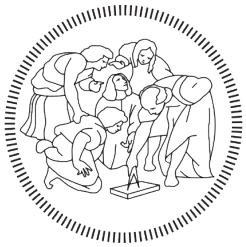


POLITECNICO
MILANO 1863

School of Industrial Engineering
Department of Mechanical Engineering

Automated Generation and Exploitation of Discrete Event Simulation Models for Decision Making in Manufacturing

Advisor: **Prof. Andrea MATTA**

Candidate: **Giovanni LUGARESI**
Student number: **891835**

A master in the art of living draws no sharp distinction between his work and his play; his labor and his leisure; his mind and his body; his education and his recreation. He hardly knows which is which. He simply pursues his vision of excellence through whatever he is doing, and leaves others to determine whether he is working or playing. To himself, he always appears to be doing both.

[L. P. Jacks, Education through Recreation, 1932]

Acknowledgments

I am grateful towards all the people who made this work possible. It is a pleasure to acknowledge their support, efforts, and contributions.

Above all, I thank professor Andrea Matta for the guidance along all the duration of my PhD, the patience and perseverance in stimulating my independent thinking, as well as the respect of my aspirations. I am thankful to all the *Sim*.LAB team, with whom I shared both work and fun moments. I am thankful to professor Tullio Tolio, who kept inspiring me to love this discipline. Also, we had the privilege to share the first fully-online teaching experience during the Covid-19 lockdown.

I thank all the students whom I had or have the honor to supervise. Part of their work has been integrated and extended in this thesis: Vincenzo Alba, Gianluca Aglio, Federico Folgheraiter, Davide Travaglini, Diego Tarasconi, Marco Zanotti, Luca Spada, Jacopo Barbieri, Sofia Gangemi, Giulia Gazzoni, Edoardo Passarin, Francesco Verucchi.

I am grateful for a number of colleagues and friends that sponsored me through the highs and lows of this experience, encouraging me to persevere with it and sharing joyful moments along the way.

I am indebted to my girlfriend Vivian for her endless patience and support. Last but not least, I am grateful for my family, that has been always besides me in every moment. A particular thought goes to my grandparents Angelo and Franca, who consistently reminded me what is really important in life.

Contents

List of Figures

List of Tables

Abstract

The latest developments in industry have involved the deployment of digital twins for both long and short term decision-making, such as supply chain management and production planning and control. The ability to take appropriate decisions online is strongly based on the assumption that digital models are properly aligned with the real system at any time. As modern production environments are frequently subject to disruptions and modifications, the development of digital twins of manufacturing systems cannot rely solely on manual efforts. Industry 4.0 has contributed to the rise of new technologies for data acquisition, storing and communication, allowing for the knowledge of the shop floor status at anytime. If a model could be generated from the available data in a manufacturing system, the development phase may be significantly shortened. However, practical implementations of automated model generation approaches remain scarce. It is also true that automatically built representations may be excessively accurate and describe activities that are not significant for estimating the system performance. Hence, the generation of models with an appropriate level of detail can avoid useless efforts and long computation times, while allowing for easier understanding and re-usability.

This research focuses on the development and adoption of automated model generation techniques for obtaining simulation-based digital models starting from the data logs of manufacturing systems, together with methods to adjust the models toward a desired level of detail. The properties and parameters of the manufacturing system, such as buffer sizes, are estimated from data through inference algorithms. The system properties are also used in a model tuning approach, which generates an adjusted model starting from the available knowledge and the user requirements in terms of complexity (e.g., number of stations). In addition, a lab-scale environment has been built with the aim to test decision-making frameworks based on digital twins within a realistic data infrastructure. The experimental results prove the effectiveness of the proposed methodology in generating proper digital models that can correctly estimate the performance of a manufacturing system. The model generation and tuning method can positively contribute to real-time simulation. Indeed, its

application within an online framework of production planning and control allows for adapting simulation models to the real system, potentially at any time a modification occurs. This way, decisions taken online are guaranteed to be referring to the current state of the factory. Thanks to this research, manufacturing enterprises will be able to reach a higher production flexibility, together with higher responsiveness to technological changes and market-demand fluctuations.

Preface

In the last decades, manufacturing environments have become significantly more complex in the attempt to satisfy a higher demand for customized products, while aggressive digitization efforts have pushed production enterprises to invest in new technologies toward higher levels of automation [3]. In modern organizations, information systems play a substantial role in the support of day-to-day operations. Meanwhile, manufacturing systems are affected by several types of disruptions, such as machine failures, quality losses, lack of materials. Hence, Production Planning and Control (PPC) approaches must take real-time actions on production and quality management of manufacturing processes. For instance, deciding to maintain a station and route parts to alternative process paths. The reaction time needs to be limited and controllable. Further, the decision-making criteria have to be always compliant with the production system requirements that may evolve during its life cycle.

The PPC tools currently used in the manufacturing industry face significant limitations. Specifically, they are slow in reacting to disruptions[1] and they are not able to evolve as fast as the real system. Traditional PPC tools typically rely on digital models of the shop floor which have been developed at the design phase, often relying on fixed thresholds or alarms; further, they do not learn nor build knowledge from data collected on the field. As a result, such models often represent old situations and might lead to wrong decisions in production. Hence, traditional tools tend to be inadequate to manage rapidly changing shop floor environments.

The time to create and/or update digital models is considerable, often ranging from a few days to even some weeks. This criticality heavily affects the industry: most shop floors in the manufacturing industry are managed with simple conservative rules implemented in the Manufacturing Execution System (MES), and with almost no use of system-level digital models. In summary, new PPC methods suited to the industrial scenario still have to be designed and MES tools must be extended to allow for the efficient online management of modern shop floors.

[1] Common reaction times are largely greater than 15 minutes.

The Industry 4.0 phenomenon has provided a set of new technologies for production environments such as Internet of Things (IoT), cloud computing, big data analytics, augmented and virtual reality, Radio Frequency Identification (RFID), artificial intelligence, and machine learning [4]. Several innovative solutions have been developed to assist production environments, such as cyber-physical systems [5] and virtual factories [6]. The future industrial scenario will be defined by how these new technologies will be exploited to shape radical changes in the methodologies used to plan and control manufacturing systems.

Modern decision-support tools are based on the coexistence of the real system with its digital counterpart, often called digital twin [7]. Digital twins have been considered as key components for the success of the Industry 4.0 initiative [8]. In general, virtual representations of physical resources can store information such as kinematic data, interfaces, production events, and performance indicators [9]. In production environments, digital twins have been exploited in processes such as assembly, scheduling, machining and logistics, with the main goal to reach a higher production efficiency [10]. With reference to production planning and control phases, digital twins can be represented by discrete-event simulation models. The addition of real-time streams of data can help production planners to evaluate solutions that are optimal for the current system state at any time [11].

The implementation of digital twins is claimed to make production processes more flexible, adaptable, and predictable [12]. Indeed, thanks to the aforementioned developments in industry and research, it is possible to imagine a situation in which the shop floor status in manufacturing companies can be retrieved anytime. This paves the way for new opportunities:

- **Tailored decision-making.** Production policies defined a-priori may be optimal within a precise set of boundaries, but often turn out to be detrimental on the system performance when applied in actual circumstances. A possible countermeasure is to design solutions which are robust against many different scenarios, although they are never truly optimal for any particular condition rather for the average performance of the system. Distinctively, online decisions can be tailored exactly considering the system status in the very moment they are examined.

- **Reduced reaction times.** The capability to exploit real-time streams of data can enhance decision-making and reaction time. The consequent increased integration of functionalities allows for a closer coupling of previously separated decisional levels. This brings benefits to production systems such as the ability to promptly respond to unexpected

events [13]. Indeed, quicker data exchange capabilities enable a tighter decisional loop, which can be based on current streams of data from the shop floor, resulting in better online decisions [5].

- **Online and offline learning.** By performing data collection from digital instances of production systems, scenarios never happened in the practice can be explored and inspected before implementing the intended actions [5]. This activity can be done both while the system is working and offline.

Thanks to the key enabling technologies of Industry 4.0, manufacturing companies will be able to build structured information from field data. Shop floors will be able to improve efficiency and resilience thanks to an intensive use of data and models shared within a digital network. In this emerging context, current MES need to be extended with advanced PPC tools for the online management of shop floors. Currently, there are no theoretical approaches in support of PPC that provide data-driven, self-adaptable, and real-time capabilities. This thesis contributes to the development of methods and approaches to favor the introduction of digitally-assisted decision-making in manufacturing systems.

Chapter 1

Introduction: State of the Art on Real-Time Simulation

Within a production planning and control scope, discrete-event simulation models constitute one of the main tools which can represent digital counterparts of manufacturing systems and may be used for both long-term planning and short-term decision-making. Real-Time[1] Simulation (RTS) is a concept that involves using simulation to take accurate decisions based on the current system state [14]. This chapter interprets the main aspects of RTS applications with the aim to identify which problems have been addressed in the literature and which still remain an issue.

The chapter is organized as follows. Section 1.1 introduces the concept of Real-Time Simulation for manufacturing systems. Section 1.2 presents selected RTS frameworks from the literature. Section 1.3 collects significant contributions exploring the role of RTS in production environments. The thesis contribution and structure are presented in section 1.4.

1.1 Real-Time Simulation

In the manufacturing field, Real-Time Simulation (RTS) has been defined as *"a computerized system capable of performing both deterministic and stochastic simulation in real-time or quasi real-time, to monitor, control, and schedule parts and resources in a discrete part manufacturing environment"* [15]. In Real-Time Simulation applications, data are acquired from a production plant and feed a simulation model of the system. Then, alternative scenarios are defined and simulated: the one that leads to the best performance – if feasible

[1] Real-time in production has to be intended in relation to product turnarounds, which can be minutes or even hours depending on the product and the planning horizon.

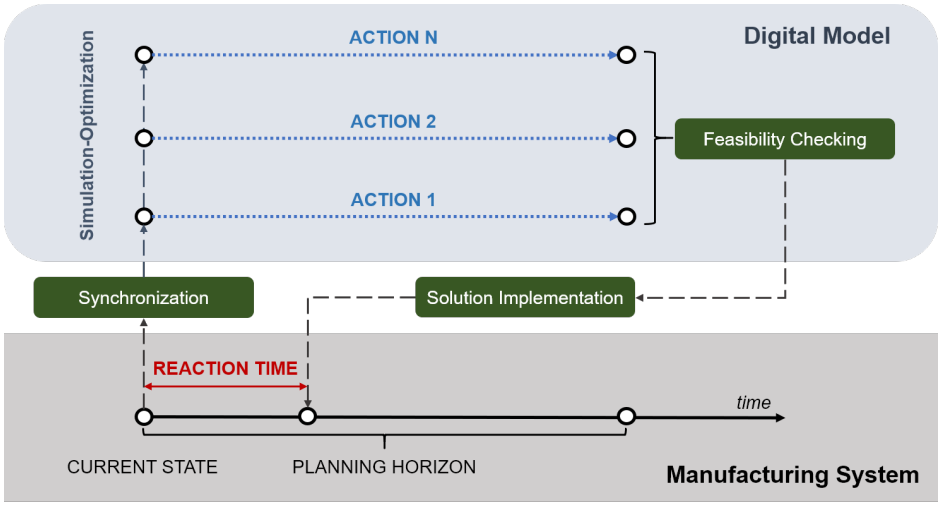


Figure 1.1: Temporal representation of a Real-Time Simulation procedure.

– can be applied online. Figure 1.1 shows a general procedure that can be followed in RTS approaches. The current state of a manufacturing system is represented in digital form via a synchronization component. Several digital instances may be generated, depending on the number of alternatives that have to be considered. The digital models are then used to simulate the behavior of the system. Each alternative action is explored, investigating the resulting performance at the end of a time horizon of interest. Finally, the action that produced the best performance is checked to verify if its application is still feasible. If the last test is successful, the solution is implemented in the system. The reaction time is determined by the duration of all the digital activities. It is mostly influenced by the complexity of the simulation-optimization problem.

1.2 Frameworks

Manivannan and Banks [16] proposed one of the first comprehensive analyses on RTS, and introduced a framework for real-time control of a manufacturing cell using simulation. More recently, other schemes have been proposed. Mirdamadi et al. [17] detailed the functionalities of a simulator for the real-time production control and described a procedure for the monitoring and execution of production in which simulation is used to determine the best control alternative. Rao et al. [3] described a novel Real-Time Simulation system for real-time shop floor control. This framework relies on the relationship between the simulator and the Manufacturing Execution System (MES). It is also shown how the software infrastructure of a MES may incorporate RTS functionalities.

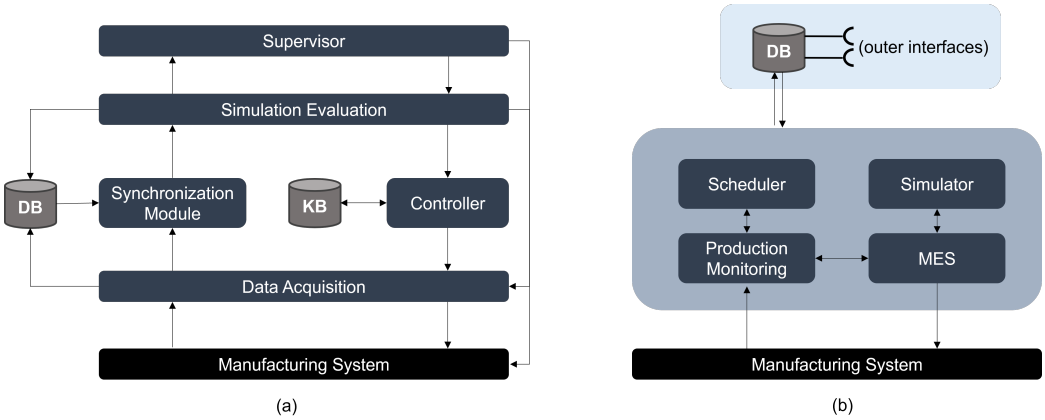


Figure 1.2: RTS architectures. (a) Manivannan and Banks (1991); (b) Monostori et al. (2007).

Indeed, the system can collect data from the physical shop floor and communicate with a scheduling controller through the MES.

Further contributions about RTS architectures may be found in [18], [19], and [20]. All the frameworks are based on real-time input data acquired while the real system is evolving, and decisions must be made as soon as the problems are identified in the system. Therefore, the simulation-optimization loop is required to have fast-answer capabilities. For example, for the evaluation of the best control policies, every alternative policy that is generated for comparison has a corresponding simulation model for estimating its performances. Each of these models has to provide a sufficient statistical warrant that the performance estimates are valid, therefore dedicating enough replications to its experiments. Most of the authors in the literature refer to computational time and, in general, fast answering capabilities, as one of the main issues to be solved for making RTS possible in the practice.

Figure 1.2 shows two architectures for RTS that are worth considering and describing in detail. Let us assume that a manufacturing system with a discrete process exists in a specific industrial context and has to be controlled according to a previously computed production plan. The first architecture (Figure 1.2a) is taken from Manivannan and Banks [16]; the main components of the platform are the following:

- *Data Acquisition* hardware is made by sensors and devices to acquire data from the field. Each manufacturing equipment that is represented in the simulation model must have its sensory hardware correspondent.
- The *Synchronization Module* is responsible for the association of data collected from the field to events in the simulated environment.

- *Static and Dynamic Databases* (DB) contain machine and tool data, product data, maintenance data, and performance measures; they consist in the interface where information received from the sensors is connected to the virtual components representing the machine tool, either continuously or periodically.
- The *Knowledge-base (KB)* is a database where the results obtained from previous simulation runs can be stored and retrieved to reduce the number of simulations. The authors suggest a structure of knowledge that allows for inheritance between similar components used in the system. Thus, when an item is added to the system, the level of knowledge about that resource is already available. Moreover, solutions already encompassed by the control system are stored and lay the foundations for the next decisions. Therefore, not all the events occurring in the system require a simulation campaign. For example, some critical events may have been already processed in the past, and if the solutions applied in those cases are available and implementable there is no need to launch new evaluations.
- *Simulation Models*: for the specific use-case, the authors refer to models characterizing the tool-wearing process of machining centers. The simulation runs are triggered by a model supervisor.
- A *Controller*. This module interacts with the KB and retrieves the necessary input data to perform simulation using the model created. The cell controller updates the KB using the simulation outputs for future use.

The second framework structure is shown in Figure 1.2b and is taken from Monostori et al. [21]. Here, the data from the plant feed a general decision-making unit, composed by a production supervisor, an MES, a scheduler, and a simulator which is based on the database of the system status. The MES is interfaced with a factory simulator that evaluates the solutions before they shall be implemented. By comparing the two architectures, we may state the following: the first architecture considers explicitly the data collection and data input from databases; the second architecture centralizes the databases of the RTS loop and foresees the need of connection to other devices/ software components. With this perspective, we may state that the framework of Figure 1.2b is more compliant with Industry 4.0. We may consider the controller of the first framework as equivalent of an MES. Differently, in the second framework the MES directly controls a simulator with no intermediate connections. As a result, the simulator of the second architecture can be seen from two point of views: (1) as a visualization of the system current performance; (2) as a tool for evaluating several scenarios.

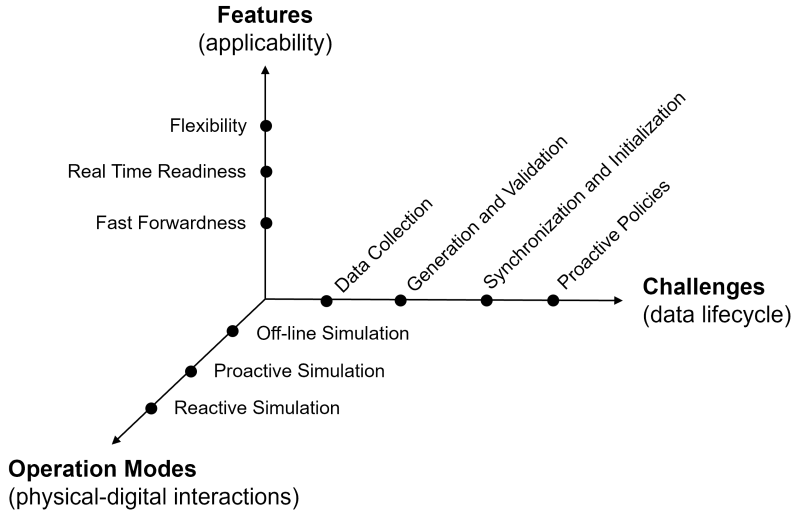


Figure 1.3: Classification axes for the literature review on Real-Time Simulation approaches.

1.3 Related Literature

In this section, we aim to revise the existing literature on Real-Time Simulation approaches. For such a task, three main classification axes have been identified, as described in Figure 1.3. The first classification method is derived from the work of Manivannan and Banks [15], and it is centered on the perspective of the data lifecycle. The second classification focuses on the interactions between the physical and the digital instances. The third axis is based on the applicability of the RTS framework.

Manivannan and Banks [15] have identified the **challenges** to overcome for the successful implementation of RTS models: (1) data handling, (2) model generation and validation, (3) model synchronization and initialization, and (4) efficient proactive scheduling models. Monostori et al. [5] described the **operation modes** for DES models in production system, dividing them in three categories: (1) offline simulation, used for sensitivity analysis and robustness evaluation of production schedules before their execution; (2) proactive simulation, used with the aim of defining online short term actions in response to expected deviations from the production plan; (3) reactive simulation, used for an online evaluation of alternatives after a disturbance has occurred. Mousavi and Siervo [22] proposed a framework with the aim to provide three main **features**: (1) flexibility: the framework has to be general in terms of input data and utility function model (for example, the sample frequency can be determined dynamically); (2) real-time readiness: the framework aims to make an effective

use of data available at timenow for simulation modeling; (3) fast-forwardness: it is always possible to feed the simulation model with different types of inputs to perform *what-if* analyses; historical data may also be used to gradually improve the response time of the system. The rest of this chapter elaborates on the significant contributions from the literature, which have been organized in Table 1.1 according to the three aforementioned classification criteria. The following sections adhere to the classification over the RTS challenges.

1.3.1 Data Handling

Data collection is one of the most time consuming practices at the beginning of a simulation project, and the optimization of this phase regards both static and real-time approaches. Automated data acquisition and communication paves the way for the exploitation of simulation for short-term decision-making [8]. Beside the dataset size issues[2], there is the need for an intelligent data-handling unit that would correctly select different subsets of data depending on the decisions to be made at the moment. In fact, data collection and the related data transfer communication standards (e.g., IEC 61499) are a central topic in Industry 4.0 related research [24, 25]. Robertson and Perera [26] proposed to automatically collect input data for a simulation model to save modeling time. The authors discussed on how to revisit the data-input methodologies for simulation and proposed to group the data-input phases exploiting an intermediate database between an Enterprise Resource Planning (ERP) system and the simulation input data. Hanisch et al. [27] contemplated two characteristics for data in real-time digital models: availability and quality. Availability means that all the necessary inputs are guaranteed either by automatic collection from the system or by successive elaboration from the existing datasets. Quality regards the degree of exactness of data. Mousavi and Siervo [22] proposed a flexible data input management system as solution for quick-response decision-making. The aim is to generalize the input procedure and make it applicable to a wide variety of manufacturing environments. Similarly, Blum and Schuh [28] defined a three-layer architecture for data analytics in real-time frameworks, composed by (1) a data layer, (2) an integration layer, and (3) a visualization layer. Since online decisions must be made as soon as the problems are identified in the system, RTS frameworks are based on real-time input data acquired while the real system is evolving. Kitazawa et al. [29] used RTS to estimate the completion time of a flow shop manual assembly. The data is collected by Bluetooth-based beacons to record

[2]For example, a typical production system can produce 152,000 data samples per second [23].

the proximity of operators to their workstation. The beacon position is used to infer the operator current working state. Altaf et al. [30] showed the integration of simulation with RFID collected data to infer the status of a wood-frame panel prefabrication plant. Similarly, Luo, Fang, and Huang [31] introduced an approach for real-time scheduling using RFID technologies to facilitate shop-floor-conditions visibility, by equipping jobs with RFID tags recording relevant information such as the job identifier, the current production stage, the arriving and processing time of the stage, and the job's due date.

1.3.2 Model Generation

For a successful RTS implementation, the digital counterparts of production systems have to reflect the current configuration of the system at any moment in time [32]. In flexible and reconfigurable manufacturing systems these updates can be done in an automated way. Yet, most contributions of RTS approaches in the literature assume the availability of simulation models, often static or generated through configuration files. However, these methods have clear limitations in keeping the pace with the frequent changes of manufacturing systems. Indeed, simulation models are very time-demanding in their building phase, especially for the data-collection activities. The high cost and time required for data collection often result in useless simulation models, because not promptly aligned with the system changes [33]. Therefore, the existing frameworks for online decision-making encounter their limits when used with high input frequencies and changeable system configurations.

According to Mathewson [34], 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 modeler's behavior*". Model generation deals with the recognition of the logical relationships between components of the manufacturing system (e.g., part routes, precedences, spacial contraints) [35, 33]. For instance, if an automated model generation procedure is established along the life cycle of the production plant, a plugin of a new machining tool or the deterioration of a machine can be detected online and the simulation model can be updated coherently [36, 37]. Further, the simulation model should have the ability to change their logic by observing the data from the manufacturing systems, for instance following a machine failure. This problem corresponds to finding the simulation model that best fits the data collected from the system, and to have the system simulation model always aligned with the real system.

1.3.3 Model Validation

Validation is defined as the *process of determining whether a simulation model is an accurate representation of the system, for the particular objectives of study* [38]. It compares the input-output transformation of the simulation model with the one of the real system, by running the model using the same input conditions derived from the real system and analyzing its behavior in terms of output measures [39].

Planning and control activities using RTS typically rely on the assumption that a simulation model for the real manufacturing system exists and has withstood the validation process [40]. Hence, if the model is generated online, validation has to be performed with respect of the data representing the current state of the system. In the literature, Autovalidation is intended as the practice through which the simulation-model ability to predict the system performances is checked online before using the model to take operative decisions; Model Structure Update deals with the validation of an already available model, that is compared with both the system structure and the process logical layout [41].

Several validation methods have been developed over the years for the off-line validation of a DES model. These methods are mostly based on statistical techniques based on the comparison between output data from the DES model and the real system [42]. A first-cut evaluation of the model performance is to compute the differences between observed and predicted responses by means of statistical indicators. Some are used very often in the practice (i.e. the mean, the median, the mean squared error) whereas others are less popular although they can be used effectively: for instance, the Coefficient of Determination and the Coefficient of Residual Mass [43], the Modeling Efficiency [44]. Usually, a simulation model is then validated by means of statistical tests, such as *t-test*- and *F-test*-based statistics [45], two-sample Kolmogorov-Smirnov test [46], regression-based tests [47], and trace-driven methods [41].

The list of validation techniques is very long [48]. However, all of them have been developed for an off-line use. Hence, they are limited when applied to RTS conditions due to their need of a large amount of data to reach a statistical significance of their results. In a real-time scenario, the digital model will be subject to very frequent changes, and simulation runs may never reach the steady state. As a consequence, all the performance measures may have to be computed in the warm-up phase. Validation techniques have to be adapted taking into considerations this volatility of the simulators. Moreover, while the emphasis of traditional validation techniques is put on a set of limited performance indicators (e.g., number of pieces produced in a time window), the new techniques may have to consider the validation of functions (e.g., uti-

lization profile of a machine) to synchronize flows at critical resources. Recent approaches have proposed to use signal processing theory for the online validation of digital models [49].

1.3.4 Model Synchronization and Initialization

The exploitation of a digital model of a factory, beside a continuous information exchange with the real system, allows to make realistic simulation-based predictions referred to the current system status [50, 51]. In general, the synchronization between the system and its digital alter can be carried out in three main ways [52]: (1) by continuously collecting data and connecting the data acquisition devices to an input data processor within the simulation software; (2) by developing a simulation model for each of the parts and resources, and restricting data collection activities to those altering temporal information; (3) by making use of past, future and current event lists. Kadar et al. [53] identified the following issues for synchronization: (1) the acquisition and validation of the input data, (2) the responsiveness of the analysis, and (3) the capability of quickly gathering the real system state to initialize the simulation model. Talkhestani et al. [54] proposed to obtain model integration in a product lifecycle management platform using an anchor-point-based method to systematically detect variations in the data structure between the digital models and the physical system.

Initialization consists in a guarantee that alternative simulation models refer to the same initial point in time. The goal is to assure that alternative production policies can be effectively compared. Namely, in order for the performance statistics of the alternative policies to be comparable to the ones of the real system, the corresponding models must be initialized to the current real system state, or at least to the same state which occurred in a certain time-frame. Therefore, RTS models start from a state in which all the variables of the model are set to the values of the physical quantities at timenow [27]. Bergmann, Stelzer, and Strassburger [55] proposed an initialization solution based on the Core Manufacturing Simulation Data (CMSD) standard, and recommended proper extensions to guarantee that data can be transferred into the digital model in a standardized way.

1.3.5 Proactive Policies

Typically, the Manufacturing Execution System (MES) includes short-term decision-making functions such as re-scheduling and dispatching algorithms [56]. These tasks can be accomplished online exploiting information coming from both

the shop-floor and any other management software. Recently, several frameworks exploiting RTS to improve online production policies have been proposed [57, 58, 59, 60]. A production re-planning strategy describes when a new production plan for a certain manufacturing system has to be generated. This can happen in two main forms [61, 62]: (1) the plan can be adapted periodically based on the present production trend; (2) the re-planning is triggered by specific events that can have an impact on the system performances, such as machine failures, urgent orders, quality issues [63]. Cardin and Castagna [32, 64] explored the decisional component of a job-shop with six workstations connected with transporters. The authors exploited a *base model* aligned with the real system and *variant models* for proactive decision-making such as the routing of parts on transporters. These decisions depend on many factors, such as the number of available transporters in the system or the expected machine breakdowns. The multitude of decision possibilities is reflected in the number of variant simulation models that are initialized to the system state and used to run experiments for a time horizon of interest. Harmonosky et al. [65] developed an heuristic approach to manage the queue of unfinished jobs at a failed machine in a flow-shop system. The main idea is to compare the expected waiting time between the failed machine and an alternative machine including a penalty term for accounting rerouting time. Simulations are done offline before any actual system breakdown occurs. With reference to manufacturing systems characterized by stochastic processing times, Framinan et al. [66] suggested that if real-time data measured on the shop-floor were exploited as a rescheduling contour condition, it would be possible to lower the expected makespan. This advantage is greater if the variability in system parameters is fairly low, while a high variability translates into high uncertainty of the results and may hinder the improvement. Mirdamadi et al. [17] described a procedure for exploiting simulation to determine the best production-control alternatives: events in the real system are clustered and tagged to identify the necessity and to assign priority to the interventions.

1.4 Thesis Contribution

The following paragraphs outline the main contents of this thesis, which are summarized in Figure 1.4.

1.4.1 Lab-Scale Models for Testing Real-Time Simulation

The literature on Real-Time Simulation approaches for manufacturing systems is rich of contributions [4]. Meanwhile, it is hard for researchers and practition-

Reference		Data Handling	Generation and Validation	Initialization and Synchronization	Proactive Policies	Offline	Proactive	Reactive	Flexibility	Real-time Readiness	Fast-forwardness
Altaf et al.	[30]	•					•			•	
Aydt et al.	[67]				•		•		•		
Bergmann, et al.	[55]			•			•	•		•	
Biesinger et al.	[35]		•			•			•		
Blum and Schuh	[28]	•					•			•	
Bohlmann et al.	[63]				•			•			•
Cardin and Castagna	[64]				•		•			•	
Cardin and Castagna	[32]			•	•		•			•	
Damiani et al.	[57]				•	•			•		
Framinan et al.	[66]	•			•		•			•	
Fujihara and Yoneda	[37]		•	•		•			•		
Hanisch et al.	[27]	•	•	•			•	•			•
Harmonosky et al.	[65]				•	•			•	•	
Kadar et al.	[53]		•	•	•			•		•	
Katz and Manivannan	[50]			•				•		•	
Khan et al.	[41]		•			•			•		
Kitazawa et al.	[29]	•			•	•					•
Low et al.	[68]	•						•			•
Lugaresi et al.	[49]		•				•			•	
Luo et al.	[31]	•			•			•	•	•	
Manivannan and Banks	[15]	•		•	•	•				•	
Martinez et al.	[51]		•	•			•			•	
Martinez et al.	[36]		•			•	•	•		•	
Mirdamadi et al.	[17]	•	•	•	•			•	•		
Mousavi and Siervo	[22]	•					•				•
Nasiri et al.	[58]	•						•			•
Pfeiffer et al.	[59]				•			•		•	
Rao et al.	[3]	•						•			•
Robertson and Perera	[26]	•				•			•		
Son and Wysk	[33]	•	•			•			•		
Suresh et al.	[60]				•		•			•	
Talkhestani et al.	[54]			•		•				•	
		Challenges				**Operation Modes**			**Features**		

Table 1.1: RTS literature items classified according to the three criteria (Figure 1.3): challenges, operation modes, features.

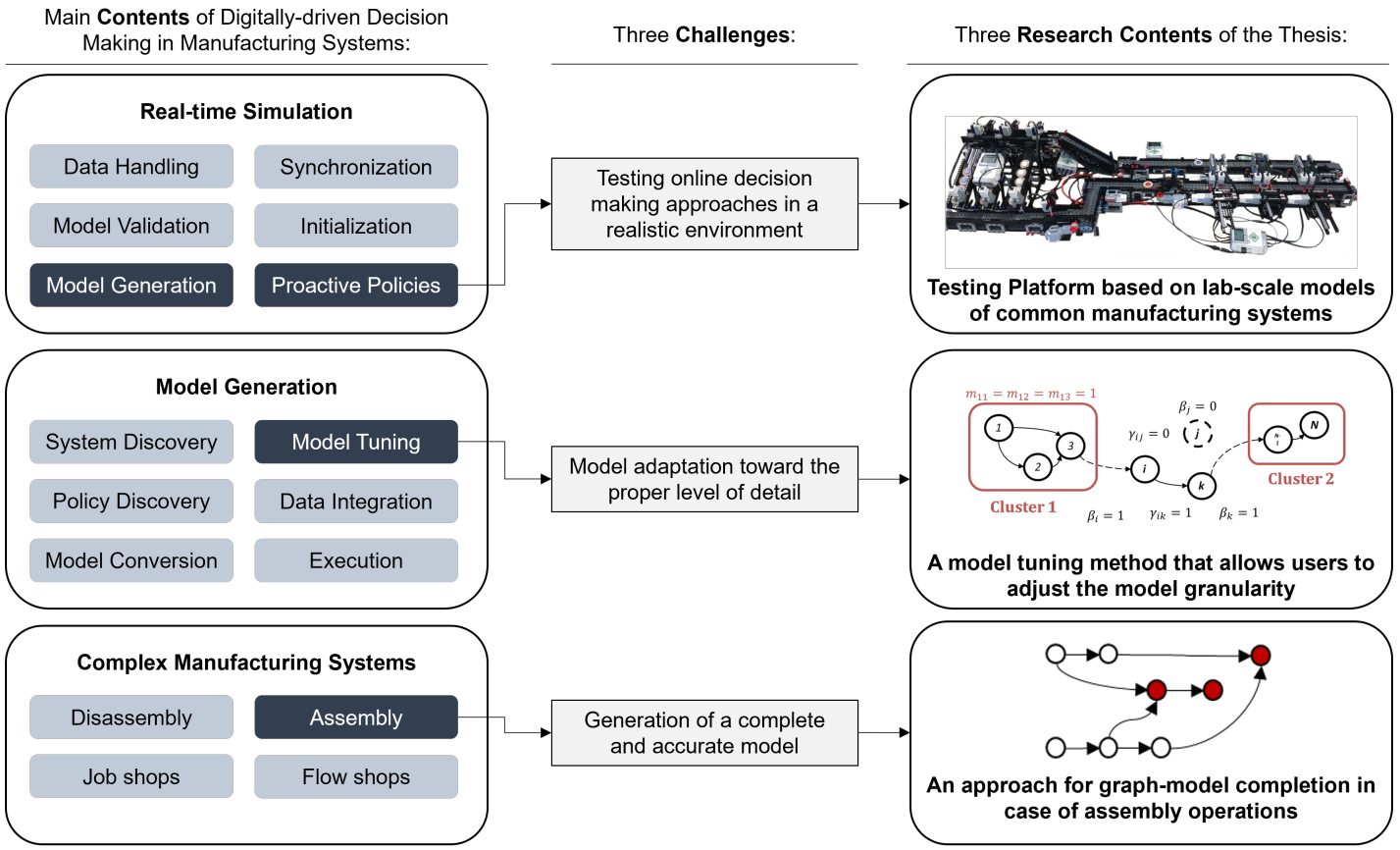


Figure 1.4: Graphical map of the thesis contents.

ers to provide a realistic environment to test their architectures and algorithms. Typically, the proposed methods would either be tested on a real system or within a digital setting. In the former case, it is necessary to allocate a consistent portion of a production facility and to change the already established strategies for a considerable time. The risk is to invest a lot of resources in arranging the system to reproduce the desired behavior, rather than iterating the proposed logic. On the other hand, the theorized algorithms and policies could be compared and validated exploiting digital models of the production systems. In this case, several issues related to the physical system might be underrated. For instance, data-collection devices must be designed properly to retrieve the desired information (e.g., current number of parts in a queue, machine status) and to guarantee the alignment between the real system and its digital counterpart. As a result, literature is rich of theoretical contributions while lacking of practical implementations of RTS.

In response to such challenges, the adoption of lab-scale models enables to examine approaches that involve information loops through real industrial components (e.g., gateways, sensors), which is not possible if validation is performed only on digital models. This way, hardware-related issues such as data sharing between the various layers of a software architecture can be investigated, while remaining capable to implement and iterate the proposed PPC logic in reasonable times. Further, the realization of physical models with components such as LEGO requires lower investments and provides higher flexibility for tests compared to similar settings in real systems.

Chapter 3 describes the development of physical models of manufacturing systems in a laboratory environment, together with their exploitation for testing real-time decision-making approaches. Figure 1.5 shows an example of a manufacturing system model developed with LEGO MINDSTORMS components. Further, an approach for online production re-scheduling is adapted for the implementation and to address real-time problems in critical operations, where time constraints are strict. The approach is data-driven, and focuses on delivering prompt actions anticipating or reacting to events in the shop floor (e.g., machine failures and tool degradations).

1.4.2 Automated Generation of Simulation Models

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

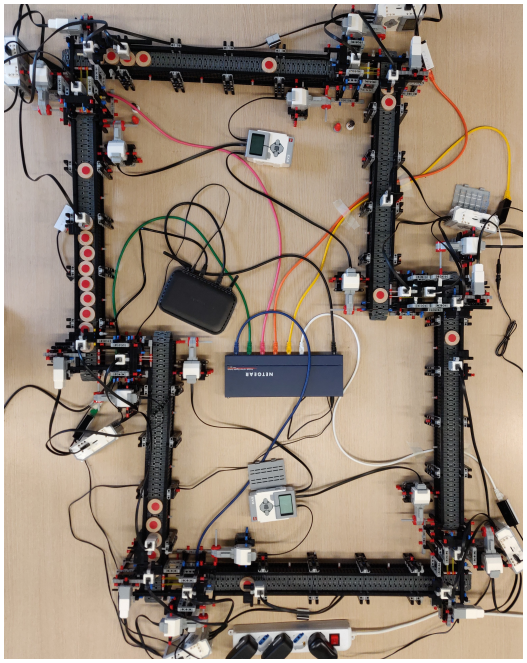


Figure 1.5: Example of LEGO-based physical model.

increasing demand for customized products highlights the necessary ability to promptly adapt processes and resources [69].

The ability to take appropriate decisions online 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 [70]. Therefore, the digital models supporting the decision-making task must be able to follow this evolution. A manual adaptation of digital models cannot be achieved in practice, because it would require large efforts and time (sometimes days or even weeks), together with a high-competency workforce spent on repetitive, non value-adding tasks. Further, the time to develop a new model may hinder the exploitation of a digital twin along the life cycle of a production system [71]. Hence, digital models should be self-adaptable and driven by sensor collected data. This problem corresponds to automatically building and adapting digital models that fit the data collected from the system, detecting its modifications and deviations, and being able to reliably predict the production behavior of the shop floor.

Within the Industry 4.0 context, 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 [72]. 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 [73]. Model adaptations are typically handled through configuration files, for instance, to select the number of parallel machines that are dedicated to a certain part type. Such approaches are not effective against modern manufacturing systems changes, in which also the system topology may change during production. More recent approaches have introduced the exploitation of process mining techniques [74], which can retrieve the system topology from the manufacturing system data [75].

Despite the aforementioned improvements, practical implementations of automated model generation remain scarce; one of the reasons for this is the difficulty in adapting the level of detail of the model [76]. As a solution, a model tuning step can remove complexities that may hinder both the understandability and the re-usability of digital models. **Chapter 4** focuses on obtaining digital models starting from the data logs of manufacturing systems and the development of an approach to tune them toward the desired level of detail.

1.4.3 Model Generation for Non-Linear Material Flows

The automated development of digital models for more complex manufacturing systems reaches the limitations of the available methodologies. Indeed, most approaches based on process mining are limited by the assumption of so-called *flat data*. Namely, a unique part ID is used to identify material flows, while in realistic context several different object types may be kept together within certain production phases (e.g., batches, packages, orders). The relationships among different objects are disregarded by the available methodologies. The result is that certain types of systems cannot be modeled entirely with an automated approach. For instance, assembly processes are very common in the production enterprises (e.g., automotive industry). In such environments, several material flows converge in assembly stations. During or after the assembly of parts, it is common that new part identifiers are assigned to the just-introduced assembled part. Traditional process mining techniques assume a single part identifier to derive the logical flows (e.g., activity precedences). As a consequence, the production system is discovered as a collection of separate models, while the assembly logic is effectively lost.

Chapter 5 explores the applicability of the developed model generation approach with respect to complex manufacturing systems such as factories with assembly operations. In this chapter, an extension to the model generation procedure from Chapter 4 is proposed to handle data logs with multiple ob-

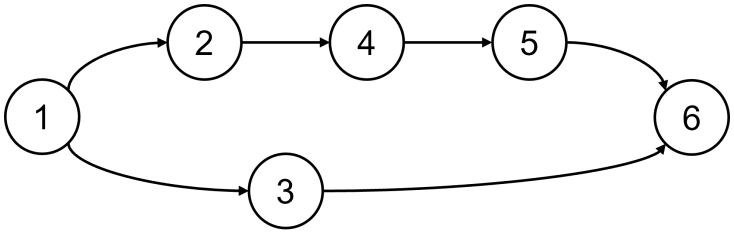


Figure 1.6: Graphical representation of the thesis chapters and suggested reading precedences. Each node represents a chapter, while the arcs are indicative of the reading precedence.

jects. This way, the digital models can include more complex production logic such as the requirement of different components to proceed coordinated with assembly operations.

1.4.4 Thesis Structure

The rest of the manuscript is organized as follows. Chapter 2 deepens the discussion on the state of the art of approaches based on Process Mining. Chapter 3 introduces the laboratory environment that has been developed to test Real-Time Simulation approaches. Chapter 4 outlines the model generation and tuning method that has been developed, while Chapter 5 extends the developed approach with a method suited for complex manufacturing processes. Finally, conclusions and considerations over future developments are collected in Chapter 6. Figure 1.6 presents a chapter precedence graph as proposal for reading the thesis.

Chapter 2

Process Mining and Model Generation in Manufacturing: State of the Art

In this chapter, the principal contributions from the literature regarding applications of process mining (PM) techniques in manufacturing are discussed. PM is a discipline aiming to discover and exploit valuable information from event logs available in information systems [1]. Since PM is agnostic, it has been applied with a plethora of datasets for several scopes. We may classify the applications depending on the scope medium: (1) material flows, hence papers using PM to retrieve the movement of physical objects, or in general information which can be linked to the work-pieces (e.g., production levels); (2) information flows, namely contributions exploiting PM to analyze all the business processes that surround the production environments. The following sections include a selection of the contributions, with particular focus to model generation.

The chapter is organized as follows: Section 2.1 presents the approaches for mining material flows; section 2.2 summarizes the contributions dedicated to the mining of information-based processes; finally, the limitations of the available approaches are explained in section 2.3.

2.1 Material-Based Mining

The simplest yet effective way to use PM analyzing material flows is to visualize the flows with a time perspective. Indeed, the transformation of data from event logs into informative visualizations is offered by most PM tools. Data can be analyzed from various perspectives, such as part identifiers, activities,

and resource levels. More aggregated indicators can be easily retrieved (e.g., flow times, waiting times) and visualized over graph models, allowing for the easy detection of bottlenecks and critical points [77]. The most common applications of PM are (1) performance evaluation, (2) process monitoring, and (3) model generation.

2.1.1 Performance Evaluation

Performance evaluation is one of the most common application of PM in manufacturing. Given that most times the structured event logs contain temporal information (i.e., timestamps of the activities), several contributions exploit it to relate temporal performance measures such as lead time and completion times [78]. Intayoad and Becker [79] analyze event-logs from manufacturing companies and use PM to extract contextual information of processed orders namely (1) the number of process activities competing for the same resources, and (2) the lead-time of the previous completed order if it was delayed. PM can also support the development of a probabilistic model (e.g., Bayesian Networks) and predictive models. The integration of these models makes it possible to calculate the occurrence probability of all activities and future behavior of the process in terms of completion time [80]. Park, Lee, and Zhu [81] developed a method for the joint data extraction, clustering, and performance evaluation for a shipbuilding manufacturing system. The approach exploits process mining to extract the sequence of operations, then clustering to find similarities among production sequences. This way, each cluster can be analyzed independently, and Data Envelopment Analysis [82] is applied to analyze its performance. The output of clustering and operation analysis can be used for improving the company production planning and control activities.

2.1.2 Process Monitoring

Other contributions focus on process monitoring. Lee et al. [83] developed an approach using fuzzy association rule mining with a recursive process mining algorithm, using RFID to continuously update the parameters of a production system and exploit them to detect anomalies. The goal is to find the relationships between production process parameters and product quality: a set of decision rules for fuzzy logic that estimate the quantitative values of the process parameters. Process Mining is responsible for identifying the hidden patterns between process parameters and the finished quality of products to generate a set of fuzzy association rules.

2.1.3 Model Generation

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.

Model Generation with an Underlying Structure

Among others, Shimizu and Van Zoest [84] 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 [85] 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. [33] 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. [86] 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. [87] 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 the system performance 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.

Model Generation without an Underlying Structure

Recent approaches exploit Process Mining (PM) for simulation model generation [88, 89]. The exploitation of process mining allows for an effective de-

velopment of simulation models for manufacturing systems such as flow lines [90]. Process Mining allows not only to estimate parameters and causal relationships, but also logical relationships such as precedences among activities. Hence, it can be used to discover the topological structure of a manufacturing system. Bergmann et al. [91] 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. [92] designed a methodology for the automated generation of formal models of robotic systems starting from the robot code structure and data. Milde and Reinhart [93] developed an approach for joint material flow discovery, parameter estimation, and control policies identification from manufacturing systems event logs. Martin et al. [94] improved inter arrival times modeling by including the mining of parts queuing at the entrance of the system in a parameter estimation methodology. The authors used the proportion of entities queuing at arrival as described in the event log to approximately estimate the parameters of the inter arrival times distribution. Denno et al. [95] developed a methodology to mine the production system structure and used genetic programming to link colored Petri Net states with exceptional system states such as blocking due to a machine unexpected failure. Ferreira and Vasilyev [96] combined PM with logical decision trees to understand the causes of process delays. Martin et al. [97] 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. [98] 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. [99] 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 [75] 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. [78] proposed a predictive model with the aim to use PM to predict online the cycle-times in industrial environments.

2.2 Information-Based Mining

Process Mining can be used effectively to derive relations among information types in a structured system. Such information can be either related to the physical processes themselves, or to the surrounding processes (e.g., production planning, procurement, sales). PM can assist production systems for both the understanding of the dynamics and the development of optimized procedures. Maiorki, Santos, and De Loures [100] classified the recent contributions related to the application of PM in the industrial context, through the reference model for industry 4.0 (RAMI4.0). Information in an event log is classified based on the architecture levels, hence it is possible to extract portion of the logs based on the required level of granularity (e.g., enterprise, factory, workstation). Jo, Noh, and Cho [101] define the scope of a manufacturing intelligence that can visualize shop floor data and detect and solve problems at the business planning level. The authors developed a system that collects data from the shop floor, monitors such data in real time, and quickly detects and responds to problems providing solutions within the shortest possible time. For such capabilities, four main modules are required: (1) data visualization, to allow for the quick identification of issues in the system; (2) process mining, to quickly gather a model of the system; (3) methods for selection and evaluation of alternatives, and (4) a simulation environment to verify the proposed solutions before implementation.

A typical challenge is that event logs for process mining are not available in the required format in information systems. For instance, customer orders are typically stored in CRM systems, while design steps are recorded by PLM tools [102]. Schuh et al. [102] proposed a UML-based data model to describe the data in a manufacturing environment in a way such that an appropriate model can be discovered through process mining. A map of different data sources within a modern information system is provided, which covers all the steps from an initial customer order to the order fulfillment. Fleig, Augenstein, and Maedche [103] proposed the integration of process mining within an ERP environment to analyze the purchase-to-pay and order-to-cash processes. The scope is to determine if new processes must be standardized or kept as individualized in the company.

2.2.1 Performance Evaluation

Once a dataset of traces is created, it is possible to study the interrelations among them at system level. For instance, the combination of events in a system may manifest in bottlenecks in other locations. The combination of

process mining and correlation analysis can identify events that are causally related one another [104]. Further, the addition of attributes to the log allows to mine additional perspectives. For instance, the yield of a particular process (e.g., semiconductor manufacturing) can be recorded in the log and used to derive high-yield paths in the production flow [105]. Another attribute of interest is cost. Indeed, if each event record refers to an activity, the hourly cost of such performance is likely to be known. As a result, the total cost of production can be estimated with the summation of the cost of the activities included in a process [106].

2.2.2 Process Monitoring

Ruschel, Santos, and Loures [107] used process mining techniques to handle shop floor data and obtain information about activity durations, process deviations, cycle-time variations and the progress of the degradation rate. Process mining is used to retrieve the model of a manufacturing process. A predictive model using Bayesian networks is built for each activity [108]. The role of process mining is therefore to feed the probabilistic models of each activity with the time series representing the durations. Varga et al. [109] used event logs from a coke refinery plant to retrieve the set of actions that operators perform frequently in similar situations. The goal is to exploit multi-temporal sequence mining to infer the causal relationship between the alarms and operator actions, together with the effects of these actions.

2.2.3 Model Generation

One of the non structured processes that can be analyzed with process mining is worker's movements. The supervision of manual operation is a very resource-demanding task. The combination of activity recognition and process mining can potentially increase efficiency and effectiveness [110, 111]. The scope is to use the activity data as input to process discovery techniques to reveal knowledge about operators. Together with a big data analysis procedure, process mining can also be used to structure the knowledge hidden in more irregular data sources, such as the text of e-mails [112]. Further, the generation of a model enables prediction capabilities: given the state of execution of a process, knowing how the situation might evolve allows the supervisors to take the proper corrective actions [113]. Flath and Stein [114] propose a toolbox for developing predictive models exploiting machine learning algorithms and the manufacturing process mapped through process mining techniques. PM is used as a support tool to develop the model, to filter the non-relevant

features and to extract process patterns. Ortmeier et al. [115] discuss on how process mining could support life cycle assessment activities in manufacturing, for instance, to identify process deviations and interruptions. Dakic et al. [116] analyze the potentials of two principal process mining software tools and develops a methodology to implement process mining techniques over a case study. The goal is to obtain a process map in the form of a directly follow graph. The authors propose to combine PM with social network analysis to improve the discovery of the organizational perspective of the enterprise.

2.2.4 Quality Management

The availability of data records of several instances in a system allow for investigating the quality perspective. Dogan and Gurcan [117] provide a guide to apply lean six sigma together with data-based analyses. The authors insert PM in a Quality Assessment framework, analyzing the role of different process mining algorithm with respect to lean six sigma approaches, such as DMAIC (define-measure-analyze-improve-control) and claim that the combination of process mining with traditional techniques allows to make effective decisions for quality problems. In general, the underlying research question is to find what combination of process steps distinguishes the parts in which a strategy (e.g., parameter setting) is successful from the parts in which the goal is not reached. Meyer et al. [118] combined process mining with control theory and proposed an iterative approach to enhance treatment strategies by predicting and preventing failures based on information from electronic records. During runtime, the deviations between the time sequence streams of parts in a model can be used to determine concept drifts, namely deviations from the nominal process behavior [119]. When a reference model is available, the quality assessment can be performed through conformance checking methods. Paszkiewicz [120] identified the production management processes that can be assessed with a conformance checking procedure: logical conformance to the formal model, respect of the imposed production policy (e.g., first-in-first-out), quality assurance, performance indicators (e.g., maximum allowed system time), condition-based rules (e.g., re-work), and workload distribution. Saraeian and Shirazi [121] developed a conformance checking module which compares real and expected characteristics of an additive manufacturing process with the goal of preventing cyber-attacks and network intrusions.

2.3 Limitations

Despite the advancements of PM-based approaches, several limitations remain for applications to manufacturing environments. This section elaborates on such limitations, with a focus on the scope of this work.

2.3.1 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 [1]. These models are not only excessive in terms of size, but also inaccurate in estimating performance measures. From intuition, the higher number of paths in the system are represented in a model, the less data are available to estimate parameters on each 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. [122] 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 [123] 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 [124] introduced patterns definitions of commonly used process model constructs, together with an iterative method that captures these manifestations

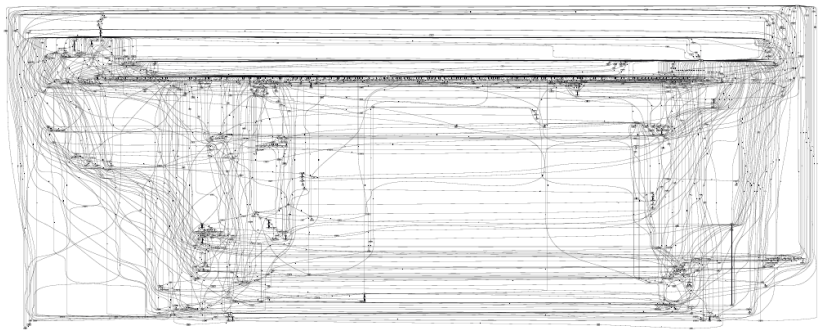


Figure 2.1: Example of a spaghetti model (from [1]).

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 used to generate a model. Prodel [125] 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.

Although some 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 are not suited for manufacturing applications. For instance, fuzzy mining [123] 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 [122] 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 [124], 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.

2.3.2 Discovery of Non-Linear Material Flows

The discovery of more complex systems suffers from the limitations of available methodologies. Indeed, most PM-based approaches are based on the assumption of a single part identifier in the datasets, while in most realistic

environments multiple object types may be involved in a production step (e.g., packaging, batching, assembly). Assembly processes are very common in production enterprises (e.g., automotive industry). Few contributions in the literature explicitly mention mining assembly operations. Denno et al. [95] mined an automotive under-body assembly system with the goal to optimize its production schedule. Rashid and Louis [126] presented a framework that jointly utilizes RFID tracking and process mining techniques to automatically generate the digital model of an assembly line. The goal is to detect any deviation in the performed process from the predefined plan. Knoll et al. [127] developed a methodology to apply process mining to internal logistics for a mixed-model assembly line. The authors used multi-dimensional process mining to automatize and improve the Value Stream Mapping methodology. The goal is to allow for a precise process discovery and classification, including performance analysis to find the parts of the plant where most of the waste is produced.

The aforementioned papers address mining assembly operations. Yet, none of them focus on simulation model generation. Despite being very common in production environments, the discovery of assembly operations has had a scarce representation in the literature. This may be due to the problem of *flattening data*, which is particularly acute for assembly processes, in which different material flows converge toward assembly stations. In such locations, the part identifiers are likely to change from a production stage to the following one. For instance, a car frame is usually assigned an identifier, while sub-components such as doors have other dedicated IDs. Once assembled, the work-in-progress part may have either a new identifier or hold one of the sub-components ID. Either way, the following events in the production system will refer to the assembled product, which is linked to multiple sub-components IDs. Hence, when aggregating events data, there are multiple flattening choices possible which lead to different views that are disconnected [128]. Therefore, the system structure overview is quickly lost as event data need to be extracted multiple times for the different views. As a consequence, the assembly positions can only be assumed or discarded from the representation, unless further manual inspections and modifications are affordable.

The following limitation is evident in the generated model. The simulation model must consider the assembly phase to correctly model the behavior of the system. Namely, the availability of all needed material upstream is a blocking condition on the assembly points. Neglecting this condition in a model generation technique may result in overestimating the performances of the real system. For example, Figure 2.2 graphically shows two different modeling options using a Petri Net. Transition 3 represents the assembly operation, while transitions 1 and 2 are the last operations done by the sub-components.

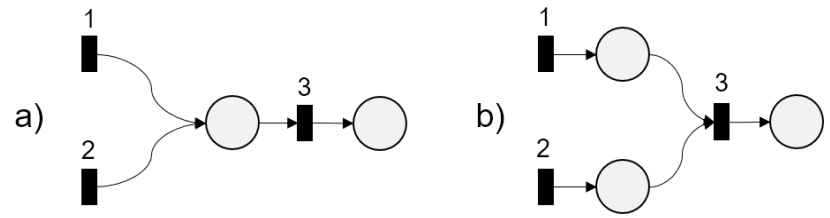


Figure 2.2: Example of assembly process modeled by Petri Nets: a) non-blocking condition (improper modeling), b) blocking condition (assembly properly modeled).

In the first case, transition 3 may fire even if only one of the upstream operations has been completed. Differently, in the second case, the assembly transition is enabled by the availability of all sub-components.

2.3.3 Joint Process Mining Approaches

Most PM applications typically rely solely on one PM algorithm. As a result, the user is forced to compromise between specific algorithm performance measures. For instance, inductive algorithms show good results in terms of scalability, fitness to the log, and precision, while heuristic algorithms are able to discover more process patterns. To cover this shortcoming, more accurate process mining approaches shall be developed. For instance, by using more than one PM algorithm within the same analysis: a joint methodology combining inductive and heuristic mining algorithms could allow to exploit the advantages of each mining algorithm while maintaining reasonably low computation times.

2.3.4 Discovery of Production Policies

PM can be applied not only for the discovery of material flows, but also for revealing which production policies are applied in the shop-floor. The simplest policies are typically related to the production plan and to the understanding of splitting ratios when multiple alternative paths are available. However, realistic environments are characterized by a large number of rules that determine their performance (e.g., maintenance policies, priority rules, lean production guidelines). PM approaches that are able to discover a complete spectrum of production policies are still not available in the literature, and more work is needed in this direction. Further, the ability to distinguish precisely among policies can also enhance the model generation, since different elements of a manufacturing system can be easily differentiated. For instance, a conveyor that serves as a buffer can be characterized by a first-in-first-out policy [93].

2.3.5 Management of Rare Events

A typical challenge of data-driven approaches is how to handle rare events. In a manufacturing environments, such events can be represented by machine breakdowns, unplanned intensive maintenance, extreme weather conditions. Rare conditions are typically more disruptive than more frequent ones [129], and may determine significant detriment to the overall system performance and profitability. Unfortunately, the distinction of such conditions is not trivial and it is often not aligned with the model generation techniques. For instance, in a model tuning methodology, rare events are often disregarded by either frequency- or causality-based scores. Further, automatically generated models may generate rare conditions themselves (e.g., not sound Petri Nets which allow for deadlocks). In this case, the challenge is to distinguish from the correct modeling of rare conditions and modeling errors.

2.3.6 Integration of Expert Knowledge

In manufacturing environments, the workforce competencies are a key requirement to remain competitive [130]. Expert knowledge is often the key driver to successful production operations. PM approaches typically rely solely on event logs as their only source of information. More work is needed toward integrating the company know how in the PM methodologies. Innovative PM approaches may be combined with Machine Learning techniques to achieve improved functionalities, such as predicting the results of certain actions or classifying different production conditions. In this context, expert knowledge can reveal essential to validate such approaches before applying them online.

Scope of Work. This thesis aims at overcoming some of the aforementioned limitations. A method for model tuning in manufacturing systems is presented in Chapter 4, while Chapter 5 presents an approach for the generation of models representing non-linear material flows, specifically assembly operations.

Chapter 3

Lab-scale Models for Testing Real-Time Simulation

In this chapter, we propose a lab-scale environment for testing Real-Time Simulation research and digital technologies for production systems. The proposed laboratory reproduces the ISA95 architecture, hence we developed both physical and digital levels and exploited IoT-compatible devices to connect them. The adoption of such models enables to examine approaches that involve information loops through real industrial components (e.g., gateways, sensors), which is not possible if validation is performed only on digital models. For instance, hardware-related issues such as data sharing between the various layers of a Cyber Physical Production System (CPPS) can be investigated, while remaining capable to implement and iterate the proposed production planning and control logic in reasonable times. Further, the realization of physical models with components such as LEGO, Arduino, fischertechnik, requires lower investments and provides higher flexibility for tests compared to similar settings in real systems.

The chapter is organized as follows. Section 3.1 introduces the problem of Real-Time Simulation within a decision making process for production. Section 3.2 introduces an architecture for the production planning and control of manufacturing systems. Section 3.3 outlines the characteristics of the case study which has been selected for this work (i.e. re-scheduling). The numerical results of the experiments are presented in section 3.4. Final remarks are collected in section 3.5.

3.1 Real-Time Simulation Problem Definition

In this section, we outline a procedure for applying Real-Time Simulation to a generic production system. Let us consider a manufacturing system on which a production policy π is implemented (e.g. priority rule). Further, define $X(t)$ a vector that describes the system state at time t (e.g., current buffer levels), and T is a time horizon of interest. Let us define $\Theta(\pi, X(t), T)$ a generic Key Performance Indicator (KPI) achieved in the time period $[t, T]$ by implementing π on a production system that is in state $X(t)$ at time t.

In general, we can consider a moment $t_e \in [0, T]$ in which a simulation-optimization cycle is launched to determine the production policy to be implemented in the remaining part of the production planning horizon $[t_e, T]$. For instance, t_e may define the instant in which a disruption occurs (e.g., machine failure). Indeed, disruptive modifications may detriment system performances, and the current production policy might not be optimal any further. Therefore, an evaluation is needed to search for a better reaction strategy. Define Π a finite set of $N + 1$ policies $\{\pi_0, \pi_1, \ldots, \pi_N\}$ defined a-priori: π_0 is the original policy implemented on the system and the remaining N policies define alternative reaction scenarios. Further, assume that a discrete-event simulation model of the manufacturing system is available and valid. The simulation model is a digital instance of the manufacturing system and represents the so-called *base model*. At t_e, the digital model is synchronized with the physical system status $X(t_e)$. The performances obtained by each policy are evaluated with digital models derived from the base simulation model. Namely, N *variant models* are created, each implementing the i-th alternative policy π_i. All simulation experiments start from the same time and system state $X(t_e)$. The solution corresponds to the new optimal system management policy π^*, which satisfies:

$$\pi^* = \arg\max_{\pi_i \in \Pi} \{\Theta(\pi_i, X(t), T)\}. \tag{3.1}$$

Notice that $\pi^* = \pi_0$ is allowed, hence this procedure contemplates also the *do-nothing* possibility. Finally, π^* is implemented in the physical system at the time $t'_e \geq t_e$ and the interval $[t_e, t'_e]$ is proportional to the computation effort required to identify the new policy. Figure 3.1 summarizes the temporal evolution of the procedure.

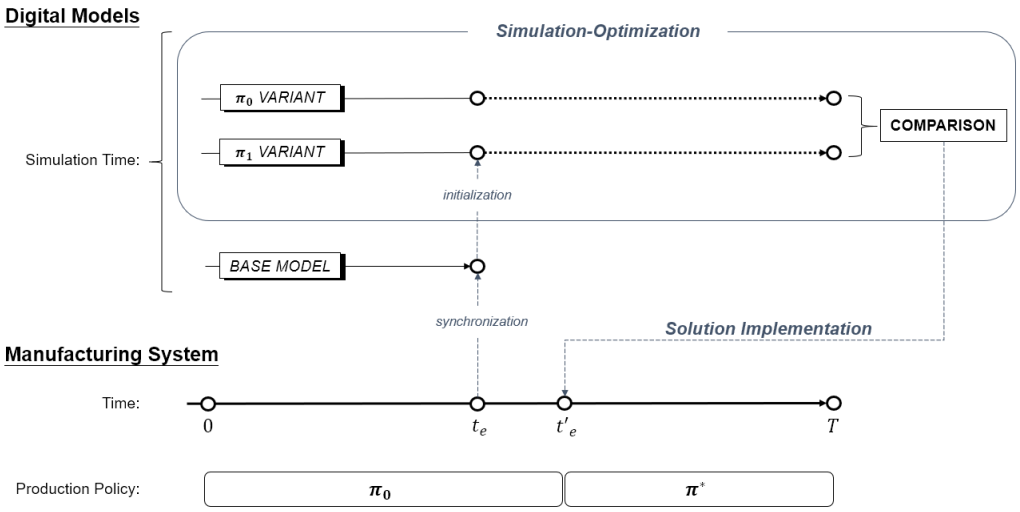


Figure 3.1: Real-Time Simulation procedure illustration with two alternative production policies.

3.2 Production Planning and Control Architecture

An established standard for production management is the ANSI-ISA95 [131]. It is based on five hierarchical levels as shown in Figure 3.2. Level 0 is associated with the physical process; level 1 refers to the governance of actuators and sensors; level 2 represents the control logic and production supervision; level 3 to manufacturing operations (i.e. Manufacturing Execution System); level 4 to the management of the entire firm (i.e. Enterprise Resource Planning). The ISA95 levels are often depicted as a pyramid, although recent research claims that this hierarchical view of the production system has been unsettled by the data sharing capabilities of IoT and cloud computing [56].
The proposed lab-scale models are part of a cyber-physical architecture with four hierarchical levels which as shown in Figure 3.3. The first level is the physical model of a production system built with structural components, sensors, actuators, and Programmable Logic Controllers (PLC). The *execution level* controls the PLCs and converts the sensor outputs into structured data to be sent to the *logic level*; it also receives and releases the motors execution commands. The *logic level* is related to the functions of monitoring and supervising the process and the MES services such as the execution of production orders. The fourth level can be composed by several tools such as simulation-optimization or modules dedicated to production management (e.g. ERP). The communication among the different levels is done by using Internet of Things standards, thus guaranteeing the connection between the physical

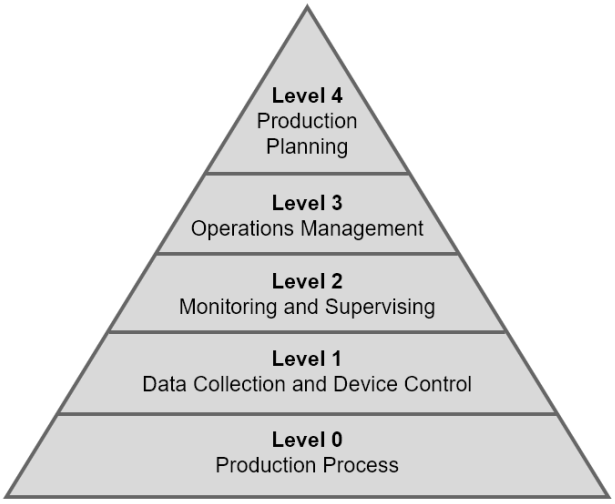


Figure 3.2: ISA95 levels [2].

and digital instances. Hence, the proposed laboratory is effectively a CPPS on a smaller scale.

3.2.1 Physical System

The physical components include both structural pieces such as beams, shafts, conveyor belts, and actuators, sensors and PLCs. The assembled models can be used to replicate the behavior of a real production line by moving parts such as spheres or discs along a proper route and reflect operation times by letting parts wait in a station for an appropriate time span. Exploiting physical system models guarantees several advantages: (1) *high flexibility*, since it is possible to build several kinds of production systems. (2) *facilitated development*, because it is relatively easy to develop a model and the whole assembly process does not require any particular tool. Indeed, small models can be built in a few hours by a single person while the most complex ones can generally be completed within days. (3) *easy management*: it is undoubtedly easier to manage a small scale model with respect to a real production system. The models are usually built in a modular fashion in order to be easily disassembled, transported, and re-assembled. (4) *limited capital expenditure and re-usability*: the cost of the models is orders of magnitude lower than the investments required in a real system. Further, almost 100% of components can be reused after a project completion. Despite the aforementioned advantages, most applications of lab-scale models in industrial engineering focus on educational purposes [132, 133].

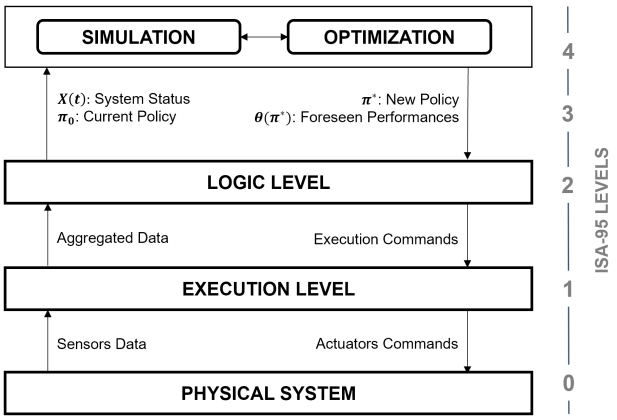


Figure 3.3: The developed architecture with reference to the ISA95 levels.

The physical models proposed in this work have been built with LEGO MINDSTORMS. Next, the common components of the proposed lab-scale manufacturing system models are presented:

- **Parts** are modeled by wooden discs ($\varnothing 35mm$) marked with a color plate. The colors are used to represent different part types and are recognized by the sensors along the system for assigning the right setup and processing times.

- **Conveyors** are controlled by dedicated electrical motors and compose the transportation system that moves the parts in the system. Each conveyor can be set to run at a specific speed which can be changed at runtime.

- **Buffers** are represented by the conveyors which bring parts from a station to another. It is possible to define a specific buffer size through the position of the downstream sensor of each station (sensor 3 in Figure 3.4a), while the maximum buffer size is superiorly limited by the length of the conveyor. Special types of buffers can be modeled as well. For instance, Figure 3.7 shows a model with a three-slide buffer system, in which each part type is stored in a dedicated slide.

- **Stations** are represented by dedicated areas which hold parts for an amount of time that mimics the setup and processing operations on the parts as well as production disruptions such as failures. A station can be in either one among three states: (1) working, (2) idle, and (3) blocked. Figure 3.4a shows an example of a station built with LEGO. A station is composed by an EV3 brick, three EV3 optical sensors, a part-entrance

system and a motor. The part-entrance system is in front of each station. A beam is driven by Motor 1 and blocks the parts in front of the station to avoid the entrance of more than one part at a time. Figure 3.4b summarizes the workflow of the station model. Sensor 1 lies over the part-entrance system to recognize if a part is waiting to be worked. When the station is idle and a part is available, the part-entrance system pushes the part inside the station. Motor 2 drives the part inside the station. Sensor 2 is placed in the middle of the station structure to check if a pallet has entered the machine and to distinguish the product type. As soon as the part has entered, Motor 2 is stopped and the station is set to working state. Sensor 3 is installed on the downstream conveyor and determines if the downstream buffer is full. When the operation is done, if there is enough space on the downstream conveyor, Motor 2 downloads the part and the station is set to idle state. On the other hand, the station is set to blocked state while the downstream buffer is full. The station model exploits three optical sensors to control the part flows.

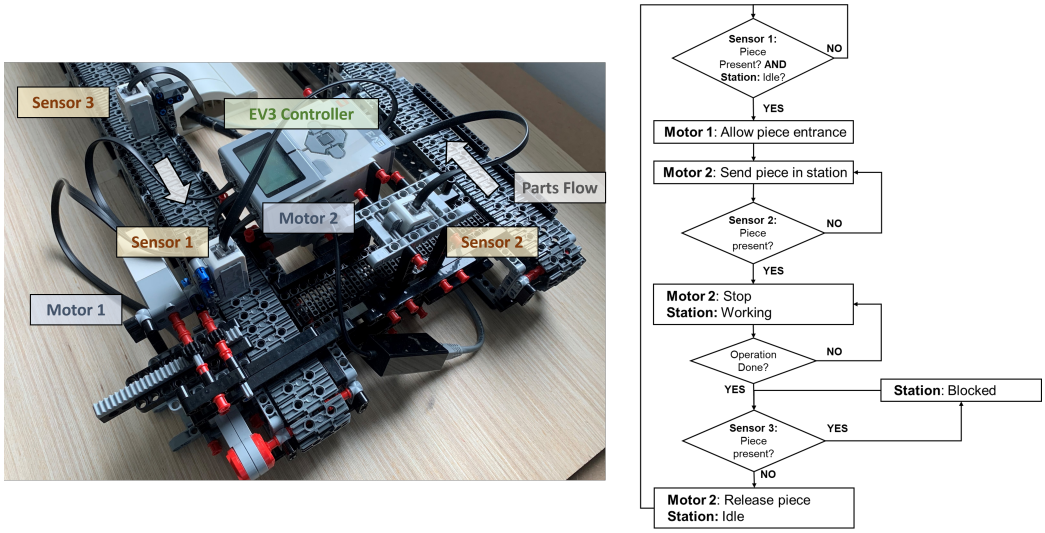
- **Programmable Logic Controllers.** Each station is controlled by an EV3 device. In this work, EV3DEV OS has been used [134]. This open-source operating system is based on Debian Linux and allows the execution of *python* scripts[1] for controlling the sensors and motors through dedicated libraries. Each EV3 is assigned an IP address in a local network and can communicate with a centralized controller. The execution level software is explained in section 3.2.2.

Further details on LEGO-based production system models can be found in related works [133].

3.2.2 Execution Level

This level represents the software running on the PLCs that controls the physical devices. In this work, the execution level consists in a script that runs on each EV3 brick. The execution level is responsible for (1) releasing start/stop commands to the motors whenever required and (2) acquiring and sharing the sensor outputs. The motors activation is triggered by specific messages that communicate the desired actions. The sensor outputs can be conveyed in either two modes: (1) *on-demand*, namely required by a higher hierarchical level or (2) *on-change*, hence triggered by specific events. The code of the execution level is object-oriented. Specifically, three classes correspond to the three

[1] The choice of *python* as programming language is not restrictive and the proposed architecture can be extended to other languages.



(a) (b)

Figure 3.4: Example of station model: (a) physical model components, (b) logical workflow.

main physical devices in the system: the EV3s, the motors, and the sensors. Each class contains an attribute which is a list of all the relative instantiated objects. The classes that have been developed (Figure 3.5a) are the following:

- The **Ev3** class represents the logic controller. All the motors and sensors executed by the controller are listed in the *peripherals_list* and are contextually instantiated at startup.
- The **Motor** class has the attributes *name* and *ev3motor*. *ev3motor* is an object available in the EV3DEV library that allows for interfacing with the motors through *python* commands.
- The **Sensor** class has the attributes *name*, *ev3sensor* and *color_seen*. *ev3sensor* is an object available in the EV3DEV library that allows controlling the EV3 sensors. *color_seen* is an attribute that indicates the last color identified by the sensor.

3.2.3 Logic Level

This level manages production rules, handles constraints and determines the characteristics of the manufacturing system (e.g., station workflow as in Figure 3.4b). At startup, the logic level sends a configuration message to the EV3s

(*ev3_config*) containing a description of the physical system logical layout (i.e. the *peripherals_list*). Further, the logic level contains the messages definition for communicating with the execution level and for exporting significant data towards other management services (e.g., time-series database with the sensor data). In this work, the logic level is a *python* script running on a central controller (e.g., an industrial PC). The code is object-oriented and consists in the following classes (Figure 3.5b). Each class contains an attribute which is a list of all the relative instantiated objects.

- The **Motor** class has two attributes: *name* is the motor identifier and *ev3* is the name of the EV3 device that controls the motor. Motor instances also possess methods corresponding to the executable actions. Whenever one of these methods is called, a corresponding message requesting the motor activation is published on the network to be processed by the execution level (section 3.2.5).
- The **Sensor** class contains the attributes *name* and *output*, where the latter is a dictionary with the indication of the sensor name and information read by the sensor. Moreover, the Sensor class possesses the *read* method that can be used for reading sensor output (*on-demand*).
- The **ColorSensor** class is a subclass of the Sensor class, with the addition of an instance attribute *ev3* representing the name of the EV3 that is connected to the sensor.

3.2.4 Fourth Level

The fourth level of the proposed cyber-physical architecture includes software components exploiting the data from the system to perform several high-level operations. For instance, simulation can be exploited for building digital twins of the physical models. The developed architecture allows to communicate the data measured on the shop-floor, hence it is able to infer the system status and use it as initial condition for simulation models. It is thus possible to simulate different production and management policies in order to determine which one is optimal.

In this work, a digital model of the manufacturing system has been built in Simulink Simevents. The synchronization of the base simulation model is done through *csv* files that contain all the information regarding the production plan and the system status. Specifically, three files describe the nominal processing times, the setup times, and the initial production schedule, respectively. Further, the current system status is represented by two files containing:

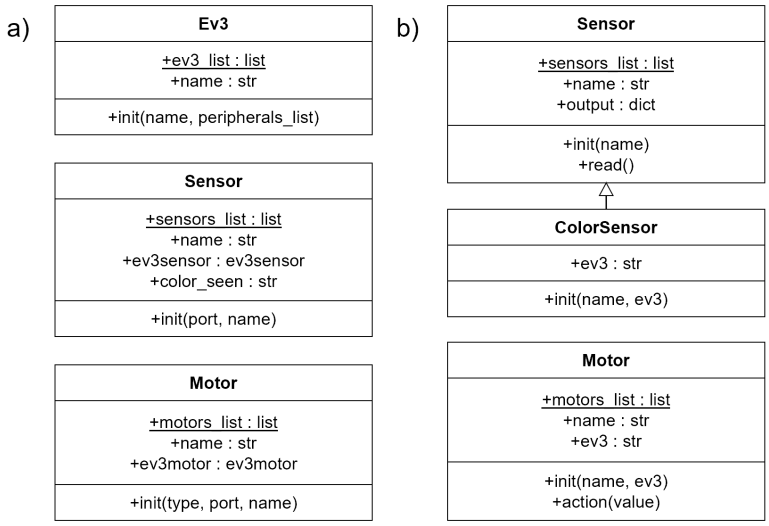


Figure 3.5: The developed classes for (a) the execution level and (b) the logic level.

(1) the job currently under process by each machine and the remaining time until each machine is expected to be in idle state, and (2) the effective production schedule to be followed, which is read each time a machine is idle. The files are shared between the logic level and the simulation model. Hence, the system status can be updated continuously during production.

It is worth to notice that other software components can be added at this level. For instance, data flows management tools are intermediary services to transfer data between two utilities. Databases allow the storage, manipulation, and query of the acquired data for obtaining useful information: for example, the time series of a machine state may be used to derive both availability and reliability indicators. Dashboards are applications for the real-time visualization of both raw data and custom indicators and indexes. Cloud computing components enable the interface with software tools such as ERP.

3.2.5 Communication Protocol

The communication between the software levels is possible thanks to an IoT infrastructure based on the Message Queue Telemetry Transfer (MQTT) protocol. This allows the PLCs to send and receive messages to any kind of IoT-compatible device connected to the network. Hence, it is possible to share and store data from the real system and the architecture levels exploiting the message-based communication protocol. Table 3.1 summarizes the messages exchanged. The messages are written in the JavaScript Object No-

Message Topic	**Source**	**Destination**	**Purpose**
data	Logic Level	Fourth Level	Extract specific data
ev3_config	Logic Level	Execution Level	EV3s configuration
logic_config	Execution Level	Logic Level	EV3s configuration
sensor/request	Logic Level	Execution Level	Sensor output (*on-demand*)
sensor/on_demand	Execution Level	Logic Level	Sensor output (*on-demand*)
sensor/on_change	Execution Level	Logic level	Sensor output (*on-change*)
motor/action/action_name	Logic Level	Execution Level	Activate motors
stop	Any	Execution/Logic Level	Stop software execution

Table 3.1: Summary of the messages exchanged across different levels of the developed architecture.

tation (JSON) format. In the following, two significant examples are explained. Messages from the *sensor/request* topic are sent from the logic level to the execution level and they contain the request to read a specific sensor output. In this case, the message contains the name of the sensor and the EV3 device which is controlling it. Messages of the topic *motor/action* are sent from the logic level to the execution level and contain the actions that should be executed by the motors. In the developed physical models the possible actions are the following: start moving at a certain speed, run for a certain amount of time at a certain speed, turn the axis to a specific angle value, run back-and-forth of a specific angle value, stop. Notice that other actions can be designed accordingly to specific system requirements. The message content is a JSON object containing the motor name, the name of the EV3 that controls it and a value that describes how to execute the prescribed action.

3.3 Case Study: Online Re-Scheduling

This section presents the case study designed to test a real-time production planning method exploiting the proposed lab-scale physical models. The case study refers to a Flexible Manufacturing System (FMS) with parallel machines producing different product types. The system has been chosen with the intent to address a significant level of complexity while maintaining a size that facilitates the understanding of the obtained results.

3.3.1 Manufacturing System

Let us refer to an FMS with parallel machines $m \in \mathbb{M}$. The system has to produce a set of jobs $k \in \mathbb{K}$ belonging to part types $j \in \mathbb{J}$ within an expected time horizon T. Let us accept the short notation $j(k)$ to indicate the part type to which the k-th job belongs. At any time $t \in [0, T]$, the production schedule $\sigma_m(t) = \{j_1(k), \ldots, j_{N_m}(k)\}$ is defined as the sequence of N_m jobs to be produced on the m-th machine from time t until completion. For instance, $\sigma_1(0) = \{1, 1, 2\}$ means that at time $t = 0$ on machine $m = 1$ three jobs are scheduled: the first two jobs belong to part type 1 and are followed by a job of part type 2. The machines are unreliable and can be subject to failures at any time. Let us define three random variables: P_{jm} is the time to process a part of type j on machine m, S_{ijm} is the setup time to switch from producing a job of part type i to a job of type j on machine m, and F_{jm} is the downtime that may occur during the processing of part type j on machine m. Hence, let $\tilde{P}_{jk}$, $\tilde{S}_{ijm}$, $\tilde{F}_{jm}$ indicate the effective processing time, setup time, and down-

time, respectively. The makespan is the time in which all the products in the production schedule have been completed. It can be written as follows:

$$C_{max} = \max_{m} \left\{ \sum_{k \in \sigma_m} \left(\tilde{P}_{j(k),m} + \tilde{S}_{j(k)-1,j(k),m} + \tilde{F}_{j(k)m} \right) \right\}. \tag{3.2}$$

The production schedule is determined exploiting the *predictable schedule* concept proposed by Arnaout [135], which is based on the idea that a robust schedule should contain adequate safety times to account for the expected disruptive events along the production. The safety time is proportional to the expected failure rate of the machines and the production activities duration. For example, if a machine is expected to spend a tenth of the available time in downtime, a predictable schedule would include one unit of safety time every ten time units of scheduled production activity. Let us call $\hat{P}_{jm}$ the expected time to process jobs of part type j on machine m, and $\hat{S}_{ijm}$ is the expected setup time to switch from producing jobs of type i to type j on machine m. The safety time to be accounted for each k-th job scheduled on machine m can be estimated with equation (3.3).

$$ST_{km} = R_m \delta_m (\hat{P}_{j(k)m} + \hat{S}_{j(k)-1,j(k),m}) \left(1 - \frac{PO_{km}}{N_m} \right) \quad \forall\, k \in \sigma_m, m \in \mathbb{M} \tag{3.3}$$

where R_m is the Mean Time To Repair (MTTR) on machine m, δ_m is the estimated number of breakdowns on machine m per unit time, PO_{km} is the k-th job position in the schedule of the m-th machine, and N_m is the total number of jobs that are scheduled on the m-th machine. Figure 3.6 shows an example of schedule including the variables exploited in equation (3.3). Once an initial schedule is set, the system starts to produce pieces. If no failures occur before a scheduled safety time, the latter is removed from the schedule and the next programmed job is anticipated. At any moment t, each machine m has to produce the remaining fraction of its schedule, $\sigma_m(t)$. The Remaining Safety Time (RST) of a machine m at time t is defined as $RST_m(t) = \sum_{k \in \sigma_m(t)} ST_{km}$. Whenever an unexpected event occurs on a machine, the corresponding RST is diminished by the expected failure time, $\hat{F}_{jm}$.

3.3.2 Rescheduling Problem

Consider the situation in which a failure occurs while a machine is working a part. Typically, the incomplete job is simply rescheduled as the next one to be produced on the failed machine as soon as it returns available. Alternatively, it

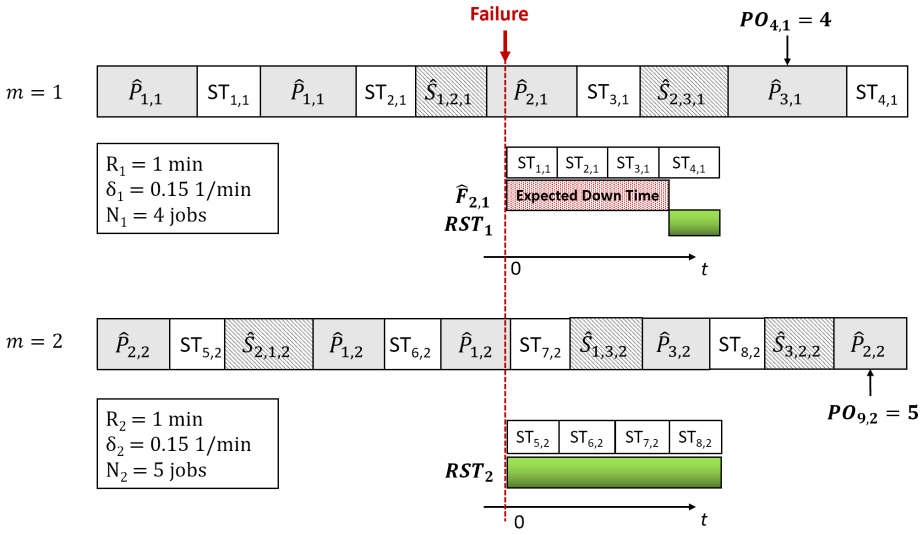


Figure 3.6: Predictable schedule example on two machines with three part types.

is possible to trigger an RTS procedure (section 3.1) in order to evaluate online if a better solution can be found. Let us introduce two policies based on the application of either one of the following reaction rules.

- **Base Policy π_0: Right Shift Repair (RSR)**. The job is rescheduled on the same machine, right after the failure has been resolved. The production schedules on the other machines do not undergo any modification.

- **Alternative Policy π_1: Modified Fit Job Repair (MFJR)**. The Fit Job Repair (FJR) rule has been proposed by Arnaout [135] and prescribes that a job which is unfinished due to a failure at time t_e has to be assigned to the machine with the highest RST. We introduce the Modified FJT (MFJT) which establishes that on the machine chosen by the FJT rule, the job is to be rescheduled in a position that also minimizes the expected setup time.

For example, consider the case in which machine $m = 1$ fails at $t_e = 100s$ and a job of type $j = 2$ has to be rescheduled (Figure 3.6). According to RSR rule, the job will be rescheduled on the same machine just after the end of the downtime, whilst the MFJR rule would reschedule it on $m = 2$, because it is the one with the highest RST. Further, since the schedule on the second machine contains jobs of type $j = 2$, the rescheduled job will be programmed after any job so to guarantee no additional setup time.

Machine: m	Initial Schedule: $\sigma_m(0)$						
1	1	1	2	2	2	3	3
2	3	3	3	1	1	2	2
3	2	2	3	3	1	1	1

Table 3.2: Initial schedule of the FMS model – list of jobs for each machine m (each job is defined by its part type).

3.3.3 Lab-Scale Model

In this case study, we refer to the production planning on an FMS composed by three non-identical parallel machines ($|\mathbb{M}| = 3$). 21 jobs of three part types have to be produced ($|\mathbb{J}| = 3$). The part type is modeled by three different wooden discs colors: blue, red, and white. Each part type can be worked by any of the three machines. The FMS physical model has been built with LEGO MINDSTORM components. Figure 3.7 shows the developed model together with the material flow. The input-output buffer can host 21 parts, 7 per type. In this buffer, parts are hosted in three dedicated sliders and can be released into the system in any sequence.

The initial schedule is shown in Table 3.2. At startup, the initial position of all the discs is in the input-output buffer. The buffer releases the discs into the system according to the production schedule. A job is released as soon as one of the machines is idle. If all the machines are busy, the buffer does not release jobs. The processing and setup times follow uniform distributions as indicated in Table 3.3, and are assigned each time a part enters a machine. Namely, each machine holds a disc for a duration equal to the corresponding sampled production time, and – similarly – it remains idle during setup times. At the end of the assigned processing time, each machine releases the disc on the downstream conveyor and returns to the idle state. In the event of failures, the downtime is modeled as a processing time. Specifically, the involved disc remains in the machine for a duration equal to the downtime before it is sent back to the input-output buffer. The corresponding job is considered as unfinished. The choice of uniform distributions is in accordance with Arnaout [136], who reminds that high variances assure disadvantageous conditions for testing scheduling algorithms.

3.3.4 Experimental Setting

In order to prove the effectiveness of online rescheduling, we have performed experiments based on the application of the RTS procedure presented in sec-

Part Type j:	Processing time [s]		
	$P_{j,1}$	$P_{j,2}$	$P_{j,3}$
1	UNIF(10,12)	UNIF(12,18)	UNIF(5,9)
2	UNIF(12,14)	UNIF(14,20)	UNIF(6,10)
3	UNIF(14,16)	UNIF(16,22)	UNIF(7,11)

		Setup time: S_{ij1} [s]		
	To j:	1	2	3
From i:	1	0	UNIF(28,32)	UNIF(24,32)
	2	UNIF(24,40)	0	UNIF(20,28)
	3	UNIF(28,36)	UNIF(30,32)	0

		Setup time: S_{ij2} [s]		
	To j:	1	2	3
From i:	1	0	UNIF(24,40)	UNIF(20,32)
	2	UNIF(30,40)	0	UNIF(20,32)
	3	UNIF(30,40)	UNIF(24,40)	0

		Setup time: S_{ij3} [s]		
	To j:	1	2	3
From i:	1	0	UNIF(30,40)	UNIF(26,32)
	2	UNIF(36,44)	0	UNIF(28,32)
	3	UNIF(40,48)	UNIF(32,40)	0

Table 3.3: Parameters of the FMS model – Processing and Setup times.

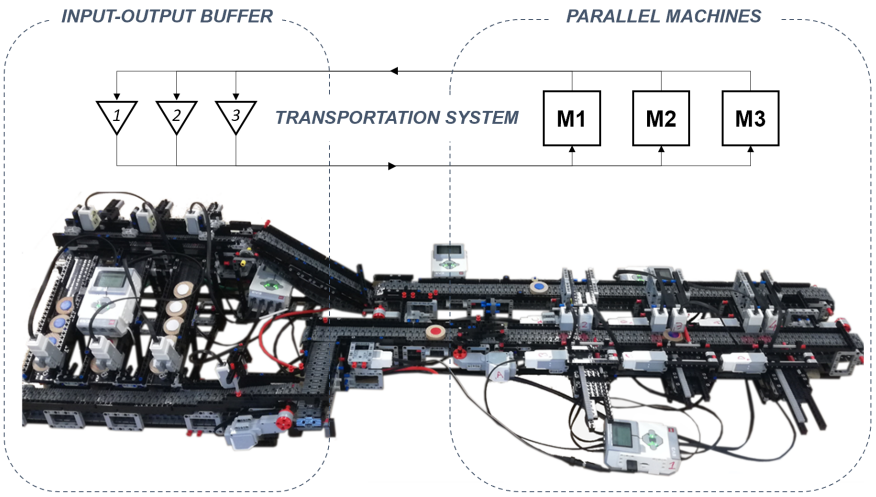


Figure 3.7: FMS model built with LEGO MINDSTORMS.

tion 3.1. We have exploited a discrete-event simulation model built in Simulink Simevents and synchronized with the system as *base model*. The physical system is set to produce according to the initial production schedule. If the schedule is modified during production, the system will produce part types accordingly until the production plan is completed. We have designed two specific cases in which a rescheduling activity is needed during production.

- **Case A: failure at deterministic time.** In this case, machine $m = 1$ fails one time at $t_e = 100s$ for a downtime of $60\,s$. This single scenario is replicated ten times, i.e. the lab-scale physical system is deployed in 10 independent experiments.

- **Case B: failure at stochastic time.** In this setting, three different failure scenarios are considered, one for each machine. In each scenario, one failure happens at a time t_e, which is sampled from an exponential distribution with mean $360s$. The failure duration is $70\,s$. Each scenario is replicated three times and the replications are independent.

Rescheduling is triggered by machine failures. The scheduler is a software at the fourth level of the architecture (section 3.2.4) and consists of a MATLAB script that controls the Real-Time Simulation procedure and communicates the updated schedule to the logic level. Namely, when a failure occurs on a machine, the MATLAB script initializes the *base* simulation model to the system status at the failure moment t_e. Then, two *variants* of this model are used to simulate the production following the schedules generated by both the reaction

rules RSR and MFJR (section 3.3.2). For each rule in each scenario, the simulations are replicated three times in order to account for the noise of machine behavior. If either one of the rules has obtained significantly better results than the other in terms of obtained makespan, the corresponding schedule is executed in the system. The implementation is done automatically by modifying the production schedule files. Let us define $C_{max}^{(S)}(\pi_i)$ the makespan obtained in a variant simulation model of the system by applying the production policy π_i. Hence, the optimal reaction rule which will be applied in the real system satisfies the following:

$$\pi^* = \arg \min_{\pi_i \in \{RSR, MFJR\}} \{C_{max}^{(S)}(\pi_i)\}. \tag{3.4}$$

The effective makespan measured on the lab-scale physical system is $C_{max}^{(L)}(\pi^*)$. Hence, the duration of each run is the effective makespan.

In order to assess the successful implementation of the rescheduling optimal policy, a full factorial design has been performed to compare the obtained results in terms of makespan C_{max} with the following factors:

- *Rescheduling* is a two-level factor that describes the following conditions: (1) Rescheduling OFF: rescheduling is not allowed and if a failure occurs the base policy is always applied (i.e. RSR). Hence, the makespan in this case is always $C_{max}^{(L)}(RSR)$. (2) Rescheduling ON: both the RSR and the MFJR rules can be applied. In accordance with the RTS procedure, the reaction rule obtaining the lowest makespan in the digital model is applied online in the physical system.
- *Evaluation tool* is a factor that indicates if the makespan has been obtained by the simulation model (i.e. the average value of three online simulation runs) or by the physical system. Hence, the factor has two levels: (S) Simulation model, (L) Lab-scale physical model.
- *Scenario* is a factor only used for case B. It has three levels that correspond to the three respective failure scenarios: 1, 2, and 3 (section 3.3.4).

All the experiments have been done using a laptop with a 1.60GHz CPU and 8.00GB memory. Section 3.4 presents the numerical results.

3.4 Numerical Results

The proposed lab-scale model and the related architecture have been used to assess the advantage of the rescheduling approach described in section

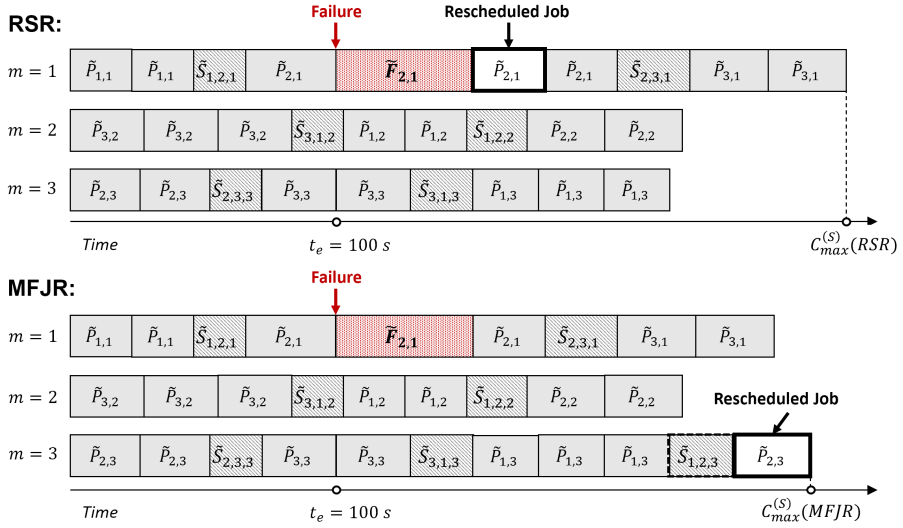


Figure 3.8: Case A – The schedules generated by RSR and MFJR rules in response to the failure on $m = 1$.

3.3.2 in the two cases listed in section 3.3.4. Table 3.4 summarizes the results that have been obtained in each experimental condition in terms of average makespan $\bar{C}_{max}$. The next two sections comment on the numerical results.

3.4.1 Case A: Deterministic Failure

In this case, the production is affected by a failure at $t_e = 100s$. Ten replications are done for each experimental condition. Since in each replication three simulation runs are performed, this experiment counts 60 data points for simulation and 20 for the physical model. Figure 3.8 shows two of the schedules generated by the two reaction rules. The MFJR rule always resulted more advantageous than RSR and has been applied online. The values of makespan measured on the physical system and foreseen by the simulation model can be found in Table 3.6.

Figures 3.9 and 3.10 show the main effects plot and the interaction plot of the results obtained in this case, respectively. Due to non normality of ANOVA residuals, we performed a non parametric test (i.e. Kruscall-Wallis) on the two factors separately. Table 3.5 shows the obtained results. The *Evaluation tool* factor is not significant and it demonstrates the alignment between the physical system and its digital counterpart. Hence, we may infer that the optimal policy found on the simulation model is also optimal on the physical system. This is confirmed by the numerical results of this case because in all the experiments the MFJR rule outperformed RSR both in the digital and physical mod-

Resch.	Case	Scenario	Simulation (S)				π^*	Lab-scale model (L)	
			$\bar{C}_{max}^{(S)}$(RSR)	95% C.I.	$\bar{C}_{max}^{(S)}$(MFJR)	95% C.I.		$\bar{C}_{max}^{(S)}(\pi^*)$	95% C.I.
ON	A	-	404.9	(403.7, 406.2)	**363.2**	(362.3, 364.2)	MFJR	364.4	(362.5, 366.3)
	B	1	404.8	(400.8, 408.7)	**379.3**	(376.7, 381.9)	MFJR	373.0	(367.0, 379.1)
		2	**370.3**	(368.8, 371.7)	375.7	(373.2, 378.3)	RSR	368.6	(365.0, 372.2)
		3	449.2	(447.2, 451.1)	**407.3**	(404.9, 409.7)	MFJR	411.3	(398.0, 424.5)
OFF	A	-	404.9	(403.7, 406.1)	363.2	(362.3, 364.2)	RSR	407.3	(404.9, 409.7)
	B	1	403.5	(399.2, 407.9)	372.4	(368.1, 376.7)	RSR	407.1	(403.5, 410.8)
		2	370.3	(368.8, 371.7)	375.7	(373.2, 378.3)	RSR	368.6	(365.0, 372.2)
		3	449.5	(446.7, 452.3)	406.7	(404.4, 409.1)	RSR	454.8	(446.1, 463.6)

Table 3.4: Cases A and B – Comparison between the makespan foreseen by the simulation model (S) and the one measured on the lab-scale model (L).

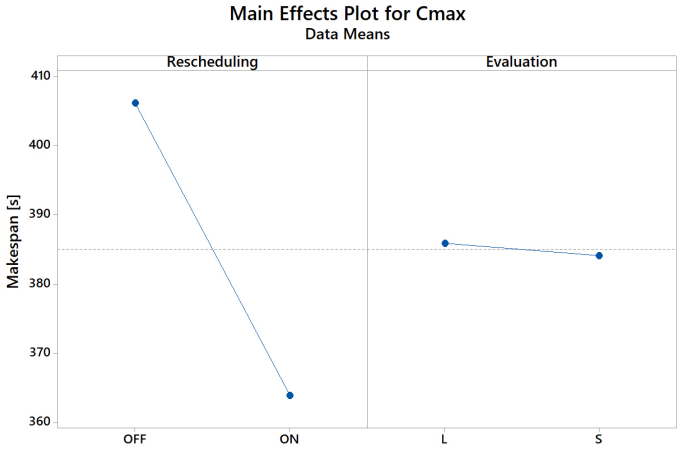


Figure 3.9: Case A – Main effects plot for C_{max} depending on the two factors: rescheduling condition, evaluation tool.

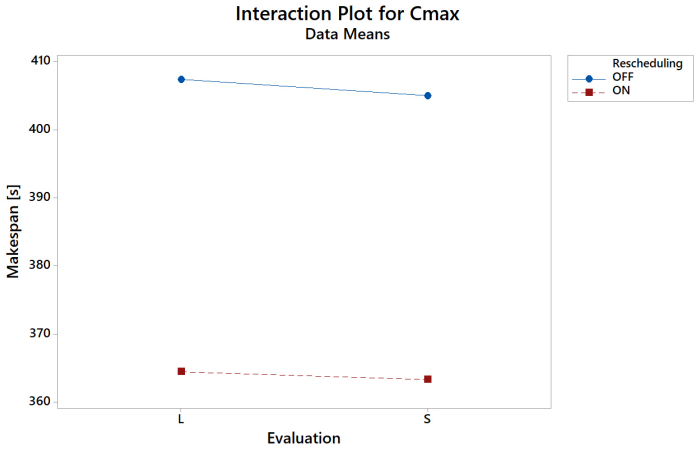


Figure 3.10: Case A – Interaction plot for C_{max} depending on the two factors: rescheduling condition, evaluation tool.

Factor	DF	H-Value	P-Value
Evaluation	1	0.9	0.344
Rescheduling	1	29.27	0.000

Table 3.5: Case A – Kruscall-Wallis test results on the two factors.

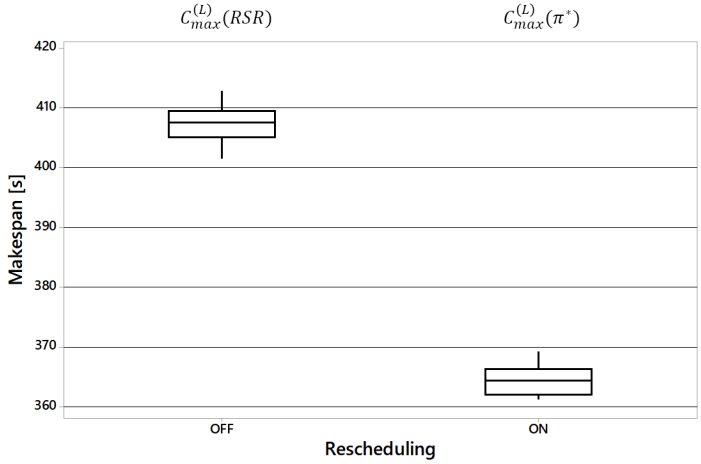


Figure 3.11: Case A – Box plots comparing the actual makespan obtained in the conditions (1) Rescheduling ON and (2) Rescheduling OFF (10 data samples).

els. The *Rescheduling* factor is significantly influencing the response. Figure 3.11 shows the box plots of the makespan obtained in the two rescheduling conditions for Case A. The average difference is $42.91s$ and it is contained in the 95% confidence interval $[40.05; 45.77]$. Finally, from the numerical results we can conclude that the online RTS-based rescheduling led to a significant improvement of the system performance.

3.4.2 Case B: Stochastic Failure

In this case, three replications are made in each experimental condition. For each replicate, three simulations for both the rules are performed. Hence, this experiment counts 18 data points for the physical system and 54 for the digital model. The values of C_{max} measured on the physical system and foreseen by the simulation model are collected in Table 3.8.

Figures 3.12 and 3.13 show the main effects plot and the interaction plot of the makespan values depending on the experimental factors, respectively. Due to missed normality of ANOVA residuals, a non-parametric test (i.e. Kruscall-Wallis) has been performed to test the influence of the three factors. Table 3.7 shows the results. The factors *Rescheduling* and *Scenario* are significantly influencing the makespan results, while the *Evaluation tool* factor is not significant. Also in this case, the alignment between the digital and physical model is demonstrated by the fact that the *Evaluation tool* factor is not significantly influencing the makespan results. Figure 3.13 shows that there is an interaction between the factors *Rescheduling* and *Scenario*. Indeed, in this case, MFJR

Rescheduling	Replication	$C_{max}^{(S)}$(RSR)	$C_{max}^{(S)}$(MFJR)	$C_{max}^{(L)}$(RSR)	$C_{max}^{(L)}$(MFJR)	$t'_e - t_e$
ON	1	403.4 402.2 409.3	360.8 362.7 366.3	369.2		57.9
	2	402.7 401.5 408.6	360.1 362.0 365.6	366.0		54.5
	3	404.1 402.8 409.9	361.5 363.3 366.9	362.8		51.2
	4	403.8 402.5 409.6	361.2 363.0 366.7	362.1		58.4
	5	400.5 399.3 406.3	357.9 359.8 363.4	366.2		53.7
	6	404.3 403.0 410.1	361.7 363.5 367.2	361.7		52.7
	7	404.1 402.8 409.9	361.5 363.3 366.9	366.0		53.7
	8	404.2 402.9 410.0	361.6 363.4 367.0	361.2		51.2
	9	403.8 402.6 409.6	361.2 363.1 366.7	366.6		53.8
	10	403.6 402.4 409.4	361.0 362.9 366.5	362.7		58.3
OFF	1	403.4 402.2 409.2	360.8 362.7 366.3		408.4	57.8
	2	402.7 401.4 408.5	360.1 361.9 365.5		412.9	53.2
	3	404.0 402.7 409.8	361.4 363.2 366.8		406.4	52.9
	4	403.8 402.6 409.6	361.2 363.1 366.7		401.4	57.4
	5	400.5 399.2 406.3	357.9 359.7 363.4		408.7	53.6
	6	404.3 403.0 410.1	361.7 363.5 367.1		405.6	52.0
	7	404.1 402.8 409.9	361.5 363.3 366.9		406.5	53.1
	8	404.2 402.9 410.0	361.6 363.4 367.0		408.9	52.1
	9	403.8 402.5 409.6	361.2 363.0 366.7		403.9	54.7
	10	403.6 402.4 409.5	361.0 362.9 366.5		411.0	58.1

Table 3.6: Case A – Experimental results.

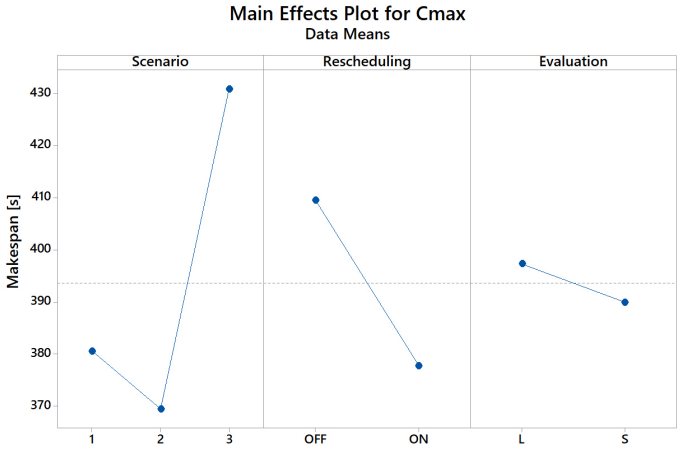


Figure 3.12: Case B – Main effects plot for C_{max} depending on the three factors: Scenario, Rescheduling condition, Evaluation tool.

Factor	DF	H-Value	P-Value
Scenario	2	26.06	0.000
Rescheduling	1	3.85	0.050
Evaluation	1	0.03	0.874

Table 3.7: Case B – Kruscall-Wallis test results on the three factors.

rule has not always been applied online. Specifically, in the second scenario, a failure occurs on $m = 3$ at $t_e = 46s$ on a job of part type $j = 2$. Given the system state, the most performing decision is to reschedule the failed job according to the RSR rule. Hence – as expected – in the second scenario there is no significant difference between enabling rescheduling or not. This demonstrates that even if a specific reaction rule could prove to be better on average, it may still perform worse in certain settings. Real-Time Simulation is able to identify these cases and it allows for a prompt evaluation of the best decision to take so that the performance of a system in a particular situation can be maximized.

3.5 Conclusions

In this work, we have proposed a lab-scale environment that exploits physical models of manufacturing systems to test production planning and control approaches based on Real-Time Simulation. By exploiting easy-to-build com-

Rescheduling	Scenario	Replication	$C_{max}^{(S)}$(RSR)	$C_{max}^{(S)}$(MFJR)	$\tilde{F}$	$C_{max}^{(L)}$(RSR)	$C_{max}^{(L)}$(MFJR)
ON	1	1	398.2 410.0 402.9	379.0 381.7 379.8	169.0		374.9
		2	288.7 276.6 280.7	262.2 256.1 257.9	168.9		374.1
		3	403.0 412.1 402.3	375.0 379.9 373.7	168.8		370.3
	2	1	368.8 365.2 368.5	374.7 376.4 373.6	46.0	366.4	
		2	373.5 368.8 368.3	373.5 384.1 388.3	46.0	365.3	
		3	372.7 369.2 372.7	371.3 373.3 379.4	46.0	368.6	
	3	1	447.6 449.5 453.1	405.2 402.4 412.0	178.7		413.3
		2	445.3 449.3 447.4	407.1 412.1 408.5	178.8		415.4
		3	450.2 452.9 447.8	406.8 406.8 405.4	178.7		405.3
OFF	1	1	401.4 399.1 415.9	374.9 366.6 376.9	169.0	408.5	
		2	398.6 404.8 406.1	371.3 380.1 377.3	169.0	405.6	
		3	397.3 403.1 405.8	372.6 362.6 369.4	169.0	407.5	
	2	1	370.8 368.5 374.3	369.9 370.8 372.9	45.9	374.9	
		2	377.0 369.9 367.0	374.3 375.7 378.1	46.0	367.3	
		3	367.3 371.2 372.0	381.7 377.8 368.1	45.9	369.5	
	3	1	449.1 443.5 451.8	408.2 402.2 410.6	178.8	454.1	
		2	448.9 451.3 447.1	404.3 407.2 411.7	178.8	458.7	
		3	456.3 451.2 447.0	406.4 405.8 404.5	178.7	451.8	

Table 3.8: Case B – Experimental results.

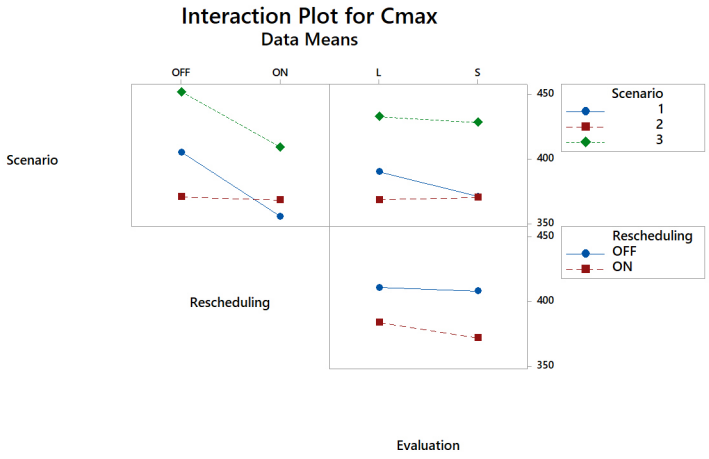


Figure 3.13: Case B – Interaction Plot for C_{max} depending on the three factors: Scenario, Rescheduling condition, Evaluation tool.

ponents such as LEGO, several types of manufacturing systems can be translated into lab-scale models in a very short time. The case study proved that RTS-based production planning and control approaches can be assessed on lab-scale models of common manufacturing systems. Differently from tests performed on real factories, the proposed lab-scale models allow for a much more versatile and cost-effective setting, while maintaining the information loop through industrial components. Hence, we believe the proposed laboratory will be beneficial in several industrial applications. For instance, the design of new production control algorithms could include a testing phase exploiting the physical models. Similarly, new devices such as PLCs could benefit from trials on lab-scale models.

Several issues still need to be solved. The synchronization between digital and physical models theoretically allows for the online identification of the optimal production policy. However, the computation time represents a major obstacle. Indeed, although in this work the online comparison has involved two alternative policies, the time span $t'_e - t_e$ has lied between 51 and 59 seconds, which is very close to the downtime duration. In general, simulation time is non negligible and it represents one of the most important challenges of RTS: the number of replications are superiorly limited by the necessity of a timely application of the prescribed actions on the system. On the other hand, reducing the simulation effort may weaken the confidence in the simulation results. Hence, more work is needed for testing production planning problems requir-

ing higher computation effort. In the future, we aim at providing more case studies based on different types of manufacturing systems and introducing more production policies alternatives to explore the applicability boundaries. Another interesting development of this work is the study and formalization of the component types which can benefit from an increased Technology Readiness Level (TRL) [137]. The TRL that can be obtained by an integrated system can also be assessed, by taking into account the interactions and compatibility between all connected components and digital models.

Chapter 4

Generation and Tuning of Digital Models

This chapter proposes a method for obtaining simulation-based digital twins starting from the data logs of manufacturing systems. The contribution of this chapter 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 chapter is organized as follows. Section 4.1 describes the problem of system discovery and model generation with a proper level of detail. Section 4.2 outlines the main steps covered by the model generation methodology. Section 4.3 presents the proposed model tuning method. The numerical experiments are listed in section 4.4, while final remarks are in section 4.5.

4.1 Problem Description

Figure 4.1 graphically explains one of the possible outcomes of an automated model generation procedure, using as example the three-station production line shown in figure 4.1a. 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 4.1b 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

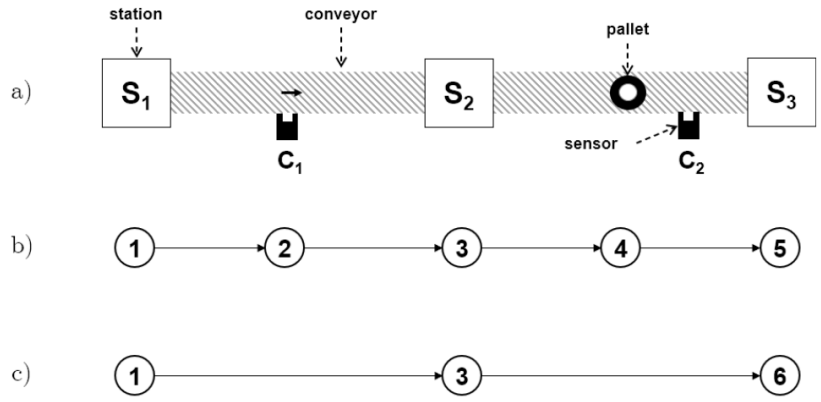


Figure 4.1: (a) Sequential 3-station production line; (b) graph model with 5 nodes; (c) graph model with 3 nodes.

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 4.1c 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 [138].

This work focuses on modeling discrete parts manufacturing systems [139]. 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 [140]. 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 characteris-

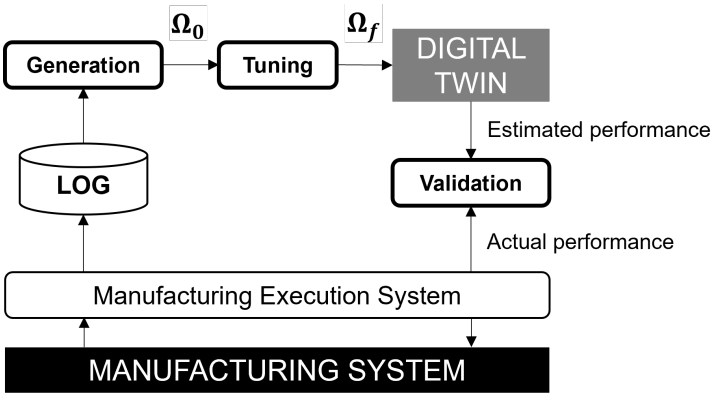


Figure 4.2: Graphical map of manufacturing system discovery and digital twin generation.

tics. 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.

4.2 Data-Driven Manufacturing System Discovery

Figure 4.2 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 Ω_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 4.3 we present the proposed model tuning method.

4.2.1 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. Fur-

Table 4.1: 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

ther, 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 Θ is the set of all the traces identified from the log[1]. A trace is the specific 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 4.1 shows an example of event-log generated by the manufacturing system of Figure 4.1. In this example, the trace of part $h = 1$ is $\theta_1 = \{S_1, C_1, S_2, C_2, S_3\}$.

4.2.2 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 Ω 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 4.1c is defined

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

Table 4.2: Notation for digital models.

Nodes $n \in \mathbb{N}$	$\mathbb{P}_n$	Predecessor nodes set.
	$\mathbb{S}_n$	Successor nodes set.
	κ_n	Buffer capacity of a node.
	ϕ_n	Frequency of a node.
	ξ_n	Number of close events on a node.
	π_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.

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, respectively, κ_n is the buffer capacity of the node, ϕ_n the frequency of occurrence of the respective activity, π_n the set of branching probabilities, ξ_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 η_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 4.2 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.

4.2.3 Model Generation

Model generation is a procedure which links the data in the event log with a digital model Ω. 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 κ_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:

$$\hat{\kappa}_n = \max k \,|\, t_S(n, h-k) \leq t_F(n, h) \quad \forall n \in \mathbb{N}. \tag{4.1}$$

The frequency ϕ_n indicates the number of times the corresponding activity has been observed in the log, while ξ_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 ζ_N is a user-defined threshold. Additional parameters define the branching policies π_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:

$$\hat{p}_s = \frac{\phi_s}{\sum_{o \in \mathbb{S}_n} \phi_o} \quad \forall s \in \mathbb{S}_n. \tag{4.2}$$

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

$$\tau_{h,n} = t_F(n, h) - t_S(n, h) \quad \forall h \in \mathbb{H}, n \in \mathbb{N}. \tag{4.3}$$

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:

$$c_a = \max k \,|\, t_F(n, h-k) \leq t_S(m, h) \quad \forall a \in \mathbb{A} \,|\, \eta_a = (n, m). \tag{4.4}$$

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 ζ_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:

$$t_{h,a} = t_S(m, h) - t_F(n, h) \quad \forall h \in \mathbb{H}, \forall a \in \mathbb{A} \,|\, \eta_a = (n, m). \tag{4.5}$$

Model generation identifies the nodes and arcs and collects them in an directed graph Ω_0, which is populated with the properties of nodes and arcs through Algorithm 1. Algorithm 1 is used to generate an initial model Ω_0 starting from an event log.

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 [141]. Among others, one of the conversion procedures that can be used is the α-algorithm, which can be found in [1]. Notice that also simulation graphs (i.e., event relationship graphs) can be obtained, for instance exploiting direct conversion from Petri Nets [142].

4.3 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 4.2.3, 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 (4.6), which determines how acceptably a model Ω represents the real production system.

$$\Phi(\Omega) = \sum_i w_i \, R_i(\Omega) \tag{4.6}$$

where each score $R_i(\Omega)$ is a function that describes *how well does the model Ω 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 4.3.1. Table 4.3 summarizes the notation used for the model tuning.

Algorithm 1: Model Generation – Find the initial model Ω_0.

Input: Event Log;
Output: Graph model Ω_0;
Definition: $Count(i|Condition)$ returns the number of times $Condition$ is satisfied for element i.
STEP 1 – Generate traces set Θ 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 Ω_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**
      $\hat{p}_s = \frac{\phi_s}{\sum_{o \in \mathbb{S}_n} \phi_o}$;
      $\pi_n \leftarrow \pi_n \cup \{(s, \hat{p}_s)\}$
    $n \leftarrow \pi_n$
return $\Omega_0 \leftarrow \{\mathbb{N}, \mathbb{A}\}$;

4.3.1 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.

$$R_1(\Omega) = r_1^{(A)} \frac{\sum\limits_{a \in \mathbb{A}} c_a}{\sum\limits_{a \in \mathbb{A}_0} c_a} + r_1^{(N)} \frac{\sum\limits_{n \in \mathbb{N}} \kappa_n}{\sum\limits_{n \in \mathbb{N}_0} \kappa_n} \tag{4.7}$$

  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.

Table 4.3: 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.
ν_{in}	maximum number of input arcs in a node: $\nu_{in} \in [0, \max_{n \in \mathbb{N}_0}\{\mathbb{P}_n\}]$.
ν_{out}	maximum number of output arcs from a node: $\nu_{out} \in [0, \max_{n \in \mathbb{N}_0}\{\mathbb{S}_n\}]$.
ι_{ij}	number of loops in which the arc (i, j) is involved.
Decision Variables:	
$\beta = \{\beta_i\}$	vector of included activities: β_i is 1 if the i-th activity is included in the graph, 0 otherwise.
$\mathbf{\Gamma} = \{\gamma_{ij}\}$	matrix of connectors between activities: γ_{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.

$$R_2(\Omega) = \frac{r_2^{(A)}}{|\mathbb{A}|} \sum_{a \in \mathbb{A}} (1 - \frac{e_a}{f_a}) + \frac{r_2^{(N)}}{|\mathbb{N}|} \sum_{n \in \mathbb{N}} (1 - \frac{\xi_n}{\phi_n}) \tag{4.8}$$

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.

$$R_3(\Omega) = \frac{1}{|\mathbb{A}|} \sum_{n \in \mathbb{N}} \sum_{m \in \mathbb{N}} \gamma_{nm} \iota_{nm} \tag{4.9}$$

where ι_{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 [143].

- **Split and merge.** R_4 is a routing score defined in [125]. 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.

$$R_4(\Omega) = r_4^{(in)} \sum_{n \in \mathbb{N}} \sum_{x \in \mathbb{S}_n} \gamma_{nx} + r_4^{(out)} \sum_{n \in \mathbb{N}} \sum_{l \in \mathbb{P}_n} \gamma_{ln} \tag{4.10}$$

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).

$$R_5(\Omega) = r_5^{(A)} \frac{\sum\limits_{a \in \mathbb{A}} f_a}{\sum\limits_{a \in \mathbb{A}_0} f_a} + r_5^{(N)} \frac{\sum\limits_{n \in \mathbb{N}} \phi_n}{\sum\limits_{n \in \mathbb{N}_0} \phi_n} \tag{4.11}$$

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

4.3.2 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 Ω_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 Ω_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* Φ *defined in equation* (4.6) *while satisfying a set of constraints (e.g., number of nodes)*.

Model tuning may be expressed as a mathematical programming model through equations (4.12) - (4.22). The decision variables are defining the selection of nodes and arcs to keep or aggregate in the model. Specifically, β_i defines if the i-th activity is kept in the representation or not, γ_{ij} guarantees the precedences of activities are maintained. The clustering of nodes is represented by variable m_{ic}.

$$\max_{\Omega} \quad \Phi(\Omega) \tag{4.12}$$

$$s.t. \quad \sum_{i} \beta_i \leq U_N^{max} \tag{4.13}$$

$$\sum_{ij} \gamma_{ij} \leq U_A^{max} \tag{4.14}$$

$$\gamma_{ij} \leq \mathbf{MXM}^T \qquad \forall i,j \tag{4.15}$$

$$\sum_{i} m_{ik} \leq 1 \qquad \forall k \tag{4.16}$$

$$\gamma_{ij} \leq \beta_i \qquad \forall i,j \tag{4.17}$$

$$\gamma_{ij} \leq \beta_j \qquad \forall i,j \tag{4.18}$$

$$\gamma_{ii} \leq L \qquad \forall i \tag{4.19}$$

$$\sum_{i} \gamma_{ij} \leq \nu_{in} \qquad \forall j \tag{4.20}$$

$$\sum_{j} \gamma_{ij} \leq \nu_{out} \qquad \forall i \tag{4.21}$$

$$\beta_i, \gamma_{ij}, m_{ci} \in \{0,1\} \qquad \forall i,j,c. \tag{4.22}$$

The optimization function (4.12) represents the maximization of the model score expressed by equation (4.6). Constraints (4.13) and (4.14) limit the number of nodes and arcs included in the model, respectively. Constraints (4.15) translate the precedences among events to precedences among clusters. Constraints (4.16) force each activity to be included in at most one cluster. Constraints (4.17) and (4.18) guarantee that arcs are included only between existing nodes. Constraints (4.19) allow or forbid for self-loops depending on the value of L. Constraints (4.20) and (4.21) limit the number of input and output arcs from each node, depending on ν_{in} and ν_{out}. Constraints (4.22) state the nature of the decision variables.

4.3.3 Solution Methodology

The model tuning problem described in equations (4.12) - (4.22) is solved with the local search algorithm described in Algorithm 3. Algorithm 3 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 4.3.3 explains the local search algorithm.

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 4.3 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:

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

  Reduction:

  $$f_a = \max\{\max_{n\in\mathbb{N}}\{\phi_n\}, \max_{b\in\mathbb{A}}\{f_b\}\} \quad \forall n, b \,|\rho(n, a) = 1 \wedge \rho(b, a) = 1. \tag{4.23}$$

  Aggregation:

  $$\phi_c = \max\{\max_{n\in\mathbb{N}}\{\phi_n\}, \max_{a\in\mathbb{A}}\{f_a\}\} \quad \forall n, a \,|\alpha(n, c) = 1 \wedge \alpha(a, c) = 1. \tag{4.24}$$

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

  Reduction:

  $$f_a = \min\{\min_{n\in\mathbb{N}}\{\xi_n\}, \min_{b\in\mathbb{A}}\{e_b\}\} \quad \forall n, b \,|\rho(n, a) = 1 \wedge \rho(b, a) = 1. \tag{4.25}$$

  Aggregation:

  $$\xi_c = \min\{\min_{n\in\mathbb{N}}\{\xi_n\}, \min_{a\in\mathbb{A}}\{e_a\}\} \quad \forall n, a \,|\alpha(n, c) = 1 \wedge \alpha(a, c) = 1. \tag{4.26}$$

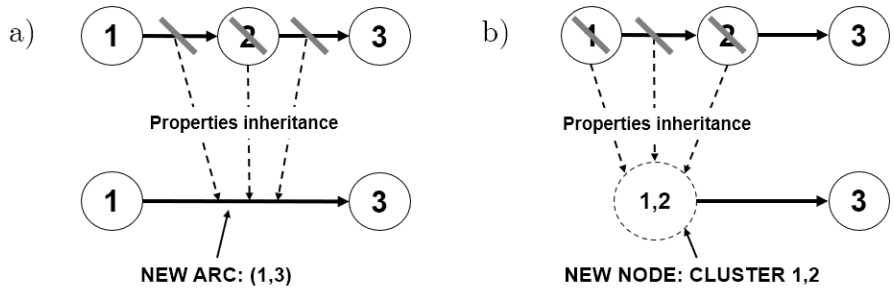


Figure 4.3: Properties inheritance example: a) reduction, b) aggregation.

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

  Reduction:

$$c_a = \sum_{n,b \,|\rho(n,a)=1 \wedge \rho(b,a)=1} \kappa_n + c_b \tag{4.27}$$

  Aggregation:

$$\kappa_c = \sum_{n,a \,|\alpha(n,c)=1 \wedge \alpha(a,c)=1} \kappa_n + c_a \tag{4.28}$$

Algorithm

Algorithm 3: Local Search – Find the most adequate model Ω_f of size U_N^{max}.

Input: Graph model Ω_0, size limit U_N^{max};
Output: Graph model Ω_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 1);
    STEP 2: Sort neighbors based on score Φ;
    STEP 3: Set $\Omega_{current} \leftarrow argmax_{\{\Omega \in \mathbb{M}_A \cup \mathbb{M}_R\}} \Phi(\Omega)$, $i \leftarrow i + 1$;
end
return $\Omega_f \leftarrow \Omega_{current}$;

Algorithm 3 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 4, which produces a set of $\sigma_A + \sigma_R$ neighbors of an

input model Ω. *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*(Ω, (n, m)) is a function that generates a neighbor of model Ω, starting from the aggregation of nodes (n, m) into a cluster, while respecting the inheritance rules defined in section 4.3.3. Similarly, *GenerateNeighborByReduction*(Ω, n) is a function that generates a neighbor of model Ω, starting from the removal of node n, while respecting the inheritance rules defined in section 4.3.3.

Algorithm 4: Model Tuning – Generate neighbors of model Ω.

Input: Model Ω, number of neighbors from aggregation σ_A, number of neighbors from reduction σ_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 $\mathbb{M}_A, \mathbb{M}_R$;

Let us call σ_A and σ_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 3. At any i-th iteration, the complete exploration of the neighbors set is achieved when $\sigma_A = \sigma_R = |\mathbb{N}_i|$. Since 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.

4.3.4 Parameters Estimation

Once a final model Ω_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 (4.3), 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:

$$\tau_{h,n} = t_F(n,h) - t_F(n,h-1) \quad \forall n \in \mathbb{C}, h \in \mathbb{H}. \tag{4.29}$$

4.3.5 Control Policies Identification

At the end of model tuning, a graph model Ω_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 [93].

4.4 Numerical Experiments

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

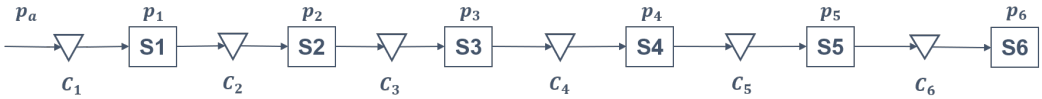


Figure 4.4: 6-station flow line used for the experiments in this work. Squares depict stations, while inter-operational buffers are represented as triangles.

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 4.3.2.

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.

4.4.1 Calibration

The goal of this section is to determine which configuration of weights contributes the most to the objective function described in equation (4.12). The function is a weighted sum of the scores defined in section 4.3.1. 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 4.4. 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.

For the calibration experiments, we have exploited a space-filling design with a 256 points over the values of the weights of objective function (4.12). Each point has been replicated five times. The replications are defined by

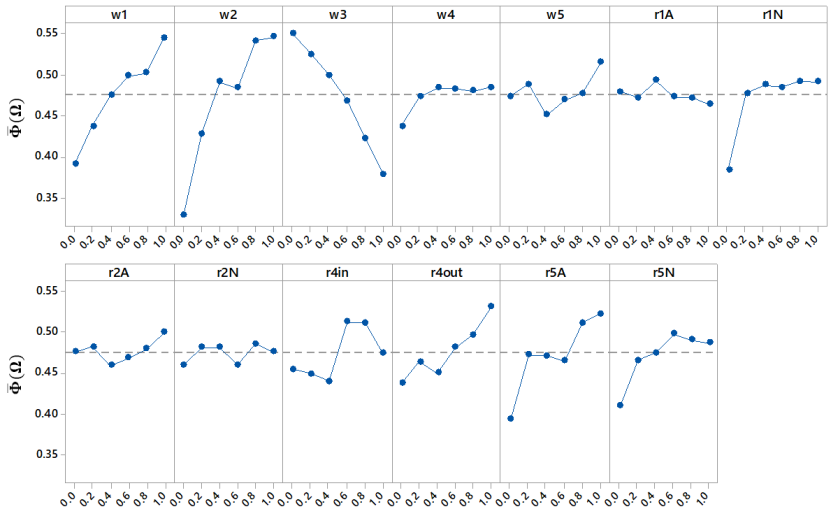


Figure 4.5: Calibration – Main effects plots of the w and r weights influence on the normalized objective function value $\bar{\Phi}(\Omega)$.

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 Φ depends on the particular configuration of the weights, we have used the normalized value $\bar{\Phi}$ for comparisons, defined as follows:

$$\bar{\Phi}(\Omega_f) = \frac{\sum_i w_i R_i(\Omega_f)}{\sum_i w_i}. \tag{4.30}$$

Figure 4.5 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 4.4 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 $r_1^{(A)} = 0$, $r_1^{(N)} = r_2^{(A)} = 1$, $r_2^{(N)} = 0$, $r_4^{(in)} = 0.8$, $r_4^{(out)} = r_5^{(A)} = 1$, $r_5^{(N)} = 0.6$.

Table 4.4: 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			

4.4.2 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 4.4.1. 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 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 Ω_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 4.5 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 [144]. Figure 4.6 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 no-

tice 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 4.7 summarizes the obtained results. Figure 4.7a represents the throughput. It is possible to see that 20000 parts are needed to reach an accurate throughput estimate. Figure 4.7b suggests that for the system time 10000 parts may be sufficient. Figure 4.7c 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 4.7d 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.

4.4.3 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 4.8a 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

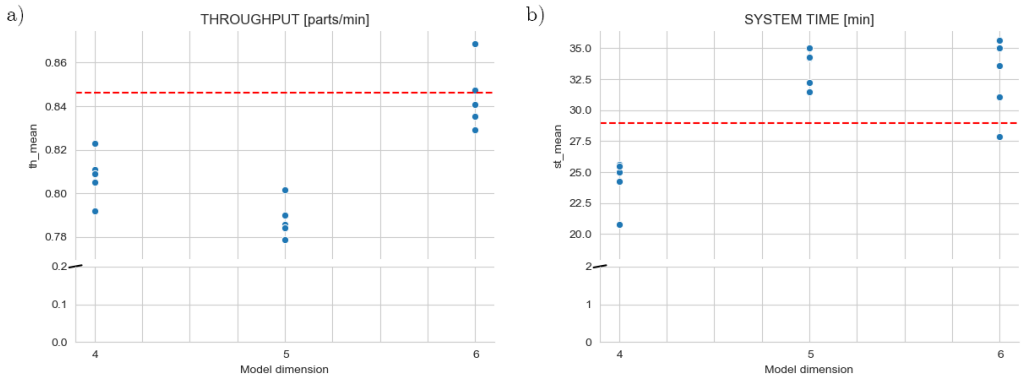


Figure 4.6: Test Case 1 – Results of the experiments in the flow line case (fifth event log): a) throughput; b) system time.

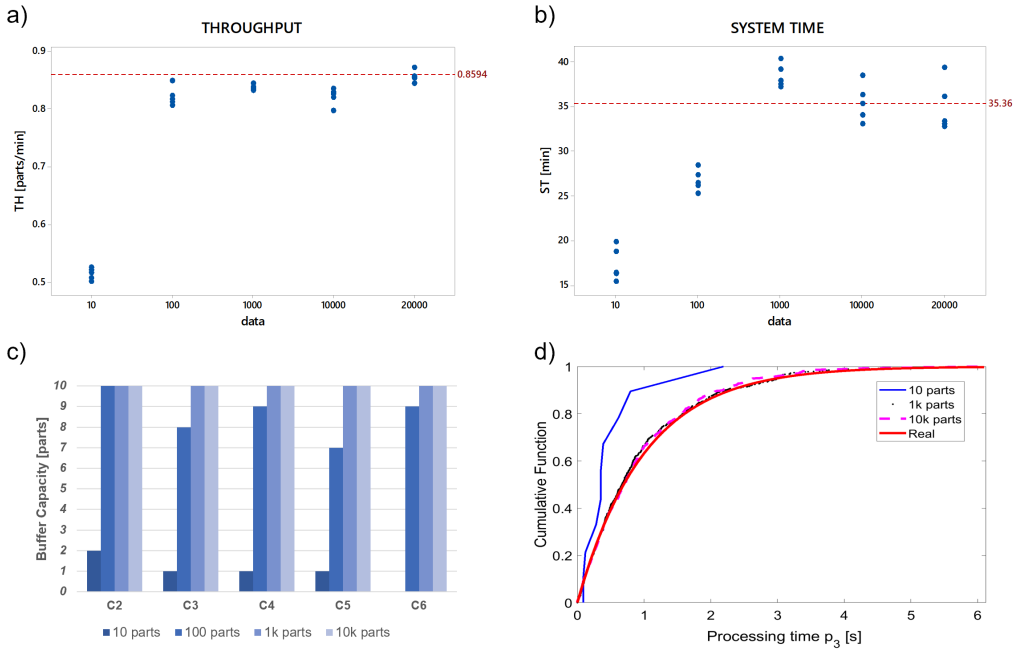


Figure 4.7: Test Case 1 – Results of the experiments with different number of input data points: a) throughput; b) system time, c) buffer capacities, d) cumulative distribution function of processing time p_3.

Table 4.5: 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
TH	4	0.780	0.807	0.831	0.820	0.808	0.835	0.729	0.763	0.810	0.809	0.799
	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
AE	4	0.069	0.021	0.028	0.048	0.038	0.028	0.083	0.055	0.054	0.004	0.043
	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
ST	4	29.0	27.2	24.4	25.4	24.2	26.1	30.9	27.0	26.1	26.6	26.7
	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
AE	4	1.9	6.1	12.5	5.9	4.7	11.8	3.5	7.0	4.8	5.5	6.4
	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

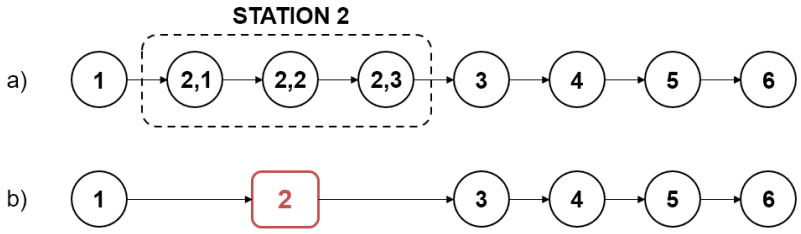


Figure 4.8: Test Case 2 – a) discovered graph for the original system; b) tuned model graph obtained with $U_N^{max} = 6$.

Table 4.6: Test Case 3 – Description of the processes on the electric motor assembly line.

Op.	Process Description	Op.	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

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 4.8b 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 4.4.2 with $U_N^{max} = 6$, hence the same considerations regarding the performance estimation are valid.

4.4.4 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 4.6 summarizes the assembly process phases.

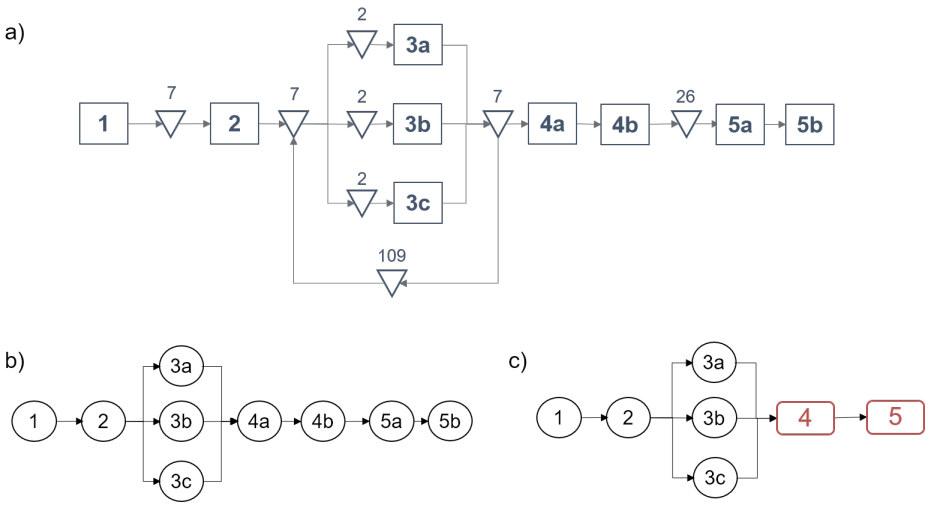


Figure 4.9: Test Case 3 – a) Layout of the stator assembly line (section for which the event log was made available). Stations are represented by squares, while inter-operational buffers are depicted as triangles. – Graph models obtained for the stator assembly line: b) complete model ($|\mathbb{N}| = 9$); c) tuned model ($U_N^{max} = 7$).

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 4.9 shows the portion of the manufacturing system for which the event log was made available by the company, and table 4.7 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 3.2 and 4.3 have been used to build a digital twin of the real system. Figure 4.9b shows the obtained complete model of the assembly line section, while Figure 4.9c 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,

Table 4.7: 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

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.

4.5 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 Industry 4.0 revolution, digitization efforts are being undertaken by both large businesses and small and medium enterprises, and a large number of users will have access to the proposed approach capabilities.

Several limitations still have to be solved. The proposed approach is agnostic with respect to its application 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. In addition, the developed local search algorithm can be enhanced toward computational efficiency and the avoidance of local optima. Notice also that model tuning can also be performed on simulation graphs [145], 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.

Chapter 5

Discovery and Model Generation for Manufacturing Systems with Assembly Operations

The goal of this chapter is to present an approach for the automated development of discrete-event simulation models for manufacturing systems with assembly operations. Its contribution is twofold: (1) it outlines the problem of generating graph models of manufacturing systems with assembly operations, and (2) it introduces a method to infer the locations of assembly operations in manufacturing systems starting from available datasets.

The chapter is organized as follows. Section 5.1 describes the discovery problem. Section 5.2 outlines the proposed method to take into account assembly operations and presents a heuristic algorithm to solve the graph-based mining. Section 5.3 presents the experiments that have been used to investigate the applicability of the approach and the numerical results. Concluding remarks are in section 5.4.

5.1 Problem Description

The model generation method presented in Chapter 4 assumes the availability of an event log with three information types: the timestamp, the activity identifier, and a single part identifier. As a result, the method is limited for more complex types of manufacturing systems, in which different material flows may converge at different locations. A common application that suffers from these features are assembly operations, which are taken as reference within this chapter.

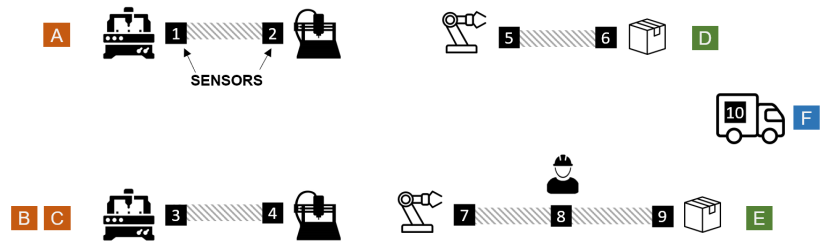


Figure 5.1: Illustrative example – Logical schema of a flow shop manufacturing system.

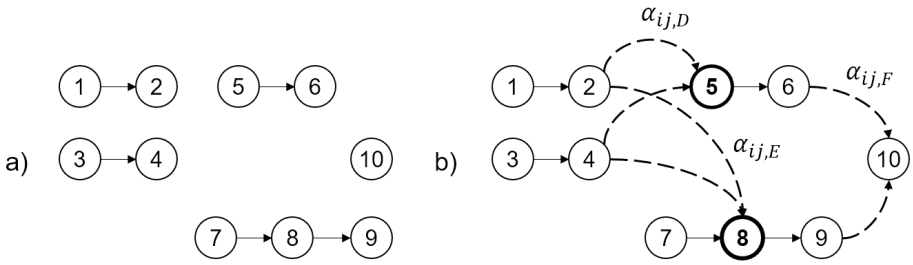


Figure 5.2: Illustrative example – Difference among graph-models generated from the event logs of the flow shop of Figure 5.1: a) traditional mining algorithm; b) assembly-oriented algorithm (bold circles represent assembly stations).

5.1.1 Illustrative Example

Let us consider as illustrative example the manufacturing system depicted in Figure 5.1. The system consists in five sectors. Each sector produces specific part types. Stations 1 and 2 produce components of type A, while stations 3 and 4 are dedicated to components B and C. Parts of type A and B are assembled into D-type products on station 5, while station 8 assembles components of type B and C in products of type E. Products D and E are joined to form a product type F in station 10. Several items of each part type may be produced, and each of them has a unique identifier (e.g., data-matrix quick response code). Each station is equipped with sensors and contributes to the creation of an event log. The model generation procedure listed in section 4.2 produces the graph model shown in Figure 5.2a. From the figure, it can be noticed that the result is a model with sectors treated as separate graph models. This is because model generation is strictly based on the single part identifier hypothesis. Since assembled parts have different identifiers from components, assembly relationships are neglected.

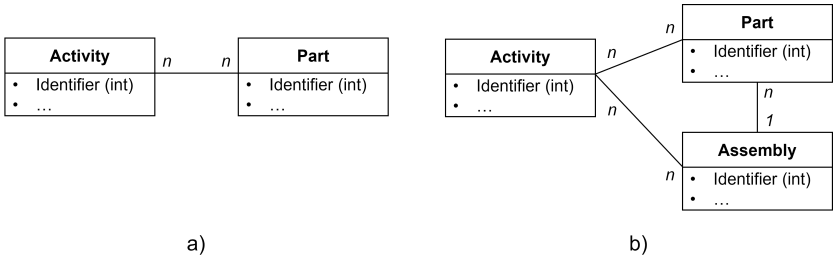


Figure 5.3: Entity relationship diagrams between parts and activities applied to manufacturing processes, relationships between part identifiers and activities: a) traditional process mining approaches, b) object-centric process mining.

5.1.2 Object-Centric Data Perspective

Van der Aalst [146] discussed the gap between real event data and flattened event logs exploited by traditional process mining techniques. The author proposed a new mining paradigm called Object-Centric Process Mining (OCPM), together with a specific logging format. The object-centric log is a collection of events. Each event is related to objects of different types (e.g., tools, packages, shipments). Moreover, basic notations and a baseline discovery approach are presented to facilitate discussion and understanding. Figure 5.3 graphically explains the difference between two mining views. In traditional mining approaches, only one part ID notion is allowed, while OCPM allows to use several part identifiers, thereby representing objects that may refer to multiple items (e.g., a component, an assembled product). Hence, OCPM can be applied to better describe production systems with assembly operations.

An OCPM-compliant representation of a production system dataset is feasible, provided the availability of relational tables connecting the components to the assembled products, which are necessary to define the object relationships. In production environments, such information is typically retrieved from the Bill of Material (BOM). The BOM includes the component-product relationships among all part types. For instance, the BOM of the products produced in the system in Figure 5.1 can be written as $\{D : [A, B]; E : [B, C]; F : [D, E]\}$. Section 5.1.3 elaborates on the types of BOM considered in this work. Table 5.1 shows an extract of the object-centric log in which one component of type A and B are assembled into a part type D. The components are coded 1 and 2, respectively, while the assembled part is coded with 3. In the new log representation, the part identifier column has been substituted by two object columns: (1) components, and (2) assembled products.

Despite the OCPM perspective, literature lacks of approaches on how to effectively use it to generate models of manufacturing systems including non-

linear material flows. This chapter aims at covering this gap with a proposed discovery method and assembly location identification approach.

Table 5.1: Object-centric event log for the assembly of product nr. 3 (type D).

		activity		**objects involved**	
event	**time-stamp**	***name***	***type***	***components***	***assembly***
1	0.00	1	start	1	-
2	0.00	3	start	2	-
3	0.35	1	finish	1	-
4	0.35	2	start	1	-
5	0.45	3	finish	2	-
6	0.45	4	start	2	-
7	0.51	4	finish	2	-
8	0.62	2	finish	1	-
9	0.62	5	start	{1,2}	3
10	0.76	5	finish	{1,2}	3
11	0.76	6	start	{1,2}	3
12	0.88	6	finish	{1,2}	3

5.1.3 Bill of Material Representations

The Bill of Materials are tables widely used and commonly available within tools such as Product Life-Cycle Management or Enterprise Resource Planning. Without loss of generality, in this work we will consider a tree-model of BOM, in which each part type is represented by a node, and it is connected with an arc to the nodes of components with which is assembled. We may distinguish two main cases, which are depicted in Figure 5.4:

- *BOM Type 1: Full part type traceability*. In the first case, each node of the BOM identifies a distinct part. Hence, each assembly step identifies a new part type. In Figure 5.4a, part types A and B are assembled in a component of type C, which is then used together with part type D for the production of the final product, E.
- *BOM Type 2: Partial part type traceability*. In the second case, multiple assembly operations may be performed on a part without changing its part type. Such practice is typical for smaller components or consumables. As example in Figure 5.4b, part types A and B are assembled in

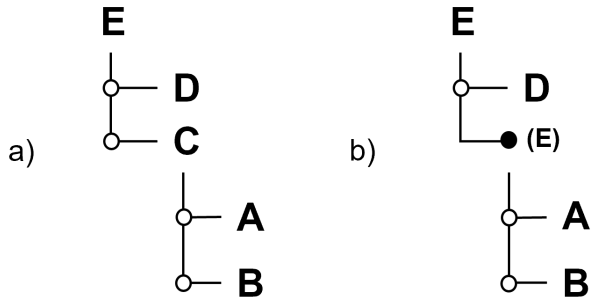


Figure 5.4: Bill of Materials representations: a) type 1: full traceability of product types, b) type 2: partial part type traceability.

a work-in-progress component which is already tagged the same way as the final product, i.e. part type E. A further assembly stage simply adds the sub-component of type D.

In this paper, we use both types of BOM representations as support to define the objects relationships in an event log.

5.1.4 Graph Completion Problem

Let us assume that a model generation procedure as the one described in Chapter 4 has been done. After the system discovery step has been completed, the graph model must be corrected to account for assembly operations. The problem consists in the addition of arcs to the graph model obtained by traditional mining steps (Figure 5.2b). We call such addition *Graph Completion*. Any node in a sub graph can be an assembly node, and each added arc can be dedicated to a product type that is produced in the system. Let us define α_{ijp} as the boolean variable that defines if the directed arc (i, j) is added to the graph for representing the assembly of product type p. The *Graph Completion* corresponds with the addition of elements to the Γ matrix defined in section 4.3. Namely, the complete graph can be defined by a Γ' matrix, in which each element γ'_{ij} is defined as follows:

$$\gamma'_{ij} = \gamma_{ij} + I(\sum_{p} \alpha_{ijp}) \quad \forall i, j \tag{5.1}$$

where $I(x)$ is 1 if $x > 0$, 0 otherwise.

For instance, referring to the illustrative example of Figure 5.1, $\gamma_{1,2} = \gamma_{3,4} = \gamma_{5,6} = \gamma_{7,8} = \gamma_{8,9} = 1$, and $\alpha_{2,5,D} = \alpha_{4,5,D} = \alpha_{2,8,E} = \alpha_{4,8,E} = \alpha_{6,10,F} = \alpha_{9,10,F} = 1$. Thanks to such addition, assembly nodes (i.e., 5, 8, and 10)

can be recognized as such. Figure 5.2b shows the corrected graph after the addition of arcs representative of the assembly of product types D, E, and F.

5.2 Proposed Method

The problem described in section 5.1.4 can be expressed as a matching between components and assembled parts. To this end, the temporal proximity of operations can be exploited as indicator of assembly locations. The general idea is that the added arcs have to be such that the temporal difference between the production of components and assembled products is minimized. Such assumption is suitable for a significant subset of manufacturing systems. For instance, flow lines in which relevant components are produced in a nearby machining area, or group technology manufacturing, in which a set of production cells coordinate to produce the work-in-progress that will converge downstream. A typical example is the automotive sector: doors are usually formed and welded in the same plant as the chassis, and converge to the main assembly line with very limited time differences [147]. Meanwhile, other production systems may not be suitable for a time-proximity-based approach. This is the case for systems relying on batch production (e.g., foundry, molding). In such systems, the completion timestamps of components will be equal for all the components of the batch. Thus, the one-to-one matching loses significance.

In the following, we refer to d_{ca} as the difference between the instant a component c is produced and the moment it is assembled with a product a. Such temporal distance is accountable only for the respective assembly station s. To account for both positive and negative temporal differences, the Mean Square Error (MSE) is identified as proper indicator of mean temporal proximity and is used as objective function of the problem.

5.2.1 Mathematical Formulation

The Graph Completion Problem (GCP) can be expressed with a mathematical programming formulation. Let us define $\mathbb{C}$ as the set of components, $\mathbb{A}$ the set of assembled products, $\mathbb{S}$ the set of stations (i.e. the nodes of the original graph model), and $\mathbb{P}$ the set of part types.

Assumptions. The following assumptions are valid for this problem. Let us assume the manufacturing system is equipped with data collection devices, and events in the system are aggregated in an event log. The log is clean from incomplete traces, and each trace belongs to a part that is either a component or an assembled product. Further, the BOM-related cardinalities are respected: for instance, if one product requires two components, then the

corresponding traces must be in the log. Finally, we assume no batching of production is present and that no preemption is allowed: products and components are processed in order, i.e. the initial sequence is identical to the final sequence.

Parameters. The following parameters can be derived by the joint preprocessing of event log and BOM: τ_c is the time instant at which component c has been produced: from the log, $\tau_c = \max_{n \in \mathbb{N}} t_F(n, c) \, \forall c \in \mathbb{C}$. t_{sa} is the time instant at which assembled product a is produced on station s, 0 otherwise. Namely, $t_{sa} = t_S(s, a) \, if \, \exists \, t_S(s, a) \, \forall a \in \mathbb{A}, \, s \in \mathbb{S}$. ρ_a is the number of components required for the assembly of assembled part a. a_s is 1 if the s-th station is compatible with assembly, 0 otherwise. A station is compatible for assembly if it produces at least one part type with sub-components. q_{ca} is 1 if the c-th component belongs to the BOM of product a, 0 otherwise. p_a is the part type of assembled product a. Finally, M is a very large number.

Decision Variables: x_{cas} is 1 if the c-th component is assigned to assembled part a on station s, 0 otherwise. d_{ca} is the temporal distance between the production of component c and the a-th assembled product.

Graph Completion Problem (GCP):

$$\min \; y \tag{5.2}$$

subject to:

$$y \geq \frac{\sum_a \sum_c d_{ca}^2}{|\mathbb{C}||\mathbb{A}|} \tag{5.3}$$

$$d_{ca} \geq \; t_{sa} - \tau_c - (1 - x_{cas})M \qquad \forall c, a, s \tag{5.4}$$

$$d_{ca} \leq \; t_{sa} - \tau_c + (1 - x_{cas})M \qquad \forall c, a, s \tag{5.5}$$

$$\sum_a \sum_s x_{cas} \leq \; 1 \qquad \forall c \tag{5.6}$$

$$\sum_c \sum_s x_{cas} = \; \rho_a \qquad \forall a \tag{5.7}$$

$$\sum_c \sum_a x_{cas} \leq \; M \, a_s \qquad \forall a \tag{5.8}$$

$$\sum_a \sum_s x_{cas} \leq \; M \, q_{ca} \qquad \forall s \tag{5.9}$$

$$\sum_c x_{cas} \leq \; M \, t_{sa} \qquad \forall a, s \tag{5.10}$$

$$d_{ca} \in \mathbb{R}; x_{cas} \in \{0, 1\}. \tag{5.11}$$

The objective function (5.2) aims at the minimization of the Mean Square Error y, defined by the temporal distance between the components and assembled

products, as stated in constraint (5.3). Constraints (5.4) and (5.5) indicate that the temporal distance d_{ca} is to be accounted only for the component-assembly pairs that are selected. The constraints are activated only for the component-product combination on the selected station, which is indicated by $x_{cas} = 1$. Constraints (5.6) state that each component can be assigned to either one assembled product, on maximum one station. Constraints (5.7) state that each assembled product has to be assigned to a number of components corresponding to the BOM requirements. Constraints (5.8) guarantee that only stations compatible with assembly operations are selected. Constraints (5.9) guarantee that only feasible assembly operations are modeled. Constraints (5.10) ensure that assembly locations are identified in accordance to where the production is recorded in the log, hence that an assembled product is not assigned to stations it did not visit. Constraints (5.11) indicate the nature of the decision variables. In total, GCP counts $|\mathbb{C}||\mathbb{A}|(1 + |\mathbb{S}|)$ variables and $1 + |\mathbb{C}| + 2|\mathbb{A}| + |\mathbb{S}|(|\mathbb{C}||\mathbb{A}| + |\mathbb{A}| + 1)$ constraints.

Retrieving the Graph Model. Once GCP is solved, the solution in terms of graph can be retrieved with a post-processing step. Indeed, the corrections to the graph model defined by the variable α can be derived with the simple procedure listed in Algorithm 7 in Appendix D.

5.2.2 Solution Procedure

In this section, we propose a solution procedure for the problem formulated in section 5.2.1. The GCP is a quadratic mathematical programming problem, which cannot be solved easily with common solvers. The proposed solution method is based on the complete enumeration of feasible assembly stations in the existing graph model. Then, the selection of the station is done following the idea of temporal proximity. Figure 5.5 summarizes the procedure steps.

Step 1: Define the BOM Levels $\mathbb{B}$

Without loss of generality, in the following we assume to analyze a subset of nodes and arcs such that one level of BOM is explored at a time. Indeed, we may separate a generic BOM $\mathbb{B}$ in a collection of levels, $\mathbb{B} = \{\mathbb{B}_1, \ldots, \mathbb{B}_i\}$. For instance, the BOM of the products produced in the system in Figure 5.1 has two levels: the first one is $\mathbb{B}_1 = \{D : [A, B]; E : [B, C]\}$, while the second is $\mathbb{B}_2 = \{F : [D, E]\}$. Hence, the problem is separated in two parts. Firstly, the system without station 10 is analyzed, so that only 1 level of sub-components is present. Then, the system composed by stations 5 to 10 is analyzed. Step 0 is dedicated to the identification and separation of the BOM levels. The

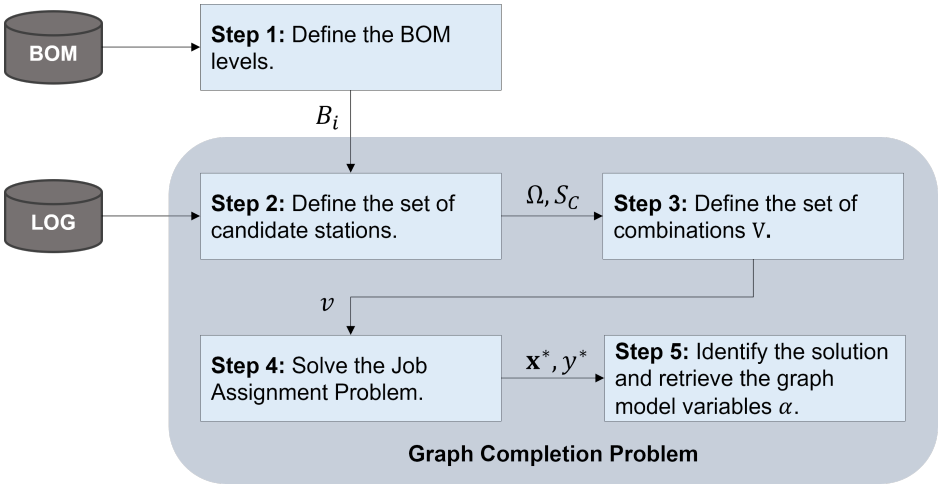


Figure 5.5: Proposed procedure for solving the Graph Completion Problem.

remaining steps take as input one BOM level at a time, hence one GCP is solved for each level.

Step 2: Define the Set of Candidate Stations $\mathbb{S}_C$

In this step, a subset of stations $\mathbb{S}_C \subseteq \mathbb{S}$ is identified. First, the graph-model generation method described in section 4.2.3 is applied. The result is a graph model Ω. Each node in the model is a station $s \in \mathbb{S}$. By exploiting the information from the BOM, it is possible to identify the stations that produce parts with sub-components. All such stations are candidate assembly locations. Let B_i be the i-th level of BOM selected at Step 0. Each station $s \in \mathbb{S}_C$ is a candidate station if it has produced an assembly included in B_i, namely $\exists t_S(s, a)|p_a \in B_i$. This step is performed by Algorithm 5 in Appendix D.

Step 3: Define the Set of Combinations $\mathbb{V}$

In this step, the possible combinations of assembly stations are identified. Starting from the results of model generation, the obtained graph can be divided in a collection of G disjunct subgraphs $\Omega = \{\Omega_1, \ldots, \Omega_G\}$. Since different product types may be produced on different stations, there is no guarantee that they will be produced on nodes from the same disjunct graph. Further, for each product type, only one node is a candidate assembly station. As a result, we may identify multiple combinations of candidate stations. Let us define $\mathbb{V}$ a collection of tuples. Each tuple corresponds to a possible combination of assembly stations for the considered part types. For instance, referring to the

system in Figure 5.1, the subcomponents of product D can either be assembled on station 5 or station 6, while for product E the candidate stations are 7, 8, and 9. Hence, $\mathbb{V} = \{(5,7),(5,8),(5,9),(6,7),(6,8),(6,9)\}$.

Step 4: Components Assignment Sub-Problem

Once a combination of stations has been selected, the remaining part of the problem regards the assignment of the components to the corresponding assemblies. The time proximity is used as indicator of the best matchings. Notice that this problem corresponds to a job assignment problem [148], in which the cost matrix is determined by the square time differences between the production time of components and assemblies.

Let us define $x_{ca}^{(v)}$ as a variable defining the component-assembly assignment in the v-th subproblem combination. $x_{ca}^{(v)}$ is 1 if the c-th component is assigned to the a-th assembled product, 0 otherwise. Further, let us define the following parameters:

- d_{ca} is defined as the square difference of the timestamps of the a-th assembled product and the c-th component. It can be written as follows:

$$d_{ca} = (t_{sa} - \tau_c)^2 \quad \forall c, a \tag{5.12}$$

- ρ_{cp} is equal to 1 if the c-th component is of part type p, 0 otherwise.
- B_{ap} is an integer value representing the number of components of type p required for the fabrication of the assembled part a.

$$\min \; y \tag{5.13}$$

subject to:

$$y \geq \frac{\sum_a \sum_c d_{ca} x_{ca}^{(v)}}{|\mathbb{C}||\mathbb{A}|} \tag{5.14}$$

$$\sum_a x_{ca}^{(v)} \leq \quad 1 \qquad \forall c \tag{5.15}$$

$$\sum_c x_{ca}^{(v)} \rho_{cp} = B_{ap} \qquad \forall a, p \tag{5.16}$$

$$x_{ca}^{(v)} \in \{0,1\}. \tag{5.17}$$

The objective function (5.13) represents the minimization of the time proximity between the components and assemblies through the Mean Square Error y, which is defined in constraint (5.14). Constraints (5.15) guarantees that

each component can be assigned to at most one assembled product. Constraints (5.16) ensure the correct number of components is assigned to each product type, in accordance to the BOM. Constraints (5.17) state the nature of the decision variables. The problem (5.13)-(5.17) has been implemented in ILOG CPLEX v12.6 and it is solved using a PC equipped with an i7-6600U CPU at 2.6 GHz and 16 GB memory.

Step 5: Identify the Solution x^*_{cas}

Finally, among the $|\mathbb{V}|$ combinations, the algorithm selects the one that guarantees the lowest MSE, indicated by v^*. The combination of v^* with the solution of the corresponding assignment problem x^*_{ca} can be used to retrieve the GCP solution x^*_{cas}. Starting from the selected solution, the corresponding nodes and arcs can be added to the graph model. The procedure to complete the graph model starting from a GCP solution x^*_{cas} is listed in Algorithm 7 in Appendix D.

Solution Scores. The algorithm provides information on both the component-product assignment and the location of the assembly operation. Hence, it is important to distinguish on both the capability to provide good assignments and to spot the correct assembly locations. Let us define x^*_{cas} as the solution of the algorithm, and w_{cas} the matrix representing the correct assignments in the system. Therefore, we may define the following indicators:

- The **assignment score** α is an indicator of the goodness of the assignments on the a-th assembled product, as follows:

$$\alpha_a = \frac{\sum_c (\sum_s |x^*_{cas} - w_{cas}|)}{|\mathbb{C}|}. \tag{5.18}$$

  For each assembled product, α_a assumes values in the interval $[0, 1]$, with 1 representing the completely correct assignments. We may additionally define $\bar{\alpha} \in [0, 1]$ as the **average assignment score**:

$$\bar{\alpha} = \frac{\sum_a \alpha_a}{|\mathbb{A}|}. \tag{5.19}$$

- λ_c is the **location assignment indicator** on the c-th assembled product, measuring how well the method has assigned components on the assembly stations:

$$\lambda_c = -\left(\sum_s \left(\frac{|\sum_a x^*_{cas} - w_{cas}|}{2} \right) - 1 \right). \tag{5.20}$$

For each component, λ_c assumes values in the interval $[0, 1]$, with 1 representing the completely correct location assignment. Similarly to α, we may define $\bar{\lambda} \in [0, 1]$ as the **average location score**:

$$\bar{\lambda} = \frac{\sum_c \lambda_c}{|\mathbb{C}|}. \tag{5.21}$$

- Additionally, we may examine the global behavior of the obtained solution. This can be done with a **reproducibility score**, defined as:

$$\Phi = \frac{\sum_c \sum_a \sum_s |x^*_{cas} - w_{cas}|}{|\mathbb{C}||\mathbb{A}||\mathbb{S}|} \tag{5.22}$$

and with a **performance score**, defined as follows:

$$Z = \frac{\sum_c \sum_a \sum_s d_{ca} \left(|x^*_{cas} - w_{cas}|\right)}{|\mathbb{C}||\mathbb{A}||\mathbb{S}|}. \tag{5.23}$$

5.3 Experiments

The solution procedure described in section 5.2.2 has been applied in three test cases:

- **Test Case 1.** A multi-stage production system with five part types, namely two assembled products and three components (single-level BOM of type 1). The scope of this experiment is to verify that the proposed approach can correctly be executed in a system with multiple assembled parts.
- **Test Case 2.** A multi stage production system with four part types, with a multi-level BOM of type 2. The scope is to show the behavior of the proposed method with respect to a multi stage production system, i.e. in which the BOM is composed by multiple levels.
- **Test Case 3.** A real production system with assembly operations within the manufacturing of tier-1 automotive components (single-level BOM of type 1). The scope of this test is twofold: (1) to verify the applicability of the proposed approach in a realistic scenario, and (2) the quantitative observation of the performance obtained with a complete model, to understand the difference with a model generated with the standard approach (Chapter 4).

In the following, we describe the test cases, the experimental settings, and the numerical results.

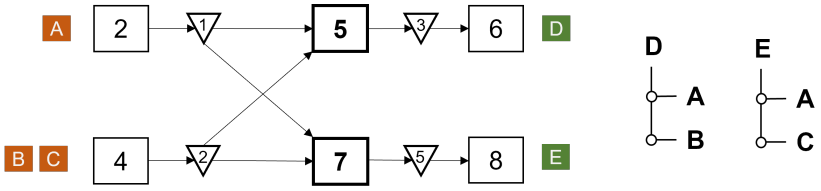


Figure 5.6: Test Case 1 – Flow shop system: stations 2 and 4 produce sub-components, while stations 5 and 7 assemble them into products of type D and E, respectively.

5.3.1 Test Case 1: Flow Shop

The first test case is a flow shop which produces two product types, D and E. The system is depicted in Figure 5.6.

Production System

The system is composed by six stations. Stations 2 and 4 produce subcomponents of type A, B, and C. Station 5 assembles parts of type A and B into products of type D, while station 7 assembles B and C into E. Each station s has a downstream buffer of size H_s and the buffer capacities are equal for all stations: $H_s = 10 \forall s$, except from stations 6 and 8 which produce products D and E as soon as the needed sub-components are available in the corresponding upstream buffers. Both interarrival times and processing times are distributed according to an exponential distribution with mean 1 min.

Experimental Setting

The manufacturing system described in Figure 5.6 has been modeled in Arena Simulation Software. Five event logs have been generated, each corresponding to an independent replication. Each replication represents the production of 1000 final products, 600 of type D and 400 or type E. Since the BOM contains a single level for both part types, one GCP has to be solved over the entire set of nodes. Further, given that two part types are being produced, the set of possible combinations of assembly stations is given by the permutations of 2 elements from the sets $\mathbb{S}_D = \{5, 6\}$ and $\mathbb{S}_E = \{7, 8\}$. Hence, for this case the set of candidate assembly stations are the tuples $\mathbb{V} = \{(5, 7), (5, 8), (6, 7), (6, 8)\}$. For each replication, the GCP solution method described in section 5.2.2 has been used to generate a graph model.

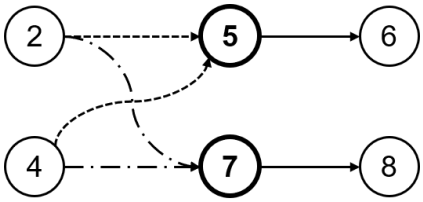


Figure 5.7: Test Case 1 – Graph model obtained by the proposed approach: the dashed lines represent the added arcs.

Results

Table 5.2 summarizes the results obtained for the first test case. From the table, we may notice that in each replication, the correct combination of assembly stations $v^* = (5, 7)$ has been selected as the one guaranteeing the minimum MSE value. Figure 5.7 shows the graph model obtained with the proposed procedure. The arcs $(2, 5)$ and $(4, 5)$ have been added to represent the production of part type D, while the arcs $(2, 7)$ and $(4, 7)$ model the assembly of part type E. The obtained scores are: average assignment score $\bar{\alpha} = 0.949$, average location score: $\bar{\lambda} = 1.0$, objective function score: $Z = 0.931$.

Table 5.2: Test Case 1 - MSE values calculated for each candidate combination of assembly stations v.

	Combinations $\mathbb{V}$:			
Replication	(5,7)	(5,8)	(6,7)	(6,8)
1	**10.706**	25.327	13.721	29.690
2	**18.299**	46.527	21.413	51.048
3	**17.242**	45.971	19.646	49.371
4	**10.316**	32.593	11.865	35.303
5	**14.398**	33.623	16.816	37.206

Note on computation time. The requirements in terms of computation time have been tested by executing the first test case while varying the problem dimension in terms of input components. Figure 5.8 shows the behavior of the average computation time. As visible from the graph, the behavior is exponentially increasing, and it is exceeding 5 minutes with 2000 components.

5.3.2 Test Case 2: Multi-Level BOM

In this case, we analyze a production system with multiple levels of Bill of Material.

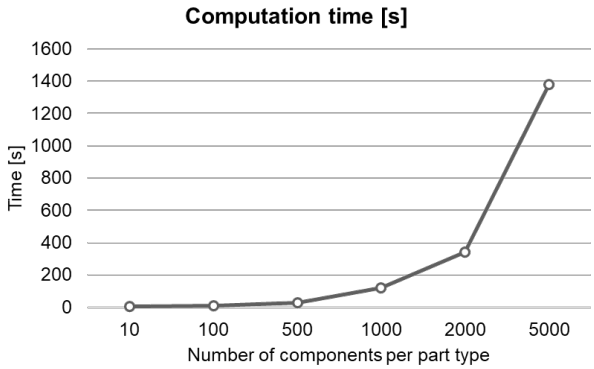


Figure 5.8: Test Case 1 – Average computation time with respect to the number of input components.

Production System

Figure 5.9 shows the structure of production system under study. Station 2 and 4 produce components of type A and B, respectively. Such components are assembled in station 7 into product type D. On station 8, the component type C is assembled on product D. Hence, the Bill of Material is of type 2 and it consists in two separate levels. The first level consists in the assembly of D and C, while the second one describes the assembly of components A and B. Each station has a processing time p_s which is distributed according to an exponential distribution with mean $1\,min$. Inter-operational buffers have 10 slots each.

Experimental Setting

The goal of this test case is to show the behavior of the developed approach in a multi-level system. For the experiments, an event log has been generated with a discrete-event simulation model (i.e. Arena) by the production of 1000 components for each type. The log contains a total of 9169 events.

Results

As first step, subsets of stations are selected based on the levels of BOM. Accordingly, we may select the set of stations for the first level as $\mathbb{N}_1 = \{5, 7, 8\}$, and for the second level, $\mathbb{N}_2 = \{2, 4, 7, 8\}$. Each node set consists in the input of the remaining steps of the method.

Level 1. The set of candidate stations is the subset of $\mathbb{N}_1$ in which the assembled components are produced, hence $\mathbb{S}_1 = \{7, 8\}$. Given that we have a

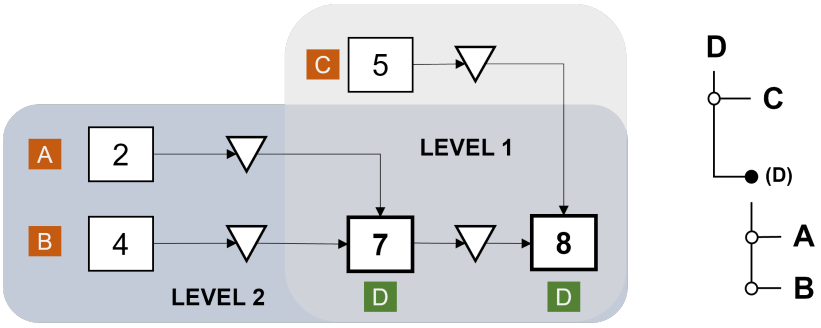


Figure 5.9: Test Case 2 – Production System under study and Bill of Material structure.

single product, both stations are candidate. Hence, the set of possible combinations is $\mathbb{V} = \{(7), (8)\}$. For each combination, the assignment problem defined in section 5.2.2 has been solved. For both sub-problems, a total of 1792 components and 792 assembled products have been assigned one another. The objective function is $z_7 = 0.00381$ and $z_8 = 0.00323$. Accordingly, station 8 is selected as candidate assembly station. Figure 5.10a shows the resulting graph model. The computation times are $63.14\,s$ and $58.23\,s$, respectively. Further, the scores defined in section 5.2.2 have been calculated: average assignment score: $\bar{\alpha} = 0.99833$, average location score: $\bar{\lambda} = 1.0$, objective function score: $Z = 0.9995$.

Level 2. In the second level of the BOM, the components are types A and B and the assembled product is D. The subset of $\mathbb{N}_2$ in which D is produced is $\mathbb{S}_2 = \{7, 8\}$. Again, both stations are candidate assembly locations, and the set of possible combinations is $\mathbb{V} = \{(7), (8)\}$. For both sub-problems, a total of 2000 components and 792 assembled products have been assigned. The objective function is $z_7 = 0.02101$ and $z_8 = 0.02957$. Hence, station 7 is selected as candidate assembly station. Figure 5.10b shows the resulting graph model. The computation times are $65.03\,s$ and $78.29\,s$, respectively. Further, the scores defined in section 5.2.2 have been calculated: average assignment score: $\bar{\alpha} = 0.99802$, average location score: $\bar{\lambda} = 1.0$, objective function score: $Z = 0.928$.

The final complete graph model is visible in Figure 5.10c.

5.3.3 Test Case 3: Real Production System

The problem to be investigated is a multi-cell production system from a tier-1 supplier of road vehicle injectors.

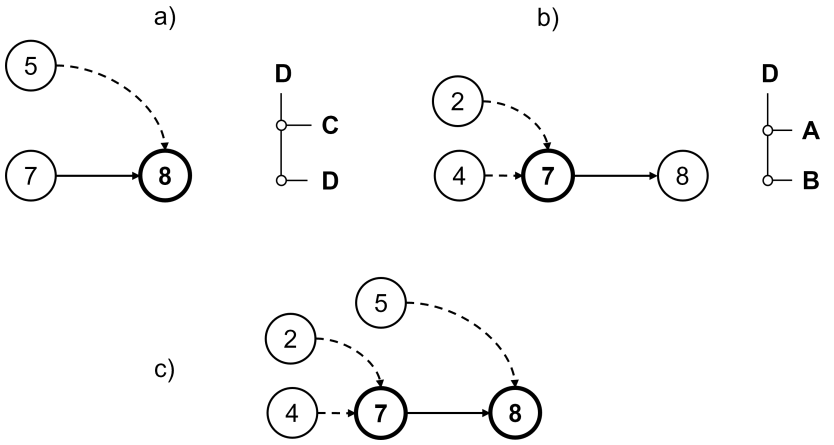


Figure 5.10: Test Case 2 – Resulting graph models depending on the levels of the BOM: a) level 1, b) level 2, c) total model.

Production System

The system consists of nine stations, as shown in Figure 5.11. Stations 1, 2, and 3 are dedicated to the production of sub-components, which are placed in buffers 1 and 3. The remaining stations produce the main part of the injector, and in station 7 the sub-components are assembled. Table 5.3 presents the detailed information about each manufacturing cell (process information is not disclosed for confidentiality reasons). The distributions of the processing times are fitted from collected field data or outputs of a more detailed simulation model of each cell in isolation. Let us denote H_s the buffer capacity of the downstream product store of each cell. The manufacturing system works with a pull production planning using kanban, hence the buffer capacities depend on the specific number of production cards issued in the shop floor. For the sake of simplicity, herewith we do not consider this phenomenon and we assume $H_s = 10 \, \forall s$; we analyze the situation in which injectors can be produced in station 7 whenever all required subcomponents are available in the corresponding buffers.

Experimental Setting

The production system has been modeled in Arena Simulation Software. Five event logs have been generated, each corresponding to an independent replication in which 1000 injectors are produced. For each replication, the procedure described in section 5.2.2 has been applied. Then, the result has been used to build a simulation model. For the sake of simplicity, the graph model

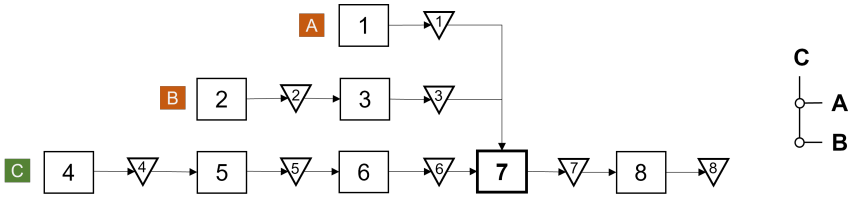


Figure 5.11: Test Case 2 – Automotive tier-1 supplier production system. Squares represent stations and triangles represents product/component stores. Station 7 is the assembly station.

Table 5.3: Test Case 2 – Parameters of the manufacturing system used for the experiments (u is a random number between 0 and 1). The processing times refer to the production of 1000 work-pieces.

Station s	Buffer H_s	Processing Time r_s [s]
1	10	$Logn(4.96, 0.52)$
2	10	$195 + Tria(30, 45, 90) + Logn(4.58, 0.542)$
3	10	$100 + Gamma(0.89, 111.91) + 240$
4	10	$156 + Gamma(1.13, 34.21)$
5	10	$Tria(80, 140, 500)$
6	10	$42 + Logn(1.57, 0.542), if\ u \leq 0.53$
		$62 + Logn(2.10, 0.632), if\ u > 0.53$
7	10	$170 + Gamma(5.94, 6.48)$
8	10	$196 + Gamma(5.75, 5.86), if\ u \leq 0.93$
		$277 + Gamma(0.94, 22.47), if\ u > 0.93$

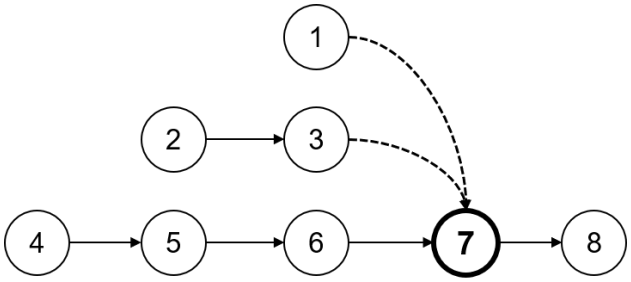


Figure 5.12: Test Case 2 – Graph model obtained by the proposed approach: dashed lines represent the added arcs.

has been converted manually into a simulation model in Arena. The obtained model has been used to generate five independent event logs, and the corresponding system performance in terms of throughput has been calculated.

Results

Table 5.4 lists the MSE value obtained for each replication. Notice that in this case, since a single product type is produced, the assembly station combinations $\mathbb{V}$ are represented by one-element tuples. Also in this case, the results table shows that in each replication the minimum value for MSE corresponds to components-assembly associations on the correct station: $s = 7$. Figure 5.12 represents the graph model that has been obtained. Station 7 is identified as the assembly location and arcs are added to the model accordingly. The obtained scores are: average assignment score $\bar{\alpha} = 0.884$, average location score: $\bar{\lambda} = 1.0$, objective function score: $Z = 0.860$. Such result allows to model correctly the assembly process on station 7, hence enabling the automated generation of the digital replica of the system of Figure 5.11.

Table 5.4: Test Case 2 - MSE values calculated for each candidate assembly station.

	Combinations $\mathbb{V}$:				
Replication	(4)	(5)	(6)	(7)	(8)
1	2041212	1478139	489741	**12833**	14530
2	764825	667953	340785	**147116**	156065
3	457012	453468	439196	**435132**	473526
4	1355473	1022026	441851	**38211**	41126
5	375479	385469	376634	**332724**	348070

Comparison with the Standard Approach

The obtained graph model has been validated by comparing the performance between two cases: (1) model with assembly operations, (2) model obtained with traditional mining method. Specifically, two performance indicators have been chosen for the comparison: IDT_7 is the vector of inter departure times from the assembly station 7: $IDT_7 = t_F(7, i) - t_F(7, i-1)$, and the System Time, defined as the time an assembled part takes to flow in the system: $ST = t_F(8, i) - t_S(4, i)$. Table 5.5 summarizes the results obtained by simulation experiments replicated 5 times each. From such results, we can notice that the exclusion of assembly stations from the representation causes an over estimation of the system performance. Notice that such difference depends on the particular system configuration and parameters.

Table 5.5: Test Case 2 - Performance validation depending on addition of assembly-related arcs out of 1000 samples (CI-HW = 95% Confidence Interval Half Width).

Assembly	Replication	IDT_7 (mean)	CI-HW	ST (mean)	CI-HW
original	-	60.9	4.0	2249.5	24.9
with	1	58.5	3.6	2114.1	21.3
	2	61.7	3.9	2231.1	23.0
	3	62.0	3.8	2251.2	25.6
	4	60.2	3.7	2201.6	25.8
	5	60.3	3.7	2175.3	22.8
without	1	15.7	0.2	680.6	10.2
	2	15.7	0.2	626.7	10.9
	3	15.6	0.2	661.6	10.1
	4	15.6	0.2	627.7	11.2
	5	15.5	0.2	617.7	11.8

5.4 Conclusions

In this chapter, we have introduced an approach that allows for the discovery and modeling of assembly processes within a simulation model generation procedure. An algorithm identifies stations in which components are assembled into final or work-in-progress products, and the corresponding material flows. With this addition, the blocking condition related to the availability of parts can be added to a simulation model, allowing for the proper estimation of

system performances. This work is beneficial for automated model generation techniques. Indeed, the proposed algorithm can be applied online, for instance updated with a fixed frequency, hence it is promising for digital twins dedicated to production planning and control. The developed technique can also be used to investigate disassembly and de-manufacturing operations, since they suffer from the same problem of dimensionality in the part identifiers.

Chapter 6

Final Remarks

This thesis has proposed approaches to build and test real-time decision-support tools based on discrete-event simulation models. This section includes the concluding remarks, with a focus on the impact of the research and its future developments.

6.1 Research Impact

This research develops a model generation and tuning method that allows to obtain digital models starting from the event logs of manufacturing systems. The capability to generate an accurate model in a short time can enable Real-Time Simulation applications. Indeed, the online application of the proposed methodology allows for adapting simulation models to the real system counterpart, potentially at any time a modification occurs. This way, decisions taken online are guaranteed to be referring to the current state of the physical system. The proposed method is beneficial for production planning and control. Manufacturing enterprises can reach a higher production flexibility, together with higher responsiveness to technological changes and market-demand fluctuations. Also model validation activities are positively affected by this research. Indeed, by generating digital models online, it is possible to compare them with the established simulation models and promptly identify misalignments.

With the addition of the graph completion problem and the corresponding solution procedure, the blocking condition related to the availability of component parts can be added to a simulation model, allowing for the proper estimation of the performance of systems with multiple part identifiers and non-linear material flows. The developed technique can also be used to investigate disassembly and de-manufacturing operations, since they suffer from the same

problem of dimensionality in the part identifiers, and the material flow dynamics are comparable to assembly processes.

Finally, the developed laboratory setting allows for testing Real-Time Simulation algorithms, while maintaining the physical dimension and the interplay between a digital model and the real production system. The lab-scale models can be programmed with general programming languages, they are economically sustainable, and there is potentially no limit in the type of system that can be reproduced. The development of such a testbed for Real-Time Simulation is innovative within the research community, and it can bring benefits also to other research groups and disciplines requiring flexible and sustainable test benches for their algorithms. For instance, the lab-scale platform can be used to test re-scheduling algorithms, maintenance policies, and production control rules. Similarly, new devices such as programmable logic controllers could benefit from trials on lab-scale models during their product development phase.

6.2 Future Developments

In the following is presented a selection of insights on how this research can be extended.

- *Extensions of the model generation and tuning techniques*. Future works shall investigate the resilience and outcomes of realistic applications, in which production systems could be composed by hundreds of activities. For this purpose, specific rules may have to be developed. A systematic study on the value of prior information has to be done. Indeed, 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. Another assumption in this work is the inclusion of only material flow split policies, and the consequent identification of routing rules solely based on the node visitation 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.

- *Systematic study on modeling different features and system types*. In this work, we have proposed a single approach to be applied regardless

of the system type. However, such an approach may under-perform in comparison to others when facing particular system features and components. For instance, while the proposed approach performs well with flow lines, it is intuitively sub-optimal when facing flow shops. Indeed, smarter model tuning moves could aggregate parallel activities or remove more than two nodes at a time if the model structure suggests it. 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 depending on the type of manufacturing system.

- *Extensions of the technique for non-linear material flows*. The approach proposed in Chapter 5 assumes perfect traces and the complete availability of BOM data. Realistic datasets are more unreliable, and proper adjustments could be required. Further, the exclusion of batched operations limits the applicability of the approach. In a system which operates in batches, the timestamps of completion for several components are equal. Since the approach relies on temporal proximity, batches would cause several equivalent solutions and a high risk of improper identifications. Assuming the absence of batched operations causes the exclusion of certain types of manufacturing systems, for instance, semi-conductor production systems. In general, this approach is ideal for systems with a certain degree of production coordination, in which components are produced in-house and not stored for long times before their usage. Future developments of this work should also address the optimization of the solution procedure. Indeed, the developed algorithm suffers greatly from the dimensionality of the problem. More research in this direction is needed, for instance with the development of meta heuristic approaches or ad-hoc branch-and-bound algorithms.

- *Development of joint mining approaches*. For instance, reliability models are typically manually developed, and they are utilized in making a large number of decisions, such as spare part purchasing, repair strategies, and maintenance scheduling. The data generated by manufacturing systems can be utilized to automatically infer these controlled actions. The goal of a joint process mining approach is to mine both material-based and information-based data to discover the model of a given production system and its policies (Figure 6.1). The research challenge is to combine the results of different mining algorithms in a unique and consistent framework.

- *Exploitation of model generation techniques for making accurate estimations*. For instance, in manufacturing systems a very important problem

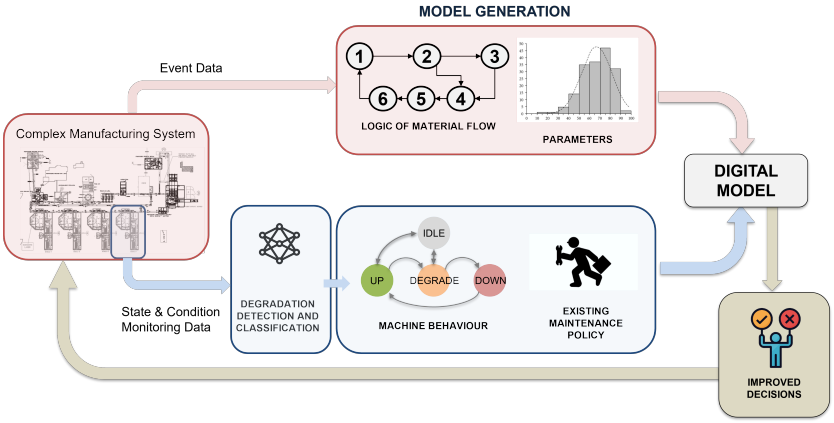


Figure 6.1: Example of a joint model generation framework.

is the identification of the bottleneck. In complex environments, the bottleneck may change during production, for instance due to micro-failures. Such a problem requires a model of the system because it strictly depends on the predicted performance. Hence, model generation can support the online application of techniques for the bottleneck identification.

- *Exploitation of the developed approaches for applications in other sectors*. Service systems (e.g., airports, hospitals) possess material flow dynamics which are comparable to manufacturing environments. Hence, they can be modeled exploiting the same approaches of this thesis. Meanwhile, these systems are very complex and require accurate models for guaranteeing the ability of taking decisions in real-time. The capability of generating digital models online can bring benefits in such application fields.

- *Exploitation of the laboratory setting for application in other fields*. The lab-scale models mimic the material flow with tagged wooden discs. Such devices can model not only manufactured goods but also other items such as people, luggages, vehicles. Hence, the proposed laboratory setting is promising in the application to other fields in which either a new decision-making logic or new technologies have to be tested.

Bibliography

[1] Wil M P Van Der Aalst. *Process mining: Data science in action*. 2016.

[2] Magnus Åkerman. *Implementing shop floor IT for Industry 4.0*. Chalmers Tekniska Hogskola (Sweden), 2018.

[3] Y Rao, F He, X Shao, and C Zhang. On-Line simulation for shop floor control in manufacturing execution system. *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 5315 LNAI(PART 2):141–150, 2008.

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

[5] 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.

[6] W Terkaj, P Gaboardi, C Trevisan, T Tolio, and M Urgo. A digital factory platform for the design of roll shop plants. *CIRP Journal of Manufacturing Science and Technology*, 26:88–93, 2019.

[7] Elisa Negri, Stefano Berardi, Luca Fumagalli, and Marco Macchi. MES-integrated digital twin frameworks. *Journal of Manufacturing Systems*, 56:58–71, jul 2020.

[8] Mengnan Liu, Shuiliang Fang, Huiyue Dong, and Cunzhi Xu. Review of digital twin about concepts, technologies, and industrial applications. *Journal of Manufacturing Systems*, 58:346–361, 2021.

[9] Rakesh Kumar Phanden, Priavrat Sharma, and Anubhav Dubey. A review on simulation in digital twin for aerospace, manufacturing and robotics. *Materials Today: Proceedings*, 38:174–178, 2021.

[10] Qinglin Qi, Fei Tao, Ying Zuo, and Dongming Zhao. Digital twin service towards smart manufacturing. *Procedia Cirp*, 72:237–242, 2018.

[11] S Tavakoli, A Mousavi, and A Komashie. A generic framework for real-time discrete event simulation (DES) modelling. In *Proceedings - Winter Simulation Conference*, pages 1931–1938, Miami, FL, 2008.

[12] Xingzhi Wang, Yuchen Wang, Fei Tao, and Ang Liu. New paradigm of data-driven smart customisation through digital twin. *Journal of manufacturing systems*, 58:270–280, 2021.

[13] Daniel Rossit and Fernando Tohmé. Scheduling research contributions to Smart manufacturing. *Manufacturing Letters*, 15:111–114, 2018.

[14] G Lugaresi and A Matta. Real-time simulation in manufacturing systems: Challenges and research directions. In *Proceedings - Winter Simulation Conference*, 2019.

[15] S. Manivannan and Jerry Banks. Design of a knowledge-based on-line simulation system to control a manufacturing shop floor. *IIE transactions*, 24(3):72–83, 1992.

[16] S Manivannan and Jerry Banks. Real-time control of a manufacturing cell using knowledge-based simulation. In *Winter Simulation Conference Proceedings*, pages 251–260, Phoenix, AZ, USA, 1991. Publ by IEEE, Piscataway, NJ, United States.

[17] S Mirdamadi, F Fontanili, and L Dupont. Discrete event simulation-based real-time shop floor control. *Proceedings of the 2007 European Conference on Modelling and Simulation*, pages 235–240, 2007.

[18] P Mullarkey, S Gavirneni, and D J Morrice. Dynamic output analysis for simulations of manufacturing environments. *Winter Simulation Conference Proceedings*, 2:1290–1296, 2000.

[19] C A Rabbath, M Abdoune, and J Belanger. Effective real-time simulations of event-based systems. *Winter Simulation Conference Proceedings*, 1:232–238, 2000.

[20] K Lee and P A Fishwick. Building a model for real-time simulation. *Future Generation Computer Systems*, 17(5):585–600, 2001.

[21] L Monostori, B Kádár, A Pfeiffer, and D Karnok. Solution Approaches to Real-time Control of Customized Mass Production. *CIRP Annals - Manufacturing Technology*, 56(1):431–434, 2007.

[22] A Mousavi and H R A Siervo. Automatic translation of plant data into management performance metrics: a case for real-time and predictive production control. *International Journal of Production Research*, 55(17):4862–4877, 2017.

[23] GE-Automation. GE Automation: the rise of industrial big data. `http://www.geautomation.com/download/rise-industrial-big-data`, 2016. Accessed April 13th, 2018.

[24] L D Xu, E L Xu, and L Li. Industry 4.0: State of the art and future trends. *International Journal of Production Research*, 56(8):2941–2962, 2018.

[25] Sigrid Wenzel and Tim Peter. Transformation of Real-time Reporting Event Data to Long-term Simulation Models. *Simulation in Produktion und Logistik 2017*, page 393, 2017.

[26] N Robertson and T Perera. Automated data collection for simulation? *Simulation Practice and Theory*, 9(6-8):349–364, 2002.

[27] A Hanisch, J Tolujew, and T Schulze. Initialization of online simulation models. In *Proceedings - Winter Simulation Conference*, volume 2005, pages 1795–1803, Orlando, FL, 2005.

[28] Matthias Blum and Guenther Schuh. Towards a Data-oriented Optimization of Manufacturing Processes. In *Proceedings of the 19th International Conference on Enterprise Information Systems, Porto, Portugal*, pages 26–29, 2017.

[29] Masaki Kitazawa, Satoshi Takahashi, Toru B Takahashi, Atsushi Yoshikawa, and Takao Terano. Real Time Workers' Behavior Analyzing System for Productivity Measurement Using Wearable Sensor. *SICE Journal of Control, Measurement, and System Integration*, 10(6):536–543, 2017.

[30] Mohammed Sadiq Altaf, Hexu Liu, Mohamed Al-Hussein, and Haitao Yu. Online simulation modeling of prefabricated wall panel production using RFID system. In *Proceedings of the 2015 Winter Simulation Conference*, pages 3379–3390. IEEE Press, 2015.

[31] H Luo, J Fang, and G Q Huang. Real-time scheduling for hybrid flowshop in ubiquitous manufacturing environment. *Computers and Industrial Engineering*, 84:12–23, 2015.

[32] Olivier Cardin and Pierre Castagna. Proactive production activity control by online simulation. *International Journal of Simulation and Process Modelling*, 6(3):177–186, 2011.

[33] Young Jun Son and Richard A Wysk. Automatic simulation model generation for simulation-based, real-time shop floor control. *Computers in Industry*, 45(3):291–308, jul 2001.

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

[35] Florian Biesinger, Davis Meike, Benedikt Kraß, and Michael 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*, 79:355–360, 2019.

[36] Gerardo Santillan Martinez, Seppo Sierla, Tommi Karhela, and Valeriy 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, IECON 2018*, Proceedings of the Annual Conference of the IEEE Industrial Electronics Society, pages 3084–3089, United States, 2018. Institute of Electrical and Electronics Engineers.

[37] Mutsumi Fujihara and Kiyoshi Yoneda. Simulation through explicit state description and its application to semiconductor fab operation. In *Proceedings of the 24th conference on Winter simulation*, pages 899–907. ACM, 1992.

[38] Averill M Law. How to build valid and credible simulation models. In *2019 Winter Simulation Conference (WSC)*, pages 1402–1414. IEEE, 2019.

[39] Osman Balci. Principles and techniques of simulation validation, verification, and testing. In *Proceedings of the 27th conference on Winter simulation*, pages 147–154, 1995.

[40] W J Davis. On-line simulation: Need and evolving research requirements. *Handbook of Simulation*, pages 465–516, 1998.

[41] Adnan Khan, Martin Dahl, Petter Falkman, and Martin Fabian. Digital Twin for Legacy Systems: Simulation Model Testing and Validation. In

2018 IEEE 14th International Conference on Automation Science and Engineering (CASE), pages 421–426. IEEE, 2018.

[42] Osman Balci. Validation, verification, and testing techniques throughout the life cycle of a simulation study. *Annals of operations research*, 53(1):121–173, 1994.

[43] Keith Loague and Richard E Green. Statistical and graphical methods for evaluating solute transport models: overview and application. *Journal of contaminant hydrology*, 7(1-2):51–73, 1991.

[44] D G Mayer and D G Butler. Statistical validation. *Ecological modelling*, 68(1-2):21–32, 1993.

[45] Ramesh Rebba, Shuping Huang, Yongming Liu, and Sankaran Mahadevan. Statistical validation of simulation models. *International Journal of Materials and Product Technology*, 25(1-3):164–181, 2006.

[46] Vance W Berger and YanYan Zhou. Kolmogorov–smirnov test: Overview. *Wiley statsref: Statistics reference online*, 2014.

[47] Jack P C Kleijnen, Bert Bettonvil, and Willem Van Groenendaal. Validation of trace-driven simulation models: a novel regression test. *Management Science*, 44(6):812–819, 1998.

[48] Robert G Sargent. Verification and validation of simulation models. *Journal of simulation*, 7(1):12–24, 2013.

[49] Giovanni Lugaresi, Gianluca Aglio, Federico Folgheraiter, and Andrea Matta. Real-time Validation of Digital Models for Manufacturing Systems: a Novel Signal-processing-based Approach. Technical report, 2019.

[50] David Katz and S Manivannan. Exception management on a shop floor using online simulation. In *Proceedings of 1993 Winter Simulation Conference-(WSC'93)*, pages 888–896. IEEE, 1993.

[51] Gerardo Santillan Martinez, Tommi Karhela, Reino Ruusu, Tuomas Lackman, and Valeriy Vyatkin. Towards a systematic path for dynamic simulation to plant operation: OPC UA-enabled model adaptation method for tracking simulation. In *IECON 2017-43rd Annual Conference of the IEEE Industrial Electronics Society*, pages 5503–5508. IEEE, 2017.

[52] J Banks, J S Carson, and B L Nelson. Discrete-Event System Simulation. *Prentice Hall*, 1996.

[53] B Kadar, A Lengyel, L Monostori, Y Suginishi, A Pfeiffer, and Y Nonaka. Enhanced control of complex production structures by tight coupling of the digital and the physical worlds. *CIRP Annals - Manufacturing Technology*, 59(1):437–440, 2010.

[54] Behrang Ashtari Talkhestani, Nasser Jazdi, Wolfgang Schloegl, and Michael Weyrich. Consistency check to synchronize the Digital Twin of manufacturing automation based on anchor points. *Proc. CIRP*, 72:159–164, 2018.

[55] Sören Bergmann, Sören Stelzer, and Steffen Straßburger. Initialization of simulation models using CMSD. In *Proceedings of the 2011 Winter Simulation Conference (WSC)*, pages 2223–2234. IEEE, 2011.

[56] Rafal Cupek, Adam Ziebinski, Lukasz Huczala, and Huseyin Erdogan. Agent-based manufacturing execution systems for short-series production scheduling. *Computers in Industry*, 82:245–258, 2016.

[57] L Damiani, M Demartini, P Giribone, M Maggiani, R Revetria, and F Tonelli. Simulation and digital twin based design of a production line: A case study. In *Lecture Notes in Engineering and Computer Science*, volume 2, 2018.

[58] M M Nasiri, R Yazdanparast, and F Jolai. A simulation optimisation approach for real-time scheduling in an open shop environment using a composite dispatching rule. *International Journal of Computer Integrated Manufacturing*, 30(12):1239–1252, 2017.

[59] András Pfeiffer, B Kádár, L Monostori, and Zoltán Vén. Situation detection in production control by applying on-line simulation. In *5th International Conference on Digital Enterprise Technology, DET 2008*, pages 225–241. Publibook, 2008.

[60] Pradeep Suresh, John M Wassick, and Jeff Ferrio. Real time performance measurement for batch chemical plants. In *Proceedings of the Winter Simulation Conference*, pages 2330–2340. Winter Simulation Conference, 2011.

[61] Iracyanne Retto Uhlmann and Enzo Morosini Frazzon. Production rescheduling review: Opportunities for industrial integration and practical applications. *Journal of manufacturing systems*, 49:186–193, 2018.

[62] Guilherme E Vieira, Jeffrey W Herrmann, and Edward Lin. Rescheduling manufacturing systems: a framework of strategies, policies, and methods. *Journal of scheduling*, 6(1):39–62, 2003.

[63] S Bohlmann, M Becker, S Balci, H Szczerbicka, and E Hund. Online simulation based decision support system for resource failure management in multi-site production environments. In *2013 IEEE 18th Conference on Emerging Technologies Factory Automation (ETFA)*, pages 1–4, 2013.

[64] Olivier Cardin and Pierre Castagna. Myopia of service oriented manufacturing systems: benefits of data centralization with a discrete-event observer. In *Service Orientation in Holonic and Multi-Agent Manufacturing Control*, pages 197–210. Springer, 2012.

[65] Catherine M Harmonosky, Robert H Farr, and Ming-Chuan Ni. Selective rerouting using simulated steady state system data. In Withers D H Nelson B L Andradottir S. Healy K.J., editor, *Winter Simulation Conference Proceedings*, pages 1293–1298, Atlanta, GA, USA, 1997. IEEE, Piscataway, NJ, United States.

[66] J M Framinan, P Perez-Gonzalez, and V.F.-V. Escudero. The value of real-time data in stochastic flowshop scheduling: A simulation study for makespan. In Chan V., editor, *Proceedings - Winter Simulation Conference*, pages 3299–3310. Institute of Electrical and Electronics Engineers Inc., 2017.

[67] Heiko Aydt, Wentong Cai, and Stephen John Turner. Dynamic specialization for symbiotic simulation-based operational decision support using the evolutionary computing modelling language (ECML). *Journal of Simulation*, 8(2):105–114, 2014.

[68] Malcolm Yoke Hean Low, Kong Wei Lye, Peter Lendermann, Stephen John Turner, Reman Tat Wee Chim, and Surya Hadisaputra Leo. An Agent-based Approach for Managing Symbiotic Simulation of Semiconductor Assembly and Test Operation. In *Proceedings of the Fourth International Joint Conference on Autonomous Agents and Multiagent Systems*, AAMAS '05, pages 85–92, New York, NY, USA, 2005. ACM.

[69] A Gunasekaran, B K Rai, and M Griffin. Resilience and competitiveness of small and medium size enterprises: An empirical research. *International Journal of Production Research*, 49(18):5489–5509, 2011.

[70] 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.

[71] 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.

[72] 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.

[73] 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.

[74] W Van Der Aalst, T Weijters, and L Maruster. Workflow mining: Discovering process models from event logs. *IEEE Transactions on Knowledge and Data Engineering*, 16(9):1128–1142, 2004.

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

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

[77] I. Sitova and J. Pecerska. Process Data Analysis Using Visual Analytics and Process Mining Techniques. In *2020 61st International Scientific Conference on Information Technology and Management Science of Riga Technical University, ITMS 2020 - Proceedings*, 2020.

[78] 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.

[79] W. Intayoad and T. Becker. Exploring the Relationship between Business Processes and Contextual Information in Manufacturing and Logistics Based on Event Logs. In *Procedia CIRP*, volume 72, pages 557–562, 2018.

[80] E. Ruschel, E.D.F. Rocha Loures, and E.A.P. Santos. Performance analysis and time prediction in manufacturing systems. *Computers and Industrial Engineering*, 151, 2021.

[81] J. Park, D. Lee, and J. Zhu. An integrated approach for ship block manufacturing process performance evaluation: Case from a Korean shipbuilding company. *International Journal of Production Economics*, 156:214–222, 2014.

[82] William W Cooper, Lawrence M Seiford, and Kaoru Tone. Data envelopment analysis. *Handbook on data envelopment analysis*, pages 1–40, 2000.

[83] C.K.H. Lee, G.T.S. Ho, K.L. Choy, and G.K.H. Pang. A RFID-based recursive process mining system for quality assurance in the garment industry. *International Journal of Production Research*, 52(14):4216–4238, 2014.

[84] 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.

[85] 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.

[86] Ralph Mueller, Christos Alexopoulos, and Leon F McGinnis. Automatic generation of simulation models for semiconductor manufacturing. In *2007 Winter Simulation Conference*, WSC '07, pages 648–657, Piscataway, NJ, USA, 2007. IEEE, IEEE Press.

[87] 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.

[88] A. Rozinat, R.S. Mans, M. Song, and W.M.P. van der Aalst. Discovering simulation models. *Information Systems*, 34(3):305–327, may 2009.

[89] 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.

[90] Giovanni Lugaresi and Andrea Matta. Automated manufacturing system discovery and digital twin generation. *Journal of Manufacturing Systems*, 59:51–66, 2021.

[91] 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.

[92] A. Farooqui, K. Bengtsson, P. Falkman, and M. 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.

[93] 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.

[94] Niels Martin, Benoit 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.

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

[96] 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, jun 2015.

[97] Niels Martin, Frank Bax, Benoit Depaire, and An Caris. Retrieving resource availability insights from event logs. In *Proceedings - 2016 IEEE 20th International Enterprise Distributed Object Computing Conference, EDOC 2016*, 2016.

[98] 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, 2017.

[99] 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.

[100] H.G. Maiorki, E.A.P. Santos, and E.F.R. De Loures. Multi-level log XES format: A RAMI4.0 perspective. In *Conference Proceedings - IEEE International Conference on Systems, Man and Cybernetics*, volume 2019-Octob, pages 1614–1620, 2019.

[101] H. Jo, S.D. Noh, and Y. Cho. An agile operations management system for green factory. *International Journal of Precision Engineering and Manufacturing - Green Technology*, 1(2):131–143, 2014.

[102] G. Schuh, A. Gützlaff, S. Schmitz, and W.M.P. van der Aalst. Data-based description of process performance in end-to-end order processing. *CIRP Annals*, 69(1):381–384, 2020.

[103] C. Fleig, D. Augenstein, and A. Maedche. Process mining for business process standardization in ERP implementation projects – An SAP S/4 HANA case study from manufacturing. In *CEUR Workshop Proceedings*, volume 2196, pages 149–155, 2018.

[104] Z. Toosinezhad, D. Fahland, O. Köroglu, and W.M.P. Van Der Aalst. Detecting system-level behavior leading to dynamic bottlenecks. In *Proceedings - 2020 2nd International Conference on Process Mining, ICPM 2020*, pages 17–24, 2020.

[105] M. Cho, G. Park, M. Song, J. Lee, B. Lee, and E. Kum. Discovery of Resource-Oriented Transition Systems for Yield Enhancement in Semiconductor Manufacturing. *IEEE Transactions on Semiconductor Manufacturing*, 34(1):17–24, 2021.

[106] T.B.H. Tu and M. Song. Analysis and prediction cost of manufacturing process based on process mining. In *ICIMSA 2016 - 2016 3rd International Conference on Industrial Engineering, Management Science and Applications*, 2016.

[107] E. Ruschel, E.A.P. Santos, and E.F.R. Loures. Establishment of maintenance inspection intervals: an application of process mining techniques in manufacturing. *Journal of Intelligent Manufacturing*, 31(1):53–72, 2020.

[108] Rolando J Kurscheidt, Eduardo A P Santos, Eduardo de FR Loures, Jose E Pecora Jr, and Jose M A P Cestari. A Methodology for Discovering Bayesian Networks Based on Process Mining. In *IIE Annual Conference. Proceedings*, page 2322. Institute of Industrial and Systems Engineers (IISE), 2015.

[109] G. Dörgo, K. Varga, M. Haragovics, T. Szabó, and J. Abonyi. Towards operator 4.0, increasing production efficiency and reducing operator workload by process mining of alarm data. *Chemical Engineering Transactions*, 70:829–834, 2018.

[110] S. Knoch, S. Ponpathirkoottam, P. Fettke, and P. Loos. *Technology-enhanced process elicitation of worker activities in manufacturing*, volume 308. 2018.

[111] F. Mannhardt, R. Bovo, M.F. Oliveira, and S. Julier. *A taxonomy for combining activity recognition and process discovery in industrial environments*, volume 11315 LNCS. 2018.

[112] H. Yang, M. Park, M. Cho, M. Song, and S. Kim. A system architecture for manufacturing process analysis based on big data and process mining techniques. In *Proceedings - 2014 IEEE International Conference on Big Data, IEEE Big Data 2014*, pages 1024–1029, 2014.

[113] S. Ferilli and S. Angelastro. Activity prediction in process mining using the WoMan framework. *Journal of Intelligent Information Systems*, 53(1):93–112, 2019.

[114] C.M. Flath and N. Stein. Towards a data science toolbox for industrial analytics applications. *Computers in Industry*, 94:16–25, 2018.

[115] C. Ortmeier, N. Henningsen, A. Langer, A. Reiswich, A. Karl, and C. Herrmann. Framework for the integration of Process Mining into Life Cycle Assessment. In *Procedia CIRP*, volume 98, pages 163–168, 2021.

[116] D. Dakic, S. Sladojevic, T. Lolic, and D. Stefanovic. Process mining possibilities and challenges: A case study. In *SISY 2019 - IEEE 17th International Symposium on Intelligent Systems and Informatics, Proceedings*, pages 161–166, 2019.

[117] O. Dogan and O.F. Gurcan. Data perspective of lean six sigma in industry 4.0 era: A guide to improve quality. In *Proceedings of the International Conference on Industrial Engineering and Operations Management*, volume 2018, pages 943–953, 2018.

[118] G. Meyer, G. Adomavicius, P.E. Johnson, M. Elidrisi, W.A. Rush, J.A.M. Sperl-Hillen, and P.J. O'Connor. A machine learning approach to improving dynamic decision making. *Information Systems Research*, 25(2):239–263, 2014.

[119] F. Stertz and S. Rinderle-Ma. *Detecting and identifying data drifts in process event streams based on process histories*, volume 350. 2019.

[120] Z. Paszkiewicz. Process mining techniques in conformance testing of inventory processes: An industrial application. In *Lecture Notes in Business Information Processing*, volume 160, pages 302–313. 2013.

[121] S. Saraeian and B. Shirazi. Process mining-based anomaly detection of additive manufacturing process activities using a game theory modeling approach. *Computers and Industrial Engineering*, 146, 2020.

[122] 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.

[123] Christian W Günther and Wil M P 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.

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

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

[126] K.M. Rashid and J. Louis. Process Discovery and Conformance Checking in Modular Construction Using RFID and Process Mining. In *Construction Research Congress 2020: Computer Applications - Selected Papers from the Construction Research Congress 2020*, pages 640–648, 2020.

[127] Dino Knoll, Gunther Reinhart, and Marco Prüglmeier. Enabling value stream mapping for internal logistics using multidimensional process mining. *Expert Systems with Applications*, 124:130–142, jun 2019.

[128] Wil M P van der Aalst. Mining Additional Perspectives. In *Process Mining*, pages 215–240. Springer, 2011.

[129] Nassim Nicholas Taleb. *The black swan: The impact of the highly improbable*, volume 2. Random house, 2007.

[130] Jay B Barney and Delwyn N Clark. *Resource-based theory: Creating and sustaining competitive advantage*. Oxford University Press on Demand, 2007.

[131] Mohsen Moghaddam, Marissa N Cadavid, C Robert Kenley, and Abhijit V Deshmukh. Reference architectures for smart manufacturing: A critical review. *Journal of manufacturing systems*, 49:215–225, 2018.

[132] Young Jae Jang and Vina Sari Yosephine. LEGO robotics based project for industrial engineering education. *International Journal of Engineering Education*, 32(3):1268–1278, 2016.

[133] Giovanni Lugaresi, Davide Travaglini, and Andrea Matta. A LEGO ® MANUFACTURING SYSTEM AS DEMONSTRATOR FOR A REAL-TIME SIMULATION PROOF OF CONCEPT. Technical report, 2019.

[134] *ev3dev*, accessed July 1, 2020. `http://www.ev3dev.org`.

[135] Jean-Paul Arnaout. Rescheduling of parallel machines with stochastic processing and setup times. *Journal of Manufacturing Systems*, 33(3):376–384, jul 2014.

[136] Jean-Paul Arnaout. Heuristics for the maximization of operating rooms utilization using simulation. *Simulation*, 86(8-9):573–583, 2010.

[137] Stanley R Sadin, Frederick P Povinelli, and Robert Rosen. The NASA technology push towards future space mission systems. In *Space and Humanity*, pages 73–77. Elsevier, 1989.

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

[139] Bart L Maccarthy and Flavio C F 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.

[140] 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*, page 1, 2020.

[141] Willibrordus Martinus Pancratius van der Aalst, K M Van Hee, and G J 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.

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

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

[144] 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.

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

[146] Wil M P van der Aalst. Object-centric process mining: Dealing with divergence and convergence in event data. In *International Conference on Software Engineering and Formal Methods*, pages 3–25. Springer, 2019.

[147] Merih Seran Uysal, Sebastiaan J van Zelst, Tobias Brockhoff, Anahita Farhang, Mahsa Pourbafrani Ghahfarokhi, Ruben Schumacher, Sebastian Junglas, Günther Schuh, and Wil M P van der Aalst. Process Mining for Production Processes in the Automotive Industry.

[148] David W Pentico. Assignment problems: A golden anniversary survey. *European Journal of Operational Research*, 176(2):774–793, 2007.

[149] John C Mankins and Others. Technology readiness levels. *White Paper, April*, 6(1995):1995, 1995.

[150] Mihály Héder. From NASA to EU: the evolution of the TRL scale in Public Sector Innovation. *The Innovation Journal*, 22(2):1–23, 2017.

[151] William L Nolte. TRL calculator. In *AFRL Assessing Technology Readiness Development Seminar*, 2005.

Appendix A

Assessment of the Technology Readiness Level (TRL)

Let us consider the case in which a new RTS-based technology has to be tested. It is reasonable to reflect on the advantages of exploiting a lab-scale environment such as the one described in section 3.2.1. An established way to assess the maturity of a technology is the Technology Readiness Level (TRL). The TRL is an nine-level indicator originally developed by NASA [137] which has been used extensively in the last 40 years to assess the maturity of a technology in the aerospace sector [149]. Nowadays, the TRL is also used to assign grants and evaluate research proposals such as European Horizon 2020 [150]. It is assumed that a TRL level cannot be reached before the previous levels are obtained. The first three levels refer to the observation of basic principles, the formulation of technology concepts and proof-of-concepts. TRL 4 is achieved if the proposed technology is validated in a laboratory. In such a setting, the operating environment is not realistic. For instance, consider load cells tested with lower weights or an actuator used to provide smaller displacements than in the intended use cases. Since the proposed lab-scale models are compatible with the ISA95 industrial standard and can incorporate IoT components, we infer that the proposed environment allows specific component types to advance their own TRL to a level higher than 4. In order to clarify our assumption, we have exploited a questionnaire developed by the US Air Force Research Laboratory (AFRL) [151]. The questionnaire consists in 274 questions. If a newly-developed technology can satisfy all the questions related to a TRL, it can be considered at that readiness level. Table A.1 summarizes the questions that we believe could be answered positively by using the lab-scale models. Specifically, we selected three types of components that could be evaluated exploiting the proposed testing environment: (1) a software component, (2) a hardware device (e.g., gateway, sensor, PLC), and (3)

a method (e.g., a scheduling algorithm). Depending on the component type, we have identified the achievable TRL level. As a result, the lab-scale setting can grant each tested component to advance on its own TRL. It is worth to notice that this analysis cannot assess the TRL of the entire manufacturing system.

A.1 Software Component

If the software component is in the loop with the lab-scale model, the laboratory can be considered a relevant environment. Indeed, if the actuators and sensors can provide the same functionality of the real system (e.g., changing routes, stopping part flows) it is possible to reproduce material flows representative of a real system behavior. If the steady state performance of the model is comparable with a real system, it is possible to design tests to address specific factory requirements such as quality control frequency or production rate. Further, the hardware processors for the lab-scale environment can be the same one as the real environment (e.g., gateways and PLCs), with no specific limitations concerning the integration among software tool components. The interfaces can be described with reference to real components. For example, the cyber-physical architecture can incorporate the same data formats required by a specific PLC model. Moreover, the whole architecture can be verified through the established communication channels, for instance by testing the conformance to priority rules among different hierarchical levels.

A.2 Hardware Component

In case of a hardware component such as a PLC or gateway, if the test scope is production planning and control and the implemented functionalities of the lab-scale model correspond to large scale systems (e.g., flow control) the laboratory setting can be considered a field environment equivalent. Also, anomalous conditions can be designed properly in the lab-scale model to be representative of a realistic situation. For instance, an anomalous flow in a real plant could be simply recorded and replicated in the lab-scale model. The hardware can be tested through the connection between the lab-scale architecture levels and the field devices (e.g., serial ports). Additionally, interfaces can be tested on the lab scale models the same way as in a realistic environment (e.g., through dashboard visualizations and database connections).

A.3 Method

In case the intended technology is a method for production planning and control (e.g. scheduling algorithm), if the stream of parts replicated in the model is realistic (e.g., steady state performances are comparable) we may consider the lab-scale environment as representative of a field environment. Indeed, the algorithm uses data coming from the field sources as inputs and outputs, regardless of the production system mechanics. Hence, if the installed devices are representative of the production system of interest, logical functions can be tested on the lab-scale models with the same expected outputs. For instance, a scheduling algorithm can be tested against realistic disruptions replicated in the physical model.

	TRL	Question
Software	5	System software architecture established
		External interfaces described as to source, format, structure, content, and method of support
		Interfaces between components/subsystems are realistic (Breadboard with realistic interfaces)
		High fidelity lab integration of system completed, ready for test in simulated environments
		Some special purpose components combined with available laboratory components
		Laboratory environment modified to approximate operational environment
		Individual functions tested to verify that they work
		Individual modules and functions tested for bugs
		Integration of modules/functions demonstrated in a laboratory environment
		Algorithms run on processor with characteristics representative of target environment
	6	Factory acceptance testing of laboratory system in laboratory setting
		Representative model/prototype tested in high-fidelity lab/simulated operational environment
		Realistic environment outside the lab, but not the eventual operating environment
		Prototype implementation includes functionality to handle large scale realistic problems
		Algorithms partially integrated with existing hardware / software systems
		Individual modules tested to verify that the module components (functions) work together
		Components are functionally compatible with operational system
		Representative software system or prototype demonstrated in a laboratory environment
		Laboratory system is high-fidelity functional prototype of operational system
		Integration demonstrations have been completed
		Production demonstrations are complete
Hardware	7	Materials and manufacturing process and procedures initially demonstrated
		Each system/software interface tested individually under stressed and anomalous conditions
		Algorithms run on processor(s) in operating environment
		Most functionality available for demonstration in simulated operational environment
		Operational/flight testing of laboratory system in representational environment
		Fully integrated prototype demonstrated in actual or simulated operational environment
		System prototype successfully tested in a field environment.
Method	8	Components are form, fit, and function compatible with operational system
		Form, fit, and function demonstrated in eventual platform/weapon system
		All functionality demonstrated in simulated operational environment
		System qualified through test and evaluation on actual platform (DT&E completed)

Table A.1: AFRL questions that can be satisfied by the proposed lab-scale models.

Appendix B

Codes

This appendix collects the codes implemented for the LEGO stations.

B.1 EV3 Station Code

In this section is reported the *python* code used for the implementation of the station workflow described in Figure 3.4.

```
#!/usr/bin/env python3

from ev3dev.ev3 import *
from time import sleep
from datetime import datetime
import time
import random

Simulation_length = 10000000
Global_Time = datetime(2018, 11, 21, 10, 12, 39, 0)
GroupID = 12
Group_Type = [1, 1]
Group_Seed = [203, 84]
random.seed(Group_Seed[GroupID])
Product_A = 'red'
number_Product_A = 0

motor_block = MediumMotor('outD')
motor_station = LargeMotor('outB')
conveyor_opticalsensor = ColorSensor(INPUT_2)
conveyor_opticalsensor.mode = 'COL-COLOR'
```

```
station_opticalsensor = ColorSensor(INPUT_1)
station_opticalsensor.mode = 'COL-COLOR'
blocking_opticalsensor = ColorSensor(INPUT_3)
blocking_opticalsensor.mode = 'COL-COLOR'

colors = ('unknown', 'black', 'blue',
'green', 'yellow', 'red', 'white', 'brown')
i = 0
in_p = 0
out_p = 0
T_in_station = 0
T_out_station = 0
T_start = datetime.now()
Simulation_Start_Time = time.time()
workingA = 0
Current_Simulation_Time = 0
TOT_Load_Time = 0
TOT_Working_Time = 0
TOT_Unload_Time = 0
TOT_Block_Time = 0

file_TW = open('WorkingTime_A_S1.txt', 'w')
file_time_input = open('time_in_S1.txt', 'w')
file_time_finish = open('time_finish_S1.txt', 'w')
file_time_output = open('time_out_S1.txt', 'w')

while i < Simulation_length:
color_stat_available = colors[station_opticalsensor.value()]
color_conv_entering = colors[conveyor_opticalsensor.value()]
if (color_stat_available == 'black') and (color_conv_entering == Product_A)
in_p = in_p + 1
T_in_station = Global_Time + (datetime.now() - T_start)
T_in_station = T_in_station.strftime("%Y %m %d %H %M %S.%f")
file_time_input.write('{} '.format(T_in_station))
file_time_input.write('%d\n' % in_p)
LoadingTime_input = time.time()

motor_block.run_forever(speed_sp=-750)
sleep(0.45)
motor_block.stop(stop_action='hold')
motor_block.run_forever(speed_sp=750)
```

```
motor_station.run_forever(speed_sp=825)
sleep(0.45)
motor_block.stop(stop_action='hold')

a = 0
b = 0
T_check_input = time.time()
while a < 1:
color_stat_ent = colors[station_opticalsensor.value()]
if color_stat_ent == Product_A:
a = 1
if (time.time() - T_check_input) > 5:
a = 1
b = 1
if b == 1:
in_p = in_p - 1

motor_station.stop(stop_action="hold")
LoadingTime = time.time() - LoadingTime_input
TOT_Load_Time = TOT_Load_Time + LoadingTime

color_stat_working = colors[station_opticalsensor.value()]
if color_stat_working == Product_A:
number_Product_A = number_Product_A + 1
print('Number of Pieces is: ', number_Product_A)
if GroupID == 0:
Tw_Astoc = random.uniform(2, 8)
elif Group_Type[GroupID] == 1:
Tw_Astoc = random.triangular(2, 6, 4)
else:
Tw_Astoc = random.triangular(2, 6, 4)
workingA = Tw_Astoc
sleep(workingA)  # Operation time assignment of Product A
TOT_Working_Time = TOT_Working_Time + workingA

T_finish_operation = Global_Time + (datetime.now() - T_start)
T_finish_operation = T_finish_operation.strftime("%Y %m %d %H %M %S.%f")
file_time_finish.write('{} '.format(T_finish_operation))
file_time_finish.write('%d\n' % number_Product_A)

color_block_available = colors[blocking_opticalsensor.value()]
```

```
if color_block_available == Product_A:
Block = 1
BlockingTime_input = time.time()
BlockingTime = 0
while Block < 2:
color_block = colors[blocking_opticalsensor.value()]
if color_block == Product_A:
Block = 1
else:
Block = 2
BlockingTime = time.time() - BlockingTime_input
TOT_Block_Time = TOT_Block_Time + BlockingTime

UnloadingTime_input = time.time()
motor_station.run_forever(speed_sp=1000)
sleep(1.4)
out_p = out_p + 1
UnloadingTime = time.time() - UnloadingTime_input
TOT_Unload_Time = TOT_Unload_Time + UnloadingTime

T_out_station = Global_Time + (datetime.now() - T_start)
T_out_station = T_out_station.strftime("%Y %m %d %H %M %S.%f")
file_time_output.write('{} '.format(T_out_station))
file_time_output.write('%d\n' % out_p)
motor_station.stop(stop_action='hold')

Current_Simulation_Time = time.time() - Simulation_Start_Time
file_TW.write('%d ' % number_Product_A)
file_TW.write('%f ' % workingA)
file_TW.write('%f ' % Current_Simulation_Time)
file_TW.write('%f ' % TOT_Load_Time)
file_TW.write('%f ' % TOT_Working_Time)
file_TW.write('%f ' % TOT_Block_Time)
file_TW.write('%f\n' % TOT_Unload_Time)

i = i + 0.000000000001
file_TW.close()
file_time_finish.close()
file_time_input.close()
file_time_output.close()
```

Appendix C

Real-time Application of Model Generation

This appendix presents a test case for the application of the model generation procedure. For this purpose, the lab-scale models of manufacturing systems described in chapter 3 have been used.

C.1 Test Case

The data is generated by a 6-station flow line lab-scale model. A schematic representation of the line is shown in Figure C.1. The physical system is a closed-loop production line composed by six stations with intermediate conveyors that operate also as buffers. We denote with b_s the buffer capacity before station s. The blocking after service rule is applied. A fixed number of pallets ($n = 20$) circulates into the system. It is assumed that station $s = 1$ is the load/unload station and a large number of unprocessed parts are waiting in front of the first station, and that a finished part can immediately leave the system. Each station can process one part at the same time. Production activities are represented by the time p_s that a station holds a part before releasing

Table C.1: Test Case: parameters of the lab-scale model depicted in Figure 4.4.

Station s	Upstream Buffer Capacity b_s	Processing Time p_s $[s]$	Failure Probability f_s	Repair Time r_s $[s]$
1	4	1	0.15	UNIF(5,60)
2	3	1.5	0.1	UNIF(5,60)
3	6	1.1	0.35	EXPO(1)
4	6	1	0.34	Max(0.5, NORM(4,2))
5	2	Max(2, NORM(2, 10))	0	0
6	4	2.5	0	0

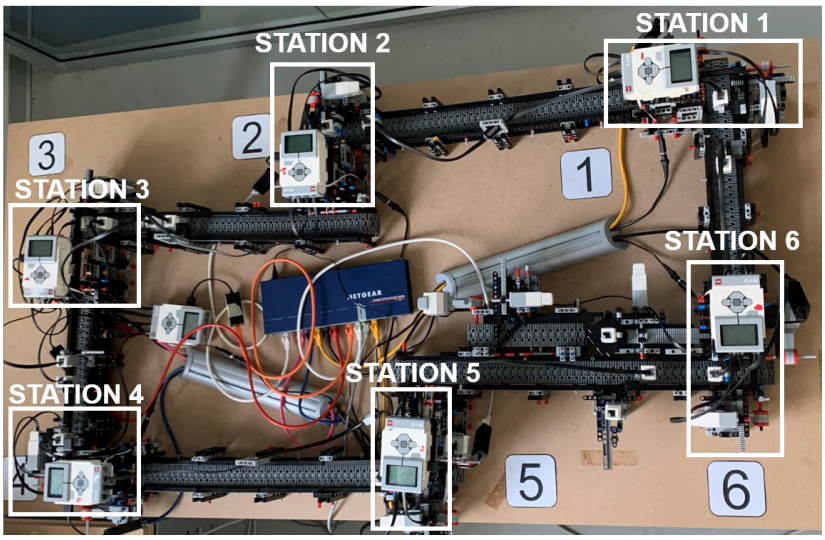


Figure C.1: Test Case: 6-station flow line used for the numerical analysis of this work.

Time-stamp	Part-ID	Activity-ID	Type
2020-11-23 16:37:40	1	1	start
2020-11-23 16:37:44	1	1	finish
2020-11-23 16:37:47	2	1	start
2020-11-23 16:37:51	2	1	finish
2020-11-23 16:37:52	1	2	start
2020-11-23 16:37:54	3	1	start
2020-11-23 16:37:57	1	2	finish

Table C.2: Test Case: portion of the event log generated by the lab-scale model of Figure 4.4.

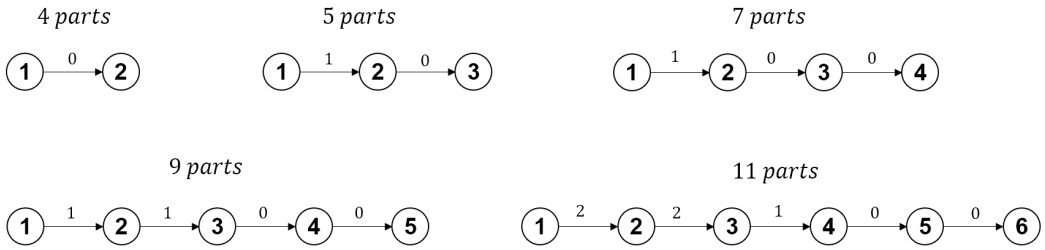


Figure C.2: Test Case: discovered models at different times (first log). Arcs are tagged with the respective buffer sizes.

it to the downstream conveyor. Stations $s \in [1, 4]$ are unreliable and may fail with probability f_s. If a failure occurs, the part is held by the station for an additional amount of time r_s, which accounts for the station repair. All stochastic quantities are sampled each time a part enters a station. The quantity $\phi_s(i)$ represents the whole operation time that the i-th part spends in a station, and its realization is as follows:

$$\phi_s(i) = \tilde{p}_s + \tilde{I}_s \tilde{r}_s \tag{C.1}$$

where $\tilde{I}_s$ is an indicator function which is 1 if $u < f_s$, 0 otherwise. u is a random number in the interval $[0, 1]$ and it is sampled each time a part enters a station. Conveyors move at a constant speed. Table C.1 reports the parameters of the lab-scale model[1].

C.2 Experiments

The system described in section C.1 has been used to produce parts for around 1 hour. This production experience has been repeated 9 times among different days, hence 9 independent event logs are available. We have applied the model generation procedure with an online setting. For practical convenience, experiments have been done in a separate time. Namely, we have selected the portion of the log such that $t_F(a, i) \leq \tau \, \forall i \in \mathbb{I}, a \in \mathbb{A}$, where $\tau \in \{25, 35, 45, 60, 100\}$, corresponding to the time to produce 4,5,7,9, and 11 parts, respectively. The goal is to observe the automated model building behavior during the initial transient phase. Further, we have recorded the models built for τ up to 2000 seconds, corresponding to the production of 142 parts.

[1]Although unrealistic, high failure probabilities have been set to increase the amount of failures that can be observed within a production session.

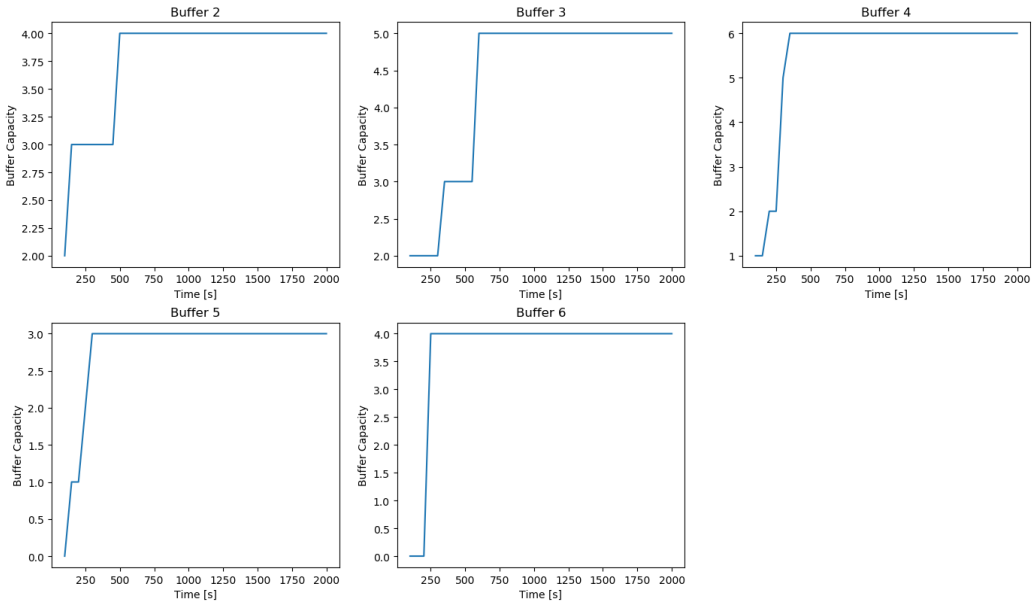


Figure C.3: Test Case: discovered buffers sizes depending on time.

The goal is to assess if the parameters of the system are correctly estimated given a certain amount of data. Finally, we have used the nine independent logs to check how parameters fitting techniques behave with different sample sizes. Namely, we have used (1) Kernel Density Estimation and (2) Empirical Cumulative Distribution Function to estimate the distribution of operation time ϕ_5.

C.3 Results

Figure C.2 shows the models developed from the first event log, depending on different values of τ. It can be noticed how the initial transient determines the capability of discovering the correct model of the system. Indeed, a time of $\tau = 100\,s$ is needed for obtaining the correct 6-station model. From Figure C.2 we can also notice that the discovered buffer sizes have not reached the real values even after $100\,s$. Figure C.3 shows the buffer capacities estimated by the model development method for $\tau \in [0, 2000]$. The convergence of all buffer capacities is reached after $\tau = 600\,s$. However, the convergence is not sufficient to obtain a correct estimation of buffer capacities. Indeed, the size of b_3 is biased by one slot even after $\tau > 2000\,s$. Figure C.4 shows the estimation of the processing time ϕ_5 among different days. We may observe that one day

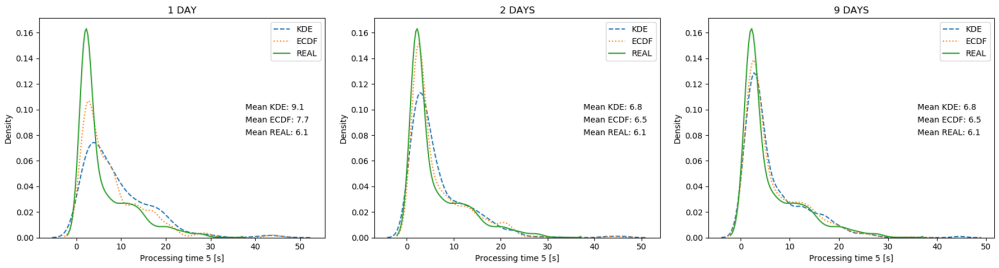


Figure C.4: Test Case: comparison among Kernel Density Estimation (KDE) and Empirical Distribution Function (ECDF) for the estimation of the operation time ϕ_5.

is not enough for an estimation of the distribution, two days can be sufficient for a rough estimation of the distribution parameters (e.g., first moment), while the correct estimation of the distribution is reached at 9 days, despite a biased estimation of the first moment which is mostly due to noise of the real data. From the results we may also infer that KDE and ECDF produce comparable results for the estimation of operation times.

Appendix D

Algorithms for Model Generation of Manufacturing Systems with Assembly Operations

This appendix presents the algorithms used in chapter 5 for the automated generation of graph models in the case of assembly operations.

D.1 Selection of Candidate Stations

In this section we elaborate on the method described in step 2 of the procedure in section 5.2.2. This step requires as input three information tables: the event log ($\mathbb{L}$), the BOM level selected at step 1, B_i, and the part types table (i.e., p_a is the part type of assembled product a). Let us define $\mathbb{E}$ as the set of events in the log. $id(e)$ is the part identifier corresponding to the e-th event in the log. Similarly, $st(e)$ is the station – or node – at which event e occurred. Algorithm 5 lists the steps to identify the set of candidate stations $\mathbb{S}_C$. Figure D.1 explains this step graphically using an example.

D.2 Definition of the set of combinations $\mathbb{V}$

In this section we elaborate on the method described in step 3 of the procedure in section 5.2.2. The obtained graph from the model generation approach (Chapter 4) can be divided in the collection of G subgraphs $\Omega = \{\Omega_1, \ldots, \Omega_G\}$. In step 3, the goal is to determine a set of tuples. For each tuple, the elements are the candidate assembly nodes for the par types of interest. Let us accept the short notation $N(\Omega_i)$ as the function returning the set

Algorithm 5: Selection of candidate stations (step 2).

Data: Event log $\mathbb{L}$, BOM i-th level B_i, part types table p_a;
Result: Set of candidate stations: $\mathbb{S}_C$;

```
for e ∈ 𝕃 do
    k ← id(e);
    if p_k ∈ B_i(0) then
        s ← st(e);
        𝕊_C ← s;
```

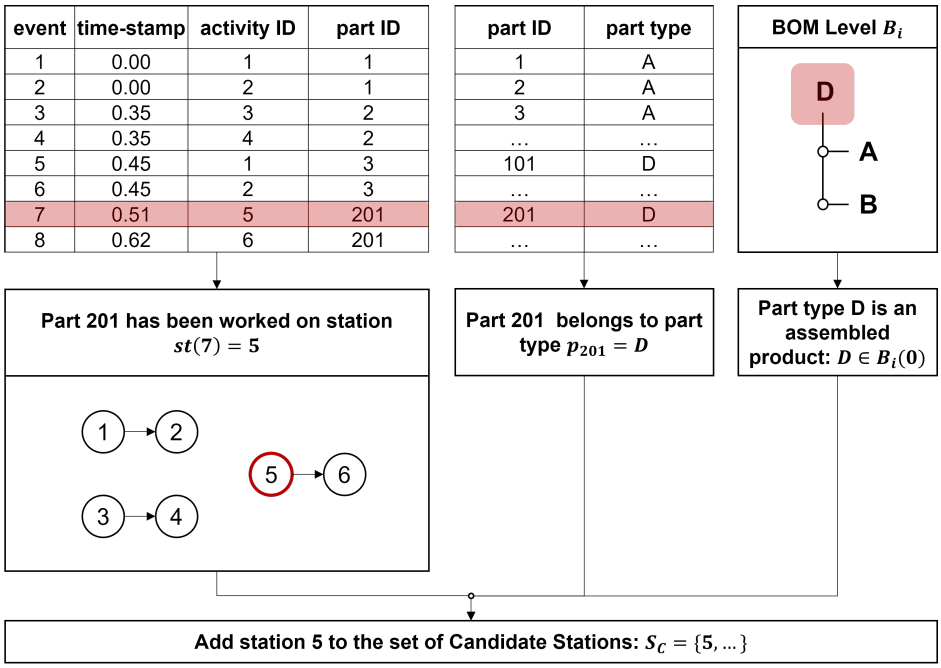


Figure D.1: Selection of candidate stations (step 2).

of nodes for the i-th subgraph. Algorithm 6 outlines the procedure of this step.

Algorithm 6: Definition of the set of combinations $\mathbb{V}$.

Data: collection of G subgraphs Ω, Candidate Stations $\mathbb{S}_C$;
Result: Set of station combinations $\mathbb{V}$;
for $\Omega_i \in \Omega$ **do**
  for $n \in N(\Omega_i)$ **do**
    if $n \in \mathbb{S}_C$ **then**
      $v \leftarrow n$;
      $\mathbb{S}_{\mathbb{C}} \leftarrow s$;
$\mathbb{V} \leftarrow v$;

D.3 Graph Model Retrieval

Once the GCP problem (section 5.2.1) has been solved, the solution in terms of graph model can be retrieved with a post-processing step. Namely, the graph additions defined by the variable α can be derived with the simple procedure listed in Algorithm 7. The algorithm analyses all the components of the

Algorithm 7: Graph model retrieval.

Data: GCP solution: x^*_{cas};
Result: Graph model addition variables: α_{ijp};
$\alpha_{ijp} \leftarrow 0 \quad \forall i, j, p$;
for $c \in \mathbb{C}$ **do**
  for $a \in \mathbb{A}$ **do**
    for $s \in \mathbb{S}$ **do**
      if $x^*_{cas} = 1$ **then**
        $l = \arg\max_{n \in \mathbb{S}} t_F(n, c)$;
        $\alpha_{l,s,p_a} \leftarrow 1$.

solution matrix x^*_{cas}. If $x^*_{cas} = 1$, it means the c-th component has been assigned to the a-th assembly on station s. In this case, the algorithm searches for the last station where a production record exists for component c. Namely, where the last station l recording the time-stamp $t_F(l, c)$. Such station is the starting node of the arc that represents the convergence of components of type p_a, toward station s. The arc is represented by $\alpha_{l,s,p_a} = 1$.

[...] e in fine il piacere che si prova in gustare e apprezzare i propri lavori, e contemplare da sé compiacendosene, le bellezze e i pregi di un figliuolo proprio, non con altra soddisfazione, che di aver fatta una cosa bella al mondo; sia essa o non sia conosciuta per tale da altrui.

[G. Leopardi, Zibaldone, 1828]