# Automated Manufacturing System Discovery and Digital Twin Generation

Giovanni Lugaresi, Andrea Matta

# Automated Manufacturing System Discovery and Digital Twin Generation

Giovanni Lugaresi[1], Andrea Matta*[1]

[1] *Department of Mechanical Engineering, Politecnico di Milano*
*Via La Masa 1, 20156 Milan, Italy*
**Corresponding Author – andrea.matta@polimi.it*

**Abstract**

The latest developments in industry involved the deployment of digital twins for both long and short term decision making, such as supply chain management, production planning and control. Modern production environments are frequently subject to disruptions and consequent modifications. As a result, the development of digital twins of manufacturing systems cannot rely solely on manual operations. Recent contributions proposed approaches to exploit data for the automated generation of the models. However, the resulting representations can be excessively accurate and may also describe activities that are not significant for estimating the system performance. Generating models with an appropriate level of detail can avoid useless efforts and long computation times, while allowing for easier understanding and re-usability. This paper proposes a method to automatically discover manufacturing systems and generate adequate digital twins. The relevant characteristics of a production system are automatically retrieved from data logs. The proposed method has been applied on two test cases and a real manufacturing line. The experimental results prove its effectiveness in generating digital models that can correctly estimate the system performance.

**Keywords:** Digital Twins; Industry 4.0; Simulation; Process Mining.

## 1. Introduction

Digital twins have been considered as key components for the success of Industry 4.0 initiatives [1]. In production environments, digital twins have been exploited in processes such as assembly, scheduling, machining, and logistics, with the goal to reach high production efficiency [2]. The correct implementation of digital twins is claimed to make production processes more flexible, adaptable and predictable, thanks to the possibility of data collection not only from physical systems, but also from digital instances [3]. In general, virtual representations of physical resources can store information such as kinematic data, interfaces, production events, and performance indicators [4].

Within a production planning and control scope, discrete event simulation models can constitute the digital counterparts of manufacturing systems, and may be used for both long term planning and short term decision making. Real-time Simulation is a concept that involves using simulation as digital model of a system with the goal to take accurate decisions based on the current system state [5]. Monostori et al. [6] described the operation modes for discrete event simulation models in production systems: (1) offline simulation is used for sensitivity analysis and robustness evaluation of production rules before their implementation, (2) proactive simulation is used online with the aim of defining corrective actions after the recognition of potential deviations from the optimal plan, and (3) reactive simulation, which is used online for the evaluation of alternatives following disruptions such as machine failures or unplanned maintenance interventions.

The ability to take appropriate decisions is strongly based on the assumption that models properly aligned with the real system are either already available or obtainable within the decision time epoch [7]. Therefore, the time to develop a new model may hinder the exploitation of a digital twin along the production systems life cycle [8]. Indeed, manufacturing systems evolve regularly due to external drivers such as the irregularity of supplies or the availability of new disruptive technologies. Also, more frequent changes may occur. For instance, robots can be moved from one system to another, manufacturing cells can be reconfigured for new products, production lines may be re-shaped following new part-mixes. Further, the increasing demand for customized products highlights the necessary ability to promptly adapt manufacturing processes and resources [9].

Industry 4.0 contributed to the rise of new technologies for data acquisition, storing and communication, allowing for the knowledge of the shop floor status at anytime [10]. The availability of real-time data suggests that if a model could be generated from available data in the manufacturing system, the development phase may be significantly shortened, enabling digital twins to be automatically aligned with their real counterparts [11]. Traditional model generation techniques use the available data to estimate model parameters, such as inter arrival times or rework ratios. These approaches are based on an existing model structure, which is typically derived from previous knowledge or from interviews with company experts [12]. More recent approaches introduced the exploitation of process mining techniques [13], which enable to retrieve the system topology from the manufacturing system data [14].

Despite the aforementioned improvements, practical implementations of automated model generation remain scarce. One of the reasons is the difficulty in adapting the level of detail of the model [15]. Indeed, the availability of a digital model may not be enough if it is excessively detailed. Figure 1 graphically explains one of the possible outcomes of an automated model generation procedure, using as example the three-station production line shown in figure 1a. Conveyors bring pallets from one station to another, and are equipped with sensors that record the correct advancement of pallets. If all stations and sensors generate data, figure 1b shows a possible outcome of an automated data-based model generation. The result is that sensors are treated as activities, thus adding unnecessary operations to the model. Let us assume to be interested in time-related performance indicators such as the system time. In this case, the information from the sensors is redundant since the elements holding the work-pieces are the three stations. Hence, for the intended goal, the model depicted in figure 1c is a more reasonable abstraction of the process, and much closer to what an experienced modeler would choose. Therefore, the ability to tune – or adjust – the model level of detail is also desirable in an automated modeling procedure. In manufacturing applications, model adaptation may refer not only to the model parameters, but also to the system layout and logical structure. A tuned model is easily understandable by the user, and it has a higher probability of being reused [16].

This paper proposes a method for obtaining simulation-based digital twins starting from the data logs of manufacturing systems. The contribution of the paper is twofold. First, it outlines a procedure to automatically discover a manufacturing system from the production data and building a discrete event simulation model for performance estimation. The model generation is inclusive of both the production system logical structure and its parameters. In addition, this work provides a method to tune the model toward a desired level of detail, removing complexities that may hinder both the understandability and re-usability of the model for taking production planning and control decisions.

The rest of the paper is organized as follows: section 2 summarizes the related literature about model generation and tuning for manufacturing systems. Section 3 introduces the model generation procedure. Then, section 4 describes the proposed method for model tuning. Section 5 outlines the numerical experiments that we have done for testing the proposed method together with the numerical results. Finally, in section 6, we discuss on the limitations and the next developments of this research.

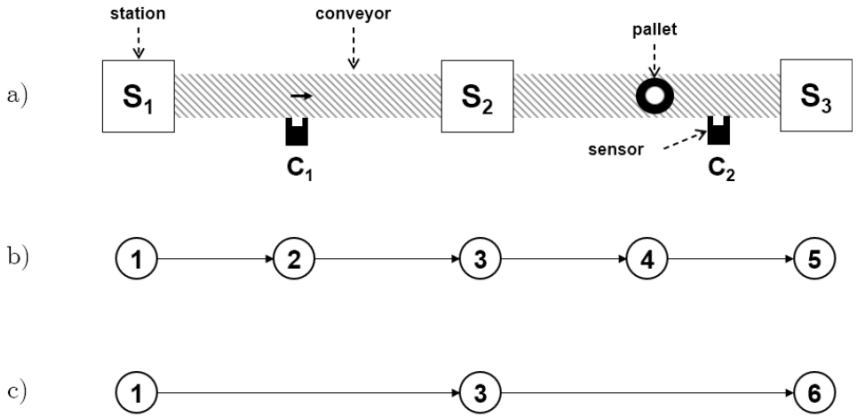


Figure 1: (a) Sequential 3-station production line; (b) graph model with 5 nodes; (c) graph model with 3 nodes.

## 2. State of the Art

In this section, we list significant contributions from the literature regarding: (1) model generation, in which a dataset from the real manufacturing system is used to build a model that can reproduce the system structure and parameters, and (2) model tuning, in which simulation models are modified with the goal to highlight most significant information. Then, we discuss on the research gap that we wish to cover with this paper.

### *2.1. Model Generation*

The automated generation of a simulation model typically starts with data collection. Data are gathered from the system and organized in event logs which record the production steps followed by each part and the corresponding time stamps. Starting from the available data, the material flows are identified. Given the sequence of activities performed by each part in the system, the structure of the production process can be inferred (e.g., system layout, operations precedences); hence, the system topology is discovered. Further, processing and waiting times can be retrieved from the data, together with the parameters of the statistical distributions describing the production dynamics. The control policies applied in the manufacturing system (e.g., routing and dispatching rules) can also be retrieved. Finally, the model may be converted to another modeling framework (e.g., from graph model to Petri Net) as well as to executable code, which allows for the validation of the obtained simulation models before being used to take decisions. We may divide model generation procedures in two main categories: (1) techniques which rely on an underlying structure, and (2) approaches which do not assume a specific logical structure. The former techniques consider that the logic of the model is known, and they interpret model generation as the translation of the logic into simulation code. The latter methods start from a more general point and infer the logic from the available data. The next two sections discuss on relevant contributions for model generation.

#### *2.1.1. Model Generation with an underlying structure*

According to Mathewson [17], a simulation generator is "*a software tool that translates the logic of a model into the code of a simulation language, enabling a computer to mimic a modelers behavior*". Among others, Shimizu and Van Zoest [18] developed an integrated software approach in which MANUPLAN models can be translated into SIMAN models. To date, this procedure enabled a simulation model of a factory to be developed in two days. Gong and McGinnins [19] developed a simulation code generator which compiles a system description into a simulation code (e.g., SIMAN). The user can update the system

design in the model by modifying an input database and running the simulation code generator. The authors demonstrated their approach with the operations and information flows of an automated guided vehicle for manufacturing cells. Son et al. [20] developed a methodology for automatically generating simulation models for specific manufacturing scenarios. A neutral language has been designed to semantically describe the simulation model, together with a model translator that converts the neutral description into syntax of specific simulation packages. Mueller et al. [21] introduced an approach in which the simulation model for a semiconductor manufacturing plant is generated from an input file that represents the simulation data specification. The simulation model is a Petri Net and it is built exploiting an object-oriented framework: several manufacturing system components are mapped into sub-graphs of a Petri Net and are represented by empty objects. Then, simulation model instances are created by populating the Petri Net data structure. Mesabbah et al. [22] proposed an automated simulation model builder adapted for health-care applications. The methodology couples model generation with machine learning algorithms that allow for the prediction of system performances based on real-time data stream. Indeed, classifiers are constructed making use of historical data to predict several patient performance metrics, such as length of stay and next activities.

*2.1.2. Model Generation without an underlying structure*

Recent approaches exploit Process Mining (PM) for simulation model generation [23, 13]. PM is a discipline aiming to discover and exploit valuable information from event logs available in information systems [24]. Process Mining allows not only to estimate parameters and causal relationships, but also logical relationships such as precedences among activities. Hence, it may be used to discover the topological structure of a manufacturing system. Bergmann et al. [25] introduced a methodology for recognizing the behavior of a manufacturing system in terms of production policies. Several data mining methods are tested (i.e. neural networks, support vector machines, decision trees) with the goal to recognize which policies are applied in the system generating data. Farooqui et al. [26] designed a methodology for the automated generation of formal models of robotic systems starting from the robot code structure and data. Milde and Reinhart [27] developed an approach for joint material flow discovery, parameter estimation, and control policies identification from manufacturing systems event logs. Martin et al. [28] improved inter arrival times modeling by including the mining of parts queuing at the entrance of the system in a model parameter estimation methodology. The authors used the proportion of entities queuing at arrival as in the event log to approximately estimate the parameters of the inter arrival times distribution. Denno et al. [29] developed a methodology to mine the production system structure and used genetic programming to link colored Petri Net states with exceptional system states. Ferreira and Vasilyev [30] combined PM with logical decision trees to understand the causes of process delays. Martin et al. [31] used PM to retrieve daily availability records from an event log, by considering a resource availability with both a temporal dimension and the possibility of intermediate interruptions. Martin et al. [32] designed an algorithm to mine how operational activities are batched within a production environment. The authors defined three batch processing types, presented a resource-activity centered approach to identify batching behavior, and introduced batch processing metrics to acquire knowledge on the batch characteristics and their influence on process execution. Pourbafrani et al. [33] designed an approach to generate automatically a system dynamics simulation model. The authors developed an algorithm to detect the relationships between specific system features such as arrival rates and waiting times using time windows and correlation measures. Popovics and Monostori [14] designed an approach for automatically gathering data from Programmable Logic Controllers (PLCs) with the aim to achieve simulation model generation capabilities. The approach is based on parsing the code of a PLC and derive useful information at an higher abstraction level. Choueiri et al. [34] proposed a predictive model with the aim to use PM to predict online the cycle-times in industrial environments.

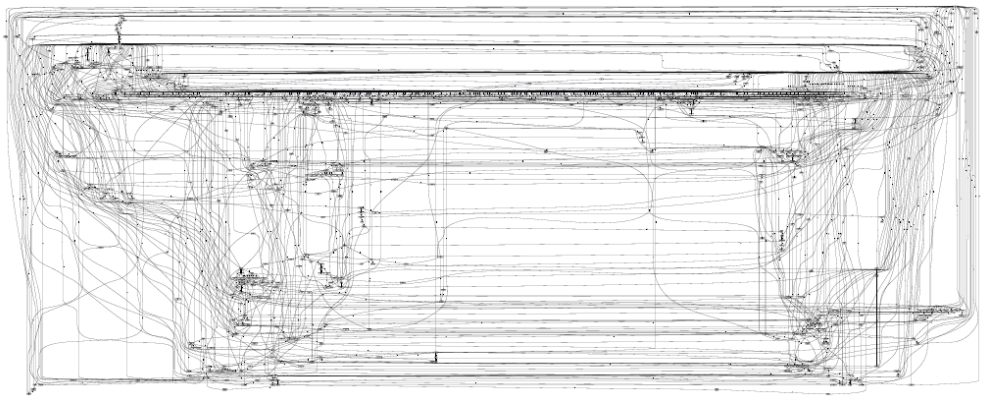

Figure 2: Example of a *spaghetti* model (from [35]).

### *2.2. Model Tuning*

The availability of an up-to-date process model can result to be insufficient since an automated model generation procedure could model some parts of the system with an excessive level of detail. A common term referring to over-complex models is the *spaghetti model* effect [35]. These models are not only excessive in terms of size, but also inaccurate in estimating performance measures. From intuition, the more we tend to represent each single path in the system, the less data are available to estimate parameters on that path such as processing and waiting times. Spaghetti models are also harder to understand, as they may differ significantly from the physical system generating the data. Therefore, once a model has been generated, it is worth noticing if the model level of detail is reasonable for its intended application. In its most basic form, model tuning is a method to modify a model toward a desired size.

Son et al. [36] introduced the concept of log profiling and trace clustering. The event log is split into homogeneous subsets for each observed case. Then, specific features are calculated for each case (for instance, the number of times a certain operator has intervened on that case) and a resulting vector of features is assigned to each case. Clustering techniques are then used to group common traces, and PM is applied separately on each cluster. Gunther and Van der Aalst [37] developed a fuzzy miner algorithm through an attribute analysis and a consequent abstraction of the mined event log. The method assigns metrics to nodes and arcs based on both significance and correlation scores. Then, three main steps are performed: (1) conflict resolution, in which activities which are connected by arcs in both directions are analyzed with the goal to remove the connection if less significant to the rest of the model; (2) edge filtering, where the least significant edges are removed; (3) nodes aggregation and abstraction: the least significant nodes are either removed or aggregated in clusters, which will inherit the precedence relationships and the connections between nodes. All the steps are based on edge and nodes cutoff parameters that are user-defined thresholds. Bose and Van der Aalst [38] introduced patterns definitions of commonly used process model constructs, together with an iterative method that captures these manifestations and creates abstractions. The method finds repeated patterns in the traces (e.g., loops) and generates hierarchical abstractions that are treated as activities and inserted in a new trace database. The corrected traces are then mined to generate a model. Prodel [39] developed a model reduction procedure to generate simulation models of a health-care provider. The author formalized the model reduction procedure with a mathematical programming model and solved it using a tabu search heuristic. The objective is to find the model that best fits the event log of the system based on replayability scores.

### 2.3. Discussion

Although PM-based approaches can be used effectively for adjusting the model level of detail, most of the available approaches are appropriate for business process mining and may not be optimal for manufacturing applications. For instance, fuzzy mining [37] exploits the activity names to derive correlation measures that can lead to activity clustering. This approach is certainly proper for a service operation (e.g., bank, call center) but may be of little use in production systems, where names could be simply sensor or machine identifiers. Trace clustering [36] effectively produces models with a lower complexity. However, this is only true for some of the discovered clusters, which are highlighting the entities following the simplest paths, while several other clusters which may be linked to complex production dynamics are still present in the log. Another option is the a-priori definition of patterns in the log [38], which is also helpful as pre-processing activity. Yet, it is strongly based on topology of the discovered relations, and has no clear link with the underlying system generating the data.

This work focuses on modeling discrete parts manufacturing systems [40]. Our focus is on how to properly build and tune digital models which are excessively exhaustive for the user purposes and have to be modified toward a reasonable level of detail. Simulation models for manufacturing systems typically represent components such as parts, resources (e.g., machines, conveyors, operators), and the paths along which the parts are flowing in the system. Hence, in a model tuning procedure, it is desirable to keep track of how much the main components of a simulation model are being represented and, possibly, reduced. Most PM contributions have defined fitness values that determine the ability of a model to reproduce the behavior in the event log [41]. However, simply being able to replay an event log may be insufficient if the resulting model produces bad predictions. In this work, we concentrate on the ability of the model to reproduce the characteristics and to estimate the performance of the system under study. Hence, we outline model generation starting from the PM-based discovery of the manufacturing system structure and characteristics. Then, we address the issue of model tuning by exploiting manufacturing systems properties such as buffer capacities and re-entrant material flows, as indicators of the degree with which a model is correctly representing the data log.

## 3. Manufacturing System Discovery

Figure 3 outlines the main steps covered by this work. Let us assume the manufacturing system is equipped with a Manufacturing Execution System that aggregates production data in an event log. The log is used to discover the manufacturing system logical structure and its parameters. A digital model $\Omega_0$ is built by a model generation procedure. Then, depending on the users requirements, the model can be tuned toward a desired level of detail. In this section, we outline the data-based system discovery and model generation, while in section 4 we present the proposed model tuning method.

### 3.1. The Event Log

Event logs are files that aggregate all the data produced by the manufacturing system. In general, the event log may contain several types of information, such as part flows, resources identifiers, and quality check outcomes. In this work, we assume the availability of event logs containing three main information types: (1) the activity identifier $n \in \mathbb{N}$, (2) the work-piece identifier $h \in \mathbb{H}$, and (3) the timestamps $t_S(n,h)$ and $t_F(n,h)$ indicating the moment at which the $n$-th activity has started and finished on the $h$-th piece, respectively. Further, we assume that the time span covered by the event log corresponds to the time horizon of interest. Let us define each row of the log as *event* and we define *trace* $\theta_h \in \Theta$ the set of events experienced by the $h$-th work-piece, where $\Theta$ is the set of all the traces identified from the log[1]. A trace is the specific

[1] In general: $|\Theta| \leq |\mathbb{H}|$.

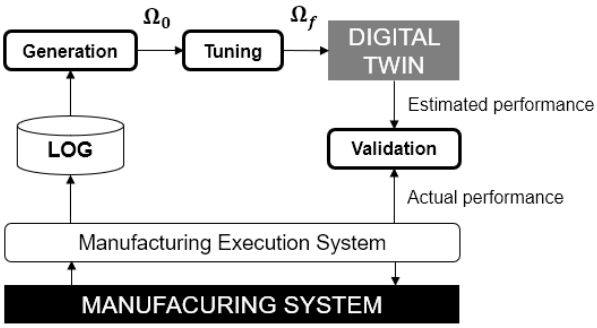


Figure 3: Graphical map of manufacturing system discovery and digital twin generation.

Table 1: Example of event log used in this work.

| **Time stamp** | **Activity** | **Part ID** | **Activity type** |
|---|---|---|---|
| 2020-11-12 09:31:32 | $S_1$ | 1 | start |
| 2020-11-12 09:31:36 | $S_1$ | 1 | finish |
| 2020-11-12 09:31:41 | $S_1$ | 2 | start |
| 2020-11-12 09:31:46 | $S_1$ | 2 | finish |
| 2020-11-12 09:31:48 | $C_1$ | 1 | |
| 2020-11-12 09:32:02 | $C_1$ | 2 | |
| 2020-11-12 09:32:12 | $S_2$ | 1 | start |
| 2020-11-12 09:32:24 | $S_2$ | 1 | finish |
| 2020-11-12 09:32:29 | $S_2$ | 2 | start |
| 2020-11-12 09:32:33 | $C_2$ | 1 | |
| 2020-11-12 09:32:34 | $S_2$ | 2 | finish |
| 2020-11-12 09:32:39 | $C_2$ | 2 | |
| 2020-11-12 09:32:44 | $S_3$ | 1 | start |
| 2020-11-12 09:32:58 | $S_3$ | 1 | finish |
| 2020-11-12 09:32:59 | $S_3$ | 2 | start |
| 2020-11-12 09:33:00 | $S_3$ | 2 | finish |

route that each part followed in the system. It can be expressed as a series of activity identifiers. Hence, each $h$-th part has a corresponding trace $\theta_h = \{n^{(1)}, n^{(2)}, \ldots, n^{(e_h)}, \ldots, n^{(\#_h)}\}$, where $\#_h$ is the number of the activities performed by the $h$-th part and $e_h$ indicates the sequential position of the activity as observed in the log for part $h$. Table 1 shows an example of event-log generated by the manufacturing system of Figure 1. In this example, the trace of part $h = 1$ is $\theta_1 = \{S_1, C_1, S_2, C_2, S_3\}$.

### *3.2. Digital Models*

We may represent a simulation model as a directed graph, in which nodes represent the manufacturing activities, and arcs represent the material flow relationships between the activities. Let us define a model $\Omega$ as a tuple $\Omega = (\mathbb{N}, \mathbb{A})$ where $\mathbb{N}$ is the set of nodes and $\mathbb{A} \subseteq \mathbb{N} \times \mathbb{N}$ is the set of arcs in the model. For instance, the graph model obtained in Figure 1c is defined by the set of nodes $\mathbb{N} = \{1, 3, 6\}$ and the set of arcs $\mathbb{A} = \{(1,3), (3,6)\}$. Nodes and arcs may also contain information about the system characteristics: the logical layout (i.e. precedences among activities), the capability of holding work-in-progress parts, the production volume over a certain time span, the routing policies, and the flow times. Hence, each node $n \in \mathbb{N}$ is a tuple $n = (\mathbb{P}_n, \mathbb{S}_n, \kappa_n, \phi_n, \pi_n, \xi_n, \tau_n)$, where $\mathbb{P}_n, \mathbb{S}_n$ are sets of predecessor and successor nodes,

Table 2: Notation for digital models.

| | | |
|---|---|---|
| **Nodes** $n \in \mathbb{N}$ | $\mathbb{P}_n$ | Predecessor nodes set. |
| | $\mathbb{S}_n$ | Successor nodes set. |
| | $\kappa_n$ | Buffer capacity of a node. |
| | $\phi_n$ | Frequency of a node. |
| | $\xi_n$ | Number of close events on a node. |
| | $\pi_n$ | Node branching policy tuples set. |
| | $T_n = \{\tau_{k,n}\}$ | Nodes flow times matrix. |
| **Arcs** $a \in \mathbb{A}$ | $\eta_a = (n,m)$ | Nodes connected by an arc. |
| | $c_a$ | Buffer capacity of an arc. |
| | $f_a$ | Number of events on an arc. |
| | $e_a$ | Number of close events on an arc. |
| | $T_a = \{t_{k,a}\}$ | Arc flow times matrix. |

respectively, $\kappa_n$ is the buffer capacity of the node, $\phi_n$ the frequency of occurrence of the respective activity, $\pi_n$ the set of branching probabilities, $\xi_n$ the number of events close in time, and $T_n$ the flow times matrix. Similarly, each arc $a \in \mathbb{A}$ is a tuple $a = (\eta_a, c_a, f_a, e_a, T_a)$, where $\eta_a$ is a tuple of connected nodes, $c_a$ the buffer capacity of the arc, $f_a$ the occurrence frequency, $e_a$ the number of close events, and $T_a$ the arc flow times matrix. Table 2 summarizes the notation for the nodes and arcs of the models used in the rest of this work. The properties of nodes and arcs are populated through the model generation procedure, which is described in the next section.

### *3.3. Model Generation*

Model generation is a procedure which links the data in the event log with a digital model $\Omega$. The first step is the identification of the *traces*. This procedure characterizes the possible sequences of events in the system. Starting from the event log, a unique set of activities $\mathbb{N}$ is created and the traces of all the $|\mathbb{H}|$ parts are identified. Then, the traces are used to retrieve precedence relationships among activities. Hence, a node exists in the model if a certain activity has been performed by at least one part, and an arc indicates that a production step has followed another in at least one trace. Specifically, arc $(n,m)$ exists if $\exists h \in \mathbb{H} | n,m \in \theta_h \wedge t_F(n,h) < t_S(m,h)$. In the following paragraphs, we elaborate on node and arc characteristics.

**Nodes**. A node identifies an activity. Each node is linked to its predecessors and successors nodes ($\mathbb{P}_n$, $\mathbb{S}_n$). Namely, two sets of nodes that represent the activities done before ($\mathbb{P}_n$) and after ($\mathbb{S}_n$) the $n$-th node, respectively. A node is also defined by the following properties: (1) capacity, (2) frequency, (3) close events. The capacity of a node $\kappa_n$ is defined as the maximum amount of work-pieces that can be processed together by the corresponding production activity. It can be estimated as follows:

$$\hat{\kappa}_n = \max k \,|\, t_S(n,h-k) \leq t_F(n,h) \quad \forall n \in \mathbb{N}. \tag{1}$$

The frequency $\phi_n$ indicates the number of times the corresponding activity has been observed in the log, while $\xi_n$ indicates the number of events that have been identified as close in time on the $n$-th node. In general, we define activities on the $n$-th node to be close in time if their time stamps satisfy $|t_F(n,h) - t_S(n,h)| \leq \zeta_N$, where $\zeta_N$ is a user-defined threshold. Additional parameters define the branching policies $\pi_n$ and the flow times. The former is a set of tuples of the kind $(s, p_s)$ where $s \in \mathbb{S}_n$ is the target node and $p_s$ the probability

that a work-piece will perform activity $s$ after the node $n$. The probability $p_s$ may be estimated as follows:

$$\hat{p}_s = \frac{\phi_s}{\sum_{o \in \mathbb{S}_n} \phi_o} \quad \forall s \in \mathbb{S}_n. \tag{2}$$

The flow times are described by a matrix, where each element indicates the time work-piece $h$ took to flow in node $n$. Hence:

$$\tau_{h,n} = t_F(n,h) - t_S(n,h) \quad \forall h \in \mathbb{H}, n \in \mathbb{N}. \tag{3}$$

**Arcs**. An arc is a connector between two nodes. The nodes connected by the $a$-th arc are collected in a tuple $\eta_a = (n,m) \subseteq \mathbb{N} \times \mathbb{N}$. We may conveniently identify an arc with its connected nodes tuple. The capacity $c_a$ of an arc is defined as the maximum amount of work-pieces that has resided on the arc at the same time, as retrieved from the event log. Namely:

$$c_a = \max k \,|\, t_F(n,h-k) \leq t_S(m,h) \quad \forall a \in \mathbb{A} \,|\, \eta_a = (n,m). \tag{4}$$

The number of work-pieces that have been observed flowing through the arc is indicated by $f_a$, while $e_a$ is the number of events that have been identified as close in time on the $a$-th arc, hence in which their time stamps satisfy $|t_S(m,h) - t_F(n,h)| \leq \zeta_A$, where $\zeta_A$ is a user-defined threshold and $\eta_a = (n,m)$. Flow times are defined by the matrix $T_a = \{t_{h,a}\}$ where each element indicates the time that the $h$-th work-piece took to flow in arc $a$:

$$t_{h,a} = t_S(m,h) - t_F(n,h) \quad \forall h \in \mathbb{H}, \forall a \in \mathbb{A} \,|\, \eta_a = (n,m). \tag{5}$$

Model generation identifies the nodes and arcs and collects them in an directed graph $\Omega_0$, which is populated with the properties of nodes and arcs through Algorithm 2 in Appendix A. Once a graph model is created, it can be converted into a simulation model that is able to estimate the system performance indicators such as production throughput or system time. Indeed, from process mining literature we know that a graph model has a Petri Net equivalent [42]. Among others, one of the conversion procedures that can be used is the $\alpha$-algorithm, which can be found in [35]. Notice that also simulation graphs (i.e., event relationship graphs) can be obtained, for instance exploiting direct conversion from Petri Nets [43]. Further details on model generation and conversion steps are available in related works [44, 45].

## 4. Model Tuning

We may define model tuning as a procedure that aims to adapt an existing model in order to satisfy complexity requirements, which can be expressed in terms of maximum number of nodes or arcs. In general, let us define $\Omega_0 = (\mathbb{N}_0, \mathbb{A}_0)$ the model generated by the procedure described in section 3.3, and $\Omega_j = (\mathbb{N}_j, \mathbb{A}_j)$ is a $j$-th alternative graph model. Among the alternative models satisfying the user-requirements in terms of level of detail, model tuning finds the one that maximizes an adequacy-related score. The score can be computed by the function $\Phi(\Omega)$ in equation (6), which determines how acceptably a model $\Omega$ represents the real production system.

$$\Phi(\Omega) = \sum_i w_i R_i(\Omega) \tag{6}$$

where each score $R_i(\Omega)$ is a function that describes *how well does the model* $\Omega$ *represent the i-th characteristic of the system*, and $w_i$ is the weight of the $i$-th score. This way, the score of a model is directly linked to the event log and the following properties of the manufacturing system: (1) buffer sizes, (2) events close in time, (3) re-entrant flows, (4) split and merge points, and (5) activity occurrence frequency. We have defined five $R_i(\Omega)$ functions which are listed in section 4.1. Table 3 summarizes the notation used for the model tuning.

Table 3: Notation for the proposed model tuning method.

| | |
|---|---|
| *Parameters:* | |
| $U_A^{max}$ | maximum number of arcs allowed: $U_A^{max} \in [1, \lvert\mathbb{A}_0\rvert]$. |
| $U_N^{max}$ | maximum number of nodes allowed: $U_N^{max} \in [2, \lvert\mathbb{N}_0\rvert]$. |
| $\mathbf{X} = \{\chi_{n,m}\}$ | matrix of precedences among activities: $\chi_{n,m}$ is 1 if $\exists n,m \in \mathbb{N}_0, h \in \mathbb{H} \lvert t_F(n,h) \leq t_S(m,h)$, 0 otherwise. |
| $L$ | boolean value that is 1 if self-loops are allowed, 0 otherwise. |
| $\nu_{in}$ | maximum number of input arcs in a node: $\nu_{in} \in [0, \max_{n\in\mathbb{N}_0}\{\mathbb{P}_n\}]$. |
| $\nu_{out}$ | maximum number of output arcs from a node: $\nu_{out} \in [0, \max_{n\in\mathbb{N}_0}\{\mathbb{S}_n\}]$. |
| $\iota_{ij}$ | number of loops in which the arc $(i, j)$ is involved. |
| *Decision Variables:* | |
| $\beta = \{\beta_i\}$ | vector of included activities: $\beta_i$ is 1 if the $i$-th activity is included in the graph, 0 otherwise. |
| $\Gamma = \{\gamma_{ij}\}$ | matrix of connectors between activities: $\gamma_{ij}$ is 1 if the $i$-th activity is followed by the $j$-th, 0 otherwise. |
| $\mathbf{M} = \{m_{ci}\}$ | matrix of cluster composition: $m_{ci}$ is 1 if the $c$-th cluster includes the $i$-th activity, 0 otherwise. |

### *4.1. Model Adequacy Score Functions*

The score functions that we have defined are the following:

- **Buffers.** $R_1$ is a function that represents the buffer capacity of the system. The aim is to favor the inclusion of nodes and arcs with a higher buffer capacity.

$$R_1(\Omega) = r_1^{(A)} \frac{\sum\limits_{a\in\mathbb{A}} c_a}{\sum\limits_{a\in\mathbb{A}_0} c_a} + r_1^{(N)} \frac{\sum\limits_{n\in\mathbb{N}} \kappa_n}{\sum\limits_{n\in\mathbb{N}_0} \kappa_n} \tag{7}$$

  where $r_1^{(N)}$ and $r_1^{(A)}$ balance the relative weight of nodes and arcs, respectively.

- **Close events.** $R_2$ is a function related to the number of events in the system which occur within a short time window. For example, in a production line the moment a part leaves a buffer may correspond to the recorded time it enters the downstream station. $R_2$ favors the exclusion of nodes and arcs related to activities that are done within short time frames.

$$R_2(\Omega) = \frac{r_2^{(A)}}{\lvert\mathbb{A}\rvert} \sum_{a\in\mathbb{A}} (1 - \frac{e_a}{f_a}) + \frac{r_2^{(N)}}{\lvert\mathbb{N}\rvert} \sum_{n\in\mathbb{N}} (1 - \frac{\xi_n}{\phi_n}) \tag{8}$$

  where $r_2^{(N)}$ and $r_2^{(A)}$ balance the relative importance of node and arc properties, respectively.

- **Re-entrant flows.** $R_3$ is related to the loops in the material flow. A loop may simply represent a station that withdraws parts from a conveyor, performs a certain activity, and then releases the parts back to the conveyor. Since the performance of the loop influences the production rate of the system,

it is desirable to preserve it in the representation. This function encourages the inclusion of parts in the model which are involved in loops.

$$R_3(\Omega) = \frac{1}{|\mathbb{A}|} \sum_{n \in \mathbb{N}} \sum_{m \in \mathbb{N}} \gamma_{nm} \iota_{nm} \tag{9}$$

where $\iota_{nm}$ is 1 if the arc $(n,m)$ is a portion of a loop in the model, 0 otherwise. Loops are identified as cycles in a directed graph $\Omega = (\mathbb{N}, \mathbb{A})$ with the algorithm defined in [46].

- **Split and merge.** $R_4$ is a routing score defined in [39]. For instance, in a manufacturing system the logic of the material flow may be defined by splitting or merging points along the physical conveyors, i.e. positions in which alternative activities following or preceding the $n$-th node are possible. In such locations, relevant decisions may have to be taken (e.g., prioritizing). Therefore, it may be desirable to keep the nodes with multiple connected arcs.

$$R_4(\Omega) = r_4^{(in)} \sum_{n \in \mathbb{N}} \sum_{x \in \mathbb{S}_n} \gamma_{nx} + r_4^{(out)} \sum_{n \in \mathbb{N}} \sum_{l \in \mathbb{P}_n} \gamma_{ln} \tag{10}$$

where $r_4^{(in)}$ and $r_4^{(out)}$ balance the relative significance of inbound and outbound arcs, respectively.

- **Frequency.** $R_5$ is a function representing the number of parts flowing in the nodes and arcs of the model. The aim is to favor the inclusion of nodes and arcs which are visited with a higher frequency (i.e. the preferred path).

$$R_5(\Omega) = r_5^{(A)} \frac{\sum_{a \in \mathbb{A}} f_a}{\sum_{a \in \mathbb{A}_0} f_a} + r_5^{(N)} \frac{\sum_{n \in \mathbb{N}} \phi_n}{\sum_{n \in \mathbb{N}_0} \phi_n} \tag{11}$$

where $r_5^{(N)}$ and $r_5^{(A)}$ balance the relative importance of nodes and arcs, respectively.

### *4.2. Mathematical Programming Formulation*

Without loss of generality, we assume that the *size* of a model is defined by the number of nodes and it is used as single criterion for pursuing model reduction goals. Let us define $U_N^{max}$ as the desired number of nodes in the model. If the desired model size is either equal or it exceeds the one of the initial model $\Omega_0$ (that is, if the number of nodes in the model satisfies the user-imposed constraint $|\mathbb{N}_0| \leq U_N^{max}$) the solution to the problem is trivial and it corresponds to $\Omega_0$. Otherwise, a reduced model has to be found. This can be obtained in two ways: (1) by reducing the number of nodes in the model, thus omitting certain activities, (2) by grouping the existing nodes and arcs in clusters $c \in \mathbb{C}$. A cluster is a node. Specifically, we indicate $m_{nc} = 1$ if node $n$ belongs to cluster $c$, 0 otherwise; an arc $a$ belongs to a cluster if its connected nodes also belong to it ($n,m \in \mathbb{C} \wedge n,m \in \eta_a$). The goal of model tuning is *to find the model* $\Omega = (\mathbb{N} \cup \mathbb{C}, \mathbb{A})$ *that maximizes the score* $\Phi$ *defined in equation* (6) *while satisfying a set of constraints (e.g., number of nodes).*

Model tuning may be expressed as a mathematical programming model through equations (12) - (22). The decision variables are defining the selection of nodes and arcs to keep or aggregate in the model. Specifically, $\beta_i$ defines if the $i$-th activity is kept in the representation or not, $\gamma_{ij}$ guarantees the precedences of activities are maintained. The clustering of nodes is represented by variable $m_{ic}$.

$$\max_{\Omega} \quad \Phi(\Omega) \tag{12}$$

$$s.t. \quad \sum_i \beta_i \ \leq \ U_N^{max} \tag{13}$$

$$\sum_{ij} \gamma_{ij} \ \leq \ U_A^{max} \tag{14}$$

$$\gamma_{ij} \leq \mathbf{MXM}^T \qquad \forall i,j \tag{15}$$

$$\sum_i m_{ik} \ \leq \ 1 \qquad \forall k \tag{16}$$

$$\gamma_{ij} \leq \beta_i \qquad \forall i,j \tag{17}$$

$$\gamma_{ij} \leq \beta_j \qquad \forall i,j \tag{18}$$

$$\gamma_{ii} \leq L \qquad \forall i \tag{19}$$

$$\sum_i \gamma_{ij} \leq \nu_{in} \qquad \forall j \tag{20}$$

$$\sum_j \gamma_{ij} \leq \nu_{out} \qquad \forall i \tag{21}$$

$$\beta_i, \gamma_{ij}, m_{ci} \ \in \{0,1\} \qquad \forall i,j,c. \tag{22}$$

The optimization function (12) represents the maximization of the model score expressed by equation (6). Constraints (13) and (14) limit the number of nodes and arcs included in the model, respectively. Constraints (15) translate the precedences among events to precedences among clusters. Constraints (16) force each activity to be included in at most one cluster. Constraints (17) and (18) guarantee that arcs are included only between existing nodes. Constraints (19) allow or forbid for self-loops depending on the value of $L$. Constraints (20) and (21) limit the number of input and output arcs from each node, depending on $\nu_{in}$ and $\nu_{out}$. Constraints (22) state the nature of the decision variables.

### *4.3. Solution Methodology*

The model tuning problem described in equations (12) - (22) is solved with the local search algorithm described in Algorithm 1. Algorithm 1 is based on the possibility to generate neighbors of a model. A neighbor is derived by means of either two moves: (1) aggregation, in which two nodes are merged in a cluster, (2) reduction, in which one node is removed from the model. The following section elaborates on the moves, while section 4.3.2 explains the local search algorithm.

#### *4.3.1. Removal and Aggregation Rules*

In both aggregation and reduction moves, new arcs may be added in order to maintain the flow of parts, and clusters may be added to represent node groups. In this case, the newly introduced elements inherit the properties of the removed parts. Figure 4 represents an example of this idea. In the reduction case, node 2 is removed from the model, hence also the connected arcs $(1,2)$ and $(2,3)$ have been removed. To guarantee the flow of parts, arc $(1,3)$ is added. The properties of the added arc are derived from the ones of all the removed elements. In the aggregation case, a new node cluster is created by merging nodes 1 and 2. We may define two support indicator functions: $\alpha(x,c)$ is 1 if element $x$ has been aggregated in cluster $c$, 0 otherwise; $\rho(x,a)$ is 1 if element $x$ has been removed and, as a consequence, arc $a$ has been added, 0 otherwise. We define the following inheritance rules:

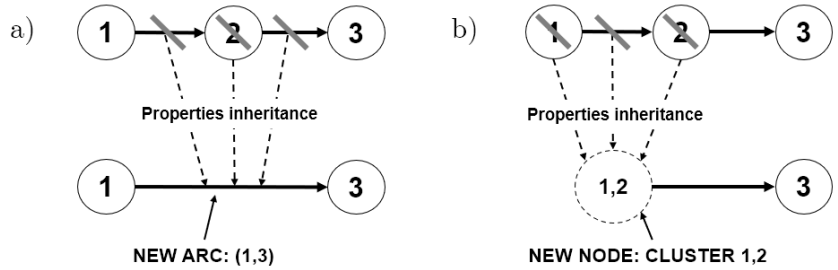


Figure 4: Properties inheritance example: a) reduction, b) aggregation.

- *Frequency*. New arcs or clusters inherit the maximum frequency of events observed in any of the removed elements.

  Reduction:

$$f_a = \max\{\max_{n\in\mathbb{N}}\{\phi_n\}, \max_{b\in\mathbb{A}}\{f_b\}\} \quad \forall n,b\,|\rho(n,a)=1\wedge\rho(b,a)=1. \tag{23}$$

  Aggregation:

$$\phi_c = \max\{\max_{n\in\mathbb{N}}\{\phi_n\}, \max_{a\in\mathbb{A}}\{f_a\}\} \quad \forall n,a\,|\alpha(n,c)=1\wedge\alpha(a,c)=1. \tag{24}$$

- *Contemporary Events*. New arcs or clusters inherit the minimum number of events close in time of the removed elements.

  Reduction:

$$f_a = \min\{\min_{n\in\mathbb{N}}\{\xi_n\}, \min_{b\in\mathbb{A}}\{e_b\}\} \quad \forall n,b\,|\rho(n,a)=1\wedge\rho(b,a)=1. \tag{25}$$

  Aggregation:

$$\xi_c = \min\{\min_{n\in\mathbb{N}}\{\xi_n\}, \min_{a\in\mathbb{A}}\{e_a\}\} \quad \forall n,a\,|\alpha(n,c)=1\wedge\alpha(a,c)=1. \tag{26}$$

- *Capacity*. New arcs or clusters inherit the sum of the capacity of the removed elements.

  Reduction:

$$c_a = \sum_{n,b\,|\rho(n,a)=1\wedge\rho(b,a)=1} \kappa_n + c_b \tag{27}$$

  Aggregation:

$$\kappa_c = \sum_{n,a\,|\alpha(n,c)=1\wedge\alpha(a,c)=1} \kappa_n + c_a \tag{28}$$

*4.3.2. Algorithm*

Algorithm 1 is a depth-first local search algorithm. At each step, a certain number of neighbors can be generated. The neighbors generation function is described by Algorithm 3 in Appendix A. Let us call $\sigma_A$ and $\sigma_R$ the number of neighbor models generated by aggregation and reduction, respectively. Hence, at each step, two sets of neighbors $\mathbb{M}_A$ and $\mathbb{M}_R$ are generated, and $|\mathbb{M}_A| = \sigma_A$, $|\mathbb{M}_R| = \sigma_R$. Let us accept the short notation $\mathbb{N}_i$, $\mathbb{A}_i$ indicating the node and arc sets of the model selected at the $i$-th iteration of Algorithm 1. At any $i$-th iteration, the complete exploration of the neighbors set is achieved when $\sigma_A = \sigma_R = |\mathbb{N}_i|$. Since

**Algorithm 1:** Local Search – Find the most adequate model $\Omega_f$ of size $U_N^{max}$.

**Input:** Graph model $\Omega_0$, size limit $U_N^{max}$;
**Output:** Graph model $\Omega_f$;
STEP 0: assign $\Omega_{current} \leftarrow \Omega_0$; $i \leftarrow 0$;
**while** $(i \leq I_{max}) \neg (\#_N(\Omega_{current}) \leq U_N^{max})$ **do**
    STEP 1: Generate $\sigma_A + \sigma_R$ neighbors of $\Omega_{current}$ (Algorithm 2);
    STEP 2: Sort neighbors based on score $\Phi$;
    STEP 3: Set $\Omega_{current} \leftarrow argmax_{\{\Omega \in \mathbb{M}_A \cup \mathbb{M}_R\}} \Phi(\Omega)$, $i \leftarrow i+1$;
**end**
**return** $\Omega_f \leftarrow \Omega_{current}$;

a neighbor is defined by the reduction of the model size by one node, $|\mathbb{N}_i|$ neighbors can be generated by means of reduction moves, while $|\mathbb{A}_i|$ neighbors by means of aggregation moves. For saving computation time, we may also set $\sigma_A, \sigma_R < |\mathbb{N}_i|$. As a consequence, it is necessary to define a neighbor generation rule, which defines which nodes are to be considered for either aggregation or reduction moves. We have defined three neighbor generation rules: (1) *Frequency*. Neighbors are generated by aggregating or reducing first the nodes with the lowest frequency; (2) *Contemporary Events*. Neighbors are generated by aggregating or reducing first the nodes with the highest number of contemporary events; (3) *Capacity*. This policy prescribes to aggregate or reduce first the nodes connected by arcs of the lowest capacity. It is worth to notice that other rules can be developed. The choice of which rule to observe depends on the intended use of the digital model and on the manufacturing system involved.

### *4.4. Parameters Estimation*

Once a final model $\Omega_f$ has been identified, the parameters governing the behavior of the physical system have to be estimated. The main parameters that influence the performance of a discrete parts manufacturing system are parts arrival behavior and part processing distributions. In this work, we assume both arrival times and processing times are available from the event log, and we use the Empirical Cumulative Distribution Function as estimate. Specifically, if a node does not belong to any cluster, the processing times of the respective activity $n$ can be derived from the event log simply by extracting the time the $h$-th part spent in activity with equation (3), whereas in case the nodes have been clustered by model tuning, we consider processing times of the cluster as the inter departure times from the cluster:

$$\tau_{h,n} = t_F(n,h) - t_F(n,h-1) \quad \forall n \in \mathbb{C}, h \in \mathbb{H}. \tag{29}$$

### *4.5. Control Policies Identification*

At the end of model tuning, a graph model $\Omega_f$ is obtained. Since model tuning may change both nodes and arcs, new split or aggregation points may have been created in the graph. Hence, control policies have to be estimated again by evaluating $\pi_n \forall n \in \mathbb{N} \cup \mathbb{C}$. It is worth to notice that the focus of this work is not on policies estimation, hence the identification of only frequency-based control policies is sufficient for our scopes. However, realistic settings can present a much higher level of complexity, and smarter mining algorithms can be developed for the identification of the control policies [27].

### 4.6. Note on Implementation

The previous sections outlined the proposed procedure to derive a digital twin of a manufacturing system. Once the proposed algorithms have been implemented, practicians can deploy the model development in production environments. In the following, summarize the main steps to be followed. Let us refer to $\lambda$ as a performance indicator of interest (e.g., system time, throughput). The user may decide: (1) The desired level of detail $U_N^{max}$; (2) an acceptable difference between the performance obtained by the digital model and the real system one, $\varepsilon_\lambda$. Namely, once a model is obtained, we may consider it as validated if equation (30) is statistically confirmed.

$$|\lambda_{real} - \lambda_{model}| \leq \varepsilon_\lambda . \tag{30}$$

Figure 5 represents a flow chart of the main steps that lead to a complete, usable digital twin. If a model in the format described in section 3.3 is already available, the user may directly consider Model Tuning, which is done if the level of detail is not representing the user expectations. If a model is tuned, its parameters and control policies have to be updated as well. Once an adjusted model is available, it can be converted and used to estimate system performance. Next, the model is validated. It is reasonable to assume that the user expectations in terms of performance will be related to the required level of detail. In the experiments of section 5, $\varepsilon$ will be quantified. If validation is passed, the model can be used as digital twin of the production system. Otherwise, a new model has to be generated, after a revision of the user's inputs.

## 5. Numerical Experiments

This section elaborates on the experiments done to validate the approach proposed in this work. Specifically, we have performed the following tests:

1. *Calibration*. The goal is to determine which configuration of weights contributes the most to the objective function of the model tuning problem described in section 4.2.
2. *Test Case 1: six-station flow line*. The scope is to verify the behavior of the proposed approach by comparing the performance estimated by the obtained simulation models with the real system one (i.e. validation).
3. *Test Case 2: six-station flow line with additional sensors*. The goal is to determine if records in the log from activities unrelated with the manufacturing process are correctly aggregated via model tuning.
4. *Test Case 3: real manufacturing line*. The objective is to apply the proposed approach to a realistic event log.

### 5.1. Calibration

The goal of this section is to determine which configuration of weights contributes the most to the objective function described in equation (12). The function is a weighted sum of the scores defined in section 4.1. Specifically, there are five weights $w$ and eight weights $r$. All the weights are in the interval $[0,1]$. Experiments have been done by taking as reference the manufacturing flow line with six stations depicted in Figure 6. The flow line processes one part type. Stations process parts in sequence and each station can process one part at a time. Each station $s$ is processing parts with processing times $p_s$. Part inter arrival times are described by the variable $p_a$. Inter arrival and processing times are distributed according to an exponential distribution with mean $1\,min$. Each station $s$ has an input buffer $C_s$. The first buffer is infinite, hence all arriving parts are accepted, while buffers $C_2$ to $C_6$ can store maximum 10 parts each. A simulation model of the flow line has been developed in Rockwell Arena and used to generate five independent event logs through simulation experiments representing the production of 1000 parts.

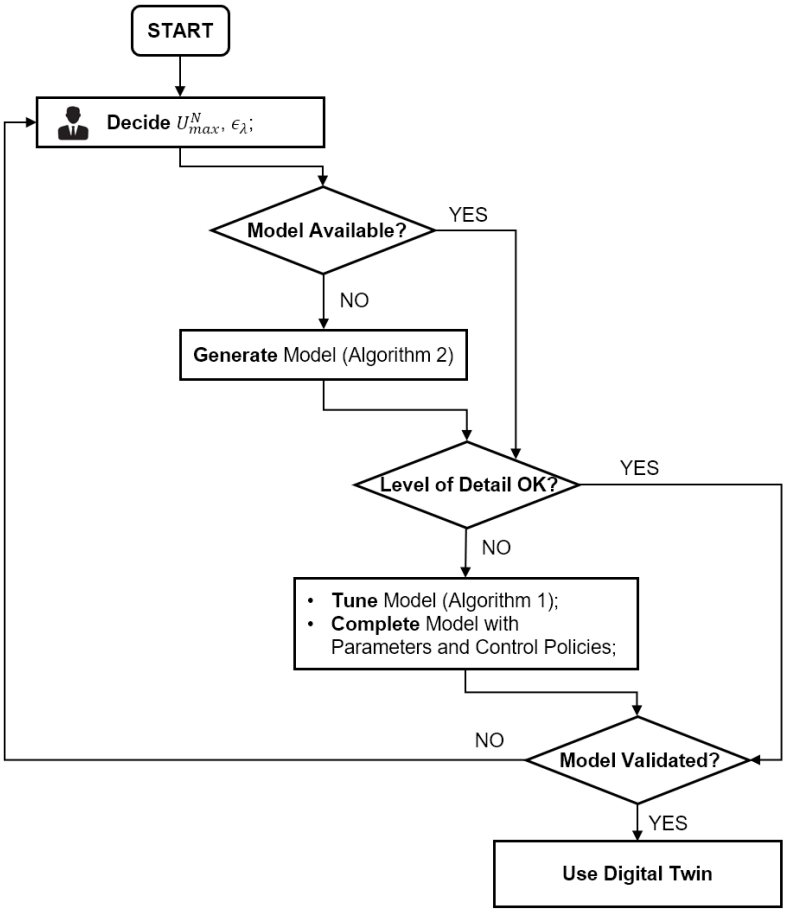


Figure 5: Graphical map of the implementation phase of the proposed approach.

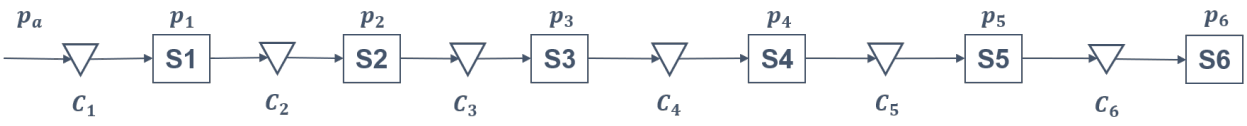


Figure 6: 6-station flow line used for the experiments in this work. Squares depict stations, while inter-operational buffers are represented as triangles.

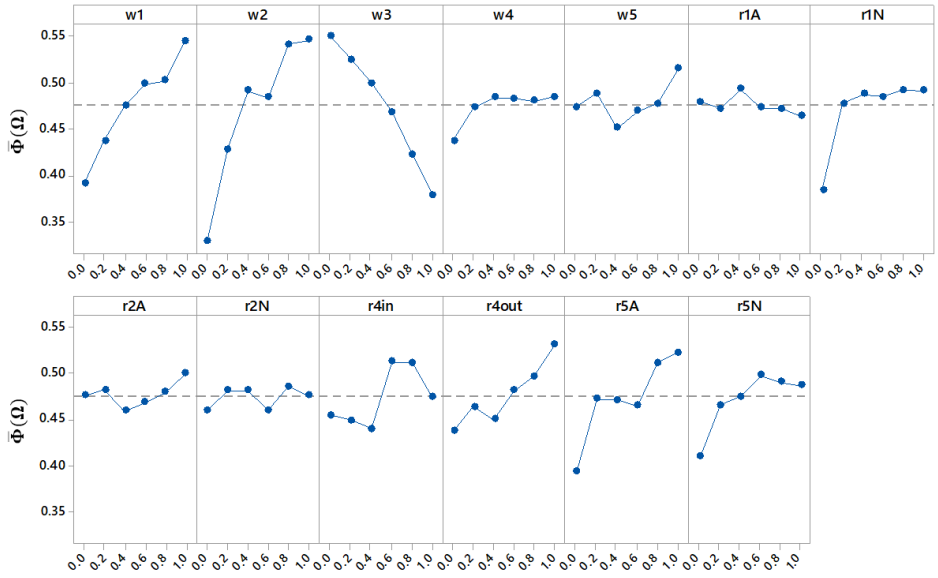


Figure 7: Calibration – Main effects plots of the $w$ and $r$ weights influence on the normalized objective function value $\bar{\Phi}(\Omega)$.

For the calibration experiments, we have exploited a space-filling design with a 256 points over the values of the weights of objective function (12). Each point has been replicated five times. The replications are defined by the event logs, which represent specific realizations of the system parameters and settings. Hence, in each replication, models have been generated from an independent log and tuned with $U_N^{max} = 3$. For each experimental point, the value of the objective function $\Phi(\Omega_f)$ has been calculated. Since the value of $\Phi$ depends on the particular configuration of the weights, we have used the normalized value $\bar{\Phi}$ for comparisons, defined as follows:

$$\bar{\Phi}(\Omega_f) = \frac{\sum_i w_i R_i(\Omega_f)}{\sum_i w_i}. \tag{31}$$

Figure 7 shows the main effects plots obtained in the calibration experiment, suggesting a high value for $w_1$, $w_2$, $w_4$, and $w_5$, whilst maintaining a low value for $w_3$. This is also coherent with the fact that $R_3$ is referred to loops, which are not present in the flow line. Table 4 presents the ANOVA analysis on the $w$ and $r$ weights. Considering a confidence level of 95%, $w_1$, $w_2$, and $w_3$ are significantly contributing to the objective function value. The fact that $w_4$ and $w_5$ are not significant is reasonable since $R_4$ is collecting the influence of routing nodes, which are not present in a flow line, and $R_5$ depends on the frequency on the nodes and arcs, which is constant for the flow line since no branching points are present. The influence of the $r$ scores is less clear from the graph, while the ANOVA table suggests that all of them have an influence except for $r_1^{(A)}$ and $r_2^{(N)}$. Given the aforementioned results, the following experiments have been done using: $w_1 = w_2 = 1$, $w_3 = 0$, $w_4 = w_5 = 0.5$ and $r_1^{(A)} = 0$, $r_1^{(N)} = r_2^{(A)} = 1$, $r_2^{(N)} = 0$, $r_4^{(in)} = 0.8$, $r_4^{(out)} = r_5^{(A)} = 1$, $r_5^{(N)} = 0.6$.

### *5.2. Test Case 1: 6-station flow line*

The first test case has been developed using the simulation model of the 6-station flow line described in section 5.1. The model has been used to generate 10 independent event logs, each representing the production of 1000 pieces. The objective of the experiment is to verify if a reduced model can still correctly

Table 4: Calibration – ANOVA table for the objective function weights $w$ and $r$.

| Source | DF | Adj. SS | Adj. MS | F-Value | P-Value |
|---|---|---|---|---|---|
| $w_1$ | 5 | 1.6803 | 0.33605 | 83.54 | 0.000 |
| $w_2$ | 5 | 3.9386 | 0.78773 | 195.81 | 0.000 |
| $w_3$ | 5 | 2.8950 | 0.57899 | 143.93 | 0.000 |
| $w_4$ | 5 | 0.0114 | 0.00227 | 0.57 | 0.727 |
| $w_5$ | 5 | 0.0827 | 0.01654 | 4.11 | 0.001 |
| $r_1^{(A)}$ | 5 | 0.0591 | 0.01182 | 2.94 | 0.012 |
| $r_1^{(N)}$ | 5 | 0.7786 | 0.15573 | 38.71 | 0.000 |
| $r_2^{(A)}$ | 5 | 0.1063 | 0.02127 | 5.29 | 0.000 |
| $r_2^{(N)}$ | 5 | 0.0575 | 0.01150 | 2.86 | 0.014 |
| $r_4^{(in)}$ | 5 | 0.6497 | 0.12994 | 32.30 | 0.000 |
| $r_4^{(out)}$ | 5 | 0.5449 | 0.10899 | 27.09 | 0.000 |
| $r_5^{(A)}$ | 5 | 0.5834 | 0.11668 | 29.00 | 0.000 |
| $r_5^{(N)}$ | 5 | 0.6181 | 0.12363 | 30.73 | 0.000 |
| Error | 1214 | 4.8838 | 0.00402 | | |
| Total | 1279 | 19.9532 | | | |

estimate the performances of the real system. This has been done by varying the required size $U_N^{max}$ in the interval $[4,6]$. Hence, for each log, three alternative models have been generated. In order to validate the generated models, each of them has been used to simulate the production of 1000 pieces in experiments replicated five times each. For this purpose, we have converted each obtained graph model $\Omega_f$ into a simulation model in Rockwell Arena, which we used to estimate the performances in terms of throughput $TH\,[parts/min]$ and system time $ST\,[min]$. Table 5 summarizes the obtained performances (mean values among the five replications). The maximum computation time to obtain a tuned model is $43.29\,s$. The warm up period has been identified with the Marginal Standard Error Rule [47]. Figure 8 shows the results obtained for the fifth log of the original system, where the throughput and system time of the original system are marked with a red dashed line and are $0.846\,parts/min$ and $29.0\,min$, respectively. We can notice how both system time and throughput are changing depending on which model version we are using to simulate. As expected, the estimated performances are worsening with smaller dimensions of the models. However, it is worth to notice that for the most reduced model ($U_N^{max} = 4$), the mean absolute error is 5% for the throughput and 7% for the system time. These errors can be acceptable depending on the intended use of the digital models. For instance, the model could be used for initial estimations and definition of lower or upper bounds on decision variables. Further, in the best cases observed, the mean absolute error is 0.23% for the throughput and 0.62% for the system time.

*Increasing the log length.* Next, we have investigated how the availability of data could influence the obtained results. For this purpose, we have used the original system for a simulation experiment producing 20000 parts. The resulting event log has been used to generate digital models with $U_N^{max} = 6$ (i.e. the most accurate). In each case, we have used a different number of data points. Specifically, we have selected the event log partition of the first $H_{max}$ parts, where $H_{max} \in \{10, 100, 1000, 10000, 20000\}$. Then, we have used the generated model to estimate: (1) the throughput, (2) the system time, (3) the buffer capacities, and (4) the processing times. Figure 9 summarizes the obtained results. Figure 9a represents the throughput. It is

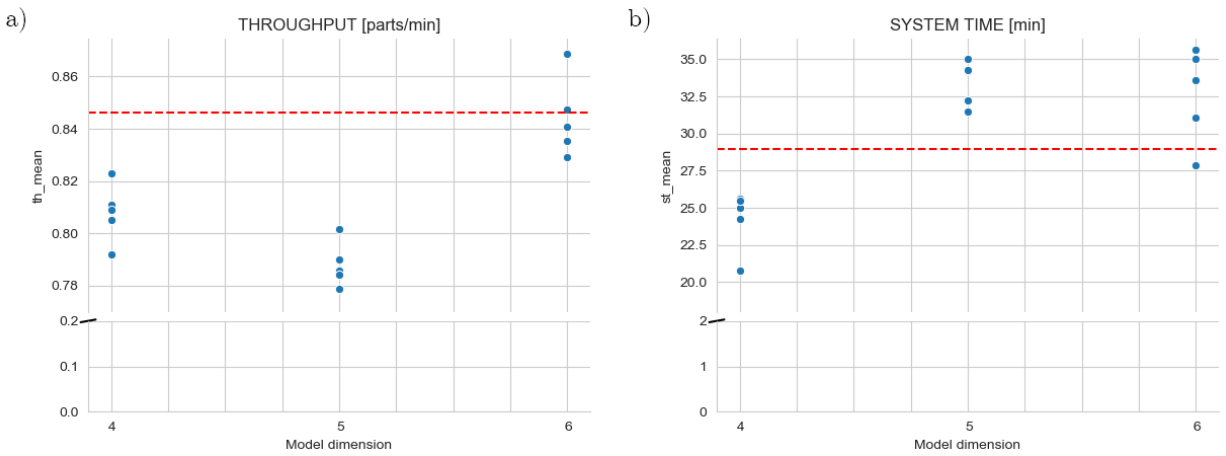


Figure 8: Test Case 1 – Results of the experiments in the flow line case (fifth event log): a) throughput; b) system time.

possible to see that 20000 parts are needed to reach an accurate throughput estimate. Figure 9b suggests that for the system time 10000 parts may be sufficient. Figure 9c shows how the estimation of the buffer capacities changes depending on the number of parts observed. The figure suggests that for reaching the correct buffer capacity estimation (i.e. $C_s = 10 \forall s \in [2,6]$), the data from 1000 parts are necessary. Figure 9d shows the Cumulative Distribution Function of the processing time $p_3$. From the figure it is possible to notice that data points from 1000 parts are needed to be close to the real function.

### *5.3. Test Case 2: 6-station flow line with sensor inputs*

The second test case has the goal to check if the tuning procedure is properly aggregating the model components. The experiment has been done by adding two virtual sensors to the second station of the line, each reporting to the event log. Figure 10a depicts the second station configuration, in which we have used the notation $S_{2,1}$ for the manufacturing operation, while $S_{2,2}$ and $S_{2,3}$ represent the passage of parts at the sensor devices. The recording operation takes a deterministic duration equal to one hundredth of the mean processing time ($p_{2,2} = p_{2,3} = \bar{p}_{2,1}/100$). These sensors may represent PLCs which record additional information, such as electrical tension values or quality check results. Although adding rows to the event log, these recordings are not relevant to the manufacturing operations and do not alter the system performance. Figure 10b represents the obtained model with $U_N^{max} = 6$. It can be noticed that the model tuning has correctly aggregated nodes from operations very close in terms of time. The obtained model is equivalent to the one derived in section 5.2 with $U_N^{max} = 6$, hence the same considerations regarding the performance estimation are valid.

### *5.4. Test Case 3: real manufacturing line*

In this test, we aim to test the behavior of the proposed model tuning method in the development of a model for a real production system. The analyzed system is an automatic assembly line of electric motor stators for hybrid vehicles. The stator is composed by several components and its assembly process consists of 18 main operations. Table 6 summarizes the assembly process phases. The stator production line is a closed loop line composed by several work stations. Each station follows a first come first served rule and can work one piece at a time. Stations are connected one another by conveyors. Each stator sub-assembly is placed on a dedicated pallet which moves on the conveyors. A portion of each conveyor is used as inter-operational buffer, and the buffer capacity is defined by the position of a dedicated sensor. Figure 11 shows

Table 5: Test Case 1 – Results obtained in terms of throughput ($TH$) and system time ($ST$) for both the complete and reduced models. For all the experiments, the Absolute Error ($AE$) is provided. The maximum half width confidence interval is 0.2% for the throughput and 2.8% for the system time.

| | | **Event logs** | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | $U_N^{max}$ | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | **Mean** |
| $TH$ | *Original* | 0.849 | 0.828 | 0.860 | 0.868 | 0.846 | 0.862 | 0.812 | 0.818 | 0.864 | 0.813 | 0.842 |
| | 4 | 0.780 | 0.807 | 0.831 | 0.820 | 0.808 | 0.835 | 0.729 | 0.763 | 0.810 | 0.809 | 0.799 |
| $TH$ | 5 | 0.801 | 0.788 | 0.822 | 0.822 | 0.788 | 0.795 | 0.772 | 0.769 | 0.812 | 0.774 | 0.794 |
| | 6 | 0.824 | 0.823 | 0.848 | 0.845 | 0.844 | 0.849 | 0.807 | 0.810 | 0.843 | 0.842 | 0.834 |
| | 4 | 0.069 | 0.021 | 0.028 | 0.048 | 0.038 | 0.028 | 0.083 | 0.055 | 0.054 | 0.004 | 0.043 |
| AE | 5 | 0.048 | 0.040 | 0.037 | 0.046 | 0.058 | 0.067 | 0.040 | 0.050 | 0.053 | 0.039 | 0.048 |
| | 6 | 0.025 | 0.005 | 0.011 | 0.022 | 0.002 | 0.014 | 0.005 | 0.008 | 0.021 | 0.029 | 0.014 |
| $ST$ | *Original* | 30.9 | 33.2 | 37.0 | 31.4 | 29.0 | 38.0 | 34.5 | 34.0 | 30.9 | 32.1 | 33.1 |
| | 4 | 29.0 | 27.2 | 24.4 | 25.4 | 24.2 | 26.1 | 30.9 | 27.0 | 26.1 | 26.6 | 26.7 |
| $ST$ | 5 | 36.4 | 32.5 | 33.5 | 30.5 | 33.4 | 34.4 | 38.6 | 36.6 | 30.7 | 34.6 | 34.1 |
| | 6 | 34.7 | 31.7 | 33.7 | 33.0 | 32.6 | 34.6 | 31.9 | 35.1 | 33.6 | 32.3 | 33.3 |
| | 4 | 1.9 | 6.1 | 12.5 | 5.9 | 4.7 | 11.8 | 3.5 | 7.0 | 4.8 | 5.5 | 6.4 |
| AE | 5 | 5.5 | 0.7 | 3.5 | 0.8 | 4.5 | 3.6 | 4.2 | 2.7 | 0.2 | 2.6 | 2.8 |
| | 6 | 3.8 | 1.5 | 3.2 | 1.6 | 3.7 | 3.3 | 2.6 | 1.1 | 2.7 | 0.2 | 2.4 |

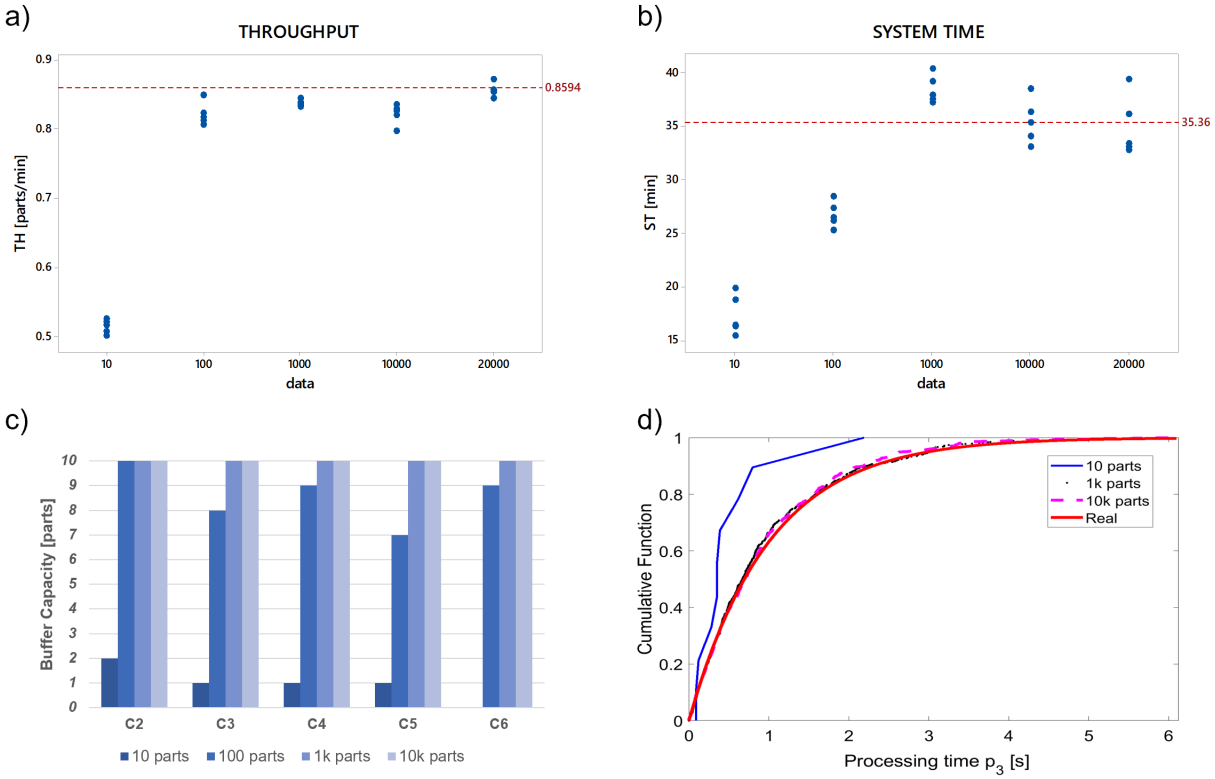


Figure 9: Test Case 1 – Results of the experiments with different number of input data points: a) throughput; b) system time, c) buffer capacities, d) cumulative distribution function of processing time $p_3$.

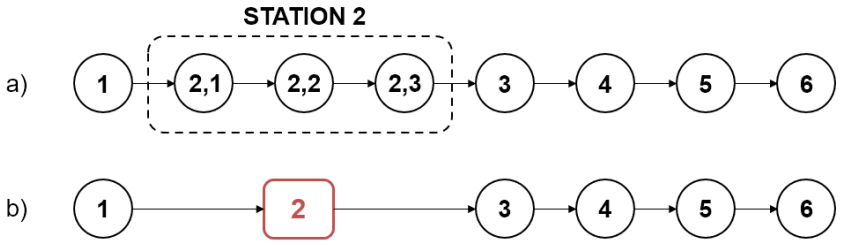


Figure 10: Test Case 2 – a) discovered graph for the original system; b) tuned model graph obtained with $U_N^{max} = 6$.

Table 6: Test Case 3 – Description of the processes on the electric motor assembly line.

| Op. Number | Process Description | Op. Number | Process Description |
|---|---|---|---|
| 0 | Load | 9 | Press |
| 1 | Marking Check | 10 | Welding |
| 2 | Paper Insertion | 11 | Blowing |
| 3 | Assembly (3 parallel stations) | 12 | Electrical Tests Pre-finishing |
| 4 | Forming (2 steps) | 13 | Finishing |
| 5 | Forming (2 steps) | 14 | Electrical Tests Post-finishing |
| 6 | Material removal | 15 | Visual Quality Check |
| 7 | Welding | 16 | Rework station |
| 8 | Assembly | 17 | Unload |

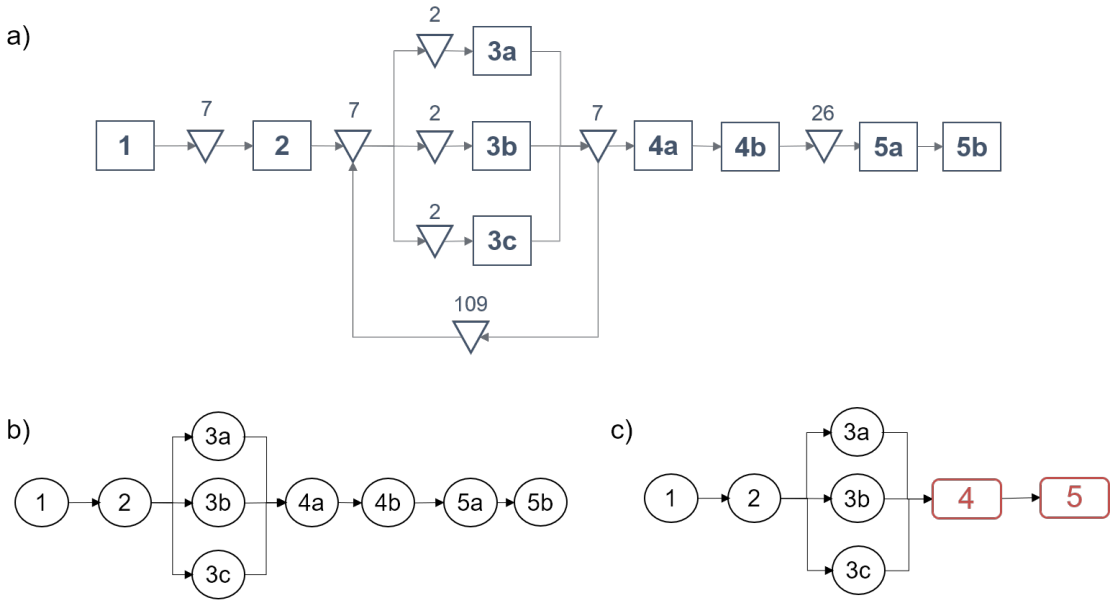


Figure 11: Test Case 3 – a) Layout of the stator assembly line (section for which the event log was made available). Stations are represented by squares, while inter-operational buffers are depicted as triangles. – Graph models obtained for the stator assembly line: b) complete model ($|\mathbb{N}| = 9$); c) tuned model ($U_N^{max} = 7$).

Table 7: Test Case 3 – Extract from the available event log for two work-pieces (Part IDs have been coded for confidentiality reasons).

| Part ID | Activity | Time stamp (start) | Time stamp (finish) | Result | Scrap |
|---|---|---|---|---|---|
| 1 | 1 | 2017-12-18 14:49:56 | 2017-12-18 14:52:11 | OK | NO |
| 2 | 1 | 2017-12-18 14:52:00 | 2017-12-18 14:53:44 | OK | NO |
| 1 | 2 | 2017-12-18 14:54:13 | 2017-12-18 14:54:20 | OK | NO |
| 2 | 2 | 2017-12-18 14:54:45 | 2017-12-18 14:54:53 | OK | NO |
| 1 | 3c | 2017-12-18 15:32:11 | 2017-12-18 15:36:03 | OK | NO |
| 1 | 4a | 2017-12-18 15:43:29 | 2017-12-18 15:43:54 | OK | NO |
| 1 | 4b | 2017-12-18 15:43:55 | 2017-12-18 15:44:14 | OK | NO |
| 2 | 3c | 2017-12-18 16:00:10 | 2017-12-18 16:04:03 | OK | NO |
| 2 | 4a | 2017-12-18 16:07:26 | 2017-12-18 16:07:51 | OK | NO |
| 2 | 4b | 2017-12-18 16:07:52 | 2017-12-18 16:08:11 | OK | NO |
| 1 | 5a | 2017-12-18 16:24:50 | 2017-12-18 16:25:47 | OK | NO |
| 1 | 5b | 2017-12-18 16:24:50 | NA | OK | NO |
| 2 | 5a | 2017-12-18 16:43:12 | 2017-12-18 16:44:11 | OK | NO |
| 2 | 5b | 2017-12-18 16:44:24 | 2017-12-18 16:45:31 | OK | NO |

the portion of the manufacturing system for which the event log was made available by the company, and table 7 shows an extract of the log.

The available event log is composed by 5000 events spanning two days of regular production. The company has used this log to estimate the effective cycle times of the assembly process. From the log we can observe that the number of recorded operations is 9. This is because operations 4 and 5 are effectively composed by two consecutive steps. Further, operation 3 can be performed in parallel over three different locations, hence there is a material flow split between operations 2 and 3, and the flow is merged again before operation 4. The model generation procedure and the tuning method proposed respectively in sections 3 and 4 have been used to build a digital twin of the real system. Figure 11b shows the obtained complete model of the assembly line section, while Figure 11c illustrates the tuned model representing the line section with $U_N^{max} = 7$ nodes. As expected, the nodes corresponding to similar operations (i.e., consecutive manufacturing steps) have been aggregated. Notice that – in this case – the assembly process layout is available, hence it is known that the analyzed section consists of 7 main operations. Therefore, further reductions would not be wise since they could exclude real process phases from the representation. This experiment has demonstrated that the model generation and tuning methodology defined in this work can be effectively used to discover the manufacturing system structure and develop its simulation model, aggregating activities which are correlated in the physical process.

## 6. Conclusions

The capability to generate an accurate discrete event simulation model in a short time is essential to achieve digital twin potentials. In this work, we described a method for discovering production systems structures and automatically develop digital models starting from the event logs of manufacturing systems. The proposed automated generation and tuning method can positively contribute to Real-time Simulation applications, since it guarantees that an updated and reasonably detailed model of the physical system can be obtained at any time within one minute and with minimal manual intervention. Being data-driven, a reasonable implementation guidance is to integrate the logic within a production control system such as a

Manufacturing Execution System. Thanks to the recent Industry 4.0 revolution, digitization efforts are being undertaken by both large businesses and small and medium enterprises. As a consequence, we believe a large number of users will have access to the proposed approach capabilities.

Several limitations still have to be solved. The proposed approach is application-agnostic and can be applied to several types of manufacturing systems. However, there may be significant differences among different system types. For instance, job shops are characterized by several independent part flows, which may result in a more complex identification of the system structure. At the same time, the estimation of certain parameters might be more straightforward; for example, the buffer sizes in a job shop may be considered as infinite. Further work shall be dedicated to develop different features of the approach targeting specific production systems. In this work, we have used a unique part identifier. As a consequence, the model generation procedure and the tuning method are not suited for assembly/disassembly lines in which multiple components may be tracked, and new part assembly identifiers may be added along the production. In this work, we have investigated relatively small model reductions. Future works shall also investigate the resilience and outcomes of realistic applications where production systems could be composed by hundreds of activities. For this purpose, specific rules may have to be developed. For instance, smarter model tuning moves could aggregate parallel activities or remove more than two nodes at a time if the model structure suggests it. In addition, the developed local search algorithm can be enhanced toward computational efficiency and the avoidance of local optima. Moreover, the correct choice of the specific model tuning rule is not trivial and this requires an assessment of the advantages of using a specific rule or objective function $\Phi$ depending on the type of manufacturing system. Further, in this work we have assumed that both inter arrival times and processing times can be estimated from the event log. In general, a situation of partial knowledge is much more realistic. For instance, the buffer capacities might be available in advance, while the processing times of certain stations might be unknown. A systematic study on the value of prior information has to be done. Another assumption in this work is the inclusion of only flow split policies, and the consequent identification of routing rules solely based on the frequency. The discovery of control policies can be extended, for instance by introducing machine learning algorithms in the model tuning framework. Also the identification of priority rules for assembly points can be addressed by future works. Notice also that model tuning can also be performed on simulation graphs [48], the proper balance between the two methods shall be examined. Last but not least, in this work we have tested the proposed approach with a real case spanning two production days. Given the data-driven model generation, the accuracy of the model may be superiorly limited by the recorded events in the log. Minimal requirements in terms of observed time of production shall be investigated.

## APPENDIX A

In this appendix, we present two of the algorithms that have been developed for model tuning. Algorithm 2 is used to generate an initial model $\Omega_0$ starting from an event log. Algorithm 3 produces a set of $\sigma_A + \sigma_R$ neighbors of an input model $\Omega$. *OrderNodesArcs* is a function that returns the ordered lists of nodes and arcs, based on the ranking that satisfies the user-defined priority rule. *GenerateNeighborByAggregation*($\Omega$, $(n,m)$) is a function that generates a neighbor of model $\Omega$, starting from the aggregation of nodes $(n,m)$ into a cluster, while respecting the inheritance rules defined in section 4.3.1. Similarly, *GenerateNeighborByReduction*($\Omega$, $n$) is a function that generates a neighbor of model $\Omega$, starting from the removal of node $n$, while respecting the inheritance rules defined in section 4.3.1.

**Algorithm 2:** Model Generation – Find the initial model $\Omega_0$.

**Input:** Event Log;
**Output:** Graph model $\Omega_0$;
**Definition:** $Count(i|Condition)$ returns the number of times $Condition$ is satisfied for element $i$.
STEP 1 – Generate traces set $\Theta$ and unique activities set $\mathbb{N}$;
STEP 2 – Activity relations:
    **for** $n \in \mathbb{N}$ **do**
        **for** $\theta_h \in \Theta$ **do**
            **for** $j \leftarrow 1$**to**$\#_i - 1$ **do**
                $\mathbb{A} \leftarrow \mathbb{A} \cup \{(n^{(j)}, n^{(j+1)})\}$;
                $\mathbb{S}_{n^{(j)}} \leftarrow \mathbb{S}_{n^{(j)}} \cup \{n^{(j+1)}\}$;
                $\mathbb{P}_{n^{(j+1)}} \leftarrow \mathbb{P}_{n^{(j+1)}} \cup \{n^{(j)}\}$;
            **end**
        **end**
    **end**
STEP 3 – Calculate frequencies:
    **for** $h \in \mathbb{H}$ **do**
        **for** $event \in \theta_h$ **do**
            **for** $n \in \mathbb{N}$ **do**
                **if** $n \in \theta_h$ **then**
                    $\phi_n = Count(n|n \in \theta_h)$
                **end**
            **end**
            **for** $a \in \mathbb{A}$ **do**
                **if** $(n,m) \in \theta_h \,|\, \eta_a = (n,m) \wedge j(n) = j(m) - 1$ **then**
                    $f_a = Count(n|n \in \theta_h)$
                **end**
            **end**
        **end**
    **end**
STEP 4 – Calculate buffer capacities:
    **for** $n \in \mathbb{N}$ **do**
        $n \leftarrow \kappa_n = \max k \,|\, t_S(n, h-k) \leq t_F(n,h)$;
    **end**
    **for** $a \in \mathbb{A}$ **do**
        $a \leftarrow c_a = \max k \,|\, t_F(n, h-k) \leq t_S(m,h) \quad n,m|\eta_a = (n,m)$;
    **end**

**Algorithm 2:** Model Generation – Find the initial model $\Omega_0$ (continued).

STEP 5 – Calculate contemporary events:
  **for** $a \in \mathbb{A}$ **do**
    $a \leftarrow e_a = Count(h \,|\, |t_S(m,h) - t_F(n,h)| \leq \zeta_A)$;
  **for** $n \in \mathbb{N}$ **do**
    $n \leftarrow \xi_n = Count(h \,|\, |t_S(n,h) - t_F(n,h)| \leq \zeta_N)$;
STEP 6 – Calculate branching policies:
  **for** $n \in \mathbb{N}$ **do**
    **for** $s \in \mathbb{S}_n$ **do**
      $\hat{p}_s = \dfrac{\phi_s}{\sum_{o \in \mathbb{S}_n} \phi_o}$;
      $\pi_n \leftarrow \pi_n \cup \{(s, \hat{p}_s)\}$
    $n \leftarrow \pi_n$
**return** $\Omega_0 \leftarrow \{\mathbb{N}, \mathbb{A}\}$;

**Algorithm 3:** Model Tuning – Generate neighbors of model $\Omega$.

**Input:** Model $\Omega$, number of neighbors from aggregation $\sigma_A$, number of neighbors from reduction $\sigma_R$, priority rule;
**Output:** neighbor model sets $\mathbb{M}_A, \mathbb{M}_R$ ;
$\{\mathbb{N}', \mathbb{A}'\} \leftarrow OrderNodesArcs(\Omega, priority\_rule)$;
**while** $|\mathbb{M}_A| < \sigma_A$ **do**
  Take first two nodes $n, m$ in $\mathbb{N}'$;
  $\Omega' \leftarrow GenerateNeighborByAggregation(\Omega, (n,m))$;
  $\mathbb{M}_A \leftarrow \Omega'$, $\mathbb{N}' \leftarrow \mathbb{N}' \setminus \{n,m\}$;
**end**
**while** $|\mathbb{M}_R| < \sigma_R$ **do**
  Take first node $n$ in $\mathbb{N}'$;
  $\Omega' \leftarrow GenerateNeighborByReduction(\Omega, n)$;
  $\mathbb{M}_R \leftarrow \Omega'$, $\mathbb{N}' \leftarrow \mathbb{N}' \setminus \{n\}$;
**end**
**return** $\mathbb{M}_A, \mathbb{M}_R$;

## References


[1] Mengnan Liu, Shuiliang Fang, Huiyue Dong, and Cunzhi Xu. Review of digital twin about concepts, technologies, and industrial applications. *Journal of Manufacturing Systems*, 2020.

[2] Fei Tao, Nabil Anwer, Ang Liu, Lihui Wang, Andrew Y.C. Nee, Liming Li, and Meng Zhang. Digital twin towards smart manufacturing and industry 4.0. *Journal of Manufacturing Systems*, 2020.

[3] Xingzhi Wang, Yuchen Wang, Fei Tao, and Ang Liu. New paradigm of data-driven smart customisation through digital twin. *Journal of Manufacturing Systems*, 2020.

[4] Rakesh Kumar Phanden, Priavrat Sharma, and Anubhav Dubey. A review on simulation in digital twin for aerospace, manufacturing and robotics. *Materials Today: Proceedings*, 2020.

[5] Giovanni Lugaresi and Andrea Matta. Real-time simulation in manufacturing systems: Challenges and research directions. In *2018 Winter Simulation Conference (WSC)*, pages 3319–3330. IEEE, 2018.

[6] 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*, 65(2):621–641, 2016.

[7] Giovanni Lugaresi, Vincenzo Valerio Alba, and Andrea Matta. Lab-scale models of manufacturing systems for testing real-time simulation and production control technologies. *Journal of Manufacturing Systems*, 58:93 – 108, 2021.

[8] Anders Skoogh, Terrence Perera, and Björn Johansson. Input data management in simulation–industrial practices and future trends. *Simulation Modelling Practice and Theory*, 29:181–192, 2012.

[9] Angappa Gunasekaran, Bharatendra K Rai, and Michael Griffin. Resilience and competitiveness of small and medium size enterprises: an empirical research. *International journal of production research*, 49(18):5489–5509, 2011.

[10] Fei Tao, Qinglin Qi, Ang Liu, and Andrew Kusiak. Data-driven smart manufacturing. *Journal of Manufacturing Systems*, 48:157 – 169, 2018. Special Issue on Smart Manufacturing.

[11] Heiner Reinhardt, Marek Weber, and Matthias Putz. A Survey on Automatic Model Generation for Material Flow Simulation in Discrete Manufacturing. *Procedia CIRP*, 81:121–126, jan 2019.

[12] T A Spedding, W L Lee, R De Souza, and S S G Lee. Adaptive simulation of a keyboard assembly cell. *Integrated Manufacturing Systems*, 8(1):50–58, 1997.

[13] Wil M.P. van der Aalst. Process mining and simulation: A match made in heaven! In *Simulation Series*, volume 50, pages 39–50. The Society for Modeling and Simulation International, 2018.

[14] Gergely Popovics and László Monostori. ISA standard simulation model generation supported by data stored in low level controllers. *Procedia CIRP*, 12:432–437, 2013.

[15] Riku-Pekka Nikula, Marko Paavola, Mika Ruusunen, and Joni Keski-Rahkonen. Towards online adaptation of digital twins. *Open Engineering*, 10(1):776–783, 2020.

[16] Stewart Robinson, Richard E Nance, Ray J Paul, Michael Pidd, and Simon JE Taylor. Simulation model reuse: definitions, benefits and obstacles. *Simulation modelling practice and theory*, 12(7-8):479–494, 2004.

[17] S C Mathewson. Simulation program generators: code and animation on a PC. *Journal of the Operational Research Society*, 36(7):583–589, 1985.

[18] Masami Shimizu and David Van Zoest. Analysis of a factory of the future using an integrated set of software for manufacturing systems modeling. In *Proceedings of the 20th conference on Winter simulation*, pages 671–677, 1988.

[19] Dah-Chuan Gong and Leon F McGinnis. An AGVS Simulation Code Generate for Manufacturing Applications. Technical report, Institute of Electrical and Electronics Engineers (IEEE), 1990.

[20] Young Jun Son, Albert T Jones, and Richard A Wysk. Automatic generation of simulation models from neutral libraries: an example. In *2000 Winter Simulation Conference Proceedings (Cat. No. 00CH37165)*, volume 2, pages 1558–1567. IEEE, 2000.

[21] Ralph Mueller, Christos Alexopoulos, and Leon F McGinnis. Automatic Generation of Simulation Models for Semiconductor Manufacturing. In *Proceedings of the 39th Conference on Winter Simulation: 40 Years! The Best is Yet to Come*, WSC '07, pages 648–657, Piscataway, NJ, USA, 2007. IEEE, IEEE Press.

[22] Mohammed Mesabbah, Waleed Abo-Hamad, and Susan McKeever. A hybrid process mining framework for automated simulation modelling for healthcare. In *2019 Winter Simulation Conference (WSC)*, pages 1094–1102. IEEE, 2019.

[23] Anne Rozinat, Ronny S Mans, Minseok Song, and Wil MP van der Aalst. Discovering simulation models. *Information systems*, 34(3):305–327, 2009.

[24] Wil Van Der Aalst, Arya Adriansyah, Ana Karla Alves De Medeiros, Franco Arcieri, Thomas Baier, Tobias Blickle, Jagadeesh Chandra Bose, Peter Van Den Brand, Ronald Brandtjen, Joos Buijs, et al. Process mining manifesto. In *International Conference on Business Process Management*, pages 169–194. Springer, 2011.

[25] Soeren Bergmann, Niclas Feldkamp, and Steffen Strassburger. Approximation of dispatching rules for manufacturing simulation using data mining methods. In *2015 Winter Simulation Conference (WSC)*, pages 2329–2340. IEEE, 2015.

[26] Ashfaq Farooqui, Kristofer Bengtsson, Petter Falkman, and Martin Fabian. From factory floor to process models: A data

gathering approach to generate, transform, and visualize manufacturing processes. *CIRP Journal of Manufacturing Science and Technology*, 24:6–16, 2019.

[27] Michael Milde and Gunther Reinhart. Automated model development and parametrization of material flow simulations. In *2019 Winter Simulation Conference (WSC)*, pages 2166–2177. IEEE, 2019.

[28] Niels Martin, Benoît Depaire, and An Caris. Using process mining to model interarrival times: investigating the sensitivity of the arpra framework. In *2015 Winter Simulation Conference (WSC)*, pages 868–879. IEEE, 2015.

[29] Peter Denno, Charles Dickerson, and Jennifer Anne Harding. Dynamic production system identification for smart manufacturing systems. *Journal of manufacturing systems*, 48:192–203, 2018.

[30] Diogo R Ferreira and Evgeniy Vasilyev. Using logical decision trees to discover the cause of process delays from event logs. *Computers in Industry*, 70:194–207, 2015.

[31] Niels Martin, Frank Bax, Benoit Depaire, and An Caris. Retrieving resource availability insights from event logs. In *2016 IEEE 20th International Enterprise Distributed Object Computing Conference (EDOC)*, pages 1–10. IEEE, 2016.

[32] Niels Martin, Marijke Swennen, Benoît Depaire, Mieke Jans, An Caris, and Koen Vanhoof. Retrieving batch organisation of work insights from event logs. *Decision Support Systems*, 100:119–128, 2017.

[33] Mahsa Pourbafrani, Sebastiaan J van Zelst, and Wil M P van der Aalst. Supporting automatic system dynamics model generation for simulation in the context of process mining. In *International Conference on Business Information Systems*, pages 249–263. Springer, 2020.

[34] Alexandre Checoli Choueiri, Denise Maria Vecino Sato, Edson Emilio Scalabrin, and Eduardo Alves Portela Santos. An extended model for remaining time prediction in manufacturing systems using process mining. *Journal of Manufacturing Systems*, 56:188–201, jul 2020.

[35] W.M.P. Van der Aalst. *Process Mining - Data Science in Action*. Springer, second edition edition, 2016.

[36] Minseok Song, Christian W Günther, and Wil M P der Aalst. Trace clustering in process mining. In *International conference on business process management*, pages 109–120. Springer, 2008.

[37] Christian W Günther and Wil MP Van Der Aalst. Fuzzy mining–adaptive process simplification based on multi-perspective metrics. In *International conference on business process management*, pages 328–343. Springer, 2007.

[38] RP Jagadeesh Chandra Bose and Wil MP Van der Aalst. Abstractions in process mining: A taxonomy of patterns. In *International Conference on Business Process Management*, pages 159–175. Springer, 2009.

[39] Martin Prodel. *Modélisation automatique et simulation de parcours de soins à partir de bases de données de santé*. PhD thesis, Lyon, 2017.

[40] Bart L Maccarthy and Flavio CF Fernandes. A multi-dimensional classification of production systems for the design and selection of production planning and control systems. *Production Planning & Control*, 11(5):481–496, 2000.

[41] A. Augusto, R. Conforti, A. Armas-Cervantes, M. Dumas, and M. La Rosa. Measuring fitness and precision of automatically discovered process models: A principled and scalable approach. *IEEE Transactions on Knowledge and Data Engineering*, pages 1–1, 2020.

[42] Willibrordus Martinus Pancratius van der Aalst, KM Van Hee, and GJ Houben. Modelling and analysing workflow using a petri-net based approach. In *Proceedings of the second Workshop on Computer-Supported Cooperative Work, Petri nets and related formalisms*, pages 31–50. of, 1994.

[43] Lee Schruben and Enver Yucesan. Transforming Petri nets into event graph models. In *Proceedings of Winter Simulation Conference*, pages 560–565. IEEE, 1994.

[44] Giovanni Lugaresi, Marco Zanotti, Diego Tarasconi, and Andrea Matta. Manufacturing systems mining: Generation of real-time discrete event simulation models. In *2019 IEEE International Conference on Systems, Man and Cybernetics (SMC)*, pages 415–420. IEEE, 2019.

[45] Diego Tarasconi and Marco Zanotti. Process mining for manufacturing systems discovery. *M.Sc. Thesis*, 2018.

[46] Keith Paton. An algorithm for finding a fundamental set of cycles of a graph. *Communications of the ACM*, 12(9):514–518, 1969.

[47] Rob J Wang and Peter W Glynn. On the marginal standard error rule and the testing of initial transient deletion methods. *ACM Transactions on Modeling and Computer Simulation (TOMACS)*, 27(1):1–30, 2016.

[48] Enver Yucesan and Lee Schruben. Structural and behavioral equivalence of simulation models. *ACM Transactions on Modeling and Computer Simulation (TOMACS)*, 2(1):82–103, 1992.