Omega 78 (2018) 85–98

Contents lists available at ScienceDirect

Omega

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

An exact approach for the robust assembly line balancing problem

Jordi Pereira [a,*], Eduardo Álvarez-Miranda [b]

[a] Faculty of Engineering and Sciences, Universidad Adolfo Ibáñez, Av. Pedro Hurtado 750, Office A-215, Viña del Mar, Chile
[b] Department of Industrial Engineering, Universidad de Talca, Merced 437, Universidad de Talca, Curicó, Chile

ARTICLE INFO

Article history:
Received 31 December 2016
Accepted 23 August 2017
Available online 28 September 2017

Keywords:
Line balancing
Robust optimization
Lower bounds
Branch-and-bound

ABSTRACT

This work studies an assembly line balancing problem with uncertainty on the task times. In order to deal with the uncertainty, a robust formulation to handle changes in the operation times is put forward. In order to solve the problem, several lower bounds, dominance rules and an enumeration procedure are proposed. These methods are tested in a computational experiment using different instances derived from the literature and then compared to similar previous approaches. The results of the experiment show that the method is able to solve larger instances in shorter running times. Furthermore, the cost of protecting a solution against uncertainty is also investigated. The results highlight that protecting an assembly line against moderate levels of uncertainty can be achieved at the expense of small quantities of additional resources (stations).

1. Introduction and Motivation

An assembly line is a production system frequently used for the mass production of standardized commodity goods. In an assembly line, unfinished products flow through consecutive workstations, where different tasks are performed on the product until it is finished and then the finished product departs the line. The assembly line balancing problem, ALBP, comprises a broad family of problems that study the optimal assignment of tasks to the workstations in order to maximize some efficiency criterion.

Among the different ALBPs, the most thoroughly studied problem in the literature is the simple assembly balancing problem, SALBP, which constitutes a basic common formulation among line balancing problems. The assumptions of the SALBP are [34]: (1) the line assembles a single product; (2) the production process is known in advance; (3) there is an identical amount of time, the cycle time c, allotted to the worker of each of the m stations to perform the station's tasks; (4) operation times of the tasks are deterministic; (5) the assignment constraints correspond only to technological limitations that force some tasks to be performed before others; (6) the workstations are serially arranged along the line; (7) all workstations are equally equipped; and (8) the objective is to optimize the line's efficiency. In this regard, line efficiency can be achieved in two ways: (1) by minimizing m for a given c, which constitutes a type-1 problem, or (2) by minimizing m for a given c, which is a type-2 problem.

Other ALBP models that differ from the SALBP are referred to as general assembly line balancing problems, GALBPs, in the literature. While the SALBP is an over-simplistic model of the real problem faced in industrial settings, it offers a basic framework to build more realistic models, to compare different solution methods, as well as for providing a pool of common ideas which can be used for different GALBPs. Therefore, it is usual for GALPBs to build upon the SALBP formulation by altering some of its assumptions.

In this work we build upon the SALBP by modifying assumption (4). The problem considers operation times to be uncertain but to take values from known intervals. The problem, which will be referred to as the type-1 robust SALBP, rSALBP-1, aims at finding an assignment of tasks to stations such that feasibility is ensured even when some operation times adversely affect the solution, that is, being robust under different possible realizations of the uncertain operation times. The rSALBP-1 follows the Bertsimas and Sim, B&S, robustness approach, see [6], in which a solution is said to be robust if it is feasible after a predefined number of adverse changes.

Sources of uncertainty on the operation times abound. Among the possible causes, the most frequent are the particularities of manual operations, the heterogeneity of the operators (some operators are more proficient than others), slight differences between assembled components, and random events. Considering operation time variability during line balancing (as in the B&S) is likely to reduce reprocessing work, to prevent work-overloads and to minimize disruptions to the line, see [11].

Another common situation which can be partially modeled by the introduction of uncertainty is mixed model assembly line balancing. When balancing mixed model assembly lines, it is usual to

* Corresponding author.
E-mail addresses: jorge.pereira@uai.cl, jorgepereiragude@gmail.com (J. Pereira).

https://doi.org/10.1016/j.omega.2017.08.020
0305-0483/

aggregate the models into a "vanilla" product (by averaging operation times according to the production plans) and then to balance the line as if a single model had to be assembled. In such a case, operation times vary from unit to unit during assembly and this variability can be represented using uncertainty in the operation times.

Eventually, considering uncertainty should lead to reducing the amount of unfinished work in the workstations. Unfinished work mostly arises when the time allotted to a station does not suffice to finish its assigned tasks due to unexpected events, and it may cause the stoppage of the whole assembly line and/or force to reprocess operations. As both solutions are expensive, a common practice in industrial settings is to use above-expected operation times in order to provide some slack to the stations. Thus, a B&S formulation offers a formal alternative to this practice and provides a more detailed method to control the amount of protection time that we allocate to each station.

1.1. Literature review

It is out of the scope of this work to make an exhaustive review of the robust optimization or the ALBP literature. Interested readers are referred to [4] for an introductory text, and to [7,15] for some reviews on the recent applications and developments on robust optimization. Likewise, The reader is referred to [16,34] for earlier reviews or to [11] for a detailed review of current research on assembly line balancing.

As for the ALBP, the state-of-the-art methods for the SALBP and some GALBP formulations are enumerative. Both exact [14,26,36] and truncated [29,38] implementations of the branch-and-bound algorithm, as well as truncated dynamic programming based approaches [3], outperform other available methods for the SALBP and have been successfully applied to some GALBP [10,35,38,41]. The quality of these enumeration based methods is credited to the availability of tight lower bounds [31] and dominance rules [34,36,40].

While there is a growing body of literature considering robustness within line balancing design, the literature is still limited when compared to other related ALBPs, like the stochastic assembly line balancing problem [see 2, [8,17,33], 39, among others].

An early approach to consider robustness on the solutions for the ALBP was proposed in [37]. This approach is based on the concept of the stability of a solution, and tries to find the maximum operation time change such that the solution remains optimal. In [37], the authors introduce this concept within the SALBP-1. Another approach based on the stability of the solution can be found in [18]. The authors investigate the SALBP-E (a version of the SALBP in which efficiency is maximized subject to constraints on the cycle time c and the number of workstations m) and propose a heuristic method to obtain Pareto-optimal solutions in terms of both the stability and the idle time.

Another common method to consider robustness and to represent uncertainty is the use of scenarios (specific realizations of the operation times). In [12], a SALBP-2 formulation is used as starting point, and the authors analyze the complexity of several special cases of the SALBP with multiple scenarios.

Previous work following the B&S or resembling robustness approaches are available in the literature. In [19], the authors consider a variation of the B&S approach in which a quota of the tasks assigned to each workstation is allowed to change from its base (nominal) value. The problem differs from the B&S approach in the number of uncertain parameters that are allowed to change; the traditional B&S approach imposes a limit on the number of parameters that may simultaneously deviate from their nominal values, whereas [19] aims at finding a solution that remains feasible when the number of tasks that simultaneously deviate from their nominal values is a percentage of the number of tasks assigned to each station. In order to solve the problem, the authors propose an enumeration method. In [20], the authors tackle the robust SALBP-2 using the B&S approach, and a solution method based on a decomposition approach is put forward. The same decomposition approach is extended in [21] for the U-shaped assembly line under the same objective function (minimization of c for a given m). The authors test the method using instances derived from the classical SALBP instance set [34]. In both cases, the results show that the robust counterpart of the SALBP is significantly harder than the original problem.

Another line balancing problem in which B&S robustness considerations have been incorporated is line balancing with heterogeneous workers, see [25]. In [25], two previous formulations of worker heterogeneity are modified in order to take task time uncertainty into account, and the authors propose a novel heuristic method to solve these formulations.

1.2. Contributions of this work

In this work the B&S robustness approach for the SALBP-1 is considered, which we denote as rSALBP-1. The proposed B&S-based model slightly departs from the deterministic model by considering independent risks for each task. Moreover, it provides a single parameter (the number of values that may jointly deviate from their common value) that the decision-maker can use both to control the desired level of conservatism, as well as to investigate the effect of uncertainty in the resulting solution. When compared to other robust-based methods used in the literature, the proposed approach requires fewer data to model uncertainty than scenario-based robust methods; focuses on the feasibility of a solution under uncertainty (whereas other methods investigate the optimality of a given solution under possible changes) and avoids the use of statistical distributions, which may be difficult to ascertain in some settings. Consequently, it provides a formulation that is easy to understand by practitioners.

The present study formulates the rSALBP-1 and extends SALBP results to incorporate B&S robustness in the model. Among the proposed methods, this work offers different adaptations of the bin packing lower bounds [34], a modified maximum flow lower bound [40], a Dantzig–Wolfe reformulation based lower bound [30] and several dominance rules [32,34,36]. The proposed algorithm is able to solve instances of moderate size within reduced running times, outperforming alternative approaches [20]. Moreover, some of the proposed adaptations are applicable to other line balancing problems. This work also reports the first successful application, to the author's knowledge, of a traditional branch-and-bound approach to tackle a B&S robust problem. While robust optimization problems are usually solved using cutting planes based methods, the special characteristics of the problem under study, which is more amenable to enumeration, as well as the quality of the proposed bounding procedures and dominance rules lead to an effective procedure, which outperforms previous ones [5,20].

1.3. Paper outline

The remaining of the paper is structured as follows. Section 2 introduces the rSALBP-1, provides a mathematical formulation and considers its relationship with the robust knapsack problem. Section 3 is devoted to the exposition of several lower and upper bounds for the problem. The branch-and-bound procedure is described in Section 4, and the results of the method are analyzed in Section 5. Finally, Section 6 puts forward some conclusions and points to possible future research directions.

2. The type-1, robust simple assembly line balancing problem, rSALBP-1

2.1. Mathematical formulation

Given a set V, $i \in V$, of indivisible tasks, each with an operation time t_i, a set of predecessor tasks P_i ,$P_i \subset V$, and an ordered set M, $j \in M$, of stations with identical allotted time c, the type-1 simple assembly line balancing problem, SALBP-1, looks for an assignment of tasks to stations such that the number of stations is minimized while satisfying time and precedence constraints.

The predecessors are often described using a directed acyclic graph [34] in which the nodes correspond to the tasks and the arcs to the precedence relations. Alternatively, the predecessors can be represented using sets of absolute precedence relations, P_i^*, obtained by transitivity from P_i, using sets of immediate successors, F_i, or sets of absolute successors, F_i^*. Precedence relations depict technological limitations on the order in which tasks can be performed (e.g., in a car, the seats must be assembled before the doors).

The time constraints ensure that the total work performed in a station does not exceed the allotted time. This value is known as the cycle time of the production system and defines the production rate (a unit is assembled each c time units).

These conditions can be formulated using an integer program, IP, in which the binary variables x_{ik} represent whether task i is assigned to station k, and binary variables y_k identify whether station k is used in the optimal solution. Using the previous notation, objective (1) and constraint sets (2)–(6) constitute a valid formulation for the SALBP-1.

$$z^* = \min \sum_{k \in M} y_k \tag{1}$$

$$\sum_{k \in M} x_{ik} = 1 \quad \forall i \in V \tag{2}$$

$$\sum_{i \in V} t_i \, x_{ik} \leq c \, y_k \quad \forall k \in M \tag{3}$$

$$\sum_{k \in M} k \, x_{jk} \leq \sum_{k \in M} k \, x_{ik} \quad \forall i \in V, \ \forall j \in P_i \tag{4}$$

$$x_{ik} \in \{0, 1\} \quad \forall i \in V, \ \forall k \in M \tag{5}$$

$$y_k \in \{0, 1\} \quad \forall k \in M \tag{6}$$

Objective (1) minimizes the number of stations. Constraint set (2) assigns each task to a station, while constraint set (3) enforces the cycle time constraints and ensures that whenever a task is assigned to a station the stations variable is set to 1. Constraint set (4) enforces precedence relations. Finally, constraint sets (5) and (6) define the domain of the variables. Note that if the sets of predecessors and successors of each task are interchanged, a new instance with the same optimal solution is obtained. This property is known as instance reversibility and extends to the robust problem discussed in the remainder of the paper.

Formulation (1)–(6) is *robustified* against deviations on the operation time of the tasks following the B&S robust approach. Accordingly, the operation time of each task has a nominal value t_i, which corresponds to its common value, but it is allowed to deviate from its nominal value and attain values from some known interval $[t_i^-, t_i^+]$. While each operation time can vary independently, we aim at finding a solution with a given protection level defined by a parameter Γ. A solution is said to be protected against uncertainty if the cycle time constraints (3) remain feasible when at most Γ operation times deviate from their nominal values.

This paper considers that Γ is a natural number, even if any non-negative real value could be used. This assumption simplifies the description of the models and methods and it has been previously adopted in the literature [see 1, 24, among others]. Nonetheless, the methods proposed hereafter could be adapted to the general case after proper modifications. Also note that while variability can reduce the nominal values of the operation times of certain tasks, smaller operation times do not affect the feasibility of the cycle time constraints. Moreover, if an operation time is increased, the most stringent condition appears when the operation time takes its upper bound value. Consequently, we focus our attention on the upper bounds of the intervals, and define the difference between the upper and the nominal operation time values as the protection time of the task, $\bar{t}_i = t_i^+ - t_i$; that is, the amount of time required to protect a station against a particular task uncertainty.

Please note that from a practical standpoint, each of the concepts has an intuitive interpretation. For instance, the nominal value corresponds to the expected operation time, usually available from time-measurements. The protection time accounts for common deviations on the nominal value which are known to exist, and the protection level Γ limits the number of common deviations that we want the station to be protected from.

Consequently, the Type-1 robust simple assembly line balancing problem, rSALBP-1, enhances the SALBP-1 by considering a *robustified* version of constraint set (3), which allows a number Γ of tasks to deviate from their nominal values.

The proposed rSALBP-1 formulation, as well as several of the lower bounds described in Section 3, requires the results of Proposition 1, which we provide below.

Proposition 1. *A* robustified *version of constraint set (3) requires protecting each constraint separately against any possible changes of* Γ *parameters in the said constraint.*

Proof. The B&S approach requires the solution to be protected against any possible change of Γ parameters. The extreme case corresponds to changes of Γ parameters in tasks assigned to a single station. Let $\hat{J}$ be the set of tasks that deviate from their nominal value, then for each station k, constraint set (7) with $|\hat{J}| = \Gamma$ must hold.

$$\sum_{i \in V} t_i x_{ik} + \sum_{i \in \hat{J}} \bar{t}_i x_{ik} \leq c y_k \qquad \forall \hat{J} \subseteq V : \ |\hat{J}| \leq \Gamma, \ k \in M \tag{7}$$

A station protected against Γ changes is also protected against $\Gamma' < \Gamma$ changes. Moreover, when the Γ changes affect different stations, each constraint is exposed to $\Gamma' < \Gamma$ changes. Consequently, protecting each constraint separately is a necessary and a sufficient condition to ensure B&S robustness. □

Given the results of Proposition 1, the rSALBP-1 can be formulated by substituting constraint set (3) for constraint set (7), which is combinatorial to Γ and the number of tasks.

Proposition 1 enables us to define incompatibility relations among tasks. Throughout the paper, two tasks i, i' are deemed incompatible if both tasks are assigned to the same station and constraint set (7) is not satisfied. This condition is equivalent to checking whether $t_i + t_{i'} + \max\{\bar{t}_i + \bar{t}_{i'}\} > c$ holds for $\Gamma = 1$, or $t_i + t_{i'} + \bar{t}_i + \bar{t}_{i'} > c$ if $\Gamma \geq 2$.

2.2. The robust knapsack problem with precedence constraints, rKPPC

The cycle time condition, constraint set (7), constitutes a special knapsack constraint. Even if the knapsack problem, KP, and some of its generalizations are computationally challenging, large-size instances can be solved within relatively short running times. Consequently, several solution procedures make use of the relationship between the KP and the SALBP [30,31,40].

In [30,31,40] a knapsack problem with precedence constraints, KPPC, is used to generate feasible station assignments that are required by different SALBP-1 through preprocessing and lower bounding procedures. Accordingly, the robust (B&S) KPPC described here is an essential component of the lower bounds described in Sections 3.2 and 3.3. The description follows [24], with the inclusion of precedence constraints [30,31]. The resulting problem is denoted by rKPPC, the robust knapsack problem with and models the conditions that feasible station assignments must comply.

In the rKPPC we are given a capacity limit c, a protection level Γ, and a set $T \subseteq V$ of tasks ($i \in T$), each with a profit p_i, a nominal positive weight t_i, and a protection weight $\bar{t}_i$. Each task i also has a set of absolute predecessors, P_i^*, and successors, F_i^*. The objective is to find a subset $J \subseteq T$ that maximizes the total profit while satisfying the capacity constraint under uncertainty, and precedence relations among selected tasks.

$$z^*_{rKPPC} = \max \sum_{i \in J} p_i \tag{8}$$

$$\sum_{i \in J} t_i + \sum_{i \in \hat{J}} \bar{t}_i \leq c \quad \forall \hat{J} \subseteq J, |\hat{J}| \leq \Gamma \tag{9}$$

$$(i, h \in J) \wedge (j \in F_i^* \cap P_h^*) \Rightarrow j \in J \quad \forall i, j, h \in T \tag{10}$$

$$J \subseteq T \tag{11}$$

Constraint (11) corresponds to the selection of a subset of tasks that constitute a station assignment according to some profit function, as expressed in objective (8). Constraint set (9) ensures cycle time feasibility against up to Γ changes, and constraint set (10) forces the selection of tasks with both their predecessors and successors within the set of selected tasks. The description of the profit is deferred to the sections in which the resolution of the problem is required, see Sections 3.2 and 3.3, as different definitions of it are used.

Whenever a rKPPC instance is to be solved, a branch-and-bound based method that implicitly takes into account precedence constraints as well as protection weights is used. The interested reader is referred to previous work [30,31] for details on the implementation.

3. Lower and upper bounds for the robust simple assembly line balancing problem

This section describes the lower and upper bounds used by the proposed solution method. The lower bounds are derived from the relationship of the problem with the SALBP, and include provisions to account for the protection times of each station. The upper bound is an adaptation of a state-of-the-art truncated enumeration heuristic for the SALBP-1, the Hoffmann heuristic [38].

3.1. Bin packing based lower bounds

The classical set of bounds for the SALBP are the bin packing bounds [23], denoted by *LB*1, *LB*2, and *LB*3 in [34]. These bounds disregard precedence constraints and can be seen as special dual-feasible functions, [13]. Updated versions of these bounds (*rLB*1, *rLB*2, *rLB*3) that take into account protection times follow.

rLB1: *LB*1 estimates the number of stations when integrality constraints are relaxed. The bound is sometimes referred to as the "trivial" bound and corresponds to $LB1 = \lceil \sum_{i \in V} t_i / c \rceil$. For the rSALBP-1, this bound should be strengthened including an estimate of the amount of protection time in each station, that is, a bound on $\sum_{k \in M} \sum_{i \in J} \bar{t}_i x_{ik}$.

In what follows, let $\langle h_1, h_2 \ldots h_{|V|} \rangle$ be an ordering of the tasks according to non-decreasing protection times.

Proposition 2. *Eq. (12) constitutes a valid lower bound on the optimal number of stations for the rSALBP-1. The bound dominates LB*1

$$rLB1 = \left\lceil \frac{\sum_{i \in V} t_i + \sum_{i=1}^{LB1} \bar{t}_{h_i}}{c} \right\rceil \tag{12}$$

Proof. Straightforward. Each station 1,...,*LB*1 performs a task, and each station must be protected against the parameter change of a task, see Proposition 1. Consequently, the *LB*1 smallest protection times can be added to the total time required by the knapsack constraints to obtain a strengthened bound. □

This bound underestimates the total protection times in three different ways: (1) it fails to take into account that, whenever Eq. (12) improves *LB*1, the number of stations increases, thus increasing the number of tasks whose protection time should be included; (2) it considers the tasks with smaller protection times when the tasks with larger protection times also need to be assigned to a station; and (3) if $\Gamma > 1$, the bound only considers the protection time of a task per station yet some stations may contain more than a single task.

The first issue can be addressed by substituting (12) for (13), whose validity follows from the proof of Proposition 2. Eq. (13) evaluates the minimum number of stations, k, such that the sum of the task times plus the estimate of the protection time required by the k stations satisfies the trivial bound.

$$rLB1 = \min \left\{ k \in \mathbb{N} : \sum_{i \in V} t_i + \sum_{i=1}^{k} \bar{t}_{h_i} \leq k \cdot c \right\} \tag{13}$$

The second issue is caused by the lack of a method to properly identify tasks that cannot share stations among them, forcing the method to select the most optimistic set of protection times. Since the task with the largest protection time is assigned to a station and contributes to the total protection time of its station, Eq. (13) can be substituted for Eq. (14).

$$rLB1 = \min \left\{ k \in \mathbb{N} : \sum_{i \in V} t_i + \sum_{i=1}^{k-1} \bar{t}_{h_i} + \bar{t}_{h_{|V|}} \leq k \cdot c \right\} \tag{14}$$

Proposition 3 generalizes (14) through the use of incompatible relations.

Proposition 3. *Let γ be the largest value for which the set of tasks $\{h_{|V|-\gamma}, \ldots, h_{|V|}\}$ are pairwise incompatible. By definition, the set is composed of at least one task, $\{h_{|V|}\}$. Then, (15) constitutes a valid lower bound for the rSALBP-1.*

$$rLB1 = \min \left\{ k \in \mathbb{N} : \sum_{i \in V} t_i + \sum_{i=1}^{k-\gamma-1} \bar{t}_{h_i} + \sum_{i=|V|-\gamma}^{|V|} \bar{t}_{h_i} \leq k \cdot c \right\} \tag{15}$$

Proof. All the nominal task times and the protection time of a minimum of k tasks must be considered. As the $\gamma + 1$ tasks with highest protection time are incompatible, they cannot share a station among themselves and it is possible to consider their protection times in the calculation of their respective stations. The remaining $k - \gamma - 1$ protection times correspond to a lower estimation of the requirement in the remaining stations. □

Finally, the third issue does not affect *rLB*1 when $\Gamma = 1$, but needs to be addressed when $\Gamma \geq 2$. For $\Gamma = 2$, any station assignment with more than one task has to be protected against changes in two tasks. Consequently, ascertaining the minimum number of stations with solitary tasks, i.e. tasks that do not share a station

with another task, would improve the bound the protection time of more tasks would be considered. A method to derive a bound on the number of solitary stations follows.

Consider a feasibility version of the rSALBP-1 in which both the cycle time, c, and the number of stations, m, are known and the objective is to verify if a solution exists. Proposition (4) provides a necessary condition for the feasibility of a solution with s solitary tasks.

Proposition 4. *Let $\langle h'_1, h'_2 \ldots h'_{|V|}\rangle$ be an ordering of tasks according to non-decreasing task times. Then, inequality (16) must hold if a solution with m stations and s solitary tasks is to exist.*

$$\sum_{i=1}^{|V|-s} t_{h'_i} + \sum_{i=1}^{2\cdot(m-s)} \tilde{t}_{h_i} \leq (m-s)c \tag{16}$$

Proof. Suppose that there are no solitary tasks. Then $s = 0$, the right-hand side of the inequality provides the total amount of time available among all stations, and the left-hand side calculates the sum of task times and a lower bound on the protection time for the stations corresponding to the $2m$ smallest protection times.

If there is a single solitary task, $s = 1$, the task requires a station for itself and the remaining tasks are assigned to $m-1$ stations. A lower bound on the total task times of the remaining $|V|-1$ tasks and $m-1$ stations should consider $|V|-1$ task times and $2(m-1)$ protection times. In both cases the smallest values are considered and thus the amount removed from the left-hand side is an optimistic evaluation of the task time allotted to the station with a solitary task. The same logic applies to larger values of s. □

Proposition 4 leads to a destructive bound on the minimum number of stations based on increasing m until feasibility is achieved, see (17).

$$rLB1 = \min\left\{ m \in \mathbb{N} : \exists s \in \mathbb{N} \left| \sum_{i=1}^{|V|-s} t_i + \sum_{i=1}^{2\cdot(m-s)} \tilde{t}_{h_i} \leq (m-s)c \right. \right\} \tag{17}$$

For higher levels of conservatism, $\Gamma \geq 3$, Eq. (17) provides a lower bound that should be improved by extending the logic of Proposition 4. For instance, if $\Gamma = 3$, a necessary condition on the existence of a feasible solution with m stations, s_1 solitary tasks and s_2 paired tasks (those tasks that belong to stations with only two tasks assigned) could be derived. Since the complexity and computational requirements of the bound increase for larger Γ values, the branch-and-bound only makes use of (17) while relying on additional bounds, see Section 3.2, to capture cases in which $\Gamma \geq 3$.

rLB2: *rLB2* is a counting bound on the number of tasks that cannot share a station among them. For the SALBP-1, two sets J_1 and J_2 are identified. J_1 contains the set of tasks i with $t_i > c/2$, and set J_2 contains the tasks with $t_i = c/2$. Since the tasks in J_1 cannot be assigned to the same station as tasks in $J_1 \cup J_2$, and tasks in J_2 can only share a station with another task in J_2, $|J_1| + \lceil |J_2|/2 \rceil$ is a valid lower bound for the SALBP-1 and, by extension, for the rSALBP-1.

Alternative definitions for sets J_1 and J_2 considering protection times are put forward. For instance, consider an auxiliary graph in which there is a vertex for each task, and the set of edges is the set of incompatible relations among tasks. Any definition of J_1 corresponds to a clique in this auxiliary graph. As finding the maximum clique is NP-hard, defining the best sets for J_1 and J_2 is not trivial. Consequently, a heuristic is proposed. The heuristic works by sequentially adding tasks to the sets until no further task can be added without breaking the properties required by J_1 and J_2. The output of the proposed heuristic depends on the initial ordering of the tasks and, thus, two different task orderings are considered: (1) non-increasing order of nominal operation times; and (2) non-increasing order of nominal plus protection times. Algorithm 1 illustrates the greedy heuristic.

Algorithm 1 Calculation of sets J_1 and J_2 for *rLB2*.

Input: Task ordering $\langle h_1, h_2 \ldots h_{|V|}\rangle$. Nominal operation time t_i and protection time $\tilde{t}_i$ of each task i.
Output: The best found lower bound $rLB2^*$ and the groups J_1^{LB2}, and J_2^{LB2}.

```
1: J1 ← {∅}, J1^LB2 ← {∅}, J2^LB2 ← {∅}, rLB2* ← 0;
2: for i := 1 to |V| do
3:     J2 ← {∅}
4:     J1 ← J1 ∪ testInsertion1(J1, hi)
5:     for j := 1 to |V| do
6:         if hj ∉ J1 then
7:             J2 ← J2 ∪ testInsertion2(J1, J2, hj)
8:         end if
9:     end for
10:    rLB2 ← |J1| + ⌈|J2|/2⌉
11:    if rLB2 > rLB2* then
12:        rLB2* ← rLB2; J1^LB2 ← J1; J2^LB2 ← J2
13:    end if
14: end for
```

Algorithm 1 considers alternative task groupings for J_1 and J_2. Tasks are considered for inclusion (lines 4 and 7) according to two functions that verify whether the task can belong to the set according to the current members of each set. Function testInsertion1(J, i) verifies that task i cannot share a station with any task in J, that is, it checks if task i is incompatible with each task in set J, and function testInsertion2(J, J', i) verifies that task i cannot share a station with any task in J nor with any two tasks in J', that is, it checks whether task i is incompatible with each task in set J, and whether task i and each pair of tasks in J' satisfy constraint (7) when assigned to the same station. Among all of the possible groupings, the best groups, J_1^{LB2} and J_2^{LB2}, constitute the output of the algorithm.

Once groups J_1^{LB2} and J_2^{LB2} are available, *rLB2* can be calculated in linear time. Therefore, *rLB2* is specially suited for enumeration based procedures. Furthermore, Algorithm 1, when used in the SALBP-1, may provide alternative J_1 and J_2 sets which may improve the quality of *LB2*.

rLB3: *rLB3* is a second counting bound. For the SALBP, the bound considers that tasks in V are divided into five disjoint subsets: (1) set J_1 composed by all of the tasks with $t_i > 2c/3$; (2) set J_2 composed by all of the tasks with $t_i = 2c/3$; (3) set J_3 composed by all of the tasks with $c/3 < t_i < 2c/3$; (4) set J_4 composed by all of the tasks with $t_i = c/3$ and (5) set J_5 with the remaining tasks. The bound is then computed as $\lceil |J_1| + 2/3|J_2| + 1/2|J_3| + 1/3|J_4| \rceil$, following a similar rationale to the one exposed in *rLB2*.

As in *rLB2*, an alternative definition of the sets $J_1, \ldots, J_4$ which takes into account protection times can strengthen the bound. The same task orderings and a similar greedy heuristic, see Algorithm 2, to the ones used for the *rLB2* are used.

Algorithm 2 relies on three functions (invoked in lines 4, 8, 13 and 17) to ascertain whether a task may belong to a specific group. Functions testInsertion1(J, i) and testInsertion2(J, J', i) correspond to the two functions used for *rLB2*. Function testInsertion3(J, J', J'', i) tests whether task i can neither share a station with any task in J, nor with any task in J' and in J'', nor with any three tasks in J''.

As in the case of *rLB2*, the procedure can be used as an alternative way to define *LB3* and, once the best sets of tasks have been determined, the bound can be computed in linear time.

Bounds on the earliest and the latest station assignment of tasks

The bin packing bounds can be used to determine the earliest and the latest station to which a task can be assigned. The bounds are based on the requirements of the sets of predecessors and successors of any given task.

Let *UB* be the number of stations required by the incumbent solution, and let e_i (l_i) denote the earliest (latest) feasible station assignment for task i if a solution with *UB*-1 stations is to exist.

Algorithm 2 Calculation of sets J_1, J_2, J_3 and J_4 for *rLB*3.

Input: Task ordering $\langle h_1, h_2 \dots h_{|V|}\rangle$. Nominal operation time t_i and protection time $\bar{t}_i$ of each task i.
Output: The best found lower bound $rLB3^*$ and the groups J_1^{LB3}, J_2^{LB3}, J_3^{LB3}, J_4^{LB3}.

$J_1 \leftarrow \{\emptyset\}$, $J_1^{LB3} \leftarrow \{\emptyset\}$, $J_2^{LB3} \leftarrow \{\emptyset\}$, $J_3^{LB3} \leftarrow \{\emptyset\}$, $J_4^{LB3} \leftarrow \{\emptyset\}$, $rLB3^* \leftarrow 0$;
for i := 1 **to** $|V|$ **do**
  $J_2 \leftarrow \{\emptyset\}$, $J_3 \leftarrow \{\emptyset\}$, $J_4 \leftarrow \{\emptyset\}$
  $J_1 \leftarrow J_1 \cup$ testInsertion1(J_1, h_i)
  for j:=1 **to** $|V|$ **do**
    $J_3 \leftarrow \{\emptyset\}$, $J_4 \leftarrow \{\emptyset\}$
    if $h_j \notin J_1$ **then**
      $J_2 \leftarrow J_1 \cup$ testInsertion1$(J_1 \cup J_2, h_j)$
    end if
    for k:=1 **to** $|V|$ **do**
      $J_4 \leftarrow \{\emptyset\}$
      if $h_k \notin J_1 \cup J_2$ **then**
        $J_3 \leftarrow J_3 \cup$ testInsertion2$(J_1 \cup J_2, J_3, h_k)$
      end if
      for l:=1 **to** $|V|$ **do**
        if $h_l \notin J_1 \cup J_2 \cup J_3$ **then**
          $J_4 \leftarrow J_4 \cup$ testInsertion3$(J_1, J_2 \cup J_3, J_4, h_l)$
        end if
      end for
      $rLB3 \leftarrow |J_1| + \lceil 2/3|J_2| + 1/2|J_3| + 1/3|J_4| \rceil$
      if $rLB3 > rLB3^*$ **then**
        $rLB3^* \leftarrow rLB3$; $J_1^{LB3} \leftarrow J_1$; $J_2^{LB3} \leftarrow J_2$; $J_3^{LB3} \leftarrow J_3$; $J_4^{LB3} \leftarrow J_4$
      end if
    end for
  end for
end for

Define $rLB1(V')$, $rLB2(V')$ and $rLB3(V')$ according to Eqs. (18), respectively. These expressions take a set V' of tasks, $V' \subseteq V$, and return an estimation of the required stations according to the packing bounds.

Let $\langle h'_1, h'_2 \dots h'_{|V'|}\rangle$ be an ordering of the tasks in V' according to non-decreasing protection times. Let k be the smallest integer such that $\sum_{i\in V'} t_i + \sum_{i=1}^{k} \hat{t}_{h'_i} \leq k \cdot c$ holds, which is the condition on the minimum number of stations to ensure one protection time per station. Given the two previous definitions, Eq. (18) corresponds to the calculation of *rLB*1 according to expression (13).

$$rLB1(V') = \frac{\sum_{i\in V'} t_i + \sum_{i=1}^{k} \bar{t}_{h_i}}{c} \tag{18}$$

Eqs. (19) and (20) correspond to the calculation of *rLB*2 and *rLB*3 respectively, and make use of the sets obtained for *rLB*2 and *rLB*3 according to Algorithms 1 and 2.

$$rLB2(V') = |J_1^{LB2} \cap V'| + 1/2|J_2^{LB2} \cap V'| \tag{19}$$

$$\begin{aligned} rLB3(V') = {} & |J_1^{LB3} \cap V'| + 2/3|J_2^{LB3} \cap V'| \\ & + 1/2|J_3^{LB3} \cap V'| + 1/3|J_4^{LB3} \cap V'| \end{aligned} \tag{20}$$

Based on Eqs. (18)–(20), the following set of inequalities on the earliest and latest stations of a task are derived, and the smallest (largest) integer value satisfying the condition corresponds to the bound on the earliest (latest) station assignment, see inequalities (21) and (22).

$$e_i \geq \max\{rLB1(P_i^* \cup \{i\}), rLB2(P_i^* \cup \{i\}), rLB3(P_i^* \cup \{i\})\} \tag{21}$$

$$l_i \leq UB - \max\{rLB1(F_i^* \cup \{i\}), rLB2(F_i^* \cup \{i\}), rLB3(F_i^* \cup \{i\})\} \tag{22}$$

The conditional inequalities test whether a task can or cannot be assigned to the same station as its predecessors or successors, see [34]. As the inequalities on the latest stations, l_i, impose restrictive conditions for an improving solution to exist, in other words, $l_i < UB$ for all tasks, effectively rejecting any solution with UB stations, it is possible that $e_i > l_i$ holds for some task i. Such an inequality would reflect the impossibility to improve the incumbent solution and implies that UB is optimal.

3.2. Max-flow based lower bound

A max-flow bound for the SALBP-1 was proposed in [40]. The bound builds upon a necessary condition for a solution with a given number of stations, $UB-1$, to exist. The logical test assimilates the problem to a bipartite max-flow problem in an auxiliary graph. The auxiliary graph contains a supply vertex for each station, and a demand vertex for each task. The supply represents the amount of time available to perform operations, and the demand represents the operation times required by the task. The auxiliary graph only contains an arc from a station to a task if the task can be assigned to the station. If the flow does not satisfy the demand of all tasks, it means that infeasibility has been detected, and a larger number of stations is required. The test corresponds to the resolution of a max-flow problem but the structure of the auxiliary graph allows an efficient algorithm that runs in $O(|V| \cdot UB')$ time when the auxiliary graph is available [40] .

As in the SALBP-1, the demand of each task corresponds to its nominal operation time. The earliest and the latest stations are calculated according to Eqs. (21) and (22), and the supply of each station corresponds to the cycle time minus a bound on the protection times required by the station. Consequently, the supply of each station, $k = 1, \dots, UB-1$, corresponds to the optimal solution of an rKPPC instance, see Section 2.2, constructed as follows: the set T of candidate tasks contains all tasks that can be assigned to the station, that is $T = \{i \in V : e_i \leq k \leq l_i\}$; the profit of each task equals its nominal operation time, $p_i = t_i$; the weight and the protection weights are equal to the nominal operation time and the protection times of the rSALBP-1 instance; and the sets of absolute predecessors (successors) of the tasks correspond to the intersection of the set of candidate tasks and the set of absolute predecessors (successors) of the original instance.

The optimal solution of the rKPPC for station k constitutes an upper bound on the sum of nominal operation times that station k can perform. Moreover, the difference between the cycle time and the rKKPC solution not only accounts for the protection times but also for unavoidable idle times within the station assignments.

In order to solve the flow problem, tasks are considered according to a non-decreasing latest station task ordering, breaking ties by prioritizing tasks with smaller earliest station bounds. For each tentative number of stations, the graph is initialized by setting the supply of any unused station to its respective maximum value, and unassigned tasks are then considered in the mentioned order. The demand of each task is then satisfied by the stations with enough supply and the lowest possible cardinality. After meeting the demand, the supply is updated and the process is repeated until all the demand is satisfied or the demand of a task cannot be met. In such a case, the logical test ensures that a solution with $UB-1$ cannot be obtained.

The test extends the logic of *LB*1 and partially takes into account precedence constraints through the use of the bounds on the earliest and latest station assignments. Moreover, while the bound requires the evaluation of multiple knapsack-like problems, if the data is available, the method is easily calculated and provides a simple bounding procedure when $\Gamma \geq 2$ that takes into account multiple protection times per station.

3.3. Dantzig–Wolfe based lower bound

An alternative formulation for the problem corresponds to a Dantzig–Wolfe decomposition, [for the SALBP case see 30,31]. For the SALBP-1, a relaxation of the decomposition provides the best lower bounds [31]. Consequently, a similar approach for the rSALBP-1 is proposed and evaluated.

In order to describe the formulation, let $\mathbf{q}$ represent a feasible station assignment in which $q_i = 1$ if the ith task belongs to the

said assignment, and $q_i = 0$ otherwise. An assignment $\mathbf{q}$ is said to be feasible if the following three conditions hold:

- The robust knapsack constraint is satisfied (that is, the sum of nominal operation and protection times of the assigned tasks does not exceed the cycle time).
- Precedence relations within the station assignments are fulfilled (that is, if tasks i and i' belong to the same assignment, tasks that are successors of task i and predecessors of task i' also belong to the assignment).
- All of the tasks in the station assignment can be assigned to the same station (that is, there exists a station k such that $e_i \leq k \leq l_i$ for all tasks with $q_i = 1$).

The first and second conditions correspond to the constraints of the rKPPC described in Section 2.2, while the third can be achieved by restricting the set of tasks T to the subset of tasks that can be assigned to a given station k, that is $T = \{i \in V : e_i \leq k \leq l_i\}$.

If the set of feasible assignments is denoted by Q, the optimal subset of assignments such that each task belongs to exactly one assignment is a lower bound for the rSALBP-1 in which precedence relations among tasks in the same station are enforced but precedence relations among tasks in different stations are not. The formulation model corresponds to a set partitioning problem, SPP, whose linear relaxation is depicted in formulation (23)–(25), and in which variables $z_{\mathbf{q}}$, $\mathbf{q} \in Q$, represent the use of station assignment $\mathbf{q}$.

$$z^* = \min \sum_{\mathbf{q} \in Q} z_{\mathbf{q}} \tag{23}$$

$$\sum_{\mathbf{q} \in Q} q_i z_{\mathbf{q}} = 1 \qquad \forall i \in 1, ..., |V| \tag{24}$$

$$0 \leq z_{\mathbf{q}} \leq 1 \qquad \forall \mathbf{q} \in Q \tag{25}$$

In order to avoid the complete enumeration of set Q, model (23)–(25) is solved with a subset of the variables, and additional assignments are priced by solving an auxiliary rKPPC for each station, see Section 2.2, in which the objective coefficients of the rKPPC are the dual prices of constraint set (24). This lower bound is computationally expensive, yet the computational experiments reported in Section 5 show that it provides a tight bound.

3.4. Hoffmann heuristic

The Hoffmann heuristic [22] and its recently proposed modifications [36,38] are among the best available upper bound methods for the SALBP-1. The heuristic enumerates all the feasible station assignments of a given station, selects the best one according to some greedy criterion (usually idle time minimization) and repeats the process for the following station until all of the tasks have been assigned. The proposed implementation builds upon the approach given in [36] and differs from the classical heuristic in the following characteristics:

1. For any given station, a maximum number of assignments is generated. The limit avoids running time issues caused by the exponentially many assignments that a station may have, see [38], and it is a parameter of the algorithm, see Section 5.
2. In order to take advantage of the reversibility of the instance, see Section 2.1, the heuristic is applied to both the original and the reverse instances.
3. The criterion used to select among alternative station assignments is modified so as to take robustness into account. hence, among the generated station assignments, the candidate is selected according to a weighted function that combines the nominal operation times, the protection times, the precedence constraints and the number of unassigned tasks. Let $V' \subset V$ be the set of unassigned tasks for any given partial solution. Then, the greedy criterion prioritizes the task assignment that minimizes $\alpha_t \sum_{i \in V'} t_i + \alpha_{\bar{t}} \sum_{i \in V'} \bar{t}_i + \beta \sum_{i \in V'} |F_i^*| - \gamma |V'|$, in which α_t, $\alpha_{\bar{t}}$, β and γ are parameters.
4. The use of different weighting parameters is known to provide different solutions [36]. Therefore, the heuristic is run with different values, see Section 4.4.

4. A branch-and-bound method for the rSALBP-1

This section describes the proposed enumeration method. The method builds upon the branch, bound and remember algorithm, BBR, which has proved to be effective in the resolution of several line balancing problems [26,36,41].

The remainder of this section is divided into four parts. Section 4.1 describes the BBR algorithm. Section 4.2 puts forward several dominance rules to reduce the search space. Section 4.3 describes two preprocessing steps, one to reduce the instance size and another to strengthen the lower bounding procedures. Finally, Section 4.4 provides additional details on the implementation.

4.1. Branch, bound and remember

The proposed enumeration method can be defined as a branch-and-bound algorithm to explore the state space of a dynamic programming, DP, formulation of the rSALBP-1. The rSALBP-1 DP formulation regards the problem as a multistage decision process, and the objective is to find the policy, i.e. a set of decisions, to follow.

The proposed DP formulation is station-oriented, see [3,34]. The state space is divided into stages, each associated to a number of closed stations. A state S_u is identified by its stage u (number of closed stations) and the subset of tasks assigned to the current and to the previous stations, $S_u \subseteq V$. A transition, $T = S_{u+1} \setminus S_u$, between a state S_u at stage u and a state S_{u+1} at stage $u+1$ corresponds to an assignment of tasks to station $u+1$ that satisfies precedence and cycle time constraints. Notice that the relation between a partial solution, its equivalent state representation, and the transitions required to reach the state are easily established and this section sometimes refers to a partial solution and to its equivalent state indistinctly.

Due to the large size of the state space, it is constructed while it is explored. Initially, the state space contains an initial unexplored state, belonging to stage 0, in which no tasks are assigned. Then, the BBR method iteratively selects an unexplored state from stage u and generates all of its transitions (feasible station assignments) to states of stage $u+1$. For each of the ensuing states, the packing and the max-flow bounds are used to fathom states than cannot lead to a better solution. Furthermore, dominance rules among states, see Section 4.2, are applied and only those non-dominated and non-fathomed states are added to the state space. The process is repeated until all of the states from the state space have been explored or fathomed.

The main difference between the BBR and the traditional depth-first branch-and-bound is that the BBR stores the complete state space in memory, hence the memory requirements of the method are increased. Storing the state space enables the method to avoid the exploration of equivalent partial solutions (partial solutions that correspond to the same state), and makes use of alternative search strategies possible. In this work, the cyclic best first search, CBFS, rule [36] is used. This rule has shown its effectiveness in other line balancing problems [26,36,41] as it introduces a diversification mechanism within the branch-and-bound search, allowing the algorithm to simultaneously explore different areas of the search tree (see [27] for a broader discussion of the advantages and drawbacks of the method).

The CBFS rule tries to explore different areas of the state space by selecting the next state to explore from different stages. The rule begins by choosing a state from the first stage, then chooses a state from the second stage, and continues choosing states from consecutive stages until reaching the UBth stage. The selection rule then cycles back to the first stage and repeats the selection order. In order to prioritize the most promising states of each stage, the next state to explore is selected by prioritizing states with minimum sum of nominal operation times among the unassigned tasks.

4.2. Dominance rules

Enumeration methods for ALBPs have traditionally relied on the application of different dominance rules to avoid the exploration of equivalent, or dominated, partial solutions. As the BBR stores the state space, comparison to previously generated states is possible, thus increasing the ability of the method to avoid work repetition. The following rules extend previous rules for the SALBP and other similar problems to include the specificities of the rSALBP.

Memory based dominance rule. This rule detects equivalent states, see [36]. Let S_u be a stored state, and $S_{u'}$ be a newly generated state. Then, if the set of tasks of both states are identical, $S_u = S_{u'}$, and both states belong to the same stage, $u = u'$, the newly generated state can be discarded. Notice that if $u < u'$, state $S_{u'}$ is strictly dominated by state S_u and can be safely ignored. Moreover, if $u > u'$, state S_u can be replaced in the state space by state $S_{u'}$.

Maximum load rule. This rule detects unnecessary transitions. Define transition $T = S_{u+1} \setminus S_u$ as maximal if no other task could be included in T satisfying precedence and cycle time constraints. Then, for every non-maximal transition, there is a maximal transition in which a superset of tasks is assigned. The number of stations required by both generated states is identical, but the number of stations required by their respective unassigned tasks cannot be larger for the state generated by the maximal transition than for the state generated by the non-maximal one. Hence, the latter is dominated by the former.

Generalized maximum load rule. States from multiple transitions may lead to a dominance relation similar to the maximum load rule, see [32].

Let S_u and S'_u be two states. If $S'_u \subset S_u$, then S'_u can be seen as a partially loaded state and S_u dominates S'_u. An exhaustive verification of the above rule would be inefficient, since verifying all supersets is a combinatorial exercise. Consequently, the proposed implementation only provides a partial test of the rule.

The procedure considers each task $i \in V \setminus S'_u$ such that $P_i \subseteq S'_u$, and tests whether state $S'_u \cup \{i\}$ is already part of the state space. If the state exists and does not require a larger number of stations, then S'_u is discarded.

Task dominance rule. The task dominance rule, also known as the Jackson rule [34,40], verifies a dominance relation among states that differ only in two tasks.

Consider two tasks i, i' such that $t_{i'} \leq t_i$, $F_{i'} \subseteq F_i$ and $\bar{t}_{i'} \leq \bar{t}_i$ hold. Then, task i is said to dominate task i'. The rule stems from the fact that whenever task i can be assigned, so does task i', but the reverse does not hold.

The classical method used to test the rule [34,36,40] verifies whether a feasible transition T in which task i' is included remains feasible to precedence and cycle time constraints when task i' is substituted for task i.

A second method [32] compares a newly generated state with the states in memory. For a given state S'_u such that $i' \in S'_u$, $i \notin S'_u$, and $P_i \subseteq S'_u$, the method verifies if the state space contains a state $S_{u'} = S'_u \setminus \{i'\} \cup \{i\}$ belonging to a stage $u' \leq u$. If that is the case, $S_{u'}$ dominates S'_u, and S'_u is discarded.

Sewell–Jacobson rule. This rule, see [36], tries to minimize memory requirements associated to the BBR by postponing the creation of some transitions. Consider a state S_u with at least two feasible transitions, T and T'. The tasks in transition T have no successors, that is $\forall i \in T,\ F_i = \{\emptyset\}$, while at least one of the tasks in the station assignment associated to transition T' has a successor, that is $i \in T', F_i \neq \{\emptyset\}$. If these conditions hold, station assignment T could be delayed to a subsequent stage without altering the solution, but the same may not hold for T'. Therefore, state $S_{u+1} = S_u \cup T$ is dominated by state $S'_{u+1} = S_u \cup T'$.

The rule is verified during the enumeration of transitions by checking the set of unassigned tasks with successors and the set of tasks in the transition with successors.

Solitary task. If a task cannot share a station with any other unassigned task, that is, the task is incompatible with all the unassigned tasks, it is favorable to assign the task to a station as soon as possible in order to reduce the number of transitions.

Consequently, for a given state S_u, if a solitary task exists, states generated by other transitions are said to be dominated by the transition associated to the solitary task. Note that the solitary task and the Sewell–Jacobson rule may lead to contradictory dominance relations. In the implementation, the Sewell–Jacobson rule is only applied if a state is not found to have solitary tasks.

4.3. Preprocessing

It is advisable to apply some preprocessing rules to the original instance before starting the enumeration procedure in order to reduce its size and to strengthen the quality of the lower bounding procedures.

Implicit precedence relations. If the latest feasible station assignment of task i is no larger than the earliest station station assignment of task i', that is if $l_i \leq e_{i'}$ holds, it is safe to assume that task i is performed before task i'. Hence, task i can be added to the set of predecessors of task i' and task i' can be added to the set of successors of task i.

As the earliest and latest station assignments of a task depend on the sets of predecessors and successors, whenever new precedence relations are found, the earliest and latest station assignments should be recalculated in order to search for additional precedence relations.

Solitary tasks. During the preprocessing step, it is possible to detect tasks that cannot share a station with any other task. These tasks are removed from the instance, creating a tighter one.

If a task i is incompatible with all other tasks, $i' \in V \setminus \{i\}$, which may share a station assignment with it, that is, tasks for which $\{k : e_i \leq k \leq l_i\} \cap \{k : e_{i'} \leq k \leq l_{i'}\} \neq \{\emptyset\}$ holds, then any station assignment comprising task i includes only the said task.

If a solitary task i exists, the instance is reduced by removing it and updating the list of predecessors (successors) of each predecessor and successor of the task. For each predecessor task, $i' \in P_i$, let $F_{i'} = F_{i'} \cup F_i$; and for each successor task, $i' \in F_i$, let $P_{i'} = P_{i'} \cup P_i$.

4.4. Scheme of the algorithm and implementation details

The scheme of the proposed procedure follows, several details on the implementation, such as parameter settings or the order of application of bounds and dominance rules, are included in this description.

1. Apply the Hoffmann heuristic described in Section 3.4. An implementation based on the recursive functions given in [14] is used to generate all feasible station assignments. The heuristic is run once for the original and for the reverse instances with every combination of $\alpha_t,\ \alpha_{\bar{t}},\ \beta,\ \gamma \in \{0, 0.025, 0.05, 0.075, 0.1\}$. A limit of 1000 assignments per station is imposed. Let UB be the number of stations of the best found solution.
2. Apply the bin packing based lower bounds $rLB1$, $rLB2$ and $rLB3$, see Section 3.1, and let LB be the best bound. If $LB = UB$, the optimal has been verified, otherwise continue.
3. Obtain the earliest and the latest assignments of each task, see Section 3.1.
4. Apply preprocessing rules to add precedence relations and to find solitary tasks. If precedence relations are found, return to the previous step, otherwise continue to the next step.
5. Apply the max-flow lower bound, update LB and check whether $LB = UB$. If optimality has not been verified, go to next step.
6. Ascertain whether the original or the reverse instance is to be solved by the BBR. The instance is selected according to an estimation of its branching factor [36].
   Let $f_m = |\{j : e_j \leq m\}|$ and $r_m = |\{j : l_j \geq UB - m\}|$, that is, the number of tasks that can be assigned to the first m and the last m stations, respectively. Calculate $f = \Pi_{i=1}^{5} f_i$ and $r = \Pi_{i=1}^{5} r_i$, and select the forward direction if $f \leq r$ and the reverse direction otherwise. As stated in [36], f should be correlated to the number of branches if the original direction is considered, and r to the number of branches if the reverse instance is solved.
   If the reverse instance is selected, previously calculated values are updated accordingly.
7. Apply the BBR method, see Section 4.1. Whenever a state is selected for exploration, the solitary task dominance rule is applied. If a solitary task is found, only this transition is generated. Otherwise, all feasible station assignments are generated executing the same recursive method used for the Hoffmann heuristic to enumerate all station assignments, see step 1.
   For each state, the dominance rules and lower bounds are applied in the following order: (1) maximum load rule; (2) classical task dominance rule; (3) Sewell–Jacobson dominance rule; (4) bin-packing based lower bounds, see Section 3.1; (5) max-flow based lower bound, see Section 3.2; (6) memory based dominance rule; (7) generalized maximum load rule; and (8) generalized task dominance rule. This order applies easier-to-compute tests earlier, and only applies those tests that require comparisons with previously stored states if the first tests do not fathom the state.
   The state space is stored in memory using a hash table data structure. The hash table allows a fast lookup, which is critical for the application of several dominance rules. Additionally, the CBFS is implemented using priority queues to store references to unexplored states of each stage. A description of both data structures can be found in [9]. The maximum number of states that can be simultaneously stored in memory is limited to 10^8 states. This limit keeps memory consumption within the capabilities of current commodity computers.
   The enumeration continues until either optimality is verified, a time limit is reached, or the number of states in memory reaches the specified limit. The time limit is set to 3600 s (1 h).

As an optional step, the Dantzig–Wolfe based lower bound described in Section 3.3 can be applied before the BBR method. If the optimality of the heuristic solution is not verified, then we can apply the enumeration step. The running times required by this step may exceed the time required by the BBR. Hence, the lower bound is optional and its usefulness is investigated in the computational results section. In order to solve the linear programming relaxations, the implementation uses the academic CPLEX library, version 12.6, with default parameter settings.

5. Computational results

The proposed methods were implemented in C++, compiled using GCC 4.4.7 and run on a 32 processors Intel Xeon E5-2650 2 GHz CPU with 128GB RAM running the Linux operating system. The implementation is single-threaded and does not make use of any form of parallelism. Moreover, the inherent machine parallelism is used to solve multiple instances simultaneously. The limit on the maximum number of states ensures that the total memory use of any execution is much smaller than the total available memory (the maximum memory usage was below 8GB in each execution).

Two computational experiments were conducted. The first experiment compares the proposed method with the results provided in [20] and uses instances derived from the classical SALBP instance set. The second experiment uses the instances proposed in [28] and investigates the impact of different instance characteristics and levels of conservatism on the performance of the proposed method.

An online supplement compares the results provided by the proposed method with two alternative approaches based on the resolution of mathematical formulations using the commercial solver CPLEX. The first of these formulations is the compact formulation given in Section 2.1, while the second corresponds to an application of the general mixed integer linear programming formulation proposed in [5,6]. These results are not reported in this section as they were not competitive when compared to the results of the proposed method and the alternative decomposition approach [20] considered in this section.

5.1. Results for instances derived from the classical SALBP set

This experiment compares the proposed method with a previous approach found in the literature for a B&S robust line balancing problem, see [20]. In [20] the authors consider a type-2 robust SALBP formulation which shares the constraint sets of the rSALBP-1, fixes the number of workstations to a certain value and aims at minimizing the cycle time, c. In order to solve the problem, the authors propose a Benders Decomposition in which the Benders cuts correspond to *robustified* cycle time constraints. The method is then implemented using CPLEX to solve both the master problem and the subproblems, and the proposed implementation is tested using an instance set derived from the classical SALBP data set, see [34]. The experiments show that the decomposition is able to find the optimal solution, but the running time requirements are high.

While the method proposed in the present work cannot directly solve the type-2 robust SALBP, in [34,41] the authors show that type-2 line balancing problems can be solved by iteratively solving type-1 instances with different cycle times until the minimum cycle time with the desired number of stations is found. Consequently, we propose a two-phase binary search method to solve the type-2 robust SALBP, see [34] for alternative approaches.

The first phase of the search aims at finding a small interval for the candidate cycle times using the lower and the upper bounds described in Sections 3.1 and 3.4. Two binary searches are conducted. The first binary search uses $rLB1$ to find a lower bound, while the second binary search uses a single execution of the Hoffmann heuristic to find an upper bound. The initial lower and upper

Table 1
Summary of results for the classical SALBP instances applied to the type-2 objective with protection level 3 ($\Gamma = 3$). Instances are grouped according to their precedence graph. Three sets of results are reported: (1) the results given in [20], (2) the results provided by the proposed method for the *scaling* instances, and (3) the results provided by the proposed method for the *randomized* instances. The average, columns CPU_{avg}, and maximum, columns CPU_{max}, running times are also provided.

Graph	$\|V\|$	m	Decomposition method [20]		Proposed method *scaling* instances		Proposed method *randomized* instances	
			CPU_{avg}	CPU_{max}	CPU_{avg}	CPU_{max}	CPU_{avg}	CPU_{max}
Buxey	29	7–14	1992.6	3677	0.2	0.3	0.2	0.3
Gunther	35	6–15	2815.5	17,702	0.4	0.7	0.4	0.8
Hahn	53	3–10	552.5	1630	6.4	13.3	7.5	17.8
Kilbridge	45	3–11	23082.8	65,212	20.5	50.1	16.6	31.7
Lutz1	32	8–12	1217.6	1716	0.9	2.3	1.0	2.5
Sawyer	30	7–14	11367.7	31,362	0.2	0.3	0.2	0.4
Tonge	70	3–6	2275.2	7653	249.5	399.0	273.2	546.5

limits for the search intervals in these binary searches are respectively set to the largest operation plus protection time of all tasks, and to the sum of operation times of all tasks.

Once the reduced search interval is found, the second phase applies a binary search in which the BBR described in Section 4.4 is used. The total running time required to solve the problem for a given instance corresponds to the sum of the running times of the two phases.

In order to compare the results of the two-phase method with the Benders decomposition approach from [20], we derive a new instance set using the same set of SALBP instances used in [20].

The new instances share the precedence graph and the nominal operation times with their corresponding SALBP instances and add protection times for each task according to a random generation procedure. In this work, protection times were created following two alternative procedures: (1) a *scaling* procedure in which the protection time of a given task i is proportional to its nominal operation time, $\tilde{t}_i = \lceil \psi t_i \rceil$; and (2) a *randomized* procedure in which the protection time of a given task i, $\tilde{t}_i$, is randomly drawn from a discrete uniform distribution $\mathcal{U}(1, \psi t_i)$.

The structure of the protection times tries to reflect two situations: *scaling* instances represent protection times completely correlated to the nominal times, while *randomized* instances represent the case in which the nominal and the protection times are correlated but some additional characteristics also influence the protection time value. In both cases, the parameter ψ controls the scale of the uncertainty interval. Instances with three different values, $\psi = \{0.1, 0.2, 0.5\}$, were generated.

For each original SALBP instance, six rSALBP instances are derived following each combination of generation procedure and ψ value.

Table 1 compares the results of the two-phase method with the results provided in [20], in which instances with a varying number of stations and different levels of conservatism, $\Gamma = 1, \ldots, 10$, were solved. Table 1 reports the running time to reach optimality when the instances are grouped according to their precedence graph. For each precedence graph, the number of tasks, column $|V|$, and the range of workstations considered, column m, are also provided. For each group of instances, the average and the maximum running times to reach optimality reported in [20], columns *Decomposition method*, and the average and the maximum running times to reach optimality provided by the two-phase method, columns *Proposed method*, are given. The results of the proposed method are given separately for the instances generated according to the *scaling* and the *randomized* generation approaches. The table only provides the results for $\Gamma = 3$, as in [20]. The results for other levels of conservatism are included in the supplementary online material.

Even when the proposed enumeration method was not originally designed to solve type-2 instances, the results reported in Table 1 illustrate the effectiveness of the proposed iterated search method as the average and maximum running times are clearly smaller for the proposed method.

Please note that the results are not completely comparable. First, the computers used to perform the tests are different (the results reported in [20] were obtained in a Pentium IV computer with a 3 GHz CPU and 2.5GB RAM); and second, the instances used in both experiments are different, even when they were created following similar generation procedures. Nevertheless, the results highlight the efficiency of the proposed method and exhibit a behavior common to other line balancing problems in which enumeration based methods outperform linear programming approaches. This behavior is attributed to the bad quality of the linear relaxations of the integer programming-based models, since the optimal solution to the linear relaxation of the SALBP equals the "trivial" bound introduced in Section 3.1. Additionally, this behavior is to be attributed to the presence of symmetries within the model, for example station assignments may be swapped between consecutive stations if no precedence relation between their respective station assignments exists. Enumeration based approaches try to avoid these issues by providing tighter lower bounds, as well as rules to minimize the exploration of equivalent partial solutions.

An analysis of the results leads to the following conclusions, which are further investigated in the second computational experiment:

(1) The differences between the instances constructed using the *scaling* and the *randomized* construction schemes are minimal. Hence, the method seems to be robust to the structure of the protection times.
(2) For the smaller instances, i.e. those with fewer than 40 tasks, improvements of several orders of magnitude are achieved. These improvements are to be attributed to the ability of the first phase of the two-phase method to obtain a tight interval of cycle times, as well as to the ability of the enumeration method to evaluate the complete state space within reduced running times.
(3) The difference between procedures is more significant for the smaller instances than for the instances derived from the Tonge precedence graph. This result is attributed to the larger average number of tasks per station of the Tonge instances. The larger number of tasks per station increases the number of alternative partial solutions that need to be enumerated, but it does not specially affect the decomposition approach.

5.2. Impact of different characteristics on the solution

As highlighted in [28], the classical SALBP instance set suffers from several shortcomings that complicate the study of a solution procedure. Consequently, the authors proposed a systematic procedure to generate instances with specific structures. The set enables

Table 2
Results for the new instance set grouped according to the different characteristics of the instance. For each group, the number (percentage) of optimal solutions found, the average running time and the average number of nodes of the BBR method are reported. Additional statistics on the average gap of the Hoffmann heuristic, column gap^h, the average optimality gap of the Dantzig–Wolfe based relaxation, column gap^{DW}, and the average running time of the said relaxation, column $t(s)^{DW}$, are also provided.

		#	#opt (%)	Av. t (s)	Av. states	gap^h	gap^{DW}	$t(s)^{DW}$
	Total	15,750	15,232 (96.71)	205.79	97,277	1.11	0.48	162.7
Structure	Block	4500	4340 (96.44)	260.16	112,462	1.25	0.28	270.8
	Chain	4500	4467 (99.27)	93.13	57,434	1.35	0.41	134.22
	Mixed	6750	6425 (95.19)	244.64	113,715	0.85	0.66	109.62
OS	Low	6750	6232 (92.33)	480.06	226,279	1.41	0.2	334.84
	Medium	6750	6750 (100)	0.11	692	1.12	0.45	44.12
	High	2250	2250 (100)	0.0	23	0.18	1.39	2.03
t	Bottom	5250	4956 (94.40)	285.97	70,237	0.51	0.43	440.42
	Middle	5250	5250 (100)	10.92	53,324	1.14	0.29	0.05
	Bimodal	5250	5026 (95.73)	320.47	168,269	1.67	0.71	47.63
$\bar{t}$	Scaling	7875	7556 (95.95)	232.16	116,141	1.21	0.49	186.57
	Randomized	7875	7676 (97.47)	179.42	78,413	1.01	0.46	138.83
ψ	0.1	5250	5191 (98.88)	83.41	49,698	0.88	0.42	148.2
	0.2	5250	5112 (97.37)	168.12	75,297	1	0.42	128.72
	0.5	5250	4929 (93.89)	365.84	166,835	1.45	0.59	211.18
Γ	1	3150	3115 (98.89)	92.06	55,978	1.19	0.45	68.05
	2	3150	3046 (96.70)	202.33	97,171	1.3	0.48	144.8
	3	3150	3043 (96.60)	229.23	106,622	1.13	0.47	135.56
	4	3150	3016 (95.75)	255.93	116,879	1	0.51	259.51
	5	3150	3012 (95.62)	249.38	109,735	0.92	0.47	205.57

the analysis of the impact of different instance characteristics on a solution procedure. The instances were constructed according to three different characteristics that depict real assembly line conditions while still posing a challenge to solution methods.

The first characteristic controls the structure of the precedence graph. Three different structures are considered: *block* constraints (some tasks have multiple predecessor and successor relations), *chain* constraints (tasks usually have a single predecessor and a single successor), and *mixed* constraints (both *block* and *chain* precedence relations exist).

The second characteristic is the ratio of precedence constraints. This ratio is measured using the order strength, OS, metric, that is the fraction of absolute precedence constraints in the instance. Three levels, *low* (OS=0.2), *medium* (OS=0.6), and *high* (OS=0.9) are considered.

Finally, the third characteristic models the distribution of the operation times. Three different distributions are used: *bottom* (the operation times follow a normal distribution with one peak at 1/10 of the cycle time), *middle* (the peak stands at 1/2 of the cycle time) and *bimodal* (the processing times follow a bimodal distribution with one peak at 1/10 and another peak at 1/2 of the cycle time).

All instances share a common cycle time, $c = 1000$, in order to highlight that task times and cycle times can be scaled down or up at convenience, and that task times are inherently fractions of cycle times and not absolute values.

The *medium* instance set, which corresponds to instances with 50 tasks, is used following the recommendation given in [28, p. 40]. For each instance of the medium set, six rSALBP instances were generated following the approach described for the classical instance set: one for each combination of instance generation method, be it *scaling* or *randomized*, and uncertainty parameter level, $\psi = \{0.1, 0.2, 0.5\}$. Hence, a total of $525 \times 6 = 3150$ instances were generated. In order to keep a single common cycle time of the second set of instances, protection times are modified to ensure that all tasks are assignable to a station. Thus, if $t_i + \bar{t}_i > c$, the protection time is set to $\bar{t}_i = c - t_i$.

Each instance of the 3150 instances is solved with five different levels of conservatism, $\Gamma = 1, \ldots, 5$ for a total of 15,750 independent executions of the BBR method.

Table 2 reports the results of the proposed method when instances are grouped according to the different characteristics of the instance, namely: (1) structure of the precedence graph, rows *structure*; (2) order strength, rows *OS*; (3) distribution of operation times, rows t; (4) method used to generate the protection time, rows $\bar{t}$; (5) uncertainty parameter, rows ψ; and (6) level of conservatism, rows Γ. For each group of instances, six different metrics are provided: (1) the number (percentage in parenthesis) of optimal solutions found by the exact method within the imposed time limit, column #opt (%); (2) the average running time in seconds of the BBR (in case the algorithm reaches the time limit, the time is considered equal to 3600 s), column av. t (s); (3) the average number of explored states, column av. states; (4) the average gap between the best known solution and the Hoffmann heuristic; (5) the average gap between the best known solution and the Dantzig–Wolfe lower bound; and (6) the average time required to solve the Dantzig–Wolfe relaxation. The fourth and fifth metrics are measured according to Eqs. (26) and (27) in which UB^h, UB and LB^{DW} stand, respectively, for the best solution found by the Hoffmann heuristic, the best known solution and the best bound provided by the Dantzig–Wolfe based relaxation.

$$gap^h = 100 \cdot \frac{UB^h - UB}{UB} \tag{26}$$

$$gap^{DW} = 100 \cdot \frac{UB - LB^{DW}}{UB} \tag{27}$$

The results show that the method is able to solve most of the instances within the imposed time limit of one hour. Over 96% of the instances were solved to optimality, with an average running time of approximately 200 s. Moreover, the gaps of the Hoffmann heuristic are small, highlighting its ability to provide optimal or near-optimal solutions within short running times. If the results of the Dantzig–Wolfe based relaxation are analyzed, their gap is small but the average running times are large, in some cases larger than the average time required by the BBR. Consequently, its application is recommended only for the evaluation of heuristic solutions for larger instance sizes, and for the evaluation of hard-to-solve instances. Specifically, the Dantzig–Wolfe bound is able to close 471 instances out of the 518 that Table 2 reports as open. Hence, only 47 of the 15,750 instances were not solved to optimality with the proposed methods. Furthermore, for each of the 47 unsolved instances, the difference between the best known lower and upper

bounds is equal to one unit; in other words, the solution reported by the method is one station above its lower bound.

Table 2 also shows that all the characteristics influence the average results provided by the proposed solution method. Specifically, instances with *low* OS and *bottom* or *bimodal* operation times are more difficult than other instances. The observation confirms the results from Section 5.1, since these instances have a larger number of alternative station assignments that need to be enumerated.

Notice that the set of characteristics that make an instance difficult differ from the characteristics reported for the SALBP-1 [26,31], in which the instances with *middle* operation times correspond to the hardest ones. This difference is to be attributed to the role of the enumeration phases. For the SALBP-1, instances with *low* OS and *bottom* or *bimodal* operation times are solved in the root node without relying on enumeration, that is, the initial lower and upper bounds coincide. For the rSALBP-1, the same does not hold and the instances are not closed before enumeration because there is an optimality gap of one or two stations. In such a case, instances with the mentioned characteristics have larger amounts of candidate station assignments and, thus, greater computational requirements.

If the impact of the characteristics associated to uncertainty is considered, the results show that the main source of differences corresponds to the uncertainty range, controlled by parameter ψ. The same result is observed if the protection levels Γ are increased, as the average running time increases and the number of optimal solutions decreases. Along the same lines, if the method used to generate the uncertainty values is taken into account, very similar solutions are found for *scaling* and *randomized* instances, although *randomized* instances seem slightly easier.

The results of the uncertainty characteristics can also be attributed to the performance of the lower bounds. If the *rLB*1 bound is considered, larger levels of conservatism lead to impoverishment of the solution quality, as both the protection times and the number of tasks contributing to the bound are underestimated. This negative result is partially palliated regarding instances with large task times. For such instances, incorporating protection times enables the detection of additional incompatibilities among tasks and improves the *rLB*2 and *rLB*3 lower bound values, since the role of the counting bounds does not affect all of the instances.

The average running times reported in Table 2 are significantly higher than the average running times for the deterministic problem reported in [26], in which the average running times of the enumeration phase are below 1 s. The difference has to be attributed to the fact that the lower bounds for the rSALBP-1 are not as efficient as the lower bounds for the SALBP-1 (in [26], the exact solution of the bin packing problem is used as a lower bound for the SALBP-1), and that the robust problem is significantly more difficult than the deterministic problem. The above conclusion is further supported by the positive correlation among the uncertainty parameter, the level of conservatism and the average time and memory requirements of the method. Increasing the parameters that control the uncertainty of the instance directly influences the performance of the algorithm, as fewer instances can be solved to optimality and larger average running times and memory requirements are to be expected.

Finally, Table 3 considers the effect of different protection levels and uncertainty intervals within the number of required stations. The results in Table 3 are compared against a baseline corresponding to the average number of stations of the optimal solution of the SALBP-1, which in turn corresponds to the robust instance set with no conservatism $\Gamma = 0$ and/or no uncertainty level $\psi = 0.0$. The results show that increasing robustness implies limited increments in station requirements.

Table 3
Average number of stations for different protection levels, columns Γ, and uncertainty intervals, columns ψ. The average number of stations when uncertainty is considered is reported in the upper left corner of the table (15.90 stations).

15.90	$\Gamma = 1$	$\Gamma = 2$	$\Gamma = 3$	$\Gamma = 4$	$\Gamma = 5$
$\psi = 0.1$	17.01	17.77	17.85	17.91	17.94
$\psi = 0.2$	18.06	19.38	19.53	19.6	19.64
$\psi = 0.5$	20.63	22.66	22.99	23.17	23.24

Note that the average number of stations required when uncertainty is present is calculated according to the best known solution. Nonetheless, the difference between the reported value and the average of the optimal number of solutions should be minimal, since the percentage of optimal solutions found by the algorithm is very high.

6. Conclusions

In this work a Bertsimas–Sim, B&S, robust version of the type-1 SALBP, denoted by rSALBP-1, is introduced and studied. A mathematical formulation, several lower bounds, a heuristic, and an exact solution method based on the branch, bound and remember enumeration technique are presented. From a computational point of view, the main conclusions of this work are:

(1) Medium-sized, 50 tasks, instances of different characteristics can be efficiently solved within an hour time limit. Moreover, the average running time required to verify optimality is smaller than the aforementioned limit. Therefore, an exact solution approach is recommended when the time available to compute a solution exceeds a minute.
(2) In case that a faster solution is sought, the Hoffmann heuristic is able to provide near optimal solutions, yielding very small optimality gaps, within very short running times. Hence, it is the recommended solution procedure for large size instances and/or small running time limits.
(3) While the Dantzig–Wolfe decomposition provides small gaps, its running time requirements are high (sometimes larger than the time required to verify optimality through enumeration). Moreover, for large size instances, it is likely that the exact enumeration procedure cannot verify optimality within the allotted time, leaving the decomposition bound as the best candidate to obtain lower bounds to evaluate the quality of a heuristic procedure.
Consequently, the method would only be recommended if lower bound comparisons are required and optimality is to be verified. When the Dantzig–Wolfe based heuristic is used, the proposed methods are able to verify optimally for 15,703 out of the 15,750 best found solutions (99.7%).
(4) Instances with low order strength and bottom and bimodal operation time distributions are distinctly more difficult than other instances. This is mainly caused by the larger amount of alternative station assignments that need to be considered.
The above result contradicts previous results for the SALBP-1 [26], in which instances with middle distribution of operation times are shown to be the most difficult. This change in behavior is caused by the quality of the available lower bounds.
(5) Increasing the uncertainty of the instance deteriorates the performance of the algorithm. This is due to the limitations of the bounding methods to capture the requirements of uncertainty. Moreover, the impact of increasing the uncertainty level is considerably larger than the impact of increasing the level of conservatism.

(6) The results show that increasing the protection level over a certain degree does not significantly increase the difficulty of the instance. This behavior is related to the number of tasks per stations. If the number of tasks in each station is smaller than or equal to a given Γ, increasing Γ does not alter the solution at all. Consequently, if the solution for large values of Γ is sought, it would be advisable to solve the deterministic problem considering the sum of the nominal and the protection times of each task as the task times, and to avoid the robustness approach.
(7) The results show that enumeration-based methods can be useful in the optimal resolution of B&S robustness approaches. Particularly, the proposed method is able to solve instances in shorter running times than the previously considered procedures.
The result may extend to other problems, specifically to problems in which branch-and-bound approaches belong to the state-of-the-art, leading to new methods that do not make use of integer-programming based approaches.
(8) The results also expose the limitations of the proposed approach, since the problem is significantly more difficult than its deterministic counterpart. Therefore, further research on the problem should address the development of efficient lower bounds to tackle instances with small (bottom distribution) tasks.

If the solutions are analyzed according to the impact of robustness within the station requirements, the following conclusions are reached:

(1) The number of additional stations required to protect a solution against uncertainty highly depends on the uncertainty levels. If the uncertainty level is very high, the number of additional stations is large. For example, a high uncertainty level like $\psi = 0.5$ indicates that solutions are protected against rises of up to 50% from their nominal operation times; hence it is logical that the number of stations increases significantly. In such a situation, it is likely that the manufacturing process would benefit from improvements that lead to reductions in the uncertainty (for example, reengineering the operations).
(2) When the uncertainty levels are small, as it is the case in highly standardized processes, small levels of conservatism require a modest number of additional stations. These stations may lead to savings since they avoid some reprocessing costs; hence it may be advisable to incur in the additional costs associated to incrementing the number of stations.

A relevant open area of research on robust assembly line balancing corresponds to the study of how to properly model the uncertainty on the operation times and the robustness requirements. Such issues are industry specific. Generally, nominal times are derived from methods-time measurement studies, while protection times should be derived from the known, but unavoidable, variability in the operation times, like micro-stoppages, defective materials, or work-overloads attributed to specific models within mixed-model assembly lines. An example of an unavoidable event affecting operation times, and thus inducing a protection time, is given in [19]. The process described corresponds to an automated assembly line in which micro-stoppages force additional manual work by an operator. Consequently, the proper operation of the system must account for these protection times and protection levels. In such a setting, line balancing should not only take these issues into account but rather highlight which sources of uncertainty have a higher impact on the performance of the assembly line.

As a final remark, a common practice in industrial settings is to increase the operation times of the tasks by some constant or ratio and to solve a deterministic problem. The formulation given in this work provides a relatively simple extension to this practice and the results provided in this paper show that more elaborated formulations can be solved within reasonable running times. Consequently, if uncertainty is an important factor within some specific settings, this work shows that it can be integrated into the decision process. Nonetheless, the considered formulation is based on the idea that the sources of uncertainty are not correlated, while in practice, the opposite is true. Hence, this work should be seen as a forward step to illustrate that basic robustness formulation can be efficiently tackled, and that more elaborate formulations can be proposed and considered.

Acknowledgments

This research has been partially funded by the research grant "Heterogeneous assembly line balancing problems with process selection features" number 1150306, from the Fondo Nacional de Desarrollo Científico y Tecnológico of the ministry of education of Chile. The authors also want to express their gratitude to the reviewers. Their suggestions have greatly improved this work.

Supplementary material

Supplementary material associated with this article can be found, in the online version, at 10.1016/j.omega.2017.08.020.

References

[1] Álvarez Miranda E, Ljubić I, Toth P. A note on the Bertsimas & Sim algorithm for robust combinatorial optimization problems. 4OR 2013;11(4):349–60. doi:10.1007/s10288-013-0231-6.
[2] Apak K, Gökçen H. A chance-constrained approach to stochastic line balancing problem. Eur J Oper Res 2007;180(3):1098–115. doi:10.1016/j.ejor.2006.04.042.
[3] Bautista J, Pereira J. A dynamic programming based heuristic for the assembly line balancing problem. Eur J Oper Res 2009;194(3):787–94. doi:10.1016/j.ejor.2008.01.016.
[4] Ben-Tal A, Ghaoui LE, Nemirovski A. Robust optimization. Princeton, New Jersey: Princeton University Press; 2009.
[5] Bertsimas D, Sim M. Robust discrete optimization and network flows. Math Program 2003;98(1–3):49–71. doi:10.1007/s10107-003-0396-4.
[6] Bertsimas D, Sim M. The price of robustness. Oper Res 2004;52(1):35–53. doi:10.1287/opre.1030.0065.
[7] Bertsimas D, Brown DB, Caramanis C. Theory and application of robust optimization. SIAM Rev 2011;53(3):464–501. doi:10.1137/080734510.
[8] Cakir B, Altiparmak F, Dengiz B. Multi-objective optimization of a stochastic assembly line balancing: a hybrid simulated annealing algorithm. Comput Ind Eng 2011;60(3):376–84. doi:10.1016/j.cie.2010.08.013.
[9] Cormen TH, Leiserson CE, Rivest RL, Stein C. Introduction to algorithms. Cambridge, Massachusetts: MIT Press; 2009.
[10] Delorme X, Dolgui A, Kovalyov MY. Combinatorial design of a minimum cost transfer line. OMEGA 2012;40(1):31–41. doi:10.1016/j.omega.2011.03.004.
[11] Dolgui A, Battaïa O. A taxonomy of line balancing problems and their solution approaches. Int J Prod Econ 2013;142:259–77. doi:10.1016/j.ijpe.2012.10.020.
[12] Dolgui A, Kovalev S. Scenario based robust line balancing: computational complexity. Discret Appl Math 2012;160(13–14):1955–63. doi:10.1016/j.dam.2012.04.011.
[13] Fekete SP, Schepers J. New classes of fast lower bounds for bin packing problems. Math Program 2001;31:11–31. doi:10.1007/s101070100243.
[14] Fleszar K, Hindi KS. An enumerative heuristic and reduction methods for the assembly line balancing problem. Eur J Oper Res 2003;145. doi:10.1016/S0377-2217(02)00204-7.
[15] Gabrel V, Murat C, Thiele A. Recent advances in robust optimization: an overview. Eur J Oper Res 2014;235(3):471–83. doi:10.1016/j.ejor.2013.09.036.
[16] Gagnon R, Gosh G. Assembly line research: historical roots, research life cycles and future directions. OMEGA 1991;19(5):381–99. doi:10.1016/0305-0483(91)90056-Y.
[17] Gamberini R, Grassi A, Rimini B. A new multi-objective heuristic algorithm for solving the stochastic assembly line re-balancing problem. Int J Prod Econ 2006;102(2):226–43. doi:10.1016/j.ijpe.2005.02.013.
[18] Gurevsky E, Battaïa O, Dolgui A. Balancing of simple assembly lines under variations of task processing times. Ann Oper Res 2012;201(1):265–86. doi:10.1007/s10479-012-1203-5.
[19] Gurevsky E, Hazır O, Battaïa O, Dolgui A. Robust balancing of straight assembly lines with interval task times. J Oper Res Soc 2013;64:1607–13. doi:10.1057/jors.2012.139.

[20] Hazır Ö, Dolgui A. Assembly line balancing under uncertainty: robust optimization models and exact solution method. Comput Ind Eng 2013;65(2):261–7. doi:10.1016/j.cie.2013.03.004.
[21] Hazır Ö, Dolgui A. A decomposition based solution algorithm for U-type assembly line balancing with interval data. Comput Oper Res 2015;59:126–31. doi:10.1016/j.cor.2015.01.010.
[22] Hoffmann TR. Assembly line balancing with a precedence matrix. Manag Sci 1962:551–62. doi:10.1287/mnsc.9.4.551.
[23] Martello S, Toth P. Knapsack problems. Chichester, UK: Wiley; 1990.
[24] Monaci M, Pferschy U, Serafini P. Exact solution of the robust knapsack problem. Comput Oper Res 2013;40:2625–31. doi:10.1016/j.cor.2013.05.005.
[25] Moreira MCO, Cordeau J-F, Costa AM, Laporte G. Robust assembly line balancing with heterogenous workers. Comput Ind Eng 2015;88:254–63. doi:10.1016/j.cie.2015.07.004.
[26] Morrison DR, Sewell EC, Jacobson SH. An application of the branch, bound, and remember algorithm to a new simple assembly line balancing dataset. Eur J Oper Res 2014;236(2):403–9. doi:10.1016/j.ejor.2013.11.033.
[27] Morrison DR, Sauppe JJ, Zhang W, Jacobson SH, Sewell EC. Cyclic best first search: using contours to guide branch-and-bound algorithms. Naval Res Logist 2017;64(1):64–82. doi:10.1002/nav.21732.
[28] Otto A, Otto C, Scholl A. Systematic data generation and test design for solution algorithms on the example of SALBPGen for assembly line balancing. Eur J Oper Res 2013;228(1):33–45. doi:10.1016/j.ejor.2012.12.029.
[29] Pape T. Heuristics and lower bounds for the simple assembly line balancing problem type 1: overview, computational tests and improvements. Eur J Oper Res 2015;240(1):32–42. doi:10.1016/j.ejor.2014.06.023.
[30] Peeters M, Degraeve Z. An linear programming based lower bound for the simple assembly line balancing problem. Eur J Oper Res 2006;168:716–31. doi:10.1016/j.ejor.2004.07.024.
[31] Pereira J. Empirical evaluation of lower bounding methods for the simple assembly line balancing problem. Int J Prod Res 2015;53(11):3327–40. doi:10.1080/00207543.2014.980014.
[32] Pereira J. Procedures for the bin packing problem with precedence constraints. Eur J Oper Res 2016;250(3):794–806. doi:10.1016/j.ejor.2015.10.048.
[33] Sarin SC, Erel E, Dar-El E. A methodology for solving single-model, stochastic assembly line balancing problem. OMEGA 1999;27:525–35. doi:10.1016/S0305-0483(99)00016-X.
[34] Scholl A, Becker C. State-of-the-art exact and heuristic solution procedures for simple assembly line balancing. Eur J Oper Res 2006;168(3):666–93. doi:10.1016/j.ejor.2004.07.022.
[35] Scholl A, Fliedner M, Boysen N. Absalom: balancing assembly lines with assignment restrictions. Eur J Oper Res 2010;200(3):688–701. doi:10.1016/j.ejor.2009.01.049.
[36] Sewell EC, Jacobson SH. A branch, bound, and remember algorithm for the simple assembly line balancing problem. INFORMS J Comput 2012;24(3):433–42.
[37] Sotskov YN, Dolgui A, Portmann MC. Stability analysis of an optimal balance for an assembly line with fixed cycle time. Eur J Oper Res 2006;168(3):783–97. doi:10.1016/j.ejor.2004.07.028.
[38] Sternatz J. Enhanced multi-Hoffmann heuristic for efficiently solving real-world assembly line balancing problems in automotive industry. Eur J Oper Res 2014;235(3):740–54. doi:10.1016/j.ejor.2013.11.005.
[39] Urban TL, Chiang W-C. An optimal piecewise-linear program for the U-line balancing problem with stochastic task times. Eur J Oper Res 2006;168(3):771–82. doi:10.1016/j.ejor.2004.07.027.
[40] Vilà M, Pereira J. An enumeration procedure for the assembly line balancing problem based on branching by non-decreasing idle time. Eur J Oper Res 2013;229(1):106–13. doi:10.1016/j.ejor.2013.03.003.
[41] Vilà M, Pereira J. A branch-and-bound algorithm for assembly line worker assignment and balancing problems. Comput Oper Res 2014;44:105–14. doi:10.1016/j.cor.2013.10.016.