Journal of Intelligent Manufacturing
https://doi.org/10.1007/s10845-021-01767-2

Multi-product scheduling through process mining: bridging optimization and machine process intelligence

Alexandre Checoli Choueiri[1] · **Eduardo Alves Portela Santos**[1]

Received: 3 May 2020 / Accepted: 25 March 2021

Abstract
Small and medium enterprises (SMEs) may not have the maturity to put forward and unfold all the benefits from an ERP based system, a vital tool for production planning. Manufacturing ubiquitous trends, however, are more approachable to SMEs, and even the more affordable tools could be of great advantage. In this paper we propose an algorithmic framework that uses process mining tools to extract the underlying industrial process via Petri nets, and then retrieve all product tree necessary information to perform the multi-level scheduling. A faster solution decoding is proposed, for algorithms that uses random-keys. Computational experiments show that the new decoding is faster than the usual, leading to promising new paths on its future uses.

Keywords Multi-product · Process mining · Scheduling · Random-keys

Introduction

In order to achieve both client expectations and cost efficiency, a well designed production planning and control requires to have the right part, at the right quantity at the right time, with the lowest possible cost (Öztürk and Ornek 2014). The core activity of planning and controlling production is the scheduling. Scheduling algorithms are responsible for determining the allocation of resources to tasks over a time period, while minimizing one or more objective functions (Pinedo 2012).

A special case of scheduling relates to multi-level products, where the sequencing of tasks should follow a product tree (dependency graph) order; some sub-parts depends directly of others to be produced (Öztürk and Ornek 2014). Na and Park (2014) points out that there are little studies concerned with the multi-level scheduling, and the ones that exists are of little practicality, since they need to make many assumptions in order to solve the problem. The scheduling of sub-parts, as a short-term production plan, usually follows from a mid-term plan executed by the MRP (material requirements planning), which estimate all material needed for the plannings (Na and Park 2014).

The problem with MRP technology systems is that their implementation is not a trivial task, requiring capital invested and commitment, and once installed, can be extremely rigid (Cullinan et al. 2010). Small and medium enterprises (SMEs) may not have the maturity to put forward and unfold all the benefits from an ERP based system. This process results in low success rates of implementation (Petroni 2002).

Since scheduling functionalities are closely related to information systems (such as ERP or even the MRP), it becomes a challenge for SMEs to achieve efficient planning and control activities in their production floor, unveiling a gap from this companies regarding a more ready to use framework. This solution would need to enable reliable scheduling optimization that are not necessarily intertwined with a robust information system and database. This framework is now a possibility, with the emerge of industry 4.0.

The concept of industry 4.0 is complex to define, and many definitions arise in literature ((Trappey et al. 2017), (Schumacher et al. 2016)). In all definitions, however, the Industry 4.0 concept is based on new technologies, such as internet of things (IoT), cloud computing, cyber-physical systems and big data. In the production floor, this data flow is put forward through intelligent monitoring; machines are connected as a collaborative community (Lee et al. 2014), providing man-

✉ Alexandre Checoli Choueiri
alexandrechecoli@gmail.com

Eduardo Alves Portela Santos
eduardo.portela@pucpr.br

[1] Pontifical Catholic University of Parana, Curitiba, Brazil

Published online: 12 April 2021

agers the possibility of using on-line data in order to make more accurate decisions.

Despite the fact that implementing the tools of industry 4.0 may also require large investments (as MRP-based systems), they are more flexible, in a sense that it decentralizes information and decision-making (Moeuf et al. 2018), becoming more accessible to SMEs. Even the more affordable tools of industry 4.0 are reported to generate great benefits for SMEs. The use of these tools to collect machine data alone, however, do not constitute intelligent manufacturing. A powerful way of dealing with this new machine-based data logs is through process mining techniques, that are able to summarize and gather information about the underneath process (see (Thiede et al. 2018) and (dos Santos et al. 2019) for recent reviews on how companies are using process mining to improve their processes).

The main concern in process mining is the discovery of processes based on event logs (e.g., performing activities or exchanging recorded messages) (Van Der Aalst 2016). With this information, discovery algorithms are executed to extract a process model from the data. The model can be mined and output on several different languages, like Petri nets (Sun et al. 2019) or even BPMN (Kalenkova et al. 2019). Following the ubiquitous manufacturing trend (Wang et al. 2018), process mining presents itself as a perfect layer between the production floor, the information system and the end user, in order to achieve machine intelligence.

In this paper, we thus propose a framework that uses the information generated by machines in the form of event logs, extracts the underlying process to a Petri net, and from the net and the data log reconstruct the product tree (dependency graph). This extraction provides all necessary information to perform the scheduling of products, in a multi-level structure. We also propose a method that uses the machinery logs to collect 3 values for each machine processing time, in a way that the stochastic behavior of the production is accounted for, in the form of different scheduling scenarios; optimistic, realistic and pessimistic. In order to perform 3 different scenarios for each scheduling, the optimization engine should be fast enough to handle the task. For this, we propose an acceleration method to decode scheduling solutions that are based on random keys, a common codification for the majority of genetic algorithms implementations. The method is laid upon two new bounds extracted from the product tree, and a binary search. The new procedure reduces the complexity of the classic random keys decoding process from $O(n)$ to $O(\log n)$.

With this new framework, we are bridging machine intelligence and optimization through the use of process mining techniques. This would enable to structure a production planning and control for SMEs, even without the use of major ERP or MRP-based systems, using the already collected data from machines.

The main contributions of this paper are as follow:

1. A new algorithm that extracts multi-level product tree (dependency graph) from Petri nets, and the necessary quantities of each product.
2. Creation of scheduling scenarios based on 3 different values, extracted from the machinery logs by confidence intervals.
3. New decoding bounds for random keys, based on the multi-level product structure, and an accelerated method to decode the solutions, using the bounds and a bisection method.

The reminder of the paper is organized as follows: in "Literature review" section we provide some basic concepts of process mining, as well as some related works conducted on multi-level product structures and the use of random keys for scheduling. In "Proposed framework" section we explain in details all components and algorithms of our approach, and how it relates to the industrial environment and machine intelligence. On "Computational results" section we describe computational results obtained by applying the new proposed bisection decoding, using a genetic algorithm as the metaheuristic. Finally, in "Conclusion" section we conclude the paper with some remarks, limitations and future works.

Literature review

In the first part of this Section we describe some basic concepts and terminology of process mining. On the second part we expose previous work on multi-level scheduling, and the use of random keys for coding solutions.

Process mining premises

More and more information systems support manufacturing processes. This generates a large amount of raw process data recorded daily and stored in huge databases. Nevertheless, organizations are still unable to extract complete information from the data (?). If the database records store the execution of process instances (logs), the use of process mining techniques are a viable option to perform knowledge extraction. The main goal of process mining is to extract process related information (e.g, automatically discover a process model) from event data. The first step on the discovery is to mine a process model. Once the model is at hand, it is possible to replay occurred events to check conformance and uncover bottlenecks (De Leoni et al. 2016).

In process mining terminology, an *event* is characterized by various properties, e.g., an event has a *timestamp*, a resource identifying the executor, associated costs, and so on. Each event must be associated with a *case*. When all

the events of a case are in chronological order, we have a *trace* (a finite non-empty sequence of events, such that each event appears only once and time is non-decreasing). Note that it is possible to have various cases that follows a same trace, but each case is different. An *event-log* is a set of *traces*. In theory, any process that has a time dimension could be stored as an event-log database, including manufacturing activities. All manufacturing activities performed on a giving product can be stored on a PO (production order), which held both needed resources (machinery, raw material quantities, time consumed) and their execution times. Considering that, hereafter we use the terms *activities* and *machines* interchangeably.

Process mining techniques are not yet wide spread among manufacturing researchers and practitioners, although that is bound to change with industry 4.0 and new machine intelligent designed methods. Myers et al. (2018) uses process mining to prevent cyber attacks in industrial systems, by analyzing anomalous log patterns, the authors use data mining and conformance checking. Bruntsch and Tseng (2016) models statistical process control, as a responsive toll using process mining. Ruschel et al. (2018) has integrated process-mining and Bayesian networks in order to find maintenance intervals, Choueiri et al. (2020) uses a hybrid model to predict remaining time (cycle-time), optimized by linear programming weights.

Multi-product scheduling and random keys

A scheduling optimization deals with the allocation of resources to tasks over a time period. A number of variants exists, regarding machine environment, process characteristics and different constraints (Pinedo 2012). The multi-product scheduling has the precedence constraints on its tasks (the product tree, or bill of material); meaning that, for a given node N_p (task) to be feasibly allocated to start at a given time period s_p, all of its child nodes N_i (if they exist) must be allocated, such that their ending times e_i repects $e_i \leq s_p \forall i \in N_i$. Some common problem characteristics for the multi-product scheduling are as follows:

1. Except for end items, subassemblies and component items (non-end items) have successors (or parents) in the product tree. Component items have no predecessors.
2. There may exist independent demand for all items on the tree.
3. A manufacturing order for an item can be realized on any resource from the set of eligible resource pool of that item.
4. A resource can be shared by items of different levels.

Na and Park (2014) commented that only a few of previous studies has focused on these problems. To the best of our knowledge, the first study that has dealt with this problem is the work of Kim and Kim (1996). The authors considered a multi-level structure where only two operations are allowed, assembly and machinery. In this way, the two operations can be viewed as only two different machines available. The problem is optimized using two meta-heuristics; simulated annealing (SA) and a genetic algorithm (GA). The authors have used random keys as a coding scheme for the solutions space.

It seems to be a tonic to tackle these problems with meta-heuristic approaches. Pongcharoen et al. (2002) has devised a design of experiments framework to optimize the GA parameters. They reported that the fine tune of the parameters was crucial for the success of the algorithm. The GA developed by Chen and Ji (2007a) has shown good results, the authors have used the same design of experiments of Pongcharoen et al. (2002) to optimize the parameter set. They have also used the random key codification scheme, and compared their results with a mixed integer programming (MIP) model formulation (Chen and Ji 2007b). Dayou et al. (2009) have developed a multi-objective MIP based on the model of Chen and Ji (2007b). The authors also devise a GA to solve the problem.

Mohammadi et al. (2010) has provided a mathematical model, and a relax and fix heuristic for the multi-product multi-level capacitated lotsizing problem with sequence-dependent setups. The authors developed two lower bounds, and compare their MIP-based heuristic against the optimal solution. Stadtler (2011) has developed a compact MIP-model for the same problem, except that there were no lead-times on the products. Karimi-Nasab and Seyedhoseini (2013) had augmented the model, in order to incorporate flexible machines with changing speeds. Yan et al. (2016) solve the multi-level capacitated lot-sizing and scheduling problem using particle swarm optimization.

The work of Chen et al. (2011) is the first, to our knowledge, to conduct a case study for the multi-level product scheduling encountered in a light source manufacturer. They have developed a GA, using random keys as a codification scheme. The steps of the GA are mainly based on previous approaches. More recently, Puongyeam et al. (2014) designed a Krill Heard (KH) based algorithm to solve the scheduling problem, the authors have also used real data to validate their approach, from the capital goods industry. Results showed that their algorithm had a better performance than other KH based methods. A mathematical model is proposed by Öztürk and Ornek (2014) for the basic problem, and then the authors provide an extension that incorporate sequence dependent setups, and transfer time between machines. The first author to propose a mathematical model for the multi-level scheduling integrated with preventive production maintenance is Chansombat et al. (2019).

Due to the high complexity of the scheduling problems, the majority of papers leans toward meta-heuristic methods, like GA and SA. For those algorithms to work properly,

the coding scheme is a crucial decision in the design of the method. An extensive review of coding schemes is presented by Cheng et al. (1996), although they are not bounded to scheduling problems, there are a variety of production problems that support RK implementation, (Hosseini and Al Khaled 2014) and (Hosseini et al. 2014)) for example. We believe that random-keys portrays a powerful way for representing multi-product scheduling solutions, but not yet well explored.

We have conducted a search on the "Web of Science" database, using the words "random key" and "scheduling", considering only journal published papers, from 1945 on. The query returned 38 papers, after removing 5 that did not addressed explicitly scheduling, the sample dropped to 33 papers. The papers histogram by year, and the percentage of used algorithms are showed in Figs. 1a and 1b, respectively.

The histogram suggests that researchers working with random keys on their papers are increasing, and the majority of them use the coding along with genetic algorithms. Only 3 of those papers, however, handles the multi-product scheduling. We believe that there is a lack of application of random keys in those problems, due to the decoding complexity, which could affect the overall efficiency of the algorithms.

We thus propose a method to speed-up the decoding of random key, based on two new bounds extracted from the multi-product tree structure.

Proposed framework

In this Section we present the steps through which our approach is constructed. The main idea behind this algorithmic framework is to serve as a layer for knowledge extraction and decision making; bridging the complex machine environment and the decision maker. This layer is built based on process mining techniques. A great advantage of process mining is its ability to prompt and readily capture and extract machine related data, and with their discovery algorithms, mine machine dependencies.

Our method provides a framework that extract all relevant multi-product scheduling information, based on event logs only. That would enable a quicker response for decision makers, in front of several productions setbacks; inoperative machines, abnormal rates of discarded parts, or even the stochastic behavior of manufacturing environments.

Figure 2 presents the main phases of the approach and how they relate to the information system and the production environment:

First, product information is gathered by machines in a collaborative way and sent to the factory information system. The data is then held in the form of event logs. These logs constitute the raw input element through which process mining techniques are able to unveil processes. The products demand serves as a direction to perform the log filtering: for each scheduling demanded product, there should be a data base filtering specific events of the product. Each one of these database segments should be individually input to the process mining algorithm.

A discovery algorithm is used to extract a Petri net that represent the production process, the seminal α-algorithm (Van der Aalst et al. 2004), for example. For each different product (for each database input) there should be a different Petri net. In Fig. 2 this is represented as the "Process Mining" arrow.

After the Petri net extraction, we use them as input to the discovery of the product-tree structures (also referred to as precedence, or dependency graph), represented as I in the Fig. 2. This first part only gathers the permissible machines and their dependencies. Phase II is responsible for determining what are the parts and sub-parts of each node of the tree, and also their respective quantities. The final information of the product trees regards their processing times; in this phase (III), we cope with the stochastic behavior of the process by collecting 3 different processing times for each machine, which will enable the different scheduling scenarios evaluation.

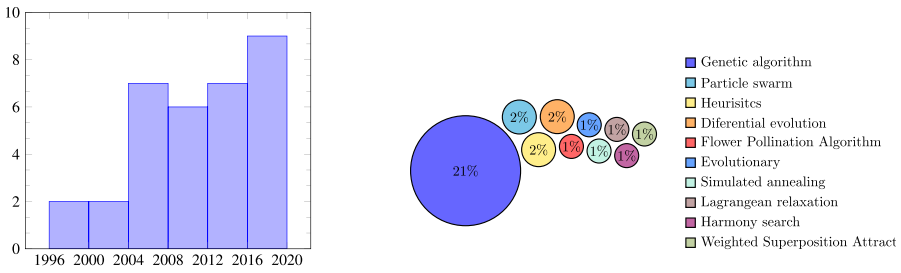
From here on, any scheduling engine (heuristic or mathematical program) that are able to cope with multi-level product could be used. We propose an improvement of a scheduling heuristic based on random keys, by extracting new lower and upper bounds for the solution decoding process, and a bisection method that reduces the complexity of the search. This phase is represented by arrow "IV" in Fig. 2. At the implementation level, we know that several phases could be mixed together, but for explanation purposes we decided to leave them separate.

On the following subsections, we provide detailed explanations for phases I, II, III and IV of the framework, as well as a few pre conditions and requirements that the database should meet, in order for the algorithms to work properly.

Data template

In order for the aforementioned framework to work properly, a few conditions are to be met. First, the information gathered by the machines should constitute an event log. Process mining algorithms require the minimal information of process ids, and time dimension (start and ending time of each activity) to be present on the log. For scheduling and scenario construction purposes, we also need, for each event, the names of the main product and its subpart that is being processed. Also, the required quantity and the actual number of produced parts. Table 1 shows a log fragment of the required information:

Lines 1 through 5 of Table 1 shows the events concerning the manufacturing of product A, by production order 1 (PO1).



(a) Scheduling with random keys - Histogram

(b) Scheduling with random keys - Algorithms proportion

Fig. 1 Random keys on web of science search

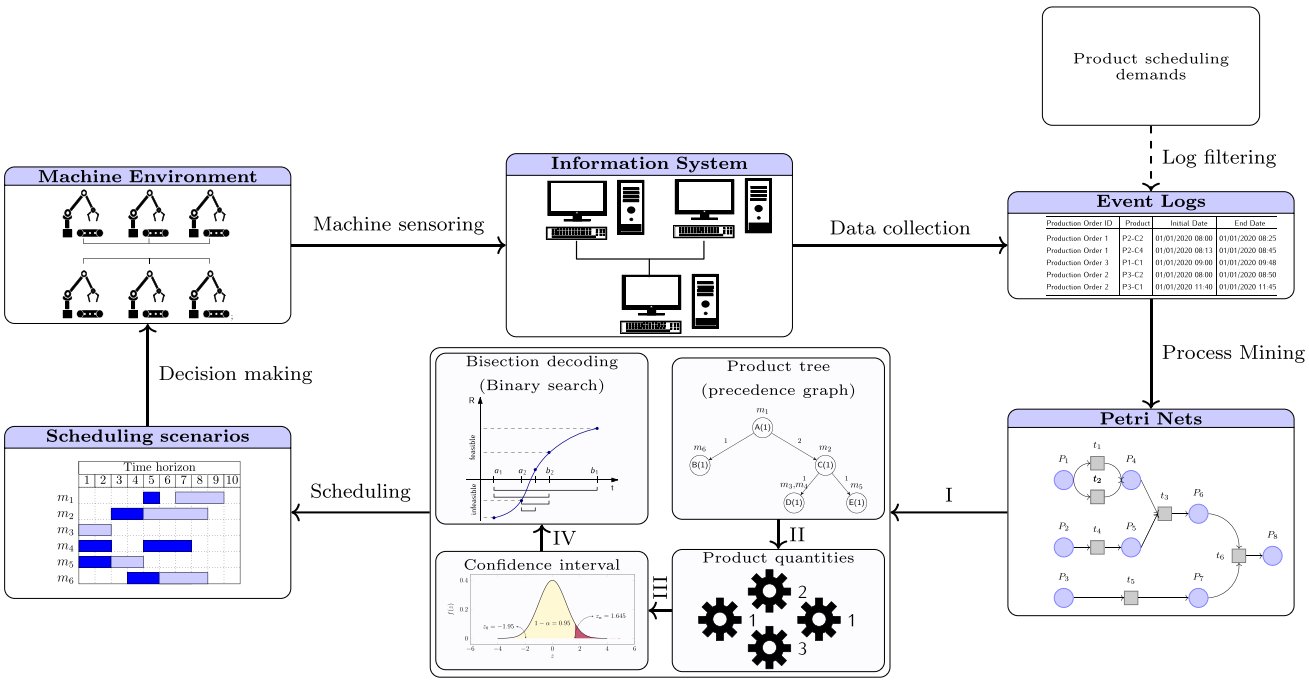
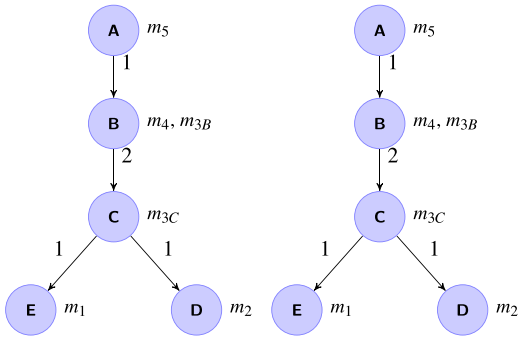


Fig. 2 Phases of the approach

Table 1 Event log fragment

PO ID	Prod.	Part	Qty.	Produced	Mach.	Start	End	**N mach.**
PO1	A	E	20	18	m_1	01/01/2020 08:20	01/01/2020 08:28	m_{1E}
PO1	A	D	20	19	m_2	01/01/2020 08:22	01/01/2020 08:43	m_{2D}
PO1	A	C	20	18	m_3	01/01/2020 08:45	01/01/2020 08:50	m_{3C}
PO1	A	B	10	9	m_3	01/01/2020 08:51	01/01/2020 09:10	m_{3B}
PO1	A	A	10	8	m_5	01/01/2020 09:10	01/01/2020 09:10	m_{5A}
PO2	A	E	20	18	m_1	02/01/2020 08:20	02/01/2020 08:28	m_{1E}
PO2	A	D	20	19	m_2	02/01/2020 08:22	02/01/2020 08:43	m_{2D}
PO2	A	C	20	18	m_3	02/01/2020 08:45	02/01/2020 08:50	m_{3C}
PO2	A	B	10	9	m_4	02/01/2020 08:51	02/01/2020 09:10	m_{4B}
PO2	A	A	10	8	m_5	02/01/2020 09:10	02/01/2020 09:10	m_{5A}



(a) Non-disjoint machine set **(b)** Disjoint machine set

Fig. 3 Phases of the approach

By the log, we see that A is composed of 4 sub-parts; B, C, D and E. Not all the planned components (Qtty.) were produced (Produced) for subpart E, an order was emitted for 20 parts and only 18 were finished. Column "Mach." contains the machines that processed the items.

The second requirement to use the approach is the machinery for each product; we assume that, for each manufacturing stage (that is, each subpart component) there is a giving set of machines available, consisting of a disjoint group of any other part of the same product. Subparts of different products, however, do not need to have disjoint machinery groups. Table 1 has 2 PO for product A (PO1 and PO2), but their components are processed on different machines, the product-tree of Fig. 3a contains all permissible machines (according to the log of Table 1), for all components of product A. At each node, the m_i represents the available machinery that the item (node letter) could be processed on, arcs between nodes represent dependencies, and their number the required quantity.

We see from Fig. 3a that component C can be processed by machine set $M_C = \{m_3\}$, whereas component B on set $M_B = \{m_4, m_3\}$. As $M_C \wedge M_B = \{m_3\} \neq \{\emptyset\}$, this configuration is not permissible, as they make a non-disjoint set. That machinery would yield a Petri net with unexpected behavior. So, in order to avoid that, a changing in the database is required: each machine belonging to joint sets, should be given another name (always mapping to the original machine). This is easily accomplished by the following procedure: first, filter all different products of the database. For each component of the product, create a new column with a machine that is a concatenation of the initial machine, and the produced subpart. This new machine creation is depicted on column "N mach." of Table 1.

This procedure guarantees the creation of non-disjoint machinery sets, but it could also happens that more than the necessary machines are created; in Table 1, for example, we would only use the new created machines m_{3B} and m_{3C}, although, it would have no problems in using all of them. Figure 3b shows the new product tree, with non-disjoint machines.

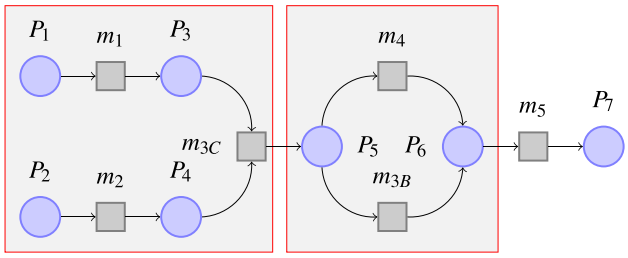


Fig. 4 Product Petri net

At any time, the new created machines could be mapped again to their original state. This transformation is important to extract the Petri nets, so the input to the process mining algorithm should have the column "N mach" as activities, and not the original "Mach". If those conditions are met, the extraction of product trees from Petri nets can be applied, we discuss the procedure on the following subsection.

I - Extracting dependency graph from Petri nets

As we mentioned, this first phase is designed to extract the product dependency graph from Petri nets. First we provide a definition of Petri nets, and their algebraic representation.

A Petri net can be defined as 4-tuple, Pn = (P, T, I, O) (Murata 1989), (Peterson 1981) where:

1. P = $\{p_1, p_2, ..., p_n\}$ is a set of places,
2. T = $\{t_1, t_2, ..., t_m\}$ is a finite set of transitions
3. I:T $\rightarrow$ P is an input function
4. O:T $\rightarrow$ P is an output function

Petri nets are able to represent discrete event systems with concurrency, a common characteristic of scheduling environments, where different product parts could be simultaneously processed. Figure 4 displays a Petri net with those characteristics:

The net of Fig. 4 represents the sequence through which a multi-level product undergoes: each place represent a product at some state (prior or after being processed), and transition are machines ((Zimmermann et al. 2001), (Saitou et al. 2002)). This specific product has a concurrency on machines m_1 and m_2 and a synchronization the parts, performed by machine m_{3C} (showed on the first box of Fig. 4). Machines m_4 and m_{3B} on the other hand, are representing a conflict (or choice), where either one of them can be used to complete the same product stage (second box). In this example, the procedure described on "Data template" section is used: that is, machine m_3 can be used on 2 processes, but it was duplicated into m_{3B} and m_{3C}.

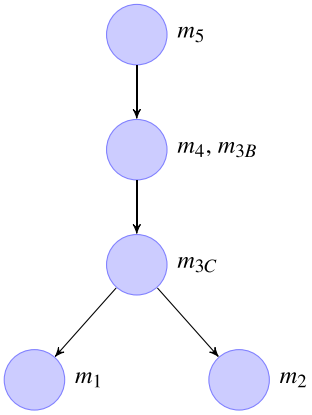


Fig. 5 Product structure from Petri net

From a Petri net, we are able to rebuild the multi-level dependency structure from a product. For example, the Petri net of Fig. 4 yields the dependency graph showed in Fig. 5.

Note that the Petri net provide the dependency graph, along with the permissible machine sets, but it is not possible to derive processing times and quantities only with the net, that would require the new discovered graph and the log (that is explained on further sections). In order to apply the algorithm that extracts the graph, we first need to transform the Petri net on its algebric notation, which facilitates the algorithmic development.

Besides the (P,T,I,O) definition, a Petri net can also be represented in a matrix form. In this form, the input and output functions are replaced by two matrices, D_p and D_m. Each matrix is m rows (one for each transition) by n columns (one for each place). The elements of the matrices are defined as follow:

- $d_p[i, j]$ is the number of tokens that transition i send to place j.
- $d_m[i, j]$, is the number of tokens that transition i consumes from place j.

The two matrices, D_p and D_m (as *plus* and *minus*) for the net of Fig. 4 are then:

$$D_m = \begin{array}{c} \\ m_1 \\ m_2 \\ m_{3C} \\ m_{3B} \\ m_4 \\ m_5 \end{array} \begin{array}{c} \begin{array}{ccccccc} P_1 & P_2 & P_3 & P_4 & P_5 & P_6 & P_7 \end{array} \\ \begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 \end{pmatrix} \end{array} \quad D_p = \begin{array}{c} \\ m_1 \\ m_2 \\ m_{3C} \\ m_{3B} \\ m_4 \\ m_5 \end{array} \begin{array}{c} \begin{array}{ccccccc} P_1 & P_2 & P_3 & P_4 & P_5 & P_6 & P_7 \end{array} \\ \begin{pmatrix} 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{pmatrix} \end{array}$$

The input and output matrices from the Petri net are the necessary inputs to our algorithm. A pseudo-code for the main routine is depicted in Algorithm 1. The main idea of the algorithm is to perform a search on the Petri nets (by its matrices), from backwards. That is necessary due to the product conditions; as it has to "end" at some point (the final product is the final place of the net), and as it follows a tree-structure, the end would be then the root of the tree, which would be the only possible place to start the search.

Algorithm 1: PetriNetTransform()

Data: Matrices $D_m[][]$ and $D_p[][]$, where each element $d_{ij} \in \{0, 1\}$
Result: List $\mathcal{L}$ of n, where n is a triplet: $(P_1, Lmachines = \{\}, P_2)$

```
L = {∅}; S = {∅}; n_0 = (∅, {∅}, ∅);
n_0.P1 = FindZeroColumnDminus(D_p);
S.add(n_0);
while S ≠ ∅ do
    n_i = S.pop();
    n_i.M = FindMachinesDplus(n_i.P_1, D_p)
    FindPlacesDminus(n_i, L, S, D_m)
RemoveFirstPlaces(L);
return L;
```

The output of the algorithm is the list of triplets $\mathcal{L}$. The elements of the triplet n are:

1. P_1: Integer representing a place (column number, for example, P1 = 3 is the same as the third column of matrix D_m- P3).
2. M: Set of integers representing transitions/machines (row number, for example, M = {1,3} represents M = {m_1, m_{3C}}).
3. P_2: Integer representing a place.

Each P_1-P_2 is an arc of the final dependency graph, and the set M are the permissible machines for the respective P_1. This structure is then easily restored to an adjacency list, or incidence matrix that represents the product tree. The list S has the same structure of the final output $\mathcal{L}$, but it is used to

control the searched nodes of the Petri net. Line 1 of Algorithm 1 initializes both lists, and a n element with null values. The routine $FindZeroColumnDplus()$ searches each column of matrix D_p, and returns the one where elements sums up to zero. That indicates a place that serves as input for no transition, that is, the final product, so the search starts at this point.

Lines 4 through 7 performs the algorithm main loop: the first triplet of S is removed (its P_1 element is already filled), and a search is conducted on $FindMachinesDplus()$ and $FindPlacesDminus()$ routines.

The $FindMachinesDplus()$ (Algorithm 2) is responsible to collect all machines permissible to P_1 to be processed on. That is conducted on each row of column P_1 on matrix D_p.

Algorithm 2: FindMachinesDplus(P_1, D_p)

Data: integer P_1, matrix $D_p[][]$ where each element $d_{ij} \in \{0, 1\}$
Result: List M of integer, each one representing one machine

```
1 M = ∅;
2 for i = 1 to D_p.rows do
3     if D_p[i][P_1] == 1 then
4         M.add(i);
5 return M;
```

With the P_1 and the list of machines M the last stage of the loop is to connect the dependency with its immediate descendants (P_2). That is achieved on $FindPlacesDminus()$ routine (3). In this routine, the list S has new elements added and the final output list $\mathcal{L}$ also has its elements inserted.

Algorithm 3: FindPlacesDminus(n_1, $\mathcal{L}$, S, D_m)

Data: n, where n is a triplet: (P_1, $Lmachines = \{\}$, P_2), $\mathcal{L}$ and S, lists of n, matrix $D_m[][]$ where each element $d_{ij} \in \{0, 1\}$

```
1 m = n.M.front();
2 for j = 1 to D_m.columns do
3     if D_p[m][j] == 1 then
4         n_current ← n;
5         n_current.P_2 = j;
6         L.add(n_current);
7         n_new = (j, ∅, ∅);
8         S.add(n_new);
```

After the main loop of Algorithm 1, considering the illustrative Petri net of Fig. 4, the list $\mathcal{L}$ contains the following elements: $\mathcal{L} = \{(7, \{6\}, 6), (6, \{4, 5\}, 5), (5, \{3\}, 3), (5, \{3\}, 4), (4, \{2\}, 2), (3, \{1\}, 1), (1, \{\emptyset\}, \emptyset), (2, \{\emptyset\}, \emptyset)\}$

Which provide the arcs of the dependency graphs, and also the two "beginnings" of the product. As the Petri net represent its states as pre-processing (place before the transition) and post processing (place after transition), at some point there would be generated a surplus place that do not map directly to some product part. The two last elements of the list L are these places, both with the $\emptyset$ for two elements. The last phase of the algorithm is to remove those places, that is done by routine $RemoveFirstPlaces()$ (line 8 of Algorithm 1). The entire list is searched for $\emptyset$ values, and those elements are simply removed.

The output becomes then:

$\mathcal{L} = \{(7, \{6\}, 6), (6, \{4, 5\}, 5), (5, \{3\}, 3), (5, \{3\}, 4), (4, \{2\}, 2), (3, \{1\}, 1)\}$

The list $\mathcal{L}$ is easily converted to an adjacency list or incidence matrix. On the next section we provide details for the calculations of subpart quantities.

II - Product quantities and labels

After the extraction of the product tree from the Petri nets, as showed in "I - Extracting dependency graph from Petri nets" section, the next step is to provide the labels of product parts and their required quantities. From Phase I, the product-tree dependencies are linked by permissible machines, as indicated in Fig. 5.

As mentioned in "Data template" section, for each product, if there were shared machines among its subparts, there should be a mapping to new machines. This process guarantees that each pair (component,machine) is unique, so the labels of each place (the name of each node of the product tree) can be gathered by replaying each trace. Considering the product tree extracted on the last subsection, in the form of list $\mathcal{L}$, and after the mapping of the M list of integers to the machine names (lines of matrix D_m), the list $\mathcal{L}$ becomes:

$$\mathcal{L} = \{(7, \{m_5\}, 6), (6, \{m_{3B}, m_4\}, 5), (5, \{m_{3C}\}, 3), (5, \{m_{3C}\}, 4), (4, \{m_2\}, 2), (3, \{m_1\}, 1)\}$$

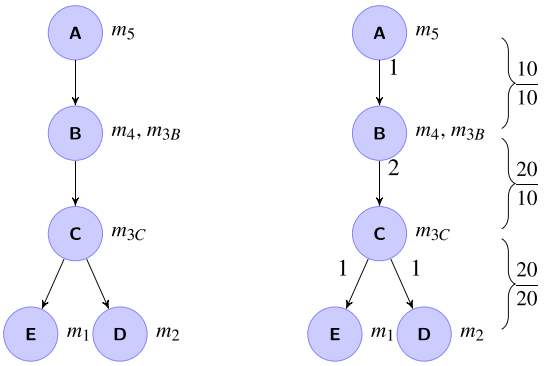
Now, a simple trace replay is performed, and the information is linked to the nodes through the machine mapping. Replaying all events of the log presented in Table 1, the labels attached to the machines of list $\mathcal{L}$ are:

$$\mathcal{L} = \{(7, \{ \overset{A}{m_5} \}, 6), (6, \{ \overset{B}{m_{3B}}, m_4 \}, 5), (5, \{ \overset{C}{m_{3C}} \}, 3), (5, \{ \overset{C}{m_{3C}} \}, 4), (4, \{ \overset{D}{m_2} \}, 2), (3, \{ \overset{E}{m_1} \}, 1), \}$$

This procedure enables the node labeling for the product tree. Figure 6a shows the nodes with their product names.

In the same manner, all relevant information can be attached to the respective node, like the ordered product quantity, and the actual produced. On the next Section this same procedure is used to gather the processing times, and construct confidence intervals for the means.

Once the dependencies are know, the tree can be used along with the log to gather the quantities required on each subpart production. For each arc (p_1, p_2) of the tree, the



(a) Labeled product tree **(b)** Product tree with quantities

Fig. 6 Product quantities and labels

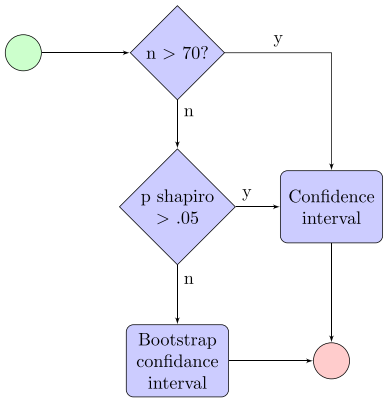


Fig. 7 Decision flow to collect time bounds

required quantity is given by the following formula (basically a bill of material explosion):

$$Qt_{p2\rightarrow p1} = \frac{Qt_{p2}}{Qt_{p1}} \tag{1}$$

Where $Qt_{p2\rightarrow p1}$ is the quantity required of product p_2 to produce one unit of p_1. Qt_{p2} and Qt_{p1} are the quantities presented on log of components p_1 and p_2.

Figure 6b shows the calculation of Eq. 1 for arcs (A,B), (B,C) and (C,D).

The last component of the tree is the processing times, presented on the next subsection.

III - Time confidence interval and scheduling scenarios

In this Section we describe how to collect the processing times of the machines. Every production process has a stochastic component describing their processing times. As we are using process mining techniques, we are able to perform prompt machine knowledge extraction, which enable us to use this stochastic behavior to improve scheduling results.

We propose the gathering of 3 processing time values for each product, optimistic, realistic and pessimistic (t_o, t_r, t_p). The method used to collect the values can be one of two; a mean confidence interval or a bootstrap confidence interval; the decision depends on the data. Figure 7 displays a decision chart that guides on what method should be used.

Basically, the flowchart checks the assumptions that the data should have, in order for the confidence interval to be constructed. If the conditions are not met, the values are collected by the bootstrap interval. The first decision concerns the sample size (n), if it has more than 70 elements, we make use of the central limit theory [1], and the confidence interval can be applied. Otherwise, we check data normality by applying a Shapiro-Wilk test (Shapiro and Wilk 1965), and evaluate its *p-value*, under the null hypothesis H_0 that the sample was drawn from a normal distribution. If the *p-value* is greater than .05 (α = .05), the normality condition is met, and the confidence interval can be constructed, otherwise we use the bootstrap interval.

Regardless of the method used, we need to gather the sample of processing times for each machine. That is easily done with the process-log and machine mapping explained in "II - Product quantities and labels" section. In the same way, a log replay is performed for each product tree, using all traces available on the database for the product, but now the attached information is their processing times.

The *mean confidence intervals* states that (1 - α)% of the processing times would fall on the interval. The α value is typically set to 5%. As we do not known the population mean μ nor variance σ^2, the interval is constructed according to the *t-student* distribution. The confidence interval for the sample mean $\overline{X}$ with significance level α CI($\overline{X}$,α) is then:

$$CI(\overline{X}, \gamma) = \overline{X} \pm \ t_\gamma \sqrt{\frac{S}{n}} \tag{2}$$

Where $\overline{X}$ is the sample mean, t_γ is the t-statistic where P($-t_c < t < t_c$) = 1 - α with $n - 1$ degrees of freedom. S is the sample variance, given by:

$$S = \frac{1}{n-1}\left(\sum_{i=1}^{n} x_i^2 - n\overline{X}^2\right) \tag{3}$$

We use the values of the interval to construct the scheduling scenarios: $t_o = \overline{X} - \ t_\gamma \sqrt{\frac{S}{n}}$, $t_p = \overline{X} + t_\gamma \sqrt{\frac{S}{n}}$ and $t_r = \overline{X}$

[1] The central limit theorem states that the sum of n independent and identically distributed random variables is approximately normally distributed (Montgomery 2017). At some cases this approximation is good even for small n ($n \leq 10$), whereas in some cases a large n is required ($n \geq 100$)

Table 2 Data summary

	Method	
Collected value	Bootstrap CI	CI
l_o	$\overline{X} - \delta^*{}_a$	$\overline{X} - t_\gamma \sqrt{\frac{S}{n}}$
l_r	$\overline{X}$	$\overline{X}$
l_p	$\overline{X} + \delta^*{}_b$	$\overline{X} + t_\gamma \sqrt{\frac{S}{n}}$

If the the assumptions to construct the parametric confidence interval are not met, we construct the bootstrap confidence interval (Efron 1992). The bootstrap confidence interval is based on the resampling idea. The first step is to create multiple samples of the dataset (with same size with replacement). Each new resample has a mean value $\overline{X}^*$. To construct confidence interval, we need to know how much the distribution of $\overline{X}$ varies around μ:

$$\delta = \overline{X} - \mu \tag{4}$$

The bootstrap principle states that we can approximate δ by:

$$\delta^* = \overline{X}^* - \overline{X} \tag{5}$$

The next step then is to compute all the δ^* values of the resamples, and then sort them in non-decreasing order. The bootstrap confidence interval is then:

$$[\overline{X} - \delta^*{}_a, \overline{X} + \delta^*{}_b] \tag{6}$$

Where a and b are percentiles of the collected δ^* values; they are the equivalent to α on the parametric confidence interval. For example, suppose we had 20 sorted δ^* values, a bootstrap confidence interval of 90% would imply $a = 5\%$ and $b = 95\%$, so $\delta^*{}_a$ is collected as the first value of the ordered δ^*s, since it is the 5 percentile.

Table 2 summarizes the gathering information for both intervals:

Those values are to be attached at each node of the product tree, Fig. 8 shows an example with attached values:

The rationale behind the different times are as follows:

1. *Optimistic* In an optimistic scenario, the processing times are as low as possible, hence using the lower values of the confidence intervals.
2. *Realistic* In an realistic scheduling scenario, the value used is equivalent to the mean processing times. That would yield the most historically "probable" processing times.
3. *Pessimistic* As expected, in a pessimistic scenario, all machines process their respective products using as long a time as possible, that is, the upper value of the intervals

In this way, industrial managers are better equipped to deal with different types of clients and suppliers; when bargaining a new client due date, managers could use the pessimistic scenario of scheduling, which would hardly be extrapolated, hence providing the client with a secure delivery date. On the other hand, the manager may use the optimistic scenario when dealing with suppliers; after optimizing the scheduling for a given time period, the optimistic scenario would implicitly provide the early date for material consumption. Then critical material could be planned and ordered accordingly, mitigating the risk of production halt due to lack of material. The different scenarios aims at minimizing a disruption effect that may occur on the future, which is a concept often encountered on predictive scheduling Yang et al. (2020).

At this point, all relevant information to perform the scheduling is attached to the product tree, so a mapping to the original machines can be done. Figure 8b shows the final product tree: the original machine remapped (m_{3B} and m_{3C} become m_3), and for each machine the interval values for scenario construction are depicted in green, yellow and red colors.

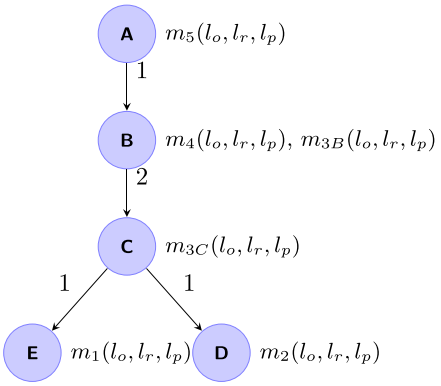
Figure 9 depicts a possible scheduling output for the product tree of Figure 8b, considering all scenarios and a client due date set to period 10.

With the 3 scenarios at hand, an industrial manager is able to make more accurate decisions. Considering the client due date, we see from Fig. 9 that in 2 of 3 scenarios the due date is not achieved (Fig. 9b and c), in the realistic scenario there would be 2 days of delay, and 8 on the pessimistic. So it would be a reasonable choice to contact the client for a possible date postponement. Also, the manager has a better bargaining power, knowing that in the worst case scenario, a plausible due date would be at time period 18.

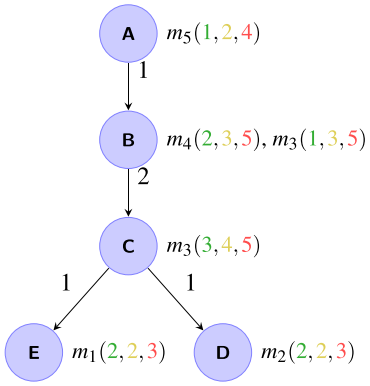
The scheduling engine should be robust enough to perform fast optimizations, given that for each "normal" procedure, the engine should be run 3 times, one for each scenario. In the following subsection we provide with an acceleration method to decode scheduling solutions that are coded as random keys. This codification can be used to several meta-heuristics. We have implemented a genetic algorithm, although the GA implementation details are not showed in this paper.

IV - Scheduling heuristic and bisection acceleration method

In this Section we present the main components of the scheduling solution representation scheme, as well as a new method to speed the decoding solutions. The procedure is

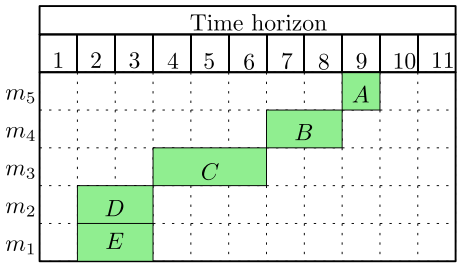


(a) Product tree with confidence interval

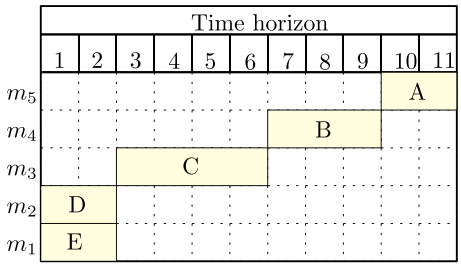


(b) Scenario tree

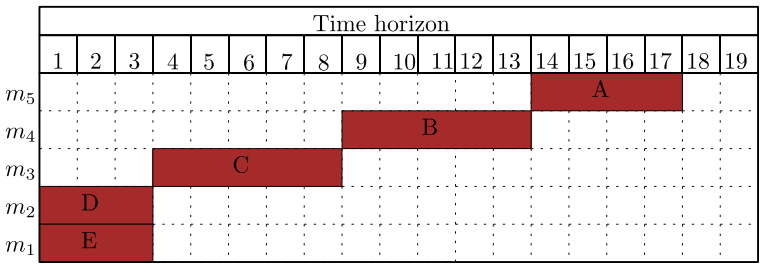
Fig. 8 Time interval product tree



(a) Optimistic



(b) Realistic



(c) Pessimistic

Fig. 9 Scheduling scenarios

based on the bisection method (binary search) (Burden and Faires 2004) for the root-finding problem. It is worth noting that this method could be applied to any heuristic procedure that handles the same coding scheme to perform its search. In this paper we have implemented a genetic algorithm (GA) based on Kim and Kim (1996).

One common characteristic of GAs is that they operate on *coding* and *solution* spaces, as showed in Fig. 10.

The evolution takes place in the coding space, whereas evaluation in the solution space. For a chromosome to be evaluated (by some objective function), it first needs to be decoded to a solution, and vice versa. How to encode and decode solutions in genetic algorithms are then questions of the utmost importance, and could determine the success or

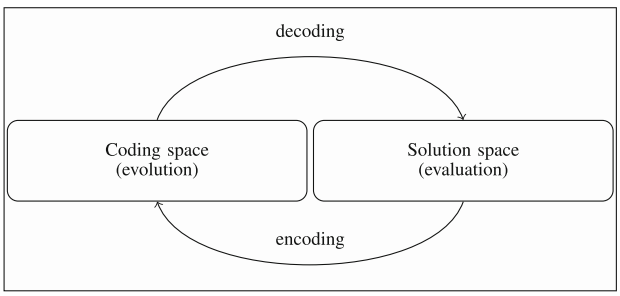


Fig. 10 Coding and solution spaces

demise of the procedure. One critical issue of encoded solution regards their feasibility; there could be encoded solutions

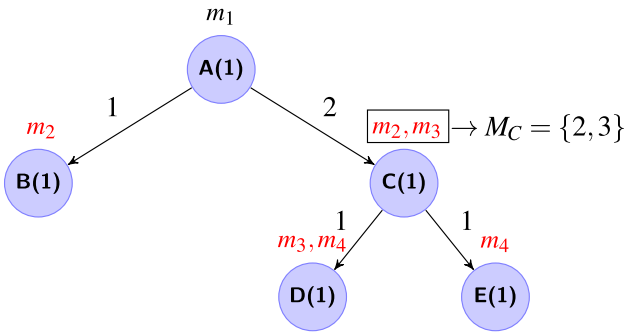


Fig. 11 Example product tree

that, when mapped to the solution space, become infeasible and therefore needs repair. Also, there are encoded solutions that has their feasibility guaranteed upon the mapping procedure.

There exist many forms of representing encoded solutions (Cheng et al. 1996), in this paper we use the random-keys representation. In this particular case, we follow the ideas presented in Kim and Kim (1996) (although the authors do not provide a generic algorithm for multi-level scheduling, as they deal only with two level products). The decoded solution is always feasible, which requires a lot of computational time; we propose an accelerated method for decode feasible solutions, based on random keys (RK).

A RK chromosome representation, firstly proposed by Bean (1994), is encoded as a vector with as many elements as the components of the product tree. Each element consists of two parts; an integer that represents the machine that will process the component, and a fractional random part $\in (0, 1)$, serving as a sorted list determining the order of loading parts in machines. We demonstrate the random key coding dynamics using the product tree of Fig. 11, as it encompasses two parallel structures and shared machines.

A possible RK codification for this tree is:

$$\begin{array}{l} \quad\ \ \text{A} \quad\ \ \text{B} \quad\ \ \text{C} \quad\ \ \text{D} \quad\ \ \text{E} \\ \text{RK} = [\ 1.22\ ,\ 2.57\ ,\ \ 2.98\ ,\ \ 3.72\ ,\ \ 4.05\ ] \end{array} \tag{7}$$

Each letter above the vector elements are symbolizing the respective sub-parts. Each fractional part is a random number between 0 and 1. Each integer part is a random number among the permissible machine set of the specific part; for instance, the permissible set for product A is $M_A = \{1\}$ (the only possible random integer for A is 1), for C is $M_C = \{2, 3\}$ (the random integer can be chosen only among 2 and 3). The decoding process of some RK requires the product dependency graph, demanded quantities and due date. The process of loading parts to machines is performed in a backward fashion: starting from the due date, the final product is loaded, then their first order components, and so on, until all products are loaded or the schedule is unfeasible. In this case, the due date has an increase of 1 unit, and the decoding is performed again, this process is repeated until a feasible schedule is decoded. An example of the decoding process with multi-products is explained in details on Appendix A.

The decoding process exemplified on Appendix A has generated a feasible solution on the its first attempt, which does not always happens. If the product due date was set to time period 6 instead of 9, all schedule were to be push back 3 time units. That would result in sub parts D and E starting on period -1, which is clearly unfeasible. If that were the case, the next step would be to increase the due date from 6 to 7 and start the decoding again, this would be repeated until the first feasible schedule is obtained. This is the most time consuming routine of the algorithm, we propose a speed-up procedure for the feasible decoding, based on the bisection method. The procedure relies on two main concepts, described below; the *due date upper* and *lower bounds*.

Definition 1 Any time period T_u, for which, when considered as a RK due date, is guaranteed to provide a feasible schedule when decoded, is called a due date upper bound.

Definition 2 Any time period T_l, for which, when considered as a RK due date, is guaranteed to provide an unfeasible schedule when decoded, is called a due date lower bound.

The bisection method we propose performs a search for a feasible decoding on the interval between T_l and T_u, without checking every time period. The problem of finding the T_l is very relevant, since the client initial due date may not be possible on the best case scenario, if this case is spotted prior to the algorithm execution, it could save computational efforts.

We derive the T_l based on the Material Requirement Planning (MRPI). The seminal work of Orlicki (1975) introduces the MRPI, a method to calculate the components necessity over a given time period (considering precedence constraints). The MRPI does not consider any resource information (machinery), so in a way, it can be seen as a scheduling with infinite capacity. In our case, that can be seen as; whenever a parallel processing is permissible according to the product tree, there is a machine available to perform it. That is the best case scenario to schedule the product, if one time unit is removed from the due date, it becomes infeasible, and therefore a *due date lower bound* T_l.

The T_u, on the other hand, is derived considering what would be the worst case scenario; if there is no parallel activities at all (represented by only one machine that has to process all the sub parts). All product parts being processed on one machine is the worst case scenario, so it is guaranteed to be feasible, and therefore a *due date upper bound* T_u.

Now, with the founded bounds it is possible to use the bisection method to accelerate the search, Algorithm 4 shows the main routine:

Algorithm 4: FindDecodeBisection()

Data: Integers T_l, T_u, T_c and Δ, vector RK of reals, product dependency graph G
Result: Integer t of first feasible due date for the product scheduling

```
1  if T_l > T_c then
2      a = T_l
3  else
4      a = T_c
5  b = T_u
6  while (b − a) ≥ Δ do
7      t = ⌊(b − a)/2⌋;
8      flag = DecodeRandomKey(p, G, RK);
9      if flag == true then
10         b = p;
11     else
12         a = t;
13 flag = false;
14 while flag == false do
15     flag = DecodeRandomKey(p, G, RK);
16     t ++;
17 return t;
```

Lines 1 through 5 of Algorithm 4 sets the interval limits, in this case, there should be a verification of the T_l, if it is smaller than the client due date (T_c), than it is not used on the interval. Lines 6 through 12 perform the binary search; at each iteration of the while loop, the interval is narrowed to its half, that is performed until the interval is $\leq \Delta$ (a provided). After the interval has been narrowed to Δ, an incremental search is performed until the first feasible decoding is made (lines 13–16).

The usual decoding scheme, presented in Bean (1994) does not provide with a time interval, as it does not have lower and upper bounds. Considering that our bounds were to be used, and that $n = T_u - T_l$, the decoding of Bean (1994) has a complexity of $O(n)$. If our proposed binary search is applied, considering the same bounds, the complexity becomes then $O(\log n)$. When used on a search heuristic, as a GA, where an enormous amount of decoding is to be made, the complexity reduction proposed produces a significant time decrease on the overall procedure. That reduction make it manageable to run the algorithm 3 times for each scheduling (in order to produce the multiple scenarios, presented in "III - Time condence interval and scheduling scenarios" subsection). On the next section we provide some computational experiments on the bisection decoding algorithm.

Computational results

In this section we present computational results considering the random keys bisection method. As the decoding needs to be performed a number of times (depending on the meta-heuristic applied), we studied the total time behavior of the decoding when applied to a genetic algorithm. We chose the GA as it has many parameters that influences the total number of decoded solutions, and an overview of time differences in this complex search space may provide a good estimate of time savings for other meta-heuristic approaches.

Since we are interested in quantify the speed performance of algorithms, we did not provide a comparison of scheduling objective functions, as we use the same GA for both random keys decoding schemes; which implies that the scheduling outputs were the same, differing only by their required computational time.

We have tested a GA on the product tree of Fig. 12. The tree has 87 components in total, and a maximum of 12 sub-part levels. The arcs have no quantity labels attached due to the picture space, but every quantity and processing time was set to 1 unit. Our scheduling comparison was conducted under the following demand: 10 units of product 0 and 10 units of product 6. Besides the GA parameters, one particularly interesting parameter is the due date of the demand; as the bisection decoding algorithm speeds up the process when the first decoding is not feasible. If the due date is far enough on the planning horizon as to be feasible on the first try, both decoding schemes are expected to have the same performance.

We have compared the results [2] of those demands for different due dates, and, from the GA algorithm we tested different values of the population size and total generations. The plots of Fig. 13 show the results.

On the plot, "NB" stands for Not Bisection where "B" stands for Bisection. The x axis varies the due date while the y axis is the total processing time in seconds. We see from the plot that for every set of GA parameters, as the delivery due date approaches 0, the NB time grows abruptly. That behavior is expected, as we know that a feasible decoding is more likely to be found on the first attempt, when the due date is far on the planning horizon.

Our decoding has a time increase practically constant while due dates approach 0. The population size does not appear to affect the decoding times, however, we note that when the population size grows, decoding times seems to follow the trend.

With this increase, we conclude that it is not viable to use the usual decoding, specially with more complex client demands and a manufacturing landscape, that most likely would already have loaded sub-parts being processed on machines, by the time of new schedules optimization.

[2] Computational tests were implemented in the C++ programming language and ran on a Ubuntu 18.04.4 server machine, with 1gb of RAM memory

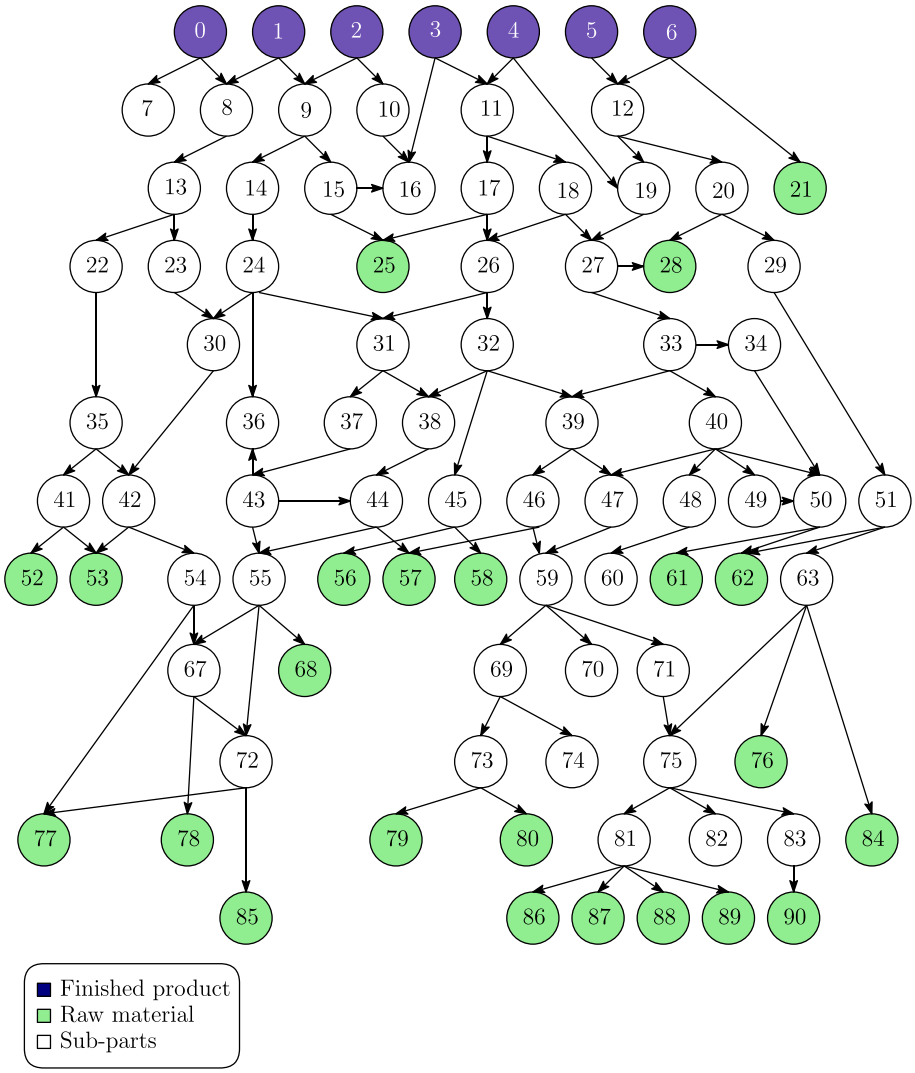


Fig. 12 Test product tree

Conclusion

In this Section we conclude the paper, first by discussing some know limitations of the proposed method and our future research works. The last subsection is dedicated the implications and final remarks.

Drawbacks and future works

One known limitation of the work regards the log template. As presented in "Data template" section, the event log should have a predefined form, in a way that process-discovery algorithms like alpha-miner could be applied. Our Petri net mining algorithm works for nets without hidden-transitions, that is, a system with at least one transition that is not directly representing an activity on the real process.

If that is the case, our Petri net mining algorithm would insert sub-parts on the product-tree that does not exist in reality. The second point is the scope of the binary search, proposed in "IV - Scheduling heuristic and bisection acceleration method" section. The method is applicable to any meta-heuristic that would perform the scheduling, but the solution codification scheme should be with random keys.

This work enables an almost effortless scheduling engine output. By using only a machinery log, all relevant information can be gathered and the multiple scenarios evaluated. For our future research work, we aim at using this framework on a pilot SME which does not yet have a scheduling engine. For that, we would use the newly proposed Petri net extraction algorithm and the bisection decoding method.

As the bisection scheme applies to any meta-heuristic framework that uses random keys as its coding scheme, we would compare 3 meta-heuristics; tabu search, simulated

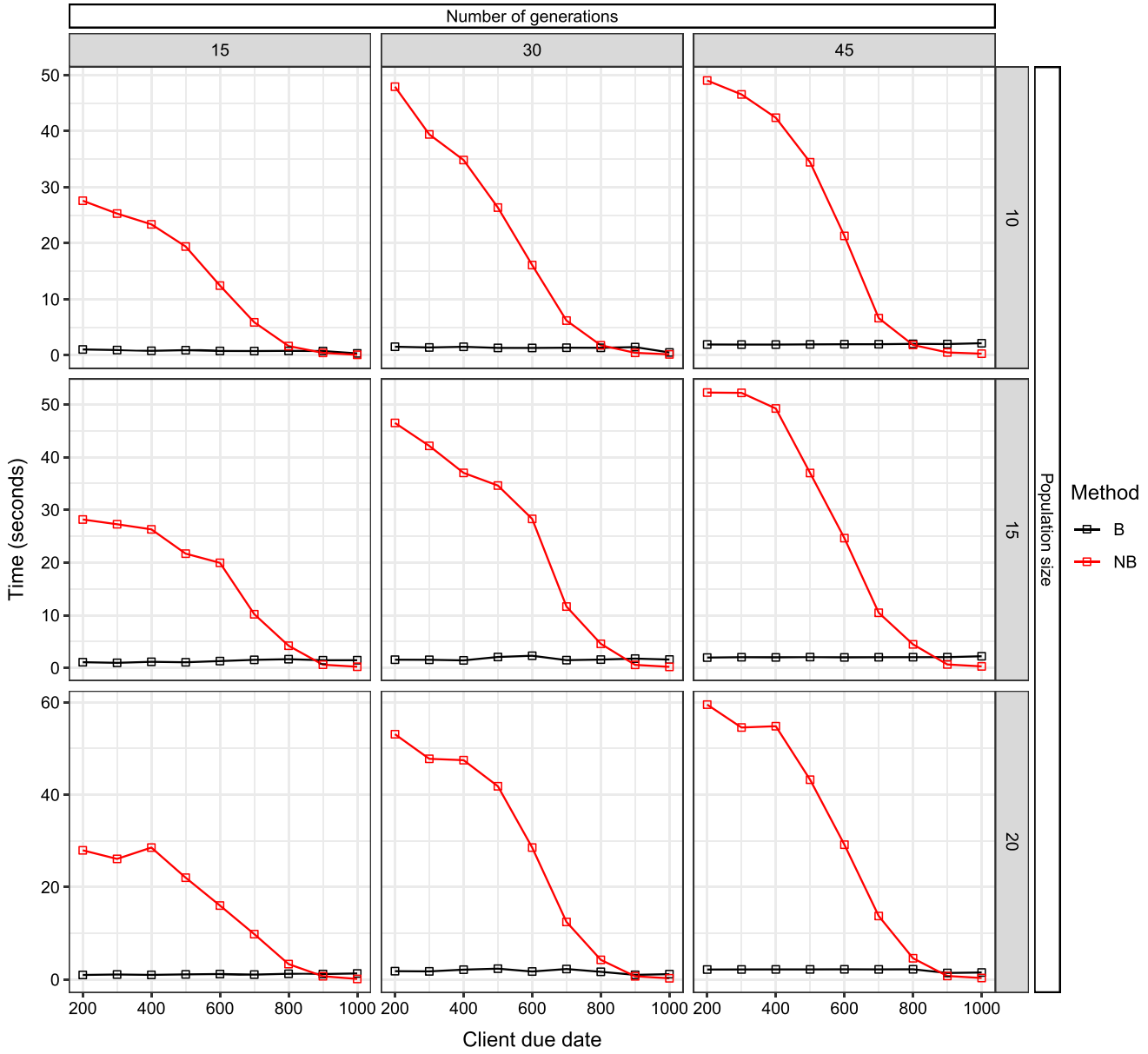


Fig. 13 Result plots

annealing and the genetic algorithm. Also a refined version of the scenarios would be to use statistical methods to predict the processing times, as suggested by Sobaszek et al. (2019).

Implication and final remarks

In this paper we have proposed an integration of the scheduling of multi-level products with machine data gathered on the production floor. This bridge was supported by process-mining techniques, which are able to promptly extract process models from event data, and to perform a series of analytical conjectures, like simulation-based analysis, recommendation services and process conformance checking. The process mining tools are adequate for SMEs to incorporate, as the learning curve to use them is very accentuated and there are high quality open source available software that can be integrated into an existing information systems.

As stated by Moeuf et al. (2018), industry 4.0 tools are easily implemented by SMEs, as it decentralizes information. In this way, the collection of data are not a problem for these companies. Our method uses process mining then, to extract models from these data in the form of Petri nets, a well suited language for computation, as they have a sound mathematical background and can be expressed in algebraic matrices forms. With the Petri nets describing the production floor processes, along with the event log, we propose an algo-

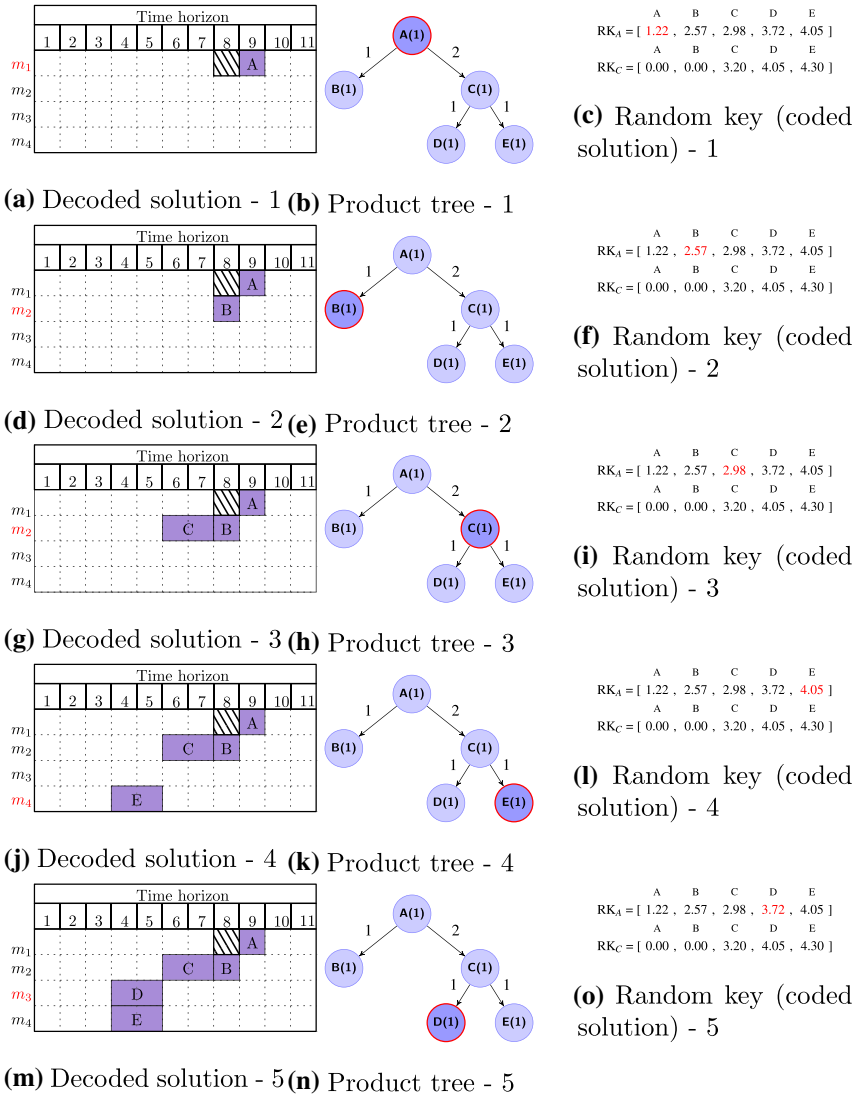


Fig. 14 Random key decoding example product A

rithm that extract the multi-level product tree (dependency graph) for each product, as well as the relevant information to perform the scheduling.

These information consist of; quantities of each sub-part on the tree, the set of permissible machines and the processing times of products on each machine. In order to deal with the stochastic behavior of the manufacturing environment, we propose the collection of 3 values by the construction of time intervals for the mean processing times, and an association of different scenarios for each value; optimistic, realistic and pessimistic.

When the scheduling is performed, production managers and practitioners would have 3 possible scenarios at hand, instead of just the one, and according to different client or supplier needs, a different production plan could be considered. In order to perform 3 different schedule outputs for each manager input, the optimization engine should be fast enough as to not compromise time restrictions; we then provide an acceleration method to decode solutions based on random-keys. This representation is a common choice for meta-heuristic scheduling algorithms, but not yet well studied for the multi-product type. We propose two bounds based on the product tree, and a procedure that reduces the complexity of the usual decoding, based on the bisection method. This method has reduced the complexity of the classical decoding from $O(n)$ to $O(\log n)$.

As an implication of our work, SMEs that does not yet posses a scheduling engine are now able to build one from scratch, with little efforts regarding systems and databases integration; by only using the log generated by machines, they are able, through the use of process mining, to perform scheduling with multiple scenarios, giving production managers a response answer for different customer demands and supplier restrictions.

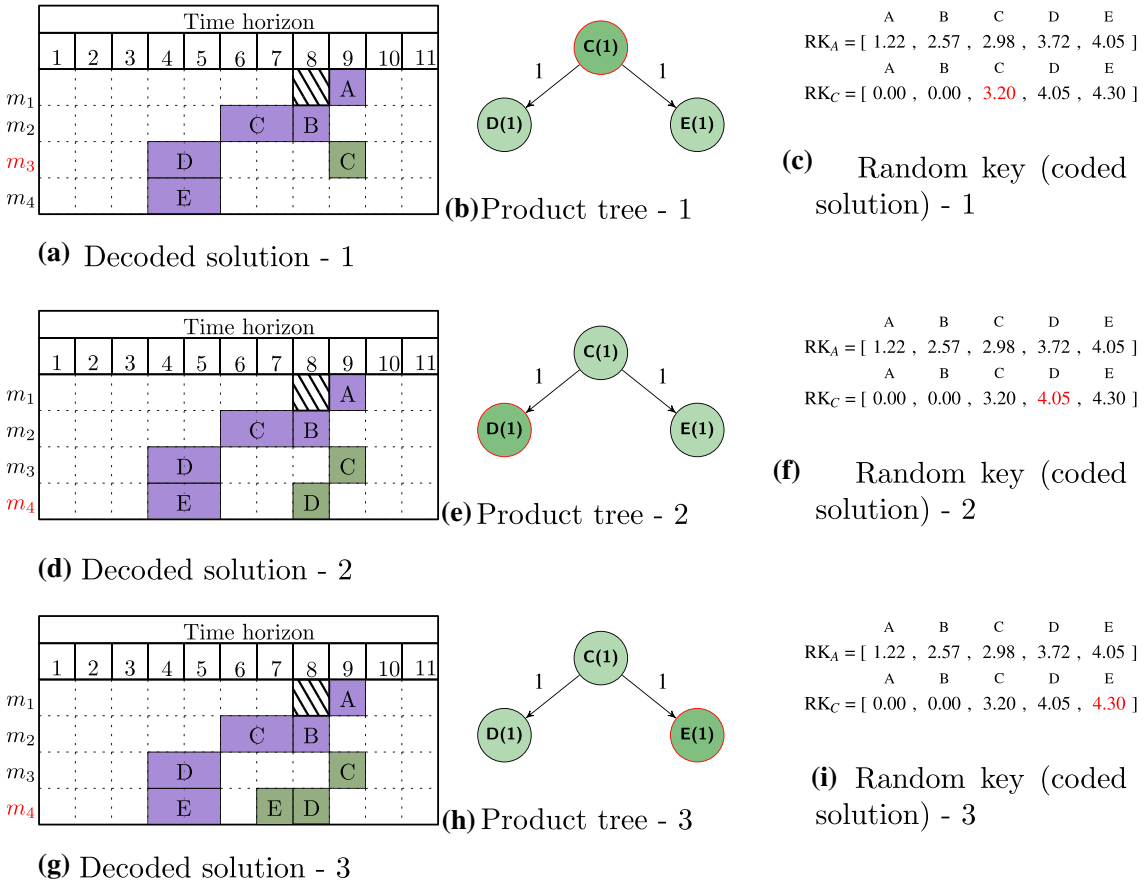


Fig. 15 Random key decoding example product C

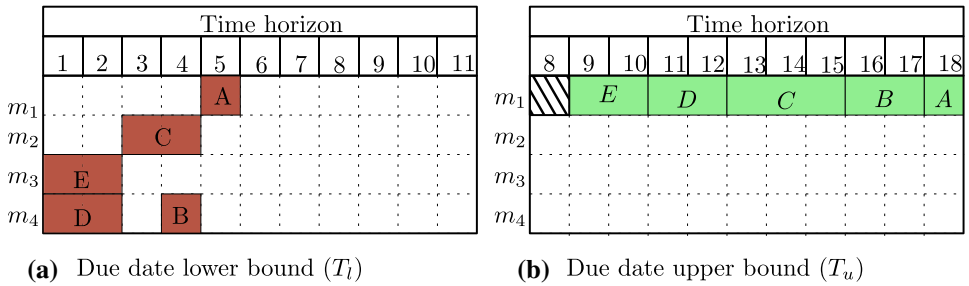


Fig. 16 Due dates lower and upper bounds

The main goal of this study is to provide a framework that facilitates complex optimization decision, as the multi-level structure, for SMEs, on a simplified way, that does not require big and expansive software solutions, as commercial ERPs often does. Our framework incorporates machinery data into a process intelligence view, resulting in concrete optimization solutions for managers and practitioners.

Funding This work was supported by CAPES (Coordenação de Aperfeiçoamento de Pessoal de Nível Superior).

A Appendix: Random key decoding example

In this Appendix we show an example of the RK decoding process. As expected, in a production scheduling, there may exist more than one product that needs to be allocated on machines, therefore, we build a decoding example for 2 products, on a set of machines that has already some parts being processed.

As we are decoding 2 products, there must exist 2 RKs, one for each product. It is important to note that the order of decoding of the RK may change the schedules; we start by decoding products with earlier due-dates. In this case, as both products are due to time period 9, we choose randomly

between then. The RKs for the example are:

$$RK_A = [\ \overset{A}{1.22}\ ,\ \overset{B}{2.57}\ ,\ \overset{C}{2.98}\ ,\ \overset{D}{3.72}\ ,\ \overset{E}{4.05}\ ]$$

$$RK_C = [\ \overset{A}{0.00}\ ,\ \overset{B}{0.00}\ ,\ \overset{C}{3.20}\ ,\ \overset{D}{4.05}\ ,\ \overset{E}{4.30}\ ]$$

Note that the values for products A, B and C were set to 0 on RK_C, as they are not used, but for implementations purposes we choose to have the RKs as a square matrix. Assuming a demand of 1 unit of product A (of Fig. 11) and also 1 unit of product C. The delivery deadline for both products is on time period 9, and there is one remaining part already scheduled on machine m_1 on time period 8.

As mentioned, the decoding is carried out on a sequential manner. Figure 14 shows the decoding process of product A and Fig. 15 of product C.

The process starts with product A (following the product tree), to be processed on machine 1 starting on time period 8. Note that here, only the integer part of the RK was used, since there is no decision (the only part that can be loaded, according to the precedence relations, is A). After A is scheduled on machine m_1, both sub parts B and C can be loaded (14d, 14e, 14f), sorting the fractional part of both RK elements, we see that B $\leq$ C (0.57 $\leq$ 0.98), so the first part loaded is B and then C (14g, 14h, 14i). The two sub parts are loaded in machine m_2, according to the integer part of the RK. Finally, after C is scheduled, both D and E can be chosen, similarly to the last step, the sorting of the fractional parts dictates that E is loaded (14j, 14k, 14l) before D (14m, 14n, 14o).

Now, the same process is applied to RK_C on Fig. 15; the first allocated part is C, using only the integer part of the key (15a, 15b, 15c). To decide what is the next loaded part, we sort the fractional part regarding subparts D and E. We see that D $\leq$ E (0.05 $\leq$ 0.30), so the first part loaded is D (15d, 15e, 15f) and then E (15g, 15h, 15i)

The next appendix demonstrate how to perform the upper and lower bounds.

B Appendix: Due dates lower and upper bounds

In this Section we perform the lower an upper bounds of product A (the decoding process depicted in Fig. 14) In Fig. 16 we present the T_l and T_u for the product of Fig. 11.

Note that, when calculating the T_l in Fig. 16a, it is not considered only the permissible set of machines, instead, all products are allowed to be processed on all machines (and there may exist infinite machines). That yield a $T_l = 4$ (MRP calculations - 1), which is clearly infeasible, since the production should start on period 0. The T_u, on the other hand (Fig. 16b) is calculated as if only one machine were available, and stating from the last period with loaded parts, in this case, there is a loaded part on period 8, which yield a $T_u = 18$.

If the client due date for the product were to be on time period 2, it would be a waste of computations trying to decode the solution starting from that point; as it is now know by the $T_l = 4$, on the best case scenario, the first feasible decoding was due to period 5. So, only by knowing *a priori* the T_l would save at least 4 unnecessary random key decoding. Also, there is no need to try to decode the solution on a period further than T_u, as it is certainly feasible, and the best decoding should be before it.

With this information we infer that the first feasible decoding is not before the T_l and also not after T_u.

References

Bean, J. C. (1994). Genetic algorithms and random keys for sequencing and optimization. *ORSA Journal on Computing*, *6*(2), 154–160.

Bruntsch, A., & Tseng, M. M. (2016). An approach for process control of responsive service processes. *CIRP Annals*, *65*(1), 459–462.

Burden, R., & Faires, J. (2004). *Numerical analysis*. Boston: Cengage Learning.

Chansombat, S., Pongcharoen, P., & Hicks, C. (2019). A mixed-integer linear programming model for integrated production and preventive maintenance scheduling in the capital goods industry. *International Journal of Production Research*, *57*(1), 61–82.

Chen, K., & Ji, P. (2007a). Development of a genetic algorithm for scheduling products with a multi-level structure. *The International Journal of Advanced Manufacturing Technology, 33*(11–12), 1229–1236.

Chen, K., & Ji, P. (2007b). A mixed integer programming model for advanced planning and scheduling (aps). *European Journal of Operational Research, 181*(1), 515–522.

Chen, K., Ji, P., & Wang, Q. (2011). A case study for advanced planning and scheduling (aps). *Journal of Systems Science and Systems Engineering*, *20*(4), 460–474.

Cheng, R., Gen, M., & Tsujimura, Y. (1996). A tutorial survey of job-shop scheduling problems using genetic algorithms-i representation. *Computers & Industrial Engineering*, *30*(4), 983–997.

Choueiri, A. C., Sato, D. M. V., Scalabrin, E. E., & Santos, E. A. P. (2020). An extended model for remaining time prediction in manufacturing systems using process mining. *Journal of Manufacturing Systems*, *56*, 188–201.

Cullinan, C., Sutton, S., & Arnold, V. (2010). Technology monoculture: Erp systems, "techno-process diversity" and the threat to the information technology ecosystem. *Advances in Accounting Behavioral Research*, *13*, 13–30.

dos Santos, Garcia C., Meincheim, A., Junior, E. R. F., Dallagassa, M. R., Sato, D. M. V., Carvalho, D. R., et al. (2019). Process mining techniques and applications-a systematic mapping study. *Expert Systems with Applications*, *133*(1), 260–295.

Dayou, L., Pu, Y., & Ji, Y. (2009). Development of a multiobjective ga for advanced planning and scheduling problem. *The International Journal of Advanced Manufacturing Technology*, *42*(9–10), 974.

De Leoni, M., van der Aalst, W. M., & Dees, M. (2016). A general process mining framework for correlating, predicting and clustering dynamic behavior based on event logs. *Information Systems*, *56*, 235–257.

Efron, B. (1992). Bootstrap methods: Another look at the jackknife. In S. Kotz & N. L. Johnson (Eds.), *Breakthroughs in statistics* (pp. 569–593). New York: Springer.

Hosseini, S., & Al Khaled, A. (2014). A survey on the imperialist competitive algorithm metaheuristic: Implementation in engineering domain and directions for future research. *Applied Soft Computing*, *24*, 1078–1094.

Hosseini, S., Al Khaled, A., & Vadlamani, S. (2014). Hybrid imperialist competitive algorithm, variable neighborhood search, and simulated annealing for dynamic facility layout problem. *Neural Computing and Applications*, *25*(7–8), 1871–1885.

Kalenkova, A., Burattin, A., de Leoni, M., van der Aalst, W., & Sperduti, A. (2019). Discovering high-level bpmn process models from event data. *Business Process Management Journal*, *25*, 995–1019.

Karimi-Nasab, M., & Seyedhoseini, S. (2013). Multi-level lot sizing and job shop scheduling with compressible process times: A cutting plane approach. *European Journal of Operational Research*, *231*(3), 598–616.

Kim, J. U., & Kim, Y. D. (1996). Simulated annealing and genetic algorithms for scheduling products with multi-level product structure. *Computers & Operations Research*, *23*(9), 857–868.

Lee, J., Kao, H. A., & Yang, S. (2014). Service innovation and smart analytics for industry 4.0 and big data environment. *Procedia Cirp*, *16*, 3–8.

Moeuf, A., Pellerin, R., Lamouri, S., Tamayo-Giraldo, S., & Barbara, R. (2018). The industrial management of smes in the era of industry 4.0. *International Journal of Production Research*, *56*(3), 1118–1136.

Mohammadi, M., Ghomi, S. F., Karimi, B., & Torabi, S. A. (2010). Rolling-horizon and fix-and-relax heuristics for the multi-product multi-level capacitated lotsizing problem with sequence-dependent setups. *Journal of Intelligent Manufacturing*, *21*(4), 501–510.

Montgomery, D. C. (2017). *Design and analysis of experiments*. Hoboken: John wiley.

Murata, T. (1989). Petri nets: Properties, analysis and applications. *Proceedings of the IEEE*, *77*(4), 541–580.

Myers, D., Suriadi, S., Radke, K., & Foo, E. (2018). Anomaly detection for industrial control systems using process mining. *Computers & Security*, *78*, 103–125.

Na, H., & Park, J. (2014). Multi-level job scheduling in a flexible job shop environment. *International Journal of Production Research*, *52*(13), 3877–3887.

Orlicki, J. A. (1975). *Material requirements planning: The new way of life in production and inventory management*. New York: McGraw-Hill.

Öztürk, C., & Ornek, A. M. (2014). Operational extended model formulations for advanced planning and scheduling systems. *Applied Mathematical Modelling*, *38*(1), 181–195.

Peterson, J. L. (1981). *Petri net theory and the modeling of systems*. New Jersey: Prentice Hall PTR.

Petroni, A. (2002). Critical factors of mrp implementation in small and medium-sized firms. *International Journal of Operations & Production Management*, *22*, 329–348.

Pinedo, M. (2012). *Scheduling: Theory, algorithm and systems*. Berlin: Springer.

Pongcharoen, P., Hicks, C., Braiden, P., & Stewardson, D. (2002). Determining optimum genetic algorithm parameters for scheduling the manufacturing and assembly of complex products. *International Journal of Production Economics*, *78*(3), 311–322.

Puongyeam H, Pongcharoen P, & Vitayasak S (2014) Application of krill herd (kh) algorithm for production scheduling in capital goods industries. In: Proc. Int. Conf. Challenges IT, Eng. Technol.(ICCIET), pp 67–72

Ruschel, E., Santos, E. A. P., & Loures, Ed F R. (2018). Establishment of maintenance inspection intervals: An application of process mining techniques in manufacturing. *Journal of Intelligent Manufacturing*, *31*, 1–20

Saitou, K., Malpathak, S., & Qvam, H. (2002). Robust design of flexible manufacturing systems using, colored petri net and genetic algorithm. *Journal of Intelligent Manufacturing*, *13*(5), 339–351.

Schumacher, A., Erol, S., & Sihn, W. (2016). A maturity model for assessing industry 4.0 readiness and maturity of manufacturing enterprises. *Procedia Cirp*, *52*(1), 161–166.

Shapiro, S. S., & Wilk, M. B. (1965). An analysis of variance test for normality (complete samples). *Biometrika*, *52*(3/4), 591–611.

Sobaszek, Ł., Gola, A., & Kozłowski, E. (2019). Prediction of variable technological operation times in production jobs scheduling. *IFAC-PapersOnLine*, *52*(13), 1301–1306.

Stadtler, H. (2011). Multi-level single machine lot-sizing and scheduling with zero lead times. *European Journal of Operational Research*, *209*(3), 241–252.

Sun, H., Du, Y., Qi, L., & He, Z. (2019). A method for mining process models with indirect dependencies via petri nets. *IEEE Access*, *7*, 81211–81226.

Thiede, M., Fuerstenau, D., & Bezerra Barquet, A. P. (2018). How is process mining technology used by organizations? A systematic literature review of empirical studies. *Business Process Management Journal*, *24*(4), 900–922.

Trappey, A. J., Trappey, C. V., Govindarajan, U. H., Chuang, A. C., & Sun, J. J. (2017). A review of essential standards and patent landscapes for the internet of things: A key enabler for industry 4.0. *Advanced Engineering Informatics*, *33*, 208–229.

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

Van Der Aalst, W. (2016). Data science in action. *Process mining* (pp. 3–23). Berlin: Springer.

Wang, X., Ong, S., & Nee, A. (2018). A comprehensive survey of ubiquitous manufacturing research. *International Journal of Production Research*, *56*(1–2), 604–628.

Yan, J., Li, L., Zhao, F., Zhang, F., & Zhao, Q. (2016). A multi-level optimization approach for energy-efficient flexible flow shop scheduling. *Journal of Cleaner Production*, *137*, 1543–1552.

Yang, Y., Huang, M., Wang, Z. Y., & Zhu, Q. B. (2020). Robust scheduling based on extreme learning machine for bi-objective flexible job-shop problems with machine breakdowns. *Expert Systems with Applications*, *158*, 113545.

Zimmermann, A., Rodriguez, D., & Silva, M. (2001). A two phase optimization method for petri net models of manufacturing systems. *Journal of Intelligent Manufacturing*, *12*(5–6), 409–420.