European Journal of Operational Research 313 (2024) 1015–1030

Contents lists available at ScienceDirect

European Journal of Operational Research

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

Production, Manufacturing, Transportation and Logistics

A new upper bound based on Dantzig-Wolfe decomposition to maximize the stability radius of a simple assembly line under uncertainty

Rui S. Shibasaki[a,*], André Rossi[b], Evgeny Gurevsky[c]

[a] MIS, Université de Picardie Jules Verne, Amiens, France
[b] LAMSADE, Université Paris-Dauphine, PSL, France
[c] LS2N, Université de Nantes, France

ARTICLE INFO

Article history:
Received 22 February 2022
Accepted 24 August 2023

Keywords:
Manufacturing
Assembly line balancing
Robustness
Stability radius
Dantzig-Wolfe decomposition

ABSTRACT

This work presents a new upper bounding approach, based on Dantzig-Wolfe decomposition and column generation, for a relatively novel problem of designing simple assembly lines to maximize their stability radius under uncertainty in task processing times. The problem considers task precedence constraints, a fixed cycle time, and a fixed number of workstations. This NP-hard optimization problem aims to assign a given set of assembly tasks to workstations in order to find the most robust feasible line configuration. The robustness of the configuration is measured by the stability radius with respect to its feasibility, *i.e.*, the maximum increase in task processing times, for which the cycle time constraint remains satisfied. The reformulation resulting from the Dantzig-Wolfe decomposition is enhanced with valid inequalities and tight assignment intervals are used to reduce the solution space of pricing sub-problems. In addition, a bisection method is proposed as a pre-processing technique to improve the initial upper bound on the stability radius, which is an input for the pricing sub-problem. Computational experiments show that the proposed approach can significantly improve the upper bound on the stability radius for the most challenging instances.

1. Introduction

Simple assembly lines are manufacturing systems consisting of sequentially ordered workstations connected by a transport mechanism (or conveyor belt). These lines are usually designed for large-scale production of a single type of product. The workstations operate simultaneously and process different tasks sequentially. All tasks assigned to a workstation must be completed within a certain time C, called *cycle time*, in order to maintain the production rate. No buffer stocks are allowed. In other words, every C units of time, an item of raw product is fed into the first workstation and then transferred from one workstation to another at the same rate. In this way, one item of finished product is obtained at the end of the line every C time units (see Scholl, 1999). Note that a workstation can only process one product item during a cycle time, and each product item can only be processed by one workstation at a time.

In addition, any task has to be fully processed by a single workstation, and precedence constraints should be satisfied. For example, if task i is supposed to be completed before starting task j for some manufacturing reason, then task j can only be assigned to the same workstation as i, or to a downstream workstation. Equivalently, this precedence constraint forbids the assignment of task j to a workstation upstream of i. Such precedence constraints are often modeled by a directed acyclic graph whose nodes represent tasks, and arcs such as (i, j) indicate that task i needs to be completed before task j starts.

In the literature, two optimization problems, called *balancing problems*, are generally studied for simple assembly lines (see Battaïa & Dolgui, 2013; Scholl & Becker, 2006). The first one, denoted as SALBP-1, aims at assigning a given set of assembly tasks to a minimal number of workstations while satisfying the precedence and cycle time constraints. The second problem, referred to as SALBP-2, seeks to minimize the cycle time for a fixed number of workstations. For other related problems, we refer the reader to the surveys: Battaïa & Dolgui (2013), Scholl & Becker (2006), and most recently Boysen, Schulze, & Scholl (2022).

[*] Corresponding author.
E-mail address: rui.sa.shibasaki@u-picardie.fr (R.S. Shibasaki).

https://doi.org/10.1016/j.ejor.2023.08.046
0377-2217/© 2023 Elsevier B.V. All rights reserved.

The balancing problem is related to the early stages of the assembly line design process. Therefore, task processing times are often based on estimates or nominal values, as mentioned in Battaïa & Dolgui (2013). These initial values can change during line operation, and deviation from them can deteriorate assembly line throughput if the cycle time is overrun. This can result in significant economic losses. Among the main factors that can lead to deviations in task processing times are: productivity, skill level, operators' fatigue and motivation; changes in product specifications and in workstation characteristics; and general delays in task execution. In this sense, it is desirable to achieve a task assignment to workstations that protects the corresponding line configuration and its throughput as much as possible from the uncertainty of task processing times. In this paper, in order to measure the aforementioned uncertainty, we use the concept of *stability radius* and address the balancing problem of a simple assembly line with a fixed cycle time and a fixed number of workstations.

The stability radius in the field of assembly line balancing was first introduced by Sotskov, Dolgui, Sotskova, & Werner (2005) and Sotskov, Dolgui, & Portmann (2006) for SALBP-2 and SALBP-1, respectively. In these papers, the authors assume that an optimal line configuration is already known and calculate its stability radius as the largest magnitude of deviation of the processing times from their nominal values, for which the optimality of the configuration is preserved. They also show that the stability radius can serve as an appropriate robustness measure. However, Sotskov et al. (2006, 2005) pointed out that computing the stability radius with respect to configuration optimality is a very difficult problem, which was confirmed in their later works (Lai, Sotskov, & Dolgui, 2019; Lai, Sotskov, Dolgui, & Zatsiupa, 2016; Sotskov, Dolgui, Lai, & Zatsiupa, 2015), where some efficient full enumeration procedures were developed. Fortunately, in Sotskov et al. (2006), it was proven that computing the stability radius with respect to configuration feasibility only is a polynomially solvable problem for any feasible configuration of SALBP-1. These positive outcomes have inspired several other studies. Thus, in Gurevsky, Battaïa, & Dolgui (2012), the results obtained for SALBP-1 and SALBP-2 were extended to the SALBP-E problem, whose objective is to minimize the product of the cycle time and the number of workstations used. Then, Gurevsky, Battaïa, & Dolgui (2013a) adapted the results of SALBP-1, obtained in Sotskov et al. (2006), for the transfer line balancing problem aiming to minimize the equipment cost, where tasks are executed simultaneously by blocks, activated sequentially in the workstations. Finally, Schepler, Rossi, Gurevsky, & Dolgui (2022) proposed a branch-and-price method for the bin-packing problem with items of uncertain size. In this last paper, the authors seek a solution with the smallest number of bins, such that the bins with at least one uncertain item have a predefined level of local stability radius.

Instead of computing the stability radius for a given line configuration, a completely new idea was proposed in Rossi, Gurevsky, Battaïa, & Dolgui (2016) for simple assembly lines, and later in Pirogov, Gurevsky, Rossi, & Dolgui (2021) for transfer lines. Considering a simple assembly line with a fixed cycle time and a fixed number of workstations, the authors focused on seeking a line configuration with the greatest stability radius with respect to its feasibility. Named SALBP-S, this problem was proven to be NP-hard in the strong sense in Rossi et al. (2016), where the authors proposed a *compact* mixed-integer linear programming (MILP) formulation to handle it, *i.e.*, a formulation with a polynomial number of constraints and variables. Based on some combinatorial aspects of SALBP-S and aiming to solve it efficiently, Rossi et al. (2016) also provided tight upper bounds for small-size instances, but optimality gaps remained significant for the larger ones.

To further reduce these gaps, the present paper proposes a new upper-bounding procedure, based on the Dantzig-Wolfe decomposition for SALBP-S. The MILP model, described in Rossi et al. (2016), is reformulated and strong valid inequalities are introduced to reinforce it. New upper bounds are obtained while solving the model's linear relaxation by column generation. A particularity is that the corresponding pricing sub-problem depends on an existing upper bound. Moreover, solving such a sub-problem can be too time-consuming. These characteristics motivated the use of preprocessing techniques for computing tight task assignment intervals (Section 3) and efficient initial upper bounds (Section 4). As far as the latter is concerned, we propose a bisection method based on *destructive improvement* bounds (Klein & Scholl, 1988; Pereira, 2015; Scholl & Becker, 2006).

The assignment intervals are computed considering the solution of a single-machine scheduling problem, as in Scholl & Becker (2006). Usually referred to as LB4, this strategy is also applied to compute lower bounds for SALBP-1 (Scholl, 1999; Scholl & Becker, 2006). Indeed, Pereira (2015) provided an empirical evaluation of lower bounds for the SALBP-1 and concluded that considering precedence relations in the calculation of bounds, as in LB4, is worthwhile. Likewise, such a strategy can produce tighter assignment intervals than the method presented by Patterson & Albracht (1975), also applied in the literature (see, for example, Morrison, Sewell, & Jacobson, 2014; Peeters & Degraeve, 2006). According to the results, the availability of tighter intervals has a positive impact on the performance of the column generation algorithm.

There are two main reasons for focusing on developing algorithms that return an upper bound on the stability radius. On the one hand, finding good upper bounds informs the decision maker on what can be expected in terms of robustness for a particular problem instance. More generally, it is common to search for lower bounds in the case of minimization problems, as in Arkhipov, Battaïa, & Lazarev (2019). On the other hand, the MILP formulation of SALBP-S requires the availability of an upper bound on the objective function value (see constraint (3) in Section 5). The *big-M* form of this constraint naturally stresses the need for the tightest possible upper bound to achieve good performances. To the best of our knowledge, Rossi et al. (2016) is the only work presenting a method to compute upper bounds for the SALBP-S, and numerical experiments show that the new method proposed in Section 4 can improve these bounds at a low computational effort.

A few other works applying Dantzig-Wolfe decomposition can be found in the literature, but for related problems. In the present case, the decomposition allowed the identification of specific valid inequalities that significantly improve the upper bound quality. Peeters & Degraeve (2006) study a Dantzig-Wolfe decomposition for the SALBP-1, where the relaxation of the precedence constraints is proposed. Following the same idea, Bautista & Pereira (2011) adapted the former approach for the case of SALBP-1 with time and space restrictions. In these works, the authors suggested relaxing the precedence constraints in order to reduce the size of the master problem, enhancing performances in terms of computational time. Here, we consider a different type of precedence constraints, which according to Ritt & Costa (2018) dominate the ones used in the aforementioned papers. We present a comparison between different versions of the Dantzig-Wolfe decomposition approach to check if previous strategies proposed in the literature are valid for addressing SALBP-S even with the stronger precedence constraints.

The numerical experiments were conducted on benchmark instances of Otto, Otto, & Scholl (2013), with 100 tasks. For the very large instances of 1000 tasks, the linear programming-based approaches tested in this work cannot be applied because of the very large size of the linear programs, indicating that a different strategy may be adopted in those cases. Nevertheless, the bisection method is still able to improve upper bounds for 1000-task instances in a few seconds of computational time.

Other approaches dealing with the uncertainty of task processing times can also be found in the literature. However, they strongly depend on the availability of extra information related to the uncertainty, which contrasts with the approach considered here. Among the most popular methods, we can distinguish a stochastic method, known as *chance-constrained* method, which considers task processing times as independent random variables with known probability distributions (see, *e.g.*, Ağpak & Gökçen, 2007; Özcan, 2018). It consists in assigning tasks in such a way that, for each workstation, the probability of satisfying the cycle time is greater than a given value, called confidence level. In the later-mentioned papers, the authors use a MILP formulation of the corresponding problem that incorporates the probabilistic cycle time constraint. From the information expressing the task processing times, they introduce new additional variables using various linearization techniques in order to obtain again a MILP formulation, this time deterministic and equivalent to the probabilistic problem studied. *Fuzzy approaches* have also been applied, viewing the task processing time values as fuzzy sets, with a known membership function. As in the stochastic case, to assign tasks to workstations, it is necessary to control the cycle time constraint. To do so, an adapted fuzzy arithmetic and an appropriate method dedicated to the comparison of fuzzy sets have to be introduced. Applications of tasks with fuzzy times are presented in Tsujimura, Gen, & Kubota (1995) and Gen, Tsujimura, & Li (1996) for SALBP-1, in Hop (2006) for a mixed-model line balancing, and in Zacharia & Nearchou (2013), for a bi-objective variant of SALBP-2. Finally, *robust approaches* can also be found in the literature. This last method assumes that a set of possible scenarios are known *a priori* and seeks a solution that remains feasible for all scenarios while offering the best possible quality in the worst case. Following this strategy, Gurevsky, Hazır, Battaïa, & Dolgui (2013b) and Pereira & Álvarez-Miranda (2018) studied the SALBP-1 with intervals of task processing times. Both papers focus on *Bertsimas and Sim's robustness*, which assumes that at most Γ tasks can take the largest processing time of their intervals at the same time, and the remaining tasks can be considered with the smallest value of their intervals. For this robustness criterion, both papers developed branch-and-bound procedures with different branching strategies and lower-bound processing techniques. The same robustness approach was studied in Hazır & Dolgui (2013), but for the SALBP-2 problem, where the authors developed an exact solution procedure similar to Benders decomposition. Finally, a *min-max robustness* approach was applied in Dolgui & Kovalev (2012) for SALBP-2, but with a discrete set of scenarios. In the latter work, the computational complexity of seeking a robust solution was presented for different types of precedence constraints.

The next section introduces basic definitions, properties, and notations. In the following, Sections 3 and 4 outline the preprocessing techniques, *i.e.*, the computation of assignment intervals, and the bisection method, respectively. The compact MILP formulation of Rossi et al. (2016) is revisited in Section 5, followed by the Dantzig-Wolfe decomposition and the column generation approach, both presented in Section 6. Section 7 provides computational experiments, where we also discuss managerial insights and how to obtain heuristic solutions using the generated columns. Finally, a conclusion is made in Section 8.

2. Basic definitions and properties

We now define notations used in further statements. Let $V = \{1, 2, \dots, n\}$ be the set of required assembly tasks and $W = \{1, 2, \dots, m\}$ be the set of available workstations. It is assumed that two sets of *uncertain tasks* exist: a set $\widetilde{V}^1 \subseteq V$ of *a priori uncertain tasks*, whose processing time may deviate from its nominal value without any additional information, and a set $\widetilde{V}^2 \subseteq V$ of *a posteriori uncertain tasks* whose uncertainty is caused by a set $\widehat{W} \subseteq W$ of *uncertain workstations*. These workstations are such that any task allocated to them becomes uncertain (even if it belongs to $V \setminus \widetilde{V}^1$). Hereinafter, the set of all uncertain tasks is denoted as $\widetilde{V} = \widetilde{V}^1 \cup \widetilde{V}^2$, and any workstation from $W \setminus \widehat{W}$ is called *certain*. The presence of certain and/or uncertain workstations can be explained by the existence of assembly lines having simultaneously two types of workstations: the ones with automatic tasks, executed by robots or machines, and workstations where tasks are operated manually. Supplementary notations related to the studied problem are given in Table 1.

Table 1
Supplementary notations.

G	is a directed acyclic graph (V, A) representing the precedence constraints, where A is the set of arcs;
t_j	is a non-negative nominal processing time of the task j;
t	is a vector expressing the nominal task processing times, *i.e.*, $(t_1, t_2, \dots, t_n)$;
$F(t)$	is the set of all feasible solutions with respect to a given vector t;
Ξ	is the set of vectors, where each of which presents possible non-negative processing time deviations for the uncertain tasks, *i.e.*, $\{\xi \in \mathbb{R}^n_{\geq 0} \mid \xi_j = 0,\ j \in V \setminus \widetilde{V}\}$;
C	is the cycle time;
V_k	is the set of all tasks that can potentially be assigned to workstation k;
$\widetilde{V}_k$	is the set of all uncertain tasks that can potentially be assigned to workstation k, *i.e.*, $\widetilde{V}_k = V_k \cap \widetilde{V}^1$, if $k \in W \setminus \widehat{W}$ and $\widetilde{V}_k = V_k$ otherwise;
$\overline{P}_j$	is the set of all direct predecessors of j in G;
P_j	is the set of all direct and indirect predecessors of j in G;
$\overline{S}_j$	is the set of all direct successors of j in G;
S_j	is the set of all direct and indirect successors of j in G;
Q_j	is the interval $[l_j, u_j]$ of workstations that can process the task $j \in V$.

The stability radius of a feasible solution $s \in F(t)$ can be defined as follows (see Sotskov et al., 2006):

$$\rho(s,t) = \max\left\{\epsilon \geq 0 | \forall \xi \in B(\epsilon) \quad (s \in F(t+\xi))\right\},$$

where $B(\epsilon) = \{\xi \in \Xi | \|\xi\| \leq \epsilon\}$. In other words, $\rho(s,t)$ is defined as the value of the radius of the greatest closed ball $B(\cdot)$, called *stability ball*.

Any element ξ of $B(\cdot)$ is evaluated based on a given norm $\|\cdot\|$. In Rossi et al. (2016), two norms l_1 ($\|\cdot\|_1$) and l_∞ ($\|\cdot\|_\infty$) were used for the stability radius, the latter being the one considered in this work. By definition, $\|\xi\|_1 = \sum_{j \in \widetilde{V}} \xi_j$ and $\|\xi\|_\infty = \max_{j \in \widetilde{V}} \xi_j$. The notation ρ_∞ denotes the stability radius in the l_∞ norm. According to Theorem 2 of Rossi et al. (2016), ρ_∞ is calculated as the minimum value, over all the uncertain workstations, of the idle time divided by the number of assigned uncertain tasks. It can be easily computed in $O(n)$ time. Moreover, the following useful property is a direct corollary from the definition of the stability radius.

Property 1. *For any solution $s \in F(t)$ and $\xi \in \Xi$ such that $\xi_j = \rho_\infty(s,t)$, $j \in \widetilde{V}$, we have $s \in F(t+\xi)$.*

This comes from the fact that the stability radius in the l_∞ norm represents the largest increase that can occur in the processing time for all uncertain tasks simultaneously, without losing the feasibility of the admissible solution being studied. In that sense, if those processing times are artificially increased by the optimal ρ_∞ value, an optimal solution will still be found. The discussion in the next two sections is essentially based on such a property. We remind that the present work addresses the problem of maximizing the stability radius in terms of solution feasibility. Since a decrease in task processing times cannot compromise solution feasibility, only positive task processing time deviations are considered.

The following two properties show that adding an extra workstation does not necessarily increase the stability radius, but extending the cycle time does. These properties are used in

Section 7.6.2 to help the assembly line manager explore options to improve the stability radius value in a practical context.

Let s be an optimal solution to a problem instance of SALBP-S with m workstations and a cycle time equal to C.

Property 2. *If $\rho_\infty(s,t)$ is set by a workstation that contains a unique task that is uncertain, then adding certain or uncertain workstations does not increase the value of the stability radius. Indeed, the stability radius value is bounded from above by $C - t_j$ where task j is uncertain, and this upper bound is reached in s. Hence, adding extra workstations does not change this upper bound on the stability radius.*

Property 3. *If $\widetilde{V}$ is nonempty, increasing the cycle time by a strictly positive amount δ always yields a strictly positive increase of the optimal stability radius value. Indeed, by Theorem 2 of Rossi et al. (2016), $\rho_\infty(s,t) = \min_{k\in W} \frac{1}{|\widetilde{V}_k|}(C - \sum_{j\in V_k} t_j)$. Increasing C does not compromise the feasibility of s, and for each workstation k such that $\widetilde{V}_k$ is nonempty, $\frac{1}{|\widetilde{V}_k|}(C + \delta - \sum_{j\in V_k} t_j) > \frac{1}{|\widetilde{V}_k|}(C - \sum_{j\in V_k} t_j)$ holds. Since $\widetilde{V}$ is nonempty, the minimum of these terms is finite and increases strictly when δ is strictly positive, so the stability radius value also increases strictly.*

3. Assignment intervals

We now discuss the pre-processing technique for defining tasks' assignment intervals. Such intervals are computed based on representing the studied problem as a single-machine scheduling problem, as proposed in Scholl (1999) and Scholl & Becker (2006). Furthermore, we consider the addition of new precedence relations as proposed by Fleszar & Hindi (2003) for SALBP-1.

According to Scholl (1999), it is possible to associate each task $i \in V$ with its earliest starting time τ_i and its latest completion time ζ_i. The method computes a lower bound on τ_i and an upper bound on ζ_i, considering the m parallel workstations as a single machine that can process tasks from time 0 to time $m \cdot C$. The calculation of τ_i requires solving the scheduling problem $1|r_j|C_{\max}$ for P_i, the set of all direct and indirect predecessors of i. The release date r_j for each task $j \in P_i$ is set to its earliest starting time. The problem $1|r_j|C_{\max}$ is polynomially solvable in $O(n \log n)$ (see Brucker, 2007). For a set of tasks denoted by J, we first set $C_{\max}$ to zero, and then iteratively update $C_{\max} = \max\{C_{\max}, r_j\} + t_j$ for each $j \in J$, in non-decreasing order of r_j.

Once τ_i is computed, we consider two well-known rounding strategies to improve τ_i (see Scholl & Becker, 2006). The first verifies if $\tau_i > \left\lfloor \frac{\tau_i}{C} \right\rfloor \cdot C$. In this case, it implies that $l_i = 1 + \left\lfloor \frac{\tau_i}{C} \right\rfloor$. Indeed, if there exists a solution where no predecessor of i is executed on workstation l_i, then task i can start at time $\left\lfloor \frac{\tau_i}{C} \right\rfloor \cdot C$ on workstation l_i, which contradicts the hypothesis $\tau_i > \left\lfloor \frac{\tau_i}{C} \right\rfloor \cdot C$. Hence, at least one predecessor $j \in \overline{P}_i$ is allocated to workstation l_i, and task i cannot start before time $\left\lfloor \frac{\tau_i}{C} \right\rfloor \cdot C + t_j$. Consequently, $\tau_i = \max\{\tau_i, \left\lfloor \frac{\tau_i}{C} \right\rfloor \cdot C + \min_{j\in \overline{P}_i} t_j\}$.

The second rounding strategy is presented as follows. Let us define $q = \left\lfloor \frac{\tau_i}{C} \right\rfloor$, so that a task i starting at time τ_i is processed by workstation $q+1$. If $\left\lceil \frac{1}{C}(\tau_i + t_i) \right\rceil > q+1$, then task i cannot be allocated to workstation $q+1$, since it cannot span over two workstations. In that case, the earliest starting time τ_i is improved to $(q+1)\cdot C$. On the contrary, τ_i remains unchanged.

For all i in V, we compute ζ_i using the same process as for τ_i, but considering the tasks of V in the reverse topological order with respect to G. Namely, considering τ_i^r, a lower bound on the earliest starting time of task i in the reverse topological order, one may set $\zeta_i = m \cdot C - \tau_i^r$.

The bounds of assignment intervals can be tightened further by considering the following situation. As proposed by Fleszar & Hindi (2003), considering $i \in V$, if there exists $j \in V \backslash P_i$ such that $\tau_i + t_i > \zeta_j - t_j$, then task j, which is not a predecessor of i, must be completed before task i can start. Consequently, arc (j,i) can be added to A without modifying the solutions of SALBP-S. Such an addition of a precedence constraint implies that j is added to P_i, which can lead to an increase in τ_i when solving the scheduling problem.

Considering the propositions of Scholl (1999), Fleszar & Hindi (2003), and Scholl & Becker (2006), the overall approach to compute the assignment intervals is summarized in Algorithm 1. For all $i \in V$, the final intervals are defined as

Algorithm 1: Overall approach for reducing assignment intervals.

```
Input: Precedence graph $G = (V, A)$ and processing time $t_j$ for
       each task $j \in V$.
Output: Reduced assignment interval $Q_j = [l_j, u_j]$ for each task
        $j \in V$.

1. Based on the work of Patterson and Albracht (1975),
   initialize $\tau_i \leftarrow \left(\left\lceil \frac{t_i + \sum_{j\in P_i} t_j}{C} \right\rceil - 1\right) \cdot C$ and
   $\tau_i^r \leftarrow \left(\left\lceil \frac{t_i + \sum_{j\in S_i} t_j}{C} \right\rceil - 1\right) \cdot C$ for each $i \in V$. In what follows,
   let us consider the tasks of $V$ in the topological order with
   respect to $G$.
2. For each task $i \in V$, set $\tau_i \leftarrow \max\{\tau_i, f(i)\}$, where $f(i)$ is the
   optimal value of the scheduling problem $1|r_j|C_{\max}$ on the
   tasks of $P_i$. Then, try to improve $\tau_i$.
3. For each task $i \in V$, set $\tau_i^r \leftarrow \max\{\tau_i^r, f^r(i)\}$, where $f^r(i)$ is
   the optimal value of the scheduling problem $1|r_j|C_{\max}$ on
   the tasks of $S_i$. Then, try to improve $\tau_i^r$. Finally, compute
   $\zeta_i \leftarrow m \cdot C - \tau_i^r$ for each $i \in V$.
4. Try to identify additional precedence arcs, as shown above.
   If at least one arc has been added to $A$, repeat Step 2, other-
   wise go to Step 5.
5. For each $i \in V$, compute its assignment interval $Q_i$, where
   $l_i \leftarrow \left\lceil \frac{t_i + \tau_i}{C} \right\rceil$ and $u_i \leftarrow \left\lceil \frac{\zeta_i}{C} \right\rceil$.
```

$$l_i = \left\lceil \frac{t_i + \tau_i}{C} \right\rceil \quad \text{and} \quad u_i = \left\lceil \frac{\zeta_i}{C} \right\rceil.$$

Note that, because of Step 1, the obtained assignment intervals are at least as tight as those returned by Patterson & Albracht (1975). Alternatively, one can say that the method by Patterson & Albracht (1975) requires to solve $1||C_{\max}$, which is a relaxation of $1|r_j|C_{\max}$, thus producing weaker bounds on τ_i and ζ_i.

The running time of Step 1 is $O(n)$. In Step 2, the scheduling problem is solved once for each task, so the total running time is $O(n^2 \log n)$ (searching for improvements can be done in linear time). Step 3 has the same running time as Step 2. Step 4 runs in $O(n^2)$ and the execution time of Step 5 is in $O(n)$. Steps 2 to 4 can be repeated only if at least one precedence arc is added to G, and the maximum number of arcs is $|A| = O(n^2)$, consequently, the overall running time is in $O(n^4)$.

Finally, notice from Property 1 that if a lower bound LB_∞ on the optimal stability radius is available, one can set $(t_j + LB_\infty)$ as the processing time for any uncertain task $j \in \widetilde{V}^1$ while computing l_j and u_j, which may further reduce the assignment intervals. In the present work, an adaptation of the heuristic proposed by Pirogov et al. (2021) is used to obtain an initial lower bound.

4. Upper bound improvement

A second pre-processing procedure is proposed in this section. Indeed, an upper bound UB_∞ on the optimal stability radius is needed as input for the MILP formulation that will be presented in Section 5. Based on combinatorial arguments, an upper bound on ρ_∞ was proposed by Rossi et al. (2016), while here we propose a bisection method to improve it.

It is assumed that we have $LB_\infty \le \rho_\infty \le UB_\infty$, where LB_∞ is the best-known value for ρ_∞ (obtained, for example, by using dedicated heuristic methods as mentioned in Section 7.1), and UB_∞ is its best known upper bound, provided combinatorially as in Rossi et al. (2016). The main idea of the procedure is to temporarily increase the processing time of each uncertain task by a certain amount Δ and to check whether this makes the problem infeasible or not. This procedure is implemented as a bisection method, which sets the value Δ in the middle of the interval $[L, U]$, where L and U are initialized to LB_∞ and UB_∞, respectively. This procedure is also based on the fact that the presence of an empty assignment interval is a sufficient condition for infeasibility. More precisely, if there exists $j \in V$ such that $l_j > u_j$, then Q_j is empty, and task j cannot be allocated to any workstation. When such a situation occurs, it can be concluded that the current increase in processing times, Δ, is too large. Hence, if no empty assignment interval is found with the increased processing times, then L is updated to Δ, otherwise such an increase causes infeasibility and the upper bound U is updated to Δ.

The procedure is formally described in Algorithm 2. Each iteration of the bisection method consists of computing Δ to halve the interval $[L, U]$. After increasing the processing time of the tasks by Δ, the assignment intervals are updated and checked, verifying if one of them is empty. The bisection method stops when $U - L$ becomes smaller than a given strictly positive parameter ϵ.

Algorithm 2: Improvement of UB_∞.

```
1 Set $L \leftarrow LB_\infty$, $U \leftarrow UB_\infty$ and $\epsilon \leftarrow 10^{-3}$.
2 while $U - L > \epsilon$ do
3     Compute $\Delta \leftarrow \frac{1}{2}(L + U)$ and set $t_j \leftarrow t_j + \Delta$ for each task $j \in \widetilde{V}$.
4     For each task $j \in V$, calculate its assignment interval $Q_j$(cf. Section 3).
5     If $l_j \le u_j$ for all $j \in V$, then set $L \leftarrow \Delta$. Otherwise, set $U \leftarrow \Delta$.
6     For each $j \in \widetilde{V}$, restore $t_j \leftarrow t_j - \Delta$.
7 Update $UB_\infty \leftarrow U$.
```

5. Compact MILP formulation

We now present a compact MILP formulation for SALBP-S. The continuous variable ρ_∞ represents the stability radius to be maximized. In addition, x_{jk} is a binary variable equal to one if and only if task j is allocated to workstation k. The continuous nonnegative variable ξ_{jk} denotes the possible deviation of the processing time of task j on workstation k. Constraints (2) ensure that each task j is assigned to exactly one workstation. As stated in constraints (4), the stability radius is bounded by the minimal perturbation, which is not greater than UB_∞ since it is limited by constraints (3). Note that constraints (2) and (3) guarantee that, among all $k \in Q_j$, only one value ξ_{jk} is non-zero for any fixed $j \in V$. Constraints (5) enforce that the total load of each workstation does not exceed C, whatever possible processing time deviations within the stability ball. Remind that if the workstation is uncertain ($k \in \widehat{W}$), then all tasks assignable to that workstation are *a posteriori* uncertain: $\widetilde{V}_k = V_k$. In contrast, if the workstation is certain ($k \in W \setminus \widehat{W}$), then $\widetilde{V}_k \subseteq V_k$. Note in the left-hand-side of constraint (5) that ξ_{jk} has no impact if $j \notin \widetilde{V}_k$. Finally, inequalities (6) express the precedence constraints, followed by the domains of variables: (7)–(9).

$$\text{Maximize} \quad \rho_\infty \tag{1}$$

$$\sum_{k \in Q_j} x_{jk} = 1, \quad \forall j \in V \tag{2}$$

$$\xi_{jk} \le UB_\infty \cdot x_{jk}, \quad \forall j \in V, \ \forall k \in Q_j \tag{3}$$

$$\rho_\infty \le \sum_{k \in Q_j} \xi_{jk}, \quad \forall j \in V \tag{4}$$

$$\sum_{j \in V_k} t_j \cdot x_{jk} + \sum_{j \in \widetilde{V}_k} \xi_{jk} \le C, \quad \forall k \in W \tag{5}$$

$$\sum_{q=k}^{u_i} x_{iq} \le \sum_{q=k}^{u_j} x_{jq}, \quad \forall (i,j) \in A, \ \forall k \in Q_i \cap Q_j \tag{6}$$

$$\xi_{jk} \ge 0, \quad \forall j \in V, \ \forall k \in Q_j \tag{7}$$

$$x_{jk} \in \{0,1\}, \quad \forall j \in V, \ \forall k \in Q_j \tag{8}$$

$$\rho_\infty \ge 0 \tag{9}$$

Since the goal is to maximize the stability radius, in any feasible assignment there will be a critical workstation k^*, where the values of ξ_{jk^*} will be equal among all tasks $j \in \widetilde{V}_{k^*}$ assigned to k^*, and their sum will be equal to the idle time, so that ρ_∞ can reach its maximum. Also, note that the number of constraints depends on the range of the intervals Q_j, $j \in V$. Tighter assignment intervals lead to fewer variables and constraints, which stresses the importance of tightening these intervals (see Section 3). A last remark concerns the influence of UB_∞. Indeed, lower values of UB_∞ can lead to shorter computation times, which justifies the search for better upper bounds.

6. Dantzig-Wolfe decomposition

This section presents a Dantzig-Wolfe decomposition of the MILP formulation, discussed in Section 5. Let S denote the bounded set of integer points defined by constraints (3), (5), (7), and (8). It is not difficult to see that this set can be decomposed by workstation such that $S = S_1 \times \ldots \times S_m$. In addition, since x is binary, S coincides with B, which is the set of extreme points of the convex hull of S, denoted by $\mathtt{conv}(S)$ (see, *e.g.*, Wolsey, 1998). Thus, a reformulation is possible based on the *convexification* of these extreme points.

Note that variables ξ represent, in fact, the local stability radius for each uncertain task with respect to the workstation where it is assigned. Since we are dealing with the l_∞ norm, these values have to be equal for all uncertain tasks assigned to the same workstation. Therefore, given a point $r \in B$, $\rho_k^{(r)}$ can be defined as the local stability radius at workstation k such that

$$\rho_k^{(r)} = \sum_{j \in \widetilde{V}_k} \bar{\xi}_{jk}^{(r)} \Big/ \sum_{j \in \widetilde{V}_k} \bar{x}_{jk}^{(r)},$$

where $\bar{x}^{(r)}_{jk}$ and $\bar{\xi}^{(r)}_{jk}$ are respectively the values corresponding to variables x_{jk} and ξ_{jk} at the point $r \in B$. Hence, with variable ρ_k, representing the local stability radius in the workstation $k \in W$, the equality $\rho_k \cdot x_{jk} = \xi_{jk}$ can be enforced for each task $j \in V$ and workstation $k \in W$. Such additional constraints are linearized and implicitly included in the pricing sub-problem (cf. Section 6.1), being hereinafter part of the definition of S (and thus S_k, $k \in W$, too). In this way, given B_k, the set of all extreme points of $\texttt{conv}(S_k)$, for all $k \in W$, SALBP-S is reformulated by (10)–(16). Variable $\lambda^{(r)}_k$ is related to the extreme point $r \in B_k$, assuming value 1 if the extreme point $r \in B_k$ is selected for the workstation $k \in W$, and value 0 otherwise.

$$\text{Maximize} \quad \rho_\infty \tag{10}$$

$$\sum_{r\in B_k} \lambda^{(r)}_k = 1, \quad \forall k \in W \tag{11}$$

$$\sum_{k\in Q_j}\sum_{r\in B_k} \bar{x}^{(r)}_{jk} \cdot \lambda^{(r)}_k = 1, \quad \forall j \in V \tag{12}$$

$$\rho_\infty \le \sum_{k\in Q_j}\sum_{r\in B_k} \bar{\xi}^{(r)}_{jk} \cdot \lambda^{(r)}_k, \quad \forall j \in V \tag{13}$$

$$\sum_{q=k}^{u_i}\sum_{r\in B_q} \bar{x}^{(r)}_{iq} \cdot \lambda^{(r)}_q \le \sum_{q=k}^{u_j}\sum_{r\in B_q} \bar{x}^{(r)}_{jq} \cdot \lambda^{(r)}_q, \quad \forall (i,j) \in A, \ \forall k \in Q_i \cap Q_j \tag{14}$$

$$\rho_\infty \ge 0 \tag{15}$$

$$\lambda^{(r)}_k \in \{0,1\}, \quad \forall r \in B_k, \ \forall k \in W \tag{16}$$

The model above is called the *master problem*, and it is denoted by DW in the sequel. Considering $\bar{\rho}_L$ the value given by the linear relaxation of the original compact MILP formulation, and $\bar{\rho}_D$ the value obtained by relaxing the integrality in DW, it is known from Geoffrion (1974) that $\bar{\rho}_\infty \le \bar{\rho}_D \le \bar{\rho}_L$, where $\bar{\rho}_\infty$ is the value of the optimal stability radius. We call $\overline{DW}$ the linear relaxation of the master problem DW.

Finally, note that each extreme point $r \in B_k$ defines a set of tasks $\{j \in V_k : \bar{x}^{(r)}_{jk} = 1\}$ related to the workstation $k \in W$, called *pattern*, whose local stability radius is $\rho^{(r)}_k$. In the following, we consider each extreme point as a pattern and call *uncertain patterns* those with at least one uncertain task. Considering $\widetilde{B}_k \subseteq B_k$, the subset of uncertain patterns in B_k ($\widetilde{B}_k = B_k$, if $k \in \widehat{W}$), the following inequalities are valid for DW:

$$\rho_\infty \le \sum_{r\in \widetilde{B}_k} \left(\rho^{(r)}_k - UB_\infty\right) \cdot \lambda^{(r)}_k + UB_\infty, \quad \forall k \in W. \tag{17}$$

Since at most one pattern is assigned per workstation, these inequalities require that ρ_∞ is less than or equal to the local stability radius of the uncertain pattern assigned to workstation k. If no uncertain pattern has been assigned, then the inequality becomes loose ($\rho_\infty \le UB_\infty$). Indeed, Proposition 1 shows that inequalities (17) may improve the upper bound given by $\overline{DW}$, *i.e.*, $\bar{\rho}_D$.

Proposition 1. *Inequalities (13) do not dominate (17), and (17) do not dominate (13).*

Proof. Using a small example, we show that there exist feasible solutions of $\overline{DW}$ satisfying (13) but not (17), and vice-versa. We consider the problem with three uncertain tasks having equal processing time $t_j = 2$, $j \in \{1,2,3\}$, and 2 workstations with cycle time $C = 10$. The precedence graph has two arcs, (1,2) and (1,3).

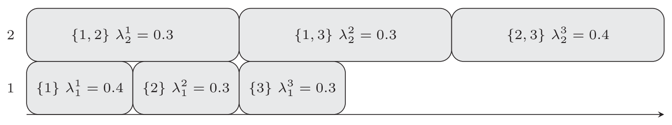

Fig. 1. Solution violating (17).

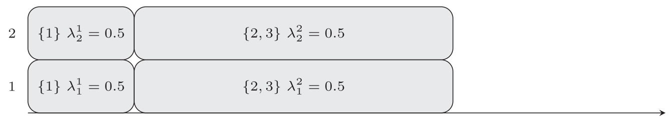

Fig. 2. Solution violating (13).

Considering the formulation (10)–(16), a feasible solution is given by patterns {1}, {2}, {3} in workstation 1, and patterns {1, 2}, {1, 3}, and {2, 3} in workstation 2. The solution is illustrated in Fig. 1, where the tasks constituting the patterns are provided in curly braces. The variable values corresponding to each pattern are, respectively, $\lambda^1_1 = 0.4$, $\lambda^2_1 = 0.3$, $\lambda^3_1 = 0.3$, $\lambda^1_2 = 0.3$, $\lambda^2_2 = 0.3$, and $\lambda^3_2 = 0.4$. The local stability radii of patterns in workstation 1 are equal to 8, and the local stability radii of patterns in workstation 2 are equal to 3.

The objective value $\bar{\rho}_D$ of the solution in Fig. 1 is 4.5. This solution violates (17) since for workstation 2 we have $0.4 \times 3 + 0.3 \times 3 + 0.3 \times 3 = 3$, which is less than 4.5.

We now consider the formulation provided by (10)–(12) and (14)–(17). A feasible solution is given by the patterns {1} and {2, 3}, in workstations 1 and 2. Fig. 2 illustrates this solution. The variable values corresponding to each pattern are all equal to 0.5. The local stability radii of patterns {1} are equal to 8, while the local stability radii of patterns {2, 3} are equal to 3.

The objective value $\bar{\rho}_D$ of the solution in Fig. 2 is 5.5. This solution violates (13) since for tasks 2 and 3 we have $0.5 \times 3 + 0.5 \times 3 = 3$, which is less than 5.5. Note that the optimal solution value of the proposed example is 3, and that both inequalities (13) and (17) are needed so that $\overline{DW}$ gives the optimal upper bound. □

6.1. Column generation

A column generation algorithm is proposed to compute $\bar{\rho}_D$. We consider that a known feasible solution is available, producing the initial subset of columns. At each iteration, m pricing sub-problems are solved to generate columns, each corresponding to a pattern and associated with a variable $\lambda^{(r)}_k$. The $\overline{DW}$ model is then re-optimized with this additional set of columns. For each workstation $k \in W$, the objective function of pricing sub-problem PS_k depends on the dual variables of $\overline{DW}$, defined as follows:

- $\mu_k \in \mathbb{R}$ is the dual variable associated with constraint (11) for all $k \in W$,
- $\kappa_j \in \mathbb{R}$ is the dual variable associated with constraint (12) for all $j \in V$,
- $\omega_j \ge 0$ is the dual variable associated with constraint (13) for all $j \in V$,
- $\pi_{ijk} \ge 0$ is the dual variable associated with constraint (14) for all $(i,j) \in A$ and for all $k \in [l_j, u_i]$,
- $\gamma_k \ge 0$ is the dual variable of DW associated with constraint (17) for all $k \in W$.

The reduced cost $\bar{c}^{(r)}_k$ of variable $\lambda^{(r)}_k$ in $\overline{DW}$ is computed as follows:

$$0 - \mu_k + \left(\rho^{(r)}_k - UB_\infty\right) \cdot \gamma_k - \sum_{j\in V_k} \bar{x}^{(r)}_{jk} \cdot \kappa_j + \sum_{j\in V_k} \bar{\xi}^{(r)}_{jk} \cdot \omega_j$$

$$- \sum_{(i,j)\in A: l_j \le k \le u_i} \sum_{q=l_j}^{k} \bar{x}^{(r)}_{ik} \cdot \pi_{ijq} + \sum_{(i,j)\in A: l_j \le k \le u_j} \sum_{q=l_j}^{\min\{k,u_i\}} \bar{x}^{(r)}_{jk} \cdot \pi_{ijq}$$

The above expression leads to a non-linear pricing objective function due to the term $\rho_k^{(r)}$, which is an unknown value (ρ_k) expressed as $\sum_{j\in\widetilde{V}_k}\xi_{jk}/\sum_{j\in\widetilde{V}_k}x_{jk}$, ξ and x being in their turn, variable vectors of the pricing sub-problem. Fortunately, a linearization is straightforward since $\xi_{jk}=\rho_k\cdot x_{jk}$, for all $j\in\widetilde{V}_k$, $k\in W$. Thus, for any $k\in W$, the pricing sub-problem PS_k is defined as follows:

$$\text{Maximize} \quad \sum_{j\in V_k}\omega_j\cdot\xi_j^k+\gamma_k\cdot\rho^k$$

$$-\sum_{j\in V_k}\left(\kappa_j+\sum_{i\in V_k:(j,i)\in A,l_i\le k\le u_j}\ \sum_{q=l_i}^{k}\pi_{jiq}-\sum_{i\in V_k:(i,j)\in A,l_j\le k\le u_j}\ \sum_{q=l_j}^{\min\{k,u_i\}}\pi_{ijq}\right)\cdot x_j^k$$

$$\xi_{jk}\le UB_\infty\cdot x_{jk},\quad \forall j\in V_k$$

$$\sum_{j\in V_k}t_j\cdot x_{jk}+\sum_{j\in\widetilde{V}_k}\xi_{jk}\le C$$

$$\xi_{jk}\le\rho_k,\quad \forall j\in V_k$$

$$\xi_{jk}\ge\rho_k-UB_\infty\cdot(1-x_{jk}),\quad \forall j\in\widetilde{V}_k$$

$$\xi_{jk}\ge 0,\quad \forall j\in V_k$$

$$x_{jk}\in\{0,1\},\quad \forall j\in V_k$$

$$\rho_k\ge 0$$

The first two sets of constraints are derived from S_k, followed by the linearization constraints and the domains of the variables. Note that the first set of linearization constraints also includes bounds for certain tasks too. Indeed, by modeling, variables ξ are also defined for certain tasks in certain workstations, but they do not impact the local stability radius. In that manner, they can be artificially set to $\rho_k^{(r)}$ to improve the upper bound, which is enforced by these additional bounding constraints.

In Peeters & Degraeve (2006), the authors proposed the addition of constraints (18) in the pricing sub-problem to reinforce the precedence relations. The effectiveness of such constraints in the present case is investigated in Section 7.3

$$x_{ik}+x_{lk}-1\le x_{jk},\quad \forall (i,j)\in A,\ \forall l\in S_j\cap V_k. \tag{18}$$

If PS_k has a feasible solution, whose objective value is p_k, and if $p_k-\mu_k-UB_\infty\cdot\gamma_k$ is strictly positive, then this solution may be included in $\overline{DW}$ as a new column. If, for all $k\in W$, no column can be included, then the current solution of $\overline{DW}$ is optimal. At this point, the column generation algorithm terminates, and the upper bound $\bar{\rho}_D$ is returned.

We recall that the dual objective value of $\overline{DW}$ is computed as $\sum_{k\in W}UB_\infty\cdot\gamma_k+\sum_{k\in W}\mu_k+\sum_{j\in V}\kappa_j$. Then, it is known that for the current feasible set of columns and the corresponding dual point, an upper bound over $\bar{\rho}_D$ is obtained by $\sum_{j\in V}\kappa_j+\sum_{k\in W}p_k$ (see Wolsey, 1998, for basic theory). This property allows us to obtain an upper bound from $\overline{DW}$ even if column generation stops before reaching optimality (due to the imposed time or iteration limit). The quality of the upper bound returned by this column generation algorithm is studied through computational experiments in the next section.

Table 2
Groups of instances.

Group	Graph Structure	OS	Times Distribution
BN_2_BM	bottleneck	0.2	bimodal
BN_2_PB	bottleneck	0.2	peak at the bottom
BN_2_PM	bottleneck	0.2	peak in the middle
BN_6_BM	bottleneck	0.6	bimodal
BN_6_PB	bottleneck	0.6	peak at the bottom
BN_6_PM	bottleneck	0.6	peak in the middle
CH_2_BM	chains	0.2	bimodal
CH_2_PB	chains	0.2	peak at the bottom
CH_2_PM	chains	0.2	peak in the middle
CH_6_BM	chains	0.6	bimodal
CH_6_PB	chains	0.6	peak at the bottom
CH_6_PM	chains	0.6	peak in the middle
MX_2_BM	mixed	0.2	bimodal
MX_2_PB	mixed	0.2	peak at the bottom
MX_2_PM	mixed	0.2	peak in the middle
MX_6_BM	mixed	0.6	bimodal
MX_6_PB	mixed	0.6	peak at the bottom
MX_6_PM	mixed	0.6	peak in the middle
MX_9_BM	mixed	0.9	bimodal
MX_9_PB	mixed	0.9	peak at the bottom
MX_9_PM	mixed	0.9	peak in the middle

7. Computational experiments

7.1. Experimental settings

The computational experiments are conducted on a CentOS Linux machine with 8 GB of RAM and an Intel Xeon CPU E5-2680 v4 processor at 2.40 GHz. All the linear and mixed-integer linear programs are addressed with the state-of-the-art solver IBM CPLEX 12.10, for which a single thread is used. The algorithms are implemented in C++, and CPLEX is called through the Concert Technology. A testbed of 525 benchmark instances[1] from Otto et al. (2013) is considered, each one having 100 tasks.

The instances are divided into 21 groups based on the probability distribution used for generating task processing times, the structure of the precedence graph, and the order strength (OS). The latter is a density indicator of precedence graphs computed as $2\cdot|A|/(n\cdot(n-1))$. Each group contains 25 instances, and their characteristics are presented in Table 2. Column 'Group' presents the label given to each group, followed by the graph structure (column 'Graph Structure'). There are three types of graphs: the ones containing bottleneck tasks, the ones containing at least 40% of tasks in chains, and the mixed ones, containing a bottlenecks and chained tasks in a random proportion. Finally, the columns 'OS' and 'Times Distribution' of Table 2 respectively present the graphs' OS and the distribution of probability used to generate task processing times. A bimodal distribution consists of a combination of two normal distributions with means centered around small and large times. The term 'peak in the bottom' stands for a normal distribution with the mean centered around small times, while 'peak in the middle' refers to normal distributions averaged to $0.5\cdot C$, where C is the cycle time set to 1000. We refer the reader to Otto et al. (2013) for more details on those probability distributions and other instance features.

The considered instances were originally designed for the problem of minimizing the number of workstations (SALBP-1). Here, the number of workstations is fixed and has been set to the best-known solution value of SALBP-1 (Morrison et al., 2014; Otto et al., 2013). Furthermore, the set of uncertain tasks $\widetilde{V}^1$ (resp. the set of uncertain workstations $\widehat{W}$) is built by taking the first $|\widetilde{V}^1|$

[1] Instances and detailed results can be found at: http://pagesperso.ls2n.fr/~gurevsky-e/data/R-ALBP.zip

(resp. $|\widehat{W}|$) elements of a random permutation of $\{1,\dots,n\}$ (resp. $\{1,\dots,m\}$) associated with each instance. We consider three different levels of uncertainty:

(i) $|\widetilde{V}^1| = \lceil 0.5n \rceil$ and $|\widehat{W}| = 0$;
(ii) $|\widetilde{V}^1| = \lceil 0.5n \rceil$ and $|\widehat{W}| = \lceil 0.5m \rceil$;
(iii) $|\widetilde{V}^1| = n$ and $|\widehat{W}| = 0$.

In the next subsections, the average results per group are analyzed. It is worth noting that the computed average values take into account the results for the three different levels of uncertainty. Since the remarks made in the sequel cover the three levels of uncertainty, we choose to present aggregated results to improve readability. The detailed results are available online as mentioned earlier.

First, the tightness of the assignment intervals returned by the method of Patterson and Albracht and the method discussed in this paper are compared in Section 7.2. More specifically, since variable x_{jk} can be set to zero if workstation k is not in the assignment interval Q_j of task j, the two methods for computing the assignment intervals are compared on the basis of the average number of such pre-fixed decision variables.

In Section 7.3, we provide a comparison between different versions of the Dantzig-Wolfe approach. The experiments intend to confirm the effectiveness of the proposed valid inequalities (17) and to test strategies previously applied in the literature for simple assembly line balancing problems without uncertainty. Notably, we test the effect of removing the precedence constraints from the master problem and adding constraints (18) to the pricing sub-problem, as these strategies have been applied in Peeters & Degraeve (2006) for the SALBP-1, providing promising results.

We recall that the main goal of the proposed Dantzig-Wolfe approach is to obtain better upper bounds for the stability radius. In Section 7.4, we investigate the quality of the upper bound provided by the linear relaxation of the proposed extended formulation (10)–(17). For that purpose, two approaches are compared: the first one, denoted by CF (compact formulation), consists in solving the compact MILP formulation of Section 5 with a time limit of 600 seconds. The second one, denoted by CFCG, is a two-stage approach relying on the compact MILP formulation followed by the Dantzig-Wolfe decomposition introduced in Section 6. In the first stage, the compact MILP formulation of Section 5 is addressed using CPLEX with a time limit of 300 seconds. Then, the resulting upper bound on the stability radius is used as the value for UB_∞ in PS_k for every $k \in W$. The second stage of CFCG (named CG) runs the column generation algorithm proposed in Section 6.1 with a time limit of 300 seconds, attempting to reduce the upper bound obtained in the first stage. Note that the best solution found by CPLEX during the first stage (including at pre-solve) is used to produce initial columns for $\overline{DW}$. Since CG may benefit from enhanced upper and lower bounds, aiming for a fair comparison, the CF approach is actually restarted at 300 seconds of computational time to take into account the enhanced bounds. This includes updating UB_∞ in inequalities (3) and recomputing assignment values with the enhanced lower bound. As one can expect, the current best solution is used as restarting point.

In Section 7.5, we discuss about the possibility of obtaining an improved feasible solution from the columns generated with the Dantzig-Wolfe approach. Finally, Section 7.6 focuses on managerial insights. We first study the impact of having uncertain workstations grouped at the beginning of the line, in the end, or randomly distributed. We then investigate the effect of increasing the number of workstations and the effect of varying the cycle time on the stability radius value.

7.2. Reductions in assignment intervals

The assignment intervals computed according to the classical method, presented in Patterson & Albracht (1975) (‘PA’), are compared with the assignment intervals returned by the method discussed in Section 3 (‘NAI’). The comparison is performed on the number of binary variables x_{jk} ($j \in V$ and $k \in W$) set to zero due to the computed assignment intervals. Note that those variables are present in the compact formulation and in the pricing subproblem. Hence, we also provide the impact of each method on the performance of the CG algorithm.

The effect of providing an initial lower bound is analyzed too. Indeed, as explained in Section 3, if a feasible solution with a strictly positive stability radius is known, this value can serve as a lower bound to the stability radius. Then, the assignment intervals can be computed after increasing the duration of each uncertain task by this lower bound.

Each row in Table 3 shows the average results for each group of instances. Columns 2 to 5 present the results when using PA, while columns 6 to 10 show the results when using NAI. Columns ‘Fixed Vars’ correspond to the average number of decision variables set to zero thanks to the assignment intervals computed with each method. These values are given for the case where no initial lower bound is used (columns ‘No LB’) and for the case where an initial solution is available (columns ‘With LB’). Here, the considered initial solution is the one obtained after solving the compact formulation for 300 seconds. Next, columns under ‘CG’ present the performances of the CG algorithm in terms of upper bounds (columns ‘Dev’) and computational times in seconds (columns ‘Time’). The values in columns ‘Dev’ correspond to deviations, computed as $(UB_X - UB^*)/UB^* \times 100$, where UB_X is the upper bound obtained with each method, and UB^* is the best upper bound found with one of them. Finally, column ‘+Arcs’ provides the average number of new precedence arcs that have been added to the precedence graph by Step 4 of Algorithm 1. These last values were obtained using an initial lower bound. In the case where no initial lower bound is available, the addition of arcs was almost never observed.

The computational time of PA and NAI is below 0.1 seconds per instance. However, when assignment intervals are computed repeatedly, as in the bisection method to produce an upper bound on the stability radius (see Section 4), the running time of NAI becomes slightly more significant as can be seen in Section 7.4.1.

The results of Table 3 show that both PA and NAI take advantage of the availability of an initial solution, which confirms that a fast heuristic is beneficial to pre-set some variables without compromising optimality. As shown in Section 3, the assignment intervals returned by NAI are at least as tight as those of PA, so the number of decision variables that can be pre-fixed to zero is always larger when NAI is used. For MX_2_PB and MX_6_PB, NAI does not improve over PA. Generally, the ‘PB’ instances (those for which the time distribution is ‘peak at the bottom’) only marginally benefit from NAI. On the whole instance set, however, the new assignment intervals increase the number of decision variables that can be pre-fixed to zero by 2.2% on average. Moreover, using a lower bound increases the number of pre-fixed variables by 1% without increasing running times significantly.

The NAI’s procedure for identifying new precedence constraints is rarely successful, especially when no initial lower bound is available. However, the column ‘+Arcs’ shows that those new constraints are slightly more frequent when an initial lower bound is used.

With respect to the impact on the performances of the CG algorithm, one can observe that the greatest advantages of the NAI are obtained for the ‘peak in the middle’ groups. In fact, these are the groups for which the NAI was the most successful in pre-fixing variables. In the best cases, the use of NAI resulted in an improve-

Table 3
Average number of binary variables pre-fixed to zero and the impact of assignment intervals on CG.

Group	PA				NAI				
	Fixed Vars		CG		Fixed Vars		CG		
	No LB	With LB	Dev	Time (s)	No LB	With LB	Dev	Time (s)	+Arcs
BN_2_BM	379	385	0.00	300.0	387	393	0.00	300.0	0.0
BN_2_PB	210	217	0.00	300.0	211	218	0.00	300.0	0.0
BN_2_PM	974	974	3.02	189.7	1052	1052	0.00	181.1	0.0
BN_6_BM	1258	1271	0.14	300.0	1260	1273	0.00	300.0	0.0
BN_6_PB	746	773	0.00	278.0	746	773	0.00	278.3	0.0
BN_6_PM	2933	2934	0.00	77.6	2977	2978	0.00	76.5	0.0
CH_2_BM	386	390	0.00	300.0	395	399	0.00	300.0	0.0
CH_2_PB	208	214	0.00	300.0	209	215	0.00	300.0	0.0
CH_2_PM	968	968	2.89	192.6	1052	1052	0.00	182.5	0.0
CH_6_BM	1272	1280	0.17	282.8	1281	1288	0.00	281.9	0.3
CH_6_PB	728	753	0.00	273.6	728	753	0.00	271.7	0.0
CH_6_PM	2975	2976	0.02	96.2	3054	3055	0.00	90.1	0.0
MX_2_BM	393	398	0.00	300.0	398	403	0.00	300.0	0.0
MX_2_PB	210	217	0.00	300.0	210	217	0.00	300.0	0.0
MX_2_PM	979	979	2.35	184.8	1039	1039	0.01	175.7	0.0
MX_6_BM	1271	1284	0.07	288.2	1273	1286	0.00	287.5	0.1
MX_6_PB	756	780	0.00	284.2	756	780	0.00	283.3	0.0
MX_6_PM	2940	2941	0.00	88.6	2978	2979	0.00	83.1	0.0
MX_9_BM	1955	1996	0.48	4.2	1983	2022	0.00	4.6	0.6
MX_9_PB	1154	1198	0.00	0.8	1161	1205	0.00	0.9	1.2
MX_9_PM	4468	4483	0.99	40.7	4627	4648	0.00	33.4	0.0
Overall Average	**1293**	**1305**	**0.48**	**208.7**	**1323**	**1335**	**0.00**	**206.2**	**0.1**

Table 4
Different versions of the Dantzig-Wolfe approach.

Version	Enforcing (17)	Enforcing (14)	Enforcing (18)
DW-0.0.1	No	No	Yes
DW-0.1.0	No	Yes	No
DW-1.0.0	Yes	No	No
DW-1.1.0	Yes	Yes	No
DW-1.1.1	Yes	Yes	Yes
DW-1.0.1	Yes	No	Yes

ment of more than 3% on upper bounds and a reduction of computational times by 8 to 10 seconds. On the overall average (last row of Table 3), using NAI provided a 0.48% improvement in terms of upper bound and a 2-second reduction in terms of computational time.

Since NAI is more successful than PA in reducing assignment intervals and thus filtering assignment variables, all methods in the following experiments implement NAI for pre-processing. Indeed, fixing variables to zero helps to reduce the problem's size, enhancing the performance of the methods, as observed in Pirogov et al. (2021). In Section 7.4.1, we provide results showing that the bisection method (Algorithm 2) also provides better upper bounds while applying NAI rather than PA.

7.3. Comparing different strategies for the Dantzig-Wolfe approach

Six versions of the Dantzig-Wolfe approach are compared on the basis of the returned upper bound for the stability radius (in Table 5) and in terms of running time (in Table 6). Each row in Table 4 characterizes a version of the Dantzig-Wolfe approach. These versions differ by the presence or absence of the valid inequalities (17), by the presence or absence of the precedence constraints (14) in the master problem, and by the presence or absence of precedence constraints (18) in the pricing sub-problem.

Tables 5 and 6 provide averaged results on the 21 groups of instances indicated by the first column. Table 5 displays the average deviation of the upper bound obtained with the corresponding version, with respect to the best available bound. The deviations are computed as $(UB_X - UB^*)/UB^* \times 100$, where UB_X corresponds to the upper bound provided by each version, and UB^* is the best upper bound found by one of them. Table 6 displays the average running times.

It can be seen in Table 5 that all the versions of the Dantzig-Wolfe approach tend to return very similar upper bounds on the stability radius for the BM and PM instances, *i.e.*, for the 'bimodal' and 'peak in the bottom' processing times distribution. By contrast, the quality of upper bounds becomes significantly different for the instances with 'peak in the middle' processing times distribution. The worst values are returned by DW-0.1.0 and DW-0.0.1, while the results obtained by DW-1.1.0 and DW-1.0.1 are significantly better, which illustrates the beneficial effect of constraint (17). More generally, all the versions for which constraint (17) is implemented outperform the versions for which this constraint is not present. For that reason, versions DW-0.0.0 and DW-0.1.1 do not produce competitive results and are not considered.

Considering the overall averages, the best results were obtained with DW-1.1.1 and DW-1.0.1, which suggests that the precedence constraints can be relaxed in the master problem provided that they are enforced in the pricing sub-problem by inequalities (18). Furthermore, the results obtained with versions DW-1.0.0 and DW-1.1.0 demonstrate that removing inequalities (18) is detrimental to upper bound quality. This suggests that reinforcing the pricing, *i.e.*, enforcing precedence inequalities (18) in the pricing sub-problem, has a positive impact on the upper bound.

Table 6 displays the running time of the seven versions of the Dantzig-Wolfe approach, where a time limit of 300 seconds is enforced for each instance. It can now clearly be seen that the worst versions in terms of upper bound, *i.e.*, DW-0.0.1 and DW-0.1.0 are also the fastest ones. This indicates that constraint (17) has a nasty effect on the running time. This is not very surprising, because the corresponding inequalities are very dense in the sense that if pattern r for workstation k contains at least one uncertain task, or if k is an uncertain workstation, then the coefficient of variable $\lambda_k^{(r)}$ in constraint (17) is nonzero.

The best versions in terms of upper bound, *i.e.*, DW-1.0.1 and DW-1.1.1 are also the slower ones. As a consequence, versions

Table 5
Average deviations to the best upper bound among versions of the Dantzig-Wolfe approach.

Group	DW-0.0.1	DW-0.1.0	DW-1.0.0	DW-1.1.0	DW-1.1.1	DW-1.0.1
BN_2_BM	0.00	0.00	0.00	0.00	0.00	0.00
BN_2_PB	0.00	0.00	0.00	0.00	0.00	0.00
BN_2_PM	821.89	821.89	50.55	51.69	5.63	3.65
BN_6_BM	1.54	1.54	1.11	0.45	0.02	0.43
BN_6_PB	0.00	0.00	0.00	0.00	0.00	0.00
BN_6_PM	521.07	696.34	239.28	226.02	0.00	0.02
CH_2_BM	0.00	0.00	0.00	0.00	0.00	0.00
CH_2_PB	0.00	0.00	0.00	0.00	0.00	0.00
CH_2_PM	527.69	595.19	6.80	8.32	11.98	2.02
CH_6_BM	2.72	2.72	2.23	1.08	0.00	1.25
CH_6_PB	0.00	0.00	0.00	0.00	0.00	0.00
CH_6_PM	438.98	474.55	82.34	77.94	0.00	2.10
MX_2_BM	0.00	0.00	0.00	0.00	0.00	0.00
MX_2_PB	0.00	0.00	0.00	0.00	0.00	0.00
MX_2_PM	461.16	480.43	24.67	26.42	5.30	2.64
MX_6_BM	2.01	2.01	1.56	0.62	0.01	0.89
MX_6_PB	0.04	0.04	0.04	0.04	0.02	0.04
MX_6_PM	390.22	446.31	115.04	110.68	0.00	0.41
MX_9_BM	0.11	0.11	0.11	0.11	0.00	0.02
MX_9_PB	0.00	0.00	0.00	0.00	0.00	0.00
MX_9_PM	453.65	554.55	306.03	254.04	0.00	0.35
Overall Average	**172.43**	**194.08**	**39.51**	**36.07**	**1.09**	**0.66**

Table 6
Average computational times for different versions of the Dantzig-Wolfe approach.

Group	DW-0.0.1	DW-0.1.0	DW-1.0.0	DW-1.1.0	DW-1.1.1	DW-1.0.1
BN_2_BM	25.04	42.62	297.84	299.55	300.00	300.00
BN_2_PB	13.24	16.46	287.12	299.52	300.00	300.00
BN_2_PM	30.62	30.02	164.32	201.41	210.54	181.14
BN_6_BM	8.18	6.60	288.44	297.80	300.00	300.00
BN_6_PB	4.72	3.55	235.11	292.88	299.37	278.26
BN_6_PM	11.27	8.31	96.40	158.40	90.48	76.52
CH_2_BM	30.66	63.51	300.00	300.00	300.00	300.00
CH_2_PB	13.07	18.33	292.30	300.00	300.00	300.00
CH_2_PM	28.63	38.86	151.25	198.98	220.10	182.49
CH_6_BM	10.03	9.39	268.08	282.07	283.47	281.87
CH_6_PB	6.26	4.59	213.82	281.34	290.71	271.70
CH_6_PM	11.72	11.28	94.84	193.71	162.19	90.06
MX_2_BM	24.45	44.71	300.00	300.00	300.00	300.00
MX_2_PB	13.24	16.81	278.38	296.67	299.19	300.00
MX_2_PM	28.26	30.05	162.30	195.86	208.10	175.74
MX_6_BM	8.27	7.32	260.06	285.72	291.82	287.46
MX_6_PB	4.88	3.87	224.80	291.84	300.00	283.31
MX_6_PM	11.32	10.21	95.29	172.34	127.20	83.08
MX_9_BM	0.60	0.48	2.18	4.00	5.04	4.57
MX_9_PB	0.08	0.06	0.43	0.58	1.55	0.88
MX_9_PM	5.07	3.41	45.67	81.66	44.06	33.40
Overall Average	**13.79**	**17.64**	**193.27**	**225.44**	**220.66**	**206.21**

DW-1.0.0 and DW-1.1.0 managed to provide better upper bounds in some cases, namely 3.5% of the instances. This is due to the fact that the addition of inequalities (18) produces harder sub-problems, which slows down the column generation algorithm. Therefore, in some cases, better upper bounds are obtained with versions DW-1.0.0 and DW-1.1.0 since they are able to perform more iterations than versions DW-1.0.1 and DW-1.1.1 within the same time limit. However, if more time is allotted for the four versions to run, *e.g.* one hour, the best upper bounds are definitely provided by versions DW-1.0.1 and DW-1.1.1 (cf. detailed results on the previously mentioned link). In fact, even though the addition of (18) slows down the solution process of the sub-problems, fewer iterations are needed to converge, which means that implementing these inequalities in the pricing sub-problem is a competitive trade-off.

In Table 7, we provide the results of the two most promising versions applying a time limit of one hour. Since, for this extended time limit, versions DW-1.0.0 and DW-1.1.0 are outperformed by DW-1.0.1 and DW-1.1.1, Table 7 displays the results of the last two versions only. Still, the complete set of results is available online, as previously mentioned. Table 7 also excludes the instances which could be previously solved within a 300-second time limit. The lines present the average results per group indicated in the first column. The last line provides the overall average values for the whole set of considered instances. Followed by the computational times (in seconds), columns ‘Dev’ present the quality of the upper bound provided by every version for each group of instances. As previously, the quality is measured by deviations computed as $(UB_X - UB^*)/UB^* \times 100$, where UB_X corresponds to the upper bound provided by each version, and UB^* is the best upper bound found by one of them. The two columns under ‘Time-Stopped’ present the number of instances for which DW-1.1.1 and DW-1.1.0 run until the time limit without converging. These values are provided for the time limit of 300 seconds (column ‘300s’) and for the time limit of one hour (column ‘3600s’). Finally, the last column provides the average improvements in the upper bounds obtained with the extra computational time. The improvements are computed as $(UB^*_{3600} - UB^*_{300})/UB^*_{300} \times 100$, where UB^*_{300} and

Table 7
Average results for two versions of the Dantzig-Wolfe approach with a time limit of one hour.

Group	DW-1.1.1		DW-1.0.1		Time-Stopped		
	Dev	Time	Dev	Time	300s	3600s	UB Improv
BN_2_BM	0.02	3406.74	0.46	3372.12	75	67	−0.50
BN_2_PB	0.00	3540.63	0.04	3165.41	75	60	−0.04
BN_2_PM	0.00	645.84	0.37	565.48	22	0	−7.42
BN_6_BM	0.01	2385.28	0.76	2448.64	75	31	−1.35
BN_6_PB	0.00	3511.35	0.46	3308.86	61	53	−0.81
CH_2_BM	0.00	3596.20	0.80	3562.79	75	73	−0.99
CH_2_PB	0.00	3559.18	0.02	3287.88	75	66	−0.02
CH_2_PM	0.00	629.53	0.13	483.73	12	0	−8.26
CH_6_BM	0.01	2125.33	1.51	2135.61	68	16	−1.98
CH_6_PB	0.01	3408.45	0.73	3145.45	60	48	−1.07
MX_2_BM	0.00	3525.41	0.44	3553.99	75	69	−0.51
MX_2_PB	0.00	3515.54	0.04	3064.18	74	58	−0.04
MX_2_PM	0.00	591.22	0.02	590.05	17	0	−14.04
MX_6_BM	0.01	2353.58	1.08	2340.89	69	24	−1.33
MX_6_PB	0.00	3462.62	0.47	3112.22	65	50	−0.59
Overall Average	**0.00**	**2683.79**	**0.49**	**2542.49**	**60**	**41**	**−2.60**

UB^*_{3600}, represent the best upper bound obtained with a time limit of 300 and 3600 seconds respectively.

From Table 7, one can observe that for 'peak in the middle' groups, DW-1.1.1 and DW-1.0.1 may converge much earlier than the time limit of 3600 seconds. For those instances, DW-1.1.1 takes 622 seconds and DW-1.0.1 takes 546 seconds on average. Indeed, the extra time is sufficient to close the duality gaps of all 'peak in the middle' instances, which in the best case (MX_2_PM) represents an improvement of 14% on the upper bound.

We remark that duality gaps are also closed for a considerable number of 'bimodal' instances with OS=0.6. However, in this case, computational times are much closer to the time limit, and the improvements in terms of upper bounds remain under 1.98%. For other groups of instances, the results are less significant. In the remaining groups, the duality gap can be closed for 8 to 10 more instances on average. In the worst case, that number decreases to 2. With respect to the upper bounds, the improvements remain under 1.07% on average.

Finally, we compare the two DW versions considering the extended time limit. We observe that DW-1.1.1 provides the best upper bounds, but spends more computational time than DW-1.0.1 (about 140 seconds on the overall average). Even though the upper bounds provided by the later version are worse, they remain close to the best ones. Indeed, the deviations of the DW-1.0.1 version reach 1.51% in the worse case (CH_6_BM), but remain under 0.5% in the overall average.

As a conclusion of this study, the only Dantzig-Wolfe version that will be considered in the sequel is DW-1.0.1, since for the considered time limit of 300 seconds it provides the best overall performance. A natural idea to enhance DW-1.0.1 is to add precedence constraints on-the-fly if they are found to be violated. This seventh version has also been tested, but the computational time turns out to be even larger than the one of DW-1.1.1, and the improvements in terms of upper bounds are not significant.

7.4. Quality of the upper bounds

We now investigate the quality of the upper bounds returned by the proposed methods. In Section 7.4.1, we first compare the upper bounds obtained by the bisection method (Algorithm 2) with the ones computed as per Rossi et al. (2016). Moreover, two versions of the bisection method are compared: the first one uses PA to compute the assignment intervals, whereas the second one relies on NAI. Then in Section 7.4.2, the quality of the upper bounds returned by CF and CFCG is compared.

7.4.1. Improvements brought by the bisection method over the initial upper bound

In Table 8, PA refers to the use of the method of Patterson & Albracht (1975), and NAI refers to the method discussed in Section 7.2 to compute the assignment intervals. The columns 'BS UB Improv (%)' provide the gains obtained with the bisection method over the upper bound computed by the combinatorial approach presented in Rossi et al. (2016). To the best of our knowledge, the latter work corresponds to the state-of-the-art and sole method for computing upper bounds for the SALBP-S, other than CF. The gains in terms of upper bound are computed as $(UB_{BS} - UB_0)/UB_0$, where UB_{BS} is the upper bound provided by the bisection method, and UB_0 is the upper bound given by the combinatorial approach. We note that $UB_{BS} \leq UB_0$, since UB_0 is an input for the bisection method, hence all gains are negative, which means that the bisection method can only yield tighter upper bounds.

As expected, the upper bound improvements are much higher when NAI is used instead of PA, and this improvement in terms of upper bound quality is obtained at a modest extra computational cost of 0.12 seconds on average. Hence, considering the results obtained with NAI, the bisection method improves upper bounds for all groups of instances except three: 'CH_2_PM', 'MX_2_BM' and 'MX_2_PM'. In general, the bisection method is less effective for instances with a PM ('peak in the middle') time distribution, and when the precedence graph is sparse (OS equal to 0.2). By contrast, the best results are obtained when the precedence graph is highly dense, as in groups whose labels start by 'MX_9' (OS equal to 0.9). This is not surprising, since the method relies on the infeasibility caused by sharp assignment intervals. Indeed, the more precedence constraints, the sharper those intervals tend to be. With respect to the time distribution, according to Otto et al. (2013) and Morrison et al. (2014), the PM instances are the hardest ones, which may explain why the improvements were less significant. The PB ('peak at the bottom') instances are those for which the improvement is the most notable.

7.4.2. Comparison of CF and CFCG

We now compare CF and CFCG, which are both allowed to run for 600 seconds on each instance. Note that the two approaches take advantage of the pre-processing strategies introduced in Sections 3 and 4, and that CPLEX is warm started with the solution returned by an adaptation of the heuristic presented in Pirogov et al. (2021). In case the heuristic could not find a feasible solution, the one provided by Morrison et al. (2014) for the SALBP-1 is used to construct a feasible solution for the SALBP-S. In

Table 8
Average reductions provided by the bisection method in the initial upper bound.

Group	PA		NAI	
	BS UB Improv (%)	Time (s)	BS UB Improv (%)	Time (s)
BN_2_BM	0.00	0.00	−2.04	0.00
BN_2_PB	−2.97	0.00	−8.88	0.00
BN_2_PM	0.00	0.00	−0.55	0.00
BN_6_BM	−19.19	0.00	−31.23	0.01
BN_6_PB	−43.38	0.00	−51.54	0.03
BN_6_PM	−0.29	0.00	−4.56	0.01
CH_2_BM	−0.33	0.00	−4.08	0.00
CH_2_PB	0.00	0.00	−1.51	0.00
CH_2_PM	0.00	0.00	0.00	0.00
CH_6_BM	−11.01	0.00	−31.19	0.04
CH_6_PB	−31.51	0.00	−51.16	0.11
CH_6_PM	−2.29	0.00	−11.23	0.22
MX_2_BM	0.00	0.00	0.00	0.00
MX_2_PB	−0.25	0.00	−3.55	0.00
MX_2_PM	0.00	0.00	0.00	0.00
MX_6_BM	−12.35	0.00	−29.34	0.03
MX_6_PB	−35.80	0.00	−54.82	0.12
MX_6_PM	−0.02	0.00	−5.42	0.32
MX_9_BM	−73.89	0.00	−82.90	0.66
MX_9_PB	−81.24	0.00	−86.44	0.44
MX_9_PM	−10.68	0.00	−23.87	0.57
Overall average	**−15.49**	**0.00**	**−23.06**	**0.12**

Table 9
Upper bound improvement and computational effort resulting from CFCG over CF.

Group	CF GAP (%)	CF UB Improv (%)	CG UB Improv (%)	DEV UB (%)	CFCG Time (s)
BN_2_BM	57.50	−0.39	0.00	0.40	600
BN_2_PB	32.32	−0.46	0.00	0.48	600
BN_2_PM	99.46	−3.05	−66.20	−65.34	481
BN_6_BM	64.52	−1.29	−1.03	0.27	600
BN_6_PB	29.90	−0.90	0.00	0.95	578
BN_6_PM	98.50	−9.67	−67.98	−64.42	377
CH_2_BM	70.54	−0.68	0.00	0.70	600
CH_2_PB	30.71	−0.84	0.00	0.87	600
CH_2_PM	99.52	−3.16	−67.76	−66.84	482
CH_6_BM	76.17	−1.85	−1.33	0.56	582
CH_6_PB	32.38	−1.21	0.00	1.29	572
CH_6_PM	99.10	−6.64	−67.35	−65.36	390
MX_2_BM	63.01	−0.42	0.00	0.43	600
MX_2_PB	35.28	−0.58	0.00	0.60	600
MX_2_PM	99.22	−3.76	−71.36	−70.48	476
MX_6_BM	68.51	−1.54	−1.02	0.55	587
MX_6_PB	27.85	−1.14	−0.01	1.20	583
MX_6_PM	99.13	−10.33	−67.48	−63.05	383
MX_9_BM	10.55	−1.62	−0.09	2.38	305
MX_9_PB	0.26	−0.81	0.00	0.97	301
MX_9_PM	92.28	−11.31	−57.56	−52.57	333

case no initial feasible solution is available to start CG, an alternative is to add artificial variables to (11) and (12), with sufficiently high negative coefficients in the objective function.

Table 9 provides the average results of CF and CFCG in terms of upper bounds and computational times for each group of instances. Column 'Group' indicates the label of each group. We let UB_{CF} and UB_{CFCG} denote the upper bound values provided by CF and CFCG, respectively.

'CF GAP' presents the average gap $(UB_{CF} - LB_{CF})/UB_{CF}$ provided by CF. As one can notice, the most difficult instances are those with 'peak in the middle' time distributions (groups whose labels finish by 'PM'). The columns 'CF UB Improv' and 'CG UB Improv' present how CF and CG improve over UB_1, the value obtained in the first stage of CFCG. Those improvements are calculated as $(UB_X - UB_1)/UB_1$, where X corresponds to CF in the first case and CFCG in the latter. Remark that $UB_X \leq UB_1 \leq UB_{BS} \leq UB_0$ since the bisection method works as a pre-processing stage for CF and CFCG, as mentioned in Section 4. From columns 'CF UB Improv' and 'CG UB Improv', one can conclude that, for the hardest instances (with 'peak in the middle' time distributions), CFCG provides far better upper bounds than CF. This is confirmed by column 'DEV UB', which shows the average upper bound deviations calculated as $(UB_{CFCG} - UB_{CF})/UB_{CFCG}$. Note that negative values indicate an improvement of CFCG over CF. From these results, it can be seen that, when dealing with the hardest instances, CFCG outperforms CF in terms of upper-bound quality. For the other groups, the deviation does not exceed 2.38% on average (group 'MX_9_BM').

Fig. 3 provides more details about the performance of CF and CFCG. The values corresponding to columns 'CF GAP' and 'DEV UB' of Table 9 are plotted individually for each instance. Hence, it is worth noting that those are average values taking into account the results for the three different levels of uncertainty. The black bars at the bottom are related to the vertical axis on the right side of the figure, while the gray lines correspond to the vertical axis on the left. As shown by the black bars, CFCG improvements over CF reached up to 100% for PM groups. Recall that these improvements are indicated by negative values, according to the definition of 'DEV UB' above.

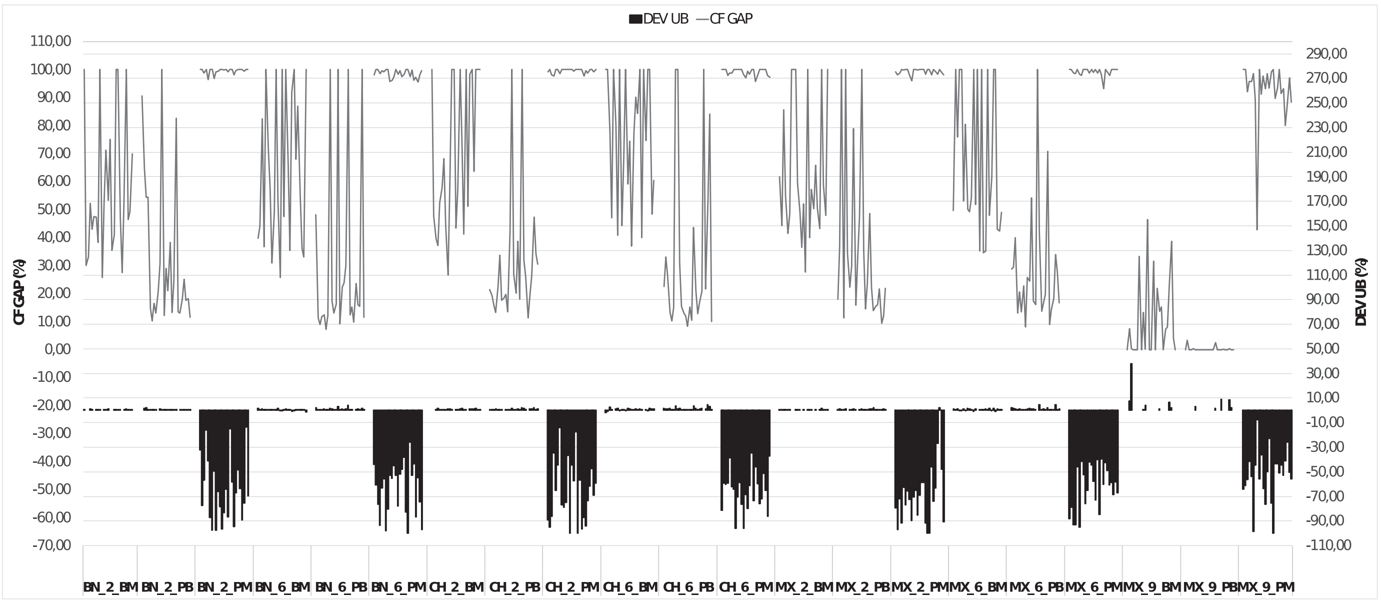

Fig. 3. Joint chart of upper bound deviations and optimality gaps.

Still in Fig. 3, when CF returns better bounds (positive-valued black bars), the deviation from UB_{CFCG} is close to zero, except for groups 'MX_9_BM' and 'MX_9_PB', where UB_{CFCG} is worse than UB_{CF} by a more significant amount. In those groups, UB_{CF} is around 40% better than UB_{CFCG} for one instance, but the deviations remain under 10% for all the others. Remark that 'MX_9_BM' and 'MX_9_PB' correspond to the lowest gaps, determined by the gray lines in the right part of Fig. 3. Indeed, the relation between the gray lines and black bars in Fig. 3 indicates that Dantzig-Wolfe decomposition is less efficient in cases where CF provides relatively small gaps. Although, one can note that for several instances, 'CF GAP' is 100%, and the proposed approach could not provide any improvement. In those cases, the best-known solution value is zero, leading to a 100% gap ever since the upper bound is not optimal. Indeed, for those cases, CF already manages to provide near-optimal upper bounds despite the high-valued gap.

A total of 144 instances out of 1575 (525 times three uncertainty levels) have been solved to optimality. Most of them belong to groups 'MX_9_BM' and 'MX_9_PB' and are solved by CF within 600 seconds. However, the optimality of the solution to 26 instances out of the 144 ones (18%) has been proven by the proposed approach, *i.e.*, CG returns an upper bound whose value is equal to the best-known solution value. This is typically the case when the best-known solution value is zero.

In terms of computational time, column 'CFCG Time' in Table 9 presents the average computational time, in seconds, spent by CFCG. For CF, the computational time is always equal to the limit of 600 seconds, except for the 118 instances solved to optimality as mentioned before. It can be observed that the column generation algorithm may converge relatively fast in some cases, especially for high levels of OS (MX_9_BM, MX_9_PM, and MX_9_PB). Since the first 300 seconds are spent during the first stage of CFCG, column generation converges in approximately 5 seconds for MX_9_BM, 1 second for MX_9_PB, and 33 seconds for MX_9_PM on average.

The very large-scale instances of 1000 tasks from Otto et al. (2013) were also used for computational experiments, but both CF and CFCG failed on them. Indeed, CPLEX could not provide any feasible solution after 1 hour of computational time, and for hard instances, even the linear relaxation of the compact formulation (1)–(9) could not be solved after 1 hour of computational time. In its turn, the bisection method remained effective for those instances, managing to decrease upper bounds by 80% in some cases. In terms of computational time, the method stopped within a few seconds for all instances. These results are reported as averages in Appendix A. The values for each instance are available at the web link mentioned earlier.

7.5. Heuristic improvement of initial solutions

When column generation converges before the time limit, an attempt to improve the lower bound is made, taking advantage of the generated columns. In this manner, integrality is reimposed to λ variables, and the current integer master problem is solved using CPLEX. A time limit is set so that CFCG cannot run for more than 600 seconds.

According to the obtained results, such a heuristic procedure provides better solutions in some of the cases where it could be executed, namely 12 out of 674 instances (2%). Concerning the computational time, this heuristic procedure is relatively fast, taking less than two minutes to run. These results are also available on the online material.

7.6. Managerial insights

We now present computational results aiming to study how the stability radius changes once some of the instance's features are modified, namely the number of workstations, the cycle time, and the positioning of uncertain workstations. Considering the best-known feasible solution (LB) and the best-known upper bound (UB) for each instance, we provide charts on the evolution of those values as the features are changed. The optimality gap (GAP), computed as $(UB - LB)/UB \times 100$, is also provided to indicate how the difficulty of instances evolves according to the modifications. All those values represent averages on the whole set of instances.

7.6.1. Impact of the position of uncertain workstations along the assembly line

In this section, we study the impact of the position of uncertain workstations along the assembly line. More precisely, in the second level of uncertainty, we have $|\widetilde{V}^1| = \lceil 0.5n \rceil$ and $|\widehat{W}| = \lceil 0.5m \rceil$, and

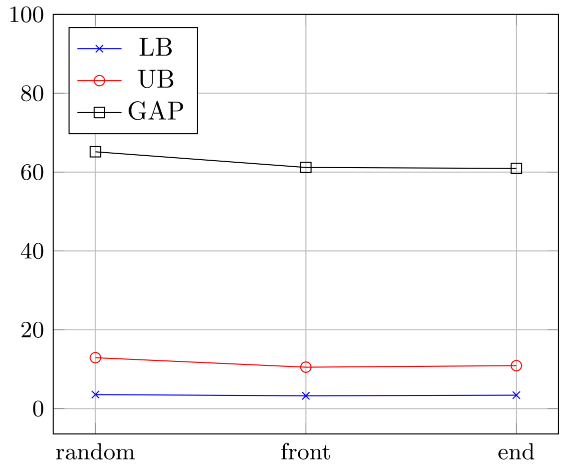

Fig. 4. Average results for different workstation distributions.

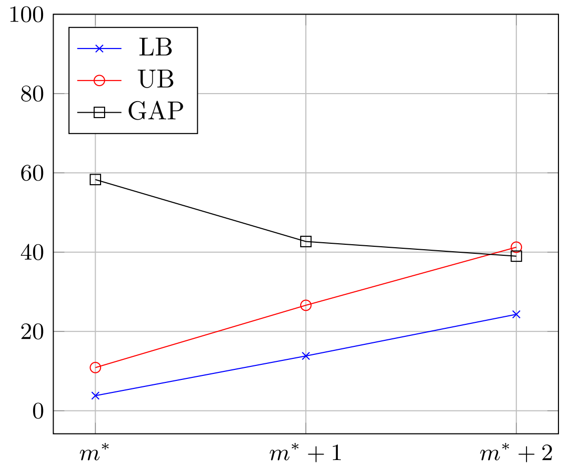

Fig. 5. Average results for the increased number of workstations.

the uncertain workstations are selected randomly. We now consider two additional scenarios: *front* refers to the situation, where the first $|\widehat{W}|$ workstations are uncertain, and *end* corresponds to the situation, where the last $|\widehat{W}|$ workstations are uncertain. Fig. 4 shows the lower and upper bounds on the stability radius, as well as the gap under these three scenarios. It can be seen that the actual position of uncertain workstations in the assembly line does not have a noticeable impact on the stability radius value. The random scenario is slightly more challenging, as its gap and the upper bound are a bit larger than in the other two scenarios. In a managerial context, this means that if we are given a way to select the uncertain tasks, then it may be slightly more beneficial to select them as the first or the last tasks in the precedence graph.

7.6.2. Increasing the number of workstations, and the cycle time

We now investigate the impact of having additional workstations in the line. We denote by $m^\star$ the situation where the number of workstations is minimum, *i.e.*, is equal to the optimal objective value of SALBP-1 when the cycle time is $C = 1000$ units of time. Then, $m^\star + 1$ and $m^\star + 2$ represent the situations, where respectively one and two new (certain) workstations are added to the line. Fig. 5 suggests that the lower bound increases by an average of 1% of the cycle time per additional workstation, and that the upper bound increases by an average of 1.5% of the cycle time per additional workstation. The gap decreases significantly when the number of workstations is at least $m^\star + 1$, which shows that the problem gets easier when the number of workstations increases by one or two units.

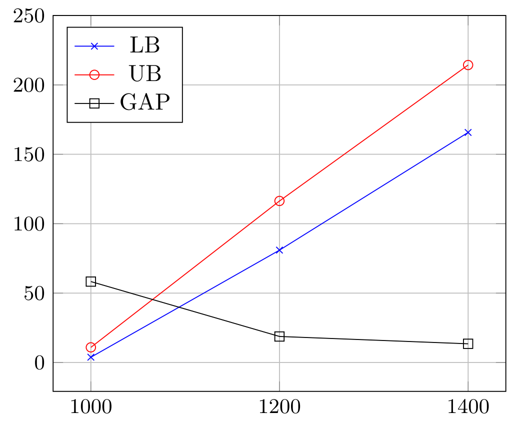

Fig. 6. Average results for increased cycle time.

Next, we consider the effects on the stability radius value in the face of an increase in cycle time, when the number of workstations is equal to the optimal objective value of SALBP-1. In Fig. 6, the cycle time C is set to 1000, 1200, and 1400 units of time. As expected, the lower and upper bounds on the stability radius both increase when C increases. The figure also suggests that when C increases, the lower bound increases by 40% of the extra time, and the upper bound increases by 50% of the extra time. The gap decreases somewhat faster than when the number of workstations increases, which can be explained by the fact that varying the cycle time does not change the number of variables in the extended formulation, whereas the number of decision variables increases by the number of patterns associated with the added workstation. Moreover, additional pricing sub-problems must be solved.

Remark that adding a new workstation leads to an increase of C units of time to the production capacity, whereas setting C to 1200 increases the production capacity by $200 \times \overline{m^\star}$ units of time, where $\overline{m^\star}$ is the average number of workstations (approximately 30 for the considered set of instances). Hence, setting the cycle time value to 1200 brings around six times more production capacity than adding a new workstation. Taking the stability radius' increase per unit of time added, we can see that, in terms of lower bound, adding one workstation gives an increase in the stability radius of 0.01 per unit of time, and adding 200 units of time to C gives an increase of 0.013 per unit of time. This can help the assembly line manager who considers adding a new workstation or extending the cycle time in an attempt to increase the stability radius: increasing the cycle time rather than the number of workstations might lead to a more significant increase in the stability radius. Moreover, adding a workstation may not impact the stability radius value at all, while increasing the cycle time always has a positive impact, as shown in Properties 2 and 3. More specifically, Property 2 shows that adding a new workstation is vain when the stability radius value is set by a workstation that processes a unique uncertain task, whereas Property 3 shows that increasing the cycle time always leads to improving the stability radius value. By contrast, increasing the cycle time has a negative effect on the assembly line throughput, and the line's productivity is often an important parameter for achieving economic sustainability. This suggests that if the stability radius of the current solution to SALBP-S is found to be insufficient, a trade-off between the stability radius value and the assembly line throughput should be found so that the best decision is made.

8. Conclusion

This paper proposes a Dantzig-Wolfe decomposition approach for the maximization of the stability radius of a simple assembly line under task processing time uncertainty. The main goal is to provide a more effective bounding procedure. In that sense, valid inequalities are also proposed to enhance the extended formulation given by the Dantzig-Wolfe decomposition. A comparison between several versions of the Dantzig-Wolfe approach is made, and the results show that the valid inequalities are quite effective in improving upper bounds. The results also indicate that it might be preferable to include precedence relations in the pricing subproblem rather than in the master problem.

Two pre-processing techniques are used to help improve performance. We show that using a 'single-machine scheduling problem' to represent the line is an efficient way to produce shorter assignment intervals, reducing the number of binary assignment variables and enhancing the algorithms' performance. A second pre-processing technique consists of a bisection method, which highly improves the existing upper bounds in less than one second of computational time. Using those improved values as an input, the Dantzig-Wolfe decomposition approach is compared to CPLEX on the compact MILP formulation of the problem. On average, the proposed approach provides better upper bounds for the most challenging instances. In those cases, improvements reached 70%, while for the less difficult instances, even though improvements in upper bounds are not achieved, the deviations to the best upper bound remain under 3%. For the cases where the optimality gap is small, the column generation converges fast. Moreover, in a few cases, better feasible solutions can be obtained in a heuristic way using the columns generated in the proposed approach.

Computational experiments also show the limitation of all the approaches based on integer linear programming: when the number of tasks gets very large (typically, when there are more than 1000 tasks), solving the linear programming relaxation of the compact model is too time-consuming. However, upper bounds on the stability radius can still be obtained very quickly with the bisection method. Analogously, existing heuristics can quickly produce lower bounds without resorting to linear programming. Consequently, depending on the size of the instance, the decision-maker can use the results proposed in this paper to assess the robustness level that can be achieved. If the stability radius value is too low to offer adequate protection against uncertainty, the proposed methods can serve as a decision-aiding tool. For example, the effects of adding a new workstation or increasing the cycle time can be evaluated quantitatively, helping to choose among alternative solutions.

Finally, future research involves applying the Dantzig-Wolfe decomposition into a branch-and-price scheme, taking advantage of better upper bounds and better heuristic solutions. That might help reduce the size of the enumeration tree, thus accelerating the algorithm. Moreover, a theoretical study on why some groups of instances, notably those with a 'peak in the middle' processing time distribution, are harder than others is a topic that needs to be fulfilled.

Appendix A. Results for very large-scale instances, with 1000 tasks

In this appendix, we present the results obtained for the very large-scale instances, with 1000 tasks. We use the same notations as in Table 2 of the main paper. As mentioned in the main text, both CF and CFCG failed in providing upper bounds for the SALBP-S. For 87.4% of the instances, even the linear relaxation of the compact formulation (1)–(9) could not be solved after 1 hour of computational time. The other instances are mainly 'peak at the bottom' ones, which are easier according to Otto et al. (2013).

Table A.1
Average reductions provided by the bisection method in the initial upper bound, for very large-scale instances.

Group	PA		NAI	
	BS UB Improv (%)	Time (s)	BS UB Improv (%)	Time (s)
BN_2_PB	0.00	0.01	0.00	0.09
BN_2_PM	0.00	0.02	0.00	0.15
BN_2_BM	0.00	0.01	0.00	0.09
BN_6_PB	−8.93	0.02	−13.86	0.33
BN_6_PM	−4.89	0.03	−7.56	0.97
BN_6_BM	0.00	0.02	0.00	0.17
CH_2_PB	0.00	0.02	0.00	0.11
CH_2_PM	0.00	0.03	0.00	0.17
CH_2_BM	0.00	0.02	0.00	0.09
CH_6_PB	−4.55	0.02	−12.42	0.50
CH_6_PM	−3.61	0.03	−10.60	1.28
CH_6_BM	0.00	0.02	0.00	0.19
MX_2_PB	0.00	0.02	0.00	0.11
MX_2_PM	0.00	0.03	0.00	0.17
MX_2_BM	0.00	0.02	0.00	0.09
MX_6_PB	−4.67	0.02	−9.08	0.35
MX_6_PM	−1.73	0.03	−5.19	1.11
MX_6_BM	0.00	0.02	0.00	0.19
MX_9_PB	−45.97	0.02	−57.63	3.18
MX_9_PM	−13.18	0.03	−14.27	2.74
MX_9_BM	−38.34	0.02	−53.33	3.67
Overall average	**−5.99**	**0.02**	**−8.76**	**0.75**

Nevertheless, the computational times for these instances remain high (solving the linear relaxation of the compact formulation took about 40 minutes on average).

For all instances, since no basic feasible solution was found for the extended formulation, the Dantzig-Wolfe approach could not be run. However, the bisection method still manages to improve the initial upper bounds of the most difficult instances.

Following the same notation as in Table 8 of the main text (which is built for instances with 100 tasks), Table A.1 presents the results with the bisection method on instances with 1000 tasks. In this case, the results were less significant, but the bisection method still improves over the initial upper bound on the classes with a dense precedence graph (OS = 0.6 and OS = 0.9), especially on groups of type 'peak in the middle' and 'peak at the bottom'. As expected, the running time of the bisection method increases with the number of tasks but is still very low. The improvement in upper bound quality, when NAI is used, is around 46% greater compared to the use of PA, which is similar to what was observed with instances of 100 tasks.

References

Arkhipov, D., Battaïa, O., & Lazarev, A. (2019). An efficient pseudo-polynomial algorithm for finding a lower bound on the makespan for the resource constrained project scheduling problem. *European Journal of Operational Research, 275*(1), 35–44.
Ağpak, K., & Gökçen, H. (2007). A chance-constrained approach to stochastic line balancing problem. *European Journal of Operational Research, 180*(3), 1098–1115.
Battaïa, O., & Dolgui, A. (2013). A taxonomy of line balancing problems and their solution approaches. *International Journal of Production Economics, 142*(2), 259–277.
Bautista, J., & Pereira, J. (2011). Procedures for the time and space constrained assembly line balancing problems. *European Journal of Operational Research, 212*(3), 473–481.
Boysen, N., Schulze, P., & Scholl, A. (2022). Assembly line balancing: What happened in the last fifteen years? *European Journal of Operational Research, 301*(3), 797–814.
Brucker, P. (2007). *Scheduling algorithms*. Springer Berlin Heidelberg.
Dolgui, A., & Kovalev, S. (2012). Scenario based robust line balancing: Computational complexity. *Discrete Applied Mathematics, 160*(13-14), 1955–1963.
Fleszar, K., & Hindi, S. H. (2003). An enumerative heuristic and reduction methods for the assembly line balancing problem. *European Journal of Operational Research, 145*(3), 606–620.
Gen, M., Tsujimura, Y., & Li, Y. (1996). Fuzzy assembly line balancing using genetic algorithms. *Computers & Industrial Engineering, 31*(3-4), 631–634.

Geoffrion, A. M. (1974). Lagrangean relaxation for integer programming. *Approaches to Integer Programming. Vol. 2 of Mathematical Programming Studies* (pp. 82–114). Springer Berlin Heidelberg.

Gurevsky, E., Battaïa, O., & Dolgui, A. (2012). Balancing of simple assembly lines under variations of task processing times. *Annals of Operations Research, 201*(1), 265–286.

Gurevsky, E., Battaïa, O., & Dolgui, A. (2013). Stability measure for a generalized assembly line balancing problem. *Discrete Applied Mathematics, 161*(3), 377–394.

Gurevsky, E., Hazır, O., Battaïa, O., & Dolgui, A. (2013b). Robust balancing of straight assembly lines with interval task times. *Journal of the Operational Research Society, 64*(1), 1607–1613.

Hazır, O., & Dolgui, A. (2013). Assembly line balancing under uncertainty: Robust optimization models and exact solution methods. *Computers & Industrial Engineering, 65*(2), 261–267.

Hop, N. (2006). A heuristic solution for fuzzy mixed-model line balancing problem. *European Journal of Operational Research, 168*(3), 798–810.

Klein, R., & Scholl, A. (1988). Computing lower bounds by destructive improvement: An application to resource-constrained project scheduling. *European Journal of Operational Research, 112*(2), 322–346.

Lai, T.-C., Sotskov, Y. N., & Dolgui, A. (2019). The stability radius of an optimal line balance with maximum efficiency for a simple assembly line. *European Journal of Operational Research, 274*(2), 466–481.

Lai, T.-C., Sotskov, Y. N., Dolgui, A., & Zatsiupa, A. (2016). Stability radii of optimal assembly line balances with a fixed workstation set. *International Journal of Production Economics, 182*, 356–371.

Morrison, D., Sewell, E., & Jacobson, S. (2014). An application of the branch, bound, and remember algorithm to a new simple assembly line balancing dataset. *European Journal of Operational Research, 236*(2), 403–409.

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.

Özcan, U. (2018). Balancing stochastic parallel assembly lines. *Computers & Operations Research, 99*, 109–122.

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

Peeters, M., & Degraeve, Z. (2006). An linear programming based lower bound for the simple assembly line balancing problem. *European Journal of Operational Research, 168*(3), 716–731.

Pereira, J. (2015). Empirical evaluation of lower bounding methods for the simple assembly line balancing problem. *International Journal of Production Research, 53*(11), 3327–3340.

Pereira, J., & Álvarez-Miranda, E. (2018). An exact approach for the robust assembly line balancing problem. *Omega, 78*, 85–98.

Pirogov, A., Gurevsky, E., Rossi, A., & Dolgui, A. (2021). Robust balancing of transfer lines with blocks of uncertain parallel tasks under fixed cycle time and space restrictions. *European Journal of Operational Research, 290*(3), 946–955.

Ritt, M., & Costa, A. M. (2018). Improved integer programming models for simple assembly line balancing and related problems. *International Transactions in Operations Research, 25*(4), 1345–1359.

Rossi, A., Gurevsky, E., Battaïa, O., & Dolgui, A. (2016). Maximizing the robustness for simple assembly lines with fixed cycle time and limited number of workstations. *Discrete Applied Mathematics, 208*, 123–136.

Schepler, X., Rossi, A., Gurevsky, E., & Dolgui, A. (2022). Solving robust bin-packing problems with a branch-and-price approach. *European Journal of Operational Research, 297*(3), 831–843.

Scholl, A. (1999). *Balancing and sequencing of assembly lines* (2nd ed.). Physica-Verlag Heidelberg.

Scholl, A., & Becker, C. (2006). State-of-the-art exact and heuristic solution procedures for simple assembly line balancing. *European Journal of Operational Research, 168*(3), 666–693.

Sotskov, Y. N., Dolgui, A., Lai, T.-C., & Zatsiupa, A. (2015). Enumerations and stability analysis of feasible and optimal line balances for simple assembly lines. *Computers & Industrial Engineering, 90*, 241–258.

Sotskov, Y. N., Dolgui, A., & Portmann, M. C. (2006). Stability analysis of an optimal balance for an assembly line with fixed cycle time. *European Journal of Operational Research, 168*(3), 783–797.

Sotskov, Y. N., Dolgui, A., Sotskova, N., & Werner, F. (2005). Stability of optimal line balance with given station set. In A. Dolgui, J. Soldek, & O. Zaikin (Eds.), *Supply chain optimisation: Product/process design, facility location and flow control* (pp. 135–149). Springer US. Volume 94 of Applied Optimization. chapter 10

Tsujimura, Y., Gen, M., & Kubota, E. (1995). Solving fuzzy assembly-line balancing problem with genetic algorithms. *Computers & Industrial Engineering, 29*(1–4), 543–547.

Wolsey, L. (1998). *Integer programming*. John Wiley & Sons.

Zacharia, P., & Nearchou, A. (2013). A meta-heuristic algorithm for the fuzzy assembly line balancing type-E problem. *Computers & Operations Research, 40*(12), 3033–3044.