Proceedings of the 2020 Winter Simulation Conference
K.-H. Bae, B. Feng, S. Kim, S. Lazarova-Molnar, Z. Zheng, T. Roeder, and R. Thiesing, eds.

GENERATION AND TUNING OF DISCRETE EVENT SIMULATION MODELS FOR MANUFACTURING APPLICATIONS

Giovanni Lugaresi
Andrea Matta

Department of Mechanical Engineering
Politecnico di Milano
Via G. la Masa, 1
Milano, 20156 , ITALY

ABSTRACT

In order to successfully develop and apply cyber-physical systems in manufacturing it is fundamental to ensure the availability of up-to-date digital models. Production systems are intricate environments and are subject to frequent changes due to both external and internal factors. Therefore, it is complex to guarantee that a digital model can correctly represent the real system at any time. Literature is rich with methods for the automated generation of discrete event simulation models. However, the generated representations may be excessively accurate and describe also unnecessary activities. The automated development of digital models with an appropriate level of detail can avoid useless efforts and misguided predictions. This paper proposes a method that generates a simulation model and adjusts its level of detail exploiting the manufacturing system properties. The method has been applied in two test cases and can be used effectively to generate both Petri Net and simulation graph models.

1 INTRODUCTION

Recently, the manufacturing industry shifted toward adopting advanced technologies for data acquisition, storing and communication. New management policies and tools exploiting real-time data streams allow for taking decisions online, enabling quick response capabilities which improve the performances of manufacturing systems. Real-time Simulation is the concept that suggests using simulation as digital model of a system with the goal to make accurate choices based on the current system state (Lugaresi and Matta 2018). The ability to take decisions online is strongly based on the assumption that the digital models are properly aligned with the real system. However, this can rarely be satisfied since discrete event simulation models typically require a considerable amount of time to be built and validated. In order to overcome this issue, recent contributions in the literature proposed methods for generating models even in short time frames (Reinhardt et al. 2019). The most promising approaches are based on process mining, which can retrieve the system structure from the data (Van Der Aalst 2018). Nevertheless, the availability of digital models is not enough if they are excessively detailed. Figure 1 illustrates this concept with an example. A three-station production line is depicted in Figure 1a. Conveyors connect each station to the following one and are equipped with sensors that record the progression of pallets. Assuming that the model is generated starting from the data log of the system, a graph model that can be obtained is shown in Figure 1b. In this model the sensors have been interpreted as activities, thus representing unnecessary operations. Hence, despite the availability of a desired model, the result is an over-complex digital representation that may bring little if no advantage to the user. The graph model displayed in Figure 1c is a more reasonable abstraction of the process, and consistent with what an experienced modeler would choose. We may conclude that the ability to tune the model level of detail is a desirable feature in an automated modeling procedure.

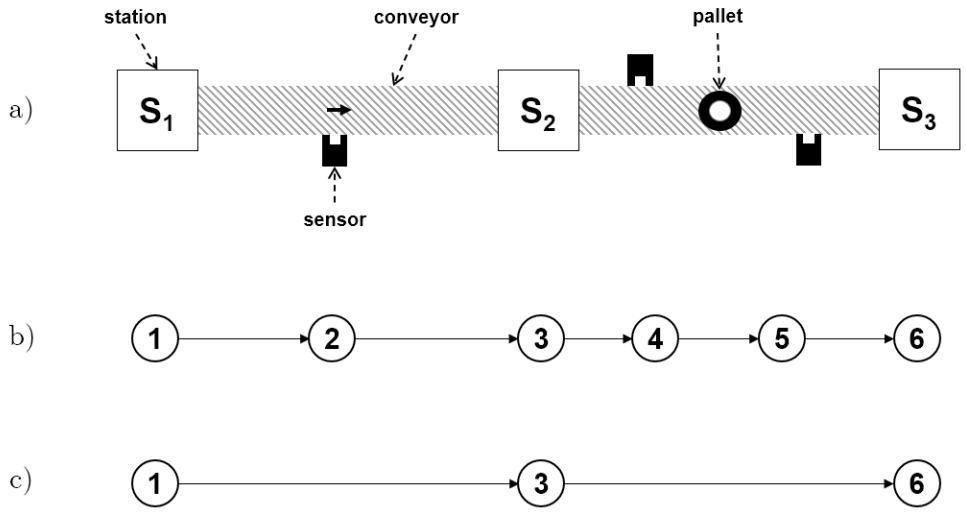

Figure 1: Problem introduction – (a) Three-station production line; (b) graph model with six nodes; (c) graph model with three nodes.

This paper proposes a model generation and tuning method that allows to obtain simulation models of a proper accuracy. The digital models are generated from the manufacturing system data logs, following a process mining approach. Model tuning is designed as an heuristic search problem where the final model size can be superiorly limited by the user. The paper is organized as follows: section 2 summarizes the significant contributions in the literature about model generation and tuning for manufacturing systems. Section 3 outlines the main components of the proposed method, while section 4 describes two test cases. Section 5 lists additional observations about this research, and section 6 presents our concluding remarks.

2 STATE OF THE ART

The automated generation of a simulation model can be summarized in the following main steps:

1. **Data collection.** Data is collected from the real system and organized in event logs recording the production steps followed by each part in the system and the corresponding time stamps (Perera and Liyanage 2000).
2. **Process topological discovery.** In this step, starting from the data available, the material flows are identified. The logical structure of the production process can be inferred from the sequence of activities performed by each work-piece (Van Der Aalst 2016).
3. **Statistical analysis.** Since processing and waiting times can be retrieved from data, the statistical distributions describing the process can be obtained, and their parameters can be estimated (Bowman and Azzalini 1997).
4. **Control policies identification.** In this phase, the rules followed by parts in the system (i.e. decision points) are retrieved (Ferreira and Vasilyev 2015).
5. **Model conversion.** The process model may be converted to another modeling framework (e.g., from graph model to Petri Net) as well as to executable code (Osais 2017).
6. **Model validation.** The generated simulation models are validated before being used to take decisions. Typically, validation is done by comparing the expected performance indicators with the ones obtained by the real system (Sargent 2013). Also decision rules classifiers have to be validated.

In this paper we focus on the first two points, with particular attention to the topological discovery.

2.1 Model Generation

The common approach to create a simulation model is to use a commercial off-the-shelf simulation package. A considerable advantage offered by such tools is that it is not necessary to be familiar with the underlying simulation language, since graphical user interfaces reduce the programming effort. Nevertheless, this process may be tiresome in case of large models. According to Mathewson (1985), a simulation generator is "*a software tool that translates the logic of a model into the code of a simulation language, enabling a computer to mimic a modelers behavior*".

Shimizu and Van Zoest (1988) developed an integrated approach in which *MANUPLAN* models can be translated into SIMAN models by a program called *SimStarter* program . To date, this procedure enabled to develop a simulation model of a factory in two days. Gong and McGinnis (1990) developed a Simulation Code Generator (SCG) which compiles a system description into a simulation code (e.g., SIMAN). The user can update the system composition in the model by modifying an input database and running the SCG. The authors demonstrated the SCG with a test case based on the information flows in an automated guided vehicle system for manufacturing cells. Son et al. (2000) developed a methodology for automatically generating simulation models for specific manufacturing scenarios. A neutral language has been designed to semantically describe a simulation model. A model translator then converts the neutral description into syntax of specific simulation packages. Mueller et al. (2007) introduced an approach in which the simulation model of a semiconductor manufacturing plant is generated from an input file that represents the simulation data specification. The simulation model is a Petri Net (PN) 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 PN data structure.

More recently, process mining approaches have also been used in applications of automated simulation model generation (Rozinat et al. 2009; Van Der Aalst 2018). Bergmann et al. (2015) introduced a methodology for recognizing the behavior of a manufacturing system in terms of production policies. Several data mining methods are tested (e.g., decision trees) with the goal to recognize which policies have been applied to the system. Farooqui et al. (2019) designed a methodology for generating formal models of robotic systems starting from the robot code structure and data. Milde and Reinhart (2019) developed an approach for joint material flow, parameters, and control policies identification from manufacturing systems event logs. In other applications, process mining has been used to retrieve specific parameters of a manufacturing system such as interarrival times (Martin et al. 2015), determinants of process delays (Ferreira and Vasilyev 2015), resource availabilities (Martin et al. 2016) batching operations (Martin et al. 2017), and production system structure (Denno et al. 2018).

2.2 Model Tuning

The availability of an up-to-date digital model may be insufficient since the data-based translation could model some parts of the system with an excessive level of detail. In the process mining literature, this is commonly known as the *spaghetti model* effect (Van Der Aalst 2016). Fuzzy process mining (Günther and Van Der Aalst 2007) has been developed to address this issue. Specifically, process models are generated from the real system event-logs. Nodes and arcs are tagged with significance and correlation properties. Then, nodes are either removed from the model following significance-based cutoff parameters or aggregated with other nodes based on thresholds on the correlation property. Several notions of correlation can be used (e.g., the similarities between the activity names). Prodel (2017) developed a model reduction procedure to generate process models of a health-care provider. Model reduction is formalized as a math programming problem and solved using a Tabu Search heuristic. The objective is to find the model that best fits the event log of the system. Lugaresi et al. (2019) formalized an integer programming model in which the solution represents a graph model where the number of nodes is imposed by the user.

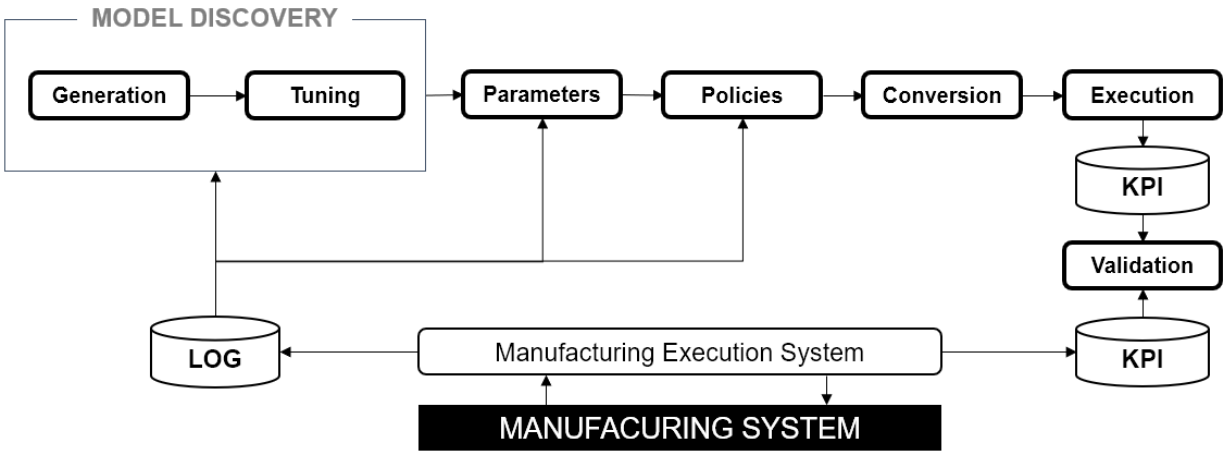

Figure 2: Graphical map of a typical simulation model generation procedure.

Although classical process mining methods can be used effectively for adjusting the model level of detail, most of the available approaches are appropriate for business process mining while they may not be suitable for manufacturing applications. For instance, exploiting the activity names to derive correlation measures can be proper for a service operation (e.g., bank, call center) but may be of little use in production systems, where names could simply correspond to sensors or machines identifiers. In this work, we aim at exploiting manufacturing systems properties (e.g., conveyors capacity, material flow) as indicators of the model ability of correctly representing the data, and we use them as main drivers of a model tuning procedure.

3 PROPOSED METHOD

Figure 2 summarizes the main steps of a typical simulation model generation procedure. It is assumed that the manufacturing system is equipped with a production management tool (e.g., a Manufacturing Execution System), which produces an event log. The event log is a file in which each row represents an event described by the following data: (1) the activity ID, (2) the work-piece ID, and (3) the time stamp *ts* indicating the moment the activity was performed by the piece. Starting from the log, the Model Discovery section includes modules aimed at the composition of a model Ω. Specifically, the Generation module provides an initial model Ω_0. The Tuning module contains libraries that adjust Ω_0 to a proper size, depending on user requirements. The final model Ω_f is used by the Parameters module, which determines the parameters of the statistical distributions describing the material flow. To do so, the event log is used again as reference. The log is also used to determine the production control policies that have been followed (e.g., priority rules). Finally, the complete model is converted into a desired language (e.g., Petri Net) by the Conversion module. The latter step also allows for assembling executable simulation code, hence to perform the model Validation based on production Key Performance Indicators (KPI).

3.1 Model Generation

Let us define a model Ω as a tuple $\Omega = (\mathbb{N}, \mathbb{A})$ where $\mathbb{N}$ is the set of nodes and $\mathbb{A}$ is the set of arcs. The Generation module creates an initial model $\Omega_0 = (\mathbb{N}_0, \mathbb{A}_0)$ starting from the event log. Specifically, the event log is used to populate a trace database, where each trace is the specific route that a work-piece followed in the system. Traces are used to retrieve precedence relationships between activities. Namely, a node exists in the model if a certain activity has been performed by at least one part in the system, and an arc indicates that a production step has followed another. Further details on the model generation steps are

available in Lugaresi et al. (2019). Based on the event log data, nodes and arcs are enriched with properties related to the manufacturing process. Following, we list the properties together with their notation.

3.1.1 Node properties

Each node $n \in \mathbb{N}$ is a tuple $n = (\mathbb{P}_n, \mathbb{S}_n, \kappa_n, \phi_n, \pi_n, \xi_n, \tau_n)$ with the following elements:

- *Predecessors* ($\mathbb{P}_n$) *and Successors* ($\mathbb{S}_n$) *nodes* are two sets of nodes that represent the activities done before and after the n-th node, respectively.
- *Capacity.* The capacity κ_n of a node is defined as the maximum amount of work-pieces that can be processed together by the corresponding production activity.
- *Frequency.* Each node n is tagged with the frequency ϕ_n with which it is observed in the log.
- *Contemporary events* (ξ_n). It is the number of events that have been tagged as contemporary on the n-th node. In general, we define two activities α and β to be contemporary if their time stamps satisfy $|ts_\alpha - ts_\beta| \leq \zeta$, where ζ is a user-defined threshold.
- *Branching probabilities (π_n).* 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.
- *Flowtimes.* It is a matrix $T_n = \{\tau_{k,n}\}$ where each element indicates the time work-piece k took to flow in node n.

3.1.2 Arc properties

Each arc $a \in \mathbb{A}$ is a tuple $a = (\eta_a, c_a, f_a, e_a, T_a)$ with the following elements:

- *Connected nodes.* The nodes that the arc connects are represented by a tuple $\eta_a = (n, m) \in \mathbb{N} \times \mathbb{N}$.
- *Capacity*. 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.
- *Frequency.* Each arc a is tagged with f_a, which is defined as the number of work-pieces that has been observed flowing through the arc.
- *Contemporary events* (e_a). It is the number of events that have been tagged as contemporary on the a-th arc.
- *Flowtimes.* It is a matrix $T_a = \{t_{k,a}\}$ where each element is the time that the k-th work-piece took to flow in arc a.

3.2 Model Tuning

The goal of a model tuning procedure is to find the model that best represents the real system data while satisfying a set of user-defined criteria. Let us define model size as the number of nodes in the model $S_N(\Omega) = |\mathbb{N}|$. In this work, we use the model size as single criterion to drive model tuning. Specifically, the Tuning module searches for the model that maximizes an utility score function indicating how well the model fits the real system data, while satisfying a constraint on the maximum number of nodes S_N^{max}. We have defined a score function composed by five parts that represent specific characteristics of a manufacturing system (e.g., buffer sizes). Since the estimated system parameters correspond to the properties of nodes and arcs, the score of a model is a link between the event log and the real system features. Following are the formulations of the score function components . For a compact notation, we introduce γ_{nm}, which is 1 if there is an arc connecting nodes n and m, 0 otherwise.

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

$$R_1(\Omega) = r_1^{(A)} \frac{\sum_{a\in\mathbb{A}} c_a}{\sum_{a\in\mathbb{A}_0} c_a} + r_1^{(N)} \frac{\sum_{n\in\mathbb{N}} \kappa_n}{\sum_{n\in\mathbb{N}_0} \kappa_n} \tag{1}$$

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

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

$$R_2(\Omega) = \frac{r_2^{(A)}}{|\mathbb{A}|} \sum_{a\in\mathbb{A}} (1 - \frac{e_a}{f_a}) + \frac{r_2^{(N)}}{|\mathbb{N}|} \sum_{n\in\mathbb{N}} (1 - \frac{\xi_n}{\phi_n}) \tag{2}$$

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

- 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 number of steps, 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 model. This function encourages the inclusion of loops in the model.

$$R_3(\Omega) = \frac{1}{|\mathbb{A}|} \sum_{n\in\mathbb{N}} \sum_{m\in\mathbb{N}} \gamma_{nm} \iota_{nm} \tag{3}$$

where ι_{nm} is 1 if the arc (n,m) is involved in a loop in the model, 0 otherwise.

- R_4 is a routing score as defined by Prodel (2017). For instance, in a flexible manufacturing system the logical routes may correspond to physical conveyors. Therefore, it is desirable to keep all the nodes with multiple connected arcs.

$$R_4(\Omega) = r_4^{(in)} \sum_{n\in\mathbb{N}} \sum_{x\in\mathbb{S}_n} \gamma_{nx} + r_4^{(out)} \sum_{n\in\mathbb{N}} \sum_{l\in\mathbb{P}_n} \gamma_{ln} \tag{4}$$

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

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

$$R_5(\Omega) = r_5^{(A)} \frac{\sum_{a\in\mathbb{A}} f_a}{\sum_{a\in\mathbb{A}_0} f_a} + r_5^{(N)} \frac{\sum_{n\in\mathbb{N}} \phi_n}{\sum_{n\in\mathbb{N}_0} \phi_n} \tag{5}$$

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

Finally, the score of a model Ω is defined as follows:

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

where w_i is the weight of the i-th score ($\sum_i w_i = 1$). Model tuning is performed with the local search heuristic explained in Algorithm 1. The algorithm is based on the possibility of generating neighbors of a model Ω. A neighbor is generated by either removing or aggregating nodes. Specifically, a model $\Omega' = (\mathbb{N}', \mathbb{A}')$ is neighbor of Ω if $\mathbb{N}' \subseteq \mathbb{N}$ and $|\mathbb{N}'| = |\mathbb{N}| - 1$. σ_A and σ_R indicate the number of neighbor

Algorithm 1

STEP 0: Generate initial model Ω_0 (Generation module); assign $\Omega_{current} \leftarrow \Omega_0$; $i \leftarrow 0$,
while $(i \leq I_{max}) \neg (S_N(\Omega) \leq S_N^{max})$ **do**
  STEP 2: Generate $\sigma_A + \sigma_R$ neighbors of Ω
  STEP 3: Sort neighbors based on score Φ
  STEP 4: Set $\Omega_{current} \leftarrow argmax_{\Omega}\Phi(\Omega)$, $i \leftarrow i+1$
  if $\Phi(\Omega_{current}) > \Phi(\Omega_{best})$ **then**
    $\Omega_{best} \leftarrow \Omega_{current}$
  end if
  go to STEP 2
end while

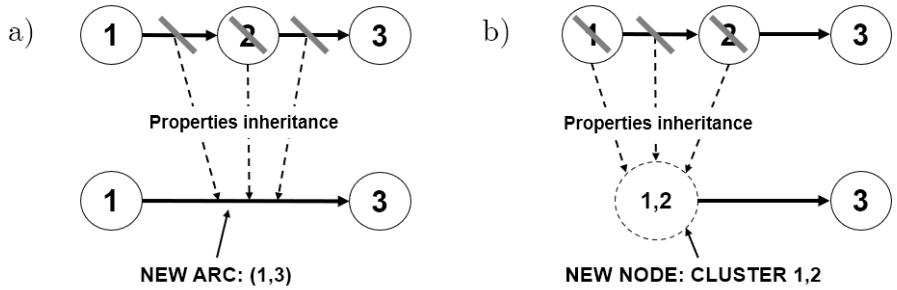

Figure 3: Examples of properties inheritance in a model tuning procedure: (a) reduction, (b) aggregation.

models generated by aggregation and reduction at each step of Algorithm 1, respectively. To generate the neighbors, a user-defined criterion is followed. In this work, we start to aggregate or remove the nodes identified by arcs with the lowest material flow ($n \in \eta_a \,|a = \arg\min_{a \in \mathbb{A}} f_a$). New arcs may be added during the neighbors generation step in order to allow the material flow, and new nodes may be added to represent activity clusters. Figure 3 represents an example of this idea. In the reduction case, node 2 is removed from the model, hence the connected arcs $(1,2)$ and $(2,3)$ are also discarded and arc $(1,3)$ is added to guarantee the flow of parts (Figure 3a). The newly added elements inherit the properties of the removed items. For instance, the inherited capacity of the arc $(1,3)$ is the sum of the capacities of node 2, arc $(1,2)$, and arc $(2,3)$. In the aggregation case, a new node is created by merging nodes 1 and 2 (Figure 3b). Also in this case, the properties of the added node are derived from the ones of the removed elements. At each iteration, Algorithm 1 selects the best neighbor model based on the obtained score Φ. Then, it uses this model as new basis to obtain neighbors of smaller size. The procedure stops when either a model of the required size Ω_f is obtained or the maximum number of iterations I_{max} is reached.

4 TEST CASES

We have tested the model generation and tuning method on two test cases: (1) an M/M/1/K queue, and (2) a flow line with six production stations and a quality station. The goal of the first test is show the model generation and conversion procedure, while in the second case we aim to show the model tuning behavior.

4.1 Test Case 1: M/M/1/K queue

In this first test, we used an M/M/1/K queue, namely with arrivals and processing times distributed according to exponential distributions, a single server and a queue capacity of K parts. A purposely built simulator has been written in *python 3.6* and used to generate data for event logs describing the following activities of 1000 parts: (1) *buffer in*, corresponding to a new part arrival, (2) *server in* and (3) *server out* corresponding

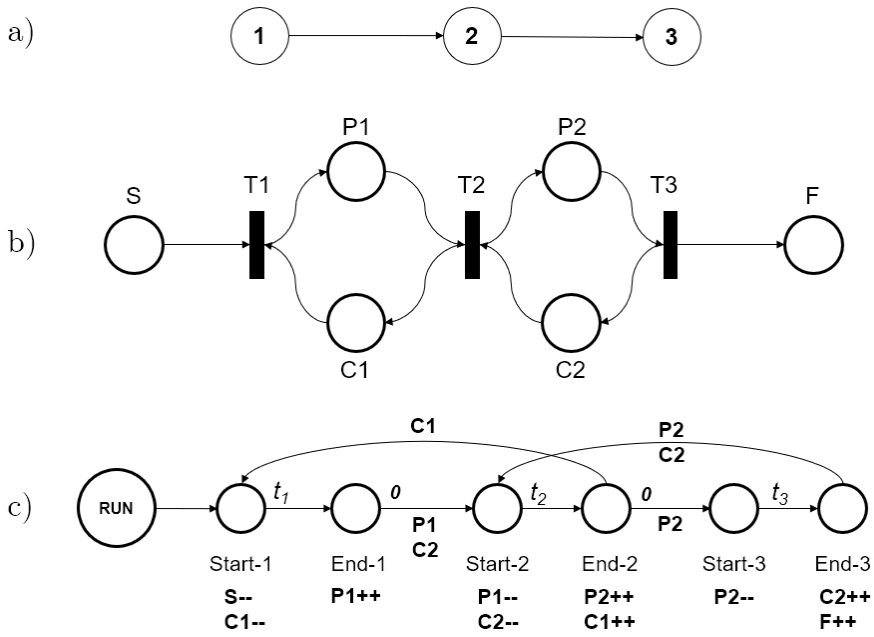

Figure 4: Test Case 1 – Different representations of the M/M/1/K queue: a) Graph model, b) Petri Net model, c) ERG model.

to the beginning and the end of the operations at the server, respectively. The arrivals and the processing times follow exponential distributions with mean $5s$ and $6s$, respectively. Figure 4 shows the different models that have been obtained. Specifically:

a) The **graph model** Ω_0 generated by the Generation module is shown in Figure 4a. Each activity in the log is mapped into a node and the arcs represent the precedence relationships between activities. Given the very simple process, model tuning is not performed and $\Omega_f = \Omega_0$.

b) The resulting **Petri Net model** of the system is in Figure 4b. Each activity recorded in the log is mapped into a transition, hence three transitions are created: T1, T2, and T3. Each arc is modeled as a reentrant flow of tokens through a place holding as many tokens as the arc capacity. Specifically, the capacity of the first arc is modeled by place C1, and – similarly – C2 for the second arc. The source and sink places (in the figure, S and F) are added before the first and after the last activity, respectively.

c) Figure 4c shows the result of the conversion of the PN model into a **simulation graph**, i.e. an Event Relationship Graph (ERG). The conversion procedure is taken from Schruben and Yucesan (1994). In short, each activity is mapped with a starting and finishing vertex, connected by an edge. Each arc in the PN becomes an edge in the ERG between the end of an activity and the beginning of another. The duration of the activities are represented by the edge delay parameters (i.e. t_1, t_2, and t_3). The number of tokens in each place is modeled by corresponding state variables with the same name as the place. For instance, the capacity of the first arc is visible in the edge connecting the *End-2* and *Start-1* vertex. The edge condition is C1, meaning that activity 1 cannot start if the buffer downstream is full. The buffer level is updated when activity 2 is finished.

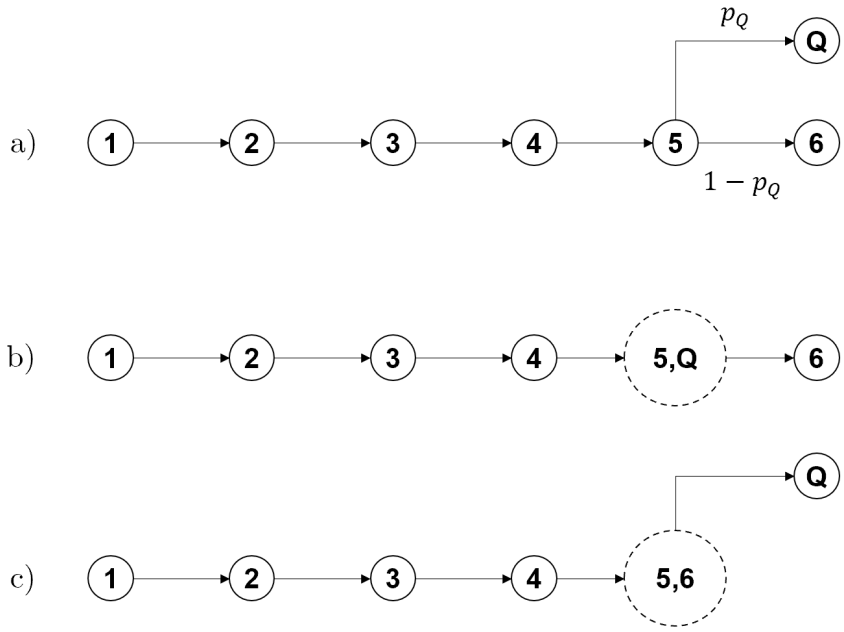

Figure 5: Test Case 2 – a) flow line graph model with six stations and a quality control station; b) graph model obtained with $p_Q < 0.5$ and $S_N^{max} = 6$; c) graph model obtained with $p_Q \geq 0.5$ and $S_N^{max} = 6$.

4.2 Test Case 2: sequential flow line

The second test has been done on a sequential flow line. The system is composed by six stations and one quality control and rework station. Parts flow through the first five stations in sequence. Then, a portion p_Q of parts go to the rework station, while the parts that do not need rework go to station 6. The logical layout of the system is shown in Figure 5a. The model has been implemented in Rockwell Arena® and used to generate event logs of 28,000 parts each. The Algorithm 1 has been implemented in *python 3.6* and we have used $\sigma_A = \sigma_R = 3$. This test is divided in two parts: (1) $S_N^{max} = 6$, and (2) $S_N^{max} = 2$.

1. **First level of model tuning.** We impose a model size reduced by one node, hence $S_N^{max} = S_N(\Omega_0) - 1 = 6$. We check how the resulting model varies by changing the quality station sampling probability p_Q. The first event log is generated by a model with $p_Q = 0.05$. As a result, the flow on the arc (5,Q) is very low and the quality station is aggregated with station 5, as shown in Figure 5b. Figure 6 shows the resulting Petri Net model. Then, we have repeated the tuning procedure using event logs generated with increasing values of p_Q, starting from 0.05 till 0.55 with a 0.05 step. As expected, for $p_Q \geq 0.5$ the arc $(5,6)$ becomes the least visited and stations 5 and 6 are aggregated (Figure 5c).
2. **Final level of model tuning.** We aim to obtain the simplest model possible: we exclude the possibility of a single-node model and we set $S_N^{max} = 2$. The algorithm takes less than 10 s to go from the first to the last model tuning step. Table 1 summarizes the model nodes obtained at each step of the model tuning procedure. As expected, the model score values Φ are lower for models with lower size, indicating that models of smaller size are farther from a complete representation of the manufacturing system. The final model Ω_f is a two-node graph composed by the clusters of activities (1, 2, 3, 4) and (5, 6, Q).

S_N^{max}	Model Nodes							Φ
7	1	2	3	4	5	6	Q	0.400
6	1	2	3	4	(5, Q)	6		0.367
5	1	2	3	4	(5, Q, 6)			0.333
4	(1, 2)	3	4	(5, Q, 6)				0.300
3	(1, 2, 3)	4	(5, Q, 6)					0.267
2	(1, 2, 3, 4)	(5, Q, 6)						0.233

Table 1: Test Case 2 – Nodes of the models obtained in each step of the model tuning procedure.

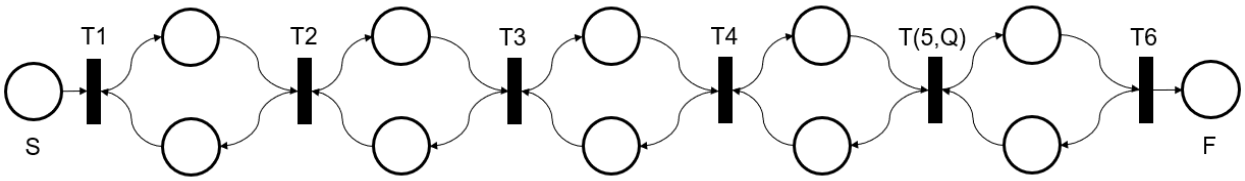

Figure 6: Test Case 2 – Petri Net model of the production line obtained with $S_N^{max} = 6$ and $p_Q < 0.5$.

5 ADDITIONAL REMARKS

Following we list our remarks on the applicability of this work:

- The event log that we have used contains only three data types. The addition of information to the log (e.g., operator identifiers, work-piece dimensions) can be used to generate richer models which may bring additional insights to the user.
- In this work, we have used event log with a single time stamp per activity. Other works manage event logs with specified intervals (i.e. start and finish times of each activity). Since both logging methods coexist in manufacturing enterprises, trace mining functions have to be tailored to the specific event log type before being used. Alternatively, a standard event log type such as the eXtensible Event Stream (XES) could be used (Van Der Aalst 2016).
- One of the main limitations of model generation is that models are generated from past information. As a result, only activities which resulted from past decisions are visible, while potential ones are hidden. This means that the generated models may forbid actions that are indeed admissible despite not recorded in the log.
- In the production line test, we used the material flow as proxy to reduce the model. Since along a sequence of operations the part flow is constant, several equivalent solutions may be produced during model tuning. Hence, we may infer this proxy is more reasonable for processes with several parallel activities in which excluding the least visited paths is of interest.
- In manufacturing applications, moving parts may trigger specific actions. If those activities are removed or aggregated in the tuning process, the resulting model might loose in accuracy. More work is needed to extend the method with an accurate decision point analysis.
- It is worth to notice that model tuning can also be performed on simulation graphs (Yucesan and Schruben 1992). Hence, the proper balance between the two methods may need to be investigated.

6 CONCLUSIONS

In this work, we have described a model generation and tuning method that allows to obtain digital models starting from the event logs of manufacturing systems. The proposed method is beneficial for production planning and control, since the capability to generate an accurate model in a short time can enable Real-time

Simulation applications. Also model validation is 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. Several issues still have to be solved. Smarter aggregation rules can be developed, for instance by clustering sets of parallel activities first if the model structure suggests it. In the future, we will run experiments to verify the algorithm behavior by using different model tuning proxies, and we will investigate the influence of the score weights on the final outcome. Further, we will also complete the model conversion procedure that allows for the estimation of production-related performance indexes (e.g., throughput, queuing time), thereby enabling model validation. Last but not least, since the availability of ERG models allows for obtaining mathematical programming formulations, we will investigate online simulation-optimization applications of the generated models (Pedrielli et al. 2018).

REFERENCES

Bergmann, S., N. Feldkamp, and S. Strassburger. 2015. "Approximation of dispatching rules for manufacturing simulation using data mining methods". In *Proceedings of the 2015 Winter Simulation Conference*, edited by L. Yilmaz, W. K. V. Chan, I. Moon, T. M. K. Roeder, C. Macal, and M. D. Rossetti, 2329–2340. Piscataway, New Jersey: Institute of Electrical and Electronics Engineers, Inc.

Bowman, A. W., and A. Azzalini. 1997. *Applied smoothing techniques for data analysis: the kernel approach with S-Plus illustrations*, Volume 18. OUP Oxford.

Denno, P., C. Dickerson, and J. A. Harding. 2018. "Dynamic production system identification for smart manufacturing systems". *Journal of manufacturing systems* 48:192–203.

Farooqui, A., K. Bengtsson, P. Falkman, and M. Fabian. 2019. "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.

Ferreira, D. R., and E. Vasilyev. 2015. "Using logical decision trees to discover the cause of process delays from event logs". *Computers in Industry* 70:194–207.

Gong, D.-C., and L. F. McGinnis. 1990. "An AGVS Simulation Code Generate for Manufacturing Applications". Technical report, Institute of Electrical and Electronics Engineers (IEEE).

Günther, C. W., and W. M. P. Van Der Aalst. 2007. "Fuzzy mining–adaptive process simplification based on multi-perspective metrics". In *International conference on business process management*, 328–343. Springer.

Lugaresi, G., and A. Matta. 2018. "Real-time Simulation in Manufacturing Systems: Challenges and Research Directions". In *Proceedings of the 2018 Winter Simulation Conference (WSC)*, edited by M. Rabe, A. A. Juan, N. Mustafee, A. Skoogh, S. Jain, and B. Johansson, 3319–3330. Piscataway, New Jersey: Institute of Electrical and Electronics Engineers, Inc.

Lugaresi, G., M. Zanotti, D. Tarasconi, and A. Matta. 2019. "Manufacturing Systems Mining: Generation of Real-Time Discrete Event Simulation Models". In *2019 IEEE International Conference on Systems, Man and Cybernetics (SMC)*, 415–420. IEEE.

Martin, N., F. Bax, B. Depaire, and A. Caris. 2016. "Retrieving resource availability insights from event logs". In *2016 IEEE 20th International Enterprise Distributed Object Computing Conference (EDOC)*, 1–10. IEEE.

Martin, N., B. Depaire, and A. Caris. 2015. "Using process mining to model interarrival times: investigating the sensitivity of the ARPRA framework". In *Proceedings of the 2015 Winter Simulation Conference*, edited by L. Yilmaz, W. K. V. Chan, I. Moon, T. M. K. Roeder, C. Macal, and M. D. Rossetti, 868–879. Piscataway, New Jersey: Institute of Electrical and Electronics Engineers, Inc.

Martin, N., M. Swennen, B. Depaire, M. Jans, A. Caris, and K. Vanhoof. 2017. "Retrieving batch organisation of work insights from event logs". *Decision Support Systems* 100:119–128.

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

Milde, M., and G. Reinhart. 2019. "Automated Model Development and Parametrization of Material Flow Simulations". In *Proceedings of the 2019 Winter Simulation Conference*, edited by N. Mustafee, K.-H. Bae, S. Lazarova-Molnar, M. Rabe, C. Szabo, P. Haas, and Y.-J. Son, 2166–2177. Piscataway, New Jersey: Institute of Electrical and Electronics Engineers, Inc.

Mueller, R., C. Alexopoulos, and L. F. McGinnis. 2007. "Automatic Generation of Simulation Models for Semiconductor Manufacturing". In *Proceedings of the 2007 Winter Simulation Conference*, edited by S. G. Henderson, B. Biller, M.-H. Hsieh, J. Shortle, J. D. Tew, and R. R. Barton, 648–657. Piscataway, New Jersey: Institute of Electrical and Electronics Engineers, Inc.

Osais, Y. E. 2017. *Computer Simulation: A Foundational Approach Using Python*. CRC Press.

Pedrielli, G., A. Matta, A. Alfieri, and M. Zhang. 2018. "Design and control of manufacturing systems: a discrete event optimisation methodology". *International Journal of Production Research* 56(1-2):543–564.

Perera, T., and K. Liyanage. 2000. "Methodology for rapid identification and collection of input data in the simulation of manufacturing systems". *Simulation Practice and theory* 7(7):645–656.

Prodel, M. 2017. *Modélisation automatique et simulation de parcours de soins à partir de bases de données de santé*. Ph. D. thesis, Lyon.

Reinhardt, H., M. Weber, and M. Putz. 2019, jan. "A Survey on Automatic Model Generation for Material Flow Simulation in Discrete Manufacturing". *Procedia CIRP* 81:121–126.

Rozinat, A., R. S. Mans, M. Song, and W. M. Van der Aalst. 2009. "Discovering simulation models". *Information systems* 34(3):305–327.

Sargent, R. G. 2013. "Verification and validation of simulation models". *Journal of simulation* 7(1):12–24.

Schruben, L., and E. Yucesan. 1994. "Transforming Petri nets into event graph models". In *Proceedings of the 1994 Winter Simulation Conference*, edited by J. D. Tew, S. Manivannan, D. A. Sadowski, and A. F. Seila, 560–565. Piscataway, New Jersey: Institute of Electrical and Electronics Engineers, Inc.

Shimizu, M., and D. Van Zoest. 1988. "Analysis of a factory of the future using an integrated set of software for manufacturing systems modeling". In *Proceedings of the 1988 Winter Simulation Conference*, edited by M. Abrams, P. Haigh, and J. Comfort, 671–677. Piscataway, New Jersey: Institute of Electrical and Electronics Engineers, Inc.

Son, Y. J., A. T. Jones, and R. A. Wysk. 2000. "Automatic generation of simulation models from neutral libraries: an example". In *Proceedings of the 2000 Winter Simulation Conference*, edited by J. A. Joines, R. R. Barton, K. Kang, and P. A. Fishwick, 1558–1567. Piscataway, New Jersey: Institute of Electrical and Electronics Engineers, Inc.

Van Der Aalst, W. M. 2016. "Data science in action". In *Process mining*, 3–23. Springer.

Van Der Aalst, W. M. 2018. "Process mining and simulation: A match made in heaven!". In *Simulation Series*, Volume 50, 39–50: The Society for Modeling and Simulation International.

Yucesan, E., and L. Schruben. 1992. "Structural and behavioral equivalence of simulation models". *ACM Transactions on Modeling and Computer Simulation (TOMACS)* 2(1):82–103.

AUTHOR BIOGRAPHIES

GIOVANNI LUGARESI is Ph.D. Candidate in the Department of Mechanical Engineering of Politecnico di Milano. His research interests include process mining, simulation-optimization in manufacturing systems, and robust optimization for production planning and control. His email address is giovanni.lugaresi@polimi.it.

ANDREA MATTA is Full Professor at Politecnico di Milano, where he currently teaches integrated manufacturing systems and manufacturing. His research area includes analysis, design and management of manufacturing and health care systems. He is Editor in Chief of the Flexible Services and Manufacturing Journal. His email address is andrea.matta@polimi.it.