International Journal of Production Research

ISSN: 0020-7543 (Print) 1366-588X (Online) Journal homepage: http://www.tandfonline.com/loi/tprs20

A batching-move iterated local search algorithm for the bin packing problem with generalized precedence constraints

Raphael Kramer, Mauro Dell'Amico & Manuel Iori

To cite this article: Raphael Kramer, Mauro Dell'Amico & Manuel Iori (2017): A batching-move iterated local search algorithm for the bin packing problem with generalized precedence constraints, International Journal of Production Research, DOI: 10.1080/00207543.2017.1341065

To link to this article: http://dx.doi.org/10.1080/00207543.2017.1341065

Published online: 06 Jul 2017.

International Journal of Production Research, 2017
https://doi.org/10.1080/00207543.2017.1341065

A batching-move iterated local search algorithm for the bin packing problem with generalized precedence constraints

Raphael Kramer*, Mauro Dell'Amico and Manuel Iori

DISMI, University of Modena and Reggio Emilia, Reggio Emilia, Italy

(*Received 24 October 2016; accepted 5 June 2017*)

In this paper, we propose a generalisation of the bin packing problem, obtained by adding precedences between items that can assume heterogeneous non-negative integer values. Such generalisation also models the well-known Simple Assembly Line Balancing Problem of type I. To solve the problem, we propose a simple and effective iterated local search algorithm that integrates in an innovative way of constructive procedures and neighbourhood structures to guide the search to local optimal solutions. Moreover, we apply some preprocessing procedures and adapt classical lower bounds from the literature. Extensive computational experiments on benchmark instances suggest that the developed algorithm is able to generate good quality solutions in a reasonable computational time.

Keywords: bin packing; assembly line balancing; generalised precedence constraints; iterated local search; batching moves

1. Introduction

The *bin packing problem* (BPP) requires to pack a set $N = \{1, 2, \dots, n\}$ of one-dimensional items, each having weight w_j, into the minimum number of identical bins of capacity C. The problem has been the subject of several research efforts since the 60's, because it models many real-life issues, arising in areas such as logistics, scheduling, computer science, industry and others (see, e.g. Wäscher, Haußner, and Schumann 2007). From the management point of view, it favours the efficient use of resources in many contexts: in inventory management, it helps to improve the storage area usage of a warehouse, indicating the best position of products and/or raw materials; in systems involving cutting processes, it defines the best cutting plane for minimising material waste; in project management, it can be used to assign tasks to time periods aiming at the minimisation of the completion time. For recent surveys on the BPP, the reader is referred to Valério de Carvalho (2002) (linear programming methods), Coffman et al. (2013) (approximation algorithms) and Delorme, Iori, and Martello (2016) (exact algorithms and computational results).

In this paper we present the *bin packing problem with generalized precedence constraints* (BPP-GP), that extends the BPP by including a set of weighted precedence relationships between pairs of items. The precedence weights take heterogeneous non-negative integer values, and are used to represent a minimum distance between the indices of the bins containing the two items in the pair. This problem generalises both the well-known simple assembly line balancing problem of type I and the bin packing problem with precedence constraints, which are cases with homogeneous precedence weights (in the former all precedences have value zero, and in the latter value one). Assembly line balancing problems, in particular, are very common in the context of industrial production (e.g. Liu and Chen 2002; Sheu and Chen 2008; Cortés, Onieva, and Guadix 2010; Otto and Otto 2014).

In addition to the applications solved by the two aforementioned particular problems, the BPP-GP models those situations where the distance between two items (operations) must be large enough to guarantee the structural properties of the products. This is a requirement arising in many industrial problems (see, e.g. Tirpak 2008) as well as in civil construction (see, e.g. Chassiakos and Sakellaropoulos 2005), where certain structures require to wait for some days to improve their resistance before aggregating new components. The high interdependence between activities is one of the main difficulties faced by construction and assembly line companies aiming at cost minimisation. When the number of activities is large, this goal can be achieved in an efficient manner only with the assistance of computational techniques, such as mathematical programmes and heuristic algorithms.

To solve the problem, we propose a simple and effective algorithm, based on the *iterated local search* paradigm and on an extremely large neighbourhood, implemented using an innovative *Batching-Move* mechanism. This mechanism builds upon that of compounding moves (see, e.g. Mouthuy 2011) and allows to design an algorithm that uses a single parameter,

*Corresponding author. Email: raphael.kramer@unimore.it

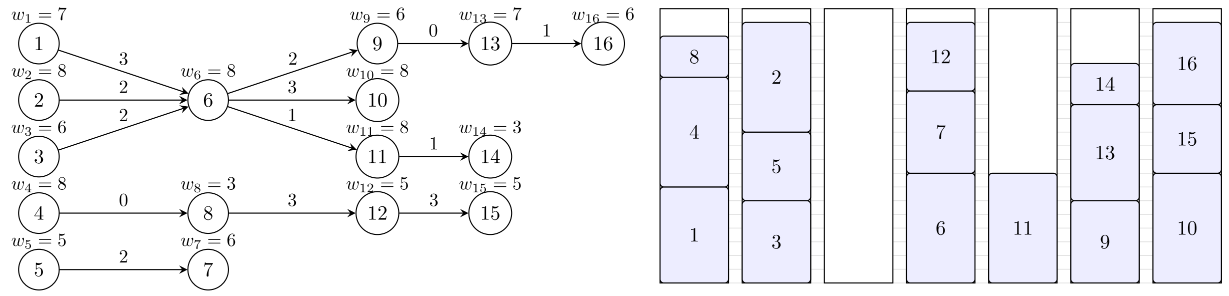

Figure 1. Precedence graph and an optimal solution for a BPP-GP instance with 16 items ($C = 20$).

thus giving robustness and cancelling the usual time-consuming tuning phases. In addition, we enriched the algorithm with classical constructive heuristics, preprocessing techniques and lower bounding procedures taken from the literature and adapted to the BPP-GP. This is a full and updated version of our work presented at the *4th International Symposium on Combinatorial Optimization* (Kramer, Dell'Amico, and Iori 2016).

The quality of the proposed algorithm is evaluated by means of extensive computational experiments on benchmark instances. On the particular problem cases, where all precedence weights take value 0 (simple assembly line balancing problem of type I) or 1 (bin packing problem with precedence constraints), we compare our algorithm with the best algorithms in the existing literature. On the newly introduced BPP-GP, we compare instead with a commercial solver. In all cases, we show that our approach finds high quality solutions in short computing times.

The remainder of this paper is organised as follows. The BPP-GP is formally defined in Section 2. Section 3 presents the related literature, particularly focusing on assembly line problems and BPP variants with precedence constraints. Section 4 describes our preprocessing and lower bounding procedures, while Section 5 presents our algorithm. Extensive computational experiments are provided in Section 6 and conclusions are finally drawn in Section 7.

2. Problem description

The BPP-GP can be formally defined as follows. We are given a set $N = \{1, 2, \dots, n\}$ of items, each having positive weight w_j, a set $M = \{1, 2, \dots, m\}$ of identical bins, each having capacity C, and an arc set A used to represent the generalised precedence relationships. With each arc $(j, k) \in A$ is associated with a non-negative integer value t_{jk}. Let i_k be the index of the bin receiving item k and i_j the index of the bin containing item j. The precedence constraints impose that $i_k - i_j \geq t_{jk}$, for all $(j, k) \in A$. The aim of the BPP-GP is to pack all items by satisfying capacity and precedence requirements, and minimising the index of the last bin in which an item is packed.

Figure 1 depicts a simple BPP-GP instance with 16 items having weights $w = [7, 8, 6, 8, 5, 8, 6, 3, 6, 8, 8, 5, 7, 3, 5, 6]$ and bins of capacity $C = 20$. The left part of the figure gives the precedence graph and the right part an optimal solution of value 7. Note that bin 3 is unused for packing but required for satisfying the generalised precedence relationships.

By introducing two sets of binary variables, y_i taking value 1 if bin $i \in M$ is used, and x_{ij} taking value 1 if item $j \in N$ is allocated to bin $i \in M$, the BPP-GP can be modelled as the following *Integer Linear Program* (ILP).

$$\min \quad \sum_{i \in M} y_i \tag{1}$$

$$\text{s.t.} \quad \sum_{i \in M} x_{ij} = 1 \qquad j \in N \tag{2}$$

$$\sum_{j \in N} w_j x_{ij} \leq C y_i \qquad i \in M \tag{3}$$

$$\sum_{i \in M} i x_{ik} \geq \sum_{i \in M} i x_{ij} + t_{jk} \qquad (j, k) \in A \tag{4}$$

$$y_i \geq y_{i+1} \qquad i = 1, \dots, m-1 \tag{5}$$

$$y_i \in \{0, 1\} \qquad i \in M \tag{6}$$

$$x_{ij} \in \{0, 1\} \qquad i \in M, j \in N. \tag{7}$$

The objective function (1) seeks to minimise the number of used bins. Constraints (2) state that every item must be assigned to exactly one bin, while constraints (3) ensure that the total weight of items packed in a bin does not exceed its capacity. Constraints (4) guarantee that precedence relationships are satisfied. Constraints (5) impose that, if a bin $i+1$ is used in the optimal solution then the precedent bin i must be used as well. Finally, constraints (6) and (7) force variables to be binary.

Under constraint (5), minimising the sum of used bins is equivalent to minimise the index of the last used bin. Note that one could be interested in the BPP-GP variant in which the aim is not to minimise the index of the last bin used, but just the number of bins in which a packing actually occurs. By looking at Figure 1, that would imply that the solution had cost 6 instead of 7 (as bin 3 would not be counted). This variant could be interesting, for example, in problems where the workforce can be moved to other projects, while the current one is on hold because of $t_{jk} \geq 2$ precedences. This BPP-GP variant can be easily modelled as an ILP by dropping constraint (5) out of model (1)–(7). Note also that this does not happen for BPP-P and SALBP-I, for which $t_{jk} \leq 1$ holds for all $(j, k) \in A$.

Some additional notation is needed to describe our procedures. We use $j \prec k$ to state that item j must precede item k. Let $\bar{N} = N \cup \{0, n+1\}$, where 0 and $n+1$ are two dummy items that represent, respectively, the beginning and the end of all activities. Let also $\bar{A} = A \cup \{(0, j), (j, n+1)\}$, for all $j \in N$, and $t_{jk} = 0$, for all $(j, k) \in \bar{A} \setminus A$. We denote $G = (\bar{N}, \bar{A})$ as the *precedence graph*. In addition, we let $head_j$, respectively $tail_j$, be the minimum number of bins that must precede, respectively succeed, j in a feasible solution.

Head and tail values can be computed by solving a longest path problem over G using the classical *critical path method* (CPM). For the example in Figure 1, this results in $head = [0, 0, 0, 0, 0, 3, 2, 0, 5, 6, 4, 3, 5, 5, 6, 6]$ and $tail = [6, 5, 5, 6, 2, 3, 0, 6, 1, 0, 1, 3, 1, 0, 0, 0]$. The computation of such values is useful for the development of practically all our algorithms below. Based on Pastor and Ferrer (2009), model (1)–(7) may be improved by imposing $x_{ij} = 0$ for all $i \in M, j \in N : i \leq head_j$ or $i > U - tail_j$, with U being an upper bound on the optimal solution value.

3. Related problems

In this section, we present a brief discussion of the large literature that studied problems involving both capacity and precedence constraints.

A generalised version of the BPP, known as the *generalized bin packing problem* (G-BPP), was introduced by Garey et al. (1976). In the G-BPP, we have unitary precedences, the items have s weights and the bins have s capacities, with $s \geq 1$. The authors proposed approximation algorithms.

The special G-BPP case in which $s = 1$ is known as the *bin packing problem with precedence constraints* (BPP-P). The problem was solved by Dell'Amico, Díaz-Díaz, and Iori (2012) by means of an exact branch-and-bound algorithm combined with reduction criteria, as well as lower and upper bounding procedures. Recently, Pereira (2016) proposed for the BPP-P a heuristic based on dynamic programming and an exact enumeration procedure. With these algorithms he was able to solve to proven optimality all the instances considered in Dell'Amico, Díaz-Díaz, and Iori (2012). To this regard, we also mention that Augustine, Banerjee, and Irani (2009) proposed a polynomial-time approximation algorithm for the complex *strip packing problem with precedence constraints*, which is the generalisation of the BPP-P in which items and bins are two-dimensional rectangles.

An optimisation problem which is a similar problem to the BPP-P but received much more attention in the literature is the *Simple Assembly Line Balancing Problem of type I* (SALBP-I). The SALBP-I aims at assigning tasks to workstations such that all precedence constraints are fulfilled, the time used in each station does not exceed a given cycle time, and the number of workstations is minimised. In the BPP notation, the workstations correspond to the bins, the tasks to the items and the cycle time to the capacity of the bins. In practice, the SALBP-I is a special case of the BPP-GP in which all precedence constraints have value $t_{jk} = 0$. For a general literature review on assembly line balancing problems we refer the interested reader to the surveys by Becker and Scholl (2006) and Scholl and Becker (2006). Interesting contributions on lower bounds and the state-of-the-art upper bounds can be found in Morrison, Sewell, and Jacobson (2014), Pape (2015), and Pereira (2015). Morrison, Sewell, and Jacobson (2014) proposed an efficient branch, bound and remember algorithm. Pape (2015) presented a survey on heuristic methods, showing the advantages and disadvantages of their use on the basis of a wide set of computational experiments. Pereira (2015) proposed an empirical evaluation of the lower bound methods for the SALBP-I, as well as a column generation algorithm that led to several improvements on the best known lower bounds so far, helping to prove the solution optimality of some open instances.

Recent related works on assembly systems were presented by Nicosia and Pacifici (2017) and Kucukkoc and Zhang (2017). In the former, a heuristic algorithm is proposed to minimise the makespan on a scheduling problem with parallel machines and jobs having precedences structured as a caterpillar graph. In the latter, a mixed-model parallel U-shaped

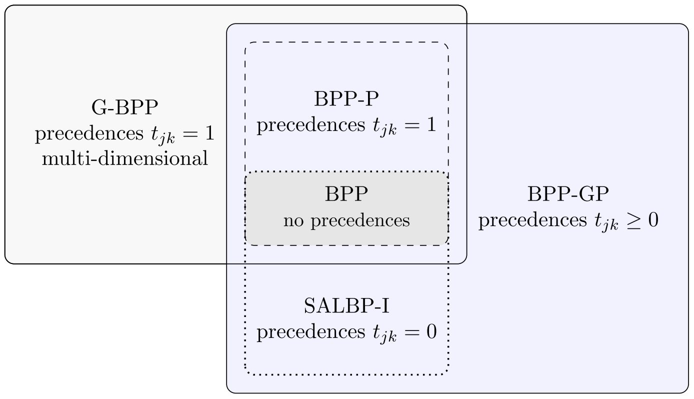

Figure 2. Illustration of generalisation problems for the BPP.

assembly line is considered, where the workstations are located between two lines, and the operators may perform activities on both lines. The problem consists in minimising the number of workstations and is solved by a heuristic algorithm.

The relation among the optimisation problems that we discussed is depicted in Figure 2. We also mention that the use of generalised precedence relationships is common in the context of *resource constrained project scheduling problem* (RCPSP), and indeed the BPP-GP can be seen as a particular case of the RCPSP with minimal time lags (see, e.g. De Reyck and Herroelen 1998; Klein 2000; Bianco and Caramia 2012). RCPSPs are very complex problems, where operations (items) may last more than one day (bin) and may consume more than one type of resource. A minimal time-lag constraint between operations j and k states that k can only start t_{jk} days after the completion of j. The RCPSP with minimal time lags reduces to the BPP-GP in the specific case where all activities have the same unitary duration and consume only one renewable resource. Using the notation proposed by Brucker et al. (1999) to classify the RCPSPs, the BPP-GP could thus be classified as $PS1|p_j = 1, prec, temp|C_{max}$. For a recent survey on RCPSP we refer the reader to Hartmann and Briskorn (2010).

4. Preprocessing and lower bound procedures

To speed-up the convergence of our algorithm, we developed a few preprocessing and lower bounding procedures. These have been obtained by adapting to the BPP-GP some classical algorithms from the literature and in particular from Dell'Amico, Díaz-Díaz, and Iori (2012).

4.1 *Preprocessing*

We make use of two preprocessing procedures. The first procedure modifies G by adding new arcs and by increasing the precedence weights as follows. Let $G_{jk} = (\bar{N}_{jk}, \bar{A}_{jk})$ be a subgraph of G, where $\bar{N}_{jk}$ denotes the subset of items in all paths connecting $j \in \bar{N}$ to $k \in \bar{N}$ (including both j and k), and $\bar{A}_{jk}$ denotes the subset of arcs that belong to these paths. In addition, let z^*_{jk} be the optimal solution value for the BPP-GP instance defined by the subgraph G_{jk}. One can note that t_{jk} can be updated to $z^*_{jk} - 1$, for every pair of items j and k in G, because in any feasible solution $i_k \geq i_j + z^*_{jk} - 1$. Computing z^*_{jk} is time consuming, thus in our implementation we use instead the continuous BPP lower bound and update t_{jk} to $\max\{t_{jk}, \lceil \sum_{j \in \bar{N}_{jk}} w_j/C \rceil - 1\}$. In case there was no direct arc between j and k, then a new arc is added to G.

The second preprocessing attempts to lift the item weights and is based on the solution of the *subset sum problem with conflicts* (SSPC). The *subset sum problem* (SSP) is to pack a set of weighted items in a capacitated bin, by maximising the total weight of the packed items without exceeding the bin capacity. The SSPC is the generalisation of the SSP where an additional family of constraints imposes pairwise conflicts among the items (if two items are in conflict, then at most one of them can be packed in the bin). In our case, a conflict between a pair of items j and k occurs when $j \prec k$ and $t_{jk} > 0$, for any $j, k \in N$. Let $z^*_{SSPC}(S_j, C_j)$ denote the optimal solution value of an SSPC instance in which $C_j = C - w_j$ and S_j is the set of items that are not in conflict with j (note that there can be conflicts among the items in S_j). The total weight of items packed in the same bin with j cannot exceed $z^*_{SSPC}(S_j, C_j)$. Thus, if $vSSPC(S_j, C_j) < C_j$ we set $w_j = w_j + C_j - z^*_{SSPC}(S_j, C_j)$, for every $j \in N$. By defining $A'_j = \{(k, l) \in A : k, l \in S_j \text{ and } t_{kl} > 0\}$, as the set of arcs associated with the items in S_j with precedence values greater than 0, the SSPC can be formulated as:

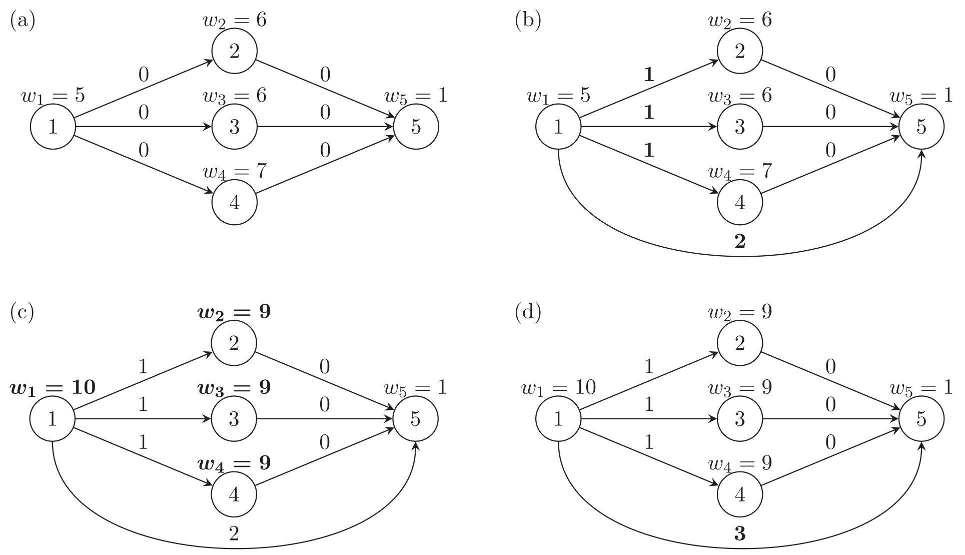

Figure 3. Example of preprocessing. (a) Original instance ($C = 10$). (b) Instance after precedence preprocessing. (c) Instance after SSPC preprocessing. (d) Instance obtained at the end of the iterative process.

$$z^*_{SSPC}(S_j, C_j) = \max\left\{\sum_{k\in S_j} w_k x_k : \sum_{k\in S_j} w_k x_k \le C_j;\ x_k + x_l \le 1,\ (k,l) \in A'_j;\ x_k \in \{0,1\},\ k \in S_j\right\}. \tag{8}$$

The two procedures are interdependent in such a way that a modification performed by the first one may have consequences on the result of the second one, and vice-versa. For this reason, we embedded them inside an iterative algorithm that invokes them, one after the other, until none of them produces a modification. A simple example is depicted in Figure 3.

4.2 *Lower bounds*

In total, five lower bound (LB) procedures derived from the literature were implemented. These procedures are either based on classical techniques originally developed for the BPP, or on the computation of the *longest path* over the acyclic precedence graph, or on combinations of these two concepts. They can be described as follows:

- **LB₁**: a trivial relaxation can be obtained by ignoring the precedences and rounding up to the nearest integer the continuous BPP lower bound, i.e. $LB_1 = \lceil \sum_{j=1}^{n} w_j / C \rceil$.
- **LB₂**: let ℓ be the length of the longest path from 0 to $n+1$ in G, then $LB_2 = \ell + 1$.
- **LB₃**: an immediate bound is given by $LB_3 = \max\{LB1, LB2\}$.
- **LB₄**: let P be the subset of items in a longest path from 0 to $n+1$, $Q = N \setminus P$, and B^P be the minimum set of bins that must be used to pack the items in P, while respecting capacity and precedence constraints. B^P can be computed easily because the items in P form a chain, so to pack two consecutive items we can use the next fit procedure if the precedence weight between them is 0, or pack them in separate bins if the value is 1 or larger. By letting φ be the maximum (fractional) weight of items in Q that can be packed in the bins of B^P, our fourth lower bound is

$$LB_4 = |B^P| + \left\lceil \frac{\sum_{j\in Q} w_j - \varphi}{C} \right\rceil. \tag{9}$$

To compute the value of φ we solve a *max-flow* procedure as follows. We create a graph in which: (1) a source node s is connected to any item $j \in Q$ with an arc of capacity w_j; (2) an item $j \in Q$ is connected with an arc of capacity w_j to a bin $i \in B^P$ if $t_{jk} < 1$ and $t_{kj} < 1$ for all items k packed in i, and $i > head_j$ and $i + tail_j \le UB$ (UB being a valid upper bound value); and (3) any bin in B^P is connected to a target node t with an arc whose capacity is equal to the residual bin capacity. The value of φ is then the max-flow value from 0 to $n+1$. A graphical example of the way LB_4 is computed is shown in Figure 4.

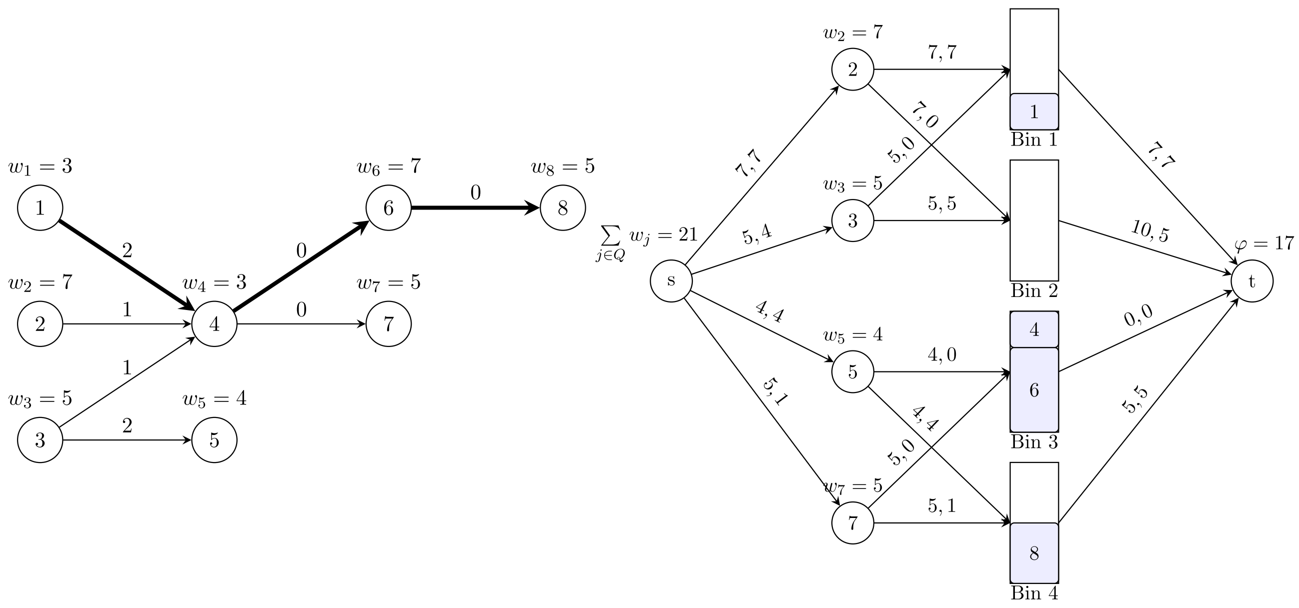

Figure 4. Example for LB_4: (a) original instance ($C = 10$) and a longest path; (b) max-flow solution (arc capacities and optimal flows are reported above the arcs). In this example $LB_4 = 5$.

- **LB₅**: let $I_{r,q} = \{j \in N : head_j \geq r \text{ and } tail_j \geq q\}$, then our fifth lower bound is obtained by computing

$$LB_5(LB_\alpha) = \max_{\substack{r=0,\ldots,\ell \\ q=0,\ldots,\ell-r}} \left\{ r + LB_\alpha(I_{r,q}) + q \right\}, \tag{10}$$

with LB_α being a valid BPP-GP lower bound (details of our implementation are reported in Section 6).

5. Heuristic solution

A simple solution approach is to use the mathematical formulation (1)–(7) introduced in Section 2 and a state-of-the-art *Mixed Integer Linear Programming* (MILP) solver with a maximum time limit. Detailed information on our computational experiments will be presented in Section 6, but it is worth disclosing here that with this approach we were able to solve no more than two-thirds of the benchmark instances with 100 items and a very small number of those with 1000 items. Therefore, we implemented a simple and powerful metaheursitic to find high quality solution for all instances.

The reference framework of our algorithm is the Iterated Local Search method introduced by Lourenço, Martin, and Stützle (2002). In the design of the algorithm we put special attention to the simplicity of the algorithm, to its reproducibility, and to insert as few parameters to be tuned as possible. The final method we propose has a single numeric parameter and a terse design. Moreover, to combine effectiveness, robustness and simplicity, we decided to adopt an extremely large neighbourhood obtained by applying, at each iteration of the local search, a *batch* of moves selected from a few different neighbourhoods. The resulting *Batching-Move Iterated Local Search* (BM-ILS) is outlined in Algorithm 1.

The algorithm starts by applying the preprocessing and lower bounding procedures presented in the previous sections. Then, it constructs a first feasible solution using some greedy methods described in Section 5.1 and starts a loop that alternates an intensification phase (procedure BM-LS of Section 5.2) and a diversification phase (procedure DESTROYSHRINKREPAIR of Section 5.3). The loop terminates when the number of bins in the best solution found equals the lower bound or when a time limit expires.

Observe that the only numerical parameter in BM-ILS is the value returned by function K_MAX() which is used to define the argument ρ of the diversification phase (see Section 5.3). Our experiments showed that the performance of BM-ILS has a very mild dependence from this value, which was set to $\max\{8, n/15\}$ to exhibit a moderate grow with the instance size.

Algorithm 1. Batching-Move Iterated Local Search

```
1: function BM- ILS($Instance$)
2:     $ImprovedInstance \leftarrow$ PREPROCESSING($Instance$)
3:     $LB \leftarrow$ COMPUTEBOUNDS($ImprovedInstance$)
4:     $x_{incumbent} \leftarrow$CONSTRUCTINITIALSOLUTION($ImprovedInstance$)          ▷ Construct first solution
5:     $x_{best} \leftarrow x_{incumbent}$
6:     repeat
7:         $x_{incumbent} \leftarrow$BM- LS($x_{incumbent}$)          ▷ Intensification using batch of moves
8:         if $z(x_{incumbent}) < z(x_{best})$ then $x_{best} \leftarrow x_{incumbent}$
9:         $\rho \leftarrow$ random(1,K_MAX($ImprovedInstance$))
10:        $x_{incumbent} \leftarrow$ DESTROYSHRINKREPAIR($x_{incumbent}, \rho$)          ▷ Diversification
11:    until $z(x_{best}) = LB$ or TIMEOUT( )
12:    return $x_{best}$
```

5.1 *Constructive procedures*

The initial solution is built using three fast independent procedures and selecting the best solution produced by them.

The first procedure is an adaptation of the classical *First Fit* algorithm (FF) for the BPP, obtained by taking care of the precedence constraints. More precisely, FF considers one item at a time (by precedences) and packs it in the first opened bin where both the capacity constraint and the precedence constrains are satisfied. If no such bin exists, one or more new bins are opened and the item is packed (note that, due to the precedence constraints, if $\overline{m}$ bins are already opened, the new bin chosen for packing is not necessarily the $(\overline{m}+1)$th, as it would happen for the BPP and the BPP-P).

The second procedure is another greedy method that fixes the packing of a bin $i = 1, 2, \ldots,$ at a time. Given the current bin i we select the set $\overline{N}_i$ of items that could be packed in i, and solve a *Subset Sum Problem with Pairing Constraints* (SSP-PC) to maximise the load of the bin. Set $\overline{N}_i$ is initialised with all the unpacked items with no predecessors, plus all the unpacked items k such that each predecessor j with $t_{jk} > 0$ has been packed in the first $i - t_{jk}$ bins. The set is extended by adding the unpacked items h such that all paths of G between an item of $\overline{N}_i$ and h have zero length. The SSP-PC aims to pack in the bin the items of $\overline{N}_i$ which maximise the bin load, without exceeding the bin capacity, and with the following additional pairing constraint. Let $A''_i = \{(j,k) \in A : j, k \in \overline{N}_i \text{ and } t_{jk} = 0\}$ be the set of arcs associated with the items in $\overline{N}_i$ with precedence values equal to 0. Then, given a pair of items $j, k \in \overline{N}_i$ with $(j,k) \in A''_i$, k can only be packed in i if j is packed too. The SSP-PC can be formulated as:

$$z^*_{SSP\text{-}PC}(i) = \max\left\{\sum_{j\in\overline{N}_i} w_j x_j : \sum_{j\in\overline{N}_i} w_j x_j \le C;\, x_k \le x_j,\, (j,k) \in A''_i;\, x_j \in \{0,1\},\, j \in \overline{N}_i\right\}. \qquad (11)$$

The second procedure iterates until all items have been assigned, or the solver is not able to find a feasible solution in the given time limit. The third procedure is to simply use a MILP solver for model (1)–(7) with a very short amount of time.

Note that the second greedy could terminate without a solution (although this never occurred in our computational experiments), but FF always produces a feasible solution, so we are guaranteed to have a starting point for the BM-ILS.

5.2 *Batching-Move Local Search*

The *Batching-Move Local Search* (BM-LS) procedure used in our method is sketched in Algorithm 2.

The BM-LS performs a series of iterations until no further improvement for the current solution is found. At each iteration it uses procedure SELECTIMPROVING to scan an *elementary* neighbourhood and to collect in $MoveSet$ all the moves which result to be improving according to a *fitness* function (see (12)). Then, it sorts the moves in $MoveSet$ by non-increasing improvement and applies, in sequence, all moves that result to be still feasible and improving after the modifications made by the application of the previous moves. This process defines a *batch* of moves for each iteration.

The use of a batch of moves of size greater than one produces a solution that is outside each single neighbourhood that has been scanned. In practice, it consists an heuristic selection of an improving solution inside an extremely large neighbourhood which may have non-polynomial size. This method resembles the idea of compounding improving moves by Ergun, Orlin, and Steele-Feldman (2006). However, the focus of that paper is on the techniques used to select a-priori *independent* moves. Instead, we completely disregard the independence of the moves, and we base the batching mechanism on: (i) effectiveness of a move (value of the fitness function computed during the scan of each neighbourhood) and (ii) feasibility of a move after the application of other moves. In particular, (ii) allows to use combinations of *dependent* moves that produce feasible solutions.

Algorithm 2. Batching-Move Local Search

```
1:  function BM- LS($x$)
2:      repeat
3:          $improve \leftarrow$ FALSE
4:          $MoveSet \leftarrow \emptyset$
5:          for $i \leftarrow 1$ to $MaxN$ do                                  ▷ MaxN = number of neighborhoods
6:              $MoveSet \leftarrow MoveSet \cup$ SELECTIMPROVING($N_i$)      ▷ Adds improving moves of neighborhood $N_i$
7:          sort $MoveSet$ by non-increasing improvement
8:          for $m \in MoveSet$ do            ▷ applies a batch of improving moves from several neighborhoods
9:              if $m$ is feasible and improving then
10:                 $x \leftarrow$ APPLYMOVE($x, m$); $improve \leftarrow$ TRUE
11:     until $improve$ = FALSE
12:     return $x$
```

The choice of the elementary neighbourhoods to be used inside the BM-LS is an important design step that we will discuss in Section 5.4. Each of these neighbourhoods consists of some movements of items from one bin to another. The size of each neighbourhood is polynomial in the input length.

Another important element to guide the search is the use of a fitness function instead of the objective function of the problem. This is quite usual in bin packing, indeed, when we move items between bins only rarely we are able to reduce the number of used bins. Hence, the most of the neighbouring solutions share the same objective function value. To evaluate this solutions it is necessary a fitness function that catches the 'shape' of the solution.

Let $\overline{M}$ be the set of bins used by the current solution and let L_i refer to the load of a bin i. The fitness function is:

$$\mathcal{F} = \sum\nolimits_{i \in \overline{M}} L_i^2. \tag{12}$$

Observe that the fitness gives a higher score to the solutions with unbalanced loads. For example, two bins each having load 5 give a score of 50, while the same bins with items reallocated to give loads 8 and 2 have score 68. The use of this fitness function induces to select solutions with some bin with small load, thus increasing the possibilities to empty this bin in a next iteration. In the few cases where the neighbouring solution has less bins than the starting one, we set the fitness to a large number.

5.3 *Perturbation*

The use of an extremely large neighbourhood, as that of our BM-ILS, allows to explore a wide area of the solutions space, but it does not avoid to be trapped by local optima. A diversification mechanism is, thus, fundamental to find high quality solutions. We implemented a Destroy-Repair diversification paradigm, introduced by Schrimpf et al. (2000), as outlined in Algorithm 3.

Algorithm 3. Destroy, Shrink and Repair a solution

```
1:  function DESTROYSHRINKREPAIR($x, \rho$)
2:      $change \leftarrow$ FALSE; $\hat{\rho} \leftarrow \rho$
3:      while $change$ = FALSE do
4:          $x_{new} \leftarrow$ DESTROYSHRINK($x, \hat{\rho}$)
5:          $x_{new} \leftarrow$ REPAIR($x_{new}$)
6:          if $x_{new}$ is feasible then
7:              $x \leftarrow x_{new}$; $change \leftarrow$ TRUE
8:          else
9:              $\hat{\rho} \leftarrow \hat{\rho} - 1$;
10:             if $\hat{\rho} = 1$ then $\hat{\rho} \leftarrow \rho$
11:     return ($x$)
```

Procedure DESTROYSHRINK is invoked on the current solution with a positive parameter ρ. The procedure randomly removes $\hat{\rho}$ items, with $\hat{\rho} = \rho$ in the first iteration. Then the partial solution is possibly shrunk if an empty bin exists and the precedence constraints allow to eliminate it by shifting the next bins by one position to the left. The next procedure REPAIR reinserts one item at a time in the first feasible bin of the partial solution, using the same order in which they were removed. If such bin does not exist, the repair mechanism tries to open a new bin in an appropriate position. Note that there are some cases in which no feasible insertion is possible. Consider, e.g. an instance with $C = 100$, $w_h = w_j = w_k = 40$, precedences $t_{hj} = t_{jk} = 0$ and a feasible solution with items h, j packed in bin i and item k packed in bin $i + 1$. The random removal eliminates k and j, in this order, then k is reinserted in bin i and we are stuck because j cannot be packed in bin i, due to the capacity constraints, and cannot be packed nor before i or after i, due to the precedence constraints. When a partial solution cannot be reconstructed, the procedure repeats the destroy-repair steps with parameter $\hat{\rho}$ reduced by one. When $\hat{\rho} = 1$ and no new feasible solution has been obtained, the procedure restarts with the initial ρ value.

5.4 *Elementary neighbourhoods*

The BM-LS asks to select a number of elementary neighbourhoods used to construct the batch of moves. We decided to start with some commonly adopted neighbourhoods, given by the application of the following moves:

- *Relocate*: move one item from its current bin to a different one;
- *Swap(1,1)*: exchange two items packed in two different bins;
- *Swap(2,1)*: exchange two items packed in the same bin with one item packed in a different bin;
- *Push*: move an item j from bin h to a bin i and an item k packed in i to a bin different from h. It can be viewed as a two-step *relocate* operator, and is especially useful when the simple relocation of j into bin i violates the bin capacity.

To define the final choice of the neighbourhoods we performed some preliminary experiments on a subset of instances. On the basis of the results obtained with these experiments, we decided to remove Swap(1,1) because its use worsened the performance of the BM-ILS. All other neighbourhoods had a positive impact, and hence the final configuration of our BM-LS invokes Relocate, Swap(2,1) and Push, in that order.

6. Computational experiments

To evaluate the performance of the proposed BM-ILS algorithm, computational experiments were performed on the BPP-GP and on two special cases, namely, the SALBP-I and the BPP-P. The SALBP-I is a well-known problem with several benchmark instances available in the literature. To the best of our knowledge there is no previous work on the BPP-GP, and thus, we extended the most recent and comprehensive benchmark set developed for the SALBP-I to the BPP-GP by adding random weights on the precedence arcs. We also considered a new set of instances, in such a way that a total of 14700 instances, ranging from 20 to 1000 items, were solved. Instances and detailed computational results are available at http://www.or.unimore.it/resources/BPP_GP/home.html.

Our algorithms were coded in C++ and executed on an Intel Xeon E5530 2.4 GHz with 24 GB of memory, running under Linux Ubuntu 14.04 LTS 64-bit. CPLEX 12.4 was adopted to solve the ILP models (1)–(7), (8) and (11), by imposing it to use a single thread. The CPLEX time limit was set to 300 s for model (1)–(7), and to just 10 s for models (8) (which corresponds to the SPPC) and (11) (which corresponds to the SSP-PC). Unless stated otherwise, a time limit of 300 s was imposed to BM-ILS. To facilitate future computational comparison, we evaluated the CPU performance of our computer using the indicators given by PassMark© Software (see https://www.cpubenchmark.net/), and obtained a CPUmark value equal to 1086 for a single thread. The greater the CPUmark value, the faster the computer, so that a computer with twice the value of another can process about twice as much data within the same time interval.

To limit the computational effort, the second preprocessing procedure of Section 4.1 and the lower bound LB_5 of Section 4.2 were run only on instances with $n \leq 100$. When computing LB_5, we adopted $LB_\alpha = L_{dff}$, where L_{dff} is a quick lower bound based on the use of dual feasible functions, proposed in Dell'Amico, Díaz-Díaz, and Iori (2012). In this way, the entire set of preprocessing and lower bounding techniques took about one second per instance on average, and less than eight seconds in the worst case. BM-ILS and the ILP models were applied after the preprocessing and lower bounding procedures.

6.1 *Benchmark instances*

The most recent benchmark set for the SALBP-I has been proposed by Otto, Otto, and Scholl (2013). The set was developed to represent the properties of real-world assembly lines, and consists of four different groups, each associated with a number of items $n \in \{20, 50, 100, 1000\}$ and containing 525 instances. For each group, the instances can be classified according to three properties: (a) the structure of the precedence graph; (b) the order strength; and (c) the distribution of item weights. A quick description of their classification follows:

- Graph structure: *bottleneck* (BN) if the instance contains items with a high number of direct predecessors and successors; *chain* (CH) if it contains a path of items having only one successor and one predecessor; and *mixed* (MIX) if it is a combination of the two previous structures.
- Order strength: the *order strength* (OS) represents how much the precedence graph forbids sequences of tasks, and is computed as the number of arcs in the graph transitive closure divided by $n(n-1)/2$. Three values of OS were tested, namely, 0.2, 0.6 and 0.9.
- Distribution of item weights: items weights were generated according to a normal distribution with mean value in $0.1C$ (*bottom*), in $0.5C$ (*middle*), or by a combination of the two (*bimodal*).

Using the classification above, Otto, Otto, and Scholl (2013) produced 21 different classes of instances, and created 25 random instances for each class and each value of n, producing in total 2100 instances. Later, Pereira (2016) extended this test bed to the BPP-P by imposing unitary weights on the arcs. In this work we further extend it to the BPP-GP, by varying the precedence weights on the arcs according to two cases:

- BPP-GP(0,1): t_{jk} values are randomly chosen with uniform distribution in the set $\{0, 1\}$;
- BPP-GP(0,3): t_{jk} values are randomly chosen with uniform distribution in the set $\{0, 1, 2, 3\}$.

In addition, we also considered new sets of instances for SALBP-I, BPP-P, BPP-GP(0,1) and BPP-GP(0,3), having $n \in \{250, 500, 750\}$. For each combination of *problem variant* and *size*, 525 instances were generated, for a total of 6300 new instances. Instances for SALBP-I were kindly provided by Alena Otto. These instances were generated by means of the *SALBPGen* by Otto, Otto, and Scholl (2013), considering the same parameterisation adopted in their paper. Then, we extended them to address the BPP-P, the BPP-GP(0,1) and the BPP-GP(0,3) by adding precedence weights as discussed in Section 6.1. These intermediate instances were generated to provide means to better evaluate the BM-ILS in our computational experiments. Moreover, they may be useful for future researches, especially for those dealing with exact algorithms.

6.2 *Comparing BM-ILS with single-move ILS algorithms*

To investigate the advantages and disadvantages of using batching moves instead of single moves, we performed some computational experiments on a subset of instances. This subset was defined by selecting the first and last instance for each class of each problem variant, with $n = 100$ and $n = 1000$. In particular, we compare BM-ILS with three single-move ILS algorithms:

- Best Improvement ILS: at each iteration, the best feasible and improving move, if any, from Relocate, Swap(2,1) and Push is performed;
- First Improvement ILS: at each iteration, the first feasible and improving move, if any, from Relocate, Swap(2,1) and Push (invoked in this sequence) is performed; note that this method is also known in the literature as *Variable Neighborhood Descent* (see, e.g. Mladenović and Hansen 1997; Hansen and Mladenović 2001);
- Random ILS: at each iteration, a random neighbourhood from Relocate, Swap(2,1) and Push is chosen, and the best move, if any, is performed; this method is also known in the literature as *Random Variable Neighborhood Descent* (see, e.g. Subramanian et al. 2010; Kramer et al. 2015).

The implementation of these three algorithms differs from BM-ILS only in the Local Search procedure. We set a time limit of 300 s for each run. The algorithm stops either if the time limit expires or the incumbent solution is equal to the best lower bound obtained using the procedures from Section 4.2. The results for this experiment are reported in Table 1. Each row reports the aggregate results for 42 instances of each size and problem variant, and columns #opt, gap (%) and time (s) report the number of instances solved to optimality, the average gap to the best known lower bound and the average computational time, respectively. Table 1 shows that BM-ILS has a consistently better performance with respect to single-move algorithms, in terms of number of problems solved to optimality, percentage gap and execution time. Among the single-move algorithms, *First Improvement ILS* and *Random ILS* are usually better than *Best Improvement ILS* because they do not explore the whole set of neighbourhoods at each iteration, consequently leading to more ILS iterations per unit of time. In the *Best Improvement ILS*, instead, the overhead of exploring the whole set of neighbourhoods slows down the convergence. The BM-ILS, however, behaves in-between the *Best Improvement ILS* and the *First Improvement ILS*, in such a way that the overhead of evaluating the whole set of neighbourhoods is amortised by the collection of all the improving moves in $MoveSet$, usually leading to better quality solutions.

Table 1. Comparing BM-ILS with single-move ILS algorithms.

			BM-ILS[a]			Best Improvement ILS[a]			First Improvement ILS[a]			Random ILS[a]		
n	problem	#inst	#opt	gap (%)	time (s)	#opt	gap (%)	time (s)	#opt	gap (%)	time (s)	#opt	gap (%)	time (s)
100	SALBP-I	42	39	0.24	100.23	36	0.41	100.99	39	0.29	100.53	39	0.24	98.53
	BPP-P	42	39	0.22	120.20	38	0.37	123.30	37	0.37	124.15	39	0.22	120.29
	BPPGP-01	42	36	0.42	101.36	35	0.47	101.59	35	0.47	101.30	35	0.47	100.22
	BPPGP-03	42	34	0.51	123.43	32	0.55	129.32	34	0.47	125.30	33	0.55	129.02
1000	SALBP-I	42	21	0.47	161.46	19	0.52	166.39	19	0.52	171.84	19	0.48	169.33
	BPP-P	42	13	0.92	220.03	11	1.13	235.30	13	0.96	214.59	13	0.96	220.66
	BPPGP-01	42	18	0.66	182.88	17	0.75	191.48	17	0.72	197.82	17	0.67	188.29
	BPPGP-03	42	4	1.92	274.86	3	2.37	287.76	4	2.11	274.70	4	2.29	275.67
		336	204	0.67	160.56	191	0.82	167.02	198	0.74	163.78	199	0.74	162.75

[a]Intel Xeon E5530 2.4 GHz with 24 GB of RAM, using a single thread. CPUmark value for single thread use: 1086.

6.3 *Comparison with state-of-the-art methods for SALBP-I and BPP-P*

Before discussing the results that we obtained on the BPP-GP instances, we first evaluate the performance of our BM-ILS algorithm on the SALBP-I and the BPP-P.

6.3.1 *Comparison on SALBP-I*

In this section we compare BM-ILS with the state-of-the-art heuristic algorithms that have been proposed in the literature to solve the SALBP-I, namely

- Beam-ACO: hybrid metaheuristic of Blum (2008);
- SALOME: bidirectional branch-and-bound algorithm of Scholl and Klein (1997, 1999);
- Tabu-Simple: tabu search algorithm by Scholl and Voß (1996), refined with some modifications by Pape (2015);
- t-bounded-DP: dynamic programming based heuristic by Bautista and Pereira (2009), also refined with some modifications by Pape (2015);
- BBR: branch, bound and remember by Morrison, Sewell, and Jacobson (2014).

These algorithms have been tested on the SALBP-I instances with $n = 100$ and $n = 1000$, and their performances are summarised in Tables 2 and 3, respectively. Each line reports results for a set of 25 instances sharing the same properties. The first four columns report the instance characteristics, namely, instance group, graph structure, order strength and weights distribution. Let BLB denote the best known lower bound, computed as the maximum among the values given in Morrison, Sewell, and Jacobson (2014), Pape (2015) and Pereira (2015), and BUB denote the best known upper bound, computed as the minimum among the values obtained by BM-ILS and by the above mentioned heuristics. The next columns in the table report the average percentage gap of a given algorithm with respect to the BLB, computed as 100(UB-BLB)/UB, with UB being the solution value produced by the algorithm.

Note that for some algorithms we provide more than one column, reporting the gap obtained with different time limits. The line right below the name of each algorithm shows indeed the time limit imposed. The results of the aforementioned methods, except the ones for BBR and BM-ILS, were collected from Pape (2015), who used a single thread of a Intel Core i7-640M with 2.8 GHz processor and 4 GB RAM (CPUmark value equal to 1311, for a single thread). BBR was run, instead, on a single core of an Intel Core i7-930 2.8 GHz quad-core processor, with 12 GB of RAM (CPUmark value equal to 1214, for a single thread). Since the CPUmark of the computer we used for our experiments is smaller than that of the competitors, using the same time limit we guarantee a fair comparison. The last line in each table reports the average results over the whole set of instances.

Table 2 shows that BBR is the algorithm that provides the best overall results on the 100-items instances, although at the expenses of one CPU hour. BM-ILS is able to provide good solutions within a short computational time. In particular it solves 497 instances to proven optimality within 180 s and two more instances within 300 s. For the non-optimal cases, it produces solutions that differ only one bin from the BLB, leading to gaps equivalent to those from state-of-the-art heuristic methods. The BM-ILS performance is even better when solving the very large instances with 1000 items, as shown in Table

Table 2. Gap (%) from the best known lower bound for SALBP-I instances with 100 items.

				Beam-ACO[a]	SALOME[a]	Tabu-Simple[a]		t-bounded-DP[a]		BBR[b]	BM-ILS[c]		
inst.	struct.	OS	distr.	180 s	180 s	180 s	3600 s	180 s	3600 s	3600 s	180 s	300 s	3600 s
1–25	BN	0.2	Bimodal	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
26–50	BN	0.2	Bottom	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
51–75	BN	0.2	Middle	1.10	1.71	0.31	0.23	0.89	0.75	0.23	0.23	0.23	0.23
76–100	BN	0.6	Bimodal	0.00	0.00	0.16	0.00	0.00	0.00	0.00	0.00	0.00	0.00
101–125	BN	0.6	Bottom	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
126–150	BN	0.6	Middle	1.23	1.41	0.23	0.15	0.15	0.15	0.15	0.15	0.15	0.15
151–175	CH	0.2	Bimodal	0.16	0.00	0.34	0.34	0.00	0.00	0.00	0.49	0.49	0.34
176–200	CH	0.2	Bottom	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
201–225	CH	0.2	Middle	0.97	2.17	0.38	0.30	0.96	0.60	0.30	0.46	0.46	0.30
226–250	CH	0.6	Bimodal	0.34	0.00	0.50	0.34	0.00	0.00	0.00	0.60	0.51	0.34
251–275	CH	0.6	Bottom	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
276–300	CH	0.6	Middle	1.45	0.85	0.27	0.13	0.07	0.00	0.00	0.34	0.27	0.13
301–325	MIX	0.2	Bimodal	0.00	0.00	0.35	0.18	0.00	0.00	0.00	0.35	0.35	0.18
326–350	MIX	0.2	Bottom	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
351–375	MIX	0.2	Middle	0.65	1.50	0.08	0.08	0.54	0.31	0.08	0.08	0.08	0.08
376–400	MIX	0.6	Bimodal	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
401–425	MIX	0.6	Bottom	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
426–450	MIX	0.6	Middle	1.43	0.80	0.14	0.14	0.07	0.07	0.00	0.14	0.14	0.07
451–475	MIX	0.9	Bimodal	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
476–500	MIX	0.9	Bottom	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
501–525	MIX	0.9	Middle	1.06	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
avg				0.40	0.40	0.13	0.09	0.13	0.09	0.04	0.13	0.13	0.09

[a]Intel Core i7-640M 2.8 GHz with 4 GB of RAM, using a single thread. CPUmark value for single thread use: 1311.
[b]Intel Core i7-930 2.8 GHz with 12 GB of RAM, using a single thread. CPUmark value for single thread use: 1214.
[c]Intel Xeon E5530 2.4 GHz with 24 GB of RAM, using a single thread. CPUmark value for single thread use: 1086.

3. Its overall average gap (0.45%) with 180 s of time limit is lower than those obtained by the state-of-the-art methods. On the instances with 100 items, giving BM-ILS one hour of time allows it to find eight more proven optimal solutions. On the larger instances with 1000 items, the one hour limit allows BM-ILS to reach a very low average gap. We also notice that the best improvements with respect to the existing methods are obtained on instances with middle task times.

6.3.2 *Comparison on BPP-P*

To the best of our knowledge, the most efficient heuristic approach for the BPP-P has been developed by Pereira (2016). He proposed a dynamic programming and an enumeration algorithm, and tested them on the benchmark set proposed by Dell'Amico, Díaz-Díaz, and Iori (2012), solving all instances to proven optimality. He also addressed the instances by Otto, Otto, and Scholl (2013) having $n = 100$, modified by imposing a precedence weight equal to one on each arc. The results that he obtained are summarised in Table 4, and refer to a new run (J. Pereira, private communication, 2016) where some implementation issues were fixed. The table also shows the results that we obtained with our BM-ILS. In particular, for each algorithm it reports the average percentage gap (gap (%)) from the BLB (also taken from J. Pereira, private communication, 2016), the average time to run to completion (time (s)) and the total number of instances unsolved to proven optimally (#uns), i.e. instances where UB>BLB. For BM-ILS it also reports the average time in which the incumbent solution was found (time_{inc}(s)). The dynamic programming method always ran to completion with a limited computational effort, whereas the enumeration algorithm was allowed to run for at most one hour and the BM-ILS for at most 300 s.

On average, BM-ILS has a better performance than the dynamic programming heuristic, showing lower gaps and number of instances unsolved to proven optimality. Not surprisingly, the enumeration algorithm has the best overall results, but still BM-ILS is capable of finding some improvements for instances having middle distribution (details on these instances are available on the cited web page). The maximum deviation between the UB value produced by BM-ILS and the BLB is one

Table 3. Gap (%) from the best known lower bound for SALBP-I instances with 1000 items.

				Beam-ACO[a]		SALOME[a]		Tabu-Simple[a]			t-bounded-DP[a]			BBR[b]	BM-ILS[c]			
inst.	struct.	OS	distr.	180 s	900 s	180 s	900 s	180 s	900 s	3600 s	180 s	900 s	3600 s	3600 s	180 s	300 s	900 s	3600 s
1–25	BN	0.2	Bottom	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
26–50	BN	0.2	Middle	3.05	2.63	1.27	1.25	0.41	0.25	0.20	3.94	3.67	3.39	2.54	0.42	0.34	0.26	0.21
51–75	BN	0.2	Bimodal	0.00	0.00	0.00	0.00	0.07	0.05	0.05	0.00	0.00	0.00	0.00	0.09	0.07	0.07	0.05
76–100	BN	0.6	Bottom	0.00	0.00	0.00	0.00	0.12	0.06	0.03	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
101–125	BN	0.6	Middle	4.00	3.74	1.87	1.87	0.72	0.40	0.31	4.47	3.97	3.51	2.87	0.56	0.48	0.40	0.36
126–150	BN	0.6	Bimodal	0.00	0.00	0.00	0.00	0.36	0.18	0.18	0.00	0.00	0.00	0.00	0.22	0.22	0.22	0.20
151–175	CH	0.2	Bottom	0.00	0.00	0.00	0.00	0.03	0.03	0.03	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
176–200	CH	0.2	Middle	4.20	4.03	2.22	2.21	0.83	0.55	0.49	4.29	4.10	3.57	3.16	0.99	0.86	0.67	0.47
201–225	CH	0.2	Bimodal	0.00	0.00	0.03	0.02	0.16	0.11	0.11	0.00	0.00	0.00	0.00	0.19	0.18	0.16	0.14
226–250	CH	0.6	Bottom	0.00	0.00	0.00	0.00	0.29	0.15	0.03	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
251–275	CH	0.6	Middle	5.20	5.05	4.01	4.00	1.91	1.52	1.34	4.29	3.75	3.27	2.91	1.82	1.67	1.44	1.23
276–300	CH	0.6	Bimodal	0.02	0.02	0.04	0.00	0.58	0.29	0.29	0.00	0.00	0.00	0.00	0.32	0.32	0.29	0.27
301–325	MIX	0.2	Bottom	0.00	0.00	0.00	0.00	0.03	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
326–350	MIX	0.2	Middle	3.34	3.01	1.49	1.49	0.50	0.35	0.31	3.88	3.66	3.23	2.73	0.51	0.44	0.34	0.31
351–375	MIX	0.2	Bimodal	0.00	0.00	0.02	0.02	0.18	0.07	0.05	0.00	0.00	0.00	0.00	0.16	0.14	0.13	0.09
376–400	MIX	0.6	Bottom	0.00	0.00	0.00	0.00	0.17	0.09	0.06	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
401–425	MIX	0.6	Middle	4.77	4.56	2.87	2.86	1.25	0.93	0.83	4.27	3.92	3.24	2.81	1.14	1.07	0.87	0.71
426–450	MIX	0.6	Bimodal	0.02	0.02	0.00	0.00	0.44	0.21	0.11	0.02	0.00	0.00	0.00	0.09	0.09	0.09	0.09
451–475	MIX	0.9	Bottom	0.06	0.03	0.03	0.03	0.81	0.55	0.41	0.00	0.00	0.00	0.00	0.15	0.15	0.15	0.15
476–500	MIX	0.9	Middle	6.61	6.36	5.25	5.23	8.85	8.53	5.72	2.90	2.52	1.99	3.39	2.02	1.89	1.71	1.55
501–525	MIX	0.9	Bimodal	0.23	0.23	1.06	0.87	1.30	0.75	0.64	0.00	0.00	0.00	0.00	0.71	0.68	0.64	0.50
avg				1.50	1.41	0.96	0.94	0.91	0.72	0.53	1.34	1.22	1.06	0.97	0.45	0.41	0.35	0.30

[a]Intel Core i7-640M 2.8 GHz with 4 GB of RAM, using a single thread. CPUmark value for single thread use: 1311.
[b]Intel Core i7-930 2.8 GHz with 12 GB of RAM, using a single thread. CPUmark value for single thread use: 1214.
[c]Intel Xeon E5530 2.4 GHz with 24 GB of RAM, using a single thread. CPUmark value for single thread use: 1086.

bin. This results was also noticed for the SALBP-I instances in the previous section, and indeed the performance of BM-ILS on the BPP-P is quite similar to that obtained for the SALBP-I. This is also confirmed from the results on the larger BPP-P instances, not addressed by J. Pereira (private communication, 2016), Pereira (2016), that are reported in the next section. It is worth mentioning that a slightly improvement on these results can be obtained by executing BM-ILS with one hour of time limit. With this configuration, 10 more instances were solved to proven optimality and the average percentage gap was reduced to 0.18%

6.4 *Results on the entire BPP-GP set of instances*

In this section, we present the computational results obtained with BM-ILS on the entire set of 8400 instances. To gain insight in the computational performance of our algorithm, we compare it with the best results obtained by a simple algorithm, called CPLEX/FF, that attempts to solve the problem by first invoking the first fit heuristic of Section 5.1 and then solving the mathematical model (1)–(7) through CPLEX. Both algorithms are stopped whenever the incumbent solution value is equal to the lower bound or when 300 s have been elapsed.

Table 5 summarises the results that we obtained on the four problem variants. Each row presents average/total results for 525 instances having the same number of items. The values in the columns have the following meanings: gap$_1$(%) is the average percentage gap from the lower bound computed as the best lower bound produced by CPLEX and by our algorithms of Section 4.2, while gap$_2$(%) is the average percentage gap from BLB (also considering the lower bounds from the literature); time (s) is the average time to run to completion; time$_{inc}$ (s) is the average time to find the incumbent solution; and #uns is the total number of instances unsolved to proven optimality. The combined use of the first fit heuristic with CPLEX is motivated by the fact that CPLEX alone failed in providing a feasible solution for a large part of the instances.

Table 4. Computational results and comparison on BPP-P instances with 100 items.

				dyn. programming[a]			enum. algorithm[a] (1h)			BM-ILS[b] (300 s)			
inst.	struct.	OS	distr.	gap (%)	time (s)	#uns	gap (%)	time (s)	#uns	gap (%)	time (S)	time_{inc}(s)	#uns
1–25	BN	0.2	Bimodal	0.00	5.10	0	0.00	0.00	0	0.18	21.66	9.66	1
26–50	BN	0.2	Bottom	0.00	3.88	0	0.00	0.00	0	0.80	48.10	0.10	3
51–75	BN	0.2	Middle	1.25	16.52	11	0.38	1691.31	5	0.23	88.19	5.34	3
76–100	BN	0.6	Bimodal	0.18	4.78	1	0.00	27.54	0	0.17	59.01	11.01	1
101–125	BN	0.6	Bottom	0.00	1.73	0	0.00	0.00	0	0.00	12.09	0.09	0
126–150	BN	0.6	Middle	0.66	5.91	8	0.00	98.77	0	0.00	240.32	9.53	0
151–175	CH	0.2	Bimodal	0.00	3.65	0	0.00	0.00	0	0.32	73.53	1.53	2
176–200	CH	0.2	Bottom	0.00	2.80	0	0.00	0.00	0	0.00	12.92	0.92	0
201–225	CH	0.2	Middle	1.57	13.90	19	0.54	2368.91	7	0.54	137.08	5.22	7
226–250	CH	0.6	Bimodal	0.00	2.20	0	0.00	0.09	0	0.32	247.40	15.31	2
251–275	CH	0.6	Bottom	0.00	1.42	0	0.00	0.00	0	0.00	24.10	0.10	0
276–300	CH	0.6	Middle	0.42	3.42	6	0.00	47.32	0	0.35	288.76	28.95	5
301–325	MIX	0.2	Bimodal	0.00	5.69	0	0.00	0.00	0	0.51	36.60	0.60	3
326–350	MIX	0.2	Bottom	0.00	3.57	0	0.00	0.00	0	1.07	70.04	22.04	4
351–375	MIX	0.2	Middle	1.47	22.27	16	0.44	2071.00	5	0.30	122.79	15.01	4
376–400	MIX	0.6	Bimodal	0.00	2.20	0	0.00	0.00	0	0.17	149.77	5.95	1
401–425	MIX	0.6	Bottom	0.00	1.15	0	0.00	0.00	0	0.00	36.06	0.06	0
426–450	MIX	0.6	Middle	0.29	3.92	4	0.00	61.39	0	0.30	300.08	20.64	4
451–475	MIX	0.9	Bimodal	0.00	1.66	0	0.00	0.00	0	0.00	300.03	0.03	0
476–500	MIX	0.9	Bottom	0.00	1.32	0	0.00	0.00	0	0.00	0.03	0.03	0
501–525	MIX	0.9	Middle	0.00	1.31	0	0.00	0.00	0	0.00	300.08	0.59	0
				0.28	5.16	65	0.06	303.16	17	0.25	122.32	7.27	40

[a]Intel Core i7-930 2.8 GHz with 12 GB of RAM, using a single thread. CPUmark value for a single thread use: 1214.
[b]Intel Xeon E5530 2.4 GHz with 24 GB of RAM, using a single thread. CPUmark value for a single thread use: 1086.

For all the 20-item groups, both CPLEX/FF and BM-ILS solve all instances to proven optimality. CPLEX/FF requires less than one second. BM-ILS also requires less than one second to find the incumbent solution, although in some cases it continues running because the lower bound value it uses for termination (computed using the algorithms in Section 4.2) is not optimal. As the number of items increases, the percentage of instances solved to proven optimality decreases and the CPU time increases for both algorithms. However, the results obtained by BM-ILS always dominate those by CPLEX/FF in terms of gap (%) and #uns.

Disregarding the instances with $n = \{250, 500, 750\}$ (which have no lower bounds in the current literature), the average gap_2 by BM-ILS is always below 1%, with a single exception for BPP-GP(0,3) and $n = 1000$ where it is slightly larger than 2%. Overall, the presence of precedences with larger weights makes the search process in BM-ILS more complex. This can be noticed by the fact that BPP-P is slightly more difficult than SALBP-I (usually showing greater values for #uns, gap (%), time (s), and time_{inc} (s)) and BPP-GP(0,3) is more difficult than BPP-GP(0,1). This can be imputed to the fact that large precedence weights imply a greater dependence among the items and their allocations, so making more difficult to find feasible moves as well as improving moves. A similar behaviour cannot be noticed for CPLEX/FF: it achieves the largest average gap (%) (slightly larger than 7%) for BPP-GP(0,3) and $n = 1000$, but the largest #uns values for BPP-GP(0,1) and SALBP-I.

By considering the new instances with $n = \{250, 500, 750\}$, we notice a peak on the $\text{gap}_2(\%)$ values for the 250- and 500-items instances, on each problem variant. The most difficult instances are, however, those with the largest n values, as shown in columns #uns.

The fact that CPLEX alone fails in finding a heuristic solution in 300 s for a very large number of instances is a further motivation for producing heuristics for the BPP-GP. Looking at the entire set of computational experiments, we can say that BM-ILS is a very robust algorithm with enough flexibility for dealing with the different variants of the problem. The computational results are more than satisfactory. We finally notice that the number of ‘open’ instances, i.e. instances for which BUB > BLB, is still very large: (i) for the instances with $n = \{20, 50, 100, 1000\}$, there are 161 unsolved cases for SALBP-I,

Table 5. Computational results on all problem variants.

		CPLEX/FF[a] (300s)				BM-ILS[a] (300s)				
problem variant	n	gap_1(%)	gap_2(%)	time (s)	#uns	gap_1(%)	gap_2(%)	time (s)	time_{inc}(s)	#uns
	20	0.00	0.00	0.09	0	0.00	0.00	26.88	0.02	0
	50	1.15	0.89	53.85	43	0.29	0.03	58.64	0.37	2
	100	4.32	3.85	133.36	219	0.63	0.13	81.35	4.32	26
SALBP-I	250	7.44	7.44	266.00	454	2.13	2.13	130.75	16.78	221
	500	6.94	6.94	300.26	524	1.62	1.62	139.95	34.78	238
	750	6.55	6.55	293.37	525	1.42	1.42	141.08	46.65	240
	1000	6.26	5.45	301.27	525	1.32	0.41	154.92	63.11	256
	20	0.00	0.00	0.05	0	0.00	0.00	55.44	0.01	0
	50	1.38	1.22	55.10	68	0.36	0.18	118.59	0.91	26
	100	4.89	4.11	125.17	200	1.11	0.25	122.32	7.27	40
BPP-P	250	7.92	7.92	219.69	371	2.55	2.55	171.99	23.05	264
	500	8.69	8.69	283.00	493	1.96	1.96	187.15	46.01	283
	750	7.98	7.98	283.31	501	1.79	1.79	209.18	84.45	327
	1000	7.60	6.80	300.31	519	1.70	0.80	215.88	95.44	350
	20	0.00	0.00	0.11	0	0.00	0.00	46.30	0.01	0
	50	1.29	1.06	56.42	61	0.34	0.11	76.89	0.33	15
	100	4.48	4.08	128.44	215	0.80	0.36	99.09	4.67	69
BPP-GP(0,1)	250	7.75	7.75	244.88	416	2.23	2.23	141.79	17.61	233
	500	7.52	7.52	291.33	506	1.77	1.77	159.91	43.62	257
	750	7.32	7.32	284.29	525	1.65	1.65	177.97	64.45	292
	1000	6.92	6.11	301.30	525	1.54	0.63	193.13	83.90	310
	20	0.00	0.00	0.01	0	0.00	0.00	56.01	0.01	0
	50	0.75	0.70	30.02	36	0.16	0.10	80.49	0.56	15
	100	4.07	3.92	93.13	161	0.59	0.41	98.43	4.19	86
BPP-GP(0,3)	250	6.65	6.65	160.85	275	2.04	2.04	157.55	37.61	260
	500	8.52	8.52	227.23	394	2.50	2.50	216.19	84.15	346
	750	7.94	7.94	230.48	438	2.71	2.71	264.45	137.14	437
	1000	8.03	7.25	282.18	484	2.93	2.04	271.42	146.65	453

[a]Intel Xeon E5530 2.4 GHz with 24 GB of RAM, using a single thread. CPUmark value for a single thread use: 1086.

382 for BPP-P, 392 for BPP-GP(0,1) and 535 for BPP-GP(0,3); while (ii) for the instances with $n = \{250, 500, 750\}$, there are 699 for SALBP-I, 837 for BPP-P, 773 for BPP-GP(0,1) and 931 for BPP-GP(0,3). Future research is thus envisaged, both in terms of exact and heuristic techniques.

7. Conclusions

This paper dealt with the bin packing problem with generalised precedence constraints, a generalisation of the well-known simple assembly line balancing problem and of the bin packing problem with precedence constraints, in which the precedences between items can assume heterogeneous non-negative values. To solve the problem we presented an iterated local search algorithm with batching moves, as well as some preprocessing and lower bounding procedures. The proposed approach, despite its simplicity and flexibility of use, was able to provide good quality solutions within a reasonable computational time, and compared well with state-of-the-art methods tailored for solving particular problem cases. Computational experiments showed that considering heterogeneous precedences raises new challenges, both for exact and heuristic methods.

Acknowledgements

We thank Alena Otto for kindly providing us with new instances for the SALBP-I, Tom Pape for giving additional details about his work, Jordi Pereira for updating us with new results for the BPP-P, and two anonymous referees for comments that improved the quality of the paper.

Disclosure statement

No potential conflict of interest was reported by the authors.

Funding

This research was partially supported by Conselho Nacional de Desenvolvimento Científico e Tecnológico (CNPq/Brazil) [grant number GDE 201222/2014-0]; Coordenação de Aperfeiçoamento de Pessoal de Nível Superior (CAPES/Brazil) [grant PVE number A007/2013].

References

Augustine, J., S. Banerjee, and S. Irani. 2009. "Strip Packing with Precedence Constraints and Strip Packing with Release Times." *Theoretical Computer Science* 410: 3792–3802.

Bautista, J., and J. Pereira. 2009. "A Dynamic Programming Based Heuristic for the Assembly Line Balancing Problem." *European Journal of Operational Research* 194 (3): 787–794.

Becker, C., and A. Scholl. 2006. "A Survey on Problems and Methods in Generalized Assembly Line Balancing." *European Journal of Operational Research* 168: 694–715.

Bianco, L., and M. Caramia. 2012. "An Exact Algorithm to Minimize the Makespan in Project Scheduling with Scarce Resources and Generalized Precedence Relations." *European Journal of Operational Research* 219 (1): 73–85.

Blum, C. 2008. "Beam-ACO for Simple Assembly Line Balancing." *INFORMS Journal on Computing* 20 (4): 618–627.

Brucker, P., A. Drexl, R. Möhring, K. Neumann, and E. Pesch. 1999. "Resource-constrained Project Scheduling: Notation, Classification, Models, and Methods." *European Journal of Operational Research* 112 (1): 3–41.

Chassiakos, A. P., and S. P. Sakellaropoulos. 2005. "Time-cost Optimization of Construction Projects with Generalized Activity Constraints." *Journal of Construction Engineering and Management* 131 (10): 1115–1124.

Coffman Jr., E. G., J. Csirik, G. Galambos, S. Martello, and D. Vigo. 2013. "Handbook of Combinatorial Optimization." Chap. in *Bin Packing Approximation Algorithms: Survey and Classification*, 455–531. New York: Springer.

Cortés, P., L. Onieva, and J. Guadix. 2010. "Optimising and Simulating the Assembly Line Balancing Problem in a Motorcycle Manufacturing Company: A Case Study." *International Journal of Production Research* 48 (12): 3637–3656.

De Reyck, B., and W. Herroelen. 1998. "A Branch-and-Bound Procedure for the Resource-constrained Project Scheduling Problem with Generalized Precedence Relations." *European Journal of Operational Research* 111 (1): 152–174.

Dell'Amico, M., J. C. Díaz-Díaz, and M. Iori. 2012. "The Bin Packing Problem with Precedence Constraints." *Operations Research* 60 (6): 1491–1504.

Delorme, M., M. Iori, and S. Martello. 2016. "Bin Packing and Cutting Stock Problems: Mathematical Models and Exact Algorithms." *European Journal of Operational Research* 255 (1): 1–20.

Ergun, Ö., J. B. Orlin, and A. Steele-Feldman. 2006. "Creating Very Large Scale Neighborhoods Out of Smaller Ones by Compounding Moves." *Journal of Heuristics* 12 (1): 115–140.

Garey, M. R., R. L. Graham, D. S. Johnson, and A. C. C. Yao. 1976. "Resource Constrained Scheduling as Generalized Bin Packing." *Journal of Combinatorial Theory, Series A* 21 (3): 257–298.

Hansen, P., and N. Mladenović. 2001. "Variable Neighborhood Search: Principles and Applications." *European Journal of Operational Research* 130 (3): 449–467.

Hartmann, S., and D. Briskorn. 2010. "A Survey of Variants and Extensions of the Resource-constrained Project Scheduling Problem." *European Journal of Operational Research* 207 (1): 1–14.

Klein, R. 2000. "Project Scheduling with Time-varying Resource Constraints." *International Journal of Production Research* 38 (16): 3937–3952.

Kramer, R., M. Dell'Amico, and M. Iori. 2016. "An Iterated Local Search Algorithm for the Bin Packing Problem with Generalized Precedence Constraints." In *4th International Symposium on Combinatorial Optimization, Book of Abstracts*, Vietri sul Mare, Italy.

Kramer, R., A. Subramanian, T. Vidal, and L. A. F. Cabral. 2015. "A Matheuristic Approach for the Pollution-routing Problem." *European Journal of Operational Research* 243 (2): 523–539.

Kucukkoc, Ibrahim, and David Z. Zhang. 2017. "Balancing of Mixed-model Parallel U-shaped Assembly Lines Considering Model Sequences." *International Journal of Production Research*: 1–18.

Liu, C.-M., and C.-H. Chen. 2002. "Multi-section Electronic Assembly Line Balancing Problems: A Case Study." *Production Planning & Control* 13 (5): 451–461.

Lourenço, H. R., O. C. Martin, and T. Stützle. 2002. "Handbook of Metaheuristics." Chap. in *Iterated Local Search*, 321–353. Norwell, MA: Kluwer Academic.

Mladenović, N., and P. Hansen. 1997. "Variable Neighborhood Search." *Computers & Operations Research* 24 (11): 1097–1100.

Morrison, D. R., E. C. Sewell, and S. H. Jacobson. 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.

Mouthuy, S. 2011. “Constraint-based Very Large-scale Neighborhood Search.” PhD diss., Université Catholique de Louvain, Louvain School of Engineering, Belgium.

Nicosia, G., and A. Pacifici. 2017. “Scheduling Assembly Tasks with Caterpillar Precedence Constraints on Dedicated Machines.” *International Journal of Production Research* 55 (6): 1680–1691.

Otto, C., and A. Otto. 2014. “Extending Assembly Line Balancing Problem by Incorporating Learning Effects.” *International Journal of Production Research* 52 (24): 7193–7208.

Otto, A., C. Otto, and A. Scholl. 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.

Pape, T. 2015. “Heuristics and Lower Bounds for the Simple Assembly Line Balancing Problem Type 1: Overview, Computational Tests and Improvements.” *European Journal of Operational Research* 240 (1): 32–42.

Pastor, R., and L. Ferrer. 2009. “An Improved Mathematical Program to Solve the Simple Assembly Line Balancing Problem.” *International Journal of Production Research* 47 (11): 2943–2959.

Pereira, J. 2015. “Empirical Evaluation of Lower Bounding Methods for the Simple Assembly Line Balancing Problem.” *International Journal of Production Research* 53 (11): 3327–3340.

Pereira, J. 2016. “Procedures for the Bin Packing Problem with Precedence Constraints.” *European Journal of Operational Research* 250 (3): 794–806.

Scholl, A., and C. Becker. 2006. “State-of-the-Art Exact and Heuristic Solution Procedures for Simple Assembly Line Balancing.” *European Journal of Operational Research* 168: 666–693.

Scholl, A., and R. Klein. 1997. “SALOME: A Bidirectional Branch-and-Bound Procedure for Assembly Line Balancing.” *INFORMS Journal on Computing* 9 (4): 319–334.

Scholl, A., and R. Klein. 1999. “Balancing Assembly Lines Effectively – A Computational Comparison.” *European Journal of Operational Research* 114 (1): 50–58.

Scholl, A., and S. Voß. 1996. “Simple Assembly Line Balancing-Heuristic Approaches.” *Journal of Heuristics* 2 (3): 217–244.

Schrimpf, G., J. Schneider, H. Stamm-Wilmbrandt, and G. Dueck. 2000. “Record Breaking Optimization Results using the Ruin and Recreate Principle.” *Journal of Computational Physics* 159: 139–171.

Sheu, D. D., and J.-Y. Chen. 2008. “Line Balance Analyses for System Assembly Lines in an Electronic Plant.” *Production Planning & Control* 19 (3): 256–264.

Subramanian, A., L. M. A. Drummond, C. Bentes, L. S. Ochi, and R. Farias. 2010. “A Parallel Heuristic for the Vehicle Routing Problem with Simultaneous Pickup and Delivery.” *Computers & Operations Research* 37 (11): 1899–1911.

Tirpak, T. M. 2008. “Developing and Deploying Electronics Assembly Line Optimization Tools: A Motorola Case Study.” *Decision Making in Manufacturing and Services* 2: 63–78.

Valério de Carvalho, J. M. 2002. “LP Models for Bin Packing and Cutting Stock Problems.” *European Journal of Operational Research* 141 (2): 253–273.

Wäscher, G., H. Haußner, and H. Schumann. 2007. “An Improved Typology of Cutting and Packing Problems.” *European Journal of Operational Research* 183: 1109–1130.