Black-box identification of discrete event systems with optimal partitioning of concurrent subsystems

Matthias Roth, Jean-Jacques Lesage, Lothar Litz

Black-box identification of discrete event systems with optimal partitioning of concurrent subsystems

Matthias Roth, Jean-Jacques Lesage, *Member, IEEE*, and Lothar Litz, *Member, IEEE*

***Abstract*—This paper proposes a data-driven method to determine concurrent parts in Discrete Event Systems (DES). The aim is to improve the results of black-box identification methods without considering any system information except of observed data. In order to allow an analysis of the collected data, the impact of concurrency on the exhibited system data is determined by two criteria. We propose to use an optimization algorithm that isolates concurrent parts of the system by minimizing concurrency expressed by the two proposed criteria within the determined subsystems. A lab-size application shows the potential of the method for real-world manufacturing systems. The aim is to deliver optimal identified models for fault detection and isolation.**

I. Introduction

Increasing the dependability of modern industrial systems like manufacturing or production systems is a constant concern in the control community. Among others, fault detection and isolation (FDI) plays a key role to achieve sufficiently reliable systems and to reduce costly downtimes. Modern FDI-techniques are often model based. The idea is to compare an observed and a modeled system output in order to detect faults in the system. For systems that can be modeled by Discrete Event Systems (DES) the diagnoser-approach is a very prominent example for the class of model-based methods [1]. It uses models that explicitly contain faulty system behaviors. As an advantage, this approach is capable of giving guarantees concerning the diagnosabilty of a system. Since each fault that has to be detected must be explicitly modeled, drawbacks are the expensive model building process and the inability to detect faults that are not part of the model. An approach that tries to avoid these drawbacks is presented in [2]. In this approach a fault-free system model is used to detect and localize faults. Since the model does not contain any faulty system behavior, it is possible to obtain the model by identification. The fault detection policy of this approach is that each observed behavior that is not reproducible by the model is considered as a fault. Although diagnosability cannot be guaranteed in this approach it has proven to be useful for real-world applications like shown in [2].

This work was supported by a grant from "Région Ile de France"

Matthias Roth is with the Institute of Automatic Control at the University of Kaiserslautern, Germany, and the LURPA, Ecole Normale Supérieure de Cachan, France. (mroth@eit.uni-kl.de)

J.-J. Lesage is with the LURPA at the Ecole Normale Supérieure de Cachan, F - 94235 Cachan Cedex, France (lesage@lurpa.ens-cachan.fr)

L. Litz is the head of the Institute of Automatic Control at the University of Kaiserslautern, D – 67653 Kaiserslautern, Germany (litz@eit.uni-kl.de)

If systems with a high degree of concurrency are considered in the identification approach, it can take very long to observe a sufficiently complete amount of data that yields an appropriate FDI-model. If the model is identified on an incomplete data basis, an unacceptable high number of false alerts occur during online monitoring [3]. In order to deal with an incomplete data base and to avoid false alerts, a distributed framework has been proposed in [3]. The main idea of this approach is to divide a given system into concurrent subsystems and to use a single model for each of these subsystems. In [3], appropriate subsystems have been defined in a heuristic way. In this paper, we propose an approach to automatically divide a system into concurrent subsystems using an optimization algorithm that reduces concurrency within each subsystem.

The paper is structured as follows: in section II, the distributed framework of [3] is motivated and shortly explained. Section III analyses how concurrency is reflected in observed system data. The automatic choice of concurrent subsystems using an optimization approach is explained in section IV. The practical relevance of the proposed method is shown section in V. Section VI contains some concluding remarks and an outlook.

II. Distributed Identification

In this work we consider closed loop DES. The closed loop consists of a controller and a plant, which is a typical configuration for industrial production systems. The identification is based on controller I/O (input/output) vectors collected during p different system cycles.

Definition 1 : The j-th I/O vector in the h-th of p system cycles is defined as $u_h(j) = (I_1(j), ..., I_s(j), O_1(j), ..., O_m(j))_h$ with I_1, .., I_s and O_1, .., O_m $\in \{1, 0\}$ denoting the considered inputs and outputs of the controller of the closed loop system. Inputs and outputs are referred to as "I/Os".

Definition 2 : If during the h-th system cycle, l_h I/O vectors have been observed, the sequence is denoted as $\sigma_h = (u_h(1), u_h(2), ..., u_h(l_h))$.

Assumption 1 : Each I/O vector is created by a new event such that $u_h(j) \neq u_h(j+1) \forall 1 \leq j < l_h$.

This assures that two successive I/O vectors differ in at least one I/O value. If each observed I/O vector is considered as a letter of an alphabet we can define the set of words with length q observed up to the t-th of p different system cycles.

A word represents an observed I/O vector sequence.

Definition 3 : The observed words of length q observed up to the t-th system cycle are denoted as $W_{\mathrm{Obs}}^{q,t} = \bigcup_{i=1}^{t} \left(\bigcup_{j=1}^{l_i - q + 1} \left(u_i(j), u_i(j+1), ..., u_i(j+q-1) \right) \right)$

With this definition it is possible to describe the behavior of a system by the language built on the basis of the observed words.

Definition 4 : The observed language of length n is $L_{Obs}^{n} = \bigcup_{i=1}^{n} W_{Obs}^{i,p}$ with p denoting the number of observed system cycles.

In this work, we assume that L_{Obs}^{n} has been observed during fault-free system cycles and thus is finite. Based on the observed language L_{Obs}^{n} it is possible to identify a monolithic automaton. The algorithm in [4] allows constructing an automaton on the basis of observed words of the parametric length k. The identified automaton is able to produce the observed language of the system and is k+1-complete ($L_{Obs}^{k+1} = L_{Ident}^{k+1}$ with L_{Ident}^{k+1} denoting the language of the identified automaton) [5]. Since for real systems with a high degree of concurrency it is by experience not possible to observe all fault-free words (I/O vector sequences) within a reasonable time, the cardinality of the word set typically evolves like shown in Fig. 1. The solid line that represents a system with a low degree of concurrency converges to a stable level after a short time. The dashed line does not stop growing. This indicates that the system language has not yet been completely observed and that new words will occur when the observation is continued. If a k+1-complete automaton is identified on such an incomplete data base, it will not be capable of reproducing the expected new fault-free I/O-vector sequences of length k+1 if they have not been observed before. This effect leads to false alerts using the fault detection policy that each non-reproducible behavior is considered as a fault.

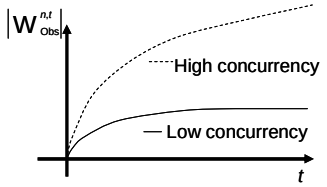

Fig. 1: Evolution of the number of observed words of length n over system cycles h

In order to avoid an unacceptably high number of false alerts, in [3] a distributed framework has been proposed. The idea is to divide a system in concurrent subsystems. For a properly chosen subsystem the observed language cardinality typically converges to a stable level within a short time due to the reduced concurrency in the subsystem. Hence, it is possible to identify a partial automaton for each subsystem that does not lead to false alerts when observing the according subsystem. In [3] it has been shown that there exist faults that cannot be detected by a single partial automaton. In order to detect faults that lead to a forbidden combined behavior of the partial automata, the framework depicted in Fig. 2 has been introduced. An upper structure consisting of the so called Permissive Observed Cross Product (POCP) and a given tolerance specification allows restricting the combined behavior of the partial automata such that even faults leading to a forbidden automata network behavior can be detected. Algorithms to construct the POCP and guidelines to design the tolerance specification are given in [3].

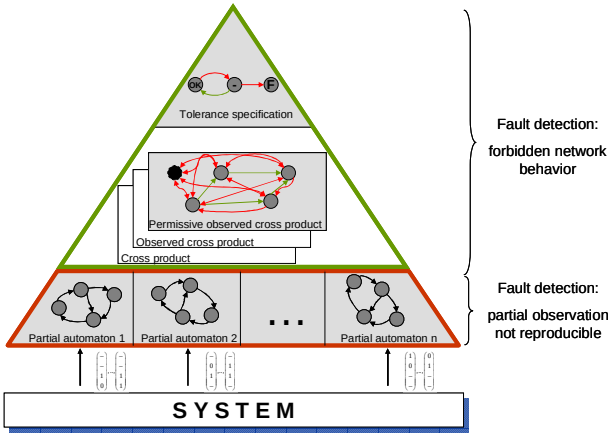

Fig. 2: Distributed framework from [3]

In [3] the necessary system partitioning has been performed in a heuristic way. The I/Os of the I/O vector have been assigned to different subsystems considering the physical system structure. This approach is usually not possible if black-box identification is performed. An approximation of the number of possible partitions is given by the Stirling number of the second kind [6]:

$$S(m,n) = \frac{1}{n!} \sum_{i=0}^{n} (-1)^i \binom{n}{i} (n-i)^m$$ with m denoting the number of I/Os and n denoting the number of non-empty partitions. In the case of 30 I/Os and three partitions (three subsystems and three identified partial automata), this leads to $2.06 \cdot 10^{14}$ possible solutions. Only a small number of these solutions lead to subsystems with reduced concurrency which allows a complete observation of the according subsystem language in short time. Assigning I/Os to appropriate subsystems is an NP-complete combinatorial problem. To facilitate finding such a partitioning, the rest of the paper introduces an automatic way to assign I/Os to concurrent subsystems by a heuristic optimization technique.

III. Manifestation of Concurrency in the Observed System Language

A. General considerations

In section II it has been lined out that we are interested in finding subsystems with a low degree of concurrency in order to avoid false alerts using the identified models for FDI. The concurrency we are dealing with is strongly

connected to the non-determinism of the physical plant in the considered closed loop DES. Due to temporal non-determinism, a physical action in the plant does not always take exactly the same time to finish. If several actions are performed in a parallel way, this non-determinism leads to several possible combined outcomes: in one system cycle a given action can finish faster than another parallel action and vice versa. This leads to numerous possible evolutions. Since we do not explicitly consider the timed behavior, its variations are only of interest if several actions are performed in a parallel way, which is considered as a concurrent behavior.

Since we want to spot concurrent behavior in the considered system using a black-box identification approach the only available information is the data that has been collected when observing different system evolutions. Hence, it must be analyzed how concurrency is reflected in the observed data. In the next two subsections, we present two phenomena that can be observed when analyzing the observed data and that are strongly related to concurrency.

B. Language growth

A first phenomenon that occurs due to concurrency has already been mentioned in section II. If a concurrent system is observed, the growth of the observed language cardinality is related to the degree of concurrency. Fig. 1 shows typical cardinality evolutions for systems with low and with high concurrency. The reason for the language growth can be seen in Fig. 3. If we represent the behavior of the concurrent system to be identified by a Petri net with two concurrent sequences there is only a limited number of possible evolutions to "move" the tokens through the net (left Petri net). If the system is represented by a Petri net with more parallel branches (right Petri net), then there are more possible system evolutions. Hence, it takes longer until the system language is completely observed which makes the cardinality converging to a stable level.

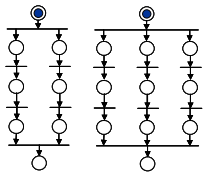

Fig. 3: Sketches of Petri nets with different degrees of concurrency

Following Definition 3 it is directly clear that $W_{Obs}^{n,t} \supseteq W_{Obs}^{n,t-1}$. Hence, if $|W_{Obs}^{n,t}| - |W_{Obs}^{n,t-1}| > 0$ (word set observed up to cycle *t* is larger than the word set up to cycle *t*-1) then some new words that have not been seen before must have been observed in the *t*-th system cycle.

C. Branching Degree

The second phenomenon is related to the structure of an automaton identified on the basis of observed system data. To facilitate the understanding of this phenomenon we shortly review the monolithic identification procedure from [4]. Suppose the following three fault-free system evolutions have been observed.

$$\sigma_1 = \left(\begin{pmatrix}1\\0\\0\\1\end{pmatrix}, \begin{pmatrix}0\\1\\0\\1\end{pmatrix}, \begin{pmatrix}0\\1\\1\\0\end{pmatrix}\right),\ \sigma_2 = \left(\begin{pmatrix}1\\0\\0\\1\end{pmatrix}, \begin{pmatrix}1\\0\\1\\0\end{pmatrix}, \begin{pmatrix}0\\1\\1\\0\end{pmatrix}\right),\ \sigma_3 = \left(\begin{pmatrix}1\\0\\0\\1\end{pmatrix}, \begin{pmatrix}0\\1\\1\\0\end{pmatrix}\right)$$

The identification algorithm of [4] with k=1 basically consists of associating each I/O vector to an automaton state and to connect states containing I/O vectors that have been observed successively. The result is a non-deterministic autonomous automaton with output (NDAAO):

Definition 5 : NDAAO = $(X, \mathbf{\Omega}, r, \lambda, \mathrm{x}_0)$ with X denoting a finite set of states, $\mathbf{\Omega}$ the output alphabet, $r : X \to 2^{\mathrm{X}}$ the non-deterministic transition relation, $\lambda : X \to \mathbf{\Omega}$ the output function and x_0 the initial state.

The result of the monolithic identification can be seen in Fig. 4 on the left side. The automaton is able to reproduce each of the observed sequences. It can be seen that the initial state has three leaving transitions that are necessary to reproduce the different observed following behaviors. In the example we assume that the global system consists of two concurrent subsystems: The first subsystem consists of the first two I/Os of the I/O vector and the second system consist of the other two I/Os. Since the changes in value of I/Os belonging to different subsystems happen concurrently, three following behaviors of the first state are possible. In one following behavior the first subsystem leads to earlier I/O changes (σ_1). In the next case, the second subsystem evolves faster (σ_2) and in the third case the two systems evolve simultaneously (σ_3).

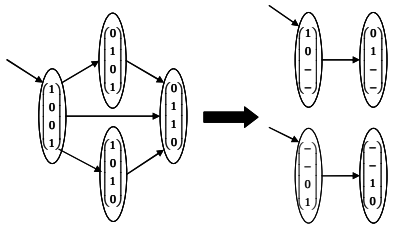

Fig. 4: Result of the monolithic and distributed identification

If the I/Os of each single subsystem are considered separately, the identified partial automata are much simpler as depicted on the right side of Fig. 4. The global output of the partial automata can be calculated by combining the two partial outputs such that "-"-symbols in one partial automaton are overwritten by I/O values from the other partial automaton [3].

Initial states of the partial automata have only one leaving transition representing one following behavior. This is a result of the reduced concurrency in the single subsystems. This observation shows that an identified automaton reflects concurrency of the considered system by the number of direct following states in the automaton. More generally, the identification of a monolithic automaton can be seen as an approximation of the reachability graph of a Petri net representing the considered system. If a system has a low

degree of concurrency, the reachability graph does not have many states with several leaving transitions. In the next section we give a criterion how this "branching degree" of an NDAAO can be quantified.

IV. Determination of Concurrent Subsystems Using an Optimization Method

A. Optimization approach

In section III it has been shown that concurrency has certain effects on the collected data or on models identified using this data. The idea of the optimization approach is to quantify these effects and to develop appropriate optimization criteria in order to find optimal subsystems that have a minimal degree of internal concurrency.

As shown in section II, partitioning the controller I/Os into appropriate subsystems is a combinatorial problem. An optimization technique that is well suited for solving this class of problems is simulated annealing [7]. It is a metaheuristic optimization approach inspired by the annealing process in metallurgy and describes a possible way to treat NP-complete combinatorial problems in practice. Slow cooling in metallurgy leads to a crystal structure that has a low internal energy which is advantageous for material properties. The optimization algorithm tries to perform a similar "slow cooling" in order to find a solution with a minimal metaphorical energy (minimized optimization criterion). The algorithm is sketched in Fig. 5.

```
procedure simulated annealing
 begin
    initialize T
    select current solution y_c at random
 repeat
    select a new solution y_new
    if eval(y_c)>eval(y_new)
       then y_c←y_new
    else if random[0,1)< e^((eval(y_c)-eval(y_new))/T)
       then y_c←y_new
    T←T*coolingRate
 until T<T_min
```

Fig. 5: Structure of simulated annealing according to [7]

It is necessary to parameterize the algorithm with the starting temperature T, the cooling rate and the minimum (stop) temperature T_{min}. The algorithm must be adapted to our optimization problem in two points: First, the structure of a possible solution *y* must be defined. Second, a method to select a new solution must be given.

Definition 6 : A solution *y* is a map that associates each I/O to one and only one of *n* subsystems $y : \mathrm{I/O} \rightarrow [1, n]$.

With this definition it is assured that each I/O can only be associated to one subsystem. The algorithm to select a new solution based on the current solution y_c is given in Fig. 6. First, the algorithm copies the current solution to y_{new}. Then it chooses *m* I/Os randomly. *m* is another parameter that must be given to the algorithm. The smaller *m*, the more similar the new solution y_{new} to the old solution is. To each of the chosen I/Os (consI/Os), the algorithm randomly assigns a new subsystem. In order to evaluate the quality of a solution, two optimization criteria are introduced in the next section.

```
procedure select a new solution
   y_new← y_c
   Choose m I/Os randomly → consI/Os
   For each I/O in consI/Os:
       repeat
           y_new(I/O)=random[1,n]
       until y_new(I/O)≠y_c(I/O)
```

Fig. 6: Algorithm to choose a new solution

B. Optimization criteria

In order to develop appropriate optimization criteria, several definitions are necessary. First, the relation of subsystems, a solution *y* (Definition 6) and I/O vectors is defined.

Definition 7 : The *i*-th subsystem sys_i is based on partial I/O vectors $u_{h,sys_i}(j) = (I_1(j), ..., I_s(j),\ O_1(j), ..., O_m(j))_h$ with $I_1,..,I_s$ and $O_1,..,O_m \in \{1, 0, -\}$ where I/Os not belonging to the *i*-th subsystem ($y(\mathrm{I/O}) \neq i$) are assigned with "-".

Assumption 2 : Each partial I/O vector u_{h,sys_i} is created by a new event in sys_i such that $u_{h,sys_i}(j) \neq u_{h,sys_i}(j+1)$.

With Assumption 2 it is assured that two successive partial I/O vectors differ in at least one I/O value.

The **first optimization criterion** is a formalization of the language growth effect described in section III.B. It counts for each subsystem the newly observed words in each new system cycle after the first one and multiplies this number with the square root of the according cycle:

$$E_1(y) = \sum_{\forall sys_i} \sum_{t=2}^{p} \sqrt{t}\,(|\,W_{Obs,sys_i}^{n,t}\,| - |\,W_{Obs,sys_i}^{n,t-1}\,|) \quad \text{with} \quad W_{Obs,sys_i}^{n,t}$$

denoting the word set of length *n* up to the *t*-th system cycle of subsystem sys_i (definition of $W_{Obs,sys_i}^{n,t}$ is straight forward following Definition 3). If there is a high degree of concurrency in one of the subsystems sys_i, then the according language growth will be important and thus lead to high values of $E_1(y)$. The term $\sqrt{t}$ was heuristically chosen and adds more weight to new words that occur at late system cycles in order to represent the fact that we want a fast convergence to stable language cardinality.

The **second optimization criterion** analyses the structure of NDAAOs identified for the subsystems. It formalizes the branching degree introduced in section III.C. As explained in section III.C, the number of following states in the identified automaton depends on the concurrency of the considered system. We now define a measure to express this branching degree:

Definition 8 : The branching degree *BD* of the identified $NDAAO_{sys_i}$ of subsystem sys_i is defined by

$$BD(NDAAO_{sys_i}) = \sum_{\forall x \in X} \begin{cases} 0 \text{ if } |\,r(x)\,| \leq 1 \\ |\,r(x)\,| - 1 \text{ if } |\,r(x)\,| > 1 \end{cases}.$$

It counts for each state of the considered NDAAO the leaving transitions. Only states with more than one leaving transition contribute to this measure since this represents possible concurrent behavior. In states with more leaving transitions, we subtract one from the number of transitions. This is done since one following transition in a state does not represent concurrency. In the example of Fig. 4, the monolithic model has a *BD* of 2 since only the first state contributes to the measure and has three leaving transitions. Each distributed model on the left side of Fig. 4 has a *BD* of 0 since each state has at most one leaving transition.

Note that the branching degree given in Definition 8 is an absolute measure and is not normalized to the size of the automaton. It only considers states that have several possible following behaviors due to a possible concurrency. Normalization to the size of the automaton (e.g. state space) could lead to a situation where the branching degree of two automata are different even if both automata contain the same number of states with more than one leaving transition and thus reflect the same degree of concurrency.

An optimization criterion based on the branching degree can be given as $E_2(y) = \sum_{\forall sys_i} BD(NDAAO_{sys_i})$ with $NDAAO_{sys_i}$ denoting the automaton identified with the algorithm of [4] for subsystem sys_i. The branching degree of the subsystem NDAAOs are summed up to get a measure for the concurrency of solution *y*.

Note, it is possible to have *BD*>0 and thus $E_2(y) > 0$ although there is no concurrency in the system. If there are several "decisions" in the system (e.g. large OR small work piece), it is possible to have several following states of a given automaton state. Since we only want to *minimize BD* we can cope with $BD_{\min}$>0.

With the definition of the optimization criteria it is now possible to perform the optimization with the algorithm from Fig. 5. The function "eval" implements one of the proposed optimization criteria. In the case of the second criterion, an own NDAAO for each subsystem has to be identified before the branching degree can be determined.

V. Application

A. Case study

The proposed method has been applied to a lab manufacturing system like shown in Fig. 7. It has 30 binary controller I/Os. During one production (or system) cycle the plant treats three work pieces. If the system is to be divided in subsystems, a heuristic solution is to group I/Os according to the three machine tools. Within the subsystems shown in Fig. 7 there are almost no concurrent actions. 62 system cycles have been observed. It could be seen that the system language of length 2 does not converge to a stable level [3]. Hence, the proposed distributed approach is to be used to identify appropriate models to monitor the systems without false alerts. In order to determine the subsystems based on the observed data, the proposed optimization approach is used.

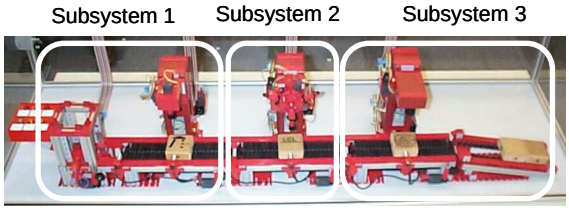

Fig. 7: Considered lab-system with heuristically chosen subsystems

B. Evaluation of the optimization results

To demonstrate that the optimization approach leads to useful results for real systems, it is necessary to perform an evaluation. An evaluation is only possible if a "good" solution is already known like for the system from Fig. 7. The good solution is to be compared to the automatically generated ones. Of course, any information concerning good or bad solutions is not available in general. Here it is only used to demonstrate the performance of the method. To decide upon the quality of an automatically generated solution by comparing it to predefined "good" results, two similarity criteria are introduced that are necessary to compare an automatically generated and a given solution:

1. Absolute number of I/Os that are shared by an automatically generated and a predefined subsystem (similarity criterion 1).

We count I/Os that are shared by a predefined and an automatically generated subsystem. The higher this value is, the more similar both subsystems are. Since this criterion does not take into account the number of "wrong" I/Os in the subsystems, we introduce a second similarity criterion:

2. Relative number of I/Os that are shared by an automatically generated and a predefined subsystem (similarity criterion 2).

In this criterion we take the value from the first criterion and divide it by the number of I/Os in the predefined subsystem.

TABLE 1: EXAMPLE OF THE EVALUATION CRITERIA

		Automatically generated subsystems		
		A 6, 9	B 1, 3, 7, 8	C 2, 4, 5
Predefined subsystems	I: 1, 2	0 (0%)	1 (50%)	1 (50%)
	II: 3, 4, 5, 6	1 (25%)	1 (25%)	2 (50%)
	III: 7, 8, 9	1 (33%)	2 (67%)	0 (0%)

Table 1 shows an example of the similarity criteria. It shows for example that in the predefined subsystem III there are three I/Os 7, 8 and 9 and in the automatically determined subsystem B there are four I/Os 1, 3, 7 and 8. The absolute number of I/Os that are shared by both subsystems is also given (2 in the case of III and B). Similarity criterion 2 is given in brackets (67% in the case of III and B).

The two criteria are used to compare a set of predefined and automatically determined subsystems. For each

automatically determined subsystem we take the maximum value of its column in the table. This assures that we compare it with the most similar predefined subsystem. Determining the maximum we first consider similarity criterion 1. If there are more cells with the same value, we decide upon similarity criterion 2. In the example we get the comparison pairs A→III, B→III and C→II. The predefined subsystem I is not compared to any automatically generated subsystem since each generated solution is more similar to another predefined one. The first similarity criterion of the complete setting is calculated by summing up all shared I/Os. For the example the subsystems A, B, C share 1 (A→III) plus 2 (B→III) plus 2 (C→II) I/Os equals to 5 I/Os with the according predefined subsystems. Hence the first similarity criterion for the setting in table 1 is 5. The second similarity criterion is determined by calculating the average of the similarity percentages. For the example we get 33% (A→III) plus 67% (B→III) plus 50% (C→II) divided by three equals 50%.

Fig. 8 shows the results of an optimization run using the criterion "language growth" (E_1 in section IV.B) that was parameterized to deliver three subsystems. The starting temperature for simulated annealing was 1000 and the cooling rate was determined such that the optimization stops after 1000 iterations. New solutions are determined with the algorithm in Fig. 6 with m=3. For the lab system such an optimization takes between 15 and 20 minutes on a normal desktop PC. For some automatically determined solutions the similarity criteria have been calculated in order to determine their distance to the predefined solution consisting of the three subsystems (tools) explained in section V.A. It can be seen that solutions with good (small) values for E_1 are very similar to the predefined solution. The optimization delivers a solution consisting of three subsystems with 25 from 30 I/Os correctly assigned (more than 83% similarity). For the optimization only 30 of 62 system cycles have been used. In none of the automatically determined subsystems a new word is observed when considering the remaining 32 system cycles for validation. Hence the subsystems are suitable to be used in the distributed framework to monitor the system.

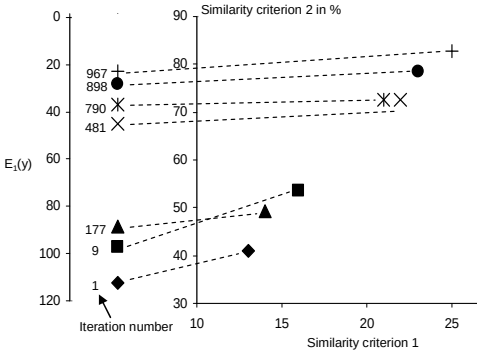

Fig. 8: Optimization results for $E_1(y)$ (language growth)

Fig. 9 shows results using the second optimization criterion (branching degree, starting temperature 1000, 1000 iterations, m=3). The optimization takes between 20 and 25 minutes since the identification of automata takes longer than the determination of the language growth. The best solution is to more than 93% similar to the predefined one. Again, only 30 of 62 system cycles have been used. The validation using the remaining collected cycles showed that the automatically determined solution is also appropriate for online-monitoring of the lab system.

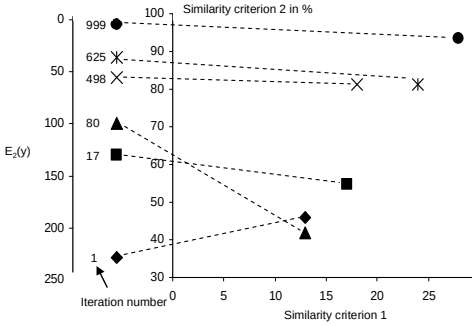

Fig. 9: Optimization results for $E_2(y)$ (branching degree)

With the best automatically determined solutions we performed tests for the lab-system. It showed that false alerts did not occur. We were also able to detect intentionally built-in faults using the approach from [3]. Faults at I/Os that have been assigned to a "wrong" subsystem often led to fault detection by the upper structure of Fig. 2.

VI. Conclusions

An approach to divide a DES into concurrent subsystems based on observed data has been proposed. One of the parameters for the optimization is the number of subsystems that are to be build. Current work aims at determining this number based on observed data and online calculation constraints from the FDI approach in [3].

References

[1] M. Sampath, R. Sengutpa, S. Lafortune, K. Sinnamohideen, D.C. Teneketzis, "Failure Diagnosis using Discrete-Event Models", IEEE Trans. on Control Systems Technology, Vol. 4, No. 2, pp. 105-124, March 1996.

[2] M. Roth, J.-J. Lesage, L. Litz: "An FDI Method for Manufacturing Systems Based on an Identified Model", *13th IFAC Symposium on Information Control Problems in Manufacturing*, INCOM'09, Moscow (Russia), pp. 1389 - 1394, June 3-5 2009

[3] M. Roth, J.-J. Lesage, L. Litz: "Distributed identification of concurrent discrete event systems for fault detection purposes", *European Control Conference 2009*, ECC 2009, Budapest (Hungary), August 23-26 2009

[4] S. Klein, J.-J. Lesage, L. Litz, "Fault detection of Discrete Event Systems using an identification approach", *16th IFAC World Congress*, CDROM paper n°02643, 6 pages, Praha(CZ), July 4-8, 2005

[5] T. Moor, J. Raisch, and S. O'Young, "Supervisory control of hybrid systems via l-complete approximations", *in Proc. of the IEE fourth Workshop on Discrete Event Systems WODES'98*, Cagliari, Italy, August 1998, pp. 426-431

[6] J.Riordan, *An Introduction to Combinatorial Analysis*, New York: Wiely, 1980

[7] Z. Michalewicz, D. B. Fogel: *How to Solve It: Modern Heuristics*, New York, Springer-Verlag: 2000