



New solution approaches for balancing assembly lines with setup times

Jordi Pereira^{a,*,}, Marcus Ritt^b

^a Innovation and Sustainability Data Lab (ISDaLab). UPF - Barcelona School of Management, C. Balmes 132-134, 08008 Barcelona, Spain

^b Instituto de Informática, Universidade Federal do Rio Grande do Sul, Av. Bento Gonçalves 9500, 91340-110 Porto Alegre, Brazil

ARTICLE INFO

Keywords:

Assembly line balancing
Setup times
Packing
Heuristics
Dantzig–Wolfe decomposition

ABSTRACT

The objective of an assembly line balancing problem is to find an efficient allocation of tasks to the stations of the line according to time and technological (precedence) constraints. In this work we study the setup assembly line balancing and scheduling problem (SUALBSP) in which the time required to perform the tasks depends on their sequence within each station. In order to solve the problem, we propose different lower and upper bounds and check the efficiency of the proposed methods when compared to the state of the art. We show experimentally that the proposed methods lead to better primal and dual bounds, and that our new exact method optimally solves over 75% of all instances from the literature with a mean relative gap of less than 2% with a two minutes time limit. Moreover, for the subset of instances previously considered by exact approaches, our new exact approach obtains and proves the optimal solution for 150 of the 167 previously open instances, a significant improvement over previous exact methods.

1. Introduction

Given a set $V = [n]$ of n tasks,¹ with task times $t_i \geq 0$, $i \in V$ and a cycle time c , the *simple assembly line balancing problem of type I* (SALBP-I) is to find an assignment $a : V \rightarrow \mathbb{N}$ of the tasks to stations, that respects the cycle time and technological constraints, and minimizes the number of stations $m = \max_{i \in V} a(i)$. The cycle time is respected if the *station time* i.e., the total time of all tasks $t(a^{-1}(k))$ of each station $k \geq 1$ is at most c , where $t(S) = \sum_{i \in S} t_i$ is the total time of a set of tasks $S \subseteq V$. Technological (or precedence) constraints are modeled as a partial order $<$ on tasks V . These constraints are satisfied when $a(i) \leq a(j)$ for all $i < j$.

The SALBP-I stands as the most extensively researched assembly line balancing problem (ALBP) in the literature. Other ALBPs include the SALBP-II, which aims at minimizing the cycle time for a given number of stations, the SALBP-E, which minimizes the idle time subject to lower and upper limits on the number of stations and the cycle time, as well as problems that integrate additional aspects of specific assembly processes into their formulations. Models that integrate these aspects or involve more intricate objectives are designated as *general assembly line balancing problems*, GALBP (Baybars, 1986).

This work focuses on setup times, one of these additional features considered in the literature. In practical applications, setup times are common and can stem from various factors, such as changing tools, preparing workers for subsequent tasks (which may include walking

times), and transportation delays. Setup times may be sequence independent, sequence dependent, or even past sequence dependent (Koulamas and Kyparisis, 2008). We focus on sequence dependent setup times as sequence independent setup times within a SALBP-I can be absorbed into the task times during preprocessing. Adding setup times often substantially increases the complexity of a problem. Given that SALBP-I is already NP-hard, even when the number of stations is limited to two (Álvarez-Miranda et al., 2023), the introduction of setup times presents further obstacles to solve the problem efficiently.

1.1. Review of previous research

There has been a large amount of research on assembly balancing in general. We focus here on research on assembly line balancing with setup times and refer the reader to Becker and Scholl (2006), Boysen et al. (2022), Dolgui and Battaia (2013), Li et al. (2020), Morrison et al. (2014), Pape (2015), Scholl and Becker (2006), Sewell and Jacobson (2012), for a general overview on the ALBP literature and a detailed exploration of state-of-the-art procedures for the SALBP-I.

Two early ALBPs that consider sequence-dependent setup times are the Sequence Dependent Assembly Line Balancing Problem, SDALBP (Scholl et al., 2008) and the General Assembly Line Balancing Problem with Setups, GALBPS (Andrés et al., 2008).

Both SDALBP and GALBPS extend the SALBP-I formulation by integrating setups, but with different assumptions. In the SDALBP, setup

* Corresponding author.

E-mail addresses: jordi.pereira@bsm.upf.edu (J. Pereira), marcus.ritt@inf.ufrgs.br (M. Ritt).

¹ For a non-negative integer n we write $[n] = \{1, 2, \dots, n\}$.

times are seen as extra time required when a task is performed after another task, whereas in the GALBPS, setup times arise only between tasks performed consecutively. A task in the SDALBP may be involved in multiple setup times depending on the order of operations within a station. In the GALBPS tasks only have a single setup time that depends on the previous operation performed at the same station. As a result, in the GALBPS, the setup time of the first operation performed on a station depends on the last task performed at the station, since its preceding task in the station is the last task executed in the previous production cycle.

Another common approach to model setup times is the Setup Assembly Line Balancing and Sequencing Problem, SUALBSP, first introduced by Scholl et al. (2013). The SUALBSP builds upon the GALBPS by considering differences between *forward setups* and *backward setups*. Forward setups arise when performing consecutive tasks within the same station. Backward setups (or *last-to-first setups*) arise among consecutive tasks between two consecutive production cycles, i.e., from the setup time required for transitioning from the final task on a production cycle to the initial task in the subsequent production cycle on the same station. This not only differentiates the requirements associated with returning to the starting position within a station or carrying out some preliminary actions before initiating the same task on a different product, but also permits to include a setup time when a station performs only one task. Note that the SUALBSP subsumes the GALBPS as a special case by setting backward setup times equal to forward setup times and not assigning a backward setup time between a task and itself.

Considering our focus on the GALBPS and the SUALBSP, we review their literature here and point the reader to Boysen et al. (2022) for a state-of-the-art review including references to other ALBPs with setup time considerations. Before discussing available approaches, we note that the literature covers multiple preprocessing strategies, as they help exact and heuristic methods likewise. The most common methods include absorbing setup times into task times, making the problem more similar to SALBP-I and strengthening SALBP-I derived lower bounds, and the identification of task pairs that cannot share stations, including station bounds for tasks, which eliminate impossible setup times (Scholl et al., 2013; Esmailbeigi et al., 2016). Relevant preprocessing procedures are explained in Section 2.2.

1.1.1. Exact algorithms

State-of-the-art exact approaches for the SALBP-I are based on branch and bound and can solve instances with up to 100 tasks usually in short running times, and instances with 1000 tasks and specific distributions of operation times can also be easily solved (Morrison et al., 2014), while most exact methods for the SUALBSP rely on mixed integer programming (MIP) approaches, either through off-the-shelf commercial solvers (Esmailbeigi et al., 2016) or logic-based Benders decomposition (Zohali et al., 2022).

MIP models have been proposed by Andrés et al. (2008), Esmailbeigi et al. (2016), and Scholl et al. (2013). Andrés et al. report solving optimally 12 of 160 instances in about half an hour, Scholl et al. have similar optimality rates, but in only 100 seconds, and a solution quality on the remaining instances that is worse than what can be obtained with very simple heuristics. Model SSBF from Esmailbeigi et al. constitutes the state of the art, but only solves about 40% of the instances with 32 to 58 tasks in half an hour and has scaling problems since the number of variables is cubic in the number of tasks.

Column generation and Benders decomposition are two prevalent MIP-based methods for assembly line balancing. These approaches separate the problem into a master problem and one or more subproblems. In a column generation approach, the subproblems identify feasible station loads, and the master problem selects the optimal subset of loads. This approach has been commonly used to generate tight lower bounds for different line balancing problems (Peeters and Degraeve,

2006; Pereira et al., 2018), and is limited to small-sized instances when trying to optimally solve the SALBP-I (Rivera Letelier et al., 2022).

When Benders decomposition is applied to assembly line balancing, the master problem searches for a solution, while the subproblems either confirm feasibility and optimality of the current solution of the master problem, or produce what is called a Benders (feasibility or optimality) cut, which excludes from the solution space of the master problem infeasible or suboptimal solutions. Note that the original method (Benders, 1962) applies to linear programs, but can be extended to non-linear master and subproblems (e.g., Geoffrion, 1972). In particular, the variant known as logic-based Benders decomposition (Hooker and Ottoson, 2003) is often applied to type-II assembly line balancing problems, as evidenced by a recent survey dedicated exclusively to this topic (Michels and Sikora, 2022). Such a strategy has been used by Zohali et al. (2022) and Young (2017) for the type II version of the SUALBSP. Notwithstanding the fact that these methods are superior to directly solving MIP models, they remain limited to medium-sized instances, and fail to solve to optimality some instances with less than 35 tasks.

More recently, Zhang and Beck (2025) proposed constraint programming and dynamic programming based formulations for both the type I and type II SUALBSP. The dynamic programming formulations are then solved through a general complete anytime beam search method (Zhang, 1998) that acts both as a heuristic or as an exact method if sufficient memory and time is allotted. The dynamic programming based approach outperforms both the constraint programming as well as previous MIP models and thus it can be considered as the state-of-the-art exact method for the SUALBSP.

1.1.2. Heuristic methods

Heuristics for ALBPs with setup times are mostly constructive, or contain a constructive component, e.g., as a method to evaluate a solution encoding within a metaheuristic approach. They mostly follow previous heuristics for other ALBPs in using station-based assignment procedures (SBAP) that build solutions station by station. These can be grouped into methods that repeatedly assign a single task to the current station, and open a new station if no more tasks can be assigned, and methods that assign complete station loads. Such procedures are mainly characterized by the rules they use to decide which task or station load to assign next.

These rules can be fully derived from other ALBPs (Talbot et al., 1986), or include setup times within their logic (Andrés et al., 2008; Martino and Pastor, 2010; Scholl et al., 2013). A special case of an assignment rule is any total order among tasks that complies with precedence constraints. These SBAPs can be embedded within a greedy randomized adaptive search procedure, GRASP (Andrés et al., 2008; Scholl et al., 2013), composited into weighted combinations of simpler rules (Martino and Pastor, 2010), or combined with a local search procedure to improve the quality of the solutions (Andrés et al., 2008; Martino and Pastor, 2010; Scholl et al., 2013). Such a local search can be further improved by optimizing the sequence of operations within a station load during the construction phase, aiming to reduce station time and thus allowing for the inclusion of additional tasks to the station load.

A similar idea to the GRASP approach of Scholl et al. (2013) was pursued by Delice (2019) who proposes a genetic algorithm that optimizes over sequences of rules, instead of sequences of tasks. Given a candidate solution, an assignment is built task by task, where the rule prescribed by the solution for each position is used to allocate the task of the highest priority.

Note that the SUALBSP can be tackled by some general solution approaches for GALBPs after some modifications. Scholl et al. (2013) explain the changes needed for Avalanche (Boysen and Fließner, 2008) to solve the SUALBSP, and Sternatz (2014) details simple changes to adapt a previously proposed variant of the Hoffmann heuristic (Hoffmann, 1963) – a truncated enumeration commonly used for the SALBP-I – to address setup times, but does not report results nor some of the modifications required to completely enumerate sequences within station loads.

Table 1
Notation used in this paper.

$[i, j], [i]$	For integers i, j , $[i, j] = \{i, i+1, \dots, j\}$, and $[i] = [1, i]$.
n	Number of tasks.
\bar{m}	Upper bound on the number of stations.
$V = [n]$	Set of all tasks.
$t_i \geq 0$	Task time of task $i \in V$.
$t(S) = \sum_{i \in S} t_i$	Total task time of tasks $S \subseteq V$.
c	Cycle time.
τ_{ij}	Forward setup time between tasks i and j .
μ_{ij}	Backward setup time between tasks i and j .
$<$	Strict partial order among tasks.
$P_i^* = \{j \mid j < i\}$	Set of all predecessors of task $i \in V$.
$F_i^* = \{j \mid i < j\}$	Set of all followers of task $i \in V$.
$\pi_s(i)$	Position of task i in sequence s .
P_i	Set of all immediate predecessors of task $i \in V$, $P_i = \{j \in P_i^* \mid F_j^* \cap P_i^* = \emptyset\}$.
F_i	Set of all immediate followers of task $i \in V$, $F_i = \{j \in F_i^* \mid F_i^* \cap P_j^* = \emptyset\}$.
$I_i = V - F_i^* - P_i^* - \{i\}$	Set of tasks that are independent from task $i \in V$ (i.e., tasks that neither precede nor succeed task i).
$C(S)$	Compactification of set S , i.e., set of tasks that need to be performed together with tasks in S if they share a station.
$LC(S)$	Lower bound of the number of workstations required to perform tasks of set S (e.g., $\lceil t(S)/c \rceil$, see Section 3).
E_i	Lower bound on the earliest station of task i (i.e., $LC(P_i^* \cup \{i\})$).
$L_i(m)$	Upper bound on the latest station of task i for a solution with m stations (i.e., $m+1 - LC(F_i^* \cup \{i\})$).
FS_i	Set of feasible stations for task $i \in V$, $FS_i = [E_i, L_i(\bar{m})]$.
C_i	Set of tasks that can share a station with task $i \in V$, $C_i = \{j \in V \mid FS_i \cap FS_j \neq \emptyset\}$.
$F_i^r (P_i^r)$	Set of tasks that may directly follow task $i \in V$, $F_i^r = (F_i \cup I_i) \cap C_i$, or precede it, $P_i^r = (P_i \cup I_i) \cap C_i$, in forward direction.
$F_i^\mu (P_i^\mu)$	Set of tasks that may directly follow task $i \in V$, $F_i^\mu = (P_i \cup I_i) \cap C_i$, or precede it, $P_i^\mu = (F_i \cup I_i) \cap C_i$, in backward direction.

1.2. Contributions and outline of the paper

In this work, we focus on the SUALBSP, considering different previous and novel lower and upper bounds for its resolution with a focus on the adaptation of methods that have been proved to provide state-of-the-art procedures for SALBP-like problems. This includes machine scheduling (Johnson, 1988) and column generation-based (Peeters and Degraeve, 2006) lower bounds, modified versions of the Hoffmann heuristic (Fleszar and Hindi, 2003; Sewell and Jacobson, 2012; Sternatz, 2014), and a cyclic best-first search enumeration method (Sewell and Jacobson, 2012). We chose to focus our study on the SUALBSP, as it subsumes the GALBPS and, while both SDALBP and SUALBSP occur in certain settings, it is more common for setup times to apply between consecutive tasks.

The remainder of the paper is structured as follows: Section 2 formalizes the SUALBSP, states some assumptions, and introduces some preprocessing and reduction rules for the problem. In addition to previously introduced rules, novel solitary tasks detection and setup time exclusion rules are proposed. Section 3 introduces a linear programming-based relaxation of the problem to compute tight lower bounds. Section 4 describes a truncated enumeration Hoffmann heuristic (Hoffmann, 1963) and a cyclic best-first search method (Sewell and Jacobson, 2012). Section 5 presents experimental results for these methods, and Section 6 concludes. Moreover, the source code of the proposed methods and detailed experimental data is available at <https://github.com/mrpritt/SUALBSP> for further investigation.

Table 1 summarizes the notation for convenience.

2. The SUALBSP

2.1. Problem definition and notation

The SUALBSP has two additional sets of parameters over the SALBP-I. For each pair of tasks $i, j \in V$, we have forward setup times τ_{ij} for regular dependencies (i.e., among consecutive tasks), and backward setup times μ_{ij} for last-to-first dependencies. Let $S^k \subseteq V$ be the subset of tasks performed by station k , its station load, and s^k be the sequence in which these tasks are performed, and let $\pi_{s^k}(i)$ be the position in which task i is performed in sequence s^k . The SUALBSP-I aims to find the minimum number of stations, m , such that the following conditions are satisfied:

1. **Task assignment:** Each task is assigned to exactly one station, i.e., $S^1 \cup \dots \cup S^m = V$ and $S^k \cap S^{k'} = \emptyset$, $\forall k, k' \in [m]$.
2. **Precedence relations:** For each pair of tasks $i, j \in V$, if $i < j$, then i is performed before j . That is, if $i \in S^k$ and $j \in S^{k'}$, then either $k < k'$, or $k = k'$ and $\pi_{s^k}(i) < \pi_{s^k}(j)$.
3. **Cycle time:** The total time of tasks assigned to each station does not exceed the cycle time, i.e., for all $k \in [m]$ we have

$$\sum_{i \in S^k} t_i + \sum_{1 \leq i < j \leq |S^k|} \tau_{s_i^k, s_{i+1}^k} + \mu_{s_{|S^k|}^k, s_1^k} \leq c. \quad (1)$$

A useful assumption, valid in almost all practical problems, is that the insertion of a task into an existing station does not decrease the station time (Scholl et al., 2013). This is the case if the following *triangle inequalities* hold,² for all triplets $i, j, h \in V$.

$$\tau_{ih} + t_h + \tau_{hj} \geq \tau_{ij} \quad (2)$$

$$\tau_{ih} + t_h + \mu_{hj} \geq \mu_{ij} \quad (3)$$

$$t_h + \tau_{hi} + \mu_{jh} \geq \mu_{ji} \quad (4)$$

If the triangle inequalities are satisfied,

$$t_i + \mu_{ii} \leq c, \quad \forall i \in V, \quad (5)$$

holds (otherwise, the instance is infeasible, since there are tasks that do not fit on a station even when alone). Please note that if the triangle inequalities do not hold, adding a task to a station load may accelerate work (i.e., reduce the total time). Although the triangle inequalities hold in practically all settings, we also consider the case where they do not, in order to compare our approach with previously proposed methods on benchmark instances that do not satisfy these inequalities.

The SUALBSP (like the SALBP) is reversible (Scholl et al., 2013); that is, one can reverse all precedence relations and setup times in an instance, solve this instance and derive a solution applicable to the original instance. This property is used in various methods for computing both lower and upper bounds.

² Scholl et al. (2013) require only (2) and (3). This neglects case (4) of inserting a task h as the first task to a station. Inserting task h to a station that holds only a single task i is covered by setting $j = i$ in (3) or (4) depending on whether task h is inserted as the second or the first to the station.

2.2. Problem reductions

In this section, we discuss preprocessing and reduction rules for the SUALBSP. These often make the problem smaller and help to improve the lower bounds discussed in the next section.

We first discuss increasing task times as described in Scholl et al. (2013). Assume, for task $i \in V$, that the minimum setup time to any potential consecutive task

$$\omega_i = \min\{\min\{\tau_{ij} \mid j \in F_i^r\}, \min\{\mu_{ji} \mid j \in F_i^\mu\}\} \quad (6)$$

is positive. In this case, a part of the setup is independent of the consecutive task, and we can shift this part into the task time by setting $t_i := t_i + \omega_i$, $\tau_{ij} := \tau_{ij} - \omega_i$, for all $j \in F_i^r$, and $\mu_{ij} := \mu_{ij} - \omega_i$, for all $j \in F_i^\mu$. Similarly, if after this transformation

$$\alpha_i = \min\{\min\{\tau_{ji} \mid j \in P_i^r\}, \min\{\mu_{ji} \mid j \in P_i^\mu\}\} \quad (7)$$

is positive, we can shift a necessary setup time of a predecessor into the task time by setting $t_i := t_i + \alpha_i$, and $\tau_{ji} := \tau_{ji} - \alpha_i$, for all $j \in P_i^r$, and $\mu_{ji} := \mu_{ji} - \alpha_i$, for all $j \in P_i^\mu$.

A novel rule we introduce tries to reduce set C_i for each task i , also reducing the sets of predecessors and successors used in (6) and (7), see Table 1. Using a lower bound $LC(S)$ on the stations required to perform a set of tasks S , we can exclude from C_i any task j such that $LC(\{i, j\}) > 1$. A stronger condition can be obtained by observing that some other tasks may need to be assigned to the same station in order to comply with the precedence constraints. Let

$$C(S) = S \cup \{k \mid \exists i, j \in S, k \in (F_i^* \cap P_j^*) \cup (P_i^* \cap F_j^*)\} \quad (8)$$

be the *compactification* of S (i.e., the set of tasks that need to be performed together with tasks in S if they share a station). Then, we can exclude task j from C_i when $LC(C(\{i, j\})) > 1$. The calculation of the bound can rely on any method such as those discussed in the next section.

We finally turn to *solitary* tasks, an extension of a rule proposed for the SALBP-I (Scholl and Klein, 1997) to account for setup times. If the triangular inequalities (2)–(4) hold, then it suffices to test for each task $i \in V$ if any other task $j \in V$ can be added to it. Otherwise, task i is only solitary if no possible predecessor or successor can fit on the same station, namely if the following conditions hold:

$$t_i + \tau_{ij} + t_j > c, \quad \forall j \in F_i^r, \quad (9)$$

$$t_j + \tau_{ji} + t_i > c, \quad \forall j \in P_i^r. \quad (10)$$

Task times of any solitary task i can be increased to the cycle time with no setup times and task i can be removed from set C_j for each task $j \in V$.

3. Lower bounds

3.1. Simple lower bounds

A simple bound follows from the total task time for all tasks, and the fact that all times are non-negative. We have combinatorial bound

$$LM1 = \lceil t(V)/c \rceil, \quad (11)$$

see Baybars (1986) for early references on this bound. Bound $LM1$ can be improved by the following observation. In an optimal solution with m non-empty stations, we need to include exactly $n - m$ forward setups, and m backward setups, which can be added to the total task time. Thus, we define for each task $i \in V$ the smallest forward setup time $\tau_i = \min\{\tau_{ij} \mid j \in V\}$ and the smallest backward setup time $\mu_i = \min\{\mu_{ji} \mid j \in V\}$, and let T_k and M_k be the sum of the k smallest elements of $\{\tau_i \mid i \in V\}$ and $\{\mu_i \mid i \in V\}$, respectively. Then, if

$$t(V) + T_{n-m} + M_m > mc, \quad (12)$$

a solution with m stations is not possible. Therefore, Scholl et al. (2013) propose the improved lower bound

$$LM1S = \min\{m \geq LM1 \mid t(V) + T_{n-m} + M_m \leq mc\}. \quad (13)$$

Counting bounds are another set of simple bounds that carry over from the SALBP-I to the SUALBSP-I (Johnson, 1988; Scholl and Becker, 2006). These bounds divide the set of tasks into subsets that can only share stations with tasks from some of these subsets. Bound $LM2$ counts the number of tasks j with $t_j > c/2$ (since they cannot share stations) plus half the number of tasks with task time $c/2$ (as these tasks can share a station with each other but not with tasks from the first set).

Bound $LM3$ follows $LM2$ but considers tasks with respect to thirds of the cycle time. The bound counts the number of tasks j with $t_j > 2c/3$, half the number of tasks with $c/3 < t_j < 2c/3$, two thirds of the number of tasks with $t_j = 2c/3$ and one third of the number of tasks with $t_j = c/3$. $LM2$ and $LM3$ are identical to their SALBP-I equivalents but may be tighter for the SUALBSP-I due to the preprocessing, which increases task times by the sequence-independent part of the setup times.

3.2. Lower bounds from machine scheduling

A lower bound from the SALBP-I that can be applied in an improved form to the SUALBSP-I is the so-called one machine lower bound, or $LM4$ (Johnson, 1988; Scholl and Becker, 2006). In $LM4$ each task is a job that requires an operation on a machine. The machine can perform only one operation at a time, and the task time of the job equals the fraction of a production cycle required by the task, i.e., $p_i = t_i/c$. As the machine must process each job, the optimal makespan is $\sum_{i \in [n]} p_i = t(V)/c$. The makespan can be seen as the work or the number of stations required to perform these tasks and, based on the fact that the number of stations is a non-negative integer, this bound equals $LM1$, i.e., $\lceil t(V)/c \rceil$.

Bound $LM4$ can be tightened by considering that tasks cannot commence or conclude until their predecessors/successors have been completed, building upon the non-divisibility of tasks. This consideration results in specific release and delivery times for each task, referred to as the head and tail of each task, which correspond to non-necessarily integral bounds on the work, in number of stations, i.e. normalized by c , required to process the tasks that must precede or succeed a given task.

Taking into account release dates and delivery times leads to problem 1 $\mid r_j, q_j \mid C_{max}$ according to the machine scheduling three index notation, which is NP-hard (Brucker, 2007). Consequently, the bound is usually further relaxed by ignoring the release dates leading to a problem solvable in time $O(n \log n)$ by ordering jobs in non-decreasing order of delivery times. Let η_i be the delivery time of job i and, w.l.o.g., assume that the jobs are ordered according to their delivery times, i.e., $\eta_1 \geq \eta_2 \geq \dots \geq \eta_n$. Then

$$LM4 = \max_{i \in [n]} \left\{ \sum_{j \in [i]} p_j + \eta_i \right\} \quad (14)$$

is a valid lower bound for the SALBP-I. Moreover, since the number of stations is known to be an integer, the bound can be rounded up when it is fractional.

The delivery times, the tails of the jobs, are obtained by bounding the work required to perform its followers. This can be done, for example, by using (14) for the instance induced by the followers of task i (i.e., set F_i^*). After determining the tail, its value may be rounded up if it can be shown that no fewer than $\lceil \eta_i \rceil$ stations are required after task i is performed.

For the SALBP-I, the condition involves evaluating the number of stations required to perform a task together with its followers. If $\eta_i < \lceil \eta_i \rceil$ and $p_i + \eta_i > \lceil \eta_i \rceil$, then at least $\lceil \eta_i \rceil$ stations are needed after the station that performs task i (the followers of task i require no fewer than $\lceil \eta_i \rceil$ stations, and task i cannot be assigned to the first of these).

On the other hand, if $p_i + \eta_i \leq \lceil \eta_i \rceil$, task i may be performed on the first of the $\lceil \eta_i \rceil$ stations required by its followers, and thus no rounding is possible.

We now introduce novel improvements to the lower bound calculation for the SUALBSP, incorporating setup times into the analysis. First, we consider the case where the *triangle inequalities* hold.

For the SUALBSP, the condition for rounding up the tail of any given task i must not only account for its task time p_i , but also for the time by setups involving task i . When the *triangle inequalities* hold, adding a task to the computation cannot reduce the total time required by the tasks and their associated setups. Consequently, only the setup times involving task i and its followers need to be considered. Let $\tau_i^F = \min_{j \in F_i} \tau_{ij}/c$ be the minimum fraction of the station time required to perform a forward setup between task i and any of its immediate followers, and $\mu_i^{F*} = \min_{j \in F_i^*} \mu_{ji}/c$ be the minimum fraction of the station time required to perform a backward setup time between any of the followers of task i to the task itself. Using this notation, the tail of the tasks can be updated according to (15).

$$\eta_i = \begin{cases} \lceil \eta_i \rceil, & \text{if } p_i + \eta_i + \tau_i^F + \mu_i^{F*} > \lceil \eta_i \rceil, \\ \eta_i + \tau_i^F, & \text{otherwise.} \end{cases} \quad (15)$$

Given that task i precedes all tasks of its tail, the *triangle inequalities* ensure that adding another task does not reduce the total time required by a subset of tasks. Since only one of the two setups involving task i can be a forward setup, the other setup must be a backward setup. The fractions of station time required by these setups are bounded by τ_i^F , since the forward setup must involve an immediate follower, and μ_i^{F*} , since the backward setup may involve any follower, respectively.

Then, when $p_i + \eta_i + \tau_i^F + \mu_i^{F*} > \lceil \eta_i \rceil$, as for the SALBP-I, at least $\lceil \eta_i \rceil$ stations are required after the station that performs task i , since i and its followers cannot be performed on $\lceil \eta_i \rceil$ stations. If, on the other hand, $p_i + \eta_i + \tau_i^F + \mu_i^{F*} \leq \lceil \eta_i \rceil$, the tail can be strengthened based on the following observation: either task i is the last task executed on its station (in which case no fewer than $\lceil \eta_i \rceil$ stations must follow), or it is immediately followed by a forward setup before any subsequent task is performed. In this case, when the *triangle inequalities* hold, the fraction of time required for the forward setup can be bounded by τ_i^F , and the tasks in the tail require at least η_i stations. Combining both scenarios, we conclude that the tail is no less than $\min\{\eta_i + \tau_i^F, \lceil \eta_i \rceil\} = \eta_i + \tau_i^F$.

If triangle inequalities do not hold, including a task may lead to a reduced total time. Consequently, we use $\tau_i^{F*} = \min_{j \in F_i^*} \tau_{ij}/c$ and $\mu_i^{F\mu} = \min_{j \in F_i^*} \mu_{ji}/c$ instead of τ_i^F and μ_i^{F*} in (15).

An alternative bound can be obtained by considering the reversibility property of the SUALBSP. In the reverse instance, the delivery times correspond to the time required to perform the predecessors of the tasks and their respective setup times. These correspond to the release date, heads, of the tasks and are denoted as a_i for each task $i \in V$.

Finally, and in addition to lower bound calculation, the heads and tails of a task can be used to ascertain the earliest and latest possible stations used by mathematical programming formulations (see Scholl and Klein, 1997; Vilà and Pereira, 2013, for similar ideas used for lower bound and variable prefixing calculations within implicit enumeration approaches),

$$E_i = 1 + \lfloor a_i \rfloor; \quad L_i(\bar{m}) = \bar{m} - \lfloor \eta_i \rfloor. \quad (16)$$

3.3. A column generation-based lower bound

3.3.1. A set partition formulation

An alternative approach to derive a lower bound is to examine the linear relaxation of a set partition based formulation of the problem (see Peeters and Degraeve, 2006, for the SALBP-I). We call this lower bound LMDW as it follows a Dantzig–Wolfe decomposition approach.

In this formulation, variables correspond to station loads with constraints enforcing task assignment and precedence constraints among

tasks assigned to different stations. The precedence constraints only make the formulation more challenging to solve without offering significantly improved bounds (Pereira et al., 2018), hence, the precedence constraints are removed from the master problem and considered exclusively in the pricing problem. As a consequence station loads no longer need to be associated with a specific station. Thus, letting $\lambda_1, \dots, \lambda_k$ be an enumeration of all feasible station loads and introducing variables x_j , for $j \in [k]$ to identify the selection of station load λ_j we obtain

$$\min. \sum_{j \in [k]} x_j \quad (17)$$

$$\text{s.t.} \sum_{j \in [k]: i \in \lambda_j} x_j = 1, \quad \forall i \in V, \quad (18)$$

$$0 \leq x_j \leq 1, \quad \forall j \in [k]. \quad (19)$$

where variables x_j have already been relaxed. The number of variables of the formulation is exponential in the number of tasks, and thus, it is necessary to iteratively solve pricing problems to find station loads that satisfy precedence constraints among their tasks, and whose station time respects the cycle time until no additional station loads modify the solution of the relaxation.

The pricing problem is to find feasible station loads, subsets of tasks L , of minimal reduced cost $1 - \sum_{i \in L} \pi_i$ where π_i are the dual variables of constraint set (18). Station load feasibility involves the following three conditions:

1. The tasks in L can be assigned to the same station, i.e., the intersection of feasible stations among them is not empty ($\bigcap_{i \in L} \text{FS}_i \neq \emptyset$).
2. Set L must be compact, i.e., $L = C(L)$.
3. The cycle time is observed, i.e., there exists a sequence s of the tasks in L such that

$$c \geq t(L) + \mu_{s_1, s_1} + \sum_{1 \leq i < j \leq |L|} \tau_{s_i, s_{i+1}} \quad (20)$$

holds, and sequence s complies with the precedence constraints, namely there is no pair $i, j \in L$ with $i < j$ such that $\pi_s(j) < \pi_s(i)$.

The first two conditions are not needed if the master problem enforces precedence constraints, but they are conditions that any station load within a feasible solution for the problem must satisfy. They are easy to impose within our procedure and reduce the number of variables within the formulation, hence we opt to enforce them in our pricing problem.

The third condition is the most challenging to solve, as it involves the main two features of the pricing problem: (1) it imposes as a knapsack-like constraint on the reduced cost of feasible station loads, and (2) for any given first task in the sequence, checking the condition is equivalent to solving the decision version of a sequential ordering problem (SOP, Escudero (1988)), namely to decide if there is a Hamiltonian path subject to precedence constraints of total length at most $c - t(L)$ with arc lengths according to setup times, either forward (from any task to another task) or backward (from any task to the first task in the sequence). So the condition can be checked by systematic evaluation of all candidates for the first task, namely all tasks in L without predecessor in L , and solving its corresponding SOP.

Algorithm 1 outlines the complete bounding procedure, including the iterative procedure for the master problem, function **computeBound**, and the recursive enumeration procedure for the pricing problem, function **enumerate**.

3.3.2. Resolution of the pricing problems

We tested solving the pricing problem optimally, but found its solution on large instances inefficient, due to its complexity. Therefore, we further relax the SOP component of 3 for a given first task $i \in L$ to an assignment problem (AP), which can be solved in polynomial time (Burkard et al., 2012). In the AP, each task is assigned to its

Algorithm 1: Computation of lower bound LMDW

```

1 function enumerate( $T, L, \bar{L}$ )
   Input : Candidate tasks  $T$ , included tasks  $L$ , excluded
         tasks  $\bar{L}$ 
   Output: Set of station loads  $A$ 
2 if earlyTermination() then return
3 foreach  $i \in T$  in non-increasing order of  $\pi_i/t_i$  do
4    $L' := \text{makeCompatible}(L, i, \bar{L})$  // check compactness
5   if  $L' \neq \emptyset$  then // is compatible?
6      $L := L \cup L'$ ;  $T := T \setminus L'$  // Update sets
7     if  $t(L) \leq c$  then // cycle time test
8       if bound( $T, L, \bar{L}$ ) then // may improve
9         incumbent?
10        if improves( $L$ ) then  $A := A \cup L$ 
11        enumerate( $T, L, \bar{L}$ ) // inclusion path
12     $L := L \setminus L'$ ;  $T := T \cup L' \setminus \{i\}$ ;  $\bar{L} := \bar{L} \cup \{i\}$ 
13    if bound( $T, L, \bar{L}$ ) then // may improve incumbent?
14      enumerate( $T, L, \bar{L}$ ) // exclusion path
15 function computeBound()
16   Find initial solution and initialize relaxation
17   while not timeLimit do
18     Solve relaxation (17)–(19)
19     enumerate( $V, \emptyset, \emptyset$ )
20     if  $A = \emptyset$  then return bound
21     updateBound()
22     if earlyTermination() then return bound
23     Add station loads in  $A$  to relaxation
24   Solve final knapsack problem and update bound
25   return bound

```

follower, and assigning task i to j costs τ_{ij} except for $j = l$, which has cost μ_{ij} . Note that relaxing the condition and thus the pricing problem enlarges the subset of feasible station loads, and therefore still produces a valid, but weaker, lower bound.

The relaxed pricing subproblems are then solved by branch and bound, where branching is performed by inclusion or exclusion of a task in L . Minimizing reduced costs $1 - \sum_{i \in L} \pi_i$ is equivalent to maximizing $\sum_{i \in L} \pi_i$, and this objective function is bounded by solving the linear relaxation of a knapsack problem. We next give an outline of the enumeration procedure and then explain these steps in detail.

The enumeration, lines 1–13, is structured as follows. Tasks are queued up for branching in non-increasing order of their ratio of dual values to task times, π_i/t_i , giving preference to short tasks with high dual values. Upon selecting a task for branching, line 3, we first consider its *inclusion*, lines 4–10, and then its *exclusion*, lines 11–13, from the station load. Inclusion first checks by calling function **makeCompatible** which tasks to include in the station load to ensure Conditions . This function finds the compactification of the set of tasks L and i , expression (8), ensures that all the tasks in the compactification can share a station with each other, $\bigcap_{i \in L'} FS_i \neq \emptyset$, and no task in the compactification has been excluded in previous branches, $L' \cap \bar{L} \neq \emptyset$. If any of these conditions is not met, the function returns an empty set and the branch is pruned, line 5, or the set of tasks, together with task i , needed for a compact assignment, i.e., $C(S \cup \{i\}) \setminus S$, is included.

Then, we check the cycle time constraint without considering setup times, i.e., whether $t(L) \leq c$ holds, and prune the branch if it does not hold. Otherwise, we bound the value of the reduced cost solving a relaxation of the problem as follows: let $c - t(L)$ be the capacity of a knapsack problem with the subset of the candidate items in T (i.e., the subset of tasks that have not been *included* nor *excluded*) that can share a station with each selected task in L and have task times no larger than the capacity of the knapsack. Each such item i gets a weight equal to

its task time t_i and a value equal to its dual value π_i . Then, we solve the continuous knapsack problem (i.e., we relax integrality constraints, precedence constraints and setup times). This can be done in time $O(n)$, since the tasks are already ordered by profit per weight. If the sum of dual values of all tasks in L plus the knapsack bound is not better than the value of the incumbent, we prune the branch (line 8). We also tested methods to include setup times and reduce the capacity of the knapsack relaxation but found them to be computationally inefficient.

If the branch has not been pruned, function **improves** (line 9) checks if the current assignment is better than the incumbent and if the relaxed 3 is satisfied for some candidate initial task i . For candidate task i , the condition is satisfied if the sum of the task time $t(L)$ plus the best AP bound on the setup times is at most the cycle time. Finally, the *exclude* branch, lines 11–13, performs similar steps but only checks for the lower bound of the relaxation before branching.

The implementation stores all station loads found during the search that improve the incumbent, line 9, and adds them to the master problem after the pricing step. This reduces running time as it improves the linear relaxation value of the master problem over including a single station load in each pricing step without significantly increasing the time required to solve each linear program. To avoid complete enumeration of the pricing subproblem solution space, we stop when we find a variable whose reduced cost is deemed as sufficiently negative. This is checked in function **earlyTermination** in line 2. In our implementation, when the reduced cost is smaller than an upper bound $\epsilon_r > 0$; i.e., pricing is larger than $1 + \epsilon_r$, where ϵ_r is a parameter of the algorithm, we stop the search.

3.3.3. Resolution of the master problem

Pricing is embedded within the master problem solution process, function **computeBound**, lines 14–24. The master problem is initialized with the station loads from a heuristic solution, and then iteratively solves restricted master problems and pricing problems until a termination condition is reached. After each pricing step, line 20, we update the lower bound of the relaxation as described in Lübbecke (2011). Let z_{min} be the optimal solution of the relaxation, z_B be the optimal solution to the master problem over the current subset of columns, and \bar{c}_{min} be the value of the optimal solution of the pricing problem. Then, $z_{min} \geq \frac{z_B}{1 - \bar{c}_{min}}$ holds and, since the solution is integer, $\lceil \frac{z_B}{1 - \bar{c}_{min}} \rceil$ is a valid lower bound. If pricing stops without an optimal reduced cost, \bar{c}_{min} is approximated by solving a continuous knapsack relaxation of the pricing subproblem that considers all tasks, their dual costs and the cycle time capacity constraint.

The relaxation is stopped when a time limit is reached, line 16, no columns with negative reduced cost are found, line 19, or if there is no potential for further enhancing the relaxation bound. This last condition is met when $\lceil z_B \rceil$ matches the best lower bound from previous iterations (Vance et al., 1994).

Finally, if the relaxation is stopped due to the time limit, we perform a final attempt to improve the lower bound. This is accomplished by bounding the optimal reduced cost for the present restricted master problem by solving a modified 0–1 knapsack problem where the smallest backward setup time from one of the chosen tasks, combined with the smallest forward setup time from the other selected tasks, is integrated within the knapsack constraint. Preliminary tests show that the modified knapsack model is easily solvable within seconds and offers a chance to enhance the final lower bound on large instances.

4. Upper bounds

We now discuss two solution approaches based on the enumeration of station loads, namely a Hoffmann-like heuristic and an extension of a station-based cyclic best-first search. The latter can be used either heuristically or as an exact solver. To the best of our knowledge, this is the first exact solver of the SUALBSP which is not based on mathematical formulations.

4.1. Hoffmann-type heuristics

The current best heuristic procedures for the SUALBSP are station-based assignment procedures (SBAP). In an SBAP, we repeatedly select a task whose predecessors have been assigned already, that fits into the current station, and assign it to the station. If no such task exists, but not all tasks are assigned, a new station is opened. Hoffmann-type heuristics, unlike SBAP methods, select complete station loads instead of single tasks in each step. To this end, they have to enumerate station loads and apply a selection rule to choose one of them to allocate to the current station.

Algorithm 2: Enumeration of station loads

```

1 function addTask(L,a,i)
2   a := push(a,i); L := L ∪ {i}
3   if SV(a) < SV(a*) then a* := a           // update best
4   foreach j ∈ Fi* do
5     kj := kj - 1
6     if kj = 0 then L := L ∪ {j}
7 function removeTask(L,a,i)
8   a := pop(a); L := L ∪ {i}
9   foreach j ∈ Fi* do
10    if kj = 0 then L := L ∪ {j}
11    kj := kj + 1
12 function enumerateRec(L,a)
13   foreach i ∈ L do
14     if fits(push(a,i)) then                // checks cycle time
15       numAssignments := numAssignments + 1
16       if numAssignments < maxAssignments then
17         addTask(L,a,i)
18         enumerateRec(L,a)
19         removeTask(L,a,i)
20 function enumerate(T) // Set of unassigned tasks T
21   ki := |Pi* ∩ T|, ∀ i ∈ V                // number of pending
    predecessors
22   L := {i ∈ T : ki = 0}; a := () // Candidates and load
23   enumerateRec(L,a)
24   return a* // Return best assignment
  
```

Algorithm 2 outlines the enumeration of station loads. The method differs from SALBP-I Hoffmann heuristics by taking into account setup times within the enumeration, function `fits`. The station load under construction is represented by a stack a . The method follows a recursive approach to enumerate all station loads, starting at line 21 by initializing the subset of tasks L with no unassigned predecessors. During enumeration, the algorithm checks whether a task can be added to the current station load according to task times and setup times, line 14. If the task fits, it is added to the load, the best station load is updated if needed, line 3, and the algorithm proceeds considering additional tasks to add to the station load.

The enumeration of station loads depends on the method used to select the best station load, function `SV` (for station value) in line 3, and the order in which stations are considered. Similarly to SBAP, Hoffmann-type heuristics can be applied in a forward manner over all stations, in a backward manner, bidirectionally, or in combinations thereof. We consider the following variants:

- allocation in a forward manner that selects the load of maximum total time (as in Hoffmann, 1963) ($SV(U) = -t(U)$, where U is the set of unassigned tasks);
- allocation in a forward manner that selects the load of minimum $SV(U) = \sum_{j \in U} (100t_j + w_1PW_j + w_2|F_j| - w_3)$ over all unassigned tasks U , for all combinations of weights with $w_1, w_2 \in \{0, 0.5, 1, 1.5, 2\}$, and $w_3 \in \{0, 1, 2, 3\}$ (following Sewell and Jacobson, 2012);

- allocation in a bidirectional manner (as in Fleszar and Hindi, 2003). For each $k \in [\bar{m}]$, allocate k stations in the forward direction, and allocate unassigned tasks in backward direction, selecting (both in forward and backward direction) the load of maximum total time;
- allocation in a bidirectional manner, selecting the load of minimum $SV(U)$, with w sampled randomly according to $w_1 \sim U[0, 1]$, $w_2 \sim U[0, 1]$, and $w_3 \sim U[0, 1]/n$ (Sternatz, 2014). As the method may provide different solutions for different sets of weights w , we impose a time limit, an algorithmic parameter, to the method.

These variants of the Hoffman heuristic are referred to, respectively, as H, SJ, FH and S (the first letter of each author surname from the original publication) in the computational experiments.

Since there can be many station loads, we limit the enumeration to a maximum number of partial or full station loads for each station controlled by a parameter of the algorithm. Thus the selected station load corresponds to the best among those generated according to their selection rule (line 16). This limit follows Sewell and Jacobson (2012) to avoid exhaustive enumeration of all feasible loads for each station.

4.2. A cyclic best-first search method

Hoffmann-type heuristics are truncated enumeration approaches in which the branching factor is limited to avoid complete enumeration. For instance, the original Hoffmann heuristic truncates the full enumeration of solutions by selecting only one station load among all candidate loads, and then proceeds to enumerate station loads for another station. An alternative is to consider more than one station load per station.

This approach was used in Sewell and Jacobson (2012), Morrison et al. (2014) for the SALBP-I, and we show here how it can be extended to the SUALBSP (specifically, changes to dominance rules, use of alternative lower bounds as well as an extended equivalence rule). The approach in Sewell and Jacobson (2012) is divided into three phases. The first phase finds an initial upper bound through a Hoffmann-like heuristic, followed by a limited enumeration step. Only if these two phases fail to verify the optimality of the current incumbent solution, complete enumeration ensues.

Limited enumeration follows a station-based branching enumeration scheme. It starts from an empty solution and generates a maximum number of candidate station loads, similar to Hoffmann-like heuristics (the limit is an algorithmic parameter). Before adding each candidate station load to a list of open subproblems, lower bounds, dominance, and equivalence tests are applied. The enumeration then continues by selecting a candidate with k stations, as described in Section 4.2.1, generating station loads for station $k+1$ in accordance with the station loads for the first k stations, checking again lower bounds, dominance, and equivalence rules before being added to the list of open subproblems. If this list becomes empty without reaching the enumeration limit, or if a feasible solution is found that meets the lower bound, the method confirms the optimality of the incumbent solution.

We now proceed to describe each of the algorithmic components separately, i.e., candidate selection, dominance, bounds and equivalence tests.

4.2.1. Search strategy

The search strategy is a cyclic best-first search, following the approach proposed in Morrison et al. (2014) for their second phase. The method creates a priority queue for each level of the search tree. In our case there are \bar{m} queues, each containing all unexplored partial solutions with a specific number of closed stations. Then, we scan each priority queue in increasing order to its associated number of stations and select the partial solution with the highest priority for exploration. Each queue prioritizes partial solutions of smaller lower bound, breaking ties by prioritizing partial solutions with the highest total task time for assigned tasks. The search ends when all queues are empty.

4.2.2. Dominance rule

Dominance rules are used to fathom partial solutions that are dominated by others. The maximum load rule applies to the SUALBSP-I if the triangle inequalities hold, i.e., a station load is dominated by another station load that performs a superset of its tasks, Jackson (1956). If triangle inequalities do not hold, however, the rule cannot be applied, because a larger set of tasks could have a smaller station time.

The implementation partially applies this rule by checking if a task can be appended to the station load as the last task to be performed. (A less permissive rule would be to search for some feasible permutation of the tasks at the station, but is computationally too costly.)

4.2.3. Lower bounds

To avoid unnecessary search, we check whether a partial solution can lead to a feasible solution with fewer stations than the incumbent. Our implementation uses lower bounds *LM1* and *LM4* described in Section 3 on an instance over the subset of unassigned tasks.

4.2.4. Equivalence rule

The SUALBSP-I can be formulated through dynamic programming, much like the SALBP-I (Jackson, 1956; Bautista and Pereira, 2009). The states of the formulation correspond to subsets of tasks, $S \subseteq V$, such that $\forall i \in S, P_i^* \subset S$, and transitions correspond to the assignment of subsets of tasks to a workstation according to station load feasibility, i.e., the assignment is compact, all tasks can be assigned to the same station, and the cycle time is observed for a sequence that satisfies precedence constraints. An optimal solution corresponds to the shortest path, i.e., the least number of transitions from the state with no assigned tasks to the state with all tasks assigned.

The state representation can be used to avoid exploring partial solutions with equivalent states (Sewell and Jacobson, 2012; Morrison et al., 2014). This can be achieved within the implicit enumeration through an associative array mapping a state to the number of stations required to reach it from the empty state. Whenever a new partial solution is generated, its equivalent state is generated and stored in the map.

The equivalence rule is then implemented by checking whether the state representation of the current partial solution is in the map. If it is, the current partial solution can be fathomed if the number of stations it needs is not smaller than the stored value. Otherwise, the value in the map is updated and the partial solution is not fathomed.

We also apply an extended equivalence rule (a similar rule was proposed in Álvarez-Miranda et al. (2024) for the SALBP-II). This rule checks whether a state with a superset of tasks and the same or fewer stations has been constructed. Specifically, for each partial solution, we check if a partial solution with one additional task is stored in the map. The additional tasks to test are those whose immediate predecessors have been assigned to some station in the current partial solution. While this operation consumes more time than the basic equivalence rule, it overcomes some of the limitations of the dominance test due to the setup times. This rule is only valid and, thus, applied if the triangle inequality is satisfied.

4.2.5. Implementation details

Algorithm 3 outlines the cyclic best-first search method. The method starts by initializing the upper bound using the Hoffmann-like heuristic, Section 4.1, as it finds the best initial solutions according to our experiments. Then, it initializes the priority queues, adding an empty partial load to the first priority queue, and an empty map.

The search continues cyclically, choosing a priority queue as described in Section 4.2.1. It then removes a partial solution from the selected queue, expands it through enumeration, and adds the generated station loads to the priority queue in the next level. The algorithm stops when all priority queues are empty, or the time limit of the algorithm is reached.

Algorithm 3: Outline of the cyclic best-first search

```

1 function enumerate( $T$ ) // Described in Algorithm 2
2 function addTask( $L, a, i$ )
3    $a := \text{push}(a, i); L := L \setminus \{i\}$ 
4   if jackson( $a$ ) then
5      $\text{numLoads} := \text{numLoads} + 1$ 
6     if  $\text{numLoads} > \text{maxLoads}$  then exit enumerate
7     if  $\text{lb}() < \bar{m}$  and equivalence() then
8       Append  $a$  to list of station loads
9   foreach  $j \in F_i^*$  do
10     $k_j := k_j - 1$ 
11    if  $k_j = 1$  then  $L := L \cup \{j\}$ 
12 function cbfs()
13   Set  $\bar{m}$  through Hoffmann-like heuristic  $S$ . // Section 4.1
14    $Q_0 := \{\emptyset\}, Q_l := \emptyset, \forall l \in [m-1]$  // Initialize queues
15   while not timeLimit do
16     if  $\sum_{l \in [0, \bar{m}-2]} |Q_l| = 0$  then return best
17     foreach  $l \in [0, \bar{m}-2]$  do // cyclically pop
18       expand
19       if  $|Q_l| > 0$  then
20         enumerate( $V \setminus Q_l.\text{pop}()$ )
21         foreach station load do
22           if completeSolution then Update best
23           else Add station load to queue  $Q_{l+1}$ 
23   return best

```

The enumeration generates up to a maximum number of candidate station loads using the procedures outlined in Algorithm 2 for Hoffmann-like heuristics, with the exception of function **addTask**.

After a candidate station load is found to be maximal, we check if it generates a complete assignment of tasks to workstations and if so, we update the incumbent solution. If not, we evaluate *LM1* and *LM4*, and then the equivalence rules (Section 4.2.4), functions **lb** and **equivalence** in line 7. If the candidate assignment is not fathomed by either test, we add the partial solution to the station load list and eventually to the corresponding priority queue. The method reports optimality if enumeration terminates with all queues empty, and the limit on the number of candidate station loads is never reached.

If optimality is not proven, we proceed to an additional enumeration phase as described in Sewell and Jacobson (2012), with two major modifications to their approach: (1) the exploration of the search tree continues to follow a cyclic best-first search strategy, rather than the breadth-first strategy used in Sewell and Jacobson (2012); and (2) we increase the limit on the number of candidate station loads, while still maintaining a cap, as preliminary experiments showed that complete enumeration without a limit results in excessive memory requirements.

5. Computational experiments

5.1. Data sets and methodology

The computational experiments use four data sets from the literature. These are the data set AMP of Andrés et al. (2008), the data set MP of Martino and Pastor (2010), and the data sets SBF1 and SBF2 of Scholl et al. (2013). Data set AMP is a subset of the MP dataset, and we use it only to be able to compare to other results that have been reported in the literature for this set. These instances have between 7 and 297 tasks, and order strengths.³ between 22.8% and 83.8%. Data

³ The order strength is the fraction of the number of precedence relations $|\{i < j \mid i, j \in V\}|$ of the maximum possible number $\binom{n}{2}$.

Table 2

For each group of instances: number of instances N , value α , the percentage of instances that satisfies the triangle inequalities Δ , the maximum forward setup time τ^{\max} and backward setup time μ^{\max} as a fraction of the average processing time.

Set	α	N	Δ [%]	τ^{\max}	μ^{\max}
SBF1	0.25	269	100.0	0.08	0.25
SBF1	0.50	269	100.0	0.17	0.50
SBF1	0.75	269	100.0	0.25	0.73
SBF1	1.00	269	100.0	0.33	0.99
SBF2	0.25	269	100.0	0.09	0.18
SBF2	0.50	269	100.0	0.18	0.32
SBF2	0.75	269	100.0	0.26	0.32
SBF2	1.00	269	100.0	0.36	0.32
AMP		160	100.0	0.11	0.11
MP		640	55.3	0.46	0.43

sets SBF1 and SBF2 come in four groups with a different value for $\alpha \in \{0.25, 0.5, 0.75, 1.0\}$, where α is the upper limit of the forward setup times as a fraction of the average task time. The SBF2 data set has been constructed such that the optimal SALBP-I solution remains optimal, so for these instances the optimal objective function values are known. All instances are available at [ALB Research Group \(2011\)](https://github.com/mrpriit/SUALBSP)

Table 2 summarizes the data sets and their main characteristics. It reports the number of instances N per set, the percentage of instances Δ that satisfy the triangle inequalities (2)–(4), and the maximum fraction τ^{\max} and μ^{\max} of forward and backward setup times of the average task time. These averages are after preprocessing, so infeasible setups are excluded. Excluded setups tend to be large, which explains why τ^{\max} values are smaller than the expected value $\alpha/2$.

All tests have been run on a PC with an AMD Ryzen 7 8845HS at 3.8 GHz, with 32 GB of main memory, running Debian Linux 12. Each test has been run with a single thread. The detailed experimental results and source code are available at <https://github.com/mrpriit/SUALBSP>.

Several tables report average relative deviations. The relative deviation of value a from value b is defined as $(a - b)/b$, and is reported in percent.

5.2. Lower bounds

In this section we compare the lower bounds. We limit the comparison to the SBF2 instance set, since for those instances the optimal solution value is known.⁴ Table 3 presents the average relative deviation from the optimal values for several lower bounds, namely for lower bound l and optimal value o , the value $(o - l)/o$ (in percent). Computation times for the combinatorial, counting and machine scheduling lower bounds, are negligible, and are not reported (we refer to these bounds as fast lower bounds). We report results for two model-based bounds, LMDW, Section 3.3 and LMSB, a linear relaxation proposed in [Esmaeilbeigi et al. \(2016\)](#) which is the best-performing bound from the literature. For these bounds, we also report the shifted geometric mean of computation times, with a shift of 1 s,⁵ namely t_{SB} and t_{DW} (in seconds). Column N gives the number of instances in each group. For better preprocessing we have obtained an upper bound through the SBAP MPRule proposed in [Martino and Pastor \(2010\)](#) before computing the lower bounds.

In Table 3 we can see that including setup times in $LM1$ makes no significant difference, since $LM1$ and $LM1S$ are essentially the same. This is due to preprocessing which improves $LM1$ in 17.1% of the

instances. Without preprocessing these bounds are about 1.4% lower. Counting bounds $LM2$ and $LM3$ are outperformed by other methods even if they provide the best bound in a few instances (specifically, most instances with the Wee-Mag precedence graph), while $LM4$ performs the best among the non model-based bounds. Both model-based bounds perform best, but require considerable time to compute. LMDW gives the best bound, and on all except 29 larger instances equals the optimal value.

Since there is no significant advantage of $LM1S$ over $LM1$, we use only $LM1$ and $LM4$ during the cyclic best-first search enumeration phase. Table 3 also introduces two combined lower bounds used in later comparisons. Bound L_f^* is the best over the fast lower bounds, bound L^* is the overall best bound. We can see that L^* essentially coincides with LMDW, while L_f^* primarily stems from $LM4$, except those instances where $LM2$ or $LM3$ outperform all other fast bounds.

5.3. Upper bounds

We next look at the four heuristics based on Hoffmann's idea of enumerating station loads, namely those explained in Section 4.1: forward allocation (H) and bidirectional allocation maximizing time (FH), and forward (SJ) and bidirectional (S) allocation maximizing a weighted sum of three criteria. These heuristics are compared to a reimplementation of the Rule-GRASP (R-G) of [Scholl et al. \(2013\)](#), which is the state-of-the-art heuristic for the SUALBSP in the literature. Note that our reimplementation compares favorably to the implementation of [Scholl et al. \(2013\)](#) mainly due to stronger preprocessing rules. As in the previous experiment, all run the MPRule once to obtain an initial upper bound. All heuristics are limited to 1000 loads per station. Heuristics H, FH, and SJ execute once, heuristic S, which samples weights randomly, is limited to 1000 iterations, and we also have limited the running time to at most 10s. Table 4 shows the average relative deviations from L_f^* , time, and optimality rates (i.e. the number of times the solution is demonstrably optimal due to matching lower and upper bounds).

We can see that all heuristics match or improve over Rule-GRASP. Heuristic H matches its performance, and FH, SJ, and S get better results. These heuristics achieve this by systematic enumeration of station loads, and without reordering of their tasks, and in the case of FH and SJ, in a fraction of the time. Heuristics FH, SJ, and H are very fast, since they execute only once, and do a fixed number of allocations. The best performing heuristic is S, with an average relative deviation of 8.0% and a time comparable to Rule-GRASP.

5.3.1. Cyclic best-first search

We finally turn to the exact search using the cyclic best-first search approach. The exact solver starts from a primal bound obtained by running the Hoffmann-based heuristic S with 100 iterations in both directions and at most 1000 station loads. An initial dual bound has been obtained by computing L_f^* as well as the Dantzig-Wolfe based lower bound LMDW for at most 12s, with a pricing cutoff ϵ_r set to -0.1 based on preliminary tests. The exact solver is run afterwards with a time limit of 120s and a limit of $l = 1500$ candidate station loads. If the enumeration phase ends with time remaining and without verified optimality, we increase limit l to 20 million and restart the search using the remaining time from the previous run as the new time limit. The results are shown in Table 5 along with the results of the method proposed by [Zhang and Beck \(2025\)](#) obtained on the same computer used for our experiments, with a time limit of 1800s.

We find that the optimality rates roughly double compared to Hoffmann-based rules, and within the time limit we can solve 1979 of all 2792 instances to proven optimality. The average running time is about 37s. Relative deviations drop from about 8% to about 1%. An analysis shows that this reduction comes in equal shares from better primal and dual bounds. For the SBF2 instances, where the optimal values are known, we can compare to true relative deviations and optimality rates and find that true relative deviations are about 0.05%

⁴ In three SALBP-I instances the solution used to derive the SUALBSP instances is one station above the optimal SALBP-I value (Dr. Scholl, private communication). They represent only 12 out of 1076 SBF2 instances and we follow [Scholl et al. \(2013\)](#) comparing to these values.

⁵ The shifted geometric mean with shift s of values v_1, \dots, v_n is $(\prod_{i \in [n]} (s + v_i))^{1/n} - s$. It allows to compare values with large ranges, without being dominated by very small or very large values, see e.g. [Achterberg \(2007\)](#).

Table 3

Comparison of combinatorial lower bounds LM1, LM1S, counting lower bounds LM2, LM3, machine scheduling lower bound LM4, the linear relaxation lower bound of the station-based model proposed in Esmailbeigi et al. (2016) LMSB, the linear relaxation lower bound LMDW, the best fast bound L_f^* , and the overall best bound L^* , on instance set SBF2. For each lower bound and instance group we report the average relative deviation from the optimal value.

α	N	LM							L_f^*	L^*	$t(s)$	
		1	1S	2	3	4	SB	DW			SB	DW
0.25	269	4.1	4.1	62.7	47.8	3.4	3.1	0.1	1.5	0.1	15.3	2.0
0.50	269	4.0	4.0	62.7	47.8	3.2	3.0	0.1	1.5	0.0	15.3	3.5
0.75	269	4.0	4.0	62.7	47.8	3.2	3.3	0.1	1.5	0.1	15.4	4.4
1.00	269	4.0	4.0	62.7	47.8	3.2	2.9	0.1	1.5	0.1	15.3	5.0

Table 4

Comparison of the relative deviations from L_f^* of Hoffmann-based rules to those of Rule-GRASP with 1000 iterations.

Set	α	Rel. dev. (%)					Time (s)					Opt. (%)				
		R-G	FH	SW	S	H	R-G	FH	SW	S	H	R-G	FH	SW	S	H
AMP		5.3	5.3	5.2	5.1	5.8	0.5	0.0	0.0	1.3	0.0	55.0	55.0	56.2	55.6	53.1
MP		9.5	9.0	9.1	8.8	9.7	2.0	0.1	0.2	2.5	0.1	35.2	37.8	37.7	38.0	35.3
SBF1	0.25	5.9	5.9	6.0	5.7	6.4	3.0	0.1	0.3	4.0	0.1	33.8	32.7	33.1	34.6	30.9
SBF1	0.50	11.4	11.2	11.2	11.0	11.8	3.5	0.1	0.3	4.5	0.1	18.2	20.1	19.7	20.4	19.0
SBF1	0.75	15.8	15.8	15.7	15.5	16.3	3.5	0.2	0.3	4.6	0.1	13.0	13.4	13.4	13.8	12.6
SBF1	1.00	18.4	18.5	18.5	18.0	19.4	3.5	0.2	0.3	4.5	0.1	11.9	11.5	11.5	12.3	10.4
SBF2	0.25	4.8	4.1	4.2	3.9	4.6	2.9	0.1	0.2	3.2	0.1	37.5	40.9	40.1	43.1	37.5
SBF2	0.50	5.9	4.2	4.4	4.2	4.8	3.1	0.1	0.2	3.4	0.1	34.6	40.9	40.1	42.0	38.3
SBF2	0.75	6.6	4.5	4.6	4.3	5.1	3.3	0.1	0.2	3.6	0.1	32.0	40.9	40.1	42.4	38.3
SBF2	1.00	7.2	4.8	5.0	4.5	5.5	3.3	0.1	0.3	3.7	0.1	30.5	40.1	38.7	40.9	37.9
		9.1	8.3	8.4	8.1	8.9	2.9	0.1	0.2	3.5	0.1	30.2	33.3	33.1	34.3	31.3

smaller and true optimality rates about 1% higher, when $\alpha > 0.25$, and about 2% higher for $\alpha = 0.25$.

Comparing our results with those reported in Esmailbeigi et al. (2016) our approach proves optimality for all but one of the considered instances. Among the instances in their largest test set (SBF2 instances with 32 to 58 tasks) our method proves optimality for 99.4% of the instances. In contrast, the method from Esmailbeigi et al. (2016) (formulation SSBF-1) finds feasible solutions for 52.6% and proves optimality for 37.2% of these instances, despite having significantly larger time limits (1800 seconds). On larger instances, solving model SSBF-1 becomes impractical, since even solving just the linear relaxation of formulation SSBF-1 takes, in average more than 12 minutes.

When comparing our results with those of Zhang and Beck (2025), the best-performing exact method previously reported, we find that all but 20 of the instances (97.5%) considered in their study (specifically, the SBF2 instances with up to 111 tasks, excluding those with Lutz2 precedence graph from the SALBP) are solved to optimality by our method. In contrast, Zhang and Beck (2025) report optimal solutions for 78% of these instances, despite using a longer time limit of 1800 seconds. Moreover, in our independent run of their method using our computing environment, we observed slight increase of the optimality rate (our run finds the optimal solution to 78.8% of the instances). Finally, when focusing on the quality of the upper bounds, we note that out of the 788 instances considered in Zhang and Beck (2025), CBFS achieves a better upper bound in 119 cases (15.1%), while their method produces a better upper bound in only 4 instances (0.5%).

If we consider all of the instances, the results in Table 5 show that CBFS outperforms the approach proposed by Zhang and Beck (2025) across all instance sets. The performance gap becomes more pronounced with increasing setup times (i.e., larger values of α), and is particularly evident in the SBF1 instances compared to SBF2. Consequently, we conclude that our method significantly advances the state of the art in solving the SUALBSP.

To gain further insight into the results of the cyclic best-first search (CBFS), we offer an alternative perspective on the results using the data produced by CBFS. Table 7 groups all instances according to four experimental factors: the number of tasks n , the order strength OS, and the average forward and backward setup times in units of the average task time. The levels are low (–) and high (+) for each factor, and have

been determined according to the minimum, median, and maximum value of the respective factors, as shown in Table 6. Table 7 provides results for each grouping of instances.

The results in Table 7 show that almost all instances where factor n is low (–) are solved to proven optimality. When the number n of tasks in the instance is high, about half of instances with high order strength can still be solved. We can also see that instances with high forward or backward setup times have lower optimality rates and higher relative deviations, with higher backward setup times being more difficult to handle than larger forward setups. This confirms previous observations that such instances are harder to solve.

Finally, we consider the relation of the optimal SALBP-I solutions to the optimal solutions of the SUALBSP. To this end, let v_S be the optimal solution value of a SUALBSP instance and L be the corresponding lower bound from Table 5. Note that the table reports the relative deviation r_{SL} of v_S from L . We now further consider the best known solution values v_I for SALBP-I for the corresponding SALBP-I instances, and consider relative deviations r_{SI} of v_S from v_I and r_{LI} of L from v_I . For set SBF2 the optimal solutions are the same, so relative deviations $r_{SL} = r_{SI}$ in Table 5 also hold relative to the optimal SALBP-I solution. For set SBF1, best lower bounds for $\alpha = 0.25, 0.50, 0.75, 1.00$ deviate from optimal SALBP-I solutions by $r_{LI} = 6.3\%, 10.9\%, 15.0\%, 18.5\%$, respectively, and for set MP by 9.6%. Therefore, looking at r_{SI} , we find that the number of stations in these instances is, due to the setup times, depending on α , about 8 to 25% higher than for the corresponding SALBP-I instances.

6. Conclusions

In this study, we contribute algorithms for line balancing problems with sequence-dependent setup times (SUALBSP). Our dual and primal solution strategies not only advance the field but also highlight the potential of adapting existing methods to more complex scenarios.

We introduce enhanced versions of the one machine scheduling and the Dantzig–Wolfe based lower bound from the simple assembly line balancing problem (SALBP) to the SUALBSP. Our findings demonstrate that the Dantzig–Wolfe based lower bounds notably surpass existing bounds, a result that stands even when computational limitations necessitate the relaxation of both the master and pricing problems. Additionally, the one machine lower bound presents

Table 5
Results for the exact cyclic best-first search and CABS.

Set	α	Rel. dev. (%)		Time (s)		Opt. (%)	
		CABS	CBFS	CABS	CBFS	CABS	CBFS
AMP		0.6	0.4	308.2	8.5	93.1	95.0
MP		2.2	1.1	617.6	34.6	77.5	85.6
SBF1	0.25	3.1	1.4	945.4	45.3	63.2	70.6
SBF1	0.50	5.3	3.0	1,025.3	57.9	57.6	65.1
SBF1	0.75	7.1	4.3	1,071.1	62.1	55.8	63.2
SBF1	1.00	8.1	5.3	1,093.7	63.4	56.5	62.5
SBF2	0.25	1.3	1.0	539.5	31.2	77.0	77.7
SBF2	0.50	1.7	0.9	617.3	35.3	72.5	75.8
SBF2	0.75	1.9	1.0	604.9	34.6	72.9	75.8
SBF2	1.00	2.0	1.0	601.7	37.2	72.1	77.0
		3.3	1.9	742.5	41.0	69.8	74.8

Table 6
Classification of instances according to different experimental factors.

Factor	Factor level	
	–	+
Number of tasks n	[1,75]	[76,297]
Order strength OS	[0,0.58]	[0.58,1]
Relative forward setup time $\bar{\tau}/\bar{t}$	[0,0.13]	[0.13,0.46]
Relative backward setup time $\bar{\mu}/\bar{t}$	[0,0.13]	[0.13,1.00]

Table 7
Optimality rates and relative deviations from the best lower bound as a function of four experimental factors: number of tasks, order strength, relative forward and backward setup times.

n	OS	$\bar{\tau}/\bar{t}$	$\bar{\mu}/\bar{t}$	N	Opt. (%)	Rel. dev. (%)
–	–	–	–	218	98.6	0.3
–	–	–	+	112	96.4	0.4
–	–	+	–	110	100.0	0.0
–	–	+	+	176	94.3	0.5
–	+	–	–	319	98.7	0.1
–	+	–	+	168	96.4	0.2
–	+	+	–	119	100.0	0.0
–	+	+	+	250	81.2	2.2
+	–	–	–	176	62.5	1.9
+	–	–	+	78	20.5	4.6
+	–	+	–	130	39.2	2.7
+	–	+	+	256	18.4	9.4
+	+	–	–	205	77.1	1.0
+	+	–	+	89	53.9	1.9
+	+	+	–	113	54.0	1.6
+	+	+	+	273	33.0	6.1

tighter bounds compared to other fast-to-compute lower bounds, like the combinatorial bounds found in previous literature.

On the primal side, we propose an adapted Hoffmann heuristic, a state of the art partial enumeration method for the SALBP. Furthermore, we introduce the first branch and bound based enumeration approach for SUALBSP. This exact method, built upon our improved primal and dual bounds, outperforms previous solutions in both efficiency and effectiveness, solving 66% of all 2952 test instances from the literature to proven optimality and achieving mean relative deviations from the best dual bounds below 3%.

Our results demonstrate the effective generalization of SALBP solution methods to the more complex GALBPs, like the SUALBSP. Although sequence-dependent setup times introduce additional complexities, the adaption of state-of-the-art SALBP methods are able to effectively manage the enlarged solution space.

Note that further improvements are still possible. For instance, exploring task reordering strategies, a method that has proved to

be successful for SALBP instances is yet to be utilized in the SUALBSP. Additionally, considering tighter relaxations to the Dantzig–Wolfe reformulation may become viable with advancing computational capabilities. The evolving landscape of linear programming software presents opportunities for more sophisticated bounding approaches and alternative integer programming methodologies in the future.

In summary, our study not only marks a significant improvement over previous approaches but also sets a foundation for future research in other line balancing problems. The methods we have developed and adapted hold promise for addressing the challenges posed by real-life assembly line scenarios, highlighting the applicability of SALBP solution methods in more general settings.

CRedit authorship contribution statement

Jordi Pereira: Writing – review & editing, Writing – original draft, Validation, Software, Methodology, Investigation, Formal analysis, Data curation, Conceptualization. **Marcus Ritt:** Writing – review & editing, Writing – original draft, Validation, Software, Methodology, Investigation, Funding acquisition, Formal analysis, Data curation, Conceptualization.

Acknowledgments

M. R. acknowledges support from Coordenação de Aperfeiçoamento de Pessoal de Nível Superior, Brasil (CAPES), Finance Code 001, CNPq (grant 437859/2018-5), and CYTED, Spain (Grant P318RT0165).

Data availability

The data is available online at <http://assembly-line-balancing.de>. The source code is available at <https://github.com/mrpritt/SUALBSP>.

References

- Achterberg, T., 2007. Constraint Integer Programming (Ph.D. thesis). Technische Universität Berlin, <http://opus.kobv.de/tuberlin/volltexte/2007/1611>.
- ALB Research Group, 2011. Homepage for assembly line optimization research. URL <http://www.assembly-line-balancing.de>.
- Álvarez-Miranda, E., Pereira, J., Vilà, M., 2023. Analysis of the simple assembly line balancing problem complexity. *Comput. Oper. Res.* 159, <http://dx.doi.org/10.1016/j.cor.2023.106323>.
- Álvarez-Miranda, E., Pereira, J., Vilà, M., 2024. A branch, bound and remember algorithm for maximizing the production rate in the simple assembly line balancing problem. *Comput. Oper. Res.* 166, 106597. <http://dx.doi.org/10.1016/j.cor.2024.106597>.
- Andrés, C., Miralles, C., Pastor, R., 2008. Balancing and scheduling tasks in assembly lines with sequence-dependent setup times. *European J. Oper. Res.* 187 (3), 1212–1223. <http://dx.doi.org/10.1016/j.ejor.2006.07.044>.
- Bautista, J., Pereira, J., 2009. A dynamic programming based heuristic for the assembly line balancing problem. *European J. Oper. Res.* 194, 787–794. <http://dx.doi.org/10.1016/j.ejor.2008.01.016>.

- Baybars, I., 1986. Survey of exact algorithms for the simple assembly line balancing problem. *Manag. Sci.* 32 (8), 909–932.
- Becker, C., Scholl, A., 2006. A survey on problems and methods in generalized assembly line balancing. *European J. Oper. Res.* 168, 694–715. <http://dx.doi.org/10.1016/j.ejor.2004.07.028>.
- Benders, J.F., 1962. Partitioning procedures for solving mixed variables programming problems. *Numer. Math.* 4, 238–252.
- Boysen, N., Flidner, M., 2008. A versatile algorithm for assembly line balancing. *European J. Oper. Res.* 184 (1), 39–56. <http://dx.doi.org/10.1016/j.ejor.2006.11.006>.
- Boysen, N., Schulze, P., Scholl, A., 2022. Assembly line balancing: What happened in the last fifteen years? *European J. Oper. Res.* 301 (3), 797–814. <http://dx.doi.org/10.1016/j.ejor.2021.11.043>.
- Brucker, P., 2007. *Scheduling Algorithms*, fifth ed. Springer.
- Burkard, R., Dell'Amico, M., Martello, S., 2012. Assignment Problems. Society for Industrial and Applied Mathematics, <http://dx.doi.org/10.1137/1.9781611972238>.
- Delice, Y., 2019. A genetic algorithm approach for balancing two-sided assembly lines with setups. *Assem. Autom.* 39 (5), 827–839. <http://dx.doi.org/10.1108/AA-11-2018-0192>.
- Dolgui, A., Battaia, O., 2013. A taxonomy of line balancing problems and their solution approaches. *Int. J. Prod. Econ.* 142, 259–277. <http://dx.doi.org/10.1016/j.ijpe.2012.10.020>.
- Escudero, L.F., 1988. An inexact algorithm for the sequential ordering problem. *European J. Oper. Res.* 37, 236–249. [http://dx.doi.org/10.1016/0377-2217\(88\)90333-5](http://dx.doi.org/10.1016/0377-2217(88)90333-5).
- Esmailbeigi, R., Naderi, B., Charkghard, P., 2016. New formulations for the setup assembly line balancing and scheduling problem. *OR Spectr.* 38 (2), 493–518. <http://dx.doi.org/10.1007/s00291-016-0433-3>.
- Fleszar, K., Hindi, K.S., 2003. An enumerative heuristic and reduction methods for the assembly line balancing problem. *European J. Oper. Res.* 145 (3), 606–620. [http://dx.doi.org/10.1016/S0377-2217\(02\)00204-7](http://dx.doi.org/10.1016/S0377-2217(02)00204-7).
- Geoffrion, A.M., 1972. Generalized benders decomposition. *J. Optim. Theory Appl.* 10 (4), 237–260. <http://dx.doi.org/10.1007/BF00934810>.
- Hoffmann, T.R., 1963. Assembly line balancing with a precedence matrix. *Manag. Sci.* 9 (4), 551–562. <http://dx.doi.org/10.1287/mnsc.9.4.551>.
- Hooker, J.N., Ottoson, G., 2003. Logic-based benders decomposition. *Math. Prog. Ser. A* 96 (1), 33–60.
- Jackson, J.R., 1956. A computing procedure for a line balancing problem. *Manag. Sci.* 2, 261–271.
- Johnson, R.V., 1988. Optimally balancing large assembly lines with “Fable”. *Manag. Sci.* 34, 240–253.
- Koulamas, C., Kyparisis, G.J., 2008. Single-machine scheduling problem with past-sequence-dependent setup times. *European J. Oper. Res.* 187, 1045–1049. <http://dx.doi.org/10.1016/j.ejor.2006.03.066>.
- Li, Z., Kucukkoc, I., Tang, Q., 2020. A comparative study of exact methods for the simple assembly line balancing problem. *Soft Comput.* 24, 11459–11475. <http://dx.doi.org/10.1007/s00500-019-04609-9>.
- Lübbecke, M.E., 2011. Column generation. In: *Wiley Encyclopedia of Operations Research and Management Science*. John Wiley & Sons, Ltd, <http://dx.doi.org/10.1002/9780470400531.eorms0158>.
- Martino, L., Pastor, R., 2010. Heuristic procedures for solving the general assembly line balancing problem with setups. *Int. J. Prod. Res.* 48 (6), 1787–1804. <http://dx.doi.org/10.1080/00207540802577979>.
- Michels, A.S., Sikora, C.G.S., 2022. A survey on benders decomposition methods applied to assembly line balancing problems. *IFAC-Pap.* 55 (10), 464–469. <http://dx.doi.org/10.1016/j.ifacol.2022.09.437>.
- Morrison, D.R., Sewell, E.C., Jacobson, S.H., 2014. An application of the branch, bound, and remember algorithm to a new simple assembly line balancing dataset. *European J. Oper. Res.* 236 (2), 403–409. <http://dx.doi.org/10.1016/j.ejor.2013.11.033>.
- Pape, T., 2015. Heuristics and lower bounds for the simple assembly line balancing problem type 1: Overview, computational tests and improvements. *European J. Oper. Res.* 240, 32–42.
- Peeters, M., Degraeve, Z., 2006. An linear programming based lower bound for the simple assembly line balancing problem. *European J. Oper. Res.* 168, 716–731.
- Pereira, J., Ritt, M., Vásquez, O.C., 2018. A memetic algorithm for the cost-oriented robotic assembly line balancing problem. *Int. J. Prod. Res.* 99, 249–261. <http://dx.doi.org/10.1016/j.cor.2018.07.001>.
- Rivera Letelier, O., Clautiaux, F., Sadykov, R., 2022. Bin packing problem with time lags. *INFORMS J. Comput.* 34, 2249–2270. <http://dx.doi.org/10.1287/ijoc.2022.1165>.
- Scholl, A., Becker, C., 2006. State-of-the-art exact and heuristic solution procedures for simple assembly line balancing. *European J. Oper. Res.* 168, 666–693.
- Scholl, A., Boysen, N., Flidner, M., 2008. The sequence-dependent assembly line balancing problem. *OR Spectr.* 30 (3), 579–609. <http://dx.doi.org/10.1007/s00291-006-0070-3>.
- Scholl, A., Boysen, N., Flidner, M., 2013. The assembly line balancing and scheduling problem with sequence-dependent setup times: problem extension, model formulation and efficient heuristics. *OR Spectr.* 35 (1), 291–320. <http://dx.doi.org/10.1007/s00291-011-0265-0>.
- Scholl, A., Klein, R., 1997. SALOME: A bidirectional branch-and-bound procedure for assembly line balancing. *INFORMS J. Comput.* 9 (4), 319–334.
- Sewell, E.C., Jacobson, S.H., 2012. A branch, bound, and remember algorithm for the simple assembly line balancing problem. *INFORMS J. Comput.* 24 (3), 433–442. <http://dx.doi.org/10.1287/ijoc.1110.0462>.
- Sternatz, J., 2014. Enhanced multi-hoffmann heuristic for efficiently solving real-world assembly line balancing problems in automotive industry. *European J. Oper. Res.* 235 (3), 740–754. <http://dx.doi.org/10.1016/j.ejor.2013.11.005>.
- Talbot, F.B., Patterson, J.H., Gehrlin, W.V., 1986. A comparative evaluation of heuristic line balancing techniques. *Manag. Sci.* 32 (4), 430–454. <http://dx.doi.org/10.1287/mnsc.32.4.430>.
- Vance, P.H., Barnhart, C., Johnson, E.L., Nemhauser, G.L., 1994. Solving binary cutting stock problems by column generation and branch-and-bound. *Comput. Optim. Appl.* 3 (2), 111–130. <http://dx.doi.org/10.1007/BF01300970>.
- Vilà, M., Pereira, J., 2013. An enumeration procedure for the assembly line balancing problem based on branching by non-decreasing idle time. *European J. Oper. Res.* 229 (1), 106–113. <http://dx.doi.org/10.1016/j.ejor.2013.03.003>.
- Young, K.D., 2017. *Logic-Based Benders Decomposition applied to the Setup Assembly Line Balancing and Scheduling Problem* (M.Sc. thesis). University of Melbourne.
- Zhang, W., 1998. Complete anytime beam search. In: *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 15, pp. 425–430.
- Zhang, J., Beck, J.C., 2025. Domain-independent dynamic programming and constraint programming approaches for assembly line balancing problems with setups. *INFORMS J. Comput.* <http://dx.doi.org/10.1287/ijoc.2024.0603>.
- Zohali, H., Naderi, B., Roshanaei, V., 2022. Solving the type-2 assembly line balancing with setups using logic-based benders decomposition. *INFORMS J. Comput.* 34 (1), 315–332. <http://dx.doi.org/10.1287/ijoc.2020.1015>.