European Journal of Operational Research 250 (2016) 794–806

Contents lists available at ScienceDirect

European Journal of Operational Research

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

Discrete Optimization

Procedures for the bin packing problem with precedence constraints

Jordi Pereira [a,b,*]

[a] Faculty of Engineering and Sciences, Universidad Adolfo Ibáñez, Av. Padre Hurtado 750, Office C216, Viña del Mar, Chile
[b] Escola Universitària d'Enginyeria Tècnica Industrial de Barcelona, Universitat Politècnica de Catalunya, C. Comte Urgell, 187 1st. Floor, 08036 Barcelona, Spain

ARTICLE INFO

Article history:
Received 15 June 2015
Accepted 24 October 2015
Available online 30 October 2015

Keywords:
Bin packing
Precedence constraints
Lower bounds
Dynamic programming
Branch-and-bound

ABSTRACT

The bin packing problem with precedence constraints (BPP-P) is a recently proposed variation of the classical bin packing problem (BPP), which corresponds to a basic model featuring many underlying characteristics of several scheduling and assembly line balancing problems. The formulation builds upon the BPP by incorporating precedence constraints among items, which force successor items to be packed into later bins than their predecessors.

In this paper we propose a dynamic programming based heuristic, and a modified exact enumeration procedure to solve the problem. These methods make use of several new lower bounds and dominance rules tailored for the problem in hand. The results of a computational experiment show the effectiveness of the proposed methods, which are able to close all of the previous open instances from the benchmark instance set within very reduced running times.

1. Introduction

The bin packing problem with precedence constraints (BPP-P) can be described as follows: a set of items, each with a non-negative weight, has to be packed into consecutively numbered bins, each with an identical capacity that limits the total weight of the items packed into the bin. Each item also presents a set of precedence relations: that is, items that need to be packed into a lower numbered bin than the item in question. The objective of the problem is to find the lowest number of bins accepting a packing of items such that all capacity constraints and precedence relations are satisfied.

The BPP-P is a closely related problem to two classical problems: (1) it can be seen as an extension of the bin packing problem in which strict precedence constraints have been incorporated; and (2) it corresponds to the simple assembly line balancing problem (SALBP-1) with a different definition of the precedence relations.

The SALBP-1 is a basic formulation of the assembly line balancing problem (ALBP). ALBPs try to find an optimal assignment of tasks (the items) among the workstations (the bins) so as to maximize the efficiency of the line. In the SALBP-1, tasks have an operation time (their weight); the workstations have a maximum allotted time to perform the tasks equal to the cycle time (the capacity of the bin); the precedence constraints represent technological constraints that require some tasks to be performed after other tasks have been finished; and the efficiency of the line is measured by minimizing the number of workstations required (the number of bins). The main difference between the SALBP-1 and the BPP-P is the rendition of the precedence constraints: that is, while the precedence relations in the SALBP-1 correspond to inequality constraints ($\leq$), the precedence constraint in BPP-P correspond to strict inequality constraints ($<$).

1.1. Literature review

While bin packing and line balancing problems have been widely studied in the literature, see Coffman, Galambos, Martello, and Vigo (1999), Valério De Carvalho (2002) and Clautiaux, Alves, and Valério de Carvalho (2010) for reviews on different aspects of the BPP, Dolgui and Battaïa (2013) for a review on the ALBP, and Scholl and Becker (2006) for a survey specifically geared towards the SALBP-1, the BPP-P has received much less attention, even when the BPP-P is, or is a part of, the basic formulation of several theoretical and practical problems.

A similar problem was introduced in the mid 70's, see Garey, Graham, and Johnson (1976), in order to model single resource constrained multiprocessor scheduling problems. This work studies a generalization of the BPP-P and its relationship with the BPP, and it also provides a heuristic with a 2.7 asymptotic performance guarantee. A similar line of work was followed by Augustine, Banerjee, and Irani (2009) in which the BPP-P is identified as the model of an FPGA dynamic reconfiguration problem, and in which an adaptation of the BPP next fit heuristic is shown to have an absolute performance guarantee of 3.

* Correspondence to: Faculty of Engineering and Sciences, Universidad Adolfo Ibáñez, Av. Padre Hurtado 750, Offce C216, Viña del Mar, Chile. Tel.: +56 32 250 3767; fax: +56 32 250 3500.
E-mail address: jorge.pereira@uai.cl, jorge.pereira@upc.edu

http://dx.doi.org/10.1016/j.ejor.2015.10.048

The literature also covers some recent attempts to solve the problem using exact and metaheuristic methods. In Dell'Amico, Díaz, and Iori (2012), the authors put forward an variable neighbourhood search (VNS) metaheuristic, along with a branch-and-bound method to solve the problem. The branch-and-bound relies on several new bounds which combine classical BPP bounds with the additional considerations derived from the precedence relations. The results show that the proposed metaheuristic and lower bounds are able to close 245 of the 269 instances derived from the classical SALBP benchmark set, see Scholl and Becker (2006). Furthermore, the branch-and-bound reaches optimality for another 21 instances within a two-hours per instance time limit.

Other authors have considered problems which include the BPP-P as a subproblem. Specifically, the ALBP literature covers several cases in which the strict inequality precedence constraints from the BPP-P portray real characteristics of the assembly work. According to Dell'Amico et al. (2012) and Tirpak (2008) reports the design and application of an automated assembly line balancing tool in which some of the tasks feature strict precedence constraints. Another classical ALBP problem in which strict precedence relations appear is the transfer assembly line balancing problem. Transfer assembly lines correspond to a special type of assembly lines in which the tasks are performed by robots with multiple tools. A frequent situation, see for example Borisovsky, Dolgui, and Kovalev (2012); Delorme, Dolgui, and Kovalyov (2012), corresponds to the assignment of a single robot in each workstation that performs all of its assigned tasks in parallel, thus forcing strict inequality precedence constraints.

While BPP and SALBP-1 solution procedures are not directly applicable to the BPP-P, their tight relationship suggests that some of the successful techniques used for these problems could be successfully applied to the BPP-P. Therefore, we proceed to discuss some references from these two problems which are relevant for the present study.

Enumeration based procedures constitute the state-of-the-art for the SALBP-1. The branch, bound and remember procedure, see Morrison, Sewell, and Jacobson (2014); Sewell and Jacobson (2012), is the current best performing exact method for the problem, and the bounded dynamic programming, see Bautista and Pereira (2009), together with a recent modification of a truncated branch-and-bound known as the multi-Hoffmann heuristic, see Sternatz (2014), are the best performing heuristic methods.

All of these methods, to different extent, make use of lower bounds and dominance rules to avoid, or to prioritize, the enumeration of parts of the search tree; to detect equivalent partial solutions, and to certify the optimality of the solutions found. We refer the reader to Pereira (2015) for a comprehensive comparison of different lower bounding procedures for the SALBP-1, and to Scholl and Becker (2006); Sewell and Jacobson (2012); Vilà and Pereira (2013); 2014) for different dominance rules used to detect symmetries and other relations among partial solutions.

The best performing methods to solve the classical BPP correspond to column generation approaches, see Valério De Carvalho (2002), but there is a broad literature on fast lower bounds which can be used on enumeration based procedures, see for example Fekete and Schepers (2001). Other packing problems feature similar characteristics to the BPP-P. Among them, we highlight the bin packing problem with conflicts (BBPC). In this problem the classical BPP formulation is extended to consider that some items cannot share a bin. The relationship between the BPPC and the BPP-P was used in Dell'Amico et al. (2012) to obtain a maximum flow based lower bound for the BPP-P which will be discussed in Section 3.1.

1.2. Contributions of this work

In this paper we address the BPP-P and offer several novel preprocessing rules, dominance rules, lower bounds and upper bounds to those found in the literature. First, we propose a preprocessing rule to reduce the number of items of the instance. Second, we propose two different lower bounds: one based on the one-machine lower bound for the SALBP, and another one based on the Dantzig–Wolfe reformulation of the problem. Third, we put forward a constructive based method built upon a dynamic programming (DP) formulation of the BPP-P. Fourth, we introduce several dominance rules in the procedure proposed in Dell'Amico et al. (2012) as well as a different exploration method based on the successful branch, bound and remember, see Sewell and Jacobson (2012). Among the proposed dominance rules, a generalization of the Jackson dominance rule, Scholl and Becker (2006), which is applicable to the SALBP and other ALBPs, is considered. Fifth, the branch-and-bound makes use of the states explored by the DP-based heuristic, and avoids work repetition between the heuristic and the exact phase of the proposed method.

This work also describes an extensive computational experiment with the proposed procedures. A first experiment using the classical SALBP instances, see Scholl and Becker (2006), allows us to compare the proposed algorithms to the results found in Dell'Amico et al. (2012). The results show that the proposed algorithms provide better lower and upper bounds in much shorter computational times than the previously available methods, allowing us to close all open instances from this set. A second experiment using a new instance set proposed in Otto, Otto, and Scholl (2013) allows us to study the effect of different characteristics of the instances into the quality of the proposed solutions. The results of this experiment show that the method provides very good solutions on a broad set of instances with different characteristics.

1.3. Paper outline

The remainder of the paper is structured as follows. Section 2 is devoted to the description of the problem, its different mathematical formulations of the problem and the notation used throughout the paper. Section 3 studies different lower bounds, while Section 4 describes both the upper bounds and dominance rules derived from the DP formulation of the problem. Section 5 gives a description of the branch-and-bound algorithm and its main differences with previous proposed methods. In Section 6 we document the computational experiment. Finally, Section 7 gives some conclusions.

2. Problem formulation and preprocessing rules

2.1. Problem formulation

A BPP-P instance is defined by n items, each with an associated non-negative weight w_i, $i \in V = [1:n]$. Note that throughout the paper for any pair of integers i, j: $i \leq j$, we denote set $\{i, i+1, \ldots, j\}$ as $[i{:}\, j]$.

Some items also present precedence relations among them. The set of immediate predecessors, and the set of immediate successors of item i are denoted as P_i, F_i; while the set of all predecessor and successor relations of item i (which can be obtained by transitivity) are denoted as P_i^* and F_i^*.

The objective of the problem is to pack all the items into a minimum cardinality subset of consecutively numbered bins S_1, ..., S_m, each with capacity c, while satisfying two sets of constraints:

- The total weight of any bin must not exceed c, that is $\sum_{i \in S_k} w_i \leq c$, $\forall k \in [1, m]$
- for all precedence relations, the bin of the predecessor must have a smaller cardinality value than the bin of the successor. That is, $\forall i \in V, j \in F_i$, if $i \in S_k$ and $j \in S_{k'}$ then $k < k'$ must hold.

Note that reversing the precedence relations (swapping the set of predecessors and successors for all of the items) does not modify the number of bins in the optimal solution. Furthermore, a solution can

be easily converted from the reversed to the original instance (e.g. by renumbering the bins). While the change does not affect the solution, a solution procedure may provide different solutions for the original and the reverse instances, and the appreciated difficulty of each instance may differ.

In some circumstances, it is more convenient to refer to the weight of the items in terms of the fraction of the bin capacity required by the item. We refer to this fraction as p_i, $p_i = w_i/c$, and the capacity constraint is then equivalent to $\sum_{i \in S_k} p_i \leq 1$, $\forall k \in [1, m]$. Similarly, precedence constraints are often represented using an acyclic directed graph $G(V, A)$ in which there is an arc from i to any item $j \in F_i$.

In order to formulate a mathematical model for the problem, it is advisable to have an upper bound on the number of required bins and bounds on the lowest and highest cardinality bin into which each item can be packed. We refer to the optimal number of bins as m^* and to its upper bound as m. We also refer to the earliest and latest bin into which item i ($i \in V$) can be assigned as e_i and l_i respectively. Finally, we denote the set of bins into which an item can be assigned as D_i, $D_i = [e_i : l_i]$.

While we can initially set $m = |V|$, $e_i = 1 (\forall i \in V)$ and $l_i = m$ ($\forall i \in V$); the number of variables can be significantly reduced if better bounds are used, see Sections 2.2 and 4 .

Using a set of binary decision variables x_{ik} whose value is equal to 1 if item i is assigned to bin k, or 0 otherwise, and a set of auxiliary binary variables y_k to represent whether an item has been assigned to bin k, a valid model for the BPP-P corresponds to (1)–(6).

$$[\text{MIN}]\, m^* = \sum_{k \in [1:m]} y_k \tag{1}$$

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

$$\sum_{i \in V} w_i \cdot x_{ik} \leq c \cdot y_k \quad \forall k \in [1 : m] \tag{3}$$

$$\sum_{k \in D_i} k \cdot x_{ik} < \sum_{k \in D_j} k \cdot x_{jk} \quad \forall i \in V;\ j \in F_i \tag{4}$$

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

$$y_k \in \{0, 1\} \quad \forall k \in [1 : m] \tag{6}$$

Objective (1) corresponds to the minimization of the number of bins required to assign all of the items. Constraint set (2) certifies that all of the items are assigned to one bin, while constraint set (3) enforces the bin capacity. Constraint set (4) models precedence constraints, and can be alternatively formulated using (7). Finally, constraint sets (5) and (6) define the domain of the variables.

$$\sum_{k \in D_i} k \cdot x_{ik} + 1 \leq \sum_{k \in D_j} k \cdot x_{jk} \quad \forall i \in V;\ j \in F_i \tag{7}$$

Note that for any given upper bound m, we could substitute m by $m - 1$, and if the above model is infeasible, then the optimal number of bins corresponds to m. Although this model is not computationally useful, it illustrates the differences between the BPP-P and the BPP and SALBP-1. A valid BPP formulation corresponds to the previous formulation without constraint set (4), while the SALBP-1 formulation corresponds to the previous formulation, substituting the $<$ "smaller than" sign from equation (4) by the $\leq$ "smaller than or equal to" sign.

A more useful formulation which is later employed to obtain lower bounds, see Section 3.3 corresponds to a Dantzig–Wolfe reformulation. Column based reformulations have proved to provide efficient solution methods for different bin packing problems, see Casazza and Ceselli (2014); Fernandes Muritiba, Iori, Malaguti, and Toth (2010); Sadykov and Vanderbeck (2013); Vanderbeck (1999), as well as to provide lower bounds for ALBPs, see Bautista and Pereira (2011); Peeters and Degraeve (2006); Pereira (2015). We proceed to describe the complete formulation, and refer to Section 3.3 for the simplified model used to obtain lower bounds.

The formulation makes use of the following additional notation: a feasible assignment of items to bin k, $(1 \leq k \leq m)$, $\vec{q}_{(k)}$ corresponds to a vector of length $|V|$ in which $q_i = 1$ if item i belongs to S_k, and $q_i = 0$ otherwise. The set of different packings which can be assigned to bin k is denoted as $Q^{(k)}$, and corresponds to all of the vectors fulfilling condition (8).

$$Q^{(k)} = \left\{ \begin{array}{c} \vec{q}_{(k)} \in \{0, 1\}^n : \left(\sum_{i \in V} w_i \cdot q_i \leq c \right) \wedge \\ \left(x_i + x_j \leq 1 \ \forall i \in V,\ j \in F_i^* \right) \wedge \\ \left(x_i = 0 \ \forall i : k \notin D_i \right) \end{array} \right\} \tag{8}$$

Using the previous notation, and a set of variables $z_{\vec{q}_{(k)}}$ which take value equal to 1 if assignment $\vec{q}_{(k)}$ belongs to the optimal solution and 0 otherwise, we can reformulate the model using equations (9) to (13).

$$[\text{MIN}]\, m^* = \sum_{k \in [1:m]} \sum_{\vec{q}_{(k)} \in Q^{(k)}} z_{\vec{q}_{(k)}} \tag{9}$$

$$\sum_{k \in D_i} \sum_{\vec{q}_{(k)} \in Q^{(k)}} q_i \cdot z_{\vec{q}_{(k)}} = 1 \quad \forall i \in V \tag{10}$$

$$\sum_{\vec{q}_{(k)} \in Q^{(k)}} z_{\vec{q}_{(k)}} \leq 1 \quad k \in [1 : m] \tag{11}$$

$$\sum_{k \in D_i} \sum_{\vec{q}_{(k)} \in Q^{(k)}} k \cdot q_i \cdot z_{\vec{q}_{(k)}} < \sum_{k \in D_j} \sum_{\vec{q}_{(k)} \in Q^{(k)}} k \cdot q_j \cdot z_{\vec{q}_{(k)}} \quad \forall i \in V, j \in F_i \tag{12}$$

$$z_{\vec{q}_{(k)}} \in \{0, 1\} \tag{13}$$

Constraint set (9) minimizes the number of selected assignments. Constraint set (10) verifies that each of the items belongs to one of the assignments in the solution. Constraint set (11) limits the number of selected assignments for any bin to 1. Constraint set (12) corresponds to the precedence constraints and constraint set (13) defines the domain of the variables.

The number of variables of the previous model is exponential to the number of items of the instance. Hence, the enumeration of all of the assignments is not advisable, and variables are added when needed by solving the so-called pricing problem, see Section 3.3.

An alternative formulation is based on a dynamic programming (DP) approach. The model builds upon the DP SALBP-1 model, see Bautista and Pereira (2009). The formulation considers the BPP-P as a multistage decision process in which each state, S, is identified by the set of items already assigned to bins; each stage is associated to a number of already assigned bins; and a transition from a state belonging to stage u to a state belonging to stage $u + 1$ corresponds to a full assignment of items to bin $u + 1$. Using this formulation, a solution corresponds to the path from an initial state $S = \{\emptyset\}$ to a final state $S = V$.

More formally, a state S belonging to stage u corresponds to an assignment of items to the set of bins $[1: u]$, $S = S_1 \cup S_2 \cup \ldots \cup S_u$. Note that the actual composition of bins is irrelevant. A valid transition between a state S from stage u, and state S' from stage $u + 1$ must satisfy:

1. $\sum_{i \in (S' - S)} w_i \leq c$
2. $\forall i \in V, \forall j \in F_i^*,\ i \notin S \rightarrow j \notin S'$

The first condition corresponds to the bin capacity constraint, while the second condition ensures that the precedence items are assigned before its successors and not the same bin (by forcing the preceding item to be assigned to an earlier bin than the succeeding item). The shortest path (smallest number of transitions) from state $S = \{\emptyset\}$ to the final state $S = V$ corresponds to the optimal solution, and the number of transitions corresponds to the optimal objective.

Note that the previous formulation suffers from the same problem as the Dantzig–Wolfe reformulation described above, since the number of states and transitions is exponential to the number of items. Nevertheless, similar formulations have been successfully used to develop an efficient heuristics, see Section 4, as well as to obtain dominance relations among different partial solutions.

2.2. Preprocessing rules

In Dell'Amico et al. (2012), the authors propose a preprocessing rule to increase the weight of the items. The basic idea is to consider all of the bin assignments that include a particular item, and to detect whether there is some capacity surplus in each of these bin assignments. If such a surplus is identified, the weight of the item is lifted in order to take into account the unavoidable surplus in subsequent bounding and preprocessing calculations.

For any given item, the problem can be formulated as a knapsack problem with conflicts, see Fernandes Muritiba et al. (2010); Hifi and Michrafy (2007).

For any item j, let V^j be the set of items which can share a bin with item j; that is, any item i in which $D_i \cap D_j \neq \emptyset$ and $w_i + w_j \leq c$ hold. Using a set of variables $x_i (i \in V^j)$, Eqs. (14)–(18) constitute a valid formulation for the problem.

$$[\text{MAX}] \sum_{i \in V^j} w_i \cdot x_i \tag{14}$$

$$\sum_{i \in V^j} w_i \cdot x_i \leq c - w_j \tag{15}$$

$$x_i + x_{i'} \leq 1 \quad i \in V^j, i' \in F_i^* \tag{16}$$

$$x_i + x_{i'} \leq 1 \quad \forall i \in V^j, \forall i' \in V^j : D_i \cap D_{i'} = \emptyset \tag{17}$$

$$x_i \in \{0, 1\} \quad \forall i \in V^j \tag{18}$$

Objective (14) maximizes the use of the capacity of the bin. As constraint (15) asserts, its maximum capacity is $c - w_j$. Consequently, if $w_i \cdot x_i < c - w_j$, w_j can be set to $c - \sum_{i \in V^j} w_i \cdot x_i$. Constraint sets (16) and (17) make sure that two items which cannot share a bin are not assigned to the same bin. Finally, (18) defines the domain of the variables.

During the preprocessing step, any item whose item weight equals the capacity of the bin (that is $w_i = c$) can be safely removed from V, as the item needs a station to itself. After removal, the immediate predecessors and successors of the item need to be modified according to these rules: (a) for any predecessor of the removed item, its set of successors should include the set of successors of the removed item; and (b) for any successor of the removed item, its set of predecessors should include the set of predecessors of the removed item.

In addition to the use of the previous rule, we also propose a novel preprocessing rule aimed at reducing the number of items in the instance by considering optimal packings for the first, or last, station of the instance.

Let L_1 be the set of items which can be assigned to the first bin (those with no precedence relations). If $\sum_{i \in L_1} w_i \leq c$, L_1 corresponds to an optimal assignment of items to the first bin, as no other item can be assigned to the said bin. Consequently, we can remove all of the items and close the first bin, effectively reducing the set of items to assign.

This rule can be consecutively applied in order to further reduce the size of the instance (if the first bin is closed, apply the rule to the second bin), until the condition does not hold. Due to the reversibility property, the same preprocessing rule can be applied to the reverse instance, assigning items to the last bin of the solution. Since the ability of the weight lifting preprocessing rule depends on the number of items (it is more likely to increase the weight of an item if the set of items is smaller), this rule should be applied before preprocessing the weights of the items.

The proposed rule is a special case of a more general rule which is used by the dynamic programming and the branch-and-bound procedure in order to reduce the number of partial solutions that need to be considered, see Sections 4 and 5. Let L_1, L_2,..., L_k be the set of items with earliest station equal to $1, 2 \ldots, k$, respectively. If there exists a partial solution defined by the sets S_1, S_2,..., S_k such that $L_1 \cup L_2 \cup \ldots \cup L_k = S_1 \cup S_2 \cup \ldots \cup S_k$, the aforementioned partial solution is an optimal partial assignment of these items, and any other assignment with k stations is dominated by the said assignment.

3. Lower bounds

In this section we introduce some of the lower bounds available in the literature for the BPP-P as well as two novel approaches which can be seen as adaptations of existing lower bounds for the SALBP-1. Note that other bounding methods are possible, as the BPP and the SALBP-P are valid lower bounds for the BPP-P, and thus any bounding method for these two problems could be used to obtain bounds.

3.1. Previous lower bounds

An initial bound derived from the BPP via the relaxation of the precedence and integrality constraints corresponds to $\lceil \sum_{i \in V} w_i / c \rceil$, or equivalently $\lceil \Sigma_{i \in V} p_i \rceil$.

Another bound based on the relationship with the BPP uses the previous bound with alternative sets of weights. Alternative sets of weights can be obtained using dual feasible functions (DFF), see Fekete and Schepers (2001) and Haouari and Gharbi (2005). A dual feasible function defines a mapping of the original weights p_i $(i \in V)$ to an alternative set of weights u_i $(i \in V)$ such that if a subset of items T can share a bin, the same applies to the alternative set of weights. Once a mapping is available, the new weights can replace the original set of weights to compute lower bounds on the optimal solution.

Multiple DFFs are available in the literature. In this work we use the rules proposed in Fekete and Schepers (2001) as well as their composition.

DFF 1: for any given $k \in \mathbb{N}$, $u^k(p) = p$ if $(k+1)p \in \mathbb{Z}$, and $u^k(p) = \lfloor (k+1)p \rfloor (1/k)$ otherwise. The values of $k = 1, 2, \ldots, 100$ are used, see Fekete and Schepers (2001).

DFF 2: for any given $\epsilon \in [0, 1/2]$, $U^\epsilon(p) = 1$ if $p > 1 - \epsilon$, $U^\epsilon(p) = p$ if $\epsilon \leq p \leq 1 - \epsilon$ and $U^\epsilon(p) = 0$ otherwise. The values of ϵ proposed in Haouari and Gharbi (2005) are considered ($\epsilon \in \{p_j \in [0, 0.5]\} \cup \{0.5\}$).

The composition of both DFFs provide the final set of dual feasible values: $v^{(k,\epsilon)} = U^\epsilon(u^k(w))$. In order to consider *DFF* 1 and *DFF* 2 as special cases of the compound DFF, we also consider $k = \infty$ and $\epsilon = 0$ during the calculation.

An alternative way to derive bounds for the BPP-P is to consider the precedence relations. Dell'Amico et al. (2012) proposes a bound based on computing the longest path of precedences among items. The calculation uses an auxiliary acyclic digraph $G'(V', A')$ in which the set of vertices V' contains V as well as two additional vertices, α, ω. These vertices are referred as the source and the destination of G'. A' contains all of the arcs in A (each with length equal to 1), as well as an arc from the source to any of the vertices in V, and an arc from any of the vertices in V to the destination (each with length equal to 0). The longest path between vertices α, ω, $l_{\alpha,\omega}$, corresponds to a lower bound on the number of bins required (as all of the vertices in this path cannot share a bin).

Additionally, the length of the longest path from the source to any other vertex i, $l_{\alpha,i}$ provides a lower bound on e_i, and the length of the longest path from vertex i to the destination vertex $l_{i,\omega}$ can be used to define l_i, that is $l_i = m - l_{i,\omega}$.

Dell'Amico et al. (2012) also considers multiple methods to derive additional edges with their respective length (a lower bound on the difference of cardinality between their respective assigned bins).

Finally, Dell'Amico et al. (2012) also proposes a combined bound which uses both the weight and the precedence constraints. The problem considers a max-flow relaxation of the problem, which was originally proposed in Gendreau, Laporte, and Semet (2004) for the bin packing problem with conflicts.

In order to compute the bound, a new auxiliary graph $G''(V'', A'')$ is required. The set of vertices V'' is divided into four sets, one subset composed exclusively by the source vertex, a second subset U ($U \subset V$), a third subset U' ($U' \subset V$), and a final subset composed by the destination vertex. The set U' is composed by all of the vertices in the longest path in $G'(V', A')$, while the remaining vertices in V belong to U ($V = U \cap U'$). A'' is composed by three sets of arcs: (1) there is an arc between the source vertex and all of the vertices in U with capacity equal to the weight of the vertex in U; (2) there is an arc with unlimited capacity between any pair of items in U and U' if the assignment of both items to a single bin is feasible (that is, their respective item weights do not surpass c, as they can be assigned to an identically numbered bin); and (3) there is an arc from each vertex i in U' to the destination vertex with capacity $c - w_i$. Let F be the max-flow from α to ω in $G''(V'', A'')$. Then, $|U'| + \lceil \left(\sum_{i \in U} w_i - F \right) / c \rceil$ is a valid lower bound on the number of bins required.

The rationale of the bound is the following: $|U'|$ bins are required to assign the items from the longest path; and F is an upper bound of the total weight of the items that can be assigned together with the items in U'. Therefore, the bound counts the number of items required by the longest path and a bound on the weight of the items which cannot be assigned with the items of the longest path.

3.2. Machine scheduling-based lower bounds

A traditional lower bound for the SALBP-1, Pereira (2015); Scholl and Becker (2006), assimilates the problem to a single machine problem with deliveries. The same assimilation can be used to derive a bound for the BPP-P after some modifications on the calculations.

The bound considers that each job (item) is to be sequentially processed in a machine and, after being processed, is to have a subsequent delivery time which can be simultaneously performed. The objective is to minimize the maximum completion time (waiting time plus machine time and delivery time) among all of the jobs, and the optimal schedule corresponds to processing the jobs in non-increasing order of their delivery times.

We denote the processing time of job i as p_i and its delivery time as η_i. The processing time of a job corresponds to the fraction of bin capacity used by the job, which was also previously referred as p_i, while its delivery time corresponds to a lower bound on the number of stations required by its successors. Note that the machine ensures that jobs are assigned to different time slots (representing the fractions of the capacity of the bins) and the completion time provides a lower bound on the bin capacity required by the item itself and its followers.

Let $< h_1, h_2, \ldots, h_n]$ be an optimal ordering of jobs, then (19) is a lower bound on the total number of bins. Furthermore, η can be rounded up as it represents the number of bins.

$$\eta = \max\{p_{h_1} + \eta_{h_1}; p_{h_1} + p_{h_2} + \eta_{h_2}; \ldots; p_{h_1} + p_{h_2} + \ldots + p_{h_n} + \eta_{h_n}\} \quad (19)$$

The previous lower bound depends on the quality of the delivery times used in the calculation. For example, in the extreme case in which no delivery times are used, the bound reduces to the initial lower bound from the BPP, but the addition of delivery times may help to detect unavoidable surplus capacities.

In order to obtain estimates of the delivery times, it is customary to solve the single machine bound on the set of successors of each of the items. The computation is arranged in such a way that the delivery time of any item is calculated before being required to calculate the delivery time of any other successor (for example by ordering the items in non-decreasing order to their number of absolute successors $|F_i^*|$).

The previous description agrees with the application of the bound for the SALBP-1, with a different rounding rule. In the SALBP-1 a task (item) can share a station (bin) with its successors, and the rounding can only be performed if it is possible to verify that no sharing is to occur. In the BPP-P, the precedence relations force items not to share a bin with its successors, hence $\eta_i = \lceil \eta \rceil$ always holds.

Using the reversibility of the problem, it is possible to calculate another set of tails which would correspond to the release dates of the jobs. The release date of a job i is denoted as a_i, and it is calculated using the set of predecessors rather than the set of successors. The lower bounds provided by using the set of predecessors and successors vary, thus it is recommended that both bounds are calculated.

In addition to the two bounds described, the release dates and delivery times of each job also provide a tighter lower bound that the longest path for the calculation of the earliest e_i and latest l_i, bin into which an item i can be assigned. For a station with m stations to exist, Eqs. (20) and (21) apply:

$$e_i = 1 + a_i \quad (20)$$

$$l_i = m - \eta_i \quad (21)$$

The previous calculation can be strengthened considering the previously obtained DFF weights, as in Pereira (2015) for the SALBP-1. If the set of processing times p_i is substituted by any dual feasible weight $\nu^{(k,\epsilon)}$, the resulting lower bound is still a valid lower bound on the number of bins required by the successors of an item. Let $\eta_i^{(k,\epsilon)}$ represent the lower bound obtained by using the set of weights $\nu^{(k,\epsilon)}$, and let η_i^* be the maximum of those weights, see (22), then we can lift $\eta_i^{(k,\epsilon)}, \forall k, \forall \epsilon$ to η_i^* as this lower bound on the number of bins is applicable regardless of the set of weights that detected it.

$$\eta_i^* = \max_{\forall k \forall \epsilon} \eta_i^{(k,\epsilon)} \quad (22)$$

For each given set of weights, the complexity of this lower bound is equal to $O(n \log n)$ as the items need to be rearranged in non-increasing order of delivery times. If the jobs and their delivery times are already available, it is possible to calculate (19) in $O(n)$ time. Consequently, and in order to use this bound in the enumeration procedures proposed in Sections 4 and 5, during a preprocessing step, see Section 5.2, the delivery dates are calculated using all of the $\nu^{(k,\epsilon)}$ weights, but during the enumeration phase only η_i is used in conjunction with the original weights of the items.

3.3. Dantzig–Wolfe decomposition lower bound

A final, more computationally intensive, bound can be obtained by the resolution of a relaxed version of the Dantzig–Wolfe reformulation, see Section 2.1. Dantzig–Wolfe reformulations have shown to be very effective for the SALBP-1, see Pereira (2015), outperforming all of the available lower bounds, including the optimal solution of the BPP relaxation of the SALBP-1. Since the precedence constraints play a stronger role on the BPP-P, it is to be expected that good lower bounds are to be obtained.

According to previous work, see Peeters and Degraeve (2006), the precedence constraints of the master problem, formulation (9)–(13), do not affect the value of the lower bound but complicate the pricing problem. Hence, the precedence constraints are removed from the master problem leading to a set packing formulation, see (23)–(25).

$$[\text{MIN}] \sum_{\vec{q} \in Q} z_{\vec{q}} \quad (23)$$

$$\sum_{\vec{q} \in Q} q_i \cdot z_{\vec{q}} = 1 \quad \forall i \in V \quad (24)$$

$$0 \leq z_{\bar{q}} \leq 1 \tag{25}$$

Note that in this formulation, removing the index to identify the bin of the assignment leads to a set of bin assignments $Q = Q^{(1)} \cup Q^{(2)} \cup \ldots \cup Q^{(m)}$, and the set of variables $z_{\bar{q}}$ does not identify the bin of an item assignment.

Although precedence relations and bin identification are removed from the master problem, they are still enforced in the pricing problem. In order to formulate the pricing problem, let $< \pi_1, \pi_2, \ldots, \pi_n >$ be the dual price of each of the constraints (24), and define a set of variables x_i, $i \in V$ to denote whether item i is to be included in the bin assignment. Then, the pricing problem corresponds to (26)–(30).

$$[\text{MIN}]rc = 1 - \sum_{i \in V} \pi_i \cdot x_i \Rightarrow [\text{MAX}] \sum_{i \in V} \pi_i \cdot x_i \tag{26}$$

$$\sum_{i \in V} w_i \cdot x_i \leq c \tag{27}$$

$$x_i + x_j \leq 1 \quad \forall i \in V,\ j \in F_i^* \tag{28}$$

$$x_i + x_j \leq 1 \quad \forall i \in V,\ j \in V,\ D_i \cap Dj = \{\emptyset\} \tag{29}$$

$$x_i \in \{0, 1\} \quad \forall i \in V \tag{30}$$

Objective (26) minimizes the reduced cost. Constraint (27) forces the capacity constraint, while constraint set (28) imposes precedence constraints. Constraint set (29) makes sure that all of the items can be assigned to the same bin and (30) defines the domain of the variables.

Formulation (26)–(30) corresponds to a knapsack problem with conflicts, which is solved using a branch-and-bound algorithm similar to the one proposed in Hifi and Michrafy (2007).

In order to minimize the required time for the pricing phase and to tighten the lower bound, the following additional elements are considered:

- The pricing problem is not always solved to optimality. Instead, an initial solution with reduced cost equal to 0 is imposed, and if the best solution is improved 10 times, the search is stopped and the incumbent solution is introduced in the master problem. The objective of this limit is to avoid the exploration of the whole search space while still introducing high quality columns in each iteration.
- The bound is calculated after an initial solution, see Section 4, is available. Then, the earliest and latest bins for each item are computed using the machine scheduling bound with a number of bins m one unit smaller than the original solution. This operation reduces the number of alternatives to consider in the pricing. If the bound provided by the reformulation is larger than $m - 1$, the current solution is optimal.
- The resolution process is stopped whenever the reformulation certifies that it may be possible to attain a solution with $m - 1$ stations.

4. Dynamic programming based upper bound

4.1. Description of the algorithm

The main drawback of dynamic programming (DP) approaches lies on the large state space needed to enumerate, a phenomenon coined by Bellman as the "curse of dimensionality". Several techniques exist to partially palliate the problem, like state space relaxation, Christofides, Mingozzi, and Toth (1981), dynasearch, Congram, Potts, and van de Velde (2002), restricted dynamic programming Gromicho, van Hoorn, Kok, and Schutten (2012), or bounded dynamic programming, Bautista and Pereira (2009). These methods only consider a subset of the state space, focusing their attention on the reduced state space.

In this case, we follow the bounded dynamic programming approach which has proved to be an effective method for several line balancing problems, see Bautista and Pereira (2009); 2011). In the bounded dynamic approach, the optimization problem uses the polyetapical graph interpretation of the DP formulation depicted in Section 2.2. The decisions are structured in the graph in which the vertices correspond to the states and the arcs to the transitions. The graph is constructed one stage at a time using the two-list approach to store current and future state, Bautista and Pereira (2009); 2011), starting from an initial stage, stage 0, with a single state $S = \{\emptyset\}$, until reaching a state in which all of the items have been assigned, $S = V$. In order to limit the number of states and transitions, the following rules are applied:

- For any given state, the transitions are constructed generating all of the bin assignments according to an enumeration scheme which only generates feasible bin assignments, see Bautista and Pereira (2009); Sternatz (2014).
- The single machine lower bound is used to avoid the development of inefficient states. Whenever a state is generated, the lower bound provides an indication on the number of additional stations required, and if it cannot improve the incumbent solution, the state is discarded. Note that the term "bounded" from bounded dynamic programming comes from the use of these lower bounds.
- If a state is not discarded by the application of the lower bound, several dominance rules, see Section 4.2, are applied. If the state is considered as dominated, it is again discarded.
- For each state, only a subset of the transitions are generated . During the enumeration of transitions, if the algorithm reaches the limit, the remaining transitions are discarded. The behaviour is controlled by a parameter of the algorithm denoted as γ, see details in Section 5.2, which controls the running time requirement of the enumeration.
- Among the transitions generated for each state, a reduction procedure is applied to select a subset of the most promising transitions, and the remaining transitions are discarded. The subset of promising transitions corresponds to those that generate states with better one machine lower bound before rounding (that is we use η rather than $\lceil \eta \rceil$). Ties are broken in favour of those states/transitions in which the longest path between any unassigned item and ω, $[\text{MIN}]_{i \notin S} l_{i,\omega}$, is minimum. If the tie persists, ties are broken favouring the first transition generated during the enumeration. This behaviour is controlled by a second parameter of the algorithm denoted as β, see Section 5.2.
- Among the subsets of most promising transitions, a subset of the most promising states is selected. The same rule is used to select the most promising states (that is, the one machine lower bound value; with ties broken by the longest path; and remaining ties broken by order of generation). This behaviour is controlled by a third parameter of the algorithm denoted as α, see Section 5.2.

The method has three control parameters: α, β and γ. Among all generated states from a given stage, at most α states are used to generate the states of the following stage. For each state, at most γ descending states are generated and only the best β of them are considered for inclusion within the reduced subset of α states.

All these parameters control the behaviour of the algorithm and try to avoid the curse of dimensionality. If these three parameters are set to ∞, the algorithm behaves as a traditional dynamic program, and the inclusion of dominance rules and bounding procedures remove irrelevant states. If $\alpha = \beta = \gamma = 1$, the algorithm behaves as a greedy heuristic, and if $\alpha = \beta = 1$ and $\gamma = \infty$, the algorithm generates all possible station assignments from any given partial solution and selects the best among them, thus resembling the Hoffmann heuristic, see Bautista and Pereira (2009); Sternatz (2014), which has proved to offer very good solutions for the SALBP-1.

4.2. Dominance rules

Enumeration based approaches like the bounded dynamic programming method described above, or the branch-and-bound method described in Section 5, benefit from the use of rules to avoid repeated exploration of equivalent or dominated partial solutions. The DP roots of the bounded dynamic programming method provide an effective method to detect equivalent partial solutions which share a single state representation, but the use of additional rules, which are usually used in branch-and-bound methods, further reduces the effort devoted to explore inefficient states and also increases the chances that the path of transitions associated to an optimal solution are not be discarded by the heuristic pruning rules.

Some of the rules make use of a memory structure that stores the states that have been previously constructed. The structure is internally stored as a hash table Cormen, Leiserson, Rivest, and Stein (2009) in order to obtain fast lookup and insertion operations, using the state (represented as an array of binary values of length $|V|$) as the key of each entry of the table, and two values per key: a flag to identify if the entry has been explored (that is, if the algorithm has used the state to generate transitions); and an integer, s, equal to the stage in which the state has been explored (that is, the number of bins required to pack all of the assigned items).

The implementation of the hash table corresponds to the *unordered map* data structure provided by the standard library of the programming language, see Josuttis (2012). A hash function is used to map the state into a slot in the hash table. As multiple states may be mapped to a single slot (an effect known as collision), each slot has an associated linked list to store the key and its values, enabling the comparison of new states with all of the previously generated ones. Consequently, the use of the hash table reduces the number of comparisons required to assure that a state has not been previously considered from all of the stored states to only those states that share the same hash value.

Note that a state is likely to be reached in two stages. In such a case the state associated to the later stage is dominated by the state with lower stage number (as it requires fewer bins). Furthermore, if a newly generated state is already in the table, the same state was reached by a different sequence of transitions. We proceed to describe each of the dominance rules.

Maximum load rule: the rule is applied during the enumeration of bin assignments, and tries to avoid the exploration of states generated by suboptimal transitions.

Any bin assignment can be classified into partial and full bin assignments. A partial bin assignment corresponds to an assignment of items to a bin in which one or more additional items could be included while still satisfying the set of constraints. If the bin assignment cannot include additional items, then it is a full assignment. Based on this statement, it is obvious that for any given state, the state resulting from a transition involving a partial bin assignment is dominated by the state resulting from a full assignment. This rule is tested during the enumeration of transitions, and it is enforced by refusing to consider any partial assignment. Note that rejected transitions are not considered in the calculation of the maximum number of generated transitions, controlled by parameter γ.

Generalized maximum load rule: This rule generalizes the maximum load rule to the case of partial solutions. Let us consider two states S, S' belonging to the same stage s and an item i, such that $S = S' \cup \{i\}$, that is, the set of items assigned to S is a superset of items assigned to S'. Then we can state that S' is dominated by S, as the best possible solution for any continuation of S' will always render the same or a worse value than the best possible continuation for S.

The rule is applied as follows: for any item which could be assigned with S' satisfying an aggregate capacity constraint (that is, $\sum_{S' \cup \{i\}} p_i \leq s$) we test whether the previous state has been generated and stored in the hash table structure memory. If such a state exists, and its stage value s is equal to or smaller than the stage of S', then S' is dominated.

Item dominance rule: This rule is an adaptation of the Jackson rule for the SALBP-1, see a description in Scholl and Becker (2006). Let i and j be two items. Item i is said to dominate item j if: (1) $w_i \geq w_j$; and (2) $F_j^* \subseteq F_i^*$ both hold. If both conditions are satisfied in equality, then we arbitrarily state the lowest-numbered item as dominating the highest-numbered item. Note that these conditions certify that if we can replace item j for item i in a partial solution, then it is advisable to do so (the remaining problem is easier to solve as item j requires less capacity and no precedence relation is modified).

The rule is tested in two different ways. The first method corresponds to the classical use of the Jackson rule, and tests the rule during the enumeration of transitions. After constructing an assignment containing item j, if we can swap item j for item i without breaking capacity or precedence constraints, the bin assignment using item j is dominated by the bin assignment using i.

The second method uses the same methodology exposed in the *generalized maximum load rule*. If we can swap item j for item i while fulfilling an aggregate capacity constraint, and we find a state previously stored in the hash table from the same stage, the new state is dominated by the state stored in the hash table. This method, which has not been previously considered in the ALBP literature, and can be easily generalized to several ALBPs.

Generalized item dominance rule: the following rule is also a novel contribution of the present paper, and it generalizes the *item dominance rule*.

Let i and j be two items. Item i is said to dominate item j if: (1) $w_i \geq w_j$; (2) the subgraph of $G(V, A)$ induced by item i and its successors (denoted as G^i) contains a subgraph isomorphism of the subgraph of $G(V, A)$ induced by item j and its successors (denoted as G^j); and (3) the weights of the items associated with each vertex in G^i coincide with the items of their mapped vertices in G^i (with the exception of vertex j which fulfils condition 1).

This dominance rule follows the same rationale than the *item dominance rule*, that is, if item j can be substituted by item i in a partial solution, it is advisable to do so (the remaining problem is easier without loss of optimality).

Note that the problem of finding a subgraph isomorphism is NP-complete. Nevertheless, the size of the subgraphs considered for the BPP-P (even with the additional requirement on the vertex weights) can be easily solved, and thus the complexity of the algorithm does not represent a practical issue. In order to ascertain if such a subgraph exists, we use a derivation of the recursive algorithm proposed in Ullmann (1976), modified to take into account the weights associated to the vertices.

5. Branch-and-bound algorithm

When the previous methods (upper and lower bounds) fail to verify the optimality of the incumbent solution, we rely on an enumeration based approach to verify optimality. The proposed method corresponds to a branch-and-bound approach known as branch, bound and remember, see Morrison et al. (2014); Sewell and Jacobson (2012). The main characteristic of the branch, bound and remember approach is the use of a memory structure that stores already explored partial solutions. Although this feature is memory consuming, it allows to avoid the re-exploration of equivalent partial solutions and enables the use of a modified exploration scheme that tries to concurrently search different areas of the branch-and-bound tree.

The remainder of the section is structured as follows: Section 5.1 is devoted to the branch, bound and remember algorithm, including a description of the memory structure and the exploration scheme. Section 5.2 summarizes the complete approach, including the integration of the state space explored by the bounded dynamic program into the branch, bound and remember framework.

5.1. Branch, bound and remember

The proposed enumeration method follows a bin-based branching rule, which mimics the station oriented branching scheme for ALBPs, see Morrison et al. (2014); Vilà and Pereira (2014). In this scheme, a branching decision consists in appending a new bin to the partial solution and defining its composition. The appended bin follows all previously defined bins (its cardinality is equal to the number of previous bins plus one) and its composition fulfils the capacity constraint as well as precedence constraints. Therefore, whenever a node is selected for branching, the algorithm generates all feasible packings that become unexplored nodes (partial solutions) of the branch-and-bound tree. Note that the proposed branching rule deviates from the branching rule proposed in Dell'Amico et al. (2012) as the procedure only branches in one direction (which is selected in the preprocessing step according to an indication of the number of alternative packings for the initial bins, see Section 5.2). Moreover, the one-machine lower bound is used to prune partial solutions which cannot lead to improved solutions.

The main differences of the proposed scheme when compared to other branch-and-bound proposals are: (1) whenever a partial solution is constructed, its equivalent state in the DP is stored in memory and pruned if it is equivalent to a previously explored partial solution; (2) as the algorithm stores all of the explored states, the dominance rules described in Section 4.2 are used to avoid repeated calculations; and (3) the algorithm uses an alternative method to decide which node to explore based on the selection of nodes associated to different levels of the search tree.

Nodes are explored using a strategy known as cyclic best-first search (CBFS), Morrison et al. (2014). In this strategy, multiple priority queues are used to decide which partial solution is to be explored next. Each priority queue is associated to a different level of the branch-and-bound tree, and keeps track of all the unexplored partial solutions associated to the said level. The order of the partial solutions in each priority queue is determined by the same rule used to prioritize states in the bounded dynamic programming algorithm (value of the one machine lower bound, with ties broken according to the length of the longest path of successors). Partial solutions are selected from each priority queue following a cyclic rule, starting from the first queue until the last queue containing an unexplored node. The search ends when none of the queues contains a node, that is, all of the vertices have been explored.

We would like to point out that the method represents nodes and branching decisions using the states and transitions of the corresponding dynamic programming formulation. Whenever a new final state with fewer stations is found, the equivalent solution is reconstructed following the transitions used to build the corresponding states stored in memory. Hence, the method can be seen as an alternative technique to the exploration of the DP state space.

Also note that storing the states is a memory demanding operation, which may lead to the premature end of the search due to the memory limits of the computer. This behaviour has not been encountered in our computational experiments, see Section 6.

5.2. General structure of the algorithm

The final method is a combination of the lower bounds, the dynamic programming heuristic and the exact enumeration procedure. Specifically, the heuristic and the exact enumeration method are linked in such a way that they try to avoid repeating calculations. The link between these methods comes from the use of the same memory structure to mark which states have already been explored. The final structure of the method follows:

1. The single machine lower bounds for the forward and backward direction of the precedence graph are solved. Let LB be the best of these bounds.
2. Solve the bounded dynamic programming heuristic in forward and backward direction to find an initial incumbent solution with UB bins. If $LB = UB$ end (the optimal solution has been found).
   The following set of parameter are used in this step: $\alpha = 1,000$, $\beta = 50$ and $\gamma = 1,000$. These parameters were used because they represent a trade-off between quality and solution time, see Bautista and Pereira (2009); 2011) for an analysis of the previous parameters on the overall performance of the bounded dynamic programming algorithm.
   After each search ends, empty the hash table. Consequently, the states considered in this step may be explored again in subsequent steps.
3. Use UB, the release dates and the delivery times provided by the machine lower bound to obtain an earliest and latest bin for each item. Note that the value used for m in Eqs. (20) and (21) is $UB - 1$. If for any given item i, the inequality $e_i > l_i$ holds, the current solution is optimal and the search is stopped.
4. Apply the preprocessing rules, Section 2.2.
5. Solve again the single machine lower bound for the reduced instance. If $LB = UB$, the current solution is optimal and the search is stopped. Otherwise, select the direction of the precedence graph using the rule proposed in Sewell and Jacobson (2012). The rule calculates the number of items which can be assigned to each station taking into account its earliest and latest station assignments. Let $f_i = |\{j : e_j < i\}|$ be the cardinality of the set of items that can be assigned to bin i, and let $r_i = |\{j : l_j > UB - i\}|$ be the cardinality of the set of items which can be assigned to bin $UB - i$. Then, select the forward direction if $\prod_{i=1}^{5} f_i \leq \prod_{i=1}^{5} r_{UB-i}$ and the reverse direction otherwise.
6. Obtain the Dantzig–Wolfe decomposition lower bound. If the lower bound obtained validates the optimality of the incumbent solution, the procedure is stopped and the optimal solution is reported.
7. Use the DP-based heuristic to find a second initial solution. The second execution uses the following set of parameters: $\alpha = 1,000$, $\beta = 50$ and $\gamma = \infty$. In this case, γ is set to ∞ in order to allow the algorithm to store all non-dominated, non-bounded descending states in the memory structure. The change leads to higher running times during the enumeration since all of the descending states from each explored partial solution are generated. As parameters α and β only control the subset of states that the DP-based heuristic explores and parameter γ is effectively eliminated by setting its value to ∞, the method does not heuristically eliminate any partial solution, but rather stores them in the memory structure for future consideration.
8. Insert in their respective priority queues all unexplored states in the memory structure.
9. Apply the branch, bound and remember enumeration procedure until reaching a maximum number of states in memory, reaching a time limit or no unexplored state exist. In the last case, return the optimal solution. Otherwise, return an upper bound and a lower bound on the optimal solution. A running time limit of 7,200 seconds, as in Dell'Amico et al. (2012), is used to stop the enumeration procedure, and the allowable number of states stored in memory is set to 100 million. This value was selected in order to maintain total memory use below the usual memory available on current commodity computers.

6. Computational experiment

In order to assess the quality of the final method, the proposed algorithms were coded in C++, compiled using GCC 4.7 and run on an Intel Xeon machine with an 8-core 2.66 gigahertz CPU and 32-gigabytes RAM, running the Linux operating system. Note that the code does not make use of any form of parallelism, and thus only one core is effectively used. Furthermore, the limit on the number of

Table 1
Summary of results for the lower and upper bounding procedures. For each precedence graph we report the average gap, the average running time and the number of optimal solutions found by the procedures proposed in Dell'Amico et al. (2012), and the procedures described in this paper.

Instances				Dell'Amico et al. (2012)			Heuristic			Machine
Name	n	$\|A\|$	$\#I$	Percent gap	t (seconds)	#Opt	Percent gap	t (seconds)	#Opt	bound
Arcus1	83	113	16	0.00	0.01	16	0.00	0.22	16	0.00
Arcus2	111	176	17	0.83	112.13	13	0.20	7.14	16	1.04
Barthold	148	175	8	0.00	0.09	8	0.00	90.66	8	0.00
Barthold2	148	175	27	0.18	63.62	25	0.00	29.66	27	0.00
Bowman	8	8	1	0.00	0.00	1	0.00	0.00	1	0.00
Buxey	29	36	7	3.31	171.43	4	1.02	0.03	6	3.31
Gunther	35	45	7	1.79	114.29	5	0.00	0.00	7	0.00
Hahn	53	82	5	0.00	0.00	5	0.00	0.02	5	0.00
Heskiaoff	28	39	6	0.00	0.00	6	0.00	0.05	6	0.00
Jackson	11	13	6	0.00	0.00	6	0.00	0.00	6	0.00
Jaeschke	9	11	5	0.00	0.00	5	0.00	0.00	5	2.85
Kilbridge	45	62	10	0.00	0.00	10	0.00	0.04	10	0.00
Lutz1	32	38	6	0.00	0.00	6	0.00	0.00	6	1.51
Lutz2	89	118	11	0.18	94.76	10	0.00	0.22	11	1.20
Lutz3	89	118	12	0.00	0.00	12	0.00	0.03	12	0.34
Mansoor	11	11	3	0.00	0.00	3	0.00	0.00	3	4.76
Mertens	7	6	6	0.00	0.00	6	0.00	0.00	6	0.00
Mitchell	21	27	6	0.00	0.00	6	0.00	0.00	6	0.00
Mukherje	94	181	13	0.61	0.65.52	11	0.00	5.61	13	0.00
Roszieg	25	32	6	1.19	66.67	5	0.00	0.00	6	1.19
Sawyer	30	32	9	2.52	133.50	6	1.60	0.07	7	4.11
Scholl	297	423	26	0.00	5.50	26	0.00	40.26	26	0.15
Tonge	70	86	16	0.00	12.91	16	0.00	0.10	16	0.32
Warnecke	58	70	16	1.21	150.22	10	0.77	0.26	12	3.32
Wee-Mag	75	87	24	0.00	0.04	24	0.00	2.25	24	0.00
Total			269	0.42	44.14	245	0.14	10.94	258	0.75

states in memory leads to a lower memory use than the total available memory (maximum reported usage was below 8 gigabytes).

Two different computational experiments were conducted. The first experiment uses the instance set used in Dell'Amico et al. (2012) and compares the results provided by the algorithms proposed in the same paper with the algorithms proposed in this work. The second experiment uses the set of instances proposed in Otto et al. (2013) and investigates the effect of the structure of the data on the results provided by the algorithm. A time limit was imposed on the execution of the procedure to each instance, which is equal to 7200 seconds, as in Dell'Amico et al. (2012).The following subsections are devoted to each experiment in turn.

6.1. Classical set of assembly line balancing instances

We first report a computational experiment with the set of instances tested in a recent study for the BPP-P, see Dell'Amico et al. (2012). These instances are derived from the classical instance set for the SALBP-1, see Scholl and Becker (2006), and comprises 269 instances with a varying number of items, from 7 to 297, and precedence constraints, 25 different precedence graphs. Different instances are derived from a single precedence graph by modifying the capacity of the bins.

Table 1 provides the general characteristics of each precedence graph (number of items; number of precedence relations; and number of instances derived from the graph) as well as a summary of the performance of the lower and upper bounds described in Dell'Amico et al. (2012) and in this paper. For each group of instances, the average gap, the average running time and the number of optimal solutions provided by both methods before relying to their respective enumeration schemes are reported. The average gap is measured as $(UB - LB)/UB \cdot 100$, in which UB and LB correspond to the best upper/lower bound provided by any of the methods described in each paper, with the exception of the branch-and-bound based procedures. Additionally, we report a column with the average gap reported by the machine scheduling bound (using the average gap formula, and considering LB as the lower bound provided by the machine scheduling bound). The last column is provided in order to illustrate the quality of the lower bounding method used throughout the enumeration phase, and to show that it is able to provide tight bounds.

The summary of results highlights the quality of the proposed lower and upper bounding methods, which are able to close 258 of the 269 instances. Specifically, Dell'Amico et al. (2012) reports that they had to rely on the branch-and-bound procedure on 24 of the 269 instances, while we rely on enumeration in 11 instances. The increase corresponds to cases in which either the lower or the upper bound were improved, see Table 2.

When the running times of both algorithms are compared, the procedure from Dell'Amico et al. (2012) reports lower running times for the groups in which they are able to close all the instances, while in the remaining cases, the algorithms proposed in this paper are faster. This behaviour is attributable to the differences between the upper bounding algorithms, as our algorithm is constructive in nature requires more time to reach a solution. Consequently, the bounded dynamic programming approach seems to provide better solutions for more difficult instances, finding better solutions that the VNS approach, see results in Table 2, but it takes longer to obtain the solution than it may take to explore the neighbourhood of a greedy solution.

The 24 instances in which the algorithms proposed in Dell'Amico et al. (2012) has to rely on enumeration are further analysed in Tables 2 and 3. Table 2 compares the results (in terms of required running time and the reported lower and upper bound) provided by the algorithms proposed in Dell'Amico et al. (2012) and the method proposed in the current paper.

These results further expose the improvements of the proposed methods. The lower bounds of the 11 instances are improved, and 4 improved solutions are found. Running times are also much smaller in all of the instances, leading us to conclude that the proposed Dantzig–Wolfe decomposition and the bounded dynamic programming heuristic provide much better initial bounds on the optimal solution than previous reported methods in the literature.

Table 2
Summary of results before enumeration for the 24 challenging instances reported in Dell'Amico et al. (2012). The results provided by the different algorithms are reported. Lower and upper bounds improved by the proposed methods are highlighted in bold.

Instance				Dell'Amico et al. (2012)			Proposed methods		
Name	n	$\|A\|$	C	LB	UB	t(seconds)	LB	UB	t(seconds)
Arcus2	111	176	6267	29	30	413.44	29	**29**	6.32
Arcus2	111	176	6837	28	29	418.38	28	29	7.47
Arcus2	111	176	7520	26	27	405.81	**27**	27	7.98
Arcus2	111	176	7916	26	27	405.61	**27**	27	6.61
Barthold2	148	175	85	50	51	412.50	50	**50**	19.10
Barthold2	148	175	121	35	36	407.73	35	**35**	28.23
Buxey	29	36	27	13	14	400.50	13	14	0.05
Buxey	29	36	30	12	13	400.52	**13**	13	0.03
Buxey	29	36	33	11	12	400.42	**12**	12	0.04
Gunther	35	45	49	15	16	400.70	**16**	16	0.00
Gunther	35	45	54	15	16	400.70	**16**	16	0.00
Lutz2	89	118	16	49	50	414.19	**50**	50	0.16
Mukherje	94	181	201	26	27	414.19	26	**26**	13.17
Mukherje	94	181	248	23	24	413.41	23	**23**	11.03
Roszieg	25	32	14	13	14	400.47	**14**	14	0.00
Sawyer	30	32	25	14	15	401.84	14	15	0.08
Sawyer	30	32	30	12	13	400.55	12	13	0.10
Sawyer	30	32	33	11	12	400.49	**12**	12	0.08
Warnecke	58	70	54	34	35	413.42	34	35	0.36
Warnecke	58	70	58	32	33	413.30	32	33	0.28
Warnecke	58	70	60	30	31	413.41	30	31	0.46
Warnecke	58	70	62	30	31	413.70	30	31	0.74
Warnecke	58	70	65	28	29	409.80	**29**	29	0.4
Warnecke	58	70	68	27	28	409.67	**28**	28	0.38

Table 3
Summary of results for the exact methods on the 24 challenging instances reported in Dell'Amico et al. (2012). The results provided by the different algorithms are reported. If no results are reported for the proposed method, the instance was solved before enumeration.

Instance				Dell'Amico et al. (2012)				Proposed method			
Name	n	$\|A\|$	C	LB	UB	$Nodes$	t(seconds)	LB	UB	$Nodes$	t(seconds)
Arcus2	111	176	6267	29	29	1,890,268	578.92				
Arcus2	111	176	6837	29	29	2,349,895	844.19	29	29	583	0.06
Arcus2	111	176	7520	27	27	701	0.73				
Arcus2	111	176	7916	27	27	296,146	106.50				
Barthold2	148	175	85	50	51	13,840,213	t. lim.				
Barthold2	148	175	121	35	36	17,708,898	t. lim.				
Buxey	29	36	27	14	14	212	0.89	14	14	47	0.00
Buxey	29	36	30	13	13	29	0.17				
Buxey	29	36	33	12	12	11	0.06				
Gunther	35	45	49	16	16	1	0.00				
Gunther	35	45	54	16	16	1	0.00				
Lutz2	89	118	16	50	50	16	0.33				
Mukherje	94	181	201	26	26	11,283	25.11				
Mukherje	94	181	248	23	23	458	4.50				
Roszieg	25	32	14	14	14	11	0.03				
Sawyer	30	32	25	15	15	990	3.31	15	15	217	0.00
Sawyer	30	32	30	13	13	85	0.39	13	13	32	0.00
Sawyer	30	32	33	12	12	19	0.09				
Warnecke	58	70	54	35	35	2,922	22.23	35	35	85	0.01
Warnecke	58	70	58	33	33	131,016	547.78	33	33	16	0.02
Warnecke	58	70	60	31	31	783,297	3661.83	31	31	494	0.02
Warnecke	58	70	62	30	31	1,707,926	t. lim.	31	31	5573	0.15
Warnecke	58	70	65	29	29	154	1.67				
Warnecke	58	70	68	28	28	398	3.67				

Note that in all of these instances the VNS reached its maximum allowed running time, 400 seconds, without verifying optimality, and the time reported above 400 seconds corresponds to the running time associated to the resolution of the different lower bounding procedures. Hence, in many instances our lower and upper bounding methods run within the time required to obtain the lower bounds in Dell'Amico et al. (2012).

Table 3 reports the results of the exact enumeration method for the 24 "challenging" instances. For those instances in which our method was able to obtain the optimal solution without relying on the exact phase of the procedure, we leave the corresponding entries in blank. For both algorithms, the branch-and-bound from Dell'Amico et al. (2012) and our proposal, we report the final lower and upper bound, the number of explored nodes of the branch-and-bound tree, and the running time in seconds.

The results further highlight the effectiveness of the method. The approach is able to solve all of the remaining instances in very short times while exploring much fewer nodes of the branch-and-bound tree. We point out that this is possible due to the following features of our algorithm: (1) the quality of the initial solution which is the

Table 4
Results for the "large" instance set proposed in Otto et al. (2013). For each group of instances, the results of the lower and upper bounds (columns "Root node") and the exact enumeration algorithm (columns "Enumeration") are provided.

Instance			Root node				Enumeration			
Structure	OS	Item weight	Percent Gap	t (seconds)	# Opt	Machine bound	Percent Gap	t (seconds)	Av. nodes	# Opt
block	low	bimodal	0	7.92	25	0	0	0	0	25
		bottom	0.27	7.51	24	0.27	0	0.03	109.16	25
		middle	1.40	23.82	11	6.25	0.74	2,745.03	3,599,579.8	16
	medium	bimodal	0.68	5.51	21	0.68	0	24.21	38,008	25
		bottom	0	2.29	25	0.44	0	0	0	25
		middle	1.11	6.61	13	5.58	0	61.77	314,655.58	25
chain	low	bimodal	0.54	5.09	22	0.70	0	0	0.36	25
		bottom	0	5.42	25	0.28	0	0	0	25
		middle	1.57	19.24	6	5.47	0.54	2563.51	5,238,787.92	18
	medium	bimodal	2.35	1.28	12	2.69	0	0.11	2,569.48	25
		bottom	0.84	0.52	23	1.95	0	0	4.76	25
		middle	2.12	3.81	5	7,38	0	31.02	461,885.56	25
mixed	low	bimodal	0	9.08	25	0	0	0	0	25
		bottom	0	6.64	25	0	0	0	0	25
		middle	1.72	23.88	10	6.93	0.66	2,288.73	2,813,964.28	18
	medium	bimodal	1.55	1.57	16	1.55	0	0.01	19.24	25
		bottom	0.63	0.56	23	0.85	0	0.00	87.48	25
		middle	1.38	4.27	12	6.55	0	44.05	380,967	25
	high	bimodal	0.79	0.08	19	0.79	0	0.00	11.52	25
		bottom	0	0.04	25	0	0	0	0	25
		middle	1.54	0.31	6	3.90	0	0.01	72.84	25
average/total			0.88	6.45	373	2.49	0.09	369.45	611,934.44	502

optimal solution for all of the instances of the set; (2) a very fast and effective lower bound which allow us to prune early nodes in the search; (3) the application of multiple dominance rules which effectively avoid the exploration of dominated areas of the enumeration tree; and (4) the use of the previously explored state space to avoid further exploration of partial solutions. The later point is to be stressed, as the reuse of the partially explored state space greatly reduces the number of branches to explore and provides a much efficient application of the dominance rules.

As a final remark, let us note that the computer reported in Dell'Amico et al. (2012) corresponds to a 3 gigahertz computer with 2 gigabytes RAM, running under Windows XP, with an imposed a 7200 seconds time limit. Consequently, the CPU times between the algorithms are not completely comparable, but it is safe to assume that the differences are negligible given the clear improvement shown by the previous results.

6.2. New set of assembly line balancing instances

A second analysis was conducted to study the influence of different instance characteristics on the performance of the proposed methods. The experiment uses the instance set proposed in Otto et al. (2013) which contains instances with different graph characteristics and number of items. We focus our analysis on the set "large", $n = 100$. We do not analyse set "huge", $n = 1000$, as previous results for the SALBP-1, Morrison et al. (2014) and Pereira (2015), as well as the results described in Section 6.1 show that a larger number of jobs do not necessarily mean more difficult instances (in fact, the algorithm proposed in Morrison et al. (2014) routinely solves SALBP-1 instances with $n = 1000$ with the exception of all of the instances showing particular distribution of processing times).

The instances proposed in Otto et al. (2013) were generated according to the characteristics of the operations performed in the manufacturing industry and are identified by their number of items, as well as three additional characteristics:

- *Structure of the precedence graph*: Three structures are used. *Block* corresponds to precedence graphs showing tasks with multiple predecessors that divide the graph in different parts. *Chain* corresponds to precedence graphs showing multiple sequences of single-predecessor, single-successor structures. *Mixed* combines the two previous structures.
- *Order strength (OS)*: A numerical measure of the ratio between the total number of precedences and the maximum number of precedences. Three different values were considered: *low* with OS = 0.2, *medium* with OS = 0.6, and *high* with OS = 0.9.
- *Item weights*: According to Otto et al. (2013), usual processing times of tasks in manufacturing industries follow a distribution with peaks at about 1/10 and/or 1/2 of the cycle time. Therefore, three distributions for item weights were considered: *bottom*, in which item weights follow a normal distribution with peak at 1/10 of the bin capacity; *middle*, with item weights following a normal distribution with peak at 1/2; and *bimodal*, with two peaks one at 1/10 and another at 1/2.

The set of instances with 100 items consists of 525 instances, corresponding to 25 instances for each combination of characteristics, with the exception of the *high* order strength, for which only mixed precedence graphs were generated. We refer to Otto et al. (2013) for further details on the instances and their availability for further comparison. Note that no previous results for the BPP-P using these instances is available, and thus we focus our study on the perceived differences on the results given by the algorithms described in this paper.

Table 4 provides the results of the computational experiment. Instances are grouped according to their characteristics, and the average result among the 25 instances is provided. For the lower bounds and the bounded dynamic programming heuristic, the average gap, the average running time, the number of verified optimal solutions and the average lower bound provided by the machine scheduling lower bound are reported. Gaps are measured as the $(UB_{root} - LB)/UB_{root}$, in which UB_{root} represents the best known solution before branching and LB corresponds to the best known lower bound or the machine scheduling lower bound. For the enumeration phase, the average gap, measured as $(UB - LB)/UB$, the average running time, the average number of nodes of the enumeration tree, and the number of verified optimal solutions are reported.

Table 5
Results for the heuristic procedure and the enumeration phase under a 600 seconds time limit.

Instance			Root node			Enumeration (600 seconds)		
Structure	OS	Item	# Opt	# Best	Percent Gap	# Opt	# Best	Percent Gap
Block	Low	Bimodal	25	25	0.00	25	25	0
		Bottom	24	25	0.27	25	25	0
		Middle	11	14	1.40	13	21	0.74
	Medium	Bimodal	21	24	0.68	25	25	0
		Bottom	25	25	0.00	25	25	0
		Middle	13	20	1.11	24	25	0.00
Chain	Low	Bimodal	22	25	0.54	25	25	0
		Bottom	25	25	0.00	25	25	0
		Middle	6	11	1.57	14	21	0.54
	Medium	Bimodal	12	25	2.35	25	25	0
		Bottom	23	25	0.84	25	25	0
		Middle	5	21	2.12	25	25	0
Mixed	Low	Bimodal	25	25	0.00	25	25	0
		Bottom	25	25	0.00	25	25	0
		Middle	10	11	1.72	15	21	0.65
	Medium	Bimodal	16	25	1.55	25	25	0
		Bottom	23	25	0.63	25	25	0
		Middle	12	21	1.38	25	25	0
	High	Bimodal	19	25	0.79	25	25	0
		Bottom	25	25	0.00	25	25	0
		Middle	6	25	1.54	25	25	0
Average/total			373	472	0.88	491	513	0.09

The algorithm is still able to solve most of the problems to optimality, as the optimal solution for 502 of the 525 instances is obtained within the imposed time limit. Nonetheless, the instances of this set are significantly more difficult than the classical SALBP-1 set, particularly the instances with "middle" *item weight distribution* and "low" *order strength*. Those instances contain items with large weights (peak approximately at half the capacity of a bin) and few precedence constraints, leading to less tight lower bounds. Despite that, the proposed algorithm efficiently handles different structures of precedence constraints and item weight distributions, as it is able to obtain most of the optimal solutions.

In order to verify the quality of the solutions under limited running times, Table 5 compares the results given by the bounded dynamic programming approach and the enumeration phase under a 600 s time limit. For each group of instances, three values are reported, namely: (1) the number of proven optimal solutions; (2) the number of best-known solutions found (when compared to the solutions obtained with the 7,200 s time limit); and (3) the optimality gap.

The results in Table 5 further highlight the difficulty of the instances with "middle" item weights and "low" order strength, as these are the only instances for which more than 600 seconds are needed to obtain the best known solution. If the best-known solution is used as an indicator of the quality of the bounded dynamic programming approach, we can see that the best solution was found in 472 of the 525 instances, including all but one of the instances in which the item weight follows a "bimodal" or "bottom" item weight distribution.

7. Conclusions

In this work we have investigated the bin packing problem with precedence constraints, a recently proposed generalization of the bin packing problem, (Dell'Amico et al., 2012), which models some characteristics of several scheduling and assembly line balancing problems. In order to provide a solution procedure for the problem, two lower bounds derived from the simple assembly line balancing problem as well as a dynamic programming based heuristic and an branch, bound and remember enumeration method are proposed. Both methods explore the state space of the dynamic programming formulation avoiding the repeated exploration of equivalent partial solutions. Furthermore, the dynamic programming heuristic and the enumeration method are designed in such a way that the same solution space is explored and the portions of the solution space already explored by the dynamic programming approach are not explored again by the branch, bound and remember method.

In addition to the application of novel lower bounds and the identification of equivalent partial solutions in the state space, both methods make use of different dominance rules to detect dominated states. The early detection of these dominated states allows the algorithm to reduce the number of states stored in memory, and to avoid redundant exploration. In addition to their usefulness for the present problem, two of the novel dominance rules, namely the generalized maximum load rule and the generalized item dominance rule, can be extended to some assembly line balancing problems, like the simple assembly line balancing problem.

The results show the effectiveness of the proposed final method, which is able to solve all of the previously considered instances, outperforming the previous state-of-the-art procedure from the literature. A second computational experiment is conducted in order to identify the characteristics of the instances which pose difficulties to the proposed method. While the proposed method does not obtain the optimal solution for all of the instances in the second set, the results indicate that most instances are still solved to optimality. Furthermore, the results of this second computational experiment show that the most difficult instances for the algorithm correspond to those with large item weights and low order strength on the precedence graph.

As a final reflection, we would like to draw attention to some of the ideas of the proposed method, like the dominance rules, which can be generalized to similar problems, like the assembly line or the transfer line balancing problem. It is our opinion that the ability of the method to share the explored state space between the heuristic and the exact phases of the algorithm, constitutes a good starting point to develop efficient solution methods for problems featuring similar characteristics.

Aknowledgements

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.

References

Augustine, J., Banerjee, S., & Irani, S. (2009). Strip packing with precedence constraints and strip packing with release times. *Theoretical Computer Science, 410*(38-40), 3792–3803. doi:10.1016/j.tcs.2009.05.024.

Bautista, J., & Pereira, J. (2009). A dynamic programming based heuristic for the assembly line balancing problem. *European Journal of Operational Research, 194*(3), 787–794. doi:10.1016/j.ejor.2008.01.016.

Bautista, J., & Pereira, J. (2011). Procedures for the time and space constrained assembly line balancing problem. *European Journal of Operational Research, 212*(3), 473–481. doi:10.1016/j.ejor.2011.01.052.

Borisovsky, P., Dolgui, A., & Kovalev, S. (2012). Algorithms and implementation of a set partitioning approach for modular machining line design. *Computers & Operations Research, 39*(12), 3147–3155. doi:10.1016/j.cor.2012.03.017.

Casazza, M., & Ceselli, A. (2014). Mathematical programming algorithms for bin packing problems with item fragmentation. *Computers & Operations Research, 46*, 1–11. doi:10.1016/j.cor.2013.12.008.

Christofides, N., Mingozzi, A., & Toth, P. (1981). State space relaxations procedures for the computation of bounds to routing problems. *Networks, 11*(2), 145–164. doi:10.1002/net.3230110207.

Clautiaux, F., Alves, C., & Valério de Carvalho, J. (2010). A survey of dual-feasible and superadditive functions. *Annals of Operations Research, 179*, 317–342. doi:10.1007/s10479-008-0453-8.

Coffman, E. G., Galambos, G., Martello, S., & Vigo, D. (1999). Bin packing approximation algorithms: combinatorial analysis. In D.-Z. Du, & P. M. Pardalos (Eds.), *Handbook of combinatorial optimization*. Springer. doi:10.1007/978-1-4757-3023-4_3.

Congram, R. K., Potts, C. N., & van de Velde, S. L. (2002). An iterated dynasearch algorithm for the single-machine total weighted tardiness scheduling problem. *INFORMS Journal on Computing, 14*(1), 52–67. doi:10.1287/ijoc.14.1.52.7712.

Cormen, T. H., Leiserson, C. E., Rivest, R. L., & Stein, C. (2009). *Introduction to algorithms*. MIT Press.

Dell'Amico, M., Díaz, J. C., & Iori, M. (2012). The bin packing problem with precedence constraints. *Operations Research, 60*(6), 1491–1504. doi:10.1287/opre.1120.1109.

Delorme, X., Dolgui, A., & Kovalyov, M. Y. (2012). Combinatorial design of a minimum cost transfer line. *Omega, 40*(1), 31–41. doi:10.1016/j.omega.2011.03.004.

Dolgui, A., & Battaïa, O. (2013). A taxonomy of line balancing problems and their solution approaches. *International Journal of Production Economics, 142*(2), 259–277. doi:10.1016/j.ijpe.2012.10.020.

Fekete, S. P., & Schepers, J. (2001). New classes of fast lower bounds for bin packing problems. *Mathematical programming, 31*, 11–31. doi:10.1007/s101070100243.

Fernandes Muritiba, A. E., Iori, M., Malaguti, E., & Toth, P. (2010). Algorithms for the bin packing problem with conflicts. *INFORMS Journal on Computing, 22*(3), 401–415. doi:10.1287/ijoc.1090.0355.

Garey, M. R., Graham, R., & Johnson, D. S. (1976). Resource constrained scheduling as generalized bin packing. *Journal of combinatorial theory (A), 21*, 257–298. doi:10.1016/0097-3165(76)90001-7.

Gendreau, M., Laporte, G., & Semet, F. (2004). Heuristics and lower bounds for the bin packing problem with conflicts. *Computers & Operations Research, 31*(3), 347–358. doi:10.1016/S0305-0548(02)00195-8.

Gromicho, J., van Hoorn, J. J., Kok, A. L., & Schutten, J. M. J. (2012). Restricted dynamic programming: a flexible framework for solving realistic VRPs. *Computers & Operations Research, 39*(5), 902–909. doi:10.1016/j.cor.2011.07.002.

Haouari, M., & Gharbi, A. (2005). Fast lifting procedures for the bin packing problem. *Discrete Optimization, 2*(3), 201–218. doi:10.1016/j.disopt.2005.06.002.

Hifi, M., & Michrafy, M. (2007). Reduction strategies and exact algorithms for the disjunctively constrained knapsack problem. *Computers & Operations Research, 34*(9), 2657–2673. doi:10.1016/j.cor.2005.10.004.

Josuttis, N. M. (2012). *The C++ standard library, second edition*. Addison Wesley.

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 Journal of Operational Research, 236*(2), 403–409. doi:10.1016/j.ejor.2013.11.033.

Otto, A., Otto, C., & Scholl, A. (2013). Systematic data generation and test design for solution algorithms on the example of SALBPGen for assembly line balancing. *European Journal of Operational Research, 228*(1), 33–45. doi:10.1016/j.ejor.2012.12.029.

Peeters, M., & Degraeve, Z. (2006). An linear programming based lower bound for the simple assembly line balancing problem. *European Journal of Operational Research, 168*(3), 716–731. doi:10.1016/j.ejor.2004.07.024.

Pereira, J. (2015). Empirical evaluation of lower bounding methods for the simple assembly line balancing problem. *International Journal of Production Research, 53*(11), 3327–3340. doi:10.1080/00207543.2014.980014.

Sadykov, R., & Vanderbeck, F. (2013). Bin packing with conflicts: a generic branch-and-price algorithm. *INFORMS Journal on Computing, 25*(2), 244–255. doi:10.1287/ijoc.1120.0499.

Scholl, A., & Becker, C. (2006). State-of-the-art exact and heuristic solution procedures for simple assembly line balancing. *European Journal of Operational Research, 168*(3), 666–693. doi:10.1016/j.ejor.2004.07.022.

Sewell, E., & Jacobson, S. (2012). A branch, bound, and remember algorithm for the simple assembly line balancing problem. *INFORMS Journal on Computing, 24*(3), 433–442. doi: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 Journal of Operational Research, 235*(3), 740–754. doi:10.1016/j.ejor.2013.11.005.

Tirpak, T. M. (2008). Developing and deploying electronics assembly line optimization tools : A Motorola case study. *Decision Making in Manufacturing and Services, 2*(1), 63–78. doi:10.7494/dmms.2008.2.2.63.

Ullmann, J. R. (1976). An algorithm for subgraph isomorphism. *Journal of the ACM, 23*(I), 31–42. doi:10.1145/321921.321925.

Valério De Carvalho, J. M. (2002). LP models for bin packing and cutting stock problems. *European Journal of Operational Research, 141*(2), 253–273. doi:10.1016/S0377-2217(02)00124-8.

Vanderbeck, F. (1999). Computational study of a column generation algorithm for bin packing and cutting stock problems. *Mathematical Programming, 86*, 565–594. doi:10.1007/s101079900096.

Vilà, M., & Pereira, J. (2013). An enumeration procedure for the assembly line balancing problem based on branching by non-decreasing idle time. *European Journal of Operational Research, 229*(1), 106–113. doi:10.1016/j.ejor.2013.03.003.

Vilà, M., & Pereira, J. (2014). A branch-and-bound algorithm for assembly line worker assignment and balancing problems. *Computers & Operations Research, 44*, 105–114. doi:10.1016/j.cor.2013.10.016.