Automated Detection of Production Cycles in Production Plants using Machine Learning

Andreas Bunte*, Henrik Ressler* and Natalia Moriz*
*inIT - Institute industrial IT, Ostwestfalen-Lippe University of Applied Sciences and Arts, Lemgo, Germany
Email: {andreas.bunte, henrik.ressler, natalia.moriz}@th-owl.de

***Abstract*—Data-driven algorithms can be used to derive new information from data. In modern production plants, this can be used to reduce manual effort, e.g. to create a behavior model. In this work, one offline and one online algorithm are introduced that can determine the production cycles automatically. The algorithms use learned automaton to detect production cycles. A first evaluation is presented, which points out differences of the algorithms. However, overall the results are promising.**

I. Introduction

Due to the evolution of production plants towards Industrie 4.0, the plants will be more adaptable, predictable, and efficient. Artificial Intelligence (AI) has a major role since it supports these goals, e.g. through orchestration, predictive maintenance, or optimization algorithms [1], [2], [3]. Up to now, the algorithms are specialized for a production plant and need some manual effort for adapting to a different plant. But the goal of the AI usage is to generate information based on the data-driven algorithms, which reduces the manual effort.

The work at hand introduces two algorithms for the detection of production cycles (PCs), which is important information for many different purposes. PCs indicate a periodic cycling loop, where typically one product is produced in each loop. If the PC is known, properties, such as the cycle time or the throughput, can be determined. The automatic detection is useful because this information is needed by other algorithms. For example, if an optimization should be performed, its objective function has to be calculated for every PC. So, the PC is has to be detected automatically or set manually.

The contributions of this work are two novels automatic cycle detection (ACD) algorithms that can detect PCs out of discrete data. One ACD algorithm uses online data, the other uses offline data, so they are feasible for different types of applications. Both algorithms rely on the Online Timed Automaton Learning Algorithm (OTALA) that learns automata based on discrete signals [4]. Furthermore, an evaluation is presented on a real-world production plant, where the results of both algorithms are compared.

This paper is structured as follows. First, related works are presented in section II. Second, the concept of the ACD algorithms are described in section III. In section IV the results of the evaluation are shown. Finally, we conclude the work.

II. Related Works

The detection of cycles in a directed graph, as learned by OTALA, can be performed by the well-known *Tortoise and Hare* algorithm. It allows the detection of cycles with a great performance and it terminates using an extension [5]. Another approach is the Depth-First-Search (DFS), as stated in [6]. However, the challenge in the detection of PCs is not to find cycles of the graph. It is to find the cycle that represents the PC and therefore such algorithms are not appropriate.

The detection of frequent unknown patterns in time series gathered from sensors is the main task of the motif discovery methods and can also indicate the PC. [7] gives a review of existing methods in time series motif discovery and summarized their advantages and disadvantages. Challenging tasks are amongst others: definition of the length of motifs, handling data streaming, and maintaining time complexity. For the PC detection, the definition of the length of motifs and handling data streaming are the crucial criteria. According to [7], only the method from Li et al. [8] can handle both. This method can be applied to one-dimensional data. Thus, multi-dimensional data has to be pre-possessed to convert it to one-dimensional data or detect motifs in each dimension separately.

The learning of parallel automata from historical data is presented in [9]. Instead of a single automaton, the approach in [9] provides a new method (ComPACT) for creation of several automata. But this method has two critical points relating to our purpose: first, the ComPACT needs a similarity measure to determine independent subsets of the set of signals. The authors employ the correlation coefficients and thus are limited to linear dependencies between the signals. The second point is the connection and time synchronization between single automata. Nevertheless, this approach could be useful for investigating complex production systems with several parallel processes within one PC. These systems are not in the focus of our work.

III. Concept of the ACD Algorithm

In this section, two ACD algorithms are introduced. They use different procedures to detect PCs, but both rely on OTALA. The online ACD algorithm can be fed with a data stream, whereas the offline ACD algorithm needs recorded data. In the following subsection, we introduce the foundation.

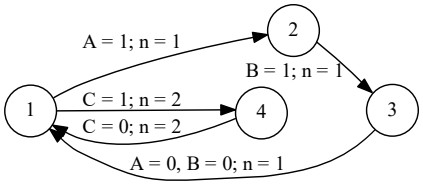
A. Foundation

OTALA is an online algorithm that is able to efficiently learn automata based on discrete signals and is constructed for the usage in production systems. The learned automata consist of states and transitions, see Fig. 1. Typically OTALA provides

978-1-7281-8956-7/20/$31.00 ©2020 IEEE

Authorized licensed use limited to: UNIVERSIDADE DE SAO PAULO. Downloaded on October 06,2020 at 18:09:09 UTC from IEEE Xplore. Restrictions apply.

the time between two states and the probability of the usage of a transition. We do not use this information, but we introduce n which is the number of passes of transitions. An automaton can be represented by a directed graph $G = (S, T)$ with the states $s_1, s_2, ..., s_K \in S, K \in \mathbb{N}$ and a set of transitions $T \subseteq S \times S$. A state $s \in S$ is defined as a tuple consisting of a unique identifier and a vector that contains the values of all discrete signals of the system. Transitions between states are triggered by a change of values of discrete signals. So, OTALA provides a directed graph that indicates the process of the production plant and thus somehow includes the PCs. Such a graph can be represented by an adjacency matrix **A**, which represents whether two states are connected.



	S1	S2	S3	S4
S1	0	1	0	2
S2	0	0	1	0
S3	1	0	0	0
S4	2	0	0	0

Fig. 1. Automaton of a PC learned by OTALA and its CAM.

Definition 1 (Adjacency matrix): Given a directed graph $G = (S, T)$ with a set of states $S, |S| = K, K \in \mathbb{N}$ and a set of transitions $T \subseteq S \times S$. An adjacency matrix $\mathbf{A} \in \{0, 1\}^{K \times K}$ is defined by its elements a_{ij} as follows:

$$a_{ij} = \begin{cases} 1, \ if \ (s_i, s_j) \in T \\ 0, \ else. \end{cases}$$

During a PC, some states can be visited multiple times. Thus, the adjacency matrix as defined above cannot represent a PC. Therefore, the adjacency matrix can be adapted to a new matrix which elements indicate how often a transition has been passed during a PC. We define the counted adjacency matrix (CAM) as follows:

Definition 2 (CAM): Given a directed graph G as in Definition 1. A counted adjacency matrix $\mathbf{M} \in \mathbb{N}^{K \times K}$ is defined by its elements m_{ij} as follows:

$$m_{ij} = \begin{cases} n_{ij}, \ if \ (s_i, s_j) \in T \\ 0, \ else. \end{cases}$$

where n_{ij} is the number of times the transition (s_i, s_j) fires during the one PC.

The CAM for the automaton from Fig. 1 is on its right-hand side. In this example the cycle (s_1, s_2, s_3, s_1) is passed once, whereas the cycle (s_1, s_4, s_1) is passed twice and the combination of them $(s_1, s_2, s_3, s_1, s_4, s_1, s_4, s_1)$ is the PC.

For the creation of CAM, we introduce a currently counted adjacency matrix (CCAM) which is applied in our approach. This matrix is built during the learning process of the automaton and its elements change over time.

Definition 3 (CCAM): At current point in time, a learning algorithm has identified a set of states $S, |S| = K, K \in \mathbb{N}$. A currently counted adjacency matrix $\mathbf{M}_{Main} \in \mathbb{N}^{K \times K}$ is defined by its elements u_{ij} as follows:

$$u_{ij} = \begin{cases} n_{ij}, \ if \ (s_i, s_j) \in T \\ 0, \ else. \end{cases}$$

where n_{ij} is the number of times the transition (s_i, s_j) has fired from the beginning of the learning process until current time point.

In this work, we introduce two algorithms that make use of learning algorithm OTALA and put out the CAM. The first algorithm extends OTALA during learning (online) and the second builds on the result of OTALA (offline).

B. Online-Approach

The online ACD algorithm is integrated into OTALA, whereby OTALA uses a data stream as input to identify transitions. To detect the cycles, a CCAM is generated out of the transitions. With every next event, it is checked whether a transition is passed for the first time. If the transition is passed more than once, it indicates a potential new PC, i.e. the CCAM contains a cycle candidate. Each cycle candidate is stored in the list M_C. Once a new candidate has been added to the list M_C, a zero matrix with the same size as this candidate matrix is created and added to the list M_R. The reference matrices are used for the evaluation process and store transitions that have fired after the corresponding cycle candidate has been identified. When a pair of reference and candidate matrix are equal, a potential PC has been detected and the candidate matrix is stored in a list of potential PCs. If there are insuperable differences between reference and candidate matrix, e.g. a transition in the reference matrix that does not occur in the candidate, the candidate and its pendant are deleted. Independently of the number of candidates, the CCAM will be feed with data, until OTALA converged. The matrix from the list of potential PCs that contains the cycle with the maximum length is the CAM and represents the identified PC. Fig. 2 shows the procedure at the point in time where a new candidate has been identified.

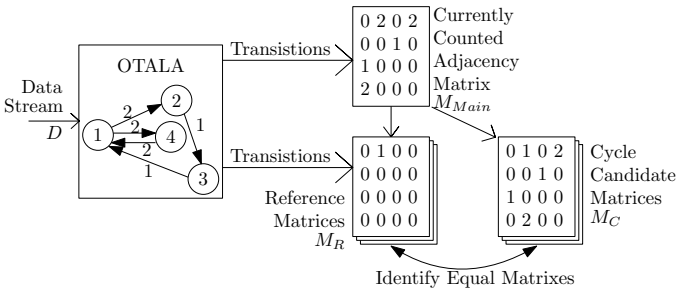


Fig. 2. Illustration of the online production cycle detection.

The pseudo-code is presented in Algorithm 1. From line 1 to line 4 the lists and matrices are initialized. OTALA is started in line 5. The loop from line 6 to line 21 is executed until OTALA converges. In line 7, a next transition is detected by OTALA. In line 8, it is checked whether the matrix $\mathbf{M}_{Main}$ is a new cycle candidate. This is the case, if the current transition is already available in $\mathbf{M}_{Main}$, so the transition is passed at least for the second time, and if the CCAM is not a multiple of an element in the lists $\mathbf{M}_C$ or $\mathbf{M}_{Cycle}$. A multiple means that the matrix can be generated by a scalar multiplication out of an existing candidate. This method is necessary because

Authorized licensed use limited to: UNIVERSIDADE DE SAO PAULO. Downloaded on October 06,2020 at 18:09:09 UTC from IEEE Xplore. Restrictions apply.

Algorithm 1: Algorithm for Online Cycle Detection

Input: Discrete data stream $d \in D$
Output: Counted Adjacency Matrix $\mathbf{M}$

```
1  Currently Counted Adjacency Matrix M_Main ← ∅
2  List of Production Cycle Matrices M_Cycle ← ∅
3  List of Cycle Candidate Matrices M_C ← ∅
4  List of Reference Matrices M_R ← ∅
5  start(OTALA)
6  repeat
7      if (t ← getNextTransistion()) then
8          if (MatrixContainsTransition(M_Main, t) and
            (notAMultiple(M_Main, M_C) or
            notAMultiple(M_Main, M_Cycle)) then
9              addToList(M_Main, M_C)
10             addToList((0), M_R) //Add zero matrix
11         M_Main ← addTransitionToMatrix(t, M_Main)
12         forall (M_Ri in M_R) do
13             M_Ri ← addTransitionToMatrix(t, M_Ri)
14             if (Equals(M_Ri, M_Ci)) then
15                 M_Cycle ← addToList(M_Ci, M_Cycle)
16                 delete(M_Ci)
17                 delete(M_Ri)
18             else if (CannotGetEqual(M_Ri, M_Ci)) then
19                 delete(M_Ci)
20                 delete(M_Ri)
21 until (OTALA converged);
22 M ← largestCyle(M_Cycle)
23 return M
```

otherwise every multiple of a PC (two cycles, three cycles,...) would be detected as a potential PC. If a new candidate has been determined, it is added to the candidate list $\mathbf{M}_C$ and a zero matrix is added to the reference matrix list $\mathbf{M}_R$. Line 11 adds the current transition to the CCAM. An iteration over all elements in the list of reference matrices $\mathbf{M}_R$ adds the current transition to each matrix $\mathbf{M}_{R_i}$, at first (line 12+13). Then, it is checked if the coupled elements $\mathbf{M}_{R_i}$ and $\mathbf{M}_{C_i}$ in the reference list $\mathbf{M}_R$ and the candidate list $\mathbf{M}_C$ are equal, see line 14. If they are equal, it is a PC candidate that is added to the list $\mathbf{M}_{Cycle}$ and deleted from the list of candidates and references. Furthermore, it is checked if the matrices can get equal in line 18. That is the case if for all elements of the reference matrix $\mathbf{M}_{R_i}$ each element is lower or equal than its pendant at the $\mathbf{M}_{C_i}$ matrix. If they cannot get equal, they are deleted from the lists (line 19+20). In line 22, the cycle is chosen, which has the longest length of the detected cycles in the list $\mathbf{M}_{Cycle}$. This is the detected PC. The CAM of this PC is returned and this algorithm is completed.

C. Offline Approach

The offline approach uses the learned automaton and extracts relevant information out of it. Therefore, a histogram over the number of passed transitions during the learning process is calculated, e.g. the bin 10 contains the number of transitions that have been passed 10 times, called *frequency*. Generally, it is expected that the lowest bin indicates the number of PC during the learning process and that only multiples of this number occur as bins in the histogram. In practice, it might be different, because of data quality issues. To deal with it, we take each bin h_i and sum up the frequencies of h_i and all multiples of it. The bin h_i with the highest sum is expected as the number of PCs. Knowing the number of PCs enables us to calculate the CAM based on the automata, by taking the CCAM after the learning process is over and divide each element of CCAM by the number of PCs. Fig. 3 shows the three steps of the offline detection. The numbers near the transition indicate the number of passes n_{ij} of a transition (s_i, s_j) during the learning process as defined in Definition 3.

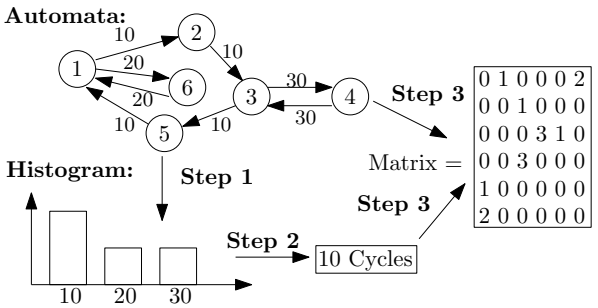


Fig. 3. Illustration of the offline cycle detection.

Since some states are reached only once due to data quality issues, we delete transitions that occur on a very low level. This is necessary because in practical applications there are regularly single state detections. Since every number is a multiple of 1, it is sure that 1 would be detected as the number of cycles. To avoid that, we can use the convergence criteria from OTALA, which suggests a convergence factor $f_{conv} \approx 10$ [4]. The convergence factor is multiplied with the number of states, to achieve the number of events in which the automaton should not change to reach the convergence. The convergence factor can be a guideline to set a minimum threshold τ that has to be reached to be respected in the PC detection. Based on our experiences a half of the convergence factor is an appropriate value. Nevertheless, τ should always be larger than 4.

The pseudo-code for the offline approach is presented in Algorithm 2. It requires a finite automaton and the threshold τ as input and provides the CAM for a single PC as output. The variable S_{Max} is used as score and the corresponding number of cycles n are initialized in line 1. The CCAM M_{Main} is created in line 2 and contains the number of passes of each transition. In the next step, the histogram is created based on M_{Main} that represents the occurrences of the transitions, see line 3. Transitions that occur less than the threshold τ are deleted from the histogram in line 4. From line 5 to line 7 there is an iteration through all histogram bins since it is checked if a bin h_j is a multiple of the bin h_i, see line 8. If that is true, the frequency of h_j is added to a temporary *sum*. In line 10-12, it is checked whether the *sum* of h_i and

Authorized licensed use limited to: UNIVERSIDADE DE SAO PAULO. Downloaded on October 06,2020 at 18:09:09 UTC from IEEE Xplore. Restrictions apply.

Algorithm 2: Algorithm for Offline Cycle Detection

Input: Finite automaton E, Threshold τ
Output: Counted Adjacency Matrix **M**

1 Score $S_{Max} \leftarrow 0$, cycles $n \leftarrow 1$
2 CCAM $M_{Main} \leftarrow getListOfTransitions(E)$
3 Histogram $H \leftarrow createHistogram(M_{Main})$
4 $H \leftarrow DeleteLowOccuringTransitions(H, \tau)$
5 **forall** *Histogram bin* $h_i \in H$ **do**
6 Score sum $\leftarrow transitionCount(h_i)$
7 **forall** *Histogram bin* $h_j \in H$ **do**
8 **if** $(Multiples(h_i, h_j))$ **then**
9 $sum \leftarrow sum + transitionCount(h_j)$
10 **if** $sum > s_{Max}$ **then**
11 $s_{Max} \leftarrow sum$
12 $n \leftarrow transitionCount(h_i)$
13 $M \leftarrow \frac{1}{n} \times M_{Main}$
14 **return** M

its multiples is larger than the maximum score. If so, the max score is updated and the number of cycles is set to the bin h_i. As the last step, the CCAM M_{Main} is multiplied with $\frac{1}{n}$ to reach the CAM of a single PC, see line 13.

IV. Evaluation

We evaluated both algorithms on good quality data and real-world data, where fast changing signals might not acquired properly. At first, we present the results of the online algorithm, which is working fine on good quality data. Fig. 4 shows the CAM of the introduced example. Since the identified CAM is equal to the CAM in Fig. 1, the correct cycles have been detected and works also for larger examples fine. Nevertheless, this approach has the disadvantage that bad quality data have a strong influence on the identified cycles. So, if there is one incorrect data point in the first cycle or its evaluation, the cycle cannot be detected correctly.

```
Cycle has been detected!

Matrix:
0 1 0 2
0 0 1 0
1 0 0 0
2 0 0 0
```

Fig. 4. Result of the online approach using the automaton in Fig. 1

The offline approach also identified the cycles of the above-mentioned example correctly. But it is also suitable for the usage in applications where the data quality is not perfect. The histogram of a real-world application from the SmartFactoryOWL is presented in Fig. 5. Since the application contains some fast processes, the data acquisition was to slow, which led to bad data quality. Eight cycles have been passed for the histogram. As can be seen, many transitions are only passed once or twice and no transition is passed 16 times, which underlines the low sampling rate. Nevertheless, the occurrences lower than four are deleted and the multiples are calculated by the algorithm. Then, we got the correct result of eight cycles, which enables us to calculate to correct CAM. Due to the larger number of cycles, the influence of bad data quality cycles is significantly reduced.

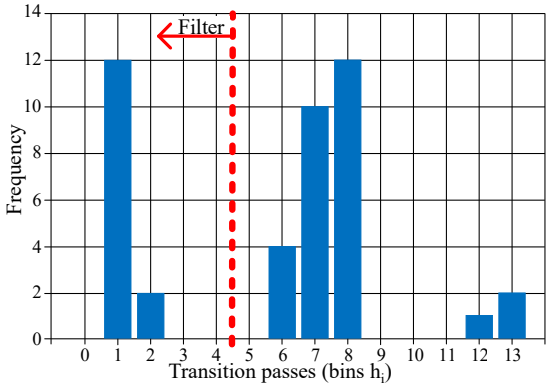


Fig. 5. Histogram of the offline approach using real world data, the number of cycles and thus the CAM can be identified correctly.

V. Discussion and Future Work

In this paper, we introduced two different approaches of how PCs can be detected automatically in production systems. Both rely on the representation of the plants' behavior due to automata, which can be learned by OTALA. We implemented both approaches and evaluated them. Both approaches work fine with good quality data. But if the data quality is not optimal, the offline approach performs better and it is more robust.

For future work, additional methods can be implemented to deal with bad data quality and thus improve the results in these cases. Furthermore, the approach should be tested on more plants to rate the reliability.

Acknowledgment

The work was partly supported by the German Federal Ministry of Education and Research (BMBF) under the project "KOARCH" (funding code: 13FH007IA6).

References

[1] World Economic Forum, "Technology and innovation for the future of production: Accelerating value creation," World Economic Forum, Genevy, Switzerland, Whitepaper, Mar 2017.
[2] J. Bughin, E. Hazan, S. Ramaswamy, M. Chui, T. Allas, P. Dahlström, N. Henke, and M. Trench, "Artificial intelligence the next digital frontier?" McKinsey Global Institute, Brussels, Discussion Paper, Jun 2017.
[3] McKinsey & Company, "Smartening up with AI - What's in it for Germany and its Industrial Sector?" Germany, Tech. Rep., Apr 2017.
[4] A. Maier, "Identification of timed behavior models for diagnosis in production systems," Ph.D. dissertation, University Paderborn, Feb 2015.
[5] D. Larchey-Wendling, "Proof pearl: Constructive extraction of cycle finding algorithms," in *Interactive Theorem Proving*, J. Avigad and A. Mahboubi, Eds. Cham: Springer International Publishing, 2018.
[6] R. Sedgewick, *Algorithmen*, 2nd ed. Pearson Studium, 2002.
[7] S. Torkamani and V. Lohweg, "Survey on time series motif discovery," *WIREs Data Mining and Knowledge Discovery*, vol. 7, no. 2, 2017.
[8] Y. Li, J. Lin, and T. Oates, *Visualizing Variable-Length Time Series Motifs*. Wiley, 2012, pp. 895–906.
[9] S. Windmann, D. Lang, and O. Niggemann, "Learning parallel automata of plcs," in *22nd IEEE International Conference on Emerging Technologies and Factory Automation (ETFA)*, 2017, pp. 1–7.

Authorized licensed use limited to: UNIVERSIDADE DE SAO PAULO. Downloaded on October 06,2020 at 18:09:09 UTC from IEEE Xplore. Restrictions apply.