Computers & Industrial Engineering 192 (2024) 110226

Contents lists available at ScienceDirect

Computers & Industrial Engineering

journal homepage: www.elsevier.com/locate/caie

Optimizing modular equipment in the design of multi-product reconfigurable production lines

Abdelkrim R. Yelles-Chaouche [a], Evgeny Gurevsky [b,*], Nadjib Brahimi [c], Alexandre Dolgui [d]

[a] Mines Saint-Étienne, LIMOS, Gardanne, France
[b] LS2N, Université de Nantes, France
[c] Rennes School of Business, France
[d] IMT Atlantique, LS2N, Nantes, France

ARTICLE INFO

Keywords:
Manufacturing
Line balancing
Multi-model
Reconfigurable
Modularity
MILP
Decomposition
Heuristic

ABSTRACT

This work addresses the problem of the design and balancing of a modular multi-product line in a reconfigurable environment. The study emphasizes the importance of modularity in the reconfigurable manufacturing systems (RMS), which allows fast, efficient, and cost-effective adaptation to changes by rearranging, moving, or removing modules. The line consists of a fixed number of modular machines arranged linearly and connected by a material handling system. Each machine has a limited number of slots for modules. The line can handle several product types per batch, and each batch requires a specific line configuration to be produced. The objective consists in identifying the minimum number of modules needed and their arrangement in different configurations to produce each product type. To solve this problem, a compact integer linear programming (ILP) formulation is initially investigated. Then an ILP reformulation, based on a decomposition approach, as well as a heuristic that uses this ILP reformulation are developed. The results demonstrate the effectiveness of the proposed methods.

1. Introduction

Modularity is considered as one of the key features of reconfigurable manufacturing systems (RMS) (see, *e.g.*, Koren et al., 1999). The term *modularity* is employed to describe the use of independent, standardized, and interchangeable units, called modules, in order to perform a variety of functions (see, *e.g.*, Huang & Kusiak, 1998). Thus, a modular design is a concept in which functionalities are divided into discrete, coherent and self-contained entities. It enables RMS to adapt to changes in a fast, efficient and cost-effective manner. This is made possible by the use of reconfigurable machines that can be easily modified by rearranging, replacing, or adding modules, which effectively subdivide the system into simplified modular components (see, *e.g.*, Baldwin & Clark, 2006). The modularity also provides flexibility in adapting to dynamic environments, as each module can be easily modified or replaced without affecting the rest of the system (see, *e.g.*, Shaik et al., 2015). Moreover, a modular manufacturing system also allows to better cope with unexpected events such as machine breakdowns, since the failure of one module does not necessarily impact the other ones. In addition, modular systems are more cost-effective to develop and modify than complex systems, as individual modules can be customized or upgraded without the need to modify the entire system.

Despite its many advantages, the design of a modular manufacturing system remains a very complex problem, since it usually involves determining the modules needed to manufacture a product, with each module performing specific tasks. Additionally, modules have to be allocated to machines that are compatible with their software and hardware. When designing a modular manufacturing system, a compromise has to be found between the number of functions/tasks the modules can perform, and their cost. Larger modules are able to perform more functions and may require fewer interfaces, reducing the number of reconfigurations required, but they can be expensive to design, develop, and maintain. In contrast, smaller modules may be less expensive, but involve more interfaces, increasing the complexity of the system and the number of necessary reconfigurations.

The objective of this study is to investigate the problem of designing and balancing a modular line capable of producing multiple products within the context of RMS. The considered line is made up of modular machines, where each module consists of physical equipment able to perform a limited number of manufacturing tasks. This modular design facilitates reconfiguration of the line to adapt it to different products or respond to changing demand (see, *e.g.*, Koren et al., 1999). This paper focuses only on the first case mentioned above, and does not address

* Corresponding author.
E-mail address: evgeny.gurevsky@univ-nantes.fr (E. Gurevsky).

https://doi.org/10.1016/j.cie.2024.110226

Available online 15 May 2024

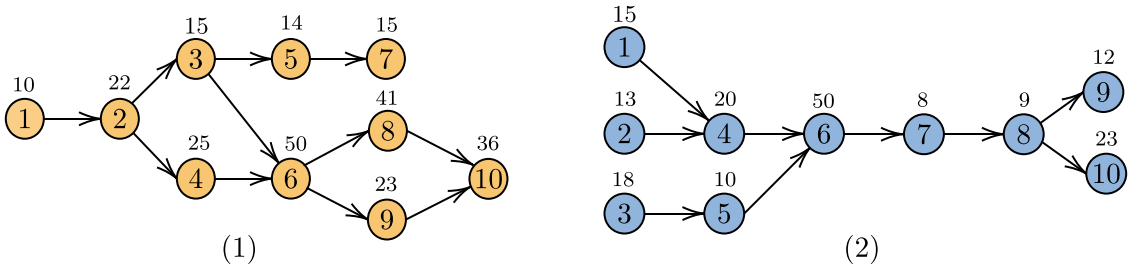

Fig. 1. (1) and (2) are the precedence graphs that correspond to products 1 and 2.

the second one, *i.e.,* responding to changes in demand, which will be the topic of future research.

More precisely, the studied line is composed of a fixed number of modular machines arranged in a linear manner and linked by a material handling system such as a conveyor belt. Each machine has a limited number of positions, called *slots*, into which modules can be placed. For the sake of simplicity, it is assumed that machine slots are compatible with all the modules. The products are processed sequentially by the modules as they move from one machine to another. Each module is capable of performing a limited number of tasks, also sequentially, following a pre-defined order that does not affect precedence constraints. The latter are usually expressed by a direct acyclic graph, whose nodes (resp. arcs) represent the tasks (resp. the precedence relations between tasks). The working time of a machine is determined by the sum of the processing time of all the tasks carried out by the modules of this machine. This time may not exceed a given threshold value, known also as the *cycle time*. Finally, the studied line can handle several product types in batches. Tasks precedence relations and the cycle time of each product type can be different. To accommodate each product type, a specific line configuration is required, which involves reconfiguring the machines by adding, moving, or removing modules in the available slots.

The goal is then to create a feasible line configuration for each product type, with the minimum number of different modules needed. This entails identifying the tasks that each module is capable of performing, and organizing these modules into different configurations to produce the various products. The modules have to be positioned within the available machines in a way that meets the production requirements. To solve this optimization problem, a compact integer linear programming (ILP) formulation is firstly proposed. Subsequently, a more efficient ILP reformulation is developed, which is based on a decomposition approach. Finally, a heuristic approach based on the ILP reformulation is proposed. It is worth noting that other objectives, such as minimizing the total cost of the modules, can also be taken into consideration. However, due to the lack of a method to calculate the cost of each created module, it was decided to focus solely on minimizing the number of modules. The proposed ILP formulation can however be easily adapted to minimize the cost-oriented objective function incorporating a potentially different cost for each created module, offering an interesting perspective for this work.

Fig. 1 provides an example of the above described problem. It shows two precedence graphs corresponding to two products. It can be noticed that the products share the same set of tasks to be executed. However, the precedence graphs and task processing times are different. The cycle time for each product is also different, equaling 95 time units for product 1 and 75 times units for product 2.

If we suppose that no module can perform more than two tasks and that a machine can hold at most two modules, then an optimal composition, represented in Fig. 2, requires six modules m_i, $i \in \{1, \ldots, 6\}$, to produce both products. Based on this composition, two configurations can be constructed as shown in Fig. 3, where configuration 1 (resp. 2) corresponds to the arrangement of these modules within the available machines so as to produce product 1 (resp. 2). Switching from one configuration to another can be done by rearranging some modules between the machines.

The rest of this paper is organized into five sections. A literature review of related problems is given in Section 2. A compact ILP formulation for the studied problem is provided in Section 3. Its more efficient ILP reformulation as well as a heuristic method based on it are presented in Section 4. The obtained numerical results for all the above-mentioned formulations and methods are reported and analyzed in Section 5. Finally, in Section 6, we make some concluding remarks and suggest some interesting directions for future research.

2. Literature review

This section presents a review of research works related to the topic of this paper. It focuses on three main axes: studies on assembly line balancing problems in general, research on multi-model assembly line balancing problems, and finally publications on equipment selection and modularity in assembly line balancing.

2.1. Assembly line balancing problem

The optimization problem addressed in this paper is closely connected to the Simple Assembly Line Balancing Problem (SALBP) (see, *e.g.*, Scholl, 1999). Given a set of tasks necessary to assemble a product, the aim of SALBP is to optimize a desired production goal by assigning all these tasks to the available workstations, while satisfying task precedence constraints, generally represented by a direct acyclic graph. Following Scholl (1999), two main optimization problems are usually studied in the literature of SALBP: minimize the total number workstations subject to a given cycle time (SALBP-1), and minimize the cycle time subject to a given number of workstations (SALBP-2). Several studies have tackled SALBP using exact and approximate approaches (see, *e.g.*, the literature review of Battaïa & Dolgui, 2013; Baybars, 1986; Erel & Sarin, 1998). Since SALBP considers only a single type of product, these approaches cannot be directly applied to solve the optimization problem studied in this paper as multiple product types are taken into account.

2.2. Multi-model assembly line balancing problem

When dealing with more than one product, two variations of SALBP are discussed in the literature (see, *e.g.*, Scholl, 1999): mixed- and multi-model assembly line balancing problems. The first concerns a single line where several products from the same family are handled simultaneously (see, *e.g.*, Battini et al., 2006; Bukchin & Rabinowitch, 2006; Erel & Gokcen, 1999). The second problem involves a single line where different product types from the same family are manufactured in separate batches (see, *e.g.*, van Zante-de Fokkert & de Kok, 1997). Such lines have to be reconfigured when switching from one product type configuration to another. Reconfiguration consists in rebalancing the line by reassigning some tasks between the existing workstations. Few research has been conducted on the multi-model assembly line balancing problem (MuMALBP). However, the emergence of flexible

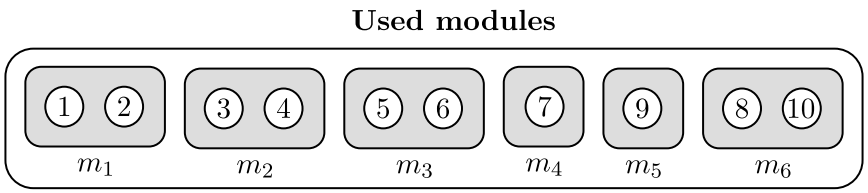

Fig. 2. The generated optimal modules.

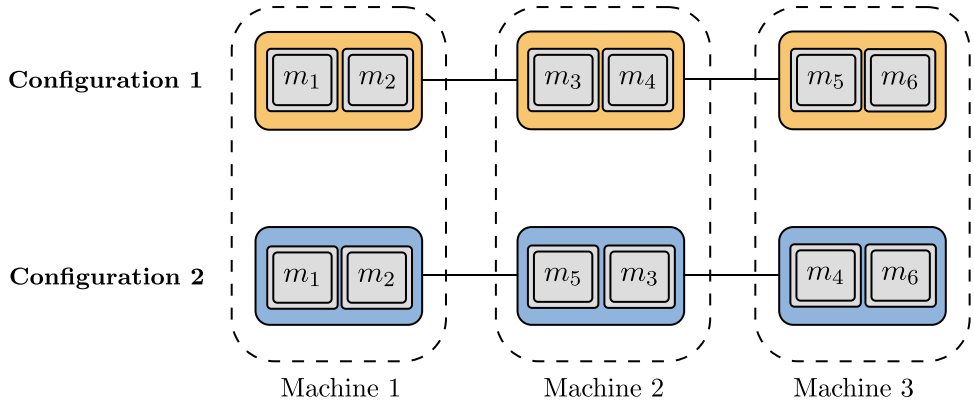

Fig. 3. Optimal product configurations that minimizes the total number of modules.

and reconfigurable manufacturing systems implies that this type of line becomes more advantageous as it can offer different configurations (see, *e.g.*, Battaïa & Dolgui, 2022). In this context, Tremblet et al. (2023) and Yelles-Chaouche et al. (2022) have considered the balancing problem of a multi-product reconfigurable line, where the aim is to minimize the total number of reassigned tasks when moving from one product configuration to another. Specifically, Yelles-Chaouche et al. (2022) studied the case where the sequence of product batches is given, while Tremblet et al. (2023) considered the scenario where no information about this sequence is provided. They both developed a mixed-integer linear programming (MILP) formulation along with a MILP-based heuristic approach to tackle this problem.

Lapierre et al. (2000) and Tóth et al. (2018) consider a multi-model printed circuit board (PCB) assembly line. In Lapierre et al. (2000), a MILP model is firstly formulated to generate an admissible line configuration for each PCB-type batch. The objective function seeks to minimize the cycle time necessary to assemble a PCB-type. A Lagrangian relaxation is also proposed to deal with large size instances of the studied problem. Latter, Tóth et al. (2018) addressed the same problem by considering PCB batches of different size and type in the presence of modular machines. The aim is to provide a workload balancing and select an appropriate machine module configuration for each batch type. The objective function minimizes the total production time, which includes the setup time for product type changing. The authors proposed an iterative approach based on a genetic algorithm for generating line configuration and an ILP formulation for line balancing. Pereira (2018) tackles a MuMALB problem which arises in textile industry. Given several types of goods, the aim is to find an appropriate line balancing for each of them. The authors proposed a hybrid approach, which combines the Hoffmann heuristic with a distribution estimation algorithm. The objective is to minimize resource and workstation costs. Berger et al. (1992) consider an MuMALB problem in a flexible manufacturing environment. The authors propose a branch-and-bound algorithm to minimize the number of workstations. To do this, the authors combined the precedence graphs for each product into a single one, transforming the problem into an SALBP-1. In the same way, Nazarian et al. (2010) deal with a multi-product hybrid line composed of flexible and non-flexible machines. Flexible ones are able to be easily reconfigured, whereas the others are dedicated machines that can perform a limited number of tasks. The authors provide a decision-support tool to determine the number of required flexible and non-flexible machines, based on the variation in task processing times over time. A MILP model and a heuristic, based on a single combined precedence graph, were developed to minimize the investment cost of the generated machines.

In this paper, an MuMALBP is considered in the presence of modular machines. Unlike the aforementioned articles, the challenge here is to identify the minimum number of different modules needed for producing several products. Each product requires a specific line configuration, defined by an arrangement of modules in the machines. The set of modules is not known in advance and needs to be generated based on the precedence graph of each product. To switch production from one product to another, the line has to be reconfigured by moving, adding or removing modules. To the best of our knowledge, this particular problem has not been addressed in the existing literature on line balancing problems.

2.3. Equipment selection and modularity in line balancing

In this paper, we consider a line balancing and equipment selection problem. It consists in assigning tasks to a set of available machines/workstations and the corresponding equipment subject to given production constraints. Some previous studies have already tackled this problem. For example, Graves and Redfield (1988) address a problem where a family of similar products has to be assembled on a line. The authors propose a methodology to find a single line configuration that is able to produce all the products and aiming to minimize the total cost, which includes capital cost related to the equipment and variable operating cost. Bukchin and Tzur (2000) propose a branch-and-bound approach to tackle the problem of selecting equipment and assigning tasks to workstations. The objective function minimizes the total equipment costs. Later in Bukchin and Rubinovitz (2003), the authors addressed the same problem by considering the minimization of the number of workstations along with equipment costs.

Finally, the closest problem to the one studied in this paper is from Belmokhtar et al. (2006). In this work, the authors consider a modular machining line comprising multi-spindle units. The aim is to design and balance a single product line by selecting and assigning

spindle units to machines subject to precedence, cycle time, and some other technological constraints. An integer linear programming (ILP) formulation was developed to minimize the investment cost associated with the selected units. Borisovsky et al. (2014) tackle the problem of designing an optimal machining line, taking into account task sequence set-up time and several technological constraints. The authors propose a reduction of this problem to a set partitioning one. This involves generating all possible workstations composing the machines and finding for each of them the optimal sequence of tasks that minimizes total set-up time, using a dynamic programming approach. Subsequently, three exact approaches were proposed to solve the set partitioning problem: constraint generation, branch-and-cut, and parallel branch-and-cut algorithms. In the same scope, Battaïa et al. (2012) tackle an equipment location problem in a machining line. The authors propose several preprocessing procedures, based on constraint analysis, in order to reduce the search space. Moreover, they developed an algorithm for generating a lower bound on the number of used equipment.

The above-mentioned research papers assume the presence of a predetermined set of equipment, with the goal being the selection of the most suitable ones to fulfill production objectives and minimize the overall cost. However, in this paper, the scenario differs as the set of equipment, or modules, needs to be constructed along with the optimal line configuration for each product.

2.4. Multi-product configuration design in RMS

Several works in the RMS literature have studied the multi-product line configuration design problem (see, *e.g.*, Yelles-Chaouche et al., 2021). In this type of problem, a set of available reconfigurable machines is considered to be known. Moreover, each machine can have different configurations, each allowing it to perform different tasks. Hence, the problem consists in determining the number of stages and the number of machines per stage in order to satisfy the product demand. Each stage can have multiple parallel machines with different configurations. For example, Youssef and ElMaraghy (2006, 2007, 2008) consider the multi-product line configuration design problem with the main objective of minimizing the capital cost. Similar objectives are considered by Dou et al. (2009, 2016). Ashraf and Hasan (2018) tackle the same problem while considering reconfigurability, operational capability and system reliability in the objective function, in addition to total cost minimization. In Saxena and Jain (2012), additional costs are introduced in the objective function such as reconfiguration, operating and maintenance costs. Bryan et al. (2013) propose an approach to design a reconfigurable assembly line while considering product evolution over time. The main objective consists in finding the optimal reconfiguration plan that minimizes the life cycle cost, calculated based on the production and reconfiguration periods of evolving products.

In the context of the multi-product configuration design problem, existing approaches typically assume that machine configurations and their operational capabilities are known and given. This is not the case in the present paper, where machine configurations have to be designed for each product by assigning appropriate modules. Moreover, while prior studies often concern mixed lines, where multiple product types are simultaneously manufactured, this paper addresses a multi-model line where each product type needs a specific line configuration. Thus, switching production from one product to another requires line reconfiguration. Finally, unlike the present study, the existing literature on configuration design in the context of RMS mainly considers unpaced flow lines, where machines are not restricted by a specific cycle time, and products are not forced to pass through all the machines. Such lines are particularly efficient for scenarios involving high-mix products and low production volumes.

In view of the various studies cited above, one can notice that

- The design and balancing problems of a multi-model line are only addressed in real industrial cases, due to the growing interest of such a line structure nowadays. However, there is a lack of generalized studies of this type of problem.
- The line balancing problem along with the equipment selection/generation was addressed in the presence of a single product only.

Summarizing this section, the main contributions of this paper are:

- **Addressing the MuMALBP with modular machines:** This paper tackles the challenge of the multi-model assembly line balancing problem (MuMALBP) in the context of modular machines, aiming to identify the minimum number of modules required for producing various products.
- **Considering line reconfiguration:** Unlike previous studies that focus on single-product lines or assume fixed equipment sets, this paper accounts for the reconfiguration of assembly lines when transitioning between products. It necessitates constructing optimal line configurations and selecting equipment for each product.
- **Providing generalized study:** While prior research often concentrates on specific industrial cases, this paper offers a comprehensive investigation of the MuMALBP, catering to the increasing interest in such line structures across manufacturing systems.
- **Integrating modular machines:** The paper introduces the integration of modular machines into the line balancing problem, where the modules are dynamically generated based on the precedence graph of each product, rather than being predefined.

3. Compact ILP formulation

Given a family of products and a set of available machines, each with a limited number of module slots, the following compact ILP formulation aims to generate the minimum number of modules required to produce all the products, while taking into account task precedence and cycle time constraints. The maximum number of tasks that can be performed by any module is also provided. Inclusion and exclusion constraints are not considered in this paper for tasks and modules, but can easily be incorporated if needed. Consequently, any task can potentially be assigned to any module and any module can be placed in any available slot within a machine. The used notations and variables are described below.

Notations:

- V is the set of all tasks;
- W is the set of available machines;
- P is the set of products;
- M is the set of all the modules that could be generated;
- $m_{\max}$ is the maximum number of modules per machine;
- $r_{\max}$ is the maximum number of tasks per module;
- $E = \{1, \ldots, |W| \cdot m_{\max}\}$ is the global set of module slots;
- $E_k = \{(k-1) \cdot m_{\max} + 1, \ldots, k \cdot m_{\max}\}$ is the set of module slots of the machine $k \in W$;
- $C^{(p)}$ is the cycle time corresponding to the product $p \in P$;
- $t_i^{(p)}$ is the processing time of the task $i \in V$ corresponding to the product $p \in P$;
- $G^{(p)} = (V, A^{(p)})$ is a directed acyclic graph representing the precedence constraints of the product $p \in P$. Here, $A^{(p)}$ is the set of arcs for $G^{(p)}$, where an arc $(i, j) \in A^{(p)}$ means that task j has to be assigned either to the same module as task i, or to a succeeding one.

Decision variables:

- x_{im} is equal to 1 if the task $i \in V$ is assigned to the module $m \in M$, 0 otherwise;

- $y_{me}^{(p)}$ is equal to 1 if the module $m \in M$ is allocated to the slot $e \in E$ in the configuration corresponding to the product $p \in P$, 0 otherwise;
- $z_{ime}^{(p)}$ is equal to 1 if the task $i \in V$ is assigned to the module $m \in M$, which in its turn is allocated in the slot $e \in E$ in the configuration corresponding to the product $p \in P$, 0 otherwise;
- s_m is equal to 1 if the module $m \in M$ is not empty, 0 otherwise.

$$\min \sum_{m \in M} s_m \tag{1}$$

$$1 \leq \sum_{m \in M} x_{im} \leq |P|, \quad \forall i \in V \tag{2}$$

$$\sum_{e \in E} y_{me}^{(p)} \leq 1, \quad \forall m \in M, \quad \forall p \in P \tag{3}$$

$$\sum_{m \in M} y_{me}^{(p)} \leq 1, \quad \forall e \in E, \quad \forall p \in P \tag{4}$$

$$\sum_{m \in M} \sum_{e \in E} z_{ime}^{(p)} = 1, \quad \forall i \in V, \quad \forall p \in P \tag{5}$$

$$x_{im} \leq s_m, \quad \forall i \in V, \quad \forall m \in M \tag{6}$$

$$x_{im} + y_{me}^{(p)} \leq z_{ime}^{(p)} + 1, \quad \forall i \in V, \quad \forall m \in M, \quad \forall e \in E, \quad \forall p \in P \tag{7}$$

$$z_{ime}^{(p)} \leq x_{im}, \quad \forall i \in V, \quad \forall m \in M, \quad \forall e \in E, \quad \forall p \in P \tag{8}$$

$$z_{ime}^{(p)} \leq y_{me}^{(p)}, \quad \forall i \in V, \quad \forall m \in M, \quad \forall e \in E, \quad \forall p \in P \tag{9}$$

$$\sum_{m \in M} \sum_{e \in E} e \cdot z_{ime}^{(p)} \leq \sum_{m \in M} \sum_{e \in E} e \cdot z_{jme}^{(p)}, \quad \forall (i,j) \in A^{(p)}, \quad \forall p \in P \tag{10}$$

$$\sum_{e \in E_k} \sum_{m \in M} \sum_{i \in V} t_i^{(p)} \cdot z_{ime}^{(p)} \leq C^{(p)}, \quad \forall k \in W, \quad \forall p \in P \tag{11}$$

$$\sum_{i \in V} t_i^{(p)} \cdot x_{im} \leq C^{(p)}, \quad \forall m \in M, \quad \forall p \in P \tag{12}$$

$$\sum_{i \in V} x_{im} \leq r_{\max}, \quad \forall m \in M \tag{13}$$

$$\left\lceil \frac{|V|}{r_{\max}} \right\rceil \leq \sum_{m \in M} s_m \leq |V| \tag{14}$$

$$s_{m+1} \leq s_m, \quad \forall m \in M \setminus \{|M|\} \tag{15}$$

$$\sum_{m \in M} y_{m,e+1}^{(p)} \leq \sum_{m \in M} y_{me}^{(p)}, \quad \forall e \in E_k \setminus \{k \cdot m_{\max}\}, \quad \forall k \in W, \quad \forall p \in P \tag{16}$$

$$z_{ime}^{(p)} = 0, \quad \forall i \in V, \quad \forall m \in M, \quad \forall e \notin \bigcup_{k \in Q_i^{(p)}} E_k, \quad \forall p \in P \tag{17}$$

$$x_{im} \in \{0,1\}, \quad \forall i \in V, \quad \forall m \in M$$

$$y_{me}^{(p)} \in \{0,1\}, \quad \forall m \in M, \quad \forall e \in E, \quad \forall p \in P$$

$$z_{ime}^{(p)} \in \{0,1\}, \quad \forall i \in V, \quad \forall m \in M, \quad \forall e \in E, \quad \forall p \in P$$

$$s_m \in \{0,1\}, \quad \forall m \in M$$

Objective function (1) minimizes the total number of non-empty modules. Expressions (2) ensure that each task has to be assigned to at least one and at most $|P|$ modules. Constraints (3) and (4) state that a module must be allocated to one slot only and that a slot cannot accommodate more than one module, regardless of the configuration. Constraints (5) express that any task has to be assigned exactly once for each product configuration. Inequalities (6) specify that a module is considered as non-empty if at least one task is assigned to it. Constraints (7), (8) and (9) force that a generated non-empty module be assigned to a slot in at least one configuration. The task precedence relations for each configuration are expressed by inequalities (10). Constraints (11) ensure that the effective workload of any machine does not exceed the cycle time, whatever the product configuration. Constraints (12) avoid generating modules whose workload exceeds the cycle time of the configuration to which they can be assigned. Inequalities (13) impose the maximum number of tasks per module. Expression (14) are optimality-based constraints that bound the optimal value of the objective function between its best and worst case scenarios. Namely, the best scenario corresponds to the case where all the generated modules are different (without common tasks) and all of them (except probably one) have the maximum admissible number of tasks. As for the worst scenario, it occurs when each module is also different, but has a single task assigned to it. For both cases, the mentioned modules are obviously common for all the configurations. Constraints (15) are symmetry-breaking ones allowing a module to be generated only if the previous one is non-empty. As for constraints (16), they are similar to those of (15), but concern the slots inside a machine. Finally, constraints (17) require each task to be assigned to a restricted subset of slots, defined by the so-called assignment interval $Q_i^{(p)} = \left[l_i^{(p)}, u_i^{(p)}\right]$, where $l_i^{(p)}$ (resp. $u_i^{(p)}$) identifies the earliest (resp. latest) machine to which task i can be assigned with respect to product p. Due to Patterson and Albracht (1975), this interval can be computed as follows:

$$Q_i^{(p)} = \left[\left\lceil \frac{t_i^{(p)} + \sum_{j \in \mathcal{P}_i^{(p)}} t_j^{(p)}}{C^{(p)}} \right\rceil, |W| + 1 - \left\lceil \frac{t_i^{(p)} + \sum_{j \in S_i^{(p)}} t_j^{(p)}}{C^{(p)}} \right\rceil \right],$$

where $\mathcal{P}_i^{(p)}$ (resp. $S_i^{(p)}$) is the set of all predecessors (resp. all successors) of task i with respect to the precedence graph $G^{(p)}$ of product p. Here, the idea of computing the earliest (resp. latest) machine index for task i is based on determining a valid lower bound on the number of machines needed to assign the set $\mathcal{P}_i^{(p)} \bigcup \{i\}$ (resp. $S_i^{(p)} \bigcup \{i\}$).

Fig. 4 illustrates the logic behind the compact ILP formulation (1)–(17). At the bottom of the figure is the set of tasks that have to be assigned to a given set of modules. These modules are then assigned to available machine slots, respecting the precedence graph and the cycle time constraints for each product configuration.

4. ILP reformulation

This section introduces a decomposition approach of the studied optimization problem. To do this, an ILP reformulation is first presented. Then a new algorithm that generates all possible feasible modules from the precedence graph is described. Additional notations and new decision variables are outlined below.

Additional notations:

- M is the set of all generated modules;
- $M_k^{(p)}$ is the set of modules that can be assigned to the machine $k \in W$ in the configuration corresponding to the product $p \in P$;
- $U_k^{(p)}$ is the set of module pairs that can never be assigned together to the same machine $k \in W$ in the configuration corresponding to the product $p \in P$;
- α_{im} is equal to 1 if the task $i \in V$ is present in the module $m \in M$;
- $c_m^{(p)}$ is the load corresponding to the module $m \in M$ for the product $p \in P$.

New decision variables:

- y_m is equal to 1 if the module $m \in M$ is used in at least one configuration, 0 otherwise.
- $\lambda_{mk}^{(p)}$ is equal to 1 if the module $m \in M$ is assigned to the machine $k \in W$ in the configuration corresponding to the product $p \in P$, 0 otherwise.

$$\min \sum_{m \in M} y_m \tag{18}$$

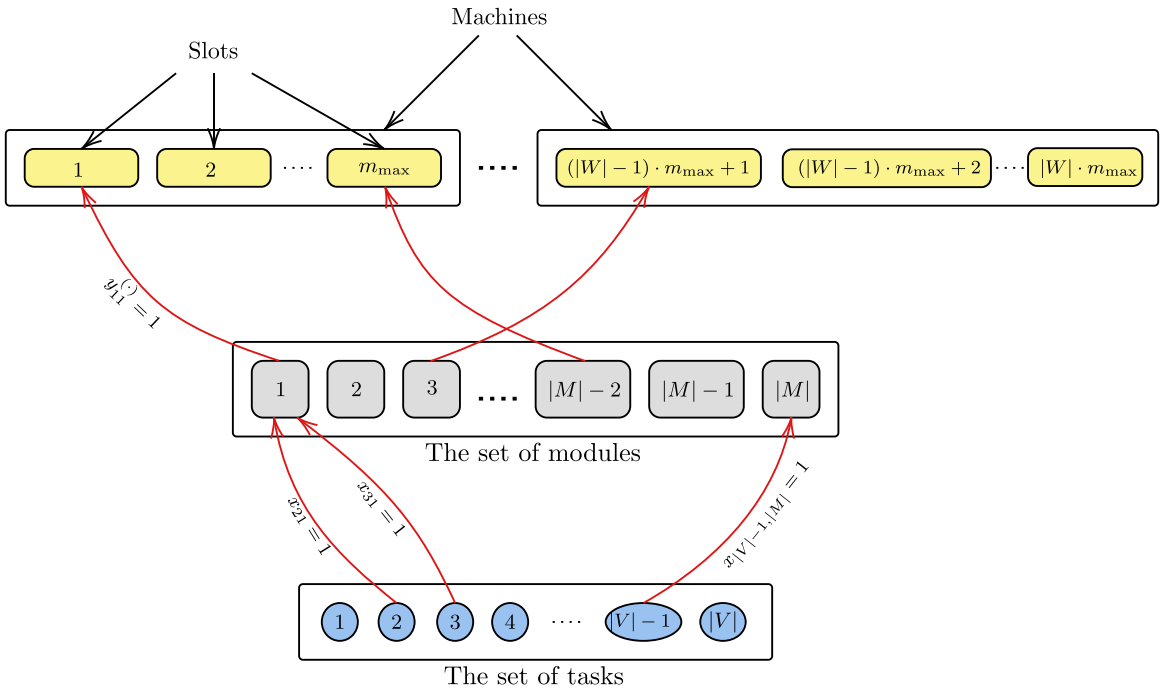

Fig. 4. An illustrative example of how the ILP model (1)–(17) works.

$$\sum_{k\in W}\sum_{m\in M}\alpha_{im}\cdot\lambda_{mk}^{(p)}=1,\quad \forall i\in V,\quad \forall p\in P \tag{19}$$

$$\sum_{m\in M}c_m^{(p)}\cdot\lambda_{mk}^{(p)}\le C^{(p)},\quad \forall k\in W,\quad \forall p\in P \tag{20}$$

$$\lambda_{mk}^{(p)}\le y_m,\quad \forall m\in M,\quad \forall k\in W,\quad \forall p\in P \tag{21}$$

$$\sum_{q=k}^{|W|}\sum_{m\in M}\alpha_{im}\cdot\lambda_{mq}^{(p)}\le\sum_{q=k}^{|W|}\sum_{m\in M}\alpha_{jm}\cdot\lambda_{mq}^{(p)},\quad \forall(i,j)\in A^{(p)},\ \forall k\in W,\ \forall p\in P \tag{22}$$

$$\sum_{m\in M\setminus M_k^{(p)}}\lambda_{mk}^{(p)}=0,\ \forall k\in W,\ \forall p\in P \tag{23}$$

$$\sum_{m\in M}\lambda_{mk}^{(p)}\le m_{\max},\quad \forall k\in W,\quad \forall p\in P \tag{24}$$

$$\lambda_{m_1,k}^{(p)}+\lambda_{m_2,k}^{(p)}\le 1,\quad \forall(m_1,m_2)\in U_k^{(p)},\quad \forall k\in W,\ \forall p\in P \tag{25}$$

$$y_m\in\{0,1\},\quad \forall m\in M$$

$$\lambda_{mk}^{(p)}\in\{0,1\},\quad \forall m\in M,\quad \forall k\in W,\quad \forall p\in P$$

Objective function (18) minimizes the total number of used modules. Equalities (19) ensure that each task has to be assigned in each configuration. Constraints (20) state that the sum of the processing time of the tasks performed by the modules assigned to a machine should not exceed the cycle time of each corresponding product. Inequalities (21) express that a module is considered as 'used' if it is assigned to at least one configuration. Precedence relations between tasks for all product configurations are modeled by (22). Expressions (23) are based on $M_k^{(p)}$ which identifies whether a module can be assigned to a machine or not. This allows to set some decision variables to zero. The set $M_k^{(p)}$ is calculated based on the task assignment interval technique, presented in Section 3. Namely, $M_k^{(p)}=\{m\in M : k\in\bigcap_{i\in m}Q_i^{(p)}\}$. Constraints (24) ensure that each machine cannot contain more than $m_{\max}$ modules. Finally, inequalities (25) avoid the assignment of conflicting modules to the same machine, because of task precedence constraints.

It is worth highlighting that this new reformulation gets rid of the slot index. Instead, constraints (25) indirectly dictate the execution sequence of modules within each machine, as they prevent conflicting modules to be allocated to it. An alternative reformulation, referred to as the slot-based reformulation, is detailed in Appendix. This alternative performs less well than (18)–(25), primarily due to the increased number of decision variables and constraints arising from large values of $m_{\max}$.

4.1. Module generation algorithm

The process of obtaining M consists in identifying the set of all feasible modules for each product p, denoted $M^{(p)}$. The set M is then formed by taking the union of all $M^{(p)}$, where $p\in P$, *i.e.*, $M=\bigcup_{p\in P}M^{(p)}$. Thus, a module is considered feasible if it contains a subset of tasks that is not affected by the precedence constraints and has a load that does not exceed the cycle time. Fig. 5 shows an example of some feasible and infeasible modules with respect to the precedence graph shown in the left hand side. In this example, the cycle time is set to 100 and the maximum number of tasks per module is 3. Hence, it is easy to notice that modules m_1, m_2, m_3 and m_4 are feasible modules, since they respect both of the aforementioned constraints. However, modules m_1', m_2' and m_3' are not feasible with respect to the precedence constraints, the cycle time, or both.

The construction of $M^{(p)}$ consists in identifying, from a given precedence graph, all the sub-graphs with a number of nodes not exceeding $r_{\max}$ and which satisfy some additional constraints detailed below. To achieve this, a new algorithm, named Pattern Generation, is developed in this paper. Initially, the algorithm starts with a set of unitary modules, *i.e.,* composed of a single task. Then, for each of them, all new possible modules are generated by adding new tasks (one at a time) such that the precedence, capacity and cycle time constraints are satisfied. To ensure that all the feasible modules are generated, the stack data structure is used in order to store the current state of modules under construction. Algorithm 4.1 provides a more detailed description of this process.

Lemma 4.1. *Let m be a feasible module. Then, any module $m\cup\{i\}$ satisfying the cycle time constraint is considered as a feasible one in terms of precedence constraints iff the following condition is fulfilled:*

$$\left(\left(\bigcup_{q\in m}S_q\right)\bigcap\mathcal{P}_i\right)\setminus(m\cup\{i\})=\emptyset. \tag{26}$$

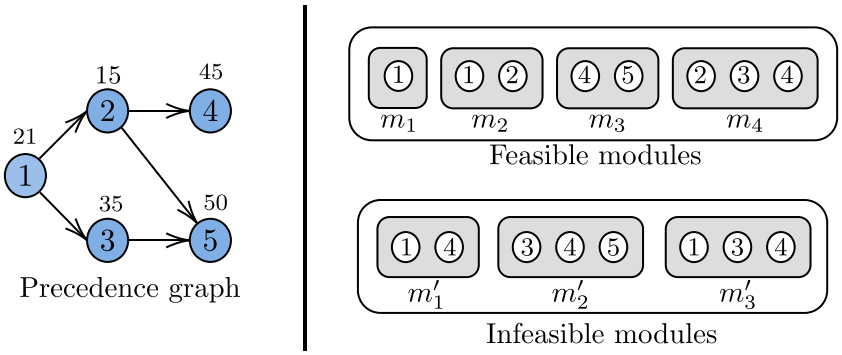

Fig. 5. An example of feasible and infeasible modules with $C = 100$ and $r_{\max} = 3$.

Algorithm 1 Pattern generation

Input: A directed acyclic graph G, a cycle time C and $r_{\max}$.

Output: A set $\mathcal{L}$ of generated modules.

Step 1: **Initialization.**

1.1 Consider an empty stack $\mathcal{X}$ of modules and set $\mathcal{L} := \emptyset$ and $k := 1$.

Step 2: **Module generation.**

2.1 Generate a new module composed only of task k and push it to the stack $\mathcal{X}$.
2.2 If $\mathcal{X}$ is not empty, then pop a new module m from $\mathcal{X}$. Otherwise, go to Step 3.
2.3 Add m to $\mathcal{L}$. If $|m| < r_{\max}$, go to Step 2.4. Otherwise, if $|m| = r_{\max}$, go to Step 2.2.
2.4 Let j be the latest task added to m. For each $i > j$, generate a new module $m \cup \{i\}$ and push it to $\mathcal{X}$, if its load does not exceed the cycle time, *i.e.*, $\sum_{q \in m} t_q \leq C - t_i$, and the intersection between all successors of m and all predecessors of i without taking into account $m \cup \{i\}$ is empty, *i.e.*, $\left(\left(\bigcup_{q \in m} \mathcal{S}_q\right) \bigcap \mathcal{P}_i\right) \setminus \left(m \cup \{i\}\right) = \emptyset$. For the latter condition, see Lemma 4.1.
2.5 Go to Step 2.2.

Step 3: **Analysis.**

3.1 If $k < |V|$, then set $k := k + 1$ and go to Step 2. Otherwise, stop. The whole set $\mathcal{L}$ of modules is generated.

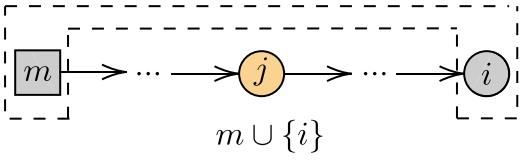

Fig. 6. A simple illustration of an infeasible module.

Proof ($\Rightarrow$). Suppose the contrary. Namely, let $m \cup \{i\}$ be a feasible module that does not satisfy (26). This means that there exists a task $j \notin m \cup \{i\}$ which is at the same time one of the predecessors of task i and one of the successors of the tasks composing m. The latter indicates that j cannot be executed neither before nor after the completion of the tasks belonging to the module $m \cup \{i\}$. Hence, j has to be in $m \cup \{i\}$, which is in contradiction with the supposition.

($\Leftarrow$) Suppose the contrary. Namely, let a module $m \cup \{i\}$ satisfies Eq. (26), but be infeasible. This will necessarily lead us to the situation depicted in Fig. 6 and, as a consequence, of non-satisfying (26). This contradicts the supposition. □

4.2. Constraint generation framework

Processing the whole ILP reformulation (18)–(25) may be a difficult challenge because of the presence of constraints (25). Indeed, enumerating and using all the inequalities (25) can lead to a significant increase in the total number of constraints, and render the reformulation (18)–(25) inefficient since any feasible solution of (18)–(24) usually violates only a few of inequalities (25).

Thus, to avoid the inefficiency in enumerating all constraints (25), a more suitable algorithmic approach is proposed. The approach is based on the constraint generation framework, generally provided by any modern commercial solver, and consists in generating only the violated constraints (25) during the search process in the nodes of the search tree, while still solving only the ILP reformulation (18)–(24). Commercial solvers can handle this efficiently, making them an appropriate solution for such a problem.

More precisely, the search process involves two scenarios for the current node in the search tree, depending on whether it is an internal or external (leaf) node. The former corresponds to a partially constructed solution, while the latter refers to a complete admissible integer solution. Algorithm 4.2 provides a detailed description of the analysis for both cases. The main objective is to identify and add violated inequalities (25) to the ILP reformulation (18)–(24) using the `LazyConstraints` and `UserCuts` routines, depending on the node's scenario.

4.3. Heuristic based on the ILP reformulation

The idea of the proposed heuristic approach is based on the reduction of the search space for the ILP reformulation (18)–(25) by

Algorithm 2 Two cases of node analysis

Case 1: **The studied node is an internal node.**

- Analyze the continuous relaxation of the corresponding partially constructed solution. If, for this relaxation, there exist a product p, a machine k and two modules m_1 and m_2 such that $(m_1, m_2) \in U_k^{(p)}$ and $\lambda_{m_1,k}^{(p)} + \lambda_{m_2,k}^{(p)} > 1$ then add into the ILP reformulation (18)-(24) the constraint $\lambda_{m_1,k}^{(p)} + \lambda_{m_2,k}^{(p)} \leq 1$ using the both `LazyConstraints` and `UserCuts` routines.

Step 2: **The studied node is an external (leaf) node.**

- Analyze the corresponding complete integer solution. If, for this solution, there exist a product p, a machine k and two modules m_1 and m_2 such that $(m_1, m_2) \in U_k^{(p)}$, $\lambda_{m_1,k}^{(p)} = 1$ and $\lambda_{m_2,k}^{(p)} = 1$ then add into the ILP reformulation (18)-(24) the constraint $\lambda_{m_1,k}^{(p)} + \lambda_{m_2,k}^{(p)} \leq 1$ using only the `LazyConstraints` routine.

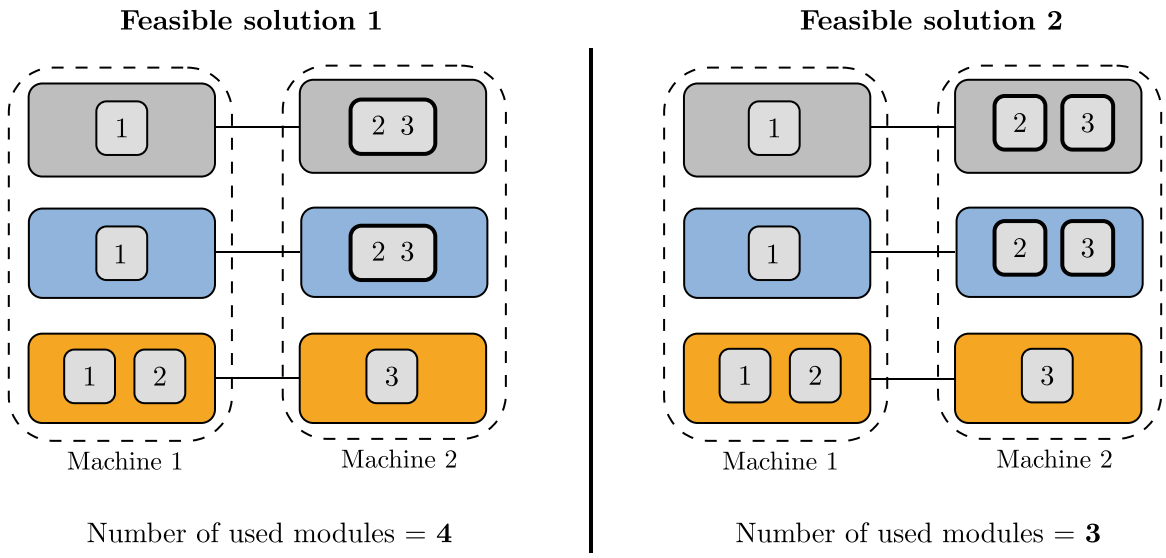

Fig. 7. Two feasible solutions (left and right) having a different number of modules used in total.

considering only the modules that are common for all product configurations. More formally, instead of M and $M_k^{(p)}$, the heuristic handles (18)–(25) with $\widehat{M}$ and $\widehat{M}_k^{(p)}$, respectively, such that $\widehat{M} = \{m \in M : m \in M^{(p)} \text{ for each } p \in P\}$ and $\widehat{M}_k^{(p)} = M_k^{(p)} \cap \widehat{M}$. Let us consider Fig. 7, which depicts two feasible solutions (1 and 2) of a problem with $r_{\max} = 2$ and $|P| = 3$. The first solution on the left uses four modules, whereas only three are required (as shown in the second solution on the right). In the first solution, the bold module (with tasks 2 and 3) is common to configurations 1 and 2, but not to configuration 3. Thus, additional modules are needed for the last configuration. To avoid such situations, the heuristic excludes modules that are not shared between all product configurations, leading more quickly to a solution such as Feasible solution 2. It is not difficult to see that in this case $\widehat{M}$ can be taken as $\bigcap_{p \in P} M^{(p)}$.

In what follows, the module exclusion process mentioned above is referred to as *module filtering*. Module filtering can be an effective approach to quickly reaching an optimal solution for most instances, as demonstrated in the computational results section. However, there may be tricky instances where an optimal solution can be composed of different (uncommon) modules. For example, consider Fig. 8, which shows two precedence graphs and let $r_{\max} = 3$. In this case, module filtering results in a solution that uses five common modules, while an optimal solution requires only four modules: m_1' and m_2' for the first product configuration, and m_3' and m_4' for the second product configuration. This highlights that the module filtering approach can only be considered as a heuristic one, even though this scenario did not occur in the tested instances.

5. Computational results

This section reports the experimental results of three developed approaches. Namely, the compact formulation (1)–(17), the ILP reformulation (18)–(25), and the ILP reformulation (18)–(25) with module filtering, hereafter referred to as the heuristic approach. As a reminder, the last two ILP reformulations are both handled within the constrain generation framework. The experiments were conducted using ILOG CPLEX 12.10.0 installed on a 1.90 GHz Intel(R) Core(TM) i7-8650U computer with 32 GB RAM. The well-known reference data instances from Otto et al. (2013) were used. The compact formulation was tested only on a set of small-size instances with $|V| = 20$, while two sets of small- and medium-size instances, with $|V| = 20$ and $|V| = 50$, were respectively used for the ILP reformulation and the heuristic approach. This decision was based on the observation that the compact formulation exhibited poor performance for small-size instances, so testing it on larger ones was deemed unnecessary. In addition, the well-known COMSOAL heuristic, developed by Arcus (1965), was employed to provide CPLEX with a warm-start feasible solution for each instance. To do this, the classic COMSOAL method was applied to each product $p \in P$ separately in order to build its feasible configuration, simply considering any task as a module of size one. This results in a feasible solution whose number of modules equals the number of tasks. The reason for adding this algorithm is that CPLEX was unable to find a feasible solution for some complex instances within the 600-second resolution time limit.

All the instances were grouped into three distinct series based on the density of their precedence graph, which is measured through its order strength (OS), defined as $\frac{2 \cdot |E^{(p)}|}{|V| \cdot (|V|-1)}$, where $E^{(p)} = \{(i,j) : i \in V \text{ and}$

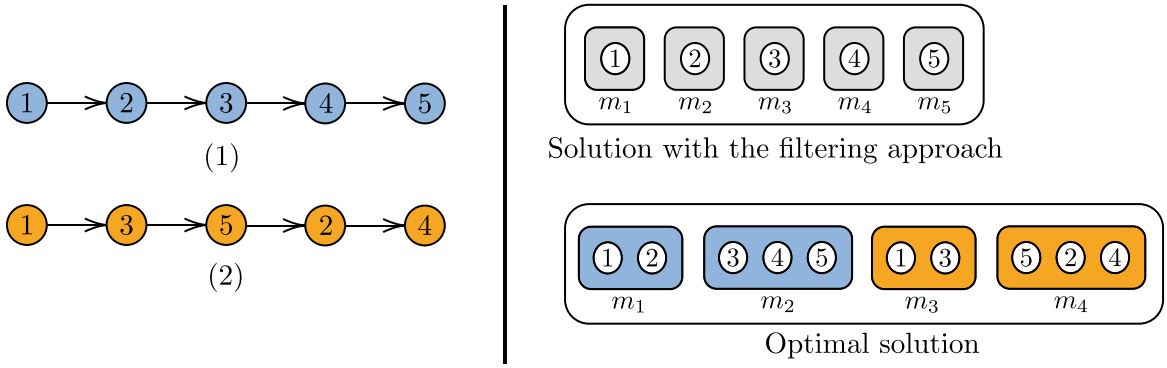

Fig. 8. Example of an optimal solution that is not composed of common modules.

Table 1
Enumeration of three series.

Series	1	2	3
OS	≈0.2	≈0.6	≈0.9

$j \in S_i^{(p)}\}$ is the set of arcs in the transitive closure of $G^{(p)}$. A higher OS value indicates a denser precedence graph. The three series correspond to instances with an $OS \approx 0.2$, $OS \approx 0.6$, and $OS \approx 0.8$ (as shown in Table 1). To construct an instance with three products, one simply needs to select three consecutive instances from the same series.

Additionally, the cycle time $C^{(p)}$ corresponding to the product $p \in P$ is set to $\left\lceil 1.5 \cdot \max_{i \in V} t_i^{(p)} \right\rceil$, whereas the number of machines and the maximal number of modules per machine are respectively fixed to

$$|W| = \max_{p \in P} \left\{ \left\lceil 1.2 \cdot \sum_{i \in V} t_i^{(p)} / C^{(p)} \right\rceil \right\} \quad \text{and}$$

$$m_{\max} = \max_{p \in P} \max \left\{ k : \sum_{i=1}^{k} t_{\pi_i}^{(p)} \leq C^{(p)} \right\},$$

where $(\pi_1, \pi_2, \ldots, \pi_{|V|})$ is a permutation of V according to the non-decreasing order of the task processing times $t_i^{(p)}$, where $i \in V$ and $p \in P$.

5.1. Comparison of ILP formulations

The experimental results for small-size instances with $|V| = 20$, using the compact ILP formulation and the ILP reformulation, are presented in Tables 2 and 3, respectively. In both tables, the first column shows the number of products, while the second and third columns indicate the maximum number of tasks per module and the OS series, respectively. The fourth column displays the total number of instances in each series. The number of instances solved to optimality and their average CPU time are given in the fifth and sixth columns, respectively. Finally, the last column shows the average GAP for the remaining instances that were not solved to optimality within the specified solving time limit of 600 s. The computation of GAP involves comparing the objective function value of the best-found solution (UB) with the lower bound value (LB) determined by CPLEX. It is expressed as $\texttt{GAP} = 100\% \cdot (\texttt{UB} - \texttt{LB})/\texttt{UB}$.

Analyzing the results reported in Table 2, it is evident that the compact ILP formulation is inefficient, with only 11% of instances solved to optimality and a relatively high GAP for the remaining 89% of them. Another noteworthy observation is that as the OS of instances increases, so does the average GAP. This is mainly attributed to the increasing number of precedence constraints, which burdens the model and makes it challenging for CPLEX to improve the upper and lower bounds. Hence, based on these findings, testing the compact ILP formulation on larger instances was deemed unnecessary.

Table 2
Computational results for the compact ILP formulation for $|V| = 20$.

$\|P\|$	$r_{\max}$	OS	#Instances	#OPT	Avg. CPU, (s.)	Avg. GAP, (%)
2	2	1	3	3	177.77	⊕
		2	199	74	293.05	20.86
		3	68	10	331.32	21.34
	3	1	3	2	11.21	10.00
		2	199	29	222.50	23.97
		3	68	4	251.27	25.57
3	2	1	2	0	⊖	21.20
		2	193	1	568.00	41.60
		3	65	0	⊖	41.90
	3	1	2	0	⊖	43.70
		2	193	0	⊖	52.04
		3	65	0	⊖	51.24

(⊕) All optimal solutions were found within the time limit of 600 s.
(⊖) No optimal solution was found within the time limit of 600 s.

In contrast to the compact ILP formulation, the ILP reformulation shows better performance on the same instances. The obtained results, as shown in Table 3, indicate that 99% of instances corresponding to $|P| = 2$ were solved to optimality in less than a minute. For these instances, the number of generated modules for $r_{\max} = 2$ (resp. $r_{\max} = 3$) ranges from 79 to 192 (resp. 81 to 722), with an average of 138.43 (resp. 365.95). As expected, the number of generated modules increases with both $r_{\max}$ and the number of products, making the problem more challenging. This can be observed in the results, where less instances with $|P| = 3$ were solved to optimality. However, the ILP reformulation was still able to reach the optimal solution for 92% of the instances. For the remaining 8%, the obtained average GAP is relatively low. Similar observations can be made for instances with $|P| = 4$, where the ILP reformulation found optimal solutions for 80% of the instances within a reasonable CPU time.

When dealing with medium size instances with $|V| = 50$, the ILP reformulation reaches its limit as it can be seen in Table 4. Out of the 614 instances with $|P| = 2$ considered in the experiments, only 33 were solved to optimality by the solver, while the rest remained unsolved within the time limit. Moreover, the relatively high average GAP for these unsolved instances suggests that they were difficult for the solver to handle. This difficulty can be attributed to the larger number of constraints and decision variables resulting from a larger set of feasible generated modules, where the average number of generated modules for instances with $|P| = 2$ (resp. $|P| = 3$) is 2039.25 (resp. 2512.76) ranging from 266 to 15 854 (resp. 311 to 17 565) modules.

5.2. Comparison of the ILP reformulation with and without module filtering

In this subsection, we compare the ILP reformulation with the heuristic approach that uses a filtering technique. Fig. 9 displays two bar plots for $r_{\max} = 2$ and $r_{\max} = 3$, respectively. Each plot shows the

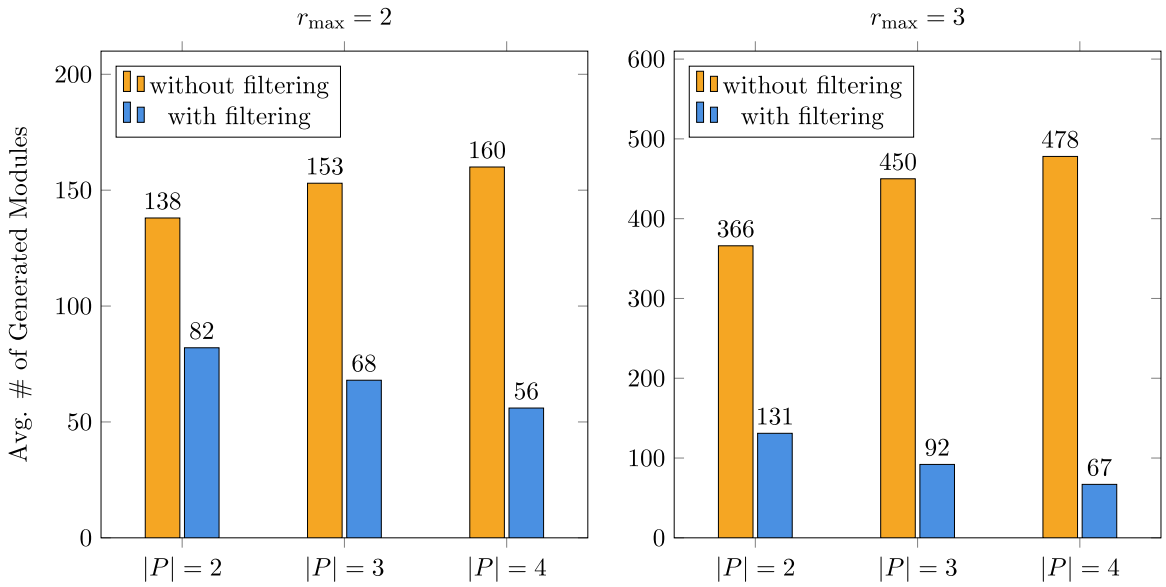

Fig. 9. A comparison between the average number of generated modules with and without the filtering algorithm for $|V| = 20$.

Table 3
Computational results for the ILP reformulation for $|V| = 20$.

$\|P\|$	$r_{\max}$	OS	#Instances	#OPT	Avg. CPU, (s.)	Avg. GAP, (%)
2	2	1	3	3	1.72	⊕
		2	199	199	8.36	⊕
		3	68	68	2.39	⊕
	3	1	3	3	2.93	⊕
		2	199	193	51.66	16.28
		3	68	68	11.03	⊕
3	2	1	2	2	1.25	⊕
		2	193	190	48.47	9.10
		3	65	65	11.57	⊕
	3	1	2	2	5.32	⊕
		2	193	158	106.08	28.15
		3	65	64	20.32	11.10
4	2	1	1	1	3.95	⊕
		2	188	174	64.45	14.98
		3	62	62	52.36	⊕
	3	1	1	1	28.23	⊕
		2	188	101	128.51	34.64
		3	62	58	94.46	22.78

(⊕) All optimal solutions were found within the time limit of 600 s.

Table 4
Computational results for the ILP reformulation for $|V| = 50$.

$\|P\|$	$r_{\max}$	OS	#Instances	#OPT	Avg. CPU, (s.)	Avg. GAP, (%)
2	2	1	15	0	⊖	50.00
		2	218	8	241.86	37.23
		3	74	10	348.74	15.61
	3	1	15	0	⊖	59.70
		2	218	5	388.86	57.86
		3	74	10	270.91	36.89
3	2	1	14	0	⊖	50
		2	218	0	⊖	48.68
		3	73	1	35.76	31.41
	3	1	14	0	⊖	74.49
		2	218	0	⊖	59.68
		3	73	0	⊖	45.84

(⊖) No optimal solution was found within the time limit of 600 s.

average number of modules generated with and without the filtering algorithm for small-sized instances ($|V| = 20$) with $|P| = 2$, $|P| = 3$, and $|P| = 4$. The orange bars represent the average number of modules without filtering, while the blue bars represent the average number of modules with filtering. It is evident from the figure that the module filtering approach significantly reduces the size of the set of modules. Additionally, it is worth noting that the number of common modules decreases with an increase in the number of products. This indicates that the search space becomes narrower as the number of products increases, leading to a faster attainment of good quality solutions.

Table 5 presents a comparison of computational results between the ILP reformulation and the heuristic approach for small-size instances with $|V| = 20$. The first three columns are similar to those in previous tables. The fourth column displays the average solving time in seconds for the exact ILP reformulation denoted as $\texttt{CPU}_\texttt{E}$. The next column shows the average solving time in seconds for the heuristic approach, denoted as $\texttt{CPU}_\texttt{H}$. Finally, the last two columns represent the average percentage GAP between the best found upper bound and the upper bound found by the exact and heuristic approaches. These gaps are computed as: $\texttt{GAP}^\texttt{B}_\texttt{E} = 100\% \cdot (\texttt{UB}_\texttt{E} - \texttt{UB}_\texttt{B})/\texttt{UB}_\texttt{B}$ for the exact approach and $\texttt{GAP}^\texttt{B}_\texttt{H} = 100\% \cdot (\texttt{UB}_\texttt{H} - \texttt{UB}_\texttt{B})/\texttt{UB}_\texttt{B}$ for the heuristic approach, where $\texttt{UB}_\texttt{E}$ (resp. $\texttt{UB}_\texttt{H}$) is the upper bound found by the exact (resp. heuristic) approach, and $\texttt{UB}_\texttt{B} = \min\{\texttt{UB}_\texttt{E}, \texttt{UB}_\texttt{H}\}$.

Analyzing Table 5, it is interesting to notice that the heuristic approach outperforms the exact one. In fact, the heuristic approach is able to find the same optimal solution for all the instances that are solved to optimality by the ILP reformulation. Furthermore, for all the instances that were not solved to optimality using the exact approach, the heuristic approach was able to find solutions of better quality. For example, for the instances corresponding to $|P| = 3$, $r_{\max} = 3$, and $OS = 2$, the heuristic approach has found solutions with an objective function value that was, on average, 20% better than those found by the exact approach. In terms of solving time, the heuristic approach was significantly faster than the ILP reformulation.

The comparison between the ILP reformulation and the heuristic approach for medium size instances with $|V| = 50$ is presented in Table 6. Similarly to the previous table, the heuristic approach outperforms the exact one in terms of finding the best quality solutions. This is evidenced by the fact that $\texttt{GAP}^\texttt{B}_\texttt{H}$ is between 0 and 2.33 on average, while $\texttt{GAP}^\texttt{B}_\texttt{E}$ shows that the exact approach finds solutions that can be 43%, on average, worse than the best solution found. However, both approaches reach the time limit of 600 s for most instances, as shown by the CPU time columns. This means that the solver is interrupted even for the heuristic approach, and the solution can potentially be improved if more time was allowed.

Table 5
Comparison of computational results between the ILP reformulation and the heuristic approach for $|V| = 20$.

$\|P\|$	r_{max}	OS	Avg. CPU_E, (s.)	Avg. CPU_H, (s.)	Avg. GAP^B_E, (%)	Avg. GAP^B_H, (%)
2	2	1	1.72	0.39	0.00	0.00
		2	8.36	1.64	0.00	0.00
		3	2.39	0.40	0.00	0.00
	3	1	2.93	0.43	0.00	0.00
		2	51.66	5.22	0.26	0.00
		3	11.03	0.25	0.00	0.00
3	2	1	1.25	0.87	0.00	0.00
		2	57.04	7.89	0.00	0.00
		3	11.57	1.12	0.00	0.00
	3	1	5.32	0.60	0.00	0.00
		2	195.77	23.51	5.03	0.00
		3	29.24	0.69	0.00	0.00
4	2	1	3.95	0.58	0.00	0.00
		2	104.33	18.36	0.79	0.00
		3	52.36	1.87	0.00	0.00
	3	1	28.23	0.61	0.00	0.00
		2	347.12	43.10	20.72	0.00
		3	127.08	1.51	1.22	0.00

Table 6
Comparison of computational results between the ILP reformulation and the heuristic approach for $|V| = 50$.

$\|P\|$	r_{max}	OS	Avg. CPU_E, (s.)	Avg. CPU_H, (s.)	Avg. GAP^B_E, (%)	Avg. GAP^B_H, (%)
2	2	1	600	600	7.24	0.00
		2	587.42	440.18	41.37	2.14
		3	566.19	469.49	10.36	2.14
	3	1	600	600	6.59	0.00
		2	595.36	539.03	26.96	0.41
		3	556.44	454.75	43.44	2.33
3	2	1	600	600	0.00	0.00
		2	600	585.80	25.18	0.04
		3	592	565.11	24.51	1.34
	3	1	600	600	0.00	0.00
		2	600	574.45	16.05	0.03
		3	600	559.24	41.94	0.87

The obtained results showed in Table 6 indicate that medium size instances are challenging even when using the filtering approach. Although the number of generated modules can be reduced as depicted in Fig. 10, the problem size remains sufficiently large and therefore more challenging to solve for the commercial solver.

6. Conclusion and perspectives

We have addressed the design of multi-product modular reconfigurable lines in this paper. While existing research has focused on designing single product lines, this paper proposes a solution for designing multi-product lines composed of modular machines. Each machine has a limited number of slots for installing modules, and the specific modules required for each product are not known in advance. The goal is to design a line configuration that meets the cycle time and task precedence constraints while minimizing the total number of modules used.

This paper introduces a compact ILP formulation that assigns tasks to modules and then allocates them to available machines, as well as an ILP reformulation based on a decomposition approach. The latter involves generating the set of all feasible modules of limited size from a precedence graph, followed by using the ILP reformulation to select the optimal number of modules from this set. Additionally, a new heuristic based on a module filtering algorithm is proposed, which reduces the set of generated modules by selecting only common modules between all the products.

Computational results demonstrate that the compact ILP formulation is not efficient for solving small-size instances, while the ILP reformulation and the heuristic approach are able to solve the same instances to optimality in a matter of seconds. For medium-size instances, the heuristic approach outperforms the ILP reformulation.

The limitations of the heuristic approach in solving medium-size instances suggest the need for exploring alternative methods such as heuristics or meta-heuristics, independent of commercial solvers. Additionally, investigating economic trade-offs between size and number of modules could be a fruitful avenue for future research. Specifically, larger modules may reduce the number of modules required, but they are also more costly to design and maintain, while smaller modules may increase the cost and effort reconfiguration. Understanding this relationship can help in making more informed decisions in the design of modular lines.

Funding

This work was financially supported by the IRT PERFORM program, managed by IRT Jules Verne (French Institute in Research and Technology in Advanced Manufacturing).

CRediT authorship contribution statement

Abdelkrim R. Yelles-Chaouche: Writing – review & editing, Writing – original draft, Visualization, Validation, Software, Methodology, Investigation, Formal analysis, Conceptualization. **Evgeny Gurevsky:** Writing – review & editing, Validation, Supervision, Methodology, Investigation, Formal analysis, Data curation, Conceptualization. **Nadjib Brahimi:** Writing – review & editing, Validation, Supervision, Formal analysis. **Alexandre Dolgui:** Writing – review & editing, Supervision, Project administration, Funding acquisition.

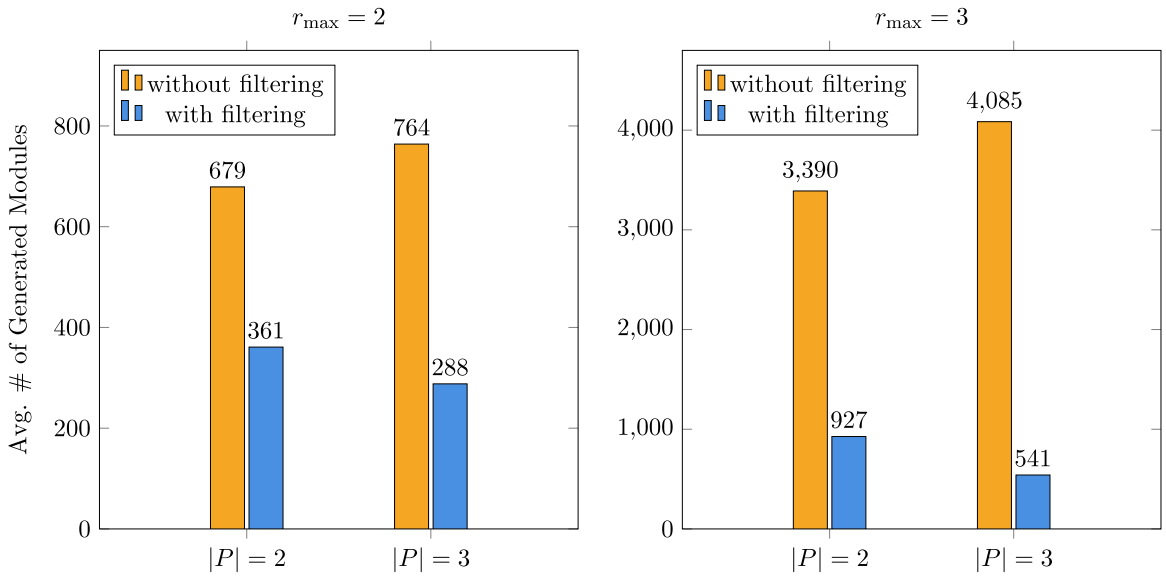

Fig. 10. A comparison between the average number of generated modules with and without the filtering algorithm for $|V| = 50$.

Data availability

Data will be made available on request.

Appendix. Slot-based ILP reformulation

Decision variables:

- y_m is equal to 1 if the module $m \in M$ is used in at least one configuration, 0 otherwise.
- $\lambda_{me}^{(p)}$ is equal to 1 if the module $m \in M$ is assigned to the slot $e \in E$ in the configuration corresponding to the product $p \in P$, 0 otherwise.

$$\min \sum_{m \in M} y_m \tag{A.1}$$

$$\sum_{m \in M} \sum_{e \in E} \alpha_{im} \cdot \lambda_{me}^{(p)} = 1, \quad \forall i \in V, \quad \forall p \in P \tag{A.2}$$

$$\sum_{m \in M} \sum_{e \in E_k} c_m^{(p)} \cdot \lambda_{me}^{(p)} \leq C^{(p)}, \quad \forall k \in W, \quad \forall p \in P \tag{A.3}$$

$$\lambda_{me}^{(p)} \leq y_m, \quad \forall m \in M, \quad \forall e \in E, \quad \forall p \in P \tag{A.4}$$

$$\sum_{m \in M} \sum_{e \in E} e \cdot \alpha_{im} \cdot \lambda_{me}^{(p)} \leq \sum_{m \in M} \sum_{e \in E} e \cdot \alpha_{jm} \cdot \lambda_{me}^{(p)}, \quad \forall (i,j) \in A^{(p)}, \ \forall p \in P \tag{A.5}$$

$$\sum_{m \in M \setminus M_k^{(p)}} \lambda_{me}^{(p)} = 0, \quad \forall e \in E_k, \ \forall k \in W, \ \forall p \in P \tag{A.6}$$

$$\sum_{m \in M} \lambda_{m,e+1}^{(p)} \leq \sum_{m \in M} \lambda_{me}^{(p)}, \quad \forall e \in E_k \setminus \{k \cdot m_{\max}\}, \ \forall k \in W, \ \forall p \in P \tag{A.7}$$

$$y_m \in \{0,1\}, \quad \forall m \in M$$

$$\lambda_{me}^{(p)} \in \{0,1\}, \quad \forall m \in M, \ \forall e \in E, \ \forall p \in P$$

The objective function (A.1) remains the same as (18). Constraints (A.2) imply that each task of each product has to be assigned to one machine slot. Inequalities (A.3) ensure that for each machine, the cycle time constraint corresponding to a given product is satisfied. Constraints (A.4) express that a module is considered as 'used' if it is assigned to at least one product configuration. Expressions (A.5) guarantee that task precedence constraints between machine slots are respected. Equalities (A.6) prevent each module $m \in M_k^{(p)}$ from being assigned in some machine slots. Finally, constraints (A.7) are used to break symmetry by stating that a module cannot be assigned to the slot $e+1 \in E_k$, if the slot $e \in E_k$ is empty.

Table A.7
Computational results for the slot-based ILP reformulation for $|V| = 20$.

$\|P\|$	$r_{\max}$	OS	#Instances	#OPT	Avg. CPU, (s.)	Avg. GAP, (%)
2	2	1	3	3	5.40	⊕
		2	199	194	40.56	9.10
		3	68	68	17.28	⊕
	3	1	3	3	23.00	⊕
		2	199	144	131.68	25.11
		3	68	65	36.82	16.63
3	2	1	2	1	27.64	9.10
		2	193	126	197.11	19.46
		3	65	51	127.36	15.65
	3	1	2	2	251.98	⊕
		2	193	57	81.69	50.05
		3	65	40	97.20	32.52

(⊕) All optimal solutions were found within the time limit of 600 s.

The slot-based reformulation was tested on the same instances with the same parameters and using the same computer. The obtained results, presented in Table A.7, demonstrate that this new reformulation does not perform as well as the originally proposed one, namely (18)–(25). This is primarily due to the large number of slots, which results in a significant number of generated decision variables and constraints. To provide a rough estimation, for instances corresponding to $|P| = 2$ and $r_{\max} = 2$, the number of slots per machine typically ranges from approximately 40 to 100. When comparing the number of instances solved to optimality, the new formulation solves 754 instances over 1060, whereas formulation (18)–(25) solves 1015 instances as presented in Table 4. It should also be noted that the computational time required for solving instances with the new reformulation is substantially longer than for formulation (18)–(25).

References

Arcus, A. L. (1965). A computer method of sequencing operations for assembly lines. *International Journal of Production Research*, *4*(4), 259–277.

Ashraf, M., & Hasan, F. (2018). Configuration selection for a reconfigurable manufacturing flow line involving part production with operation constraints. *International Journal of Advanced Manufacturing Technology*, *98*(5–8), 2137–2156.

Baldwin, C. Y., & Clark, K. B. (2006). Modularity in the design of complex engineering systems. In *Understanding complex systems* (pp. 175–205). Springer Berlin Heidelberg.

Battaïa, O., & Dolgui, A. (2013). A taxonomy of line balancing problems and their solution approaches. *International Journal of Production Economics*, *142*(2), 259–277.

Battaïa, O., & Dolgui, A. (2022). Hybridizations in line balancing problems: A comprehensive review on new trends and formulations. *International Journal of Production Economics*, *250*, Article 108673.

Battaïa, O., Gurevsky, E., Makssoud, F., & Dolgui, A. (2012). Equipment location in machining transfer lines with multi-spindle heads. *Journal of Mathematical Modelling and Algorithms in Operations Research*, *12*(2), 117–133.

Battini, D., Faccio, M., Ferrari, E., Persona, A., & Sgarbossa, F. (2006). Design configuration for a mixed-model assembly system in case of low product demand. *International Journal of Advanced Manufacturing Technology*, *34*(1–2), 188–200.

Baybars, İ. (1986). A survey of exact algorithms for the simple assembly line balancing problem. *Management Science*, *32*(8), 909–932.

Belmokhtar, S., Dolgui, A., Guschinsky, N., & Levin, G. (2006). Integer programming models for logical layout design of modular machining lines. *Computers & Industrial Engineering*, *51*(3), 502–518.

Berger, I., Bourjolly, J.-M., & Laporte, G. (1992). Branch-and-bound algorithms for the multi-product assembly line balancing problem. *European Journal of Operational Research*, *58*(2), 215–222.

Borisovsky, P. A., Delorme, X., & Dolgui, A. (2014). Balancing reconfigurable machining lines via a set partitioning model. *International Journal of Production Research*, *52*(13), 4026–4036.

Bryan, A., Jack Hu, S., & Koren, Y. (2013). Assembly system reconfiguration planning. *Journal of Manufacturing Science and Engineering*, *135*(4), 13.

Bukchin, Y., & Rabinowitch, I. (2006). A branch-and-bound based solution approach for the mixed-model assembly line-balancing problem for minimizing stations and task duplication costs. *European Journal of Operational Research*, *174*(1), 492–508.

Bukchin, J., & Rubinovitz, J. (2003). A weighted approach for assembly line design with station paralleling and equipment selection. *IIE Transactions*, *35*(1), 73–85.

Bukchin, J., & Tzur, M. (2000). Design of flexible assembly line to minimize equipment cost. *IIE Transactions*, *32*(7), 585–598.

Dou, J., Dai, X., & Meng, Z. (2009). Optimisation for multi-part flow-line configuration of reconfigurable manufacturing system using GA. *International Journal of Production Research*, *48*(14), 4071–4100.

Dou, J., Li, J., & Su, C. (2016). Bi-objective optimization of integrating configuration generation and scheduling for reconfigurable flow lines using NSGA-II. *International Journal of Advanced Manufacturing Technology*, *86*(5–8), 1945–1962.

Erel, E., & Gokcen, H. (1999). Shortest-route formulation of mixed-model assembly line balancing problem. *European Journal of Operational Research*, *116*(1), 194–204.

Erel, E., & Sarin, S. C. (1998). A survey of the assembly line balancing procedures. *Production Planning and Control*, *9*(5), 414–434.

Graves, S. C., & Redfield, C. H. (1988). Equipment selection and task assignment for multiproduct assembly system design. *International Journal of Flexible Manufacturing Systems*, *1*(1), 31–50.

Huang, C.-C., & Kusiak, A. (1998). Modularity in design of products and systems. *IEEE Transactions on Systems, Man, and Cybernetics - Part A: Systems and Humans*, *28*(1), 66–77.

Koren, Y., Heisel, U., Jovane, F., Moriwaki, T., Pritschow, G., Ulsoy, G., & Van Brussel, H. (1999). Reconfigurable manufacturing systems. In *Manufacturing technologies for machines of the future* (pp. 627–665). Springer Berlin Heidelberg.

Lapierre, S. D., Debargis, L., & Soumis, F. (2000). Balancing printed circuit board assembly line systems. *International Journal of Production Research*, *38*(16), 3899–3911.

Nazarian, E., Ko, J., & Wang, H. (2010). Design of multi-product manufacturing lines with the consideration of product change dependent inter-task times, reduced changeover and machine flexibility. *Journal of Manufacturing Systems*, *29*(1), 35–46.

Otto, A., Otto, C., & Scholl, A. (2013). Systematic data generation and test design for solution algorithms on the example of SALBPGen for assembly line balancing. *European Journal of Operational Research*, *228*(1), 33–45.

Patterson, J. H., & Albracht, J. J. (1975). Assembly-line balancing: Zero-one programming with Fibonacci search. *Operations Research*, *23*(1), 166–172.

Pereira, J. (2018). Modelling and solving a cost-oriented resource-constrained multi-model assembly line balancing problem. *International Journal of Production Research*, *56*(11), 3994–4016.

Saxena, L. K., & Jain, P. K. (2012). A model and optimisation approach for re-configurable manufacturing system configuration design. *International Journal of Production Research*, *50*(12), 3359–3381.

Scholl, A. (1999). *Balancing and sequencing of assembly lines*. Physica-Verlag HD.

Shaik, A. M., Kesava Rao, V. V. S., & Srinivasa Rao, C. (2015). Development of modular manufacturing systems – a review. *International Journal of Advanced Manufacturing Technology*, *76*(5–8), 789–802.

Tóth, A., Knuutila, T., & Nevalainen, O. S. (2018). Machine configuration and workload balancing of modular placement machines in multi-product PCB assembly. *International Journal of Computer Integrated Manufacturing*, *31*(9), 815–830.

Tremblet, D., Yelles-Chaouche, A. R., Gurevsky, E., Brahimi, N., & Dolgui, A. (2023). Optimizing task reassignments for reconfigurable multi-model assembly lines with unknown order of product arrival. *Journal of Manufacturing Systems*, *67*, 190–200.

van Zante-de Fokkert, J. I., & de Kok, T. G. (1997). The mixed and multi model line balancing problem: a comparison. *European Journal of Operational Research*, *100*(3), 399–412.

Yelles-Chaouche, A. R., Gurevsky, E., Brahimi, N., & Dolgui, A. (2021). Reconfigurable manufacturing systems from an optimisation perspective: a focused review of literature. *International Journal of Production Research*, *59*(21), 6400–6418.

Yelles-Chaouche, A. R., Gurevsky, E., Brahimi, N., & Dolgui, A. (2022). Minimizing task reassignments under balancing multi-product reconfigurable manufacturing lines. *Computers & Industrial Engineering*, *173*, Article 108660.

Youssef, A. M. A., & ElMaraghy, H. A. (2006). Modelling and optimization of multiple-aspect RMS configurations. *International Journal of Production Research*, *44*(22), 4929–4958.

Youssef, A. M. A., & ElMaraghy, H. A. (2007). Optimal configuration selection for reconfigurable manufacturing systems. *International Journal of Flexible Manufacturing Systems*, *19*(2), 67–106.

Youssef, A. M. A., & ElMaraghy, H. A. (2008). Availability consideration in the optimal selection of multiple-aspect RMS configurations. *International Journal of Production Research*, *46*(21), 5849–5882.