Automated Generation of Digital Models for Production Lines through State Reconstruction

Lulai Zhu[1], Giovanni Lugaresi[2] and Andrea Matta[1]

***Abstract*— Thanks to the rapid advances in information technologies, digital twins have been widely adopted in the manufacturing industry to support production planning and control. At the core of a digital twin is a digital model that mirrors the physical system in a virtual space. It is inefficient to develop digital twins by modeling the considered systems manually. Although significant research effort has been made to automate the generation of digital models, most approaches so far impose strong assumptions on the available data or cannot precisely capture the behavior of the physical system. Noticing the current gap, we propose in this paper a novel approach for automatically generating a graph representation of a production line from an event log through state reconstruction. The feasibility of the proposed approach has been demonstrated on three simulated instances.**

I. INTRODUCTION

In era of Industry 4.0, manufacturers are continuously embracing cutting-edge information technologies such as Internet of Things, cloud computing, big data and artificial intelligence [1]. These constitute a rich technology stack to develop and deploy digital twins for production lines. As high-fidelity virtual replicas of physical systems, digital twins facilitate a diversity of collaborative services that could enhance day-to-day operations and achieve a higher productivity [2]. A large enterprise, for example an automaker, may have up to thousands of production lines [3]. Building a digital model—the most basic component of a digital twin—for each production line is a time-consuming and error-prone task, which necessitates automatic model generators.

A model generator is a tool for interpreting the topology and behavior of a physical system into code, thereby allowing a computer to undertake the job of a human modeler [4]. In the context of manufacturing, model generation entails recognizing the characteristics of individual machines and their logical relationships, e.g. spatial constraints, temporal statistics and scheduling policies [5], [6]. If a model generator is deployed across the life cycle of a production line, the addition or removal of a machine can be reflected in the digital model promptly [7], [8].

Recent approaches take advantage of process mining [9] to generate digital models from historic data. Process mining is a field of inquiry devoted to discovering, validating and improving operational processes based on event logs accessible in information systems. It is agnostic with respect to the application domain and has proven effective on a plethora of data sets. Some key performance indicators like average flow times can be evaluated directly with processing mining techniques.

To extend the framework introduced in [10], we propose a new approach that automates the generation of a graph model for a production line from an event log through state reconstruction. The novelty of the approach lies in its ability to reconstruct the trajectory of the system state along the event log, which affords an overall insight into the system at any point in time. As a result of this improvement, all the attributes including processing time distributions can be estimated accurately under relatively mild assumptions.

The rest of the paper is organized as follows. Section II reviews the related work to highlight the motivation behind. Section III describes the physical system and defines the digital model, while Section IV discusses the details of the proposed model generation approach. A validation of the approach on three simulated instances is reported in Section V. Conclusive remarks are presented in Section VI.

II. RELATED WORK

According to Kritzinger et al. [11], a digital twin is a digital object that represents an existing physical object and exchanges data with it in a bidirectional manner. The digital object alone without any connections from or to the physical object is called the digital model. Digital twins are specifically developed for their intended purposes, for example design assistance and property validation [12]. With digital twins, a manufacturer could parallelize the executions of product development and production planning. This helps to reduce the integration time of new product variants and identify potential bottlenecks at an early stage [6].

Simulation-based digital twins are commonly applied in the production phase. They are operable offline for analyzing sensitivity and robustness against uncertainties, or online for anticipating deviations from the plan and selecting the best action after a disturbance [13]. In the online operation mode, the digital model has to align with its physical counterpart and adapt to changes in the system configuration at all times, which cannot rely solely on human experts. Automatic approaches for generating and updating digital models would therefore markedly promote the deployment of digital twins in the manufacturing industry [14].

Pourbafrani and van der Aalst [15] proposed a novel framework that exploits statistical and machine learning

*This work was supported by the European Union's Horizon Europe research and innovation program under grant agreement No. 101092021 (AUTO-TWIN)

[1]Lulai Zhu and Andrea Matta are with the Department of Mechanical Engineering, Politecnico di Milano, Via La Masa 1, Milano 20156, Italy {lulai.zhu,andrea.matta}@polimi.it

[2]Giovanni Lugaresi is with CentraleSupélec, Université Paris-Saclay, 3 Rue Joliot Curie, 91190 Gif-sur-Yvette, France giovanni.lugaresi@centralesupelec.fr

techniques to generate causal-loop diagrams from coarse-grained event logs. The main idea is to identify the underlying relations between system variables. Causal-loop diagrams can be used to simulate and predict high-level system dynamics. Lugaresi and Matta [10] proposed a process mining-based approach to automated model generation and tuning for production lines. The generated models are extended directed graphs that have a simulation equivalence. More complex systems such as assembly lines can also be discovered through an extension of the approach to object-centric process mining [16].

Despite being contributive, these approaches resort to a sequential discovery procedure that does not keep track of the system state. Most attributes are then blindly estimated without taking into account their state-dependent nature. For example, a certain scheduling policy may take effect only when a station is in the degraded condition. In this case, the correct behavior of the system can hardly be captured without awareness of how its state evolves over time, resulting in a loss of accuracy. The motivation for the proposed model generation approach is indeed to overcome such a major limitation among previous work.

III. Problem Formulation

This work deals with the problem of automatically generating digital models for production lines. In the following two subsections, we provide a description of the physical system under study and the definition of the digital model used to represent it.

A. Physical System

The physical system to be discovered is a production line comprising a number of stations connected by conveyors, splitters and mergers for discrete-part manufacturing. Each station integrates a buffer with serial storage slots and a machine with parallel processing units to carry out a single production step on incoming parts. After being processed at one station, a part is transferred to another station responsible for the next production step. If the buffer of the latter happens to be full at the moment, the part is forced to wait at the former until a storage slot in the buffer becomes empty, i.e. blocking after service (BAS). Work in progress (WIP) is under strict control so that a raw part may be dispatched only when the total number of parts in the system is less than a configured limit. This is helpful to prevent deadlock and optimize performance [17]. A part is removed from the system once it is finished. For clarity, we refer to stations where the production process is initiated and terminated as the source and sink stations respectively. Note that the source stations fetch raw parts from the inventory directly, hence no buffers as opposed to the others.

Figure 1 illustrates an instance of the physical system, which consist of four stations, namely S1, S2, S3 and S4. As can be seen, S1 and S4 serve as a source and a sink station, while S2 and S3 form a parallel substructure. S1 initiates the production process by pulling a raw part from the inventory if it has an idle processing unit and the WIP is below the maximum limit. Parts from S1 fork into two separate flows to S2 and S3. After that, they join again into a single flow to S4, where the production process is terminated.

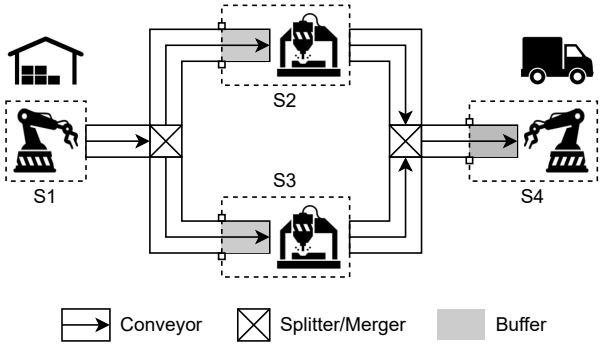


Fig. 1: An instance of the physical system with a parallel substructure.

TABLE I: A fragment of the event log expected from the parallel instance.

Time	Station ID	Part ID	Activity
…	…	…	…
2023-01-29 12:47:02	S1	P3	EXIT
2023-01-29 12:47:06	S1	P2	EXIT
2023-01-29 12:47:19	S2	P2	ENTER
2023-01-29 12:47:32	S3	P1	EXIT
2023-01-29 12:47:32	S3	P3	ENTER
2023-01-29 12:47:45	S4	P1	ENTER
2023-01-29 12:48:34	S4	P1	EXIT
2023-01-29 12:50:48	S2	P2	EXIT
2023-01-29 12:50:54	S3	P3	EXIT
2023-01-29 12:50:57	S4	P2	ENTER
…	…	…	…

We require the physical system to be equipped with a data acquisition module that records an event when a part enters or exits the machine of a station and collates the recorded events into a log based on their temporal order. In our setup, an event is a quadruple $e = (t, s, p, a)$, where $e.t$ is the occurrence time of the event, $e.s$ and $e.p$ are the IDs of the involved station and part respectively, and $e.a$ is the activity performed in the event. This quadruple uniquely differentiates every event recorded by the data acquisition module, since it is impossible for a part to enter or exit the same machine repeatedly at the same time. Let $\mathcal{E}$ be the set of the recorded events. The event log L can be viewed as the chronological permutation of elements in the set $\mathcal{E}$, i.e. $L : [1\,..\,|\mathcal{E}|] \leftrightarrow \mathcal{E}$[1] such that $\forall i \in [1\,..\,|\mathcal{E}| - 1], L(i).t \leq L(i+1).t$. Particularly, we deem simultaneous events to be naturally prioritized and preserve their original order in the log. Table I reports a fragment of the event log expected from the parallel instance illustrated in Figure 1.

B. Digital Model

To model the aforementioned physical system, a few assumptions are made on its dispatching policy and routing topology. We presume that the system applies an egalitarian dispatching policy. When the WIP is less than the maximum

[1] $[x\,..\,y]$ denotes an integer interval between x and y; $\mathcal{X} \leftrightarrow \mathcal{Y}$ denotes a bijective mapping between $\mathcal{X}$ and $\mathcal{Y}$.

TABLE II: THE PARAMETERS OF THE DIGITAL MODEL.

Symbol	Definition
$\mathcal{S}$	Set of the stations
$\mathcal{C}$	Set of the connections
$\mathcal{S}_{\text{source}}$	Set of the source stations
$\mathcal{S}_{\text{sink}}$	Set of the sink stations
b_s*	Buffer capacity at the station s
m_s	Machine capacity at the station s
P_s	Processing time distribution at the station s
$r_{s,s'}$	Routing probability on the connection (s, s')
$T_{s,s'}$	Transfer time distribution on the connection (s, s')
n_{max}	Maximum allowable WIP

* b_s is constantly zero if $s \in \mathcal{S}_{\text{source}}$.

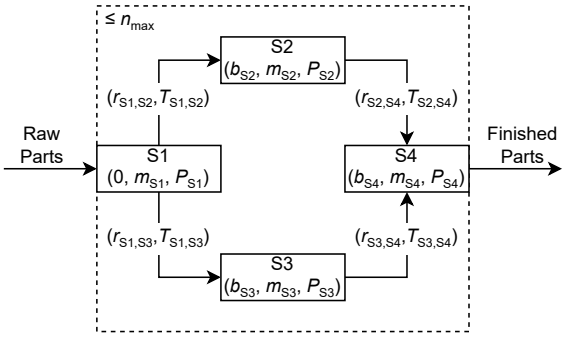


Fig. 2: The digital model adapted for the parallel instance.

limit, every source station with an idle processing unit has equal chance of winning a raw part. This is trivially true if the number of the source stations is merely one. Besides, we presume the routing topology of the system to be well formed in the sense that no connections exist from any stations to a source station or from a sink station to any stations. Such connections may pose complex scheduling issues and are thus scarcely seen in reality. For example, an additional rule needs to be enacted so as to decide whether a source station accepts a raw part from the inventory or one from some station in the case of contention.

Under these assumptions, we model the physical system using an extended directed graph, where nodes represent stations in the system, and edges represent connections between the stations. Table II summarizes the parameters of the digital model. The topology of the system is depicted by four sets: $\mathcal{S}$, $\mathcal{C}$, $\mathcal{S}_{\text{source}}$ and $\mathcal{S}_{\text{sink}}$. $\mathcal{S}$ is the set of the stations, $\mathcal{C}$ is the set of the connections, $\mathcal{S}_{\text{source}}$ is the set of the source stations, and $\mathcal{S}_{\text{sink}}$ is the set of the sink stations. To characterize each station $s \in \mathcal{S}$, we assign the corresponding node three attributes, which are the buffer capacity b_s, the machine capacity m_s and the processing time distribution P_s at the station. Likewise, the edge corresponding to each connection $(s, s') \in \mathcal{C}$ is assigned two attributes: the routing probability $r_{s,s'}$ and the transfer time distribution $T_{s,s'}$ on the connection. The maximum allowable WIP, in particular, is treated as a global attribute n_{max}. Figure 2 illustrates the digital model adapted for the parallel instance illustrated in Figure 1.

The choice of an extended directed graph rather than a formal mathematical model, e.g. a Petri net [18] or a queueing network [19], or a simulation model executable by specific software, e.g. AnyLogic or Arena, stems from a trade-off between conciseness, expressiveness and flexibility. Compared to a Petri net, the extended directed graph treats stations and connections as individual nodes and edges, thereby being more intuitive and tractable. Blocking in a queueing network is confined between immediately connected stations, which makes the inclusion of the transfer times difficult. The extended directed graph does not have this limitation. Nevertheless, one can straightforwardly map it onto a closed queueing network with BAS [20] if there is just one source station and the transfer times are negligible, and may also translate its semantics into a generalized stochastic Petri net [21] or a simulation model with extra effort.

IV. PROPOSED APPROACH

We now elaborate the proposed approach to the automated generation of digital models for production lines. To help with a better understanding, an overview of the approach is first drawn, followed by a clarification of technical details for all the steps.

A. Overview

As explained in the last section, a production line can be represented as an extended directed graph. We automatically generate such a digital model from an event log through a pipeline of five steps, specifically data extraction, trace analysis, topology mining, state reconstruction and attribute mining. Figure 3 provides an overview of the proposed approach. Given an event log in a format equivalent to Table I, we begin by extracting sets, subsets and sublogs that are indispensable in the subsequent steps. The traces of parts involved in the event log are then collected, and the system topology is mined from the set of the collected traces, which leads to a bare directed graph without attributes. Next, we reconstruct the state of the system after every event by recursively computing its relative states and determining its initial state. The attributes of the stations and connections are mined from the state trajectory in combination with the event log, and an extended directed graph is ultimately constructed.

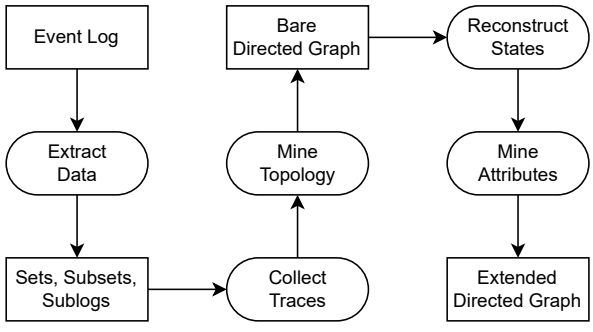


Fig. 3: An overview of the proposed approach.

B. Data Extraction

As a preparation, the following sets, subsets and sublogs are extracted from the input event log L:

- the set $\mathcal{E}$ of events recorded in the event log, i.e. $\mathcal{E} = \{e \mid \exists i \in [1\,..\,\infty), e = L(i)\}$;
- the set $\mathcal{S}$ of stations involved in the event log, i.e. $\mathcal{S} = \{s \mid \exists e \in \mathcal{E}, s = e.s\}$;
- the set $\mathcal{P}$ of parts involved in the event log, i.e. $\mathcal{P} = \{p \mid \exists e \in \mathcal{E}, p = e.p\}$;
- the subset $\mathcal{E}_s$ of events related to each station $s \in \mathcal{S}$, i.e. $\mathcal{E}_s = \{e \mid e \in \mathcal{E} \wedge e.s = s\}$;
- the subset $\mathcal{E}_p$ of events related to each part $p \in \mathcal{P}$, i.e. $\mathcal{E}_p = \{e \mid e \in \mathcal{E} \wedge e.p = p\}$;
- the sublog L_s of events related to each station $s \in \mathcal{S}$, i.e. $L_s : [1\,..\,|\mathcal{E}_s|] \leftrightarrow \mathcal{E}_s$ such that $\forall j \in [1\,..\,|\mathcal{E}_s| - 1], L_s(j).t \le L_s(j+1).t$;
- the sublog L_p of events related to each part $p \in \mathcal{P}$, i.e. $L_p : [1\,..\,|\mathcal{E}_p|] \leftrightarrow \mathcal{E}_p$ such that $\forall j \in [1\,..\,|\mathcal{E}_p| - 1], L_p(j).t \le L_p(j+1).t$.

We regard the sublogs as chronological permutations of events as well. These sets, subsets and sublogs will be extensively used later.

C. Trace Collection

By our definition, the trace θ_p of a part p is the sequence of stations that it went through, i.e. $\theta_p : [1\,..\,h_p] \to \mathcal{S}_p$, where h_p is the length of the trace, and $\mathcal{S}_p$ is the set of stations visited by the part. If the input event log does not cover the entire production period, the sublogs of events related to a minority of parts are probably truncated. To cope with potential incompleteness, we collect the trace of each part p from the sublog L_p with respect to the activity performed in the first recorded event:

- when $L_p(1).a = \text{ENTER}$,

$$\hat{\theta}_p(k) = L_p(2k-1).s \quad \text{for } k \in [1\,..\,\hat{h}_p], \tag{1}$$

  where $\hat{h}_p = \lceil |\mathcal{E}_p|/2 \rceil$;
- when $L_p(1).a = \text{EXIT}$,

$$\hat{\theta}_p(k) = \begin{cases} L_p(1).s & \text{if } k = 1, \\ L_p(2k-2).s & \text{if } k \in [2\,..\,\hat{h}_p], \end{cases} \tag{2}$$

  where $\hat{h}_p = \lceil (|\mathcal{E}_p| - 1)/2 \rceil + 1$.

Consider the fragment of the event log reported in Table I as an example. By applying (1) and (2), $\hat{\theta}_{\text{P1}} = (\text{S3}, \text{S4})$, $\hat{\theta}_{\text{P2}} = (\text{S1}, \text{S2}, \text{S4})$ and $\hat{\theta}_{\text{P3}} = (\text{S1}, \text{S3})$.

D. Topology Mining

Four sets, i.e. $\mathcal{S}$, $\mathcal{C}$, $\mathcal{S}_{\text{source}}$ and $\mathcal{S}_{\text{sink}}$, together depict the topology of the system. Among them, the set $\mathcal{S}$ of the stations has been available since the first step. Let $\hat{\Theta}$ be the set of traces collected from the input event log. As shown in Algorithm 1, the set $\mathcal{C}$ of the connections can be found by simply iterating over each unique trace $\hat{\theta} \in \hat{\Theta}$ with length $\hat{h}$. Recall that we presume no connections from any stations to a source station or from a sink station to any stations. The sets of the source and sink stations, $\mathcal{S}_{\text{source}}$ and $\mathcal{S}_{\text{sink}}$, are identified accordingly. As long as all the connections have been observed in the event log, the digital model is isomorphic to the physical system. For example, the set of traces collected from the fragment of the event log reported in Table I is $\{(\text{S3}, \text{S4}), (\text{S1}, \text{S2}, \text{S4}), (\text{S1}, \text{S3})\}$, which is enough to detect the actual topology of the parallel instance illustrated in Figure 1.

Algorithm 1: Mining the topology of the physical system.

Input: $\hat{\Theta}$, $\mathcal{S}$ **Output:** $\mathcal{C}$, $\mathcal{S}_{\text{source}}$, $\mathcal{S}_{\text{sink}}$

```
C := ∅, S_source := ∅, S_sink := ∅
for θ̂ ∈ Θ̂
    for k ∈ [1 .. ĥ − 1]
        C := C ∪ {(θ̂(k), θ̂(k + 1))}
for s ∈ S
    if ∄s′ ∈ S, (s′, s) ∈ C
        S_source := S_source ∪ {s}
    if ∄s′ ∈ S, (s, s′) ∈ C
        S_sink := S_sink ∪ {s}
```

E. State Reconstruction

To encode the system state, we assign every station $s \in \mathcal{S}$ an index $K(s)$ with $K : \mathcal{S} \leftrightarrow [1\,..\,|\mathcal{S}|]$. The state of a station s is governed jointly by the number $n_{K(s),1}$[2] of parts being transferred to its buffer or queued therein and the number $n_{K(s),2}$ of parts being processed by its machine or blocked therein. Based on this notion, the state of the entire system can be written as a vector $\boldsymbol{n} = (n_{k,l}) = (n_{1,1}, n_{1,2}, n_{2,1}, n_{2,2}, \ldots, n_{|\mathcal{S}|,1}, n_{|\mathcal{S}|,2}) \in \mathbb{N}^{2|\mathcal{S}|}$. We denote by $\boldsymbol{n}(i)$ the system state after the i-th event in the input log. In particular, $\boldsymbol{n}(0)$ is the initial state of the system before the first event.

We infer the trajectory of the system state from the change brought by each event. If the sublog of events related to a part is forward truncated at a non-sink station with an EXIT activity, the next station to which the part proceeded is uncertain. State reconstruction is thus viable only up to but excluding the ξ-th event such that

$$\begin{aligned} \xi = \min\{i \mid (\exists p \in \mathcal{P}, L(i) = L_p(|\mathcal{E}_p|)) \\ \wedge L(i).s \notin \mathcal{S}_{\text{sink}} \wedge L(i).a = \text{EXIT}\}. \end{aligned} \tag{3}$$

Let $\boldsymbol{e}_{k,l} \in \mathbb{N}^{2|\mathcal{S}|}$ be a vector of all zeros except for a one in the (k,l)-th entry. The change of the system state due to the occurrence of the i-th event is given by

$$\begin{aligned} &\Delta\boldsymbol{n}(i) = \boldsymbol{n}(i) - \boldsymbol{n}(i-1) \quad \text{for } i \in [1\,..\,\xi), \\ &= \begin{cases} +\boldsymbol{e}_{k,2} & \text{if } L(i).s \in \mathcal{S}_{\text{source}} \wedge L(i).a = \text{ENTER}, \\ -\boldsymbol{e}_{k,1} + \boldsymbol{e}_{k,2} & \text{if } L(i).s \notin \mathcal{S}_{\text{source}} \wedge L(i).a = \text{ENTER}, \\ -\boldsymbol{e}_{k,2} + \boldsymbol{e}_{k',1} & \text{if } L(i).s \notin \mathcal{S}_{\text{sink}} \wedge L(i).a = \text{EXIT}, \\ -\boldsymbol{e}_{k,2} & \text{if } L(i).s \in \mathcal{S}_{\text{sink}} \wedge L(i).a = \text{EXIT}, \end{cases} \end{aligned} \tag{4}$$

[2] $n_{K(s),1}$ is constantly zero if $s \in \mathcal{S}_{\text{source}}$.

where $k = K(L(i).s)$ is the index of the station involved in the i-th event, and $k' = K(L_{L(i).p}(L^{-1}_{L(i).p}(L(i)) + 1).s)$ is the index of the next station that the part involved in the i-th event was routed to.

(4) enables us to recursively compute the relative state $\mathring{\boldsymbol{n}}(i)$ of the system after the i-th event with reference to its initial state $\boldsymbol{n}(0)$:

$$\mathring{\boldsymbol{n}}(i) = \boldsymbol{n}(i) - \boldsymbol{n}(0) = \begin{cases} \mathbf{0} & \text{if } i = 0, \\ \mathring{\boldsymbol{n}}(i-1) + \Delta\boldsymbol{n}(i) & \text{if } i \in [1\,..\,\xi). \end{cases} \tag{5}$$

Suppose that every entry of the system state has ever reached zero, i.e. $\forall (k,l) \in [1\,..\,|\mathcal{S}|] \times \{1,2\}, \exists i \in [0\,..\,\xi), n_{k,l}(i) = 0$. We determine the initial state of the system and then its state after the i-th event as

$$\hat{\boldsymbol{n}}(0) = -\left(\min_{i \in [0..\xi)} \mathring{n}_{k,l}(i)\right), \tag{6}$$

$$\hat{\boldsymbol{n}}(i) = \mathring{\boldsymbol{n}}(i) + \hat{\boldsymbol{n}}(0) \quad \text{for } i \in [1\,..\,\xi). \tag{7}$$

Notably, the prerequisite for (6) is immediately satisfied if the event log has captured the beginning of the production period, in which case $\boldsymbol{n}(0) = \mathbf{0}$.

F. Attribute Mining

Knowing the system state after every event greatly facilitates attribute mining. Capacity attributes inclusive of the WIP limit can be obtained directly from the state trajectory as follows:

- the buffer capacity at each station $s \notin \mathcal{S}_{\text{source}}$,

$$\hat{b}_s = \max_{i \in [0..\xi)} \hat{n}_{K(s),1}(i); \tag{8}$$

- the machine capacity at each station $s \in \mathcal{S}$,

$$\hat{m}_s = \max_{i \in [0..\xi)} \hat{n}_{K(s),2}(i); \tag{9}$$

- the maximum allowable WIP,

$$\hat{n}_{\max} = \max_{i \in [0..\xi)} \sum_{(k,l) \in [1..|\mathcal{S}|] \times \{1,2\}} \hat{n}_{k,l}(i). \tag{10}$$

(8), (9) and (10) are exact if and only if (6) holds and the actual values of the capacity attributes have ever been hit, i.e. $\exists i \in [0\,..\,\xi), n_{K(s),1}(i) = b_s$, $\exists i \in [0\,..\,\xi), n_{K(s),2}(i) = m_s$ and $\exists i \in [0\,..\,\xi), \sum_{(k,l) \in [1..|\mathcal{S}|] \times \{1,2\}} n_{k,l}(i) = n_{\max}$.

Two different types of flow time samples are acquirable from the input event log. The first type of sample is the flow time(s) of a part p from entering to exiting the machine of a station s:

$$\begin{aligned} &L_p(j).t - L_p(j-1).t \quad \text{for } j \in [2\,..\,|\mathcal{E}_p|] \\ &\quad \text{s.t. } i < \xi \wedge L_p(j).s = s \wedge L_p(j).a = \text{EXIT}, \end{aligned} \tag{11}$$

where $i = L^{-1}(L_p(j))$ is the log index of the j-th event in the sublog L_p. Both processing and blocking times may be contained in such a sample. Let s' be the ID of the next station visited by the part p, i.e. $s' = L_p(j+1).s$, and d_{release} the reaction delay from a storage slot in a buffer becoming empty until the first blocked part being released. To retain those with only processing times, we reject a flow time sample of the first type if the part p was blocked before exiting the machine of the station s, i.e. $s \notin \mathcal{S}_{\text{sink}} \wedge \hat{n}_{K(s'),1}(i) = \hat{b}_{s'} \wedge (\exists i' \in [1,i], L(i).t - L(i').t \leq d_{\text{release}} \wedge \hat{n}_{K(s'),1}(i'-1) = \hat{b}_{s'})$. Samples left are used to fit the processing time distribution at the station s.

The second type of sample is the flow time(s) of a part p from exiting the machine of a station s to entering that of a station s':

$$\begin{aligned} &L_p(j).t - L_p(j-1).t \quad \text{for } j \in [2\,..\,|\mathcal{E}_p|] \\ &\quad \text{s.t. } i < \xi \wedge L_p(j-1).s = s \wedge L_p(j).s = s' \\ &\qquad\qquad \wedge L_p(j).a = \text{ENTER}, \end{aligned} \tag{12}$$

where $i = L^{-1}(L_p(j))$ is the log index of the j-th event in the sublog L_p. Such a sample may contain both transfer and queueing times. Let d_{seize} be the reaction delay from a processing unit in a machine becoming idle until the first queued part being seized. To retain those with only transfer times, a flow time sample of the second type is rejected if the part p was queued before entering the machine of the station s', i.e. $\hat{n}_{K(s'),2}(i) = \hat{m}_{s'} \wedge (\exists i' \in [1,i], L(i).t - L(i').t \leq d_{\text{seize}} \wedge \hat{n}_{K(s'),2}(i'-1) = \hat{m}_{s'})$. The transfer time distribution on the connection (s, s') is fit to the remaining samples.

Let $f_{s,s'}$ be the frequency of parts passing through the connection (s, s'), which is essentially the number of the corresponding flow time samples acquired previously. We calculate the routing probability on each connection $(s, s') \in \mathcal{C}$ as

$$\hat{r}_{s,s'} = \frac{f_{s,s'}}{\sum_{(s,s'') \in \mathcal{C}} f_{s,s''}}. \tag{13}$$

Routing probabilities yielded from (13) readily sum up to one, hence no need to be normalized.

At this stage, all the parameters of the extended directed graph have been mined, including the four sets that define the topology, the attributes of the stations and connections as well as the WIP limit. The digital model for representing the physical system can be finally built by populating a graph data structure accordingly.

V. Validation

To validate the proposed approach, we implement and encapsulate it in a Python module. Three instances of the physical system, on which the validation is conducted, are simulated using Arena. For benchmarking, all the code and data to reproduce the experiment results are shared at `https://github.com/zhululai/auto-model-gen-sr` under the BSD 3-Clause and CC BY 4.0 licenses respectively.

A. Experiment Setup

Apart from the parallel instance introduced in Section III, two other instances of the physical system are created for the validation of the proposed approach. Figures 4 and 5 illustrates these two instances. Like the parallel one, both of them comprise four stations whose IDs are S1, S2, S3 and S4. S1 and S4 in the two instances play the same roles, i.e. a source and a sink station, as those in the parallel instance. However, substructures formed by S3 and S4 therein are serial and cyclic respectively.

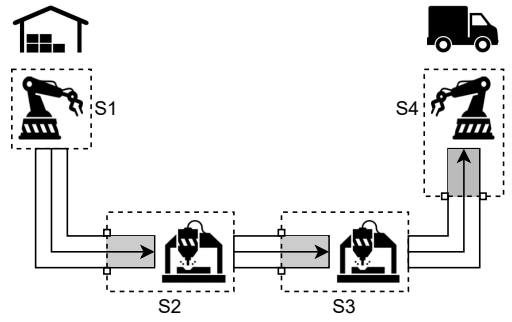


Fig. 4: An instance of the physical system with a serial substructure.

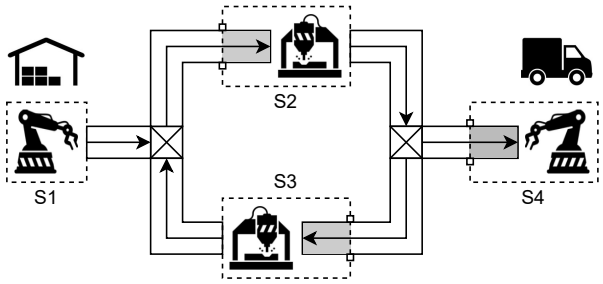


Fig. 5: An instance of the physical system with a cyclic substructure.

Tables III and IV report the attributes of stations and connections in the serial, parallel and cyclic instances. Correctly determining the initial state of the system and mining the capacity attributes of the stations requires the WIP limit to be bounded between the maximum capacity of the buffer or machine at each station and the total capacity of stations in the system minus the former, i.e. $\max_{s\in\mathcal{S}}\{b_s, m_s\} \leq n_{\max} \leq \sum_{s\in\mathcal{S}}(b_s+m_s) - \max_{s\in\mathcal{S}}\{b_s, m_s\}$. If the system has cyclic substructures, the WIP limit must meanwhile be less than the total capacity of stations within any cycle, i.e. $n_{\max} < \min_{o\in\mathcal{O}}\sum_{s\in\mathcal{S}_o}(b_s + m_s)$, where $\mathcal{O}$ is the set of cycles in the system topology, and $\mathcal{S}_o$ is the set of stations within the cycle $o \in \mathcal{O}$. This is to prevent deadlock. The capacity attributes of stations in the three instances are chosen such that a WIP limit satisfying the above conditions exist. As can be calculated from Table III, the feasible range of the WIP limit is from 6 to 12 for every instance. We assume the processing and transfer times of parts in the instances to be Erlangianly distributed. The expectations of the distributions are chosen in view of the fact that processing times are often much larger and more diverse than transfer times in a real system. To reflect their difference in variability, the coefficient of variation is set to 0.5 for the processing time distributions and to 0.2 for the transfer time distributions.

The actual topology of the system can be easily detected, provided that all the connections have been observed in the input event log. We are therefore primarily interested in how the accuracy of the proposed approach in attribute mining would be affected by the system topology, the WIP limit and the log length. To answer this question, nine alternatives are developed from the three instances by configuring the WIP limit to the lower bound, the median and the upper bound of its feasible range, i.e. 6, 9 and 12. Each of them is used to synthesize an event log that span the production of 12000 parts. Three longitudinally overlapping partitions of the resultant event log are then retrieved, starting from the 1001st to the 1100th, to the 2000th and to the 11000th completion respectively. The preceding procedure is repeated multiple times with five random seeds: 14561, 25971, 31131, 22553 and 12121. As such, we acquire a total of 135 truncated event logs with a length of 100, 1000 or 10000 completions. Table V reports the settings of these event logs. A digital model is generated from each event log, which is carried out on a Windows 10 laptop with eight Intel(R) Core(TM) i7-8565U CPUs @ 1.80 GHz and 16 GB RAM.

TABLE III: THE ATTRIBUTES OF STATIONS IN THE SERIAL, PARALLEL AND CYCLIC INSTANCES.

Instance	Station	Attribute*			
		b_s	m_s	$\mu(P_s)$	$\sigma(P_s)$
Serial	S1	0	1	50	25
	S2	4	2	150	75
	S3	4	1	75	37.5
	S4	6	1	100	50
Parallel	S1	0	2	100	50
	S2	4	1	150	75
	S3	4	1	150	75
	S4	6	1	75	37.5
Cyclic	S1	0	1	75	37.5
	S2	6	2	100	50
	S3	4	1	50	25
	S4	4	1	75	37.5

* $\mu(P_s)$ and $\sigma(P_s)$ stand for the mean and the standard deviation of the processing time distribution on the station s.

TABLE IV: THE ATTRIBUTES OF CONNECTIONS IN THE SERIAL, PARALLEL AND CYCLIC INSTANCES.

Instance	Connection	Attribute*		
		$r_{s,s'}$	$\mu(T_{s,s'})$	$\sigma(T_{s,s'})$
Serial	(S1, S2)	1	10	2
	(S2, S3)	1	10	2
	(S3, S4)	1	10	2
Parallel	(S1, S2)	0.5	10	2
	(S1, S3)	0.5	10	2
	(S2, S4)	1	10	2
	(S3, S4)	1	10	2
Cyclic	(S1, S2)	1	10	2
	(S2, S3)	0.4	10	2
	(S2, S4)	0.6	10	2
	(S3, S2)	1	10	2

* $\mu(T_{s,s'})$ and $\sigma(P_{s,s'})$ stand for the mean and the standard deviation of the transfer time distribution on the connection (s, s').

B. Experiment Results

Unsurprisingly, the topologies of the generated models and the corresponding instances are identical in all the cases. Figure 6 compares the the average times spent in generating digital models for the serial, parallel and cyclic instances. It takes longer to generate a digital model for the serial instance than for the parallel one, because a completion in the former

TABLE V: THE SETTINGS OF THE ACQUIRED EVENT LOGS.

Dimension	Settings
System Topology	Serial, Parallel, Cyclic
WIP Limit	6, 9, 12
Log Length	100, 1000, 10000 completions
Log Truncation	Both sides
Random Seed	14561, 25971, 31131, 22553, 12121

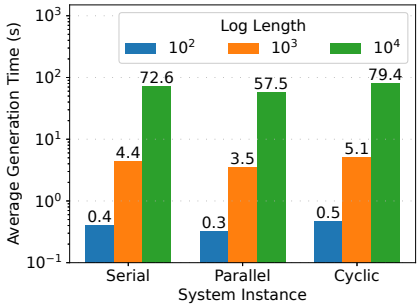


Fig. 6: The average times spent in generating digital models for the serial, parallel and cyclic instances.

adds eight events whilst that in the latter only counts six events. For the same reason, generating a digital model for the cyclic instance needs even more time, on average 79.4 s when the input event log covers 10000 completions.

We evaluate the discrepancy between a digital model and its physical counterpart in a station attribute x according to the following metric:

$$\max_{s\in\mathcal{S}} \left| \frac{\hat{x}_s - x_s}{x_s} \right|, \tag{14}$$

where $\hat{x}_s$ and x_s are the estimated and the true value of the attribute x at the station s. (14) is basically the maximum absolute relative error of the attribute x among all the stations. The discrepancy of the digital model from the physical system in a connection attribute y is evaluated similarly as

$$\max_{(s,s')\in\mathcal{C}} \left| \frac{\hat{y}_{s,s'} - y_{s,s'}}{y_{s,s'}} \right|, \tag{15}$$

where $\hat{y}_{s,s'}$ and $y_{s,s'}$ are the estimated and the true value of the attribute y on the connection (s, s').

Figures 7, 8 and 9 show the average maximum errors in estimating the attributes of stations and connections in the serial, parallel and cyclic instances respectively. Among all the attributes, the buffer capacity and the transfer time distribution are the most difficult to estimate. The major cause of the inaccuracy is the lack of information in the input event log. Under a low WIP limit, the number of parts being transferred to some buffer or stored therein may never hit the capacity of that buffer. As for a high WIP limit, the flow time sample of a part from exiting the machine of one station to entering that of another may always contain a blocking time. A medium WIP limit is recommended to avoid such situations. It is as expected that the errors of the estimated values reduce as the length of the event log increases. As suggested in Figures 7b, 8b and 9b, 1000 completions are typically adequate to gain a good estimation of every attribute under a medium WIP limit.

VI. CONCLUSIONS

This paper proposes an automatic model generation approach that explicitly takes into account the state of the physical system. The mining of the capacity attributes thus becomes more straightforward than ever, and the processing and transfer time distributions may now be accurately estimated. Moreover, the state trajectory gives an overall insight into the system at any moment, which possibly paves the way for the analysis of historical performance, the identification of scheduling policies and the discovery of complex manufacturing systems in future work.

Several limitations of the proposed approach are noteworthy. The state space considered is limited by the data acquisition capabilities of the physical system. This could result in various levels of granularity depending on the system configuration. The approach also requires a fair amount of data to ensure a precise representation, which hinders it from being adapted for certain applications. Last but not least, the validation experiments are performed on simple simulated instances. Further research is needed to confirm its applicability to real-world production lines.

REFERENCES

[1] F. Tao, Q. Qi, A. Liu, and A. Kusiak, "Data-Driven Smart Manufacturing," *Journal of Manufacturing Systems*, vol. 48, pp. 157–169, 2018.

[2] Q. Qi, F. Tao, Y. Zuo, and D. Zhao, "Digital Twin Service Towards Smart Manufacturing," *Procedia CIRP*, vol. 72, pp. 237–242, 2018.

[3] L. Wozniak and P. Clements, "How Automotive Engineering is Taking Product Line Engineering to the Extreme," in *Proceedings of the 19th International Conference on Software Product Line*. ACM, 2015, pp. 327–336.

[4] S. C. Mathewson, "Simulation Program Generators: Code and Animation on a PC," *Journal of the Operational Research Society*, vol. 36, no. 7, pp. 583–589, 1985.

[5] Y. J. Son and R. A. Wysk, "Automatic Simulation Model Generation for Simulation-Based, Real-Time Shop Floor Control," *Computers in Industry*, vol. 45, no. 3, pp. 291–308, 2001.

[6] F. Biesinger, D. Meike, B. Kraß, and M. Weyrich, "A Digital Twin for Production Planning Based on Cyber-Physical Systems: A Case Study for a Cyber-Physical System-Based Creation of a Digital Twin," *Procedia CIRP*, vol. 79, pp. 355–360, 2019.

[7] M. Fujihara and K. Yoneda, "Simulation through Explicit State Description and Its Application to Semiconductor Fab Operation," in *Proceedings of the 24th Winter Simulation Conference*. ACM, 1992, pp. 899–907.

[8] G. S. Martínez, S. Sierla, T. Karhela, and V. Vyatkin, "Automatic Generation of a Simulation-Based Digital Twin of an Industrial Process Plant," in *Proceedings of the 44th Annual Conference of the IEEE Industrial Electronics Society*. IEEE, 2018, pp. 3084–3089.

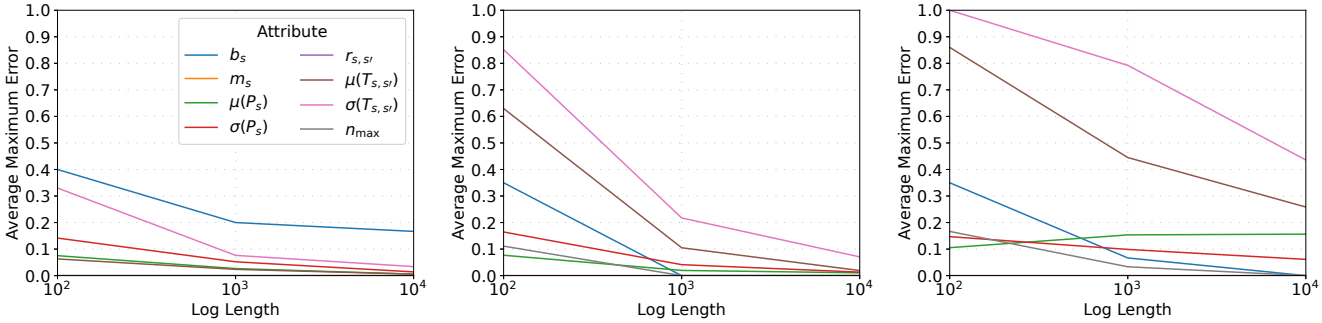
[9] W. M. P. van der Aalst, *Process Mining: Data Science in Action*. Springer, 2016.

[10] G. Lugaresi and A. Matta, "Automated Manufacturing System Discovery and Digital Twin Generation," *Journal of Manufacturing Systems*, vol. 59, pp. 51–66, 2021.

[11] W. Kritzinger, M. Karner, G. Traar, J. Henjes, and W. Sihn, "Digital Twin in Manufacturing: A Categorical Literature Review and Classification," *IFAC-PapersOnLine*, vol. 51, no. 11, pp. 1016–1022, 2018.

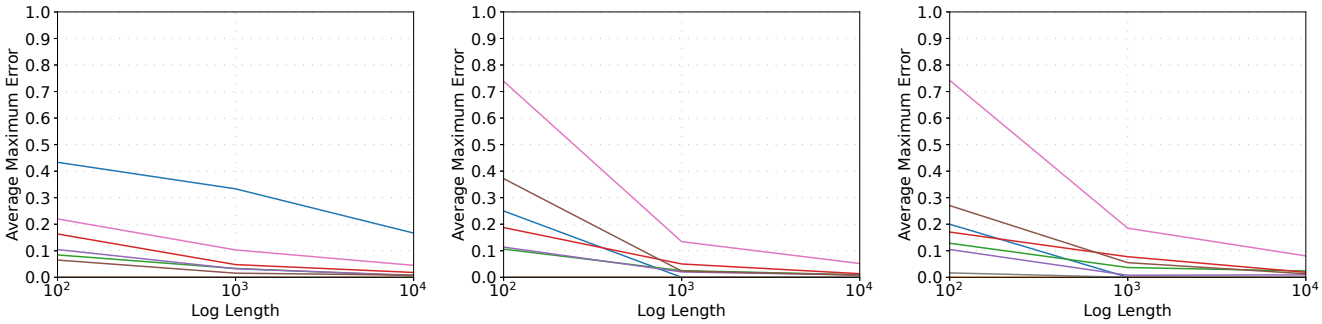
[12] S. Boschert and R. Rosen, "Digital Twin—The Simulation Aspect," *Mechatronic Futures: Challenges and Solutions for Mechatronic Systems and Their Designers*, pp. 59–74, 2016.

[13] L. Monostori, B. Kádár, T. Bauernhansl, S. Kondoh, S. Kumara, G. Reinhart, O. Sauer, G. Schuh, W. Sihn, and K. Ueda, "Cyber-Physical Systems in Manufacturing," *CIRP Annals*, vol. 65, no. 2, pp. 621–641, 2016.



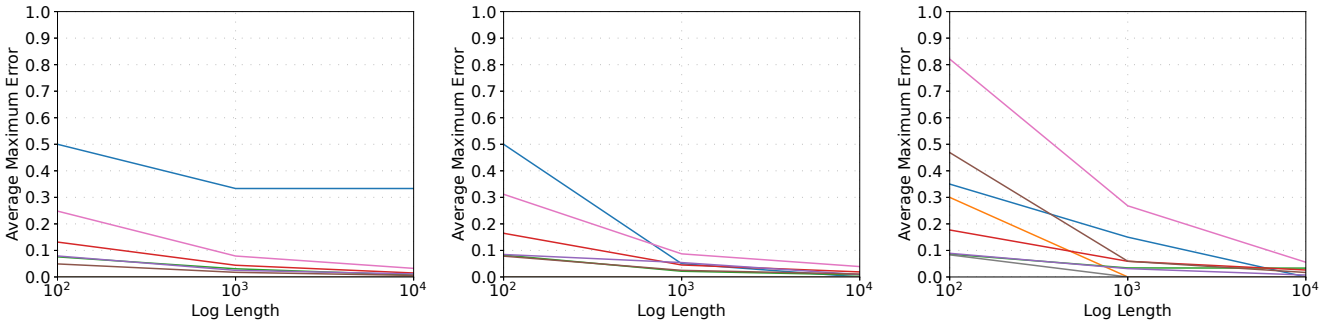
(a) $n_{\max} = 6$ (b) $n_{\max} = 9$ (c) $n_{\max} = 12$

Fig. 7: The average maximum errors in estimating the attributes of stations and connections in the serial instance.



(a) $n_{\max} = 6$ (b) $n_{\max} = 9$ (c) $n_{\max} = 12$

Fig. 8: The average maximum errors in estimating the attributes of stations and connections in the parallel instance.



(a) $n_{\max} = 6$ (b) $n_{\max} = 9$ (c) $n_{\max} = 12$

Fig. 9: The average maximum errors in estimating the attributes of stations and connections in the cyclic instance.

[14] G. Lugaresi and A. Matta, "Real-Time Simulation in Manufacturing Systems: Challenges and Research Directions," in *Proceedings of the 2018 Winter Simulation Conference*. IEEE, 2018, pp. 3319–3330.

[15] M. Pourbafrani and W. M. P. van der Aalst, "Discovering System Dynamics Simulation Models Using Process Mining," *IEEE Access*, vol. 10, pp. 78 527–78 547, 2022.

[16] G. Lugaresi and A. Matta, "Discovery and Digital Model Generation for Manufacturing Systems with Assembly Operations," in *Proceedings of the 17th IEEE International Conference on Automation Science and Engineering*. IEEE, 2021, pp. 752–757.

[17] J. A. Buzacott and J. G. Shanthikumar, *Stochastic Models of Manufacturing Systems*. Pearson, 1993.

[18] F. Bause and P. S. Kritzinger, *Stochastic Petri Nets: An Introduction to the Theory*. Vieweg, 2002.

[19] G. Bolch, S. Greiner, H. de Meer, and K. S. Trivedi, *Queueing Networks and Markov Chains: Modeling and Performance Evaluation with Computer Science Applications*. John Wiley & Sons, 2006.

[20] H. G. Perros, *Queueing Networks with Blocking*. Oxford University Press, 1994.

[21] M. A. Marsan, G. Balbo, G. Conte, S. Donatelli, and G. Franceschinis, *Modelling with Generalised Stochastic Petri Nets*. John Wiley & Sons, 1995.