Journal of Manufacturing Systems 67 (2023) 190–200

Contents lists available at ScienceDirect

Journal of Manufacturing Systems

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

Technical paper

Optimizing task reassignments for reconfigurable multi-model assembly lines with unknown order of product arrival

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

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

ARTICLE INFO

Keywords:
Manufacturing
Reconfigurability
Multi-model
Assembly line balancing
Assembly line design
MILP
Heuristics
Constraint generation
Reconfigurable manufacturing systems

ABSTRACT

This paper deals with the multi-model assembly line balancing problem (MuMALBP) in a reconfigurable environment. The considered line is composed of a fixed number of workstations and can produce different products in batches. Each product requires an appropriate line configuration. Thus, when the product changes, the line has to be reconfigured to satisfy new requirements related to task precedence and cycle time constraints. Reconfiguring the line consists in reassigning certain tasks between the existing workstations. The objective of this paper is to design a line configuration for each product while minimizing the maximum number of task reassignments whatever the sequence of product arrival. To solve this NP-hard problem, a mixed-integer linear program (MILP) is first formulated. Subsequently, a constructive heuristic and a MILP-based heuristic, named Halt-and-Fix, are developed to solve large-size problem instances. All the approaches are tested and approved on a dataset derived from the well-known instances of the assembly line balancing literature. The numerical results show that the Halt-and-Fix heuristic provides a better trade-off between solution quality and CPU time, compared to the constructive heuristic and the MILP formulation.

1. Introduction

Nowadays, integrating reconfigurability and product diversity in the design phase of production lines is a major challenge for industry and academic researcher. This can be noticed in recent literature reviews such as the one proposed by Bortolini et al. [1]. Indeed, this phase requires making important decisions such as layout design, process planning, and line balancing. While many works have addressed layout design and process planning problems, few consider the line balancing problem in a multi-product reconfigurable environment.

Line balancing is an important and crucial step in the design phase as it has a direct impact on the performance of the line. More specifically, it aims at allocating a given set of tasks, required to manufacture a product, to a set of workstations composing the line. Mainly, this has to be done under various constraints such as task precedence relations and a desired production rate. The former defines a partial order, usually represented by a directed acyclic graph, according to which the tasks have to be executed, while the latter is defined by a cycle time that indicates the period between the release of two successive products. Mostly studied for simple assembly lines, the line balancing is usually viewed in the literature as an optimization problem [see, *e.g.*, 2] and called consequently as the simple assembly line balancing problem (SALBP).

The SALBP aims to design a line for a single product only. However, when dealing with different products, two extensions of SALBP can be found in the literature [see, *e.g.*, 3]. The first one, named as mixed-model assembly line balancing problem (MiMALBP), arises when different products are produced simultaneously on the same line. Such a line is typically used when the reconfiguration effort, to switch from one product type to another, can be reduced enough to be neglected. If this is not the case, then a multi-model assembly line balancing problem (MuMALBP) is considered, where each product is performed separately in batches and the line has to be reconfigured accordingly. Fig. 1 illustrates the three different line types according to the product flow. In this figure, a single model line, associated with SALBP, can produce one type of product only, represented by a triangle shape. In a mixed-model line, several product types having

* Corresponding author at: Mines Saint-Étienne, LIMOS, Gardanne, France.
E-mail address: abdelkrim.yelles@emse.fr (A.R. Yelles-Chaouche).

https://doi.org/10.1016/j.jmsy.2023.02.001
Received 23 September 2022; Received in revised form 10 January 2023; Accepted 1 February 2023

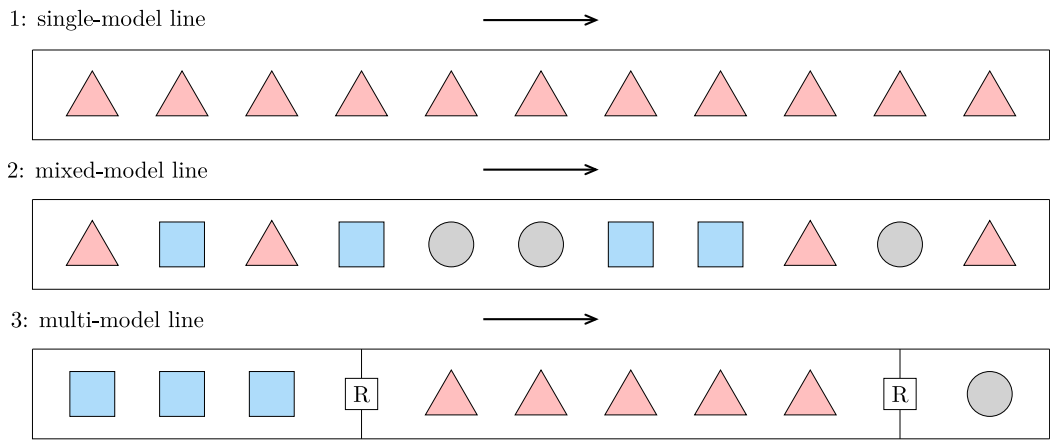

Fig. 1. Three types of assembly line for single or multiple products.

triangular, round and square shapes go through the line simultaneously. Finally, a multi-model line is able to handle several product types in separate batches. Thus, a reconfiguration denoted as "R", is required to switch from the square shape type to the triangle one (and from triangle to round shape). Usually, the reconfiguration is associated with time-consuming and costly manipulations involving the reassignment of some tasks between available workstations. Therefore, optimizing the number of these reassignments is a logical goal for MuMALBP. Despite its importance and an actual growing interest in reconfigurable manufacturing systems (RMS), such an objective has received little attention in the literature [see, *e.g.*, 4]. This was also noticed through the recent literature review of Battaïa and Dolgui [5] where they reported such problem among the underdeveloped topics in the literature of ALBP. Hence, in the present paper, we try to fill this gap and study such a problem in the valuable context of RMS, where no information is available regarding the sequence of product arrival.

The article is organized into seven sections. Section 2 reviews the literature on MuMALBP. Section 3 provides a detailed description of the studied problem. In Section 4, its MILP formulation is proposed. Subsequently, a constructive heuristic and a MILP-based heuristic are introduced in Section 5. Section 6 presents numerical experiments and results. Finally, the conclusion and perspectives are provided in Section 7.

2. Literature review

RMS was introduced by Koren et al. [6] as a solution to the fluctuating nature of today's market, characterized by high product diversity, shared production assets, and volatile supply and demand [7]. Indeed, RMS is mainly composed of reconfigurable machines (RM), which have a modular structure allowing them to be easily reconfigured by adding, moving, or removing modules. The design of an RMS is a challenging problem and several approaches have been proposed in the literature to solve it. However, most authors consider reconfiguration as a parameter associated with RM, while few have attempted to optimize the cost, time, or effort associated with the reconfiguration process [see, *e.g.*, 4].

There are very few studies about the MuMALBP in the literature [5, 8]. In fact, while SALBP and MiMALBP constitute a large majority of studies, there are few publications on MuMALBP. This can be seen through the classification provided by Boysen et al. [9], where the number of studies related to MiMALBP is three times more than those related to MuMALBP. Furthermore, the few studies on the multi-model problem usually simplify it to make it easier to solve. For example, some papers dealing with MuMALBP consider it as a derived version of MiMALBP or try to convert it to SALBP [see, *e.g.*, 10]. To do so, a commonly used method consists in combining the precedence graphs and adapting the task processing times of each product [see 11]. The advantage of this approach is that there is no need to reconfigure the line. This usually leads to a solution with a performance trade-off between the different products, which may decrease its overall efficiency. In addition, the technique for combining precedence graphs works efficiently only when the graphs are similar [see 12]. However, when the objective is to minimize the number of task reassignments (or the associated time and cost), such approaches may provide very poor bounds which cannot be efficiently exploited.

Table 1 presents the most relevant publications on MuMALBP. The second column of the table indicates the objective(s) to minimize or maximize; the third column states the different solution approaches; and finally the last one indicates whether the study was based on or related to an industrial application.

Most of the industrial applications are found in the electronics industry. Kabir and Tabucanon [15] consider the assembly of printing calculators, which are still commonly used in some applications. Lapierre et al. [17] and Koskinen et al. [26] study the problem of placing components on printed circuit boards (PCBs). Such a problem is characterized by a variety of PCB types and multiple placement machines for the components. Pastor et al. [18] explore a problem in which 4 types of white goods are produced. The latter application in the electronics industry is the manufacturing of liquid crystal displays (LCDs and more particularly the thin film transistor TFT-LCDs). Chen et al. [25] consider such problems while integrating practical characteristics such as multi-skilled worker and operator efficiency. Other applications of MuMALBP include shipbuilding [21], clothing industry [23] and engine manufacturing [24].

As indicated in Table 1, studies on MuMALBP consider different objectives, but most of them attempt either to minimize the number of workstations [11,14,16,19,25] or the cycle time [17,20,22,24]. The reason why most researchers have studied the latter objectives is that these have been the traditional objectives in most line balancing problems. The earliest studies of [13,14] have as objective the minimization of the cost which can be composed of labor costs, inventory carrying costs, and setup costs, among others. Pastor et al. [18] and Qu and Jiang [21] attempt to maximize the yield and efficiency of the production line, respectively. In particular, Pastor et al. [18] consider yield maximization in a multi-objective framework which also includes workload balancing and minimizing the dispersion of worker tasks. Among the above studies, Asl et al. [24] treat a multi-objective MuMALBP, where one of the objectives consists in maximizing the number of common tasks between workstations.

In the same scope, Fisel et al. [27] studied the problem of balancing assembly lines with the objective of optimizing their changeability and flexibility to cope with different scenarios of possible market evolution. The authors defined flexibility as the ability of the line configuration to handle different product mix. As for changeability, it was defined as

Table 1
Summary of publications focusing on MuMALBP.

Reference	Objective	Solution approach	Industrial application
van Zante-de Fokkert and de Kok [11]	Min. number of workstations	Heuristic	×
Buxey et al. [13]	Min. production cost	Simulation	×
Chakravarty and Shtub [14]	Min. total cost	Heuristic	×
Kabir and Tabucanon [15]	Min. number of workstations	Simulation	printing calculators
Kimms [16]	Min. number of workstations	Heuristic + column generation	×
Lapierre et al. [17]	Min. cycle time	Lagrangian relaxation	printed circuit board
Pastor et al. [18]	Max. yield of the line	Tabu search	white goods
Yu and Shi [19]	Min. number of workstations	Genetic algorithm	×
Liao [20]	Min. cycle time	MILP	×
Qu and Jiang [21]	Max. efficiency of the line	Memetic algorithm	shipbuilding
Christensen et al. [22]	Min. cycle time	Heuristic	×
Pereira [23]	Min. cost associated with the workstations	Heuristic	clothing
Asl et al. [24]	Min. cycle time + others	MILP	engine manufacturing
Chen et al. [25]	Min. number of workstations + resources	Genetic algorithm + MILP	liquid-crystal displays
Koskinen et al. [26]	Min. tardiness	Heuristic	printed circuit board
Fisel et al. [27]	Min. reconfiguration cost & Max. flexibility	Multi-objective optimization	car assembly line
Yelles-Chaouche et al. [28]	Min. total number of task reassignments	MILP + MILP-based heuristic	×

the capacity of a balanced line to change its configuration, mainly by reallocating its assembly tasks and resources.

Similarly, Yelles-Chaouche et al. [28] considered the MuMALBP in a reconfigurable environment. Thus, given a sequence of product arrival, the objective is to generate a line configuration for each of them that minimizes the total number of task reassignments when switching from one product configuration to another. However, each product configuration is designed on the basis of a given order of product arrival. This means that the line is not able to adapt if the mentioned order is changed or modified. Therefore, in this paper, the objective is to fill this gap by considering a MuMALBP where a line configuration is designed for each product without knowing their arrival order. The MuMALBP, except for some very specific form of precedence constraints, is known to be NP-hard in the strong sense [see, *e.g.*, 2]. Therefore, most of the solution approaches proposed to solve their different extensions are approximate. Among these approaches in the existing literature, we find meta-heuristics including genetic algorithms and tabu search [18,19,25], simple construction heuristics [14], and Lagrangian based approximate methods [17].

Based on the above-described observations and the identified gaps, we can summarize our contributions as follows:

- Studying the problem of MuMALBP in a reconfigurable environment while considering that the order of product arrival is unknown,
- Proposing a new MILP formulation to generate product configurations that minimize the number of task reassignments between workstations when switching from one product batch to another,
- Design and implementation of a constructive heuristic as well as a MILP-based heuristic, named as Halt-and-Fix.

3. Problem description and formulation

In this section, a detailed description of the problem is given and a corresponding mathematical formulation is introduced.

3.1. Problem definition

We consider a reconfigurable manufacturing line processing several products from the same family in separate batches. In order to be finished, each product requires the execution of a given set of tasks. Each task is characterized by a processing time and has to be executed following the so-called precedence constraints. Since the products belong to the same family, we consider that they share the same set of tasks. However, task processing times, precedence constraints, as well as cycle times, may be different. As a consequence, an admissible distinct line configuration is necessary for each product. Such a configuration is referred to as an assignment of tasks to a given number of workstations so that the corresponding precedence and cycle time constraints are satisfied. Since the line is reconfigurable, it is possible to switch from one product configuration to another, called reconfiguration, by reassigning certain tasks between the existing workstations. For example, in machining centers of mechanical parts, a task reassignment can be seen as a relocation of a tool from one machine to another. Without loss of generality, we suppose that all the tasks are compatible with all the workstations. Moreover, we do not consider the inclusion and exclusion constraints that force certain tasks to be assigned, or not, to the same workstation.

In order to ensure fast and smooth reconfiguration, it is important to optimize the number of task reassignments performed when moving from one configuration to another. This is especially important when the sequence in which batches arrive is not known in advance. In this case, it is relevant to design the line so that the maximum number of task reassignments between any two line configurations is minimized.

To better illustrate the above-described optimization problem, Fig. 2 shows an example where three products, expressed by their corresponding precedence graphs, are considered. The cycle times are 30, 29, and 27, respectively, and the manufacturing line is composed of 4 workstations. The right-hand side of Fig. 2 shows an optimal solution represented by three line configurations corresponding to the three products. For this solution, we can notice that 3 task reassignments would be necessary to switch from the first line configuration to the second one. Similarly, 5 (resp. 4) tasks should be reassigned to move from the third to the second (resp. from the third to the first) line configuration. Here, 5, representing the objective function value for this solution, is the maximum number of task reassignments between any two line configurations.

It is important to note that the studied problem is hard to solve since even finding an admissible line configuration for a given product is equivalent to the SALBP-F problem, which is known to be NP-complete in the strong sense [see, *e.g.,* 2].

3.2. Mathematical formulation

The problem of minimizing the maximum number of task reassignments is formulated in this paper as a MILP model using the following notations and variables.

Notations:

- V is the set of tasks,
- W is the set of available workstations,
- P is the set of products,
- $U = \{(p, q) \in P \times P : p < q\}$ is the set of all possible ordered pairs of products,
- $C^{(p)}$ is the cycle time associated with the product $p \in P$,
- $t_i^{(p)}$ is the processing time of the task $i \in V$ for the product $p \in P$,

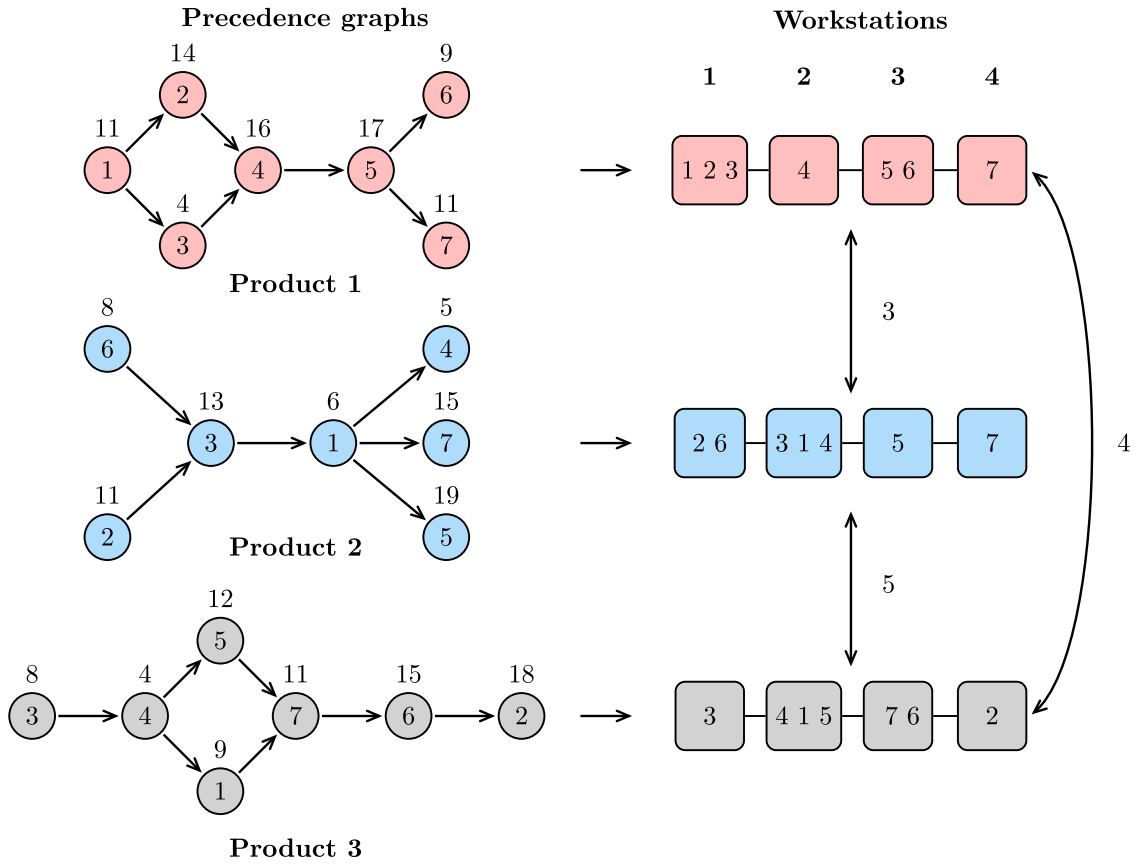

Fig. 2. Optimal line configurations for 3 products with their given precedence graphs.

- $Q_i^{(p)}$ is the so-called assignment interval, associated with a set of workstations of the configuration corresponding to the product $p \in P$, where task i can be potentially assigned,
- $G^{(p)} = (V, A^{(p)})$ is the directed acyclic graph corresponding to the precedence constraints of the product $p \in P$. Here, V is the set of tasks associated with the set of vertices of $G^{(p)}$ and $A^{(p)}$ is the set of arcs between tasks, where $(i, j) \in A^{(p)}$ means that task j has to be assigned to the same workstation as task i, or to a succeeding one.

Variables:

- $x_{ik}^{(p)}$ is equal to 1 if the task $i \in V$ is assigned to the workstation $k \in W$ in the configuration corresponding to the product $p \in P$, 0 otherwise,
- $z_{ik}^{(p,q)}$ is equal to 1 if the task $i \in V$ is assigned to the same workstation $k \in W$ in both configurations corresponding to products p and q, 0 otherwise,
- $r^{(p,q)}$ is the number of task reassignments between two configurations corresponding to products p and q,
- $r_{\max}$ is the maximum number of task reassignments between any two configurations.

$$\min \quad r_{\max} + \frac{1}{|V| \cdot |U|} \cdot \sum_{(p,q) \in U} r^{(p,q)} \tag{1}$$

subject to:

$$|V| - \sum_{i \in V} \sum_{k \in W} z_{ik}^{(p,q)} \leq r^{(p,q)} \leq r_{\max}, \quad \forall (p,q) \in U \tag{2}$$

$$z_{ik}^{(p,q)} \leq \frac{1}{2} \cdot \left(x_{ik}^{(p)} + x_{ik}^{(q)} \right), \quad \forall i \in V, \quad \forall k \in W, \quad \forall (p,q) \in U \tag{3}$$

$$\sum_{k \in W} x_{ik}^{(p)} = 1, \quad \forall i \in V, \quad \forall p \in P \tag{4}$$

$$\sum_{q=k}^{|W|} x_{iq}^{(p)} \leq \sum_{q=k}^{|W|} x_{jq}^{(p)}, \quad \forall (i,j) \in A^{(p)}, \ \forall k \in Q_i^{(p)} \bigcap Q_j^{(p)}, \ \forall p \in P \tag{5}$$

$$\sum_{i \in V} t_i^{(p)} \cdot x_{ik}^{(p)} \leq C^{(p)}, \quad \forall k \in W, \quad \forall p \in P \tag{6}$$

$$x_{ik}^{(p)} = 0, \quad \forall i \in V, \ \forall k \notin Q_i^{(p)}, \quad \forall p \in P \tag{7}$$

$$x_{ik}^{(p)} \in \{0,1\}, \quad \forall i \in V, \quad \forall k \in W, \quad \forall p \in P$$

$$z_{ik}^{(p,q)} \in \{0,1\}, \quad \forall i \in V, \quad \forall k \in W, \quad \forall (p,q) \in U$$

$$r^{(p,q)} \geq 0, \quad \forall (p,q) \in U$$

$$r_{\max} \geq 0$$

The principal goal of the objective function (1) is to find a solution minimizing $r_{\max}$. Furthermore, among the solutions having the same value of $r_{\max}$, the compound expression of (1) allows to favor those minimizing the total number of task reassignments that is confirmed below by Lemma 3.1. The number of task reassignments for any two configurations as well as its maximum value is determined by constraints (2). Constraints (3) allow each variable $z_{ik}^{(p,q)}$ to be equal to 1 if and only if task i is assigned to the same workstation k for both configurations corresponding to products p and q. Inequalities (4) are used to ensure that each task $i \in V$ is assigned to exactly one workstation for each product $p \in P$. The respect of precedence constraints for each product is done by expressions (5). These latter impose that any task i that precedes task j cannot be assigned to a workstation that succeeds the one where j is assigned. The cycle time constraint for each product is expressed by (6). Finally, constraints

(7) enhance the MILP model by fixing any variable $x_{ik}^{(p)}$ to 0, if task i cannot be assigned to workstation k in the configuration corresponding to product p with respect to the assignment interval $Q_i^{(p)}$.

It is important to mention that the above-described MILP formulation can handle products that require a different number of tasks. For example, if task j does not belong to product p, it is sufficient to consider this task as a free dummy one for this product and set $t_j^{(p)} = 0$.

The lower (resp. upper) bound of the assignment interval $Q_i^{(p)}$ refers to the earliest (resp. latest) workstation in which task i can be assigned for the configuration corresponding to product p. In this paper, the manner of computing assignment intervals is inspired by one of the approaches, presented in [2, p. 44–47] for computing a lower bound, noted as LB_4, on the number of workstations for the SALBP-1 problem. These latter are calculated here as follows:

$$Q_i^{(p)} = \left[\left\lceil \frac{\theta_i^{(p)}}{C^{(p)}} \right\rceil, 1 + \left\lfloor \frac{\zeta_i^{(p)}}{C^{(p)}} \right\rfloor \right].$$

The left-hand (resp. right-hand) expression of the interval described above indicates a lower bound on the number of workstations necessary for allocating task i and all its predecessors (resp. successors). The central idea of that approach consists in considering the line balancing problem as a single machine scheduling one. To do this, for each task i and product p, two following parameters are introduced: earliest completion time $\theta_i^{(p)}$ and latest starting time $\zeta_i^{(p)}$. Based on the cycle time $C^{(p)}$ for each product p, these parameters can be computed using the following dynamic programming expressions:

$$\theta_i^{(p)} = \max\left\{ \alpha_i^{(p)}, \left(\left\lceil \frac{\alpha_i^{(p)} + t_i^{(p)}}{C^{(p)}} \right\rceil - 1 \right) \cdot C^{(p)} \right\} + t_i^{(p)},$$

$$\zeta_i^{(p)} = \min\left\{ \beta_i^{(p)}, \left(\left\lfloor \frac{\beta_i^{(p)} - t_i^{(p)}}{C^{(p)}} \right\rfloor + 1 \right) \cdot C^{(p)} \right\} - t_i^{(p)},$$

where $\alpha_i^{(p)} = \max\left\{ \sum_{j \in \mathcal{P}_i^{(p)}} t_j^{(p)}, \max_{j \in \overline{\mathcal{P}}_i^{(p)}} \theta_j^{(p)} \right\}$ and $\beta_i^{(p)} = \min\left\{ C^{(p)} \cdot |W| - \sum_{j \in \mathcal{P}_i^{(p)}} t_j^{(p)}, \min_{j \in \overline{S}_i^{(p)}} \zeta_j^{(p)} \right\}$. Here, $\theta_0^{(p)} = 0$ and $\zeta_{|V|+1}^{(p)} = C^{(p)} \times |W|$ represent respectively the dummy start and the dummy end of the schedule, $\overline{\mathcal{P}}_i^{(p)}$ (resp. $\overline{S}_i^{(p)}$) correspond to the set of direct predecessors (resp. successors) of task i for product p and finally $\mathcal{P}_i^{(p)}$ (resp. $S_i^{(p)}$) represents the set of all predecessors (resp. successors) of task i for product p.

Lemma 3.1. *The mono-objective function (1) has an equivalent bi-objective lexicographic representation, where the first objective is $r_{\max}$ and the second one is $\sum_{(p,q)\in U} r^{(p,q)}$ (both to be minimized).*

Proof (⇒). Let r, expressed by the pair $\left(r_{\max}, \sum_{(p,q)\in U} r^{(p,q)}\right)$, be an optimal solution for the problem studied with the mono-objective function (1), noted as F_1, and let us prove that r is also optimal for the aforementioned bi-objective lexicographic representation of (1), noted as F_2.

Assume that this is not the case and there exists a solution s, expressed respectively by the pair $\left(s_{\max}, \sum_{(p,q)\in U} s^{(p,q)}\right)$, which dominates r in the lexicographical sense. The latter means that one of the following two cases occurs: (*i*) $s_{\max} = r_{\max}$ and $\sum_{(p,q)\in U} s^{(p,q)} < \sum_{(p,q)\in U} r^{(p,q)}$ or (*ii*) $s_{\max} < r_{\max}$.

For the first case, we obviously have the following:

$$F_1(s) = s_{\max} + \frac{1}{|V| \cdot |U|} \cdot \sum_{(p,q)\in U} s^{(p,q)} < r_{\max} + \frac{1}{|V| \cdot |U|} \cdot \sum_{(p,q)\in U} r^{(p,q)} = F_1(r).$$

For the second case, since $0 \le s_{\max} < r_{\max} \le |V|$ and due to the definition of $s^{(p,q)}$ and $r^{(p,q)}$, we deduce the following two evident inequalities $\sum_{(p,q)\in U} s^{(p,q)} < |V| \cdot |U|$ and $0 < \sum_{(p,q)\in U} r^{(p,q)}$. Based on these observations, we obtain

$$F_1(s) = s_{\max} + \frac{1}{|V| \cdot |U|} \cdot \sum_{(p,q)\in U} s^{(p,q)} < s_{\max} + 1 \le r_{\max}$$

$$< r_{\max} + \frac{1}{|V| \cdot |U|} \cdot \sum_{(p,q)\in U} r^{(p,q)} = F_1(r).$$

For both cases, we derive that $F_1(s) < F_1(r)$, which is in contradiction with the optimality of r for F_1. As a consequence, r is also optimal for F_2.

(⇐) Now, let s be a lexicographically optimal solution for F_2. And suppose by contradiction that s is not optimal for F_1. Hence, there exists an optimal solution r for F_1 such that $F_1(r) < F_1(s)$. Moreover, since r is optimal for F_1, then it is necessarily optimal for F_2 (see the proof of ⇒). The latter leads us to conclude that $s_{\max} = r_{\max}$ and $\sum_{(p,q)\in U} s^{(p,q)} = \sum_{(p,q)\in U} r^{(p,q)}$, which is in contradiction with $F_1(r) < F_1(s)$. Thus, s is also optimal for F_1. □

4. Constructive heuristic

In the literature, many constructive heuristics have been developed to solve large size instances for SALBP [see, *e.g.*, 2]. Such approaches take advantage of task precedence relationships in order to generate priority rules for assigning tasks to workstations. They are known to provide good quality solutions for SALBP and are often used to generate a first feasible solution for an exact method [10]. To the best of our knowledge, no such heuristic has been developed in the literature regarding MuMALBP. To fill this gap, we propose in this paper a new one, inspired by the COMSOAL algorithm [29], usually used for SALBP.

The main idea of the proposed heuristic is to simultaneously construct line configurations for all the products based on their corresponding precedence graph. More precisely, given the number of workstations and the cycle time of each product, the algorithm uses the so-called candidate list $CL_k^{(p)}$, which contains all the tasks assignable to current workstation k of product p. This list is built in the following way. The set of unassigned tasks of product p is examined and task j is added to $CL_k^{(p)}$ if both of the following conditions are satisfied: (*i*) j has no predecessors or all predecessors of j are already assigned, (*ii*) assigning of j does not violate the cycle time $C^{(p)}$. Then, $CL_k^{(p)}$ is analyzed for each p. Namely, if there are common tasks in $CL_k^{(p)}$ for each product p, so one of these tasks is assigned to workstation k in all the configurations. Otherwise, for each product p, one task is randomly selected from $CL_k^{(p)}$ and assigned to current workstation k. For both cases described above, $CL_k^{(p)}$ is rebuilt for each p and their analysis is repeated. If $CL_k^{(p)}$ is empty for at least one product p, then the next empty workstation is opened for all product configurations. This new workstation becomes current for each product p and the algorithm continues to assign tasks to them. The heuristic stops either when all tasks for all products are assigned, or when it is no longer possible to open a new workstation. The first case indicates that a feasible solution is found, while the second one implies that an infeasible solution is reached. A random parameter is also added to the heuristic process in order to cover as many solutions as possible. A formal description of the constructive heuristic is given by Algorithm 1.

As Step 4 is subject to the random parameter α, the heuristic can therefore be repeated in order to explore and compare different feasible solutions. It is worth mentioning that if a feasible solution with no task reassignment is found, the heuristic stops since an optimal solution is reached. Similarly, if the heuristic does not find a better solution, compared to the current best one, after a limited number of iterations, it stops and the best solution is returned.

As we will see later, the heuristic described above provides good quality solutions when dealing with instances including 2 products. However, it reaches its limits when solving instances with 3 or more products. This can be explained by the fact that the set RCL_k (see Step 4 in Algorithm 1) is more likely to be empty when dealing with a larger number of products. One way to handle this situation is to consider common tasks not for all products, but for each possible combination of products $(p, q) \in U$. Such an approach is complex and difficult because it requires the development of specific rules to select and assign for each product an appropriate task.

Algorithm 1: Constructive heuristic

1. Open the first empty workstation, having no task assigned, for all product configurations, *i.e.*, set $k := 1$.
2. If there exists at least one unassigned task, then construct a candidate list $CL_k^{(p)}$ for each $p \in P$. Otherwise, go to Step 8.
3. If $CL_k^{(p)} = \emptyset$ for at least one $p \in P$ and $k < |W|$, then open the next empty workstation for all product configurations, *i.e.*, set $k := k + 1$, and go to Step 2. If $CL_k^{(p)} = \emptyset$ for at least one $p \in P$ and $k = |W|$, then go to Step 7.
4. Let $RCL_k := \bigcap_{p \in P} CL_k^{(p)}$ be a set of common assignable tasks for current workstation k between all product configurations. If $RCL_k \neq \emptyset$, then go to Step 5 with probability α or go to Step 6 with probability $1 - \alpha$. Otherwise, go to Step 6.
5. Choose randomly one task from RCL_k and assign it to workstation k for all product configurations and go to Step 2.
6. For each $p \in P$, choose randomly one task from $CL_k^{(p)}$ and assign it to workstation k of the configuration corresponding to product p and go to Step 2.
7. Stop, an inadmissible line configuration is reached.
8. Stop, an admissible line configuration is found.

5. MILP-based heuristic

In most cases, large size instances of line balancing problems are hard to solve to optimality [2]. One of the main difficulties regarding exact methods is the proof of optimality, which is a real challenge for commercial MIP solvers. Accordingly, to address the studied problem, a MILP-based heuristic, named as Halt-and-Fix, is proposed in this paper. The central idea of this heuristic consists in reducing the MILP search space by exploiting feasible solutions found by a solver during the resolution process [28]. In order to avoid poor quality solutions, a particular reduction mechanism is used.

Note that the Halt-and-Fix heuristic is different from the well-known Relax-and-Fix heuristic. The latter heuristic consists in solving a sequence of derived sub-problems with relaxed integrality constraints on selected variables. Then, some of them are fixed for the original problem. The Halt-and-Fix heuristic, which is presented below, is contrary based on a partial solving of the problem in order to obtain a feasible solution which is then used to fix the value of some variables for the next iteration.

The Halt-and-Fix is an iterative heuristic whose iteration consists in running the MIP solver for a short period of time T, analyzing the current best feasible solution found and adding specific constraints to fix some variables of the problem. Within this time period T, three cases may arise depending on the solving process. The first case occurs when the solver stops before reaching T, meaning that an optimal solution has been found. In this situation, the algorithm ends and returns this solution as the final one. The second case happens when the time limit T is reached by the solver, but no feasible solution better than the current best one was found. In that case, the solver continues until a new better solution is found or a global time limit dedicated to the heuristic is exceeded. For the third case, the solver is interrupted when a better solution is finally found either by reaching or exceeding T and the following additional constraints are generated based on it:

$$\sum_{k \in W} z_{ik}^{(p,q)} = 1, \quad \forall (p,q) \in U, \quad \forall i \in K(p,q). \tag{8}$$

Here, $K(p,q)$ represents all the tasks assigned to the same workstation for both configurations corresponding to products p and q in the current best solution found. Then, the heuristic repeats its iteration including for the MILP model both the aforementioned solution as a warm start one and a new set of constraints (8). The principal idea of (8) is to enforce each task belonging to $K(p,q)$ to be assigned to the same workstation (without indicating which one) for both configurations corresponding to products p and q for all further iterations of the heuristic. This allows to quickly reduce the MILP search space and, as a consequence, to speed up the solving process, especially since the set $K(p,q)$ is updated for any $(p,q) \in U$ at each iteration.

A more formal presentation of the Halt-and-Fix heuristic is described below by Algorithm 2.

To assess the performance of the Halt-and-Fix heuristic compared to both the constructive heuristic and the MILP model sole, computational experiments were conducted and a comparison between both methods is detailed in the next section of this paper.

6. Numerical experiments

In this section, a description of the used instances for the numerical experiments is first given. Then, computational results obtained for the MILP model (referred to as exact approach), and both heuristics are shown, compared, and analyzed.

6.1. Used instances and input data

For the numerical experiments, benchmark instances of [30] were used. Each of these latter consists of a precedence graph and the value of the processing time for each task. Among the available instances, only those having a directed acyclic weakly connected precedence graph were taken into account.

For these instances, the enumeration of precedence graph vertices follows the topological order. However, such graph structure might lead to trivial instances for the studied problem. Indeed, if for each product the task numbers follow a topological order in their precedence graph, then the configurations generated can be naturally similar. Thus, tasks with smaller numbers will always be assigned to the beginning of the line, and those with larger numbers are more likely to be assigned to the latest workstations. This is an interesting observation that reflects the reality of the field. However, it makes it difficult to assess the performance of the approaches developed, as shown in the tables below.

Thus, in order to properly evaluate the performance of the exact and heuristic approaches, numerical experiments were conducted on both the original and modified instances. These latter are more difficult to solve, as their precedence graph vertices are shuffled according to a randomly generated permutation. Fig. 3 shows an example of such a precedence graph modification for 2 instances, where the original and modified graphs are respectively displayed on the left-hand and right-hand sides, linked by a randomly generated permutation.

Numerical experiments were conducted on two sets of instances[1] of medium ($|V| = 50$) and large ($|V| = 100$) sizes. Each set was classified into three series based on the density of precedence graphs, also called the order strength (OS). This latter is computed as $OS = \frac{2 \cdot |E^{(p)}|}{|V| \cdot (|V|-1)}$, where $E^{(p)} = \left\{ (i,j) \mid i \in V, j \in S_i^{(p)} \right\}$ is the set of arcs in the transitive closure of $G^{(p)}$. As a result, the first series corresponds to the instances having an $OS \approx 0.2$. The second series includes instances with an $OS \approx 0.6$, and the last one is for those with an $OS \approx 0.9$. The number

[1] http://pagesperso.ls2n.fr/~gurevsky-e/data/RMS.zip

Algorithm 2: Halt-and-Fix heuristic

1. Set $K(p,q) := \emptyset$ for each $(p,q) \in U$. Set the best feasible solution $s^{(B)}$ as an empty one.
2. Start to solve the MILP model (1)–(8) with $s^{(B)}$ as a warm start solution. If an optimal solution is found within the time period T, then go to Step 4. Otherwise, if the time period T is expired and no optimal solution is found, then continue to solve and go to Step 3 only if a new feasible solution s better than $s^{(B)}$ is found.
3. Interrupt solving and based on the solution s, update the set $K(p,q)$ for each $(p,q) \in U$. Reset $s^{(B)} := s$ and repeat Step 2.
4. Stop, the final heuristic solution is found.

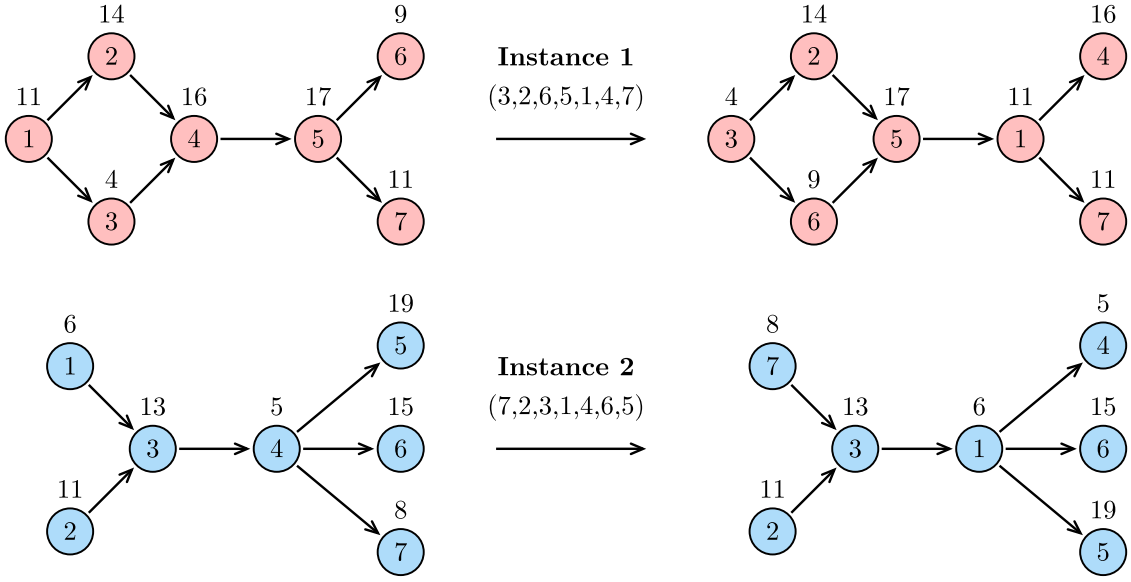

Fig. 3. Generation of two modified instances with $|V| = 7$.

of products is set to $|P| = 2$ and $|P| = 3$. Hence, for example, to construct an instance of the studied problem with two (resp. three) products, two (resp. three) successive instances from the same series have to be considered. Finally, the cycle time of each product is set to $C^{(p)} = \left\lceil 1.5 \cdot \max_{i \in V} t_i^{(p)} \right\rceil$, and the number of workstations is fixed to $|W| = \max_{p \in P} \left\{ \left\lceil 1.2 \cdot \sum_{i \in V} t_i^{(p)} / C^{(p)} \right\rceil \right\}$.

6.2. Computational results

This subsection presents, analyses and compares the obtained computational results of the exact approach and both heuristics. To solve the MILP formulation, CPLEX 20.1.0 was used for both approaches. In addition, the heuristics were coded using Julia 1.5.4 along with JuMP modeling language.

The experiments were conducted on a computer having a 3.7 GHz Intel(R) Core(TM) i3-6100 processor with 8 GB RAM. The global solving time limit was fixed to 600 s. For the Halt-and-Fix heuristic, the iteration time limit T was set to 10 s for each instance. For the constructive heuristic, the random parameter α was fixed to 0.01, and the maximum number of iterations without improving the solution was set to 500,000.

6.2.1. Exact approach results

The computational results for the MILP formulation sole are given in Tables 2 and 3. The former illustrates the results obtained for the modified instances, while the latter displays those obtained for the original ones. In both tables, the first three columns represent the number of products, the number of tasks, and the OS series number, respectively. The fourth column shows the total number of instances within each series. The number of instances solved to optimality and their average CPU time are respectively given in the fifth and sixth columns. As for the instances which were not solved to optimality within 600 s, their average GAP is displayed in the last column.

Table 2 shows that only 19.07% of all the instances were solved optimally. It can be seen that instances with $OS \approx 0.2$ are harder to solve than the other ones. This can be noticed through the following two factors: (1) an important number of instances unsolved to optimality and (2) their corresponding average GAP, which is relatively high. However, we can observe that for the other OS series (resp. 2 and 3) more instances were optimally solved. Moreover, we can also remark through the average GAP, that the higher the OS is, the easier the instances are to solve to optimality. Indeed, it is obvious that the average GAP and CPU decrease significantly as the OS increases. Nevertheless, despite this trend, it is clear that the number of instances unsolved to optimality and their corresponding average GAP remains relatively high, especially for large size instances corresponding to $|V| = 100$, where only a few instances were solved to optimality. In addition, for one instance, noted in the table by (+1), CPLEX was not able to find any feasible solution within 600 s. In view of the obtained results, it is clear that the MILP formulation is not efficient when dealing with medium and large-size instances.

In addition to these numerical experiments, Table 3 shows the results for the same instances but without applying random permutations to their precedence graphs. In this table, it is clear that medium-size instances (with $|P| = 2$ or $|V| = 50$) are easier to solve to optimality than the modified ones. However, in the case corresponding to $|P| = 3$ and $|V| = 100$, no optimal solution was found. Furthermore, the average GAP for these instances ($\approx$ 100%) is not a reliable measure and therefore provides no information about their complexity. This is mainly due to the fact that the lower bound is in most cases equal to 0 since all the tasks follow the topological order in the corresponding precedence graphs. This is not the case for Table 2 with modified instances.

Table 2
Computational results for the MILP model for modified instances.

$\|P\|$	$\|V\|$	OS	#Instances	#OPT	Avg. CPU, (s.)	Avg. GAP, (%)
2	50	1	15	2	181.3	53.46
		2	219	136	148.4	18.60
		3	74	74	16.4	⊕
	100	1	39	0	⊖	94.73
		2	219	0	⊖	52.94
		3	73(+1)	1	569.7	22.84
Global			640	212 (33.17%)	105.3	44.84
3	50	1	14	0	⊖	85.21
		2	218	0	⊖	42.64
		3	73	31	276.1	7.34
	100	1	38	0	⊖	97.11
		2	218	0	⊖	60.11
		3	72(+1)	0	⊖	29.30
Global			634	31 (4.88%)	276.1	49.30

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

Table 3
Computational results for the MILP model for unmodified instances.

$\|P\|$	$\|V\|$	OS	#Instances	#OPT	Avg. CPU, (s.)	Avg. GAP, (%)
2	50	1	15	15	4.0	⊕
		2	219	215	4.0	83.33
		3	74	74	8.6	⊕
	100	1	39	28	154.8	100
		2	218	186	142.6	100
		3	74	55	130.2	93.70
Global			640	573 (89.53%)	69.0	95.72
3	50	1	14	13	209.2	⊕
		2	218	158	65.1	83.79
		3	73	58	136.2	26.46
	100	1	38	0	⊖	100
		2	217	0	⊖	100
		3	73	0	⊖	99.65
Global			634	229 (36.11%)	91.28	94.31

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

To summarize, this exact approach seems to be efficient in finding promising feasible solutions but struggles to improve them or prove that they are optimal.

6.2.2. Heuristics results

This subsection provides a comparison between the numerical results obtained by the constructive and Halt-and-Fix heuristics. Similarly to the previous subsection, computational experiments were conducted by solving both unmodified (Table 4) and modified (Table 5) instances with $|P| = 2$ and $|V| = 50$, and $|P| = 2$ and $|V| = 100$.

In both tables, column 3 (resp. 4) provides the gap between the upper bound obtained by the constructive heuristic (resp. Halt-and-Fix heuristic) and the best upper bound found between these two approaches, denoted as $\texttt{GAP}^{\texttt{B}}_{\texttt{C}}$ (resp. $\texttt{GAP}^{\texttt{B}}_{\texttt{H}}$). More precisely, $\texttt{GAP}^{\texttt{B}}_{\texttt{C}} = 100\% \cdot (\texttt{UB}_{\texttt{C}} - \texttt{UB}_{\texttt{B}})/\texttt{UB}_{\texttt{B}}$, and $\texttt{GAP}^{\texttt{B}}_{\texttt{H}} = 100\% \cdot (\texttt{UB}_{\texttt{H}} - \texttt{UB}_{\texttt{B}})/\texttt{UB}_{\texttt{B}}$, where $\texttt{UB}_{\texttt{C}}$ (resp. $\texttt{UB}_{\texttt{H}}$) represents the upper bound found by the constructive heuristic (resp. Halt-and-Fix heuristic), and $\texttt{UB}_{\texttt{B}} = \min\{\texttt{UB}_{\texttt{C}}, \texttt{UB}_{\texttt{H}}\}$. Columns 5 and 6 provide the average CPU time over all instances for the constructive and Halt-and-Fix heuristics, denoted as $\texttt{CPU}_{\texttt{C}}$ and $\texttt{CPU}_{\texttt{H}}$, respectively. Finally, the last two columns show the average upper bound found by the constructive heuristic (resp. Halt-and-Fix heuristic), denoted as $\texttt{UB}_{\texttt{C}}$ (resp. $\texttt{UB}_{\texttt{H}}$).

From Table 4, it can be noticed that both heuristics provide good quality solutions in a reasonable CPU time. This can be seen through average $\texttt{UB}_{\texttt{C}}$ and $\texttt{UB}_{\texttt{H}}$, which are close to the theoretical lower bound of 0 for both sets of instances corresponding to $|V| = 50$ and $|V| = 100$. It can also be observed that the constructive heuristic is faster with a global average CPU time of 11.4 s versus 52.5 s for the Halt-and-Fix heuristic. This being said, based on $\texttt{GAP}^{\texttt{B}}_{\texttt{C}}$ and $\texttt{GAP}^{\texttt{B}}_{\texttt{H}}$, the Halt-and-Fix heuristic has found better solutions in most cases. This can be explained by the fact that the constructive heuristic struggles to improve the solution and gets stuck in local optima.

The same conclusions can be reached when solving the more challenging modified instances. Here, and based on Table 5, the Halt-and-Fix heuristic outperforms the constructive heuristic in terms of solution quality. In fact, the Halt-and-Fix heuristic was able to find the best solution in 99% of cases for both $|V| = 50$ and $|V| = 100$ with $\texttt{GAP}^{\texttt{B}}_{\texttt{H}} \approx 0\%$. The constructive heuristic is certainly faster, but finds solutions relatively far from the best one with a $\texttt{GAP}^{\texttt{C}}_{\texttt{H}}$, which varies between 15% and 80% on average.

Unlike the previous table where the difference between $\texttt{UB}_{\texttt{C}}$ and $\texttt{UB}_{\texttt{H}}$ is quite small, this one turns out to be more important when dealing with difficult instances. This shows that the Halt-and-Fix heuristic is more efficient as it allows to continuously improve the solution by adding new constraints at each iteration. Moreover, since the Halt-and-Fix is a MILP-based heuristic, it can be used to solve instances with more than 2 products. This is not the case for the constructive heuristic. For this reason, the constructive heuristic is not considered in the next parts of the experiments.

6.2.3. Comparison between the exact approach and the halt-and-fix heuristic

In this subsection, the results of the Halt-and-Fix heuristic are analyzed and compared with those obtained with the exact approach.

Table 4
Comparison between both heuristics for unmodified instances with $|P| = 2$.

$\|V\|$	OS	Avg. GAP^B_C, (%)	Avg. GAP^B_H, (%)	Avg. CPU_C, (s.)	Avg. CPU_H, (s.)	Avg. UB_C	Avg. UB_H
50	1	0.05	0.03	0.1	3.7	0.00	0.00
	2	44.37	5.07	6.3	4.7	0.52	0.05
	3	143.06	57.20	12.8	3.8	4.55	1.76
100	1	207.73	266.71	20.9	180.1	2.09	2.69
	2	305.01	0.09	22.2	95.8	3.08	0.00
	3	357.18	66.26	16.1	59.2	6.22	1.12
Global		185.59	31.73	11.4	52.5	2.55	0.48

Table 5
Comparison between both heuristics for modified instances with $|P| = 2$.

$\|V\|$	OS	Avg. GAP^B_C, (%)	Avg. GAP^B_H, (%)	Avg. CPU_C, (s.)	Avg. CPU_H, (s.)	Avg. UB_C	Avg. UB_H
50	1	80.18	0.09	51.7	25.7	20.20	11.56
	2	45.19	0.06	17.4	22.0	34.30	23.88
	3	20.63	0.04	8.38	8.44	43.00	35.72
100	1	73.61	0.78	26.6	280.4	65.21	41.41
	2	48.24	0.02	22.5	211.5	80.05	54.55
	3	15.43	0.01	47.4	250.9	89.68	78.47
Global		42.40	0.00	22.7	124.2	62.83	43.64

Table 6
Comparison between MILP and Halt-and-Fix heuristic for modified instances.

$\|P\|$	$\|V\|$	OS	Avg. GAP^B_E, (%)	Avg. GAP^B_H, (%)	Avg. CPU_E, (s.)	Avg. CPU_H, (s.)	Avg. UB_E, (s.)	Avg. UB_H, (s.)
2	50	1	5.00	3.43	600	25.7	11.69	11.56
		2	0.53	2.89	319.5	23.6	23.34	23.88
		3	0.00	0.73	15.2	8.44	35.47	35.72
	100	1	22.52	5.47	600	366.6	47.28	41.41
		2	21.77	0.08	600	221.7	66.51	54.55
		3	4.04	0.43	599.6	244.8	81.36	78.47
Global			9.58	1.56	436.3	136.16	48.24	43.64
3	50	1	17.07	0.00	600	77.3	23.72	20.44
		2	6.31	0.83	600	70.2	32.03	30.38
		3	0.24	3.56	462.4	16.4	39.18	40.46
	100	1	34.24	0.00	600	555.8	93.28	78.49
		2	27.12	0.03	600	431.9	85.31	68.62
		3	6.77	0.30	600	337.0	89.67	84.65
Global			14.73	0.74	584.1	248.3	60.07	52.73

Tables 6 and 7 show this comparison on the same modified and unmodified instances as those used in the previous subsection. In these tables, the fourth (resp. fifth) column represents the gap between the upper bound obtained with the exact approach (resp. the heuristic) and the best upper bound found between these two approaches. These gaps are computed as $GAP^B_E = 100\% \cdot (UB_E - UB_B)/UB_B$ for the exact approach and $GAP^B_H = 100\% \cdot (UB_H - UB_B)/UB_B$ for the heuristic, where UB_E (resp. UB_H) is the upper bound found by the exact (resp. heuristic) approach and $UB_B = \min\{UB_E, UB_H\}$. The sixth and seventh columns represent the average CPU time over all instances for the exact method (CPU_E) and the heuristic approach (CPU_H). Finally, the last two columns provide the average upper bound found by the exact method (UB_E) and the Halt-and-Fix heuristic (UB_H), respectively.

From Table 6, we can see that the heuristic provides better overall performance than the exact approach. Indeed, the average GAP, CPU time, and UB of the heuristic are significantly smaller than the exact approach. Moreover, for the instances corresponding to $|P| = 2$ and $|V| = 50$, for which optimal solutions were mainly found by the exact approach, the heuristic also manages to find good quality solutions, within a few seconds only. For the large instances of $|V| = 100$, most of the best solutions were found by the heuristic in a significantly small CPU time. Indeed, the heuristic succeeds to solve most instances before reaching the solving time limit of 600 s.

Table 7 provides the same comparison on unmodified instances. Unlike modified ones, the objective values of these instances are usually equal to 0, which causes very large and insignificant GAP values. To avoid this, the $GAP^B_{(\cdot)}$ values are calculated as follows $GAP^B_E = 100\% \cdot (UB_E - UB_B)/\max\{1, UB_B\}$ for the exact approach and $GAP^B_H = 100\% \cdot (UB_H - UB_B)/\max\{1, UB_B\}$ for the heuristic. As can be seen in this table, the heuristic is able to provide good-quality solutions that are close to optimal ones. For example, the heuristic was successful to find all optimal solutions for the instances corresponding to $|P| = 2$ and $|V| = 50$. When dealing with larger size instances of $|P| = 3$ and $|V| = 50$, the heuristic provides a better solution than the exact approach in 75% of the cases. As for the instances with 100 tasks, 96% and 74% of the best solutions were found by the Halt-and-Fix heuristic for $|P| = 2$ and $|P| = 3$, respectively. An exception is noticeable in the table regarding the instances corresponding to $|P| = 3$, $|V| = 50$ and $OS = 3$, where the exact approach has found better solutions and outperformed the heuristic. This can be explained by the fact that such instances are characterized by dense precedence graphs that reduce the number of possible feasible solutions and leads the heuristic to local optima.

For a better overview, Fig. 4 shows the obtained results for modified instances for both the exact approach (blue-colored round markers) and the Halt-and-Fix heuristic (red-colored square markers). In this figure, the top two graphs represent the average gap from the best found solution, denoted as Avg. $GAP^B_{(\cdot)}$ and obtained for the instances with $|V| = 50$ and $|V| = 100$ for each OS series. The two graphs below show the average CPU time comparison between the two approaches.

As it can be seen in these graphs, the heuristic performs better than the exact approach providing solutions with an average GAP^B_H up to 10% better for $|V| = 50$ and up to 25% better for $|V| = 100$. As for the average CPU time, it clearly visible that the proposed heuristic outperforms the exact approach for both instance sizes.

Table 7
Comparison between MILP and Halt-and-Fix heuristic for unmodified instances.

$\|P\|$	$\|V\|$	OS	Avg. GAP^B_E, (%)	Avg. GAP^B_H, (%)	Avg. CPU_E, (s.)	Avg. CPU_H, (s.)	Avg. UB_E, (s.)	Avg. UB_H, (s.)
2	50	1	0.00	0.00	4.0	3.7	0.00	0.00
		2	0.00	0.00	14.9	4.7	0.05	0.05
		3	0.00	0.00	8.6	3.8	1.76	1.76
	100	1	892.74	9.97	280.4	180.1	9.97	2.69
		2	354.19	0.00	211.5	95.8	3.54	0.00
		3	7.64	1.29	250.9	59.2	1.26	1.12
Global			175.92	1.96	124.3	52.5	1.74	0.48
3	50	1	29.04	0.00	237.1	57.5	0.29	0.00
		2	3.60	7.19	212.4	22.0	0.72	0.87
		3	0.00	34.79	231.5	11.4	4.32	6.01
	100	1	463.39	5.91	600.0	598.2	89.02	83.26
		2	2758.40	2.71	600.0	431.6	65.29	31.17
		3	383.13	40.15	600.0	222.9	29.41	12.64
Global			1017.88	12.34	415.3	219.3	28.98	14.42

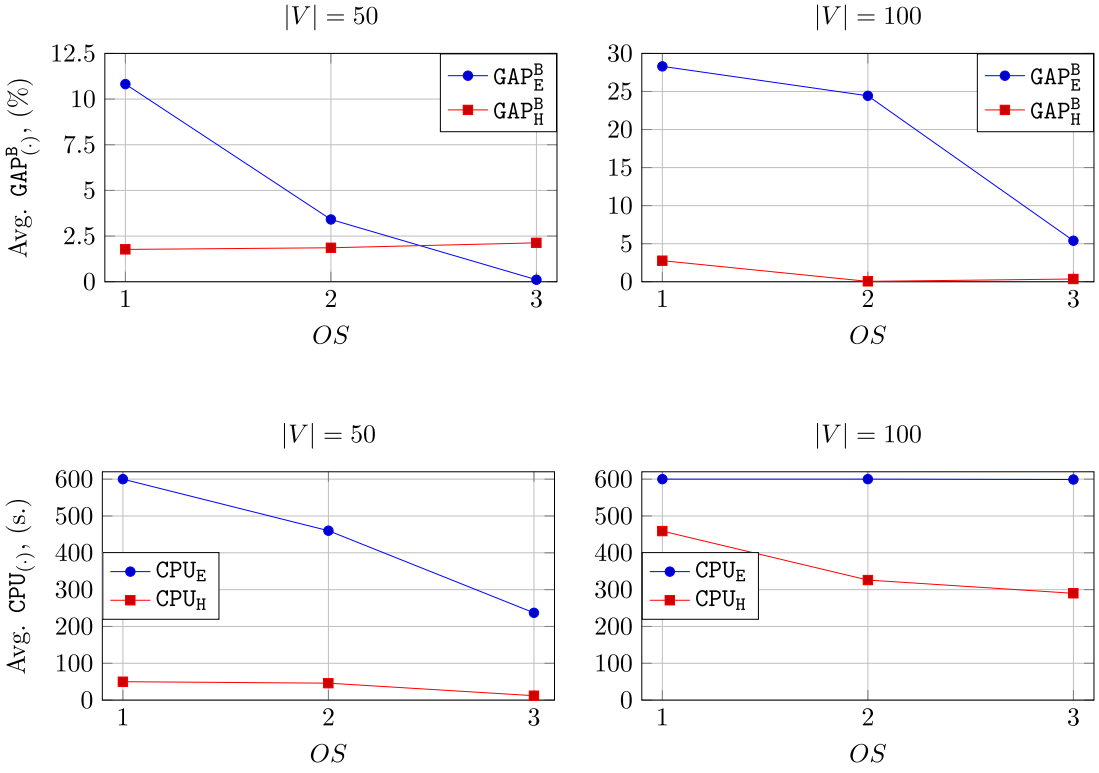

Fig. 4. Graph comparison between MILP and Halt-and-Fix heuristic for modified instances.

Even if the heuristic has shown better performance than the exact approach, it remains dependent on the MILP model. As a consequence, for some instances, it is difficult to find an initial or intermediate solution quickly within an iteration, which causes the heuristic to become more time-consuming.

7. Conclusion and perspectives

In this paper, a reconfigurable multi-product line balancing problem was studied. Under the assumption that the sequence of product arrival is not known in advance, the addressed optimization problem was focused on finding an appropriate line configuration for each product so as to minimize the maximum number of task reassignments between any two line configurations.

To tackle this problem, a MILP formulation was first proposed enhanced by some new pre-processing techniques for computing task assignment intervals. Then, two heuristics were developed. The first one is a constructive heuristic, which uses the precedence graph of each product to generate feasible line configurations. The second one, named Halt-and-Fix, consists in generating new constraints for the original MILP formulation based on a current feasible solution. The first numerical comparison between the two developed heuristics showed that the Halt-and-Fix outperforms the constructive heuristic in terms of solution quality. Moreover, since the Halt-and-Fix is a MILP-based heuristic, it makes it possible to handle instances with more than 2 products, whereas the constructive heuristic is difficult to adapt when the set of products increases. Then, the second comparison between the Halt-and-Fix and exact approaches was given. It was shown that the heuristic has been more efficient and managed to find solutions of better quality in a shorter CPU time than the exact approach.

Several possible extensions can be explored for future research. One of the most promising is based on taking into account available historical data about the frequency of product arrival and switching from one product to another. This may be achieved through a finer stochastic MILP model, which better reflects reality than the deterministic one. Moreover, we are also looking forward to studying the

effect of task reassignments in an integrated equipment selection for the MuMALBP problem. The objective will be to generate advanced line configurations while considering additional technological constraints, imposed by available resources [31]. Finally, the constructive heuristic, proposed in this paper, can be used to provide a warm start feasible solution for the Halt-and-Fix method, in order to improve its performance.

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).

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

References

[1] Bortolini M, Galizia FG, Mora C. Reconfigurable manufacturing systems: Literature review and research trend. J Manuf Syst 2018;49:93–106.
[2] Scholl A. Balancing and sequencing of assembly lines. 2nd ed.. Physica-Verlag Heidelberg; 1999.
[3] Becker C, Scholl A. A survey on problems and methods in generalized assembly line balancing. European J Oper Res 2006;168(3):694–715.
[4] Yelles-Chaouche AR, Gurevsky E, Brahimi N, Dolgui A. Reconfigurable manufacturing systems from an optimisation perspective: a focused review of literature. Int J Prod Res 2021;59(21):6400–18.
[5] Battaïa O, Dolgui A. Hybridizations in line balancing problems: A comprehensive review on new trends and formulations. Int J Prod Econ 2022;250:108673.
[6] Koren Y, Heisel U, Jovane F, Moriwaki T, Pritschow G, Ulsoy G, Van Brussel H. Reconfigurable manufacturing systems. CIRP Ann 1999;48(2):527–40.
[7] Koren Y, Shpitalni M. Design of reconfigurable manufacturing systems. J Manuf Syst 2010;29(4):130–41.
[8] Sivasankaran P, Shahabudeen P. Literature review of assembly line balancing problems. Int J Adv Manuf Technol 2014;73(9–12):1665–94.
[9] Boysen N, Fliedner M, Scholl A. A classification of assembly line balancing problems. European J Oper Res 2007;183(2):674–93.
[10] Battaïa O, Dolgui A. A taxonomy of line balancing problems and their solution approaches. Int J Prod Econ 2013;142(2):259–77.
[11] van Zante-de Fokkert JI, de Kok TG. The mixed and multi model line balancing problem: a comparison. European J Oper Res 1997;100(3):399–412.
[12] Boysen N, Fliedner M, Scholl A. Assembly line balancing: Joint precedence graphs under high product variety. IIE Trans 2009;41(3):183–93.
[13] Buxey GM, Slack ND, Wild R. Production flow line system design - A review. AIIE Trans 1973;5(1):37–48.
[14] Chakravarty AK, Shtub A. Balancing mixed model lines with in-process inventories. Manage Sci 1985;31(9):1161–74.
[15] Kabir MA, Tabucanon MT. Batch-model assembly line balancing: A multiattribute decision making approach. Int J Prod Econ 1995;41(1–3):193–201.
[16] Kimms A. Minimal investment budgets for flow line configuration. IIE Trans 2000;32(4):287–98.
[17] Lapierre SD, Debargis L, Soumis F. Balancing printed circuit board assembly line systems. Int J Prod Res 2000;38(16):3899–911.
[18] Pastor R, Andrés C, Duran A, Pérez M. Tabu search algorithms for an industrial multi-product and multi-objective assembly line balancing problem, with reduction of the task dispersion. J Oper Res Soc 2002;53(12):1317–23.
[19] Yu H, Shi W. A genetic algorithm for multi-model assembly line balancing problem. In: IEEE International Symposium on Assembly and Manufacturing (ISAM 2013). p. 369–71.
[20] Liao L. Construction and comparison of multi-model and mixed-model assembly lines balancing problems with bi-objective. J Ind Prod Eng 2014;31(8):483–90.
[21] Qu S, Jiang Z. A memetic algorithm approach for batch-model assembly line balancing problem of sub-block in shipbuilding. Proc Inst Mech Eng B 2014;228(10):1290–304.
[22] Christensen MK, Janardhanan MN, Nielsen P. Heuristics for solving a multi-model robotic assembly line balancing problem. Prod Manuf Res 2017;5:410–24.
[23] Pereira J. Modelling and solving a cost-oriented resource-constrained multi-model assembly line balancing problem. Int J Prod Res 2018;56(11):3994–4016.
[24] Asl AJ, Solimanpur M, Shankar R. Multi-objective multi-model assembly line balancing problem: a quantitative study in engine manufacturing industry. Opsearch 2019;56(3):603–27.
[25] Chen JC, Chen Y, Chen T, Kuo Y. Applying two-phase adaptive genetic algorithm to solve multi-model assembly line balancing problems in TFT–LCD module process. J Manuf Syst 2019;52:86–99.
[26] Koskinen J, Raduly-Baka C, Johnsson M, Nevalainen OS. Rolling horizon production scheduling of multi-model PCBs for several assembly lines. Int J Prod Res 2020;58(4):1052–73.
[27] Fisel J, Exner Y, Stricker N, Lanza G. Changeability and flexibility of assembly line balancing as a multi-objective optimization problem. J Manuf Syst 2019;53:150–8.
[28] Yelles-Chaouche AR, Gurevsky E, Brahimi N, Dolgui A. Minimizing task reassignments under balancing multi-product reconfigurable manufacturing lines. Comput Ind Eng 2022;173:108660.
[29] Arcus AL. A computer method of sequencing operations for assembly lines. Int J Prod Res 1965;4(4):259–77.
[30] Otto A, Otto C, Scholl A. Systematic data generation and test design for solution algorithms on the example of SALBPGen for assembly line balancing. European J Oper Res 2013;228(1):33–45.
[31] Battaïa O, Dolgui A, Guschinsky N, Levin G. Integrated configurable equipment selection and line balancing for mass production with serial–parallel machining systems. Eng Optim 2014;46(10):1369–88.