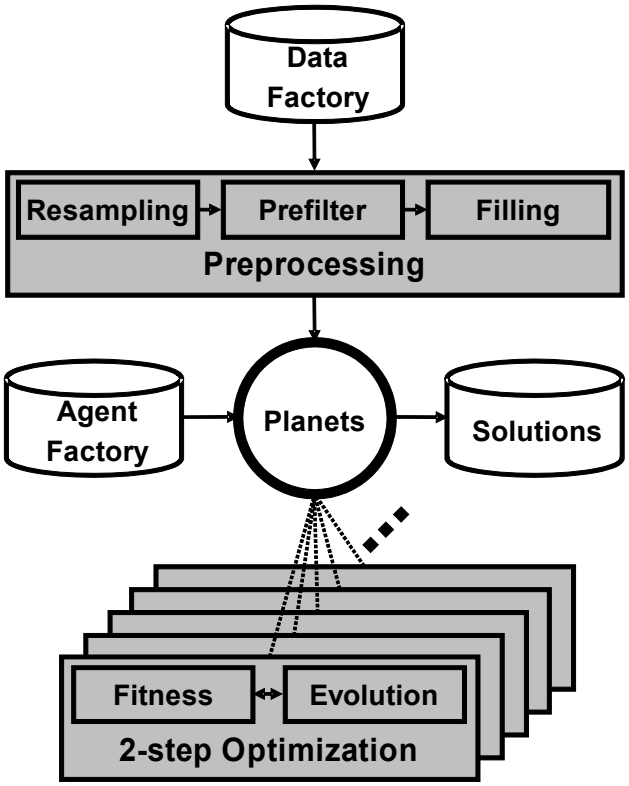

Figure 4. System overview

which we pass the information to our system, in order to imitate the incomplete information one has in real data.

2) Resampling: The data produced in the Data Factory has two main weaknesses: The samples are asynchronous and aperiodic. In order to get a time series of data samples we perform the following steps:

- Interpolation and FIR Filter (finite impulse response)
  For each sequence we interpolate the given values and smooth the result with a convolution.
- Error Correction
  The interpolated data is equalized with the original samples gained from the Data Factory.
- Downsampling
  We pick euklidian equidistant samples from each sequence and combine them to data samples with a timestamp.

During the learning process the data samples will not stay in their chronological ordering. To be able to perform time derivation, it is necessary to save the chronological neighbors for each sample.

After these two initial steps we have build a time series of equidistant data samples p each of which consists of a timestamp p_{time}, a vector $p_{\mathrm{data}} = [p_{\mathrm{out}}, p_{\mathrm{in}}]$, with $p_{\mathrm{out}} \in \mathbb{R}$ and $p_{\mathrm{in}} \in \mathbb{R}^m$, containing the output and input data and its chronological neighbors p^{pre} and p^{post}. With P we denote the set of all such data samples. Furthermore we define $p_{\mathrm{in}}^{\Delta} \in \mathbb{R}^m$ with

$$(p_{\mathrm{in}}^{\Delta})_j := \frac{1}{2}\left(\frac{(p_{\mathrm{in}})_j - (p_{\mathrm{in}}^{\mathrm{pre}})_j}{p_{\mathrm{time}} - p_{\mathrm{time}}^{\mathrm{pre}}} + \frac{(p_{\mathrm{in}})_j - (p_{\mathrm{in}}^{\mathrm{post}})_j}{p_{\mathrm{time}} - p_{\mathrm{time}}^{\mathrm{post}}}\right)$$

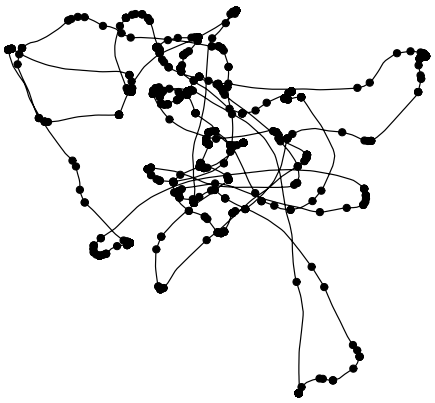

Figure 5. Weighted Random Prefilter: Samples in curved areas are chosen with a higher probability.

$$p_{\mathrm{out}}^{\Delta} := \frac{1}{2}\left(\frac{p_{\mathrm{in}} - p_{\mathrm{in}}^{\mathrm{pre}}}{p_{\mathrm{time}} - p_{\mathrm{time}}^{\mathrm{pre}}} + \frac{p_{\mathrm{in}} - p_{\mathrm{in}}^{\mathrm{post}}}{p_{\mathrm{time}} - p_{\mathrm{time}}^{\mathrm{post}}}\right).$$

3) Data Prefilter: In general the amount of data delivered by the data factory is too large for our framework, that is providing planets of a predetermined size. To choose the samples, which should be passed to the planets, we are using four different techniques.

No Prefilter The last 729 samples are passed to the planets.

Random Prefilter We choose the samples randomly. This approach serves as a reference, the other, more expensive techniques have to compete with.

Weighted Random Prefilter The samples are again chosen randomly, but the samples are chosen with different probabilities. This probability corresponds to the angle between $p_{\mathrm{in}}^{\mathrm{pre}} - p_{\mathrm{in}}$ and $p_{\mathrm{in}}^{\mathrm{post}} - p_{\mathrm{in}}$. The smaller this angle is, the more likely the sample is passed to the planets (figure 5).

k-means Prefilter We are are using a cluster algorithm to chose the samples. We have chosen *k-means* [5] for two reasons. With k-means we are able to determine the number of clusters to be build in a set of data, i.e. the data rate. Moreover the clusters generated by this algorithm are formed spherical, what is more suitable for our purpose, compared to e.g. density based clusters. We subdivided the data from the data factory in blocks. In each of these blocks we build a fixed number of clusters. Only the centers of these clusters were passed on.

4) Data Filling: In the last step of the data preprocessing the data samples are arranged on a 2D surface of a so-called planet (see Figure 6). The surface of the planets is built in a recursive pattern of squares containing nine elements, filled meander like. This method leads to the planet size $9^3 = 729$. This arrangement has the advantage, that the data set used for the local optimization consists of data samples, which may be spread more wideley across the input sequences.

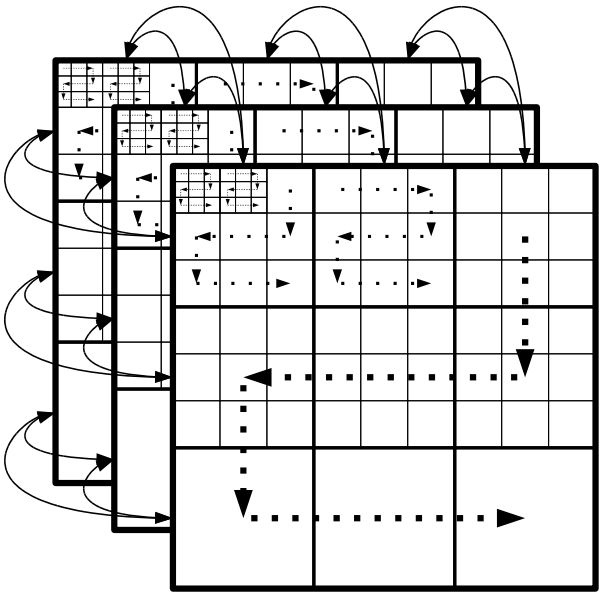

Figure 6. Planet-based data mapping and inter-planet paths

B. Multi-Agent-based Learning Strategy

The lower part of figure 4 contains the planets, the agent factory for filling the planets due to the planet configuration and the 2-step optimization algorithm dedicated to every planet.

1) Agent Description: The agents have 4 essential features: an age, an energy level, an area and their model function mf, approximating f. Moreover a replication mechanism is implemented, meaning the agents are able to produce a child and put it on an area. The age and the energy level are increased after each iteration. All operations an agent can perform, have an energy effort, by which the energy level is lowered, if the operation is executed. The area provides data samples to learn from and calculate the error of the model function. The model function $\mathrm{mf}\colon \mathbb{R}^m \to \mathbb{R}$ is stored in a tree representation (figure 7). This function is composed of elementary operations [8]. In the current test configuration the agents are allowed to use $\sin, +, *, /$ the variables $x_1, \ldots, x_m$ and a set of parameters within their model function. Furthermore the agents have the ability to learn from their local data and improve their model function by executing different *evolutionary operations* to change the structure of the model function, described in II-B2, and a local optimization algorithm to calibrate the parameters.

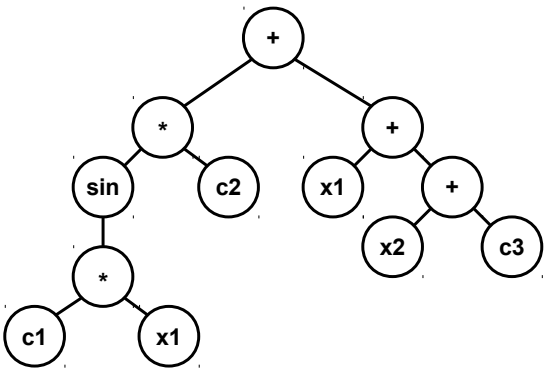

Figure 7. A tree representation of $c_2 \sin(c_1 x_1) + x_1 + x_2 + c_3$

In each iteration the software agents perform the following operations:

- *Calculate Fitness* The individual evaluates the error of his model function with respect to a chosen metric. According to this error the energy level is recalibrated. If it is negative, the agent is removed from the planet and his child, if present, is put on his position.
- *Move* The agent moves to another area, meaning the local test data he uses is modified, so that he can use new data in the next iteration. If the agent carries a child it is set to the former area. Agents in a multi planet system can travel with a small probability to different planets.
- *Local Optimization* The model functions parameters are improved by trying to reduce the error of the current local data with respect to the chosen search metric.
- *Evolutionary Operation* One of the evolutionary operations, explained below, is performed.
- *Nomination* The agents elect a few individuals with the highest fitness values and age on each planet. Next the global fitness value of these agents is calculated. The 25 best agents form the so-called *Elite Population*, containing the best dissimilar agents. The model function of the elite agents are evaluated on the whole data set. If any of these functions has an error below a certain error bound, the algorithm terminates and this function is returned. Copies of these agents are then spread across all planets to distribute their information to other agents.

2) Evolutionary Operations: The agents can perform four different evolutionary operations to produce a child:

- *Mutation* The agents model function gets changed randomly: Either a subtree of the model function is exchanged or new operations are inserted.
- *Crossover* When an agent moves it may happen that the chosen area is allready occupied with another individual. If that is the case, a subtree of the individuals model function is replaced by a randomly chosen, suitable subtree of the other agents model function.
- *Replication* The agent duplicates himself.
- *Global Optimization* The agents, which own enough energy or are not adult yet, optimize the parameters of their model function in the Memetic Coprocessor, explained below.

3) Memetic Coprocessors: The algorithm chosen for the *local* parameter optimization is resource-saving, because it is executed for all agents in every iteration. In

the Memetic Coprocessors, running on an extra processor core, we are executing more sophisticated algorithms for a *global* optimazation. In the current configuration we use a downhill-simplex algorithm [7].

C. Parallelization

Our framework provides the capability of running the evolutionary algorithm described above on several planets at the same time. If this is the case, some of the areas on each planet get marked as so called beam areas. After each iteration copies of all individuals placed on such an area are send to a randomly chosen area on a randomly chosen planet, provided the chosen area is not yet occupied by an agent. In the experiments 3 of the 729 areas on every planet were marked as beam areas. Our implementation associates each planet to one processor core, on an additional processor core a universe supervisor is executed. This supervisor manages the elite population using the data from all planets and controls the termination condition. The information exchange between the cores is implemented via a non-blocking Message Passing Interface.

III. Search Metrics

The agents can use different metrics serving as a measure for the error of their model functions mf. For a single data sample $p \in P$ we can calculate the error of the model function with these functions:

$$\mathrm{Abs}(p, \mathrm{mf}) := |p_{\mathrm{out}} - \mathrm{mf}(p_{\mathrm{in}})|$$

$$\mathrm{Euk}(p, \mathrm{mf}) := (p_{\mathrm{out}} - \mathrm{mf}(p_{\mathrm{in}}))^2$$

$$\mathrm{Log}(p, \mathrm{mf}) := \ln(1 + \mathrm{Abs}(p, \mathrm{mf}))$$

For an indexed subset $F \subset P$ we aggregate these values using

$$\mathrm{Mean}_f(F) := \frac{1}{|F|} \sum_{p \in F} f(\mathrm{mf}, p)$$

$$\mathrm{Max}_f(F) := \max_{p \in F} f(\mathrm{mf}, p)$$

$$\mathrm{PartialMean}_f(F) := \mathrm{Mean}_f(\bar{F})$$

$$\mathrm{PartialMax}_f(F) := \mathrm{Max}_f(\bar{F})$$

where f is one of the three functions described above and $\bar{F} \subset F$ is a set of randomly chosen data samples, with $|\bar{F}| = \lfloor \frac{1}{5}|F| \rfloor$.

For the next metric $\mathrm{CSDelta}(F)$ we choose a small offset $\delta \in \mathbb{R}_{\geq 0}$ and define $\delta_i \in \mathbb{R}^m$ to be the vector containing only zeros except for a δ at the i-th position and set $h = \lfloor \frac{|F|-1}{2} \rfloor$. With that vector we define:

$$K_{d,i}(F) := \left| \frac{(p_h)^{\Delta}_{\mathrm{out}}}{((p_i)^{\Delta}_{\mathrm{in}})_d} - \frac{\mathrm{mf}((p_h)_{\mathrm{in}}) - \mathrm{mf}((p_i)_{\mathrm{in}} + \delta_d)}{\mathrm{mf}((p_h)_{\mathrm{in}}) - \mathrm{mf}((p_h)_{\mathrm{in}} + \delta_d)} \right|$$

In most cases not all of the variables $x_1, \ldots, x_m$ are actually used in the model function. The function u pics the dimensions, which are really of interest:

$$\mathrm{u}(d) := \begin{cases} 1, & \text{if } x_d \text{ is used in } \mathrm{mf} \\ 0, & \text{else} \end{cases}$$

With $K_{d,i}$ and u we can define R:

$$R(F) := \sum_{d=1}^{m} \mathrm{u}(d) \left(\sum_{i=0}^{h-1} (K_{i,d}(F) + \mathrm{Abs}(p_i)) \right)$$

and

$$m := |F| \cdot \mathrm{Mean}_{\mathrm{Abs}}(F).$$

Finally we define CSDelta by:

$$\mathrm{CSDelta}(F) := \frac{R(F)\,(R(F) + m)}{|F|}$$

In the next section we discuss the influence of these metrics.

IV. Experiments

For all experiments we present in the following subsections the hardware configuration, the setup and the specific results.

A. Hardware

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.

B. Setup

In all experiments we used $f = 8\sin(2x_1) + x_1 + x_2 + 65$ as the generator function and produced 10 data input sequences with 100,000 elements each. We ran 100 iterations of all experiments with 8 Planets. We terminated a run, if the mean error became smaller than 0.001 or the runtime exceeded 15 minutes.

1) Prefilter: Moreover we started experiments in which we exchanged the prefilter module. We ran experiments with all prefilters described in (II-A3). In these experiments we used the $\mathrm{Mean}_{\mathrm{Euk}}$ metric.

2) Metrics: In a first set of experiments we studied the influence of the search metrics listed below.

- $\mathrm{Mean}_{\mathrm{Euk}}$
- $\mathrm{Max}_{\mathrm{Euk}}$
- $\mathrm{Mean}_{\mathrm{Abs}}$
- $\mathrm{Max}_{\mathrm{Abs}}$
- $\mathrm{PartialMean}_{\mathrm{Abs}}$
- $\mathrm{PartialMax}_{\mathrm{Abs}}$
- CSDelta

In these runs the prefilter was set to random.

C. Results

1) Data preprocessing: The results for the prefilters are shown in figure 8 and table I. K-means, the most expensive prefilter, has the highest median runtime and the highest standard deviation as well. So we can conclude that this technique is inappropriate for our problem. The best approach seems to be the weighted random prefilter, though the number of aborts is the highest, it has a median runtime significant smaller than all other prefilters.

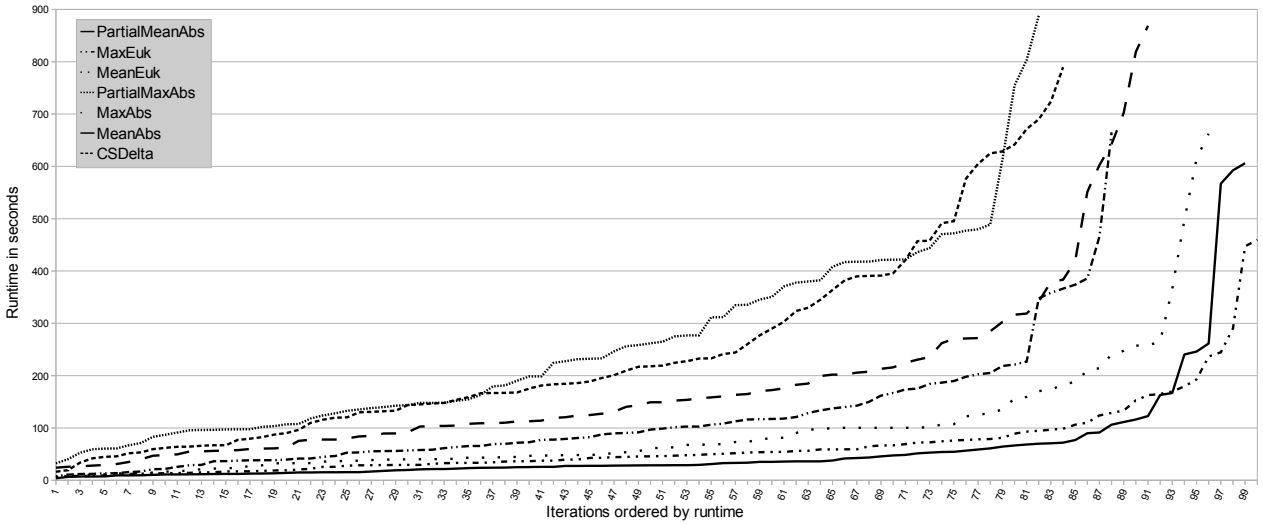

Figure 9. Comparison of the different search metrics

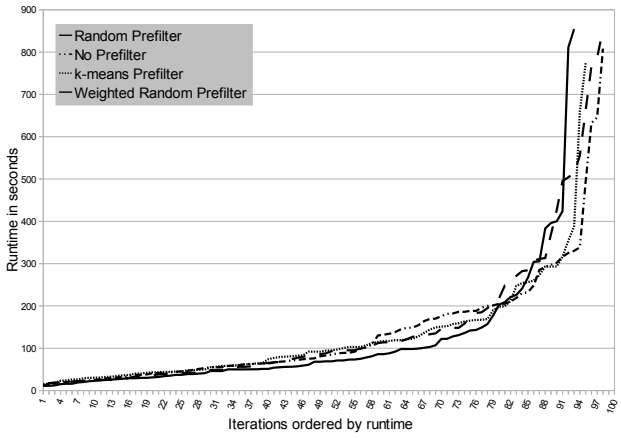

Figure 8. Comparison of the prefilters

Table I
RESULTS FOR THE PREFILTERS

Prefilter	Median	Min	Max	Std. Dev.	Aborts
Wgd. Random	61.07	11.05	854.18	141.44	7
No	82.07	14.10	808.19	137.95	2
Random	87.04	10.08	853.73	175.07	2
k-means	92.06	14.09	773.64	120.95	5

2) Search metrics: The results for the metrics are shown in figure 9 and table II. In figure 9 we combine the runtimes for different search metrics. This figure combines all three criteria: The runtime and its distribution as well as the detection rates. Although the two partial metrics require roughly $1/5$ compute power of the Max/Mean metrics the overall runtime is bad. We attribute this poor performance to the coarseness of the resulting target function. The complex heuristic takes advantage of a locally smoother optimization direction. The most complex metric (CSDelta) is not competitive in this experiment. However

Table II
RESULTS FOR THE METRICS

Metric	Median	Min	Max	Std. Dev.	Aborts
MeanAbs	28.56	3.87	605.92	105.38	1
MeanEuk	45.98	4.87	459.89	77.11	0
MaxAbs	54.64	6.35	662.44	113.73	4
MaxEuk	81.65	7.63	665.09	111.07	12
PartialMeanAbs	127.34	24.13	868.79	166.78	9
CSDelta	184.02	16.38	789.63	188.24	16
PartialMaxAbs	221.44	32.31	888.33	177.29	18

it should be mentioned that this one has a high potential for fine grade parallelization. But in the end the more simple metrics perform best.

V. SUMMARY AND FURTHER WORK

The identification of a process model is the basic requirement for successfully modelling and simulation. This paper presents experimental results for varying approaches within the preprocessing and the multi agent-based optimization steps of the process identifcation algorithm. We analyse and evaluate these results with regard to the overall identification characteristis and the performance behaviour. Furthermore every experimental result verifies the identification results itself and serves as a prove of concept. The preprocessing results emphasize the relevance of efficient data conditioning and show the impact of intelligent data compression like the Weihgted Random Prefilter.

The different metrics used in the optimization step points up the influence of the consideration from neighbourhood correlations.

The further work has two key aspects of activity: Parallelization and multi-metrics.

In the current version the 2-step multi-agent-based algorithm is realized with a modular and scalable architecture providing efficient parallelization of the identification process. To speed-up the run time further we will focus on

special architectures for massively parallelization.
The multi-metric model will realize an automatic switch to other metrics based on the data configuration and the local data condition to improve the benefit of the optimization step.

REFERENCES

[1] Sebastian Bohlmann, Volkhard Klinger, and Helena Szczerbicka. 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.*, 2009.

[2] Sebastian Bohlmann, Volkhard Klinger, and Helena Szczerbicka. 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, July 2010.

[3] Sebastian Bohlmann, Volkhard Klinger, and Helena Szczerbicka. System Identification with Multi-Agent-based Evolutionary Computation Using a Local Optimization Kernel. In *Submitted to ICMLA 2010 (International Conference on Machine Learning and Applications)*, 2010.

[4] Sebastian Bohlmann, Volkhard Klinger, Helena Szczerbicka, and Matthias Becker. A data management framework providing online-connectivity in symbiotic simulation. In *24th EUROPEAN Conference on Modelling and Simulation, Simulation meets Global Challenges*, Kuala Lumpur, Malaysia, June 2010.

[5] Tapas Kanungo, David M. Mount, Nathan S. Netanyahu, Christine D. Piatko, Ruth Silverman, and Angela Y. Wu. An efficient k-means clustering algorithm: Analysis and implementation. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 24:881–892, 2002.

[6] Averill M. Law and W. David Kelton. *Simulation Modeling and Analysis*. McGraw-Hill, 2000.

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

[8] Michael Schmidt and Hod Lipson. 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, 2007. ACM.

[9] Paavo Viitamki. *Hybrid modeling of paper machine grade changes*. PhD thesis, Helsinki University of Technology, Espoo, Finland, 2004.

AUTHOR BIOGRAPHIES

SEBASTIAN BOHLMANN is a Ph.D. candidate at Department of Simulation and Modelling - Institute of Systems Engineering at the Leibniz Universität 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>.

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>.

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

HELENA SZCZERBICKA is head of the Department of Simulation and Modelling-Institute of Systems Engineering at the Leibniz Universität 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, modelling methodology, queuing theory, stochastic Petri Nets and distributed simulation. Her email address is <hsz@sim.uni-hannover.de>.