Change-point visualization and variation analysis in a simple production line: a process mining application in manufacturing

Available online at www.sciencedirect.com

ScienceDirect

Procedia CIRP 99 (2021) 573–579

www.elsevier.com/locate/procedia

14th CIRP Conference on Intelligent Computation in Manufacturing Engineering, CIRP ICME '20

Change-point visualization and variation analysis in a simple production line: a process mining application in manufacturing

Edoardo Chiò[a], Arianna Alfieri[a], Erica Pastore[a,*]

[a]*Politecnico di Torino, c.so Duca degli Abruzzi 24, Torino 10129, Italy*

* Corresponding author. Tel.: +39-011-090-8302. *E-mail address:* erica.pastore@polito.it

Abstract

In production systems, digital twins must be always aligned with the real system to guarantee an effective decision making process in a continuously changing environment. To allow the alignment, digital models can be updated with process mining techniques through data collected by sensors. This paper addresses the issue of detecting changes in the production system by analyzing data collected from sensors. Using raw collected data, a procedure is proposed to compute and plot relevant system measures that could help change identification. Simulation is used to test the effectiveness of the procedure in a realistic medium size production line.

Keywords: Production Systems; Process Mining; Change Identification; Industry 4.0; Digital Twin

1. Introduction

Firms of the Industry 4.0 era need to be completely integrated and flexible to adapt the production to product customization and to frequent market changes. In this context, the real production system must have a continuously aligned digital counterpart (*digital twin*) to address planning and/or control problems in such a flexible environment.

Currently, digital models are defined in the system design phase, and they risk, if never updated, to become obsolete with respect to the current state of the real system, leading to wrong decisions and losses of efficiency. If the system changes due to unexpected events (e.g., failures) or managerial decisions (e.g., the addition of a new machine in a production stage by exploiting the *plug & produce* concept), and its digital twin is not promptly adapted to the change, the decisions taken by the use of digital twin risk to be no longer effective. Thus, keeping the digital twin continuously aligned with the real system is crucial to support an effective decision making process. To allow this alignment, the instantaneous identification of changes in the real system is a fundamental issue. To this purpose, data mining techniques on data, made available by the increasing computerization and sensorization of production lines (with a much larger availability of new data and information about the processes), can be used to keep the continuous alignment between real systems and their digital models.

Process Mining, a recent field of data mining, aims to build process models starting from process event logs. An event log is a strictly structured database in which each record represents an event occurred in the process [1]. Many algorithms have been developed to exploit event logs to understand the process structure (process discovery), to verify the alignment of the model with the real process (conformance checking), and to get deeper insights in the process (model enhancement) [2, 3]. Event logs are generated in the form of a data stream in which every new event should be evaluated to understand whether the underlying process is changing. The issue of creating a process model from a data stream has been addressed with different methods. Starting from process discovery algorithms developed in [3] (such as the Heuristic Miner), many adaptations have been proposed in the literature. Two different adaptations of the Heuristic Miner are proposed in [4]: the first iteratively applies a modified version of the

10.1016/j.procir.2021.03.122

algorithm on rolling windows (with a specific width) of data (Heuristic Miner with Sliding Window, SW); the second one, instead, works on buckets whose width depends on the maximum approximation error (Heuristic Miner with Lossy Counting, LC). The last approach is based on an algorithm proposed in [5]. A modified version of the Heuristic Miner was developed in [6], which is based on building initial prefix-trees to extract sequential patterns of events from the stream and on continuously updating this structure through "pruning" and "decaying" of relations and activities. When dealing with the system change detection, different kinds of drift (or variation) have to be considered. A simple drift classification is made in [7], which divides temporary from permanent variations (based on the lifespan), and sudden, recurring, gradual and incremental drifts (based on how they manifest).

In the last fifteen years, process mining discovery and change detection techniques have been applied to various industries; business processes [8] and healthcare processes [9] are the main fields of application. Only a few works applied process mining to production processes. Among these, a methodology based on process discovery algorithms is proposed in [10] to create the model of a manufacturing system and to deal with infrequent behaviors while keeping the model up to date. A probabilistic neural net is used to distinguish exceptional messages, while a genetic algorithm allows to continuously refresh the model as the data flows.

In all the cited papers, the model is automatically updated as new events are added in the event log, without explicitly considering if and when a variation happened in the process. The objective of this paper is to contribute to the literature of the application of process mining in production systems. Specifically, the problem of identifying and understanding if, when and how the real process changes is addressed, thus leading the way to future studies on how to efficiently update production process digital models.

The rest of the paper is organized as follows. Section 2 describes the analyzed production system and the possible variations occurring in it, and Section 3 addresses the process mining technique proposed to detect the change. The numerical experiment and its results are presented in Section 4, while Section 5 concludes the paper with the main insights, limitations and ideas for future research.

2. System Description

In this paper, a serial-parallel production system characterized by $i=1,\dots,S$ stages (or *Activities*) and $j=1,\dots,N_i$ parallel identical machines (or *Resources*) at each stage i is considered. Before each stage i there is a finite capacity incoming buffer B_i, except for the first one B_1, which is uncapacitated to allow all the arriving jobs to enter the line. In addition to incoming buffers, after each stage there is an outgoing buffer whose capacity is equal to the number of machines in the stage (*congestion stock*).

The inter-arrival time I of jobs to the system is assumed to be deterministic, while the transfer time from one stage to another is assumed negligible. The machine processing time, instead, is stochastic but its probability distribution and the related parameters are assumed known.

For the rest of the paper the following notation is used:

- L_i: capacity of buffer B_i;
- R_{ij}: j-th resource of stage i;
- T_i: processing time distribution of each resource R_{ij} (index j is not used as all the resources in the same stage i are identical);
- M_i: average processing time of resources at stage i (i.e., the mean of distribution T_i);
- $C_i = N_i / M_i$: average capacity of stage i;
- C_{min}: average capacity of the bottleneck (BN) stage;
- $F = 1 / I$: arrival rate of jobs to the system.

Both stable ($F < C_{min}$) and unstable ($F > C_{min}$) processes are considered. However, for sake of simplicity, only systems that have initially highly congested or almost void steady state buffers are studied. No failures or setups are assumed to occur.

Data are assumed to have been already collected in a complete and static event log. The event log is assumed to have been collected from two different configurations of sensors in each stage of the line:

1. one sensor before the incoming buffer, one between the buffer and the resources, one before the congestion stock. (Fig. 1);
2. one sensor before the incoming buffer, one at the entrance of each resource, and one at the exit of each resource. (Fig. 2).

The sensors convey the following information:

- *CaseID*: identification number unique for each job;
- *EventID*: identification number unique for each event;
- *Activity*: identification number unique for each stage;
- *Resource*: identification number unique for each machine (collected only for the second configuration of sensors);
- *Timestamp:* timestamp, i.e., the time when the event occurs.

Three different types of changes are considered:

- variation of the buffer capacity;
- variation of number of parallel resources in a stage;
- variation of processing times in single resources.

A variation can occur at any stage (i'), which is defined according to the position of stage i' with respect to the bottleneck stage i''. If $i' < i''$ the variation is called *top variation*, otherwise, if $i' > i''$, the variation is called *bottom variation*.

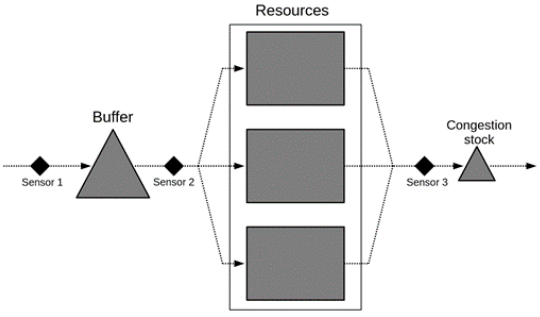


Fig. 1. First sensor configuration in a single stage with 3 parallel resources

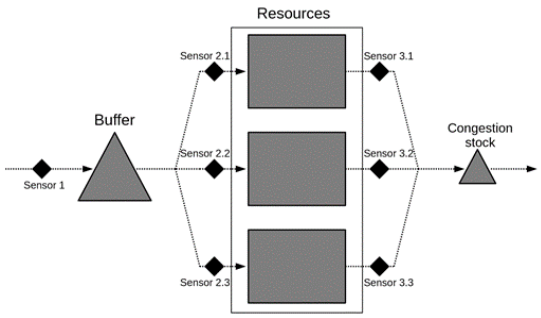


Fig. 2. Second sensor configuration in a single stage with 3 parallel resources

Single or multiple variations can occur at the same stage in different time instants. Moreover, according to the classification proposed in [7], only permanent (with respect to the lifespan) and sudden drifts (with respect to how they manifest) are studied.

3. Data reorganization

To detect the changes in the system, the data collected by the sensors are analyzed. The sensor output is an event log containing the five fields previously described (Table 1): CaseID, EventID, Activity, Resource, Timestamp.

Table 1. An event log section

CASEID	EVENTID	ACTIVITY	RESOURCE	TIME
9146	224835	4	0	638115
9146	227134	4	1	646182.5
9146	227154	4	1	646262.2
9147	224858	4	0	638199.6
9147	227157	4	1	646262.2
9147	227177	4	1	646340.6
9148	224881	4	0	638271.7
9148	227180	4	1	646340.6
9148	227200	4	1	646415.1
9149	224907	4	0	638354.7
9149	227203	4	1	646415.1
9149	227223	4	1	646494.1
9150	224930	4	0	638446.2
9150	227226	4	1	646494.1
9150	227246	4	1	646565

To extract information from the event log, data are reorganized as follows. The log is divided in different data frames, one for each different Activity. Each Activity frame is called case list (Table 2), and it includes the events of the log classified in three categories:

- entrance of a job in the stage buffer related to the Activity (*Timestamp_buff*);
- departure of a job from the stage buffer and its entrance in a resource of the stage (*Timestamp_res*);
- departure of a job from the resource and its entrance in the congestion stock of the stage (*Timestamp_end*).

Table 2. A case list section

CASE ID	TIMESTAMP_BUFF	TIMESTAMP_RES	TIMESTAMP_END
9146	638115	646182.5	646262.2
9147	638199.6	646262.2	646340.6
9148	638271.7	646340.6	646415.1
9149	638354.7	646415.1	646494.1
9150	638446.2	646494.1	646565

Each record (i.e., each row) of the case list represents the three different event types occurring to a job in a single stage. Within the case list, the timestamp differences are created. They refer to qualities related to the flowing of jobs in the system. As an example, consider Fig. 3.

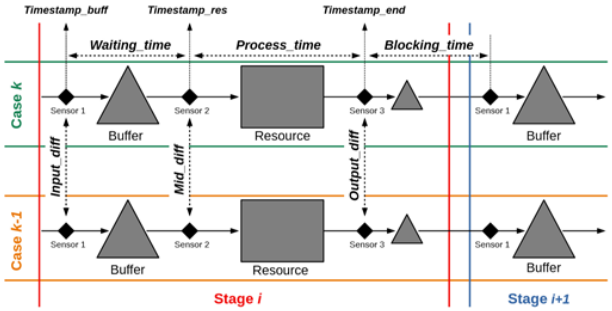


Fig. 3. Timestamps and intervals scheme.

Table 3. Same case intervals

CASEID	WAITING_TIME	PROCESSING_TIME	BLOCKING_TIME
9146	8067.515	79.6887	47.11337
9147	8062.591	78.40707	45.18427
9148	8068.841	74.55483	26.16633
9149	8060.458	78.99909	39.311
9150	8047.883	70.8386	52.87607

Table 4. Consecutive case intervals

CASEID	INPUT_DIFF	MID_ DIFF	OUTPUT_ DIFF
9146	79.22356	78.35248	79.6887
9147	84.61259	79.6887	78.40707
9148	72.15771	78.40707	74.55483
9149	82.93689	74.55483	78.99909
9150	91.57459	78.99909	70.8386

The timestamp differences are classified in two categories, and they are computed for each job *k* and stage *i*:
Same case intervals refer to the differences between the timestamps of a same job (Table 3):

- waiting time of job *k* in the buffer of stage *i*
  Waiting_time(k,i) = Timestamp_res(k,i) – Timestamp_buff(k,i);
- processing time of job *k* at stage *i*
  Process_time(k,i) = Timestamp_end(k,i)- Timestamp_res(k,i);

- blocking time of job *k* in the congestion buffer of stage *i*
  Blocking_time(k,i) = Timestamp_buff(k,i+1)- Timestamp_end(k,i).

Consecutive case intervals refer to the differences between consecutive job arrivals in the same sensor position (Table 4):

- *Input_diff(k,i) = Timestamp_buff(k,i) - Timestamp_buff(k-1,i)*;
- *Mid_diff(k,i) = Timestamp_res(k,i) - Timestamp_res(k-1,i)*;
- *Output_diff(k,i) = Timestamp_end(k,i) - Timestamp_end(k-1,i)*.

The case list and the timestamp differences are used to detect the process behavior and the changes in the system through graphs. Specifically, two types of plots are created. The first type of plots shows the pairs of intervals (timestamps) as dependent variables, and CaseID as independent variable. In the second type of plot, instead, the dependent variables are mean, variance, skewness, and kurtosis of timestamps. These quantities are computed using a moving window, characterized by a width and a shifting frequency, on the case list. These quantities are plotted as function of CaseID, to deepen the understanding of how process variations can change the event log data.

4. Numerical experiments

A simulator is used to replicate a production system and create the event logs collected by the sensors. A production line with 8 stages and 15000 jobs is considered. The processing times of all the resources are assumed to follow triangular distributions.

In the experiments, some factors have been varied to analyze the behavior of the algorithm in various scenarios.

Specifically, the factors are:

- number of machines per stage (j = {1,2,3});
- time occurrence of a change in the system within the simulation (after 5000, 7500, 10000, 12500 jobs have been processed by the first stage);
- moving window length (100, 200, 300, 400, 500 jobs);
- window updating frequency (25, 50, 75 jobs);
- stage of the change occurrence (i = {4,6});
- bottleneck stage (BN) stage (i = {4,6}).

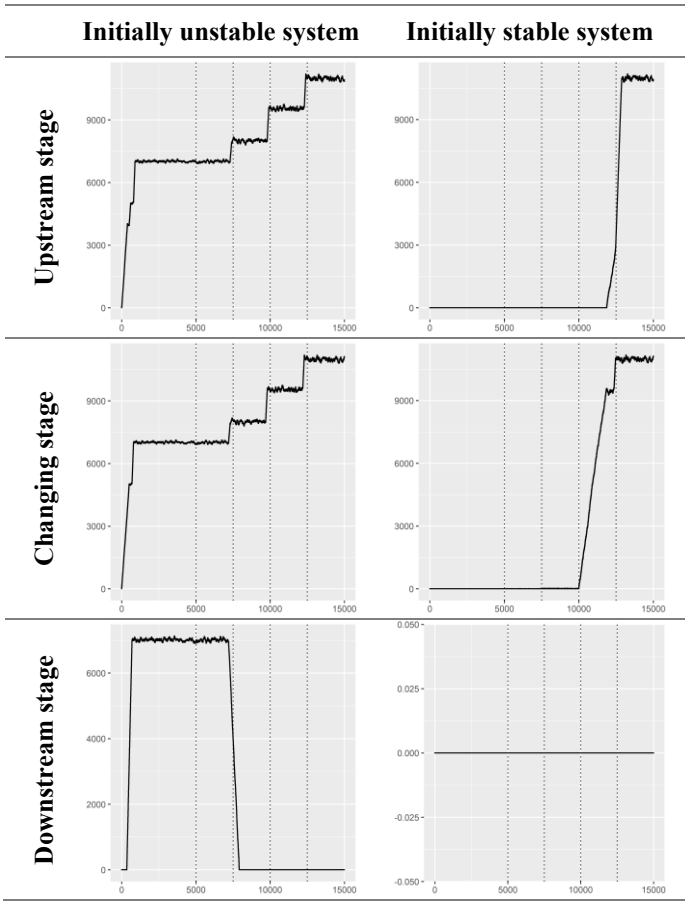
Each simulation run generates an event log that is transformed in a case list as described in section 3. The data of the case list are plotted in different combinations. For conciseness reasons, the results presented in the paper only refer to an increase in the mean processing time in one of the 4th stage resources, while the bottleneck is placed in the 6th stage (*top variation*).

4.1. Waiting time

The waiting time interval of the case list has been plotted as a function of CaseID. Fig. 4 shows the case of a system composed of one resource per stage. Specifically, the graphs in Fig. 4 show the stage before the one where the variation occurs, the changing stage, the following stage, and the BN stage, respectively. The plots of an initially unstable (on the left) and an initially stable (on the right) system are placed side by side to compare the different behaviors.

In an initially unstable system, all the buffers upstream the bottleneck (BN buffer included), after the transient period, are saturated (i.e., the average waiting time is equal to the ratio between the buffer capacity and the BN capacity, L_i / C_{min}). Instead, the buffers downstream the bottleneck are almost empty (i.e., zero waiting time). If a stage, after a variation, has C_i / C_{min}, it becomes the new BN. In the waiting time plots, this variation can be identified when the waiting time in its buffer grows, as in the upstream buffers, and the waiting time in the downstream buffers decreases to 0.

In an initially stable system, instead, all the buffers are empty (the waiting time is 0); in the BN buffer there is a cloud of dots as there are jobs waiting in it, but most of them have an almost null waiting time. When the changing stage gets $C_i < C_{min}$, if C_i is still larger than the inter-arrival rate F, no variation occurs. Only if C_i becomes lower than F, the change can be identified (i.e., it becomes visible); in this case, all the buffers in the upstream stages (and in the changing position) become saturated, and the average waiting time increases to L_i / C'_{min} (where C'_{min} is the capacity of the new BN).



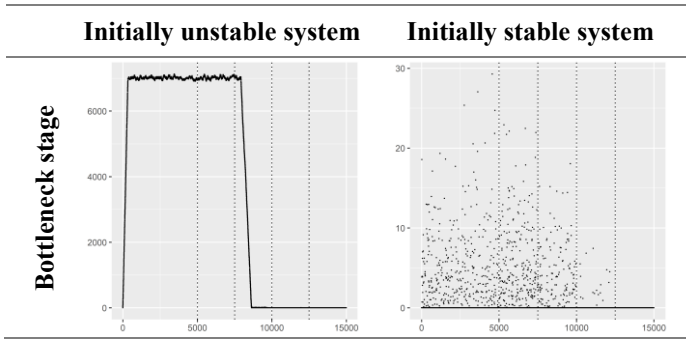


Fig. 4. Waiting time plots.

In conclusion, the graphs behave accordingly to the theory, and this type of plots is useful to identify a change in the processing time and to understand if the BN stage changes with respect to its initial position.

4.2. Processing time

The processing time is plotted as a function of CaseID in Fig. 5. The graphs are related to a line composed of three resources per stage. The plots compare an initially unstable and an initially stable system. The sensors are placed according to the second configuration (Fig. 2), to isolate the behavior of the changing resource. The top graphs show the three resources together, while the bottom ones show only the resource affected by the variation.

Regardless of the system stability, the plots show that the processing time grows. However, in an unstable system every increase appears sooner than in the stable one. This happens because the more the buffers are saturated, the longer jobs are processed, because of the increase in the processing time.

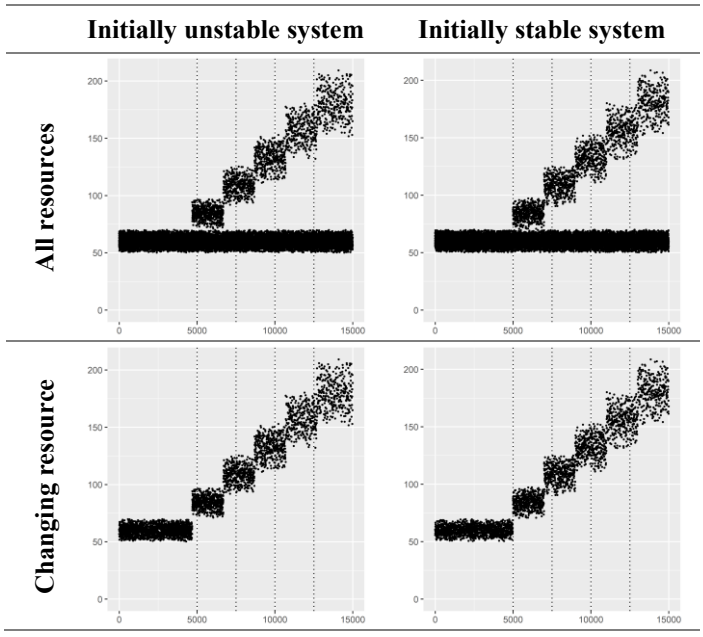


Fig. 5. Processing time plots.

This type of plots can then be useful to understand whether and in which stage a processing time change or a resource activation (or deactivation) is happening, independently from the system stability.

4.3. Blocking time

The blocking time is plotted as a function of CaseID in Fig. 6. The plots refer to a line composed of single resource stages. The graphs show the blocking time in the stage before the changing stage in an initially unstable (on the left) and initially stable (on the right) system.

In an unstable system, the blocking occurs in all the stages upstream the BN. If the capacity in the changing stage decreases below the capacity in the BN, blocking no longer occurs in all the stages between the changing stage and the initial BN (in a *top variation* situation), while the blocking time in the upstream stages increases. The plot on the left shows the described situation.

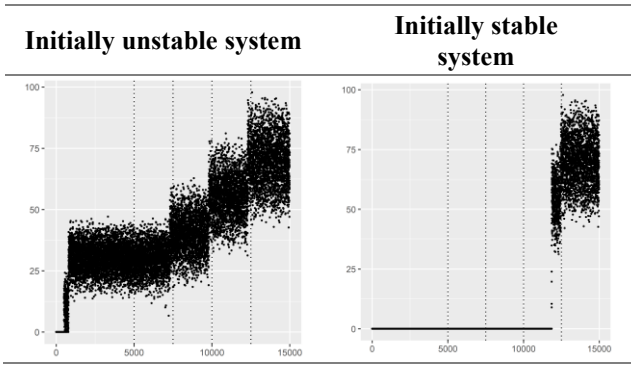


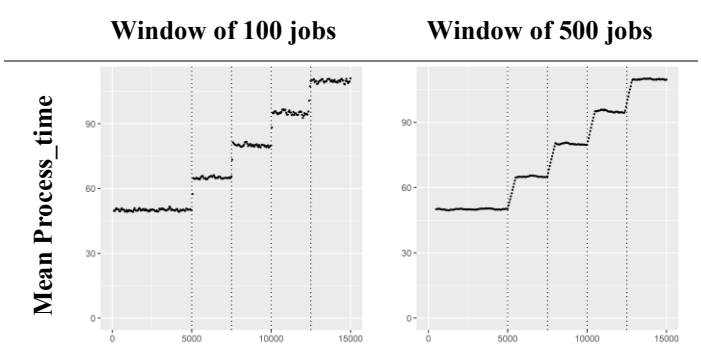
Fig. 6. Blocking time plots.

Instead, the blocking does not occur if the system is stable. Only when the capacity in the changing stage decreases below the inter-arrival rate ($C_i < F$), the system becomes unstable and the blocking occurs in all the stages upstream, as can be seen in the graph on the right.

This type of plots clearly shows when a stable system becomes unstable and vice versa.

4.4. Changes in the parameters of a distribution

As discussed in section 3, various moments of the distributions of performance measures have been calculated through moving windows. Plotting these moments as a function of CaseID can be used to identify the changes in the distribution parameters.



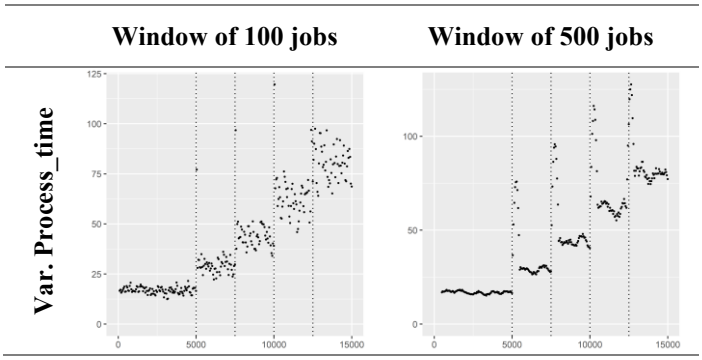


Fig. 7. Moments of the processing time distribution.

The plots in Fig. 7 are related to a line with one resource for stage and stable behavior. The graphs show the moving windows of the mean (upper graphs) and of the variance (lower graphs) of the processing time of the resource affected by the change. Two different window lengths have been used (100 jobs on the left and 500 jobs on the right) while the update frequency is set to 50 jobs in all the graphs.

The plots show that the shorter the window, the faster the change can be identified; however, a larger window better filters data from noise and outliers. One of the main issues is finding the suitable window length to balance detection speed and accuracy. The variance plots have peaks when the processing time changes: this happens because the windows include pre-change and post-change processing times. After each peak, the variance stabilizes on larger levels because, in the tests, each processing time variation affects the mean and variance of its distribution, by inflating both.

By looking at the plots of the moving windows of the mean and of the variance, changes on the parameters of the distribution can be spotted. The moving average plots show a change in the mean of the distribution (regardless the length of the window), however it does not show a change in the variance of the distribution. Instead, by plotting the moving window of the variance, changes in the mean of the distribution can be spotted with peaks in the graph, while the change in the variance of the distribution can be spotted by shifts of the curve in the graph.

5. Conclusions

In this paper, the problem of the automatic identification of changes in a production system is addressed. Specifically, the paper addresses the issue of extracting information about the changes in the system from sensor collected data. The available data are manipulated and reorganized so as to define the so-called *timestamps*, and relevant timestamp differences are computed. These differences are plotted as a function of the jobs flowing in the production system to evaluate if they are able to show occurred system changes. This methodology was applied to a simple flow line through simulation, and results showed that the usefulness of each plot depends on the state of the system at the time the change occurred, and on the type of occurred change.

Some results, being in accordance with the production system theory, seems obvious, however the purpose of this work is not to explain the system behavior rather to identify changes. Thus, the results show which subset of data should be analyzed to detect different type of changes, to reduce the big data load that will be available in the future full sensorized lines.

The main limit of this research is that currently the use of plots to identify changes implies a human operator evaluating them. Hence, ongoing research focuses on automatically identifying changes in the system without (or with a reduced) human intervention. The objective is to develop an algorithm that automatically detects and classifies changes from timestamp difference plots.

Other issues that future research will have to consider are those related to the assumptions made in the paper. First, the paper considered only extreme cases with respect to the initial buffer levels: completely empty buffers (and stage capacities much larger than those requested by the inter-arrival times) or completely full buffers (and stage capacities much smaller than those requested by the inter-arrival times). Intermediate cases, more representative of real stable systems, need to be considered. To this purpose, also other types of systems, other types of changes (e.g., failures or setup times), and not sudden (i.e., incremental) changes will be considered.

With respect to the plot procedure, when data windows and updating frequencies are used, as in the plots of the distribution moments, parameters to choose the most adequate window length and frequency value should be defined. Moreover, as in real situations it is necessary to work with group of data, the use of data windows and updating frequencies has to be evaluated also for the other type of plots (e.g., waiting time or processing time plots).

To have more information of possible distribution changes, also the use of concentration indices, applied to the data windows, will be evaluated.

References

[1] Van Der Aalst, W., Adriansyah, A., De Medeiros, A. K. A., Arcieri, F., Baier, T., Blickle, T., & Burattin, A. (2011, August). Process mining manifesto. In International Conference on Business Process Management. Springer, Berlin, Heidelberg. p. 169-194

[2] Song, M. & Van Der Aalst, W.M.P. (2007). Supporting process mining by showing events at a glance. In WITS 2007 - Proceedings, 17th Annual Workshop on Information Technologies and Systems. Social Science Research Network. p. 140–145

[3] Van Der Aalst, W.M.P. (2016). Process mining: data science in action. Springer Verlag.

[4] Burattin, A., Sperduti, A., & van der Aalst, W. M. (2014, July). Control-flow discovery from event streams. In 2014 IEEE Congress on Evolutionary Computation (CEC). IEEE. p. 2420-2427

[5] Manku, G. S., & Motwani, R. (2002, January). Approximate frequency counts over data streams. In VLDB'02: Proceedings of the 28th International Conference on Very Large Databases. Morgan Kaufmann. p. 346-357

[6] Hassani, M., Siccha, S., Richter, F., & Seidl, T. (2015, December). Efficient process discovery from event streams using sequential pattern mining. In 2015 IEEE symposium series on computational intelligence. IEEE. p. 1366-1373

[7] Bose, R. J. C., van der Aalst, W. M., Žliobaitė, I., & Pechenizkiy, M. (2011, June). Handling concept drift in process mining. In International Conference on Advanced Information Systems Engineering. Springer, Berlin, Heidelberg. p. 391-405

[8] Tiwari, A., Turner, C. J., & Majeed, B. (2008). A review of business process mining: state-of-the-art and future trends. Business Process Management Journal, 14(1), 5-22.

[9] Rojas, E., Munoz-Gama, J., Sepúlveda, M., & Capurro, D. (2016). Process mining in healthcare: A literature review. Journal of biomedical informatics, 61, 224-236.

[10] Denno, P., Dickerson, C., & Harding, J. A. (2018). Dynamic production system identification for smart manufacturing systems. Journal of manufacturing systems, 48, 192-203.