Solving the Multi-Choice Two Dimensional Shelf Strip Packing Problem with Time Windows

Horn, Matthias; Demirovic, Emir; Yorke-Smith, Neil

DOI
10.1609/icaps.v33i1.27229

Publication date
2023

Document Version
Final published version

Published in
Proceedings International Conference on Automated Planning and Scheduling, ICAPS

Proceedings of the Thirty-Third International Conference on Automated Planning and Scheduling (ICAPS 2023)

Solving the Multi-Choice Two Dimensional Shelf Strip Packing Problem with Time Windows

Matthias Horn*, **Emir Demirović**, **Neil Yorke-Smith**

Algorithmics group, Delft University of Technology, The Netherlands
{m.g.horn,e.demirovic,n.yorke-smith}@tudelft.nl

Abstract

In the tool coating field, scheduling of production lines requires solving an optimisation problem which we call the *multi-choice two-dimensional shelf strip packing problem with time windows*. A set of rectangular items needs to be packed in two stages: items are placed on shelves, which in turn are placed on one of several available strips. Crucially, the item's width depends on the chosen strip and each item is associated with a time window such that items can only be placed on the same shelf if their time windows overlap. In collaboration with an industrial partner, this real-world optimisation problem is tackled in this paper by both exact and heuristic methods. The exact method is an arc-flow-based integer linear programming formulation, solved with the commercial solver CPLEX. Experimental evaluation shows that this approach can solve instances to proven optimality with up to 20 different item sizes. Larger, more realistic instances are solved heuristically by an adaptive large neighbourhood search, using first fit and best fit decreasing approaches as repair heuristics. In this way, we obtain high-quality solutions with a remaining optimality gap below 3.3% for instances with up to 2000 different item sizes. The work reported is due to be incorporated into an end-to-end decision support system with the industrial partner.

1 Introduction

In the *multi-choice two-dimensional shelf strip packing problem with time windows* (M2SSPTW), a set of rectangular items needs to be packed into different types of strips with finite width but infinite height. The packing is done in two stages. First the items are placed on shelves which are, in the second step, placed on a strip. Hence, a shelf is a set of items that share the same horizontal line of a strip. A special feature is that the width of an item depends on the chosen strip and that each shelf has an additional height that also depends on the chosen strip. In addition, each item is associated with a time window (TW) s.t. items can only be placed on the same shelf if their TWs overlap. The objective is to minimise the total height over all strips. Figure 1 shows an example of an M2SSPTW instance with two strips and six items.

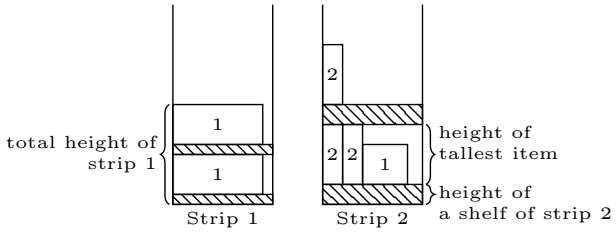

Figure 1: Packing example of an M2SSPTW instance with two strips $S = \{1, 2\}$ and two item types $I = \{1, 2\}$ with three items per type. The width of the item depends on the strip (see e.g. the width of an item of type 1 for strip 1 and strip 2).

One application of the M2SSPTW problem is in the field of tool coating, where it arises as a crucial sub-problem. In coating facilities, a batch of tools has to be coated with a coating material. For that purpose, a so-called *planetary* is assembled with tools and then put into one of several coating machines. The assembly of a planetary takes place in several steps. First, tools must be placed on cups. Each cup has a certain number of insert positions where tools can be positioned. Depending on the cup chosen, a tool can cover more than one insert position when placed on that cup. Since the space between insert positions varies with the chosen cup, the number of covered insert positions also varies with the cup. Second, a planetary contains several bars on which the cups equipped with tools are stacked. Finally, the assembled planetary is loaded into a coating machine and a feasible start time for the coating process is selected.

The overall coating problem consists of both bin packing and scheduling aspects. The M2SSPTW covers the first part, which deals with the placement of tools on cups. In this scenario the tools correspond to items and the cups correspond to shelves/strips. The width of an item corresponds to the number of covered insert positions when the tool is placed on a specific cup. The second part considers the scheduling aspects of the problem by determining which coating machine should process which fully equipped planetary at which time.

So far, our industrial partner has not used any optimisation algorithm. The employees decided how tools should be

*Currently at TU Vienna

placed on the planetary ad hoc, trying to save as much space as possible. However, they must make these decisions under time pressure to equip the planetary in time, so they may not consistently achieve the best solution.

A future deployment will include cooperation with a software company that provides a graphical interface for a static planning tool and incorporates the industrial partner's database system. The next step will be to embed our algorithms in this static planning tool. The final goal is to provide an end-to-end decision support system where our algorithms are a crucial part of solving the optimisation problem dynamically.

To our knowledge, the combination of shelf strip packing, strip-depending items and TWs is not considered in the literature so far. In contrast, it is a problem faced daily in the (coating) industry. We contribute by, first, formalising the real-world problem as M2SSPTW problem. Second, we devise approaches to solve the problem exactly by an arc-flow based *integer linear programming* (ILP) formulation and heuristically by an *adaptive large neighbourhood search* (ALNS). The initial solutions of the ALNS are constructed by adapted *first fit decreasing* (FFD) and *best fit decreasing* (BFD) heuristics. The ALNS uses five destruction heuristics and eight repair heuristics. At each iteration a destruction and a repair heuristic are randomly selected and applied on the incumbent solution in order to find improved solutions. The destruction heuristic removes part of the solution whereas the repair heuristic tries to reinsert the removed parts in a different way. The ALNS is able to obtain solutions with an optimality gap of at most 3.3% in a reasonable time.

The remainder of this work is organised as follows. After discussing related work in Section 2 we provide a formal problem description in Section 3. To solve instances to proven optimality, Section 4.1 describes an arc-flow based ILP formulation. In Section 4.2, we devise an FFD and a BFD approach to solving larger instances heuristically. Section 4.3 provides an ALNS meta-heuristic to improve obtained solutions from FFD or BFD even further. Computational Results will be reported and discussed in Section 5. Finally, Section 6 concludes this work with an outline of promising next research directions.

2 Related Work

The M2SSPTW problem is related to the well-known strongly NP-hard *bin packing* (BP) problem, e.g. see (Garey and Johnson 1978, 1979). The BP problem and numerous of its variants are well studied in the literature and various approximation algorithms have been devised (Coffman Jr. et al. 2013).

The most related problem is the *two-dimensional shelf strip packing* (2SSP) problem (Caprara, Lodi, and Monaci 2005). The M2SSPTW generalises the 2SSP with two main extensions: the M2SSPTW problem considers time windows and the width of items depends on the chosen strip type. For the 2SSP, an asymptotic (fully) polynomial time approximation scheme has been proposed by Caprara, Lodi, and Monaci (2005) that solves a series of ILPs. Let n be the number of items and let ε be the accuracy, the algorithm runs in $O(n \log n)$ time with a multiplicative constant double exponential in $1/\varepsilon$. To get a fully polynomial running time, the *linear programming* (LP) relaxations of the ILPs are solved approximately, which is paid with a huge constant that is added to the objective value. In Section 5, we will present the results of the algorithm (without LP rounding).

Another related problem is the *multi-choice vector bin packing* (MVBP) problem introduced by Patt-Shamir and Rawitz (2012), where multi-dimensional items have several incarnations s.t. the sizes in each dimension depend on the chosen incarnation. Furthermore, there are different bin types, each with its own cost and multi-dimensional size. The task is to pack each item with a chosen incarnation in a set of bins s.t. the overall bin cost is minimised. The aspect of different item incarnations applies to some extend also for the M2SSPTW problem with the main difference that items of the M2SSPTW problem *depend on the chosen strip*. The suggested solution approach by Patt-Shamir and Rawitz (2012) is a polynomial-time approximation algorithm if the number of considered dimensions for the items is constant. It is possible to transform an instance of the M2SSPTW problem to an instance of the MVBP problem by adding at least for each strip type and for each distinct item height an additional dimension to the MVBP instance. Then the number of dimensions for the items of the MVBP instance is no longer constant. Consequently, the proposed algorithm will no longer be a polynomial time algorithm. Therefore, applying their approximation algorithm to the M2SSPTW problem does not seem promising.

The MVBP problem can be solved exactly by an arc-flow formulation-based integer program proposed by Brandão and Pedroso (2013, 2016). The basic idea is to create for each bin type a directed graph with one root node and one target node such that an arc corresponds to a specific incarnation of an item that is packed into a bin of the corresponding bin type. A path from the root node to the target node describes a feasible packing of a subset of items indicated by the arcs along the path. Hence, to solve the MVBP problem, paths must be selected such that all items appear precisely one time in a specific incarnation on one of the chosen paths and thereby minimising the total bin cost. That can be done by a flow formulation-based ILP. To create the graphs, Brandão and Pedroso (2013, 2016) suggest a dynamic programming-based procedure that assigns to each node a state that consists of a vector. This state vector contains a dimension for each item dimension that indicates the current occupancy in that dimension. Furthermore, Brandão and Pedroso (2013, 2016) can remove a many nodes and arcs by lifting states and using compression techniques based on solving one-dimensional knapsack problems. In Section 4.1, we will use an adaptation of this procedure that considers time window constraints and non-additive item heights to solve the M2SSPTW directly without transforming it into the MVBP problem.

Another variant is the MVBP with conflict constraints, which was heuristically solved by Benazouz and Faure (2015, 2018) using a first fit decreasing algorithm. Their algorithm is strongly adapted to their industrial problem of automatically assigning power plants control function. Nev-

ertheless, their work is relevant for us since time window constraints can be modelled as conflict constraints. In Section 4.2, we will introduce our own first fit decreasing algorithm to solve the M2SSPTW problem.

3 Problem Description

An instance of the M2SSPTW consists of a set of n_I item types $I = \{1, 2, \dots, n_I\}$ and n_S different strips $S = \{1, 2, \dots, n_S\}$. Every item type $i \in I$ has a height $h_i^{\text{item}} \in \mathbb{R}_{>0}$ and a time window $W_i = [e_i, l_i]$ with an earliest day $e_i \in \mathbb{N}_{\geq 0}$ and a latest day $l_i \in \mathbb{N}_{\geq 0}$. Quantity $q_i \in \mathbb{N}_{>0}$ indicates the number of items of type i that must be packed into a strip. Each strip $s \in S$ has a width of one and each shelf that is packed into strip s is associated with a height $h_s^{\text{shelf}} \in \mathbb{R}_{>0}$. Since the width of an item depends on the item type and on the strip, we define $w_{is} \in \mathbb{R}_{>0}$ as the width of an item of type i that is packed into a shelf of strip s. Note that $w_{is} \in \mathbb{R}_{>0}$ can also be greater than one to indicate that an item of type i cannot be packed into a shelf of strip s.

A solution consists of a set $\mathcal{S}$ of shelves. Let $s^* \in \mathcal{S}$ be a shelf that is packed into strip $\text{st}(s^*) \in S$. A shelf s^* contains a set of items $\mathcal{I}(s^*) = \{(i, q), \dots\}$ where tuple (i, q) indicates the number of items q of item type $i \in I$ that are placed on s^*. The time window $W(s^*) = [e(s^*), l(s^*)]$ of s^* is the intersection of all time windows from all items that are placed on s^*, i.e., $W(s^*) = \bigcap_{(i,q)\in\mathcal{I}(s^*)} W_i$. Furthermore, let $w_s(s^*) = \sum_{(i,q)\in\mathcal{I}(s^*)} q\, w_{is}$ be the total width of all items placed on s^* if assigned to strip s. A solution is feasible if all items are placed on a shelf s.t. $w_{\text{st}(s^*)}(s^*) \leq 1$ and $W(s^*) \neq \emptyset$ holds for each $s^* \in \mathcal{S}$. The objective is to minimise the total height, i.e.,

$$\min \sum_{s^*\in\mathcal{S}} \left(h^{\text{shelf}}_{\text{st}(s^*)} + \max_{(i,q)\in\mathcal{I}(s^*)} h_i^{\text{item}} \right). \tag{1}$$

Note that the M2SSPTW is NP-hard since the BP problem can be reduced to it.

4 Solution Approaches

4.1 Integer Linear Programming

Before we solve instances of the M2SSPTW problem heuristically, we examine up to which size we can optimally solve instances. To this end, we use ideas from Brandão and Pedroso (2013, 2016) to model the M2SSPTW problem as an arc-flow formulation. The basic idea is that we create for each strip $s \in S$ an arc-flow graph $G_s = (V_s, A_s)$ with node set V_s and arc set A_s. The node set includes a source node S_s and a target node T_s. An arc corresponds to the insertion of an item of a specific item type i into a shelf of strip s. Each path from S_s to T_s corresponds to a specific *feasible placement* of items on a shelf of strip s. The height of a path is the largest item's height plus the height h_s^{shelf}. To solve the M2SSPTW problem, paths must be selected s.t. all items are placed on any strip and the total height of the selected paths is minimised.

The M2SSPTW problem is solved by using an ILP to compute a flow from S_s to T_s for each graph G_s s.t. all items are placed on any strip and the objective function is minimised. To create graph G_s each node $v \in V_s$ is associated with a state $\sigma_v = (h(v), w(v), e(v), l(v))$, where $h(v)$ corresponds to the so far largest height of the assigned items to the shelf from S_s to node v, width $w(v)$ corresponds to the so far claimed width, and $[e(v), e(l)]$ describes the remaining time window during that items can still be placed on the shelf. The state of the source node is $\sigma_{\text{S}_s} = (0, 0, \min_{i\in I} e_i, \max_{i\in I} l_i)$.

Note that this state definition differs from the definition in Brandão and Pedroso (2013, 2016) since they solve the *(multi-choice) vector bin packing ((M)VBP)* problem where the sizes of each dimension of items are added up if the items are put into the same bin. However, we consider rectangular items where only the widths are added up. For the heights of the items placed on the same shelf, we have to compute the maximum. In principle, we can model the M2SSPTW problem as an MVBP problem and use the same state definition suggested by Brandão and Pedroso (2013, 2016). However, this requires additional dimensions for the items of the MVBP problem for at least every strip times every distinct item height. Therefore this approach seems not to be promising. Instead, we use the described state definition that requires a constant number of four dimensions.

We still use the (slightly adapted) dynamic programming based procedure proposed by Brandão and Pedroso (2013, 2016) to create and compress for each strip an arc-flow graph G_s. To this end, we need to define our own transition function $\tau_s(\sigma_u, i) = \sigma_v$ with $h(v) = \max(h(u), h_i^{\text{item}})$, $w(v) = w(u) + w_{is}$, $e(v) = \max(e(u), e_i)$, and $l(v) = \min(l(u), l_i)$ that describes the transition from one state to another state if an item of type i is inserted into a shelf of strip s given state σ_u if $w(u) + w_{is} \leq 1$ and $[e(u), l(u)] \cap W_i \neq \emptyset$ holds. Otherwise transition function maps to the infeasible state $\hat{0}$, i.e., $\tau_c(\sigma_u, i) = \hat{0}$. The arc set of graph G_s is defined as

$$\begin{aligned} A_s = &\{(u, v, i) \mid \tau_s(\sigma_u, i) = \sigma_v,\ i \in I,\ u, v \in V_s \setminus \{\text{T}_s\}\} \\ &\cup \{(u, \text{T}_s, -1) \mid u \in V_s \setminus \{\text{S}_s\}\} \end{aligned} \tag{2}$$

Note that no node in V_s maps to the infeasible state. Hence, arcs that correspond to infeasible transitions are not included in G_s. Furthermore, the last term of Eq. (2) adds so called *loss arcs*, which do not indicate item assignments but connect each node to the target node. Figure 2 shows a small example of an arc-flow graph with one strip and two item types.

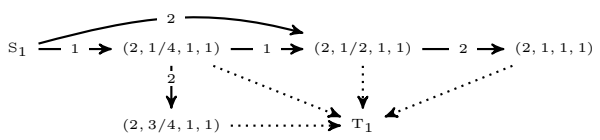

Figure 2: Arc-flow graph example with one strip and two item types $I = \{1, 2\}$ with $w_{11} = 1/4$, $w_{12} = 1/2$, and $q_1 = q_2 = 2$. Both item types have a height of two and the same time window $[1, 1]$. Dotted arcs correspond to loss arcs, which connect nodes to the target node.

Finally, to break symmetries, we have to define an ordering among items: see Brandão and Pedroso (2016). The items are sorted in non-increasing order according to their

heights. To break ties, we prefer the widest item. For more details on the algorithms to create and compress arc-flow graphs, we refer to (Brandão and Pedroso 2013, 2016).

After creating an arc-flow graph G_s for each strip $s \in S$, the following ILP model is solved using integer flow variables $0 \leq f_{sa} \leq q_i$ to represent the flow over arc $a = (u, v, i) \in A_s$. The objective function

$$\sum_{s \in S} \sum_{a=(u,\mathrm{T}_s,-1) \in A_s} f_{sa} \left(h(u) + h_s^{\mathrm{shelf}}\right) \tag{3a}$$

minimises the total height of the strips by taking into account the height of each used shelf and the height of the tallest item placed on that shelf. The flow conservation constraints

$$\sum_{a \in \delta_u^-} f_{sa} - \sum_{a \in \delta_u^+} f_{sa} = 0 \tag{3b}$$

for each $s \in S, u \in V_s \setminus \{\mathrm{S}_s, \mathrm{T}_s\}$ ensures that only source and target nodes produces and consumes flow, respectively. The expressions δ_u^- and δ_u^+ denote incoming and outgoing arcs, respectively. Constraints

$$\sum_{s \in S} \sum_{a=(u,v,i) \in A_s} f_{sa} = q_i \tag{3c}$$

for each $i \in I$ ensure that all items are inserted into a shelf.

Besides the presented arc-flow-based ILP model, we also considered an alternative ILP that uses binary decision variables to indicate the assignment of an item to a shelf of a specific strip. However, preliminary results showed that this ILP model does not compete with the ILP model (3a)–(3c).

4.2 First Fit and Best Fit Decreasing

A *first fit decreasing* (FFD) approach is used to find a heuristic solution for the M2SSPTW problem quickly. For this purpose, the item types are sorted in non-increasing order according to their heights. To break ties between two items of the same height, we prefer the wider item, i.e. item type i that maximises $\min_{s \in S} w_{is}$. According to this order, the first item type i is selected and all items of this type are placed on a shelf. This is done by iterating over all shelves that are opened so far. If items can be feasibly inserted into a shelf then we insert them. An item can be feasibly inserted into a shelf as long as the shelf, containing the item, *can be assigned to at least one strip*. Hence, we do not immediately assign shelves to a strip. If there are remaining items of type i after the iteration over all opened shelves then a new shelf is opened. This procedure is repeated for all item types until all items are placed on a shelf. Finally, all shelves are assigned to a strip s.t. shelf $s^* \in \mathcal{S}$ is assigned to strip $s \in S$ that minimises h_s^{shelf} and $w_s(s^*) \leq 1$ holds.

The *best fit decreasing* (BFD) approach does not insert items at the first possible position, like FFD, but instead inserts items at the *best* possible position according to a heuristic criterion. Let $\mathcal{S}$ be the set of shelves created so far, initialised with $\mathcal{S} = \emptyset$. Again, the item types are considered in the same order as for FFD. As many items of type $j \in I$ as possible are inserted into shelf s^* that meets the time window constraints and maximises the term $\max_{s \in S} \lfloor (1 - w_s(s^*))/w_{js} \rfloor$, that is the maximum number of items that can be inserted into a shelf from $\mathcal{S}$. A new shelf is opened if there is no shelf to insert items feasibly and there are remaining items left of type j. These steps are repeated until every item of type j is inserted into a shelf. Then the next item type is considered until all items of all item types are inserted. Finally, as with FFD, all shelves are assigned to a specific strip.

4.3 Adaptive Large Neighbourhood Search

The idea of *adaptive large neighbourhood search* (ALNS) is to apply a set of destruction and repair heuristics on an initial solution in order to find improved solutions. At each iteration, a destruction heuristic and a repair heuristic are randomly selected. The destruction heuristic removes some parts of the solution and the subsequently applied repair heuristic tries to reinsert the removed parts in a different way s.t. an improved solution may be obtained. The ALNS meta-heuristic was first successfully applied by Ropke and Pisinger (2006). Recently, ALNS has also been successfully applied to two-dimensional bin-packing problems (He et al. 2021; Zeng and Zhang 2021).

Destruction Heuristics The presented destruction heuristics are inspired by Ropke and Pisinger (2006). For the M2SSPTW problem, there are two kinds of destruction heuristics. Those that remove a certain number of items from shelves, and those that remove a certain number of whole shelves from a solution. Let $d^{\mathrm{alns}} \in (0, 1]$ be the proportion of items/shelves that are removed from the solution. Therefore the number of items to be removed is chosen randomly from $1, \ldots, \lfloor d^{\mathrm{alns}} \sum_{i \in I} q_i \rfloor$ and the number of shelves to be removed is chosen randomly from $1, \ldots, \lfloor d^{\mathrm{alns}} |\mathcal{S}| \rfloor$.

Random Shelf Removal. Remove uniformly randomly k selected shelves from $\mathcal{S}$.

Random Item Removal. Select randomly k shelves from $\mathcal{S}$. For each selected shelf s^* sort the items $\mathcal{I}(s^*) = \{(i_1, q_1), (i_2, q_2), \ldots, (i_n, q_n)\}$ in non-increasing order s.t. i_1 is the tallest item type and i_n the smallest item type. Then sample uniformly a random number x from interval $[1, n]$. If $x \leq n/2$ then all items of types i_x till i_n are removed from s^*; otherwise all items of types i_1 till i_x.

Related Shelf Removal. This removal heuristic follows the ideas by Shaw (1997, 1998). Hence, shelves are deleted that are related s.t. items placed on these shelves might be efficiently reinserted together again. The *proximity* $\eta_{s_1^*, s_2^*}^{\mathrm{shelf}}$ measures the dissimilarity of two shelves $s_1^*, s_2^* \in \mathcal{S}$ as

$$\begin{aligned} \eta_{s_1^*, s_2^*}^{\mathrm{shelf}} = \alpha^{\mathrm{shelf}} \min_{s \in S} \max\{0, w_s(s_1^*) + w_s(s_2^*) - 1\} \\ + (1 - \alpha^{\mathrm{shelf}}) \left| \frac{(e(s_1^*) + l(s_1^*)) - (e(s_2^*) + l(s_2^*))}{2 \max_{i \in I} l_i} \right| \end{aligned} \tag{4}$$

where the first term measures how well the items of two shelves would fit into just one shelf and the second term covers the temporal aspect, i.e., the distance between the centre of the time windows $W(s_1^*) = [e(s_1^*), l(s_1^*)]$ and $W(s_2^*) = [e(s_2^*), l(s_2^*)]$ of shelves s_1^* and s_2^*, respectively. Variable $\alpha^{\mathrm{shelf}} \in [0, 1]$ is a weight parameter that controls

the emphasis on the difference between the first and the second term of Eq. (4).

The following procedure is applied to remove k shelves from $\mathcal{S}$. Let D be the set of selected shelves.

1. select a shelf $s^* \in \mathcal{S}$ at random; set $D \leftarrow \{s^*\}$
2. while $|D| < k$ do
   - randomly select a shelf s^* from D
   - compute $\eta^{\text{shelf}}_{s^*,s^{*'}}$ for each shelf $s^{*'} \in \mathcal{S} \setminus D$
   - let $\{s^*_1, s^*_2, \dots\}$ be the sequence of shelves in $\mathcal{S} \setminus D$ sorted according to non-decreasing proximity $\eta^{\text{shelf}}_{s^*,s^*_i}$, $i = 1, 2, \dots, |\mathcal{S} \setminus D|$
   - select shelf s^*_j with $j = \lceil |\mathcal{S} \setminus D| \, \mathcal{R}^p \rceil$, where $\mathcal{R}$ is a uniformly distributed random number from $[0, 1)$ and p is a parameter set to six (see (Ropke and Pisinger 2006)).
   - $D \leftarrow D \cup \{s^*_j\}$

Related Item Removal. Similar to the related shelf removal heuristic, the related item removal heuristic attempts to find similar item types s.t. item of these types can be inserted back together more efficiently. Instead of removing whole shelves from the solution, this heuristic removes all items of certain item types.

The proximity $\eta^{\text{item}}_{i_1,i_2}$ of two item types $i_1, i_2 \in I$ is:

$$\eta^{\text{item}}_{i_1,i_2} = \alpha^{\text{item}} \min_{s \in S} \max\{0, w_{i_1 s} + w_{i_2 s} - 1\} + (1 - \alpha^{\text{item}}) \left(1 - \frac{|W_{i_1} \cap W_{i_2}|}{\min\{|W_{i_1}|, |W_{i_2}|\}}\right) \tag{5}$$

where $\alpha^{\text{item}} \in [0, 1]$ is a weight parameter that controls the emphasis of the difference between the first and the last term. The first term measures how well items of the two item types would fit into one shelf and the second term measures how well the corresponding time window overlap.

To select k item types a similar procedure is used as by the related shelf removal heuristic.

Open Shelf Removal. This removal heuristic attempts to remove *open* shelves, i.e., shelves that have some space left. We consider also a shelf as open if it can be moved to another strip s.t. there is leftover space on the shelf and further items can be inserted. Let $\{s^*_1, s^*_2, \dots\}$ be the sequence of shelves in $\mathcal{S}$ sorted according to non-increasing values of $\max_{s \in S} 1 - w_s(s^*)$. Apply following procedure to select k shelves from $\mathcal{S}$. Let D be the set of selected shelves.

1. while $|D| < k$ do
   - select shelf s^*_j with $j = \lceil |\mathcal{S} \setminus D| \, \mathcal{R}^p \rceil$, where $\mathcal{R}$ is a uniformly distributed random number from $[0, 1)$ and parameter $p = 6$.
   - $D \leftarrow D \cup \{s^*_j\}$

Repair Heuristics Repair heuristics attempt to reinsert items previously removed by one of the removal heuristics.

Best Fit and First Fit Heuristics. We use the FFD and BFD algorithms from Section 4.2 to repair a solution. In addition to a deterministic version of the algorithms, we also use a non-deterministic version that introduces some randomness in the following way: We iterate over the already sorted item types. With a probability of 25% an item type gets swapped with its direct successor. In case the of the FFD algorithm, we additionally shuffle the shelves in $\mathcal{S}$.

Close Shelf Heuristics. This heuristic attempts to add items by prioritising closing the remaining space on open shelves. An *open shelf* is a shelf that can hold at least one other item. Compared to BFD, this heuristic changes the point of view from items to shelves. Instead of choosing the best possible shelf to insert a specific item, this heuristic chooses the best possible item to close a specific shelf.

First, all open shelves are sorted according to some criterion which we will describe at the end of this paragraph. The heuristic starts with the first shelf s^* according to the chosen sorting criterion. From the set of non yet inserted items, select q_j items of type j that can be feasible inserted into s^* and minimise the term $\min_{s \in S} 1 - w_s(s^*) - q_j \, w_{js}$. In the case of ties, we prefer the item type from which more items can be inserted. Note that the strip type of s^* can be changed during the insertion. This step is repeated until no additional item can be feasibly placed on s^*. Then the next shelf, according to the sorted sequence of open shelves, is selected and the previous steps are repeated. If all open shelves are processed, the BFD heuristic is used to insert remaining items.

We use two different sorting criteria for shelves. The first criterion sorts the shelves in non-decreasing order according to their tallest contained item height, whereas the second sorts the shelves in non-decreasing order according to the number of non-inserted items that can be feasibly inserted into the shelf. Both sort criteria prefer the shelf with less available space in case of ties. Furthermore, we consider non-deterministic versions of the close shelf heuristic by iterating over the sorted shelves and swapping with a probability of 25% two subsequent shelves. In total, we use four sorting criteria for the close shelf heuristic.

In preliminary experiments, we also considered using the ILP model from Section 4.1 within the LNS as a repair operator to solve subproblems optimally. It turns out that CPLEX takes too long to solve the subproblems, such that the overall performance of the LNS breaks down and provides worse solutions than without using the ILP model.

Selection of Destruction and Repair Heuristics At each iteration a destruction and a repair heuristic are randomly selected by the roulette wheel selection principle. In the following, we describe the selection of a destruction heuristic. The same procedure can be applied similarly to select a repair heuristic. Let h^{des} be the number of used destruction heuristics and let w^{des}_i be the current weight of the i-th destruction heuristic, $1 \leq i \leq h^{\text{des}}$. The probability to select the i-th destruction heuristic is $w^{\text{des}}_i / \sum_{j=1}^{h^{\text{des}}} w^{\text{des}}_j$.

The weights are updated by evaluating the performance of the corresponding heuristic from earlier iterations by tracking for each heuristic i the score σ^{des}_i that measures how well heuristic i performed. For this purpose, the search process is divided into segments of l^{alns} iterations. At the beginning of each segment, the scores are reset to zero. All weights are

initialised with 1.0 and updated after each segment by

$$w_i^{\text{des}} \leftarrow (1-\gamma) w_i^{\text{des}} + \gamma \, \sigma_i^{\text{des}} / n_i^{\text{des}} \tag{6}$$

for all heuristics i, where n_i^{des} denotes the number of times heuristic i was applied during the last segment and $\gamma \in [0,1]$ is a parameter to control the speed of the weight adjustment.

The score of selected heuristic i is updated by

$$\sigma_i^{\text{des}} \leftarrow \sigma_i^{\text{des}} + \begin{cases} \sigma_1 & \text{if found new best solution} \\ \sigma_2 & \text{if found better solution than current} \\ \sigma_3 & \text{if acc. worse solution than current} \\ 0 & \text{otherwise} \end{cases} \tag{7}$$

after every iteration. We use the same parameter values as Ropke and Pisinger (2006): $\sigma_1 = 33$, $\sigma_2 = 9$, $\sigma_3 = 13$.

We also accept worse solutions based on a simulating annealing approach to avoid local optima. We accept a solution $\mathcal{S}'$ with a total height H' that is worse than the current solution $\mathcal{S}$ with a total height H with probability $\exp(-(H'-H)/T^{\text{alns}})$, where $T^{\text{alns}} \geq 0$ is the current temperature. At every iteration the temperature is decreased by $T^{\text{alns}} \leftarrow T^{\text{alns}} \, c^{\text{alns}}$, were $c^{\text{alns}} \in [0,1]$ is the cooling rate. At the beginning the temperature is initialised with $T^{\text{alns}}_{\text{start}}$.

5 Computational Results

All tested algorithms were implemented in C++. All tests were performed on a single core of an Intel Xeon E5-6248R processor with a memory limit of 32GB RAM. The ILP models from Section 4.1 were solved with CPLEX 20.1 using default settings. Instances are available at https://doi.org/10.4121/20382753.

We created two non-trivial sets, A and B, of random benchmark instances. The first set A is generated completely randomly, whereas the second set B is based on real-world data.[1] Each instance class of benchmark set A consists of 30 instances for each combination of $n_I \in \{10, 20, 50, 100, 150, 200, 250\}$ item types and $n_S \in \{3, 4, 5\}$ strips. Set B is based on the scenario from Section 1 where tools have to be coated. Further, we assume that there are much smaller tools (items) than larger tools and that there are four different cups (shelves). Hence, we create 30 instances for each combination of $n_I \in \{750, 1000, 1250, 1500, 1750, 2000\}$ and a fixed number of strips, $n_S = 4$. Small tools can only be placed on the first and second cups; the largest tools can only be placed on the third and fourth cups; medium tools can be placed on any cup. To this end, we choose a thin item with a probability of 0.5 and a medium or wide item with a probability of 0.25 each.

The parameters of the devised ALNS algorithm from Section 4.3 were tuned to quickly obtain high-quality solutions within a CPU time limit of 5 minutes per run. To this end, we used the automatic parameter configuration tool *irace* (López-Ibáñez et al. 2016) and tuned parameters for benchmark sets A and B separately. We created two independent tuning instance sets

[1]Real data from industrial tool coating problem instances. Due to commercial confidentiality, the actual data cannot be disclosed.

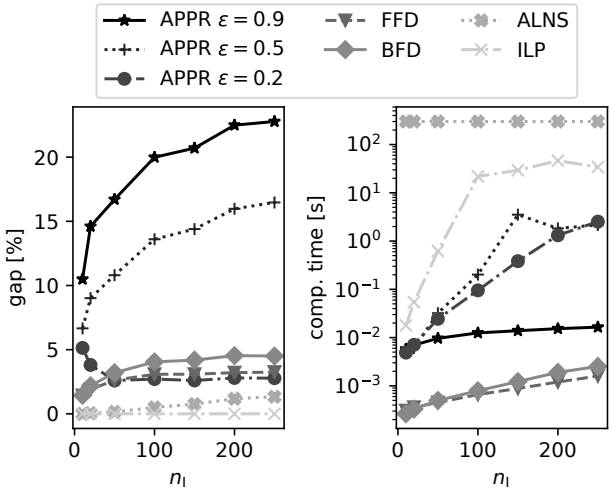

Figure 3: Optimality gaps and computation times obtained from APPR for different values of ε as well as from FFD, BFD, ALNS, and ILP (CPLEX).

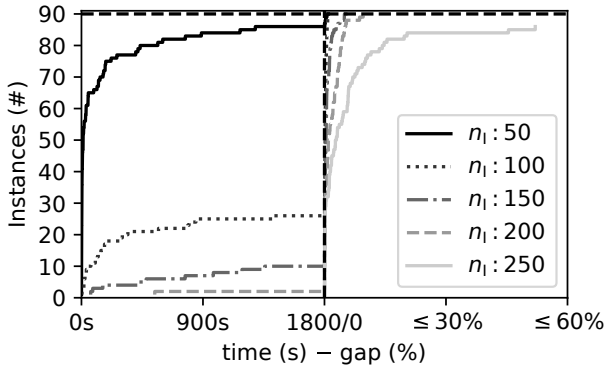

Figure 4: Performance plot of CPLEX results using the arc-flow-based ILP formulation from Section 4.1 with a time limit of 1800 seconds.

for the benchmark sets A and B by creating for each instance class five instances and tuned seven parameters: reaction factor $\gamma \in \{0.0, 0.1, 0.2, \ldots, 1.0\}$, start temperature $T^{\text{alns}}_{\text{start}} \in \{10, 15, \ldots, 60\}$, cooling factor $c^{\text{alns}} \in \{0.9, 0.95, 0.99, \ldots, 0.999999\}$, segment length $l^{\text{alns}} \in \{10, 20, 50, 100, 200, \ldots, 50000\}$, destroy rate $d^{\text{alns}} \in \{1, 10, 15, 20, 25, \ldots, 90\}$, and weight parameters $\alpha^{\text{shelf}}, \alpha^{\text{item}} \in \{0.0, 0.1, \ldots, 1.0\}$, for the related shelf and related item removal heuristics. We used a budget of 10000 runs. In this way the following configurations were obtained by `irace`: $\gamma = 0.7$, $c^{\text{alns}} = 0.99995$, $T^{\text{alns}}_{\text{start}} = 45$, $l^{\text{alns}} = 50000$, $d^{\text{alns}} = 10\%$, $\alpha^{\text{shelf}} = 0.2$, and $\alpha^{\text{item}} = 0.3$ for benchmark set A and $\gamma = 0.4$, $c^{\text{alns}} = 0.99$, $T^{\text{alns}}_{\text{start}} = 20$, $l^{\text{alns}} = 200$, $d^{\text{alns}} = 1\%$, $\alpha^{\text{shelf}} = 0.4$, and $\alpha^{\text{item}} = 0.3$ for benchmark set B.

In the following we will present first results of experiments that compare the approximation algorithm from

	n_S	n_I	FFD %-gap	FFD $\sigma_{\%\text{-gap}}$	FFD t[s]	BFD %-gap	BFD $\sigma_{\%\text{-gap}}$	BFD t[s]	ALNS %-gap	ALNS $\sigma_{\%\text{-gap}}$	ALNS %-opt	ILP (CPLEX) %-gap	ILP (CPLEX) $\sigma_{\%\text{-gap}}$	ILP (CPLEX) %-opt	ILP (CPLEX) t[s]
A	3	10	4.67	3.46	<1	1.20	1.75	<1	0.05	0.14	83	**0.00**	0.0	100	<1
		20	5.16	3.86	<1	2.19	1.92	<1	0.06	0.15	73	**0.00**	0.0	100	<1
		50	6.39	3.06	<1	2.95	0.93	<1	0.29	0.24	3	**0.00**	0.0	100	5
		100	6.33	2.33	<1	4.45	0.94	<1	0.72	0.33	0	**0.02**	0.05	70	172
		150	6.31	2.08	<1	4.34	0.86	<1	1.10	0.28	0	**0.08**	0.08	10	280
		200	6.47	1.61	<1	4.97	0.72	<1	1.59	0.33	0	**0.22**	0.27	3	299
		250	6.30	1.53	<1	5.18	0.72	<1	2.02	0.39	0	**0.48**	0.48	0	299
	4	10	5.23	4.76	<1	1.41	1.84	<1	0.11	0.42	87	**0.00**	0.0	100	<1
		20	5.43	2.91	<1	2.27	1.15	<1	0.11	0.27	73	**0.00**	0.0	100	<1
		50	6.51	2.35	<1	4.47	1.75	<1	0.37	0.32	3	**0.00**	0.0	100	59
		100	7.67	2.11	<1	5.78	1.13	<1	1.00	0.29	0	**0.24**	0.19	0	299
		150	7.46	1.43	<1	6.25	1.02	<1	1.65	0.31	0	**1.34**	1.58	0	299
		200	7.17	1.04	<1	6.19	0.85	<1	**2.04**	0.30	0	4.01	2.80	0	300
		250	7.66	1.20	<1	6.55	0.81	<1	**2.65**	0.48	0	6.35	3.00	0	302
	5	10	6.20	4.27	<1	1.44	1.52	<1	0.13	0.35	77	**0.00**	0.0	100	<1
		20	6.89	3.64	<1	3.43	1.79	<1	0.13	0.35	63	**0.00**	0.0	100	<1
		50	7.47	2.12	<1	4.84	1.46	<1	0.52	0.48	3	**0.07**	0.20	83	160
		100	8.80	2.18	<1	6.49	1.18	<1	1.55	0.50	0	**0.92**	0.93	0	298
		150	8.36	1.85	<1	6.92	1.02	<1	**2.05**	0.42	0	5.97	5.35	0	300
		200	8.64	1.35	<1	7.26	1.19	<1	**2.50**	0.38	0	11.27	5.80	0	300
		250	8.76	1.15	<1	7.41	1.02	<1	**3.08**	0.49	0	15.57	11.27	0	304
B	4	750	5.70	0.47	<1	4.31	0.42	<1	**3.23**	0.27	0	-	-	-	-
		1000	5.45	0.43	<1	4.09	0.39	<1	**3.08**	0.31	0	-	-	-	-
		1250	5.05	0.29	<1	3.87	0.32	<1	**2.88**	0.18	0	-	-	-	-
		1500	4.91	0.26	<1	3.82	0.26	<1	**2.88**	0.23	0	-	-	-	-
		1750	4.84	0.31	<1	3.86	0.32	<1	**2.85**	0.19	0	-	-	-	-
		2000	4.67	0.25	<1	3.71	0.24	<1	**2.82**	0.30	0	-	-	-	-

Table 1: Aggregated main results of FFD, BFD, ALNS, and CPLEX for benchmark sets A and B.

Caprara, Lodi, and Monaci (2005) with our own devised algorithms FFD, BFD, and ALNS. The goal of the second type of experiments is to solve as many instances as possible with CPLEX using the ILP model from Section 4.1. The last experiments compare the performance of FFD, BFD, ALNS, and CPLEX for bechmark sets A and B.

Our first experiments compare the approximation algorithm from Caprara, Lodi, and Monaci (2005), denoted as APPR, with the FFD, BFD, and ALNS algorithm from Sections 4.1–4.3. For ALNS we used the tuned parameters and BFD to construct an initial solution. The obtained solution quality of the APPR algorithm depends on the accuracy parameter $\varepsilon \geq 0$. Note that APPR cannot handle multiple choice properties of items as well as time window constraints. Therefore, in the following experiments, we consider only instance classes from set A with $n_C = 4$ by using only the first cup and ignoring all time windows. Figure 3 compares obtained average optimality gaps and average computation times for different values of $\varepsilon \in \{0.2, 0.5, 0.9\}$. The optimality gaps are computed by $100\% \cdot (\mathrm{OBJ} - \mathrm{LB})/\mathrm{LB}$ where OBJ is the obtained objective value from one of the used approaches and LB is the best lower bound obtained from the ILP model. As expected, we obtain better results from APPR if we decrease the accuracy value ε. Solutions obtained with values 0.9 and 0.5 for ε cannot compete with the solution quality obtained from the other considered approaches. However, for $\varepsilon = 0.2$, the obtained optimality gaps are smaller than the gaps obtained from FFD and BFD for instances with more than 50 item types. Note that for smaller values of ε, the APPR algorithm could not compute a solution within the memory limit of 32GB. The best gaps are always obtained from CPLEX, followed by ALNS. Regarding computation times, the fastest approaches are FFD and BFD.

Our second experiments disclose the performance of CPLEX using the ILP model from Section 4.1. The goal is to solve as many instances to proven optimality as possible by using a time limit of 1800 seconds. All instances with up to 20 item types can be solved to proven optimality within at most 15 seconds. For the remaining instances of set A, Figure 4 provides a performance plot by aggregating over the number of strips. Hence, for this plot, we group all instances of set A by the number of different item types such that a group contains 90 instances with three, four and five strips. The plot in Figure 4 is divided into two parts: the left part shows the number of instances that CPLEX has solved within a certain number of seconds. For instances that could not be solved to proven optimality within 1800 seconds, the right part shows the remaining optimality gap for the number of instances. Almost all instances with 50 item types can be solved to optimality. The remaining five instances have a small average optimality gap of 0.41%. No instances with

250 item types can be solved to proven optimality within 1800 seconds; for five instances CPLEX cannot find a feasible solution. The remaining instances have an average optimality gap of 4.77%. While CPLEX provides excellent results for instances of set A, for most of the larger instances of set B, CPLEX could not obtain even a single feasible solution within the given time limit.

Finally, Table 1 presents the main aggregated (heuristic) results obtained from FFD, BFD, ALNS, and CPLEX for benchmark sets A and B. Columns %-gap state the average optimality gap of final solutions and columns $\sigma_{\%\text{-gap}}$ provide the corresponding standard deviations. To compute the optimality gap, we use the best lower bound obtained from CPLEX for instances of set A, for the larger instances of set B, we solved the LP relaxation of the ILP model from Section 4.1. Columns %-opt provides the percentage of instances that could be solved optimally. Columns t list the average computation times of the algorithm. In this scenario, we are interested in obtaining high-quality heuristic solutions in a short time. Therefore we use a time limit of 300 CPU seconds for all considered approaches. For the ALNS, we used the tuned parameters and BFD to obtain an initial solution. Table 1 reveals that FFD and BFD can solve, on average, all instance classes in under 30 milliseconds. Generally, if the number of strips increases, the obtained solution quality worsens. This is observed for all considered approaches. On average, BFD produces better solutions with an optimality gap of at most 7.5% than FFD, which produces solutions with a gap of at most 8.8%. For instances of set A with three and four strips and up to 150 item types, CPLEX provides excellent results. However, ALNS is able to provide better solutions for instances with five strips and more than 150 item types within the same time. CPLEX could not derive a feasible solution for instances of set B within the time limit, whereas the best solutions obtained from ALNS have a gap of at most 3.3%. Interestingly, for instances of set B, the quality of obtained solutions increases with the number of item types. This is because these instances are based on real-world instances with a relatively high proportion of small items. This makes it easier for the heuristics to find good solutions since small items can be used to fill up the remaining spaces.

6 Conclusions

We considered the industrial problem of packing a set of rectangular items into a set of strips in two stages. We devised an exact ILP approach that can solve instances with up to 20 different item types to proven optimality. For problem instances with up to 2000 item types, we designed FFD/BFD approaches that can greedily construct solutions in under one second. These solutions are further improved by an ALNS that uses five different destruction heuristics and eight repair heuristics. After five minutes, the solutions obtained from ALNS have an average optimality gap below 3.3%.

The following steps of the deployment process will be to deeply analyse our obtained solutions by evaluating historical data provided by our industrial partner. However, we can already report that using our algorithms can save enormous time due to the obtained computation times of our algorithms. Furthermore, already based on the results, the senior management is excited to introduce scheduling technology into their core production process.

Further research may consider more repair and destruction heuristics for the ALNS to obtain even better solutions. On the exact side, column generation approaches may be a promising direction to solve larger instances to optimality.

Acknowledgements

Thanks to the anonymous reviewers. This work was partially supported by TAILOR, a project funded by the EU Horizon 2020 programme under grant 952215.

References

Benazouz, M.; and Faure, J. M. 2015. Safety-Level Aware Bin-Packing Approach for Control Functions Assignment. In *15th IFAC Symposium on Information Control Problems in Manufacturing*, volume 48, 507–512.

Benazouz, M.; and Faure, J.-M. 2018. Safety-Level Aware Bin-Packing Heuristic for Automatic Assignment of Power Plants Control Functions. *IEEE Transactions on Automation Science and Engineering*, 15(2): 602–612.

Brandão, F.; and Pedroso, J. P. 2013. Multiple-choice vector bin packing: Arc-flow formulation with graph compression. Technical report. DCC-2013-13, Faculdade de Ciencias da Universidade do Porto.

Brandão, F.; and Pedroso, J. P. 2016. Bin packing and related problems: General arc-flow formulation with graph compression. *Computers & Operations Research*, 69: 56–67.

Caprara, A.; Lodi, A.; and Monaci, M. 2005. Fast Approximation Schemes for Two-Stage, Two-Dimensional Bin Packing. *Mathematics of Operations Research*, 30(1): 150–172.

Coffman Jr., E. G.; Csirik, J.; Galambos, G.; Martello, S.; and Vigo, D. 2013. *Bin Packing Approximation Algorithms: Survey and Classification*, 455–531. New York, NY: Springer.

Garey, M. R.; and Johnson, D. S. 1978. "Strong" NP-Completeness Results: Motivation, Examples, and Implications. *Journal of ACM*, 25(3): 499–508.

Garey, M. R.; and Johnson, D. S. 1979. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. New York, NY, USA: W. H. Freeman & Co.

He, K.; Tole, K.; Ni, F.; Yuan, Y.; and Liao, L. 2021. Adaptive large neighborhood search for solving the circle bin packing problem. *Computers & Operations Research*, 127: 105140.

López-Ibáñez, M.; Dubois-Lacoste, J.; Cáceres, L. P.; Birattari, M.; and Stützle, T. 2016. The irace package: Iterated racing for automatic algorithm configuration. *Operations Research Perspectives*, 3(Supplement C): 43–58.

Patt-Shamir, B.; and Rawitz, D. 2012. Vector bin packing with multiple-choice. *Discrete Applied Mathematics*, 160(10): 1591–1600.

Ropke, S.; and Pisinger, D. 2006. An Adaptive Large Neighborhood Search Heuristic for the Pickup and Delivery Problem with Time Windows. *Transportation Science*, 40(4): 455–472.

Shaw, P. 1997. A new local search algorithm providing high quality solutions to vehicle routing problems. Technical report, Departure of Computer Science, University of Strathclyde, Scotland.

Shaw, P. 1998. Using constraint programming and local search methods to solve vehicle routing problems. In *International Conference on Principles and Practice of Constraint Programming (CP 98)*, volume 1520 of *LNCS*, 417–431. Springer.

Zeng, J.; and Zhang, X. 2021. An Adaptive Large Neighborhood Search for Two-Dimensional Packing with Conflict Penalty. In *2021 8th International Conference on Dependable Systems and Their Applications (DSA)*, 12–19. IEEE.