Parallel Batch Processing for the Coating Problem

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

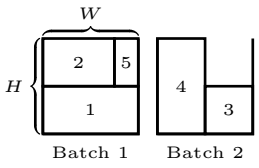
Algorithmics group, Delft University of Technology, The Netherlands
{m.g.horn,e.demirovic,n.yorke-smith}@tudelft.nl

Abstract

We solve a challenging scheduling problem with parallel batch processing and two-dimensional shelf strip packing constraints that arises in the tool coating field. Tools are assembled on so-called planetaries (batches) before they are loaded into coating machines to get coated. The assembling is not trivial and must fulfil specific constraints, which we refer to as shelf strip packing constraints. Further, each tool is associated with a starting time window s.t. tools can only be put on the same planetary if their time window overlap. The objective is to minimise the makespan and the number of required planetaries. Since the problem naturally decomposes into scheduling and packing parts, we tackle the problem with a two-phase logic-based Benders decomposition approach. The master problem assigns items to batches. The first phase solves as subproblem the packing problem by checking if the assignment is feasible, whereas the second phase solves the scheduling subproblem. The approach is compared with a monolithic mixed integer linear programming approach as well as a monolithic constraint programming approach. Experimental evaluation shows that our proposed approach outperforms the state-of-the-art benchmarks by solving more instances to optimality in a shorter time.

1 Introduction

An essential task in the tool coating industry is to assemble so-called *planetaries* with tools that should be coated with the same coating material before they are put into a coating machine. To increase productivity, assembling as many tools as possible on one planetary is important. Further, tools are assigned with a time window that indicates when the coating process should start s.t. tools can only be put on the same planetary if their time window overlap. The assembly takes place in two steps: (1) tools are placed on cups and (2) cups are put on planetaries. This advanced bin packing problem can be modelled as the *two-dimensional shelf strip packing* (2DSSP) problem (Caprara, Lodi, and Monaci 2005). Another important aspect in the industry is not only to assemble planetaries but also to consider the underlying scheduling problem, i.e., selecting for each assembled planetary a coating machine and a specific time when coating should start. The coating duration depends on the coating material and between two subsequent coating processes, a specific setup time is needed to, e.g., clean the machine.



item	h_i	w_i
1	1	4
2	1	3
3	1	2
4	2	2
5	1	1

Figure 1: Example of an SPBP instance with five items, width $W = 4$, and height $H = 2$. At least two batches are needed to pack all items, ignoring time windows and items' family type. The first batch consists of two shelves with items $\{1\}$ and $\{2, 5\}$, respectively, whereas the second batch consists of one shelf with items $\{3, 4\}$.

Therefore, this work covers a challenging variant of the *scheduling with parallel batch processing* (SPBP) problem to model important aspects of the above described industrial problem. In this scenario, two-dimensional items can only be grouped into batches when they fulfil a 2DSSP constraint. The batches are then sequentially processed by one of multiple homogeneous machines s.t. the makespan and the number of batches are minimised. The machines have a specific width and height. To fulfil the 2DSSP constraint, items are first grouped into so-called shelves (cups) s.t. items within a shelf can only be placed side by side and the total width of the items does not exceed the width of the machines. The tallest item of a shelf defines the height of a shelf. In the second step, the shelves are grouped into batches (planetaries) by placing them on top of each other s.t. the shelves' total height is at most the maximum height of the machines. Each item belongs to a specific family and is associated with a time window. Items can only be packed into the same batch if they belong to the same family and their time windows overlap. A batch's processing time depends on the packed items' family. In addition, there is a sequence-dependent setup time between batches that also depends on the families. Figure 1 shows an example of five items with different dimensions that are grouped into two batches. Note that our problem definition abstracts away some of the more complicated application-specific details (e.g., different characteristics for different machines or different cup and planetary

*Currently at TU Vienna

types) in order to investigate the core complexities of the SPBP problem with 2DSSP constraints.

To our knowledge, the combination of SPBP and 2DSSP constraint is not considered in the literature so far. This paper tries to solve the SPBP problem with 2DSSP constraints exactly by a *logic-based Benders decomposition* (LBBD) approach that splits the problem into a master problem and two subproblems. The master problem assigns items to a minimum number of batches by solving a *mixed-integer linear programming* (MIP) model. The first subproblem checks if the assigned items fulfil the 2DSSP constraints by solving a network-flow-based (NF) MIP model over a *binary decision diagram* (BDD). To strengthening the master problem, the NF model is added to it as a linear relaxation each time the 2DSSP constraints are violated. Finally, the second subproblem uses a *constraint programming* (CP) model to schedule the batches to minimise the makespan.

We contribute to the literature by, first, formalising the real-world problem. Second, we devise and compare multiple modelling approaches to solve the problem exactly: a monolithic MIP model, a monolithic CP model, and an LBBD approach. Third, to do so we formulate a BDD model of the 2DSSP constraint.

2 Related Work

Variants of the SPBP problem are well-studied in the literature. An important field where batch scheduling finds its application is semiconductor manufacturing, where batching can be modelled as a bin packing problem (Mönch et al. 2011). A recent survey (Fowler and Mönch 2021) distinguishes between compatible and incompatible family types, different machine environments and performance measures. An SPBP problem with compatible family types means that items of different families can be batched together, whereas incompatible types allow only to batch items of the same family together. The problem considered in this work considers items with incompatible family types and a parallel machine environment. Besides various works on single and multi machines, i.e., Trindade, de Araújo, and Fampa (2021); Queiroga et al. (2021); İbrahim Muter (2020), the literature also considers different kinds of flow shop and job shop environments, i.e., Shahnaghi et al. (2016b,a); Wang and Luh (1997). For more information on alternative objective functions and proposed solution techniques, we refer to the survey from Fowler and Mönch (2021).

A related problem to the SPBP problem with 2DSSP constraints is the *single batch processing problem with two-dimensional bin packing constraints* (SBPM-2D) introduced by Li and Zhang (2018), which deals with two-dimensional items and two-dimensional batches. Items are allowed to be rotated and are always placed parallel to the machine's edge. They presented a *mixed integer programming* (MIP) model to solve the problem exactly, as well as heuristic algorithms to solve larger instances. The heuristic algorithms include a biased random-key genetic algorithm, four single-sequence-based heuristics, and a hybrid greedy algorithm. There are two main differences between the SPBP problem with 2DSSP constraints and the SBPM-2D problem: (1) the latter considers a single machine, whereas this work considers multiple parallel machines, and (2) the two-dimensional items can be rotated and freely moved so that they fit into a rectangular area whereas here the items are further constrained by arranging them into shelves.

Another relevant work by Tang and Beck (2020) solves a *two-stage bin packing and hybrid flowshop scheduling problem* (2BPHFSP) that arise in composites manufacturing. The objective is to minimise the number of open bins and the total job tardiness. The work proposes five solution approaches; a three-phase LBBD approach, a monolithic CP approach, and various hybrid/CP heuristics. While the problem itself is not identical with our problem (one-dimensional two-stage bin packing vs. 2DSSP and hybrid flowshop scheduling vs. multi machine scheduling), we will use a similar three-phase LBBD approach to solve the SPBP problem. The first stage, the master problem, groups items into batches s.t. the second stage, the first subproblem, checks if the assigned items fulfil the packing constraints. If this is the case, the third stage schedules the batches to machines s.t. the total tardiness is minimised.

3 Problem Description

The SPBP problem with 2DSSP constraints consists of a set of n items $I = \{1, 2, \ldots, n\}$ and a set of m machines $M = \{1, 2, \ldots, m\}$. Each item $i \in I$ is associated with a width $w_i \in \mathbb{N}_{>0}$, a height $h_i \in \mathbb{N}_{>0}$, a release time $r_i \in \mathbb{N}_{>0}$, and a deadline $d_i \in \mathbb{N}_{>0}$. Note that the time window $[r_i, d_i]$ refers to the start time of processing item i on a machine. In addition, each item belongs to a family $a_i \in A$ from the set of families $A = \{1, 2, \ldots, o\}$ with o family types. For convenience, we define $I_a = \{i \in I \mid a_i = a\}$ as the subset of items that belong to family a. Let $R_a(t) = \{i \in I_a \mid r_i = t\}$ and $D_a(t) = \{i \in I_a \mid d_i = t\}$ be the sets of all items of family a that have the same release times and deadlines t, respectively. The task is to group the items into *batches* that will be processed sequentially on a machine $j \in M$. Items can only be grouped into a batch if their time windows overlap, if all items belong to the same family and if they can be feasibly packed into the machine. Thereby the items are placed on a two-dimensional strip with width $W \in \mathbb{N}_{>0}$ and height $H \in \mathbb{N}_{>0}$ in two stages. First, a batch of items is grouped into shelves where items are placed horizontally on a shelf s.t. the total width of the placed items is at most W. The height of a shelf is determined by its tallest assigned item. Second, the shelves are packed vertically into the machine. If the total height of the shelves is at most H then the packing is feasible and the machine can process the items. The processing time $p_a \in \mathbb{N}_{>0}$ depends on the loaded item's family $a \in A$. Between two batches $a, b \in A : a \neq b$, the set-up time δ_{ab} also depends on item families. The starting time of a batch must be within the assigned item's time window.

A solution to the SPBP problem consists of a set of batches $\mathcal{B}$. Each batch $k \in \mathcal{B}$ is associated with a set of assigned items $I(k)$, a family $a(k)$, a time window $[r(k),\, d(k)] := \cap_{i \in I(k)} [r_i,\, d_i]$, an assigned machine $m(k)$, and an assigned starting time $s(k) \in [r(k),\, d(k)]$.

Time Window Decomposition Most of the approaches devised in Section 5 will make use of a so-called *time window decomposition*. Instead of assuming that a solution can have at most n batches and the assigned items determine the time window and the family of a batch, we assume that there are already predefined batches $k \in \mathcal{B}$ with an assigned time window $[r(k), d(k)]$ and an assigned family $a(k)$ in advance. Only items that overlap entirely with the batch's time window, i.e. items of set $X(k) = \{i \in I_{a(k)} \mid r_i \leq r(k) \wedge d(k) \leq d_i\}$, can be assigned to batch k. Furthermore, we require that the intersection of the items' time window must result in $[r(k), d(k)]$, i.e. there must be at least one item assigned to the batch each from set $R(r(k))$ and $D(d(k))$. The advantages are two-folded: (1) in this way, symmetries are removed since not every item can be assigned to every batch, and (2) it is more convenient to define logic-based Benders cuts since batches with a certain time window can be addressed directly (see Eq. (8) and (9) in Section 5.3). To this end, let set $\mathcal{T}_a = \{[r_i, d_i] \cap [r_j, d_j] \mid i, j \in I_a\}$ be the set of all possible intersected time windows of items of family a, which can be computed in at most $O(n^2)$ steps. For each family $\bar{a} \in A$ and each time window $[\bar{r}, \bar{d}] \in \mathcal{T}_{\bar{a}}$ we add $\min\{|R(\bar{r})|, |D(\bar{d})|\}$ batches k to $\mathcal{B}$ with release time $r(k) := \bar{r}$, deadline $d(k) := \bar{d}$, and family $a(k) := \bar{a}$.

4 BDDs for the 2DSSP Constraint

In order to model the 2DSSP constraint, our approaches in Section 5.1 will use a network flow model on an exact zero-suppressed *binary decision diagram* (BDD) that represents each possible assignment of a set X of items to shelves. The main idea is that the model is infeasible if and only if it is impossible to assign the items in X so that the 2DSSP constraints are satisfied.

Originally, BDDs were introduced to represent Boolean functions graphically. In the last decade, BDDs found their application also in combinatorial optimisation, where they are successfully applied to well-studied standard optimisation problems (Andersen et al. 2007; Cire and van Hoeve 2013; Bergman et al. 2014; Kinable, Cire, and van Hoeve 2017). For a detailed introduction we refer to the textbook by Bergman et al. (2016) and to the survey of recent advances by Castro, Cire, and Beck (2022).

In the context of this work, a BDD $\mathcal{D}_X = (\mathcal{N}_X, \mathcal{A}_X)$ for items $X \subseteq I$ is a layered-directed acyclic multi-graph with node set $\mathcal{N}_X$, arc set $\mathcal{A}_X$, and root node $\mathbf{r} \in \mathcal{N}_X$. The nodes are partitioned into $|X| + 1$ layers L_q,$0 \leq q \leq |X|$. The first layer L_0 is a singleton containing only $\mathbf{r}$. An arc $\alpha = (u, u') \in \mathcal{A}_X$ with source node $u \in L_q$ has always a target node $u' \in L_{q'}$ in a successive layer, i.e., $q' > q$. Each path from $\mathbf{r}$ to a node of the last layer represents the feasible assignment of items from X to a shelf as follows. Each layer L_q, $0 \leq q < |X|$ is associated with an item $l(q) \in X$. Each node $u \in L_q$ has at most two outgoing arcs. A *0-arc* represents the decision to assign the item to the shelf not and is always associated with an arc value $v_\alpha = 0$. In contrast, a *1-arc* represents the assignment of the item and is associated with value $v_\alpha = 1$. To compile a BDD, we use the top-down construction (TDC) (Bergman et al. 2016) method by

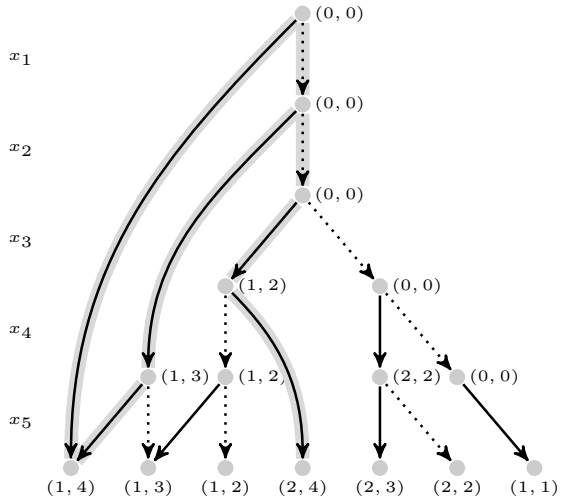


Figure 2: Example of an exact zero-suppressed BDD for item set $X = \{1, 2, 3, 4, 5\}$. The items' dimensions can be taken from Figure 1 with $W = 4$. The numbers on the side of the nodes refer to the nodes' state (h, w). If we assume $H = 4$, a feasible packing is possible with three shelves. The accentuated arcs indicate the corresponding flow.

associating each node $u \in \mathcal{N}_k$ with a state $(h(u), w(u))$ that consists of the current height $h(u)$ and the current with $w(u)$ of the shelf depending on the items that are already assigned along the paths from $\mathbf{r}$ to u. When item $i \in X(k)$ can be feasibly assigned to the shelf, i.e., $w(u) + w_i \leq W$, than a successor state from state $(h(u), w(u))$ can be computed by $(\max\{h(u), h_i\}, w(u)+w_i)$. If the item is not assigned, then the successor state is unchanged. Figure 2 shows an example of an exact BDD for five items.

Compilation The TDC compiles the BDD layer-by-layer, starting with the first layer with node $\mathbf{r}$ and state $(0, 0)$. At each layer L_q, $0 \leq q < |X|$ all successor states are created for each node $u \in L_q$ and corresponding nodes u' are inserted into L_{q+1} together with arcs (u, u'). If multiple nodes have the same states then those nodes are merged into one node s.t. all incoming arcs are redirected to this newly merged node. Then layer L_{q+1} is processed in the same way. This procedure is repeated until the last node is reached. Note that we do not merge nodes in the last layer to one target node, which is usually done in the literature. Instead, we use the nodes' state information to get the actual height of the shelves. During compilation, we use a *dynamic variable ordering* (Bergman et al. 2012), meaning that we do not assign items to layers in advance but rather decide at each current layer which item of the set of not yet assigned items is assigned to the layer. We always select the item that will produce the fewest successor states. Furthermore, we produce a zero-suppressed BDD (Minato 1993) as follows. During the compilation of the BDD, each time when a node at the current layer will have only an outgoing zero-arc, we do not create a successor node. Instead, we move the node from the current layer to the next layer.

Network-Flow Model To check if the 2DSSP constraint holds for a given set $X \subseteq I$ of items, we compile a BDD $\mathcal{D}_X = (\mathcal{N}_X, \mathcal{A}_X)$ and define a network-flow (NF) model $\mathrm{NF}(\mathcal{D}_X, x)$ with flow variables $f_\alpha \in \mathbb{N}_{\geq 0}$, $\alpha \in \mathcal{A}_X$ and binary variables x_i, $i \in I$ (Castro, Cire, and Beck 2022). The idea is that model $\mathrm{NF}(\mathcal{D}_X, x)$ is feasible if and only if the 2DSSP constraint is fulfilled for all items in $i \in X$ for which $x_i = 1$ holds. The model is as follows.

$$\mathrm{NF}(\mathcal{D}_X, x) = \{f_\alpha \in \mathcal{A}_X : \tag{1a}$$

$$\sum_{\alpha \in \mathcal{A}^{\mathrm{o}}(u)} f_\alpha - \sum_{\alpha \in \mathcal{A}^{\mathrm{i}}(u)} = 1 \quad u \in \mathcal{N}_X \setminus (\{\mathbf{r}\} \cup L_{|X|}) \tag{1b}$$

$$\sum_{\alpha \in \mathcal{A}^{\mathrm{o}}(u), u \in L_q} v_\alpha \, f_\alpha = x_{l(q)} \qquad 0 \leq q < |X| \tag{1c}$$

$$\sum_{\alpha \in A^{\mathrm{i}}(u), u \in L_{|X|}} h(u) \, f_\alpha \leq H\} \tag{1d}$$

The flow conservation constraints (1b) ensure that the incoming flow is always equal to the outgoing flow for each node in the BDD. The terms $\mathcal{A}^{\mathrm{o}}(u)$ and $\mathcal{A}^{\mathrm{i}}(u)$ refer to outgoing and incoming arcs of node u, respectively. Constraints (1c) establish that for each selected item $i \in X$ that should be part of the packing, i.e., $x_i = 1$, there goes exactly one unit of flow over a 1-arc of the corresponding layer. The last Constraints (1d) secure that total shelf height of the selected paths is at most the maximum allowed height H. The network flow model will be used in the MIP formulation in Section 5.1 to model the 2DSSP constraints as well as in the LBBD approach in Section 5.3 to solving packing subproblems. Note that there is also a strong connection between network-flow models based on BDDs and arc-flow models. In fact, the 2DSSP can be also modelled by an arc-flow formulation by adapting the ideas from Brandão and Pedroso (2016). However, preliminary experiments showed that the above model (1a)–(1d) often provides a stronger relaxation than the considered arc-flow formulation.

5 Solution Approaches

In the following sections, we solve the SPBP problem with 2DSSP constraints by using different MIP and CP formulations, which are solved by CPLEX and ILOG CP Optimizer, respectively, and notably using a LBBD approach.

5.1 Mixed Integer Programming Formulation

The proposed MIP model uses a *discrete event formulation* to model the scheduling aspects of the SPBP problem. Hence, for each pair of batches $k_1, k_2 \in \mathcal{B}$ we use binary decision variables $v_{jk_1k_2}$ to indicate if k_2 is directly scheduled after k_1 on machine j (=1) or not (=0). Binary decision variable y_{jk} indicates if batch $k \in \mathcal{B}$ is assigned to machine $j \in M$ (=1) or not (=0). To express if item $i \in X(k)$ is assigned to batch $k \in \mathcal{B}$, binary decision variables x_{ki} are used. Finally, binary decision variables z_k indicate if batch $k \in \mathcal{B}$ is used (=1) or not (=0). The first model term,

$$\min \theta + \sum_{k \in \mathcal{B}} z_k, \tag{2a}$$

minimises the makespan θ and the number of used batches. The packing aspects are modelled by

$$\sum_{k \in \mathcal{B} | i \in X(k)} x_{ki} = 1 \qquad i \in I \tag{2b}$$

$$x_{ki} \leq z_k \qquad k \in \mathcal{B}, i \in X(k) \tag{2c}$$

$$z_k \leq \sum_{i \in R(r(k))} x_{ki} \qquad k \in \mathcal{B} \tag{2d}$$

$$z_k \leq \sum_{i \in D(d(k))} x_{ki} \qquad k \in \mathcal{B} \tag{2e}$$

$$\mathrm{NF}(\mathcal{D}_{X(k)}, \{x_{ki} \mid i \in X(k)\}) \qquad k \in \mathcal{B} \tag{2f}$$

where Constraints (2b) ensure that each item is assigned to exactly one batch and Constraints (2c) provide that if an item is assigned to batch k then k must also be used. Note that the model uses the time window decomposition, i.e. each batch $k \in \mathcal{B}$ has a predefined time window $[r(k), d(k)]$ and a predefined family $a(k)$ s.t. only items from set $X(k) \subseteq I$ can be feasibly assigned to k. Constraints (2d) and (2e) establish that for each used batch k the intersection of the assigned items' time windows results in the batch's time window $[r(k), d(k)]$. The last Constraints (2f) cover the 2DSSP constraints by creating for each batch k a BDD and adding the network flow constraints (1a)–(1d) to the MIP model. Note that due to the time window decomposition, all items in $X(k)$ have the same family and their time windows overlap. Constraints

$$\sum_{j \in M} y_{jk} = z_k \qquad k \in \hat{\mathcal{B}} \tag{2g}$$

$$\sum_{k_2 \in \mathcal{B}} v_{jk_1k_2} = y_{jk_1} \qquad k_1 \in \hat{\mathcal{B}}, j \in M \tag{2h}$$

$$\sum_{k_1 \in \mathcal{B}} v_{jk_1k_2} = y_{jk_2} \qquad k_2 \in \hat{\mathcal{B}}, j \in M \tag{2i}$$

cover the scheduling part, where batch $\hat{k} \in \hat{\mathcal{B}} = \mathcal{B} \cup \{\hat{k}\}$ represents a dummy batch indicating the first and last assigned batch to a machine. Constraints (2g) ensure that every used batch is assigned to exactly one machine and Constraints (2h) and (2i) guarantee that every assigned batch has a direct successor and direct predecessor, respectively. The big-M constraints

$$C_{k_2} - C_{k_1} + V(1 - v_{jk_1k_2}) \geq \delta_{a(k_1)a(k_2)} + p_{a(k_2)} \tag{2j}$$

for each pair of batches $k_1, k_2 \in \hat{\mathcal{B}}$, $k_1 \neq k_2$, and each machine $j \in M$, take the sequence dependent setup times into account with constant $V = \delta_{a(k_1)a(k_2)} + d(k1) + p_{a(k1)} - r(k2)$ and completion times $C_k \in \mathbb{Z}_{\geq 0}$ for each batch k. The final set of constraints

$$z_{\hat{k}} = 1 \tag{2k}$$

$$C_{\hat{k}} = 0 \tag{2l}$$

$$r(k) \leq C_k - p_{a(k)} \leq d(k) \qquad k \in \mathcal{B} \tag{2m}$$

$$C_k \leq \theta \tag{2n}$$

ensure that each batch starts within its time window and compute the makespan θ.

5.2 Constraint Programming Formulation

The proposed CP formulation for the SPBP problem models the 2DSSP constraints by explicitly assigning items to

shelves and shelves to batches. It does not use the time window decomposition described in Section 3 since the following model outperformed the time window decomposition approach in preliminary tests. Since items are batched together if they belong to the same family, we assume that there are for each family type $a \in A$ at most $|I_a|$ shelves and batches. Let $S_a = B_a = \{1, 2, \ldots, |I_a|\}$ be the index sets for shelves and batches, respectively. Integer decision variables $x^{\text{it}}_{ai} \in S_a$ represent the assignment of item $i \in I_a$, $a \in A$ to a shelf from S_a. In the same manner, integer decision variables $x^{\text{sh}}_{as} \in B_a$ represent the assignment of shelf $s \in S_a$ to a batch from B_a. The scheduling aspects are modelled by optional interval variables b_{ak} and y_{jak} for $j \in M$, $k \in B_a$, and $a \in A$, where b_{ak} is present in the model if and only if batch k of family type a is used and y_{jak} is present if and only if batch k is scheduled on machine j. The CP model is as follows. The term

$$\min \max_{k \in B_a, a \in A} \text{end}(b_{ak}) + \sum_{a \in A} \text{cntDiff}(\{x^{\text{sh}}_{ai} \mid i \in I_a\}) \quad (3a)$$

minimises the completion of the last scheduled batch, i.e. the makespan and the number of used batches, by using the ILOG global constraint IloCountDifferent. The constraints

$$\text{pack}(W, \{x^{\text{it}}_{ai} \mid i \in I_a\}, \{w_i \mid i \in I_a\}), \quad a \in A \quad (3b)$$

use the global constraint IloPack and assign each item to a shelf so that the assigned items do not exceed the width W. Unfortunately, we can not use the global constraint IloPack to assign shelves to batches since the constraint expects to know the height of the shelves as a constant in advance. However, this is not the case since the height h^{sh}_{as} of a shelf $s \in S_a$ depends on the assigned items computed by constraints

$$h_i \leq h^{\text{sh}}_{a,x^{\text{it}}_{ai}}, \quad i \in I_a, a \in A \quad (3c)$$

using the IloElement constraint to express variable x^{it}_{ai} in the index of h^{sh}_{as}. Constraints

$$\sum_{s \in S_a} h^{\text{sh}}_{as}(x^{\text{sh}}_{as} = b) \leq H, \quad b \in B_a, a \in A \quad (3d)$$

ensure that the total height of batch b of type a does not exceed H by adding up the heights of each assigned shelf. Thereby, the expression $(x^{\text{sh}}_{as} = b)$ is interpreted as one if it is true and zero otherwise. To ensure that only items with overlapping time windows are assigned to the same batch constraints

$$r_i \leq r^{\text{sh}}_{a,x^{\text{it}}_{ai}} \leq d^{\text{sh}}_{a,x^{\text{it}}_{ai}} \leq d_i \qquad i \in I_a, a \in A \quad (3e)$$

$$r^{\text{sh}}_{as} \leq r^{\text{b}}_{a,x^{\text{sh}}_{as}} \leq d^{\text{b}}_{a,x^{\text{sh}}_{as}} \leq d^{\text{sh}}_{s} \qquad s \in S_a, a \in A \quad (3f)$$

are added to the model, using the integer decision variables r^{sh}_{as}, r^{b}_{ak} and d^{sh}_{as}, d^{b}_{ak} representing the release time and deadline of shelves and batches, respectively. Constraints

$$\max(\{x^{\text{sh}}_{ai} \mid i \in I_a\}) < \text{cntDiff}(\{x^{\text{sh}}_{ai} \mid i \in I_a\}) \quad (3g)$$

$$(s >= \max(\{x^{\text{it}}_{ai} \mid i \in I_a\})) \rightarrow (x^{\text{sh}}_{as} = 0),\ s \in S_a \quad (3h)$$

for each $a \in A$ break some symmetries by ensuring that the index of used batches is consecutive and an unused shelf is always assigned to batch 0. Finally, interval variables are connected by

$$(k \leq \max(\{x^{\text{sh}}_{as} \mid s \in S_a\})) \leftrightarrow \text{Pre}(b_{ak}) \quad (3i)$$

$$\text{Pre}(b_{ak}) \rightarrow (r^{\text{b}}_k \leq \text{start}(b_{ak}) \leq d^{\text{b}}_k) \quad (3j)$$

for $k \in B_a, a \in A$, where the first set of constraints enforce that the interval variable b_{ak} is present in the model if and only if a shelf is assigned the batch $k \in B_a$. The second set of constraint enforces that the start time of the batch is between the batch's release time and deadline. Note that Pre refers to the ILOG constraint IloPresenceOf. The constraints

$$\text{alternative}(b_{ak}, \{y_{jak} \mid j \in M\}), \quad k \in B_a, a \in A \quad (3k)$$

$$\text{noOverlap}(\{y_{jak} \mid k \in B_a, a \in A\}, \delta), \quad j \in M \quad (3l)$$

schedules a batch to a specific machine and ensure that batches assigned to the same machine do not overlap in time. Finally constraints

$$\text{Pre}(y_{jk}) \rightarrow \bigvee_{p \in B_a : p \geq k} \text{Pre}(y_{j-1p}) \quad (3m)$$

for all $k \in B_a$, $a \in A$, $j \in M \setminus \{0\}$ break symmetries by enforcing that if batch k is scheduled at machine j then there must be a batch assigned to machine $j-1$ with an higher index than k.

5.3 Logic-Based Benders Decomposition

The SPBP problem naturally decomposes into a two-dimensional packing problem and a parallel machine scheduling problem. Hence, applying a decomposition approach to solve the SPBP problem seems promising. The *logic-based benders decomposition* (LBBD) approach (Hooker et al. 1995), a generalisation of the classical Benders decomposition (Benders 1962), has been applied to a wide range of discrete optimisation problems (Hooker 2019). The idea is to decompose the problem into a *master problem* (MP) and one or more *subproblems*. At each iteration, the MP fixes a subset of the decision variables. Afterwards, the subproblems are solved by fixing the remaining variables subject to the already fixed decision variables. The solutions of the subproblems form a feasible solution to the optimisation problem. From this solution, logic-based Benders cuts are deduced and added to the MP s.t. the MP chooses a different configuration in subsequent iterations. Since the MP considers only a subset of variables, the MP is a relaxation of the problem. Its solution provides a lower bound that grows non-decreasingly with further iterations. As soon as the objective value of the solutions of the subproblems is equal to the lower bound provided by the MP, the LBBD approach proves optimality and terminates.

For the SPBP, the MP assigns items to batches. Our approach uses two kinds of subproblems. The first kind, a feasibility problem, checks for each batch if the assigned items can be feasibly packed according to the 2DSSP constraints. If this is not the case, then a corresponding logic-based Benders cut is added that forbids the item assignment, and the MP is solved again. If every item assignment is feasible, the second kind of subproblem is solved. This subproblem is a scheduling problem that assigns each batch to a machine and minimises the makespan. After the scheduling subproblem is solved, a corresponding cut is added to the MP. This procedure is repeated until the time limit is succeeded or optimality proven. Figure 3 illustrates the described LBBD approach.

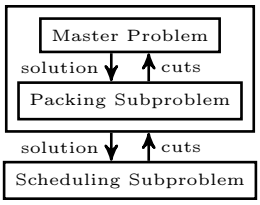


Figure 3: Overview - Logic Based Benders Decomposition

Master Problem The MP assigns items to batches using the time window decomposition described in Section 3. The MIP model

$$\min \quad \theta + \sum_{k \in \mathcal{B}} z_k \tag{4a}$$

$$\text{Constraints (2c)–(2e)} \tag{4b}$$

$$\sum_{i \in X(k)} x_{ki}\, w_i\, h_i \leq z_k\, W\, H \qquad k \in \mathcal{B} \tag{4c}$$

$$z_k\, (r(k) + p_{a(k)}) \leq \theta \qquad k \in \mathcal{B} \tag{4d}$$

$$\text{Packing Benders Cuts} \tag{4e}$$

$$\text{Scheduling Benders Cuts} \tag{4f}$$

assigns values to binary decision variables x_{ki} for each batch $k \in \mathcal{B}$ and item $i \in X(k)$ to indicate if i is assigned to k (=1) or not (=0) while minimising the makespan θ and the number of used batches. Binary variables z_k indicate if batch k is used (=1) or not (=0). Constraints (4b) secure that each item is assigned to exactly one batch and if a batch is used then there must be items assigned to it s.t. the items' time window results in the batch's time window. Constraints (4c) are a relaxation of the 2DSSP constraints by saying that the total area of assigned items must be smaller than the area of the batch. Constraints (4d) estimate the makespan by making use of the fact that if a batch is used than the makespan must be at least the release time of the batch plus the corresponding processing time.

Packing Subproblem Let $\bar{\mathcal{B}} \subseteq \mathcal{B}$ be the set of used batches at the current iteration of the LBBD approach after the MP was solved. To check for each batch k, if the item assignment $I(k)$ is feasible and fulfils the 2DSSP constraints we create a BDD $\mathcal{D}_{I(k)}$ and solve the network-flow model

$$\min \sum_{\alpha \in \mathcal{A}^{\mathrm{i}}(u),\, u \in L_{|I|}} h(u)\, f_\alpha \tag{5a}$$

$$\mathrm{NF}(\mathcal{D}_{I(k)}, \{\bar{x}_{ki} \mid i \in I(k)\}) \tag{5b}$$

$$\bar{x}_{ki} = 1 \qquad i \in I(k), \tag{5c}$$

minimising the total shelf height with binary decision variables $\bar{x}_{ki}$ to indicate if item i is used (=1) or not (=0). The usefulness of this variables will be explained in the next paragraph. Since we only want to check if model (5a)–(5c) is feasible, we terminate CPLEX as soon as a feasible solution (a witness for feasibility) is found. If this is the case then we proceed with the next batch in $\bar{\mathcal{B}}$. Otherwise, if no feasible solution exits, we add feasibility cuts

$$\sum_{i \in I(k)} x_{k'i} \leq |I(k)| - 1, \quad k' \in \mathcal{B} : I(k) \subseteq X(k') \tag{6}$$

to the maser problem in order to forbid item assignment $I(k)$ in subsequent iterations.

To strengthen the cuts (6), we are interested in finding the core conflict set of items responsible for violating the 2DSSP constraints. Hence, we want to find a smaller set $I'(k) \subseteq I(k)$ s.t. model (5a)–(5c) is still infeasible. Here come variables $\bar{x}_{ki}$ into play. To this end, we sort items in $I(k)$ according to their area in non-decreasing order, assuming that larger items are much more responsible for infeasibility. Then we remove the first item i' according to this sorted sequence and solve the model (5a)–(5c) again using the *same BDD* by setting the corresponding $\bar{x}_{ki'}$ to zero. If the model is still infeasible then we proceed with the second item of the sorted sequence. This step is repeated until the model becomes the first time feasible. Then the remaining items, together with the last item i' under consideration, become set $I'(k)$. The procedure can be speed up by using binary search.

If at least one batch violates the 2DSSP constraints, the MP is solved again with the newly added feasibility cuts (6).

Since the network-flow model provides a strong linear relaxation, we also add stronger cuts by creating a BDD $\mathcal{D}_{X(k)}$ for each batch k where $I'(k) \subseteq X(k)$ holds and adding the corresponding network-flow model as a linear relaxation to the MP. We will discuss the impact of this in Section 6.

Scheduling Subproblem Suppose all batches $\bar{\mathcal{B}}$ have a feasible item assignment. In that case, the scheduling subproblem is solved by assigning each batch to a machine and finding a possible start time to minimise the makespan. This is done by a CP model that uses interval variables b_k with processing time $p_{a(k)}$ to represent batch $k \in \bar{\mathcal{B}}$ and optional interval variables y_{jk} to represent the assignment of k to machine $j \in M$. The model

$$\min \max_{k \in \bar{\mathcal{B}}} \mathrm{end}(b_k) \tag{7a}$$

$$r(k) \leq \mathrm{start}(b_k) \leq d(k) \qquad k \in \bar{\mathcal{B}} \tag{7b}$$

$$\mathrm{alternative}(b_k, \{y_{jk} \mid j \in M\}) \qquad k \in \bar{\mathcal{B}} \tag{7c}$$

$$\mathrm{noOverlap}(\{y_{jk} \mid k \in \bar{\mathcal{B}}\}, \delta) \qquad j \in M \tag{7d}$$

minimises the makespan subject to Constraints (7b) which ensure that each batch starts within its time window. Constraints (7c) use the global ILOG constraint IloAlternative which secure that each batch is assigned to exactly one machine. Constraints (7d) use the global constraint IloNoOverlap and establish that batches that are assigned on the same machine do not overlap by taking the setup times δ into account. In addition, we add similar symmetry-breaking constraints (3m) as in Section 5.2 for the CP model.

If the CP solver finds an optimal solution to model (7a)–(7d) with makespan θ^* then the logic-based Benders cut

$$\theta \geq \theta^*\, (1 - \sum_{k \in \bar{\mathcal{B}}} (1 - z_k)) \tag{8}$$

is added to the MP to ensure that if the batches $\bar{\mathcal{B}}$ are selected again then the makespan is at least θ^*. Otherwise, if the model is infeasible then we add the constraints

$$\sum_{k \in \bar{\mathcal{B}}} z_k \leq |\bar{\mathcal{B}}| - 1 \tag{9}$$

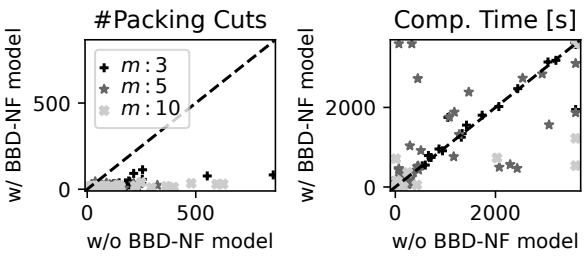


Figure 4: Comparing the impact of adding linear relaxations of BBD-based network-flow models to the master problem.

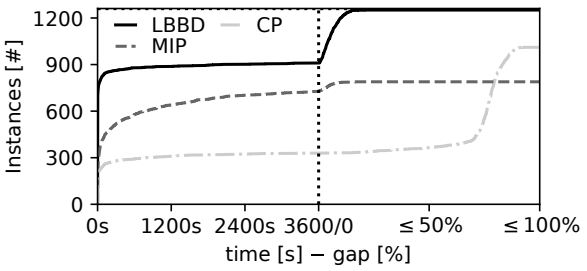


Figure 5: Performance plot of results obtained from LBBD, MIP, and CP over all 1260 instances.

to the master model, forbidding the selection of set $\bar{\mathcal{B}}$ in subsequent iterations. As for the packing subproblem, we try to strengthen Constraints (8) and (9) by reducing the size of $\bar{\mathcal{B}}$. To this end, we sort the batches according to the start times in non-decreasing order in case an optimal feasible solution could be found. If the model is infeasible then the batches are sorted according to how many times a batch overlaps with other batches in non-decreasing order.

6 Computational Results

All proposed 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 MIP and CP models were solved with ILOG CPLEX 20.1 and CP Optimizer 20.1, respectively, both using default settings. The time limit in all experiments is set to one hour.

We created a random benchmark instance set. Each instance class of the benchmark set consists of 30 instances for each combination of $n \in \{20, 40, \ldots, 140\}$ items, $m \in \{5, 10\}$ machines, and $o \in \{5, 10\}$. In total we created 1260 instances, available at https://doi.org/10.4121/21641630.

Our experiments aim to solve as many instances as possible to proven optimality. Thereby we compare the LBBD approach from Section 5.3 with the MIP approach and the CP approach from Sections 5.1 and 5.2, respectively.

Our first experiments show the impact of the LBBD approach if a *linear relaxation of the NF* (LRNF) model (1a)–(1d) based on a created BDD is added to the MP as discussed in Section 5.3. The left scatter plot in Figure 4 shows the number of added feasibility cuts (6) to the MP when the LRNF model is added and when not, while the right plot shows the total computation time of LBBD. Clearly, adding the LRNF to the MP reduces the number of required cuts (6) to get a feasible solution regarding 2DSSP constraints. However, this only significantly impacts the total computation times for some instances. This is because adding the LRNF leads to longer solving times of the MP which does not pay off in each case. Nevertheless, by adding the LRNF to the MP, we can solve three more instances optimally and find a feasible solution for five more instances. Therefore, we stick to this option in the following experiments.

Figure 5 provides a performance plot over all instances. The plot is divided into two parts; The left part reveals the number of instances that could be solved within a certain number of seconds. If instances cannot be solved to optimality within one hour, the right part provides information about the remaining optimality gap for the number of instances. The optimality gaps are computed by $100\% \cdot (\mathrm{OBJ} - \mathrm{LB})/\mathrm{LB}$ where OBJ is the obtained objective value and LB is the lower bound obtained from the considered approaches. The LBBD is able to solve 909 instances to optimality and finds for 9 instances no feasible solution within one hour. The highest remaining optimality gap is 20.8%. CPLEX can solve 729 instances optimally but cannot find a solution for 471 instances at all. Finally, the CP approach finds a feasible solution for 1011 instances but can only prove optimality for 330 instances.

Table 1 presents the main aggregated results obtained from LBBD, MIP, and CP. Columns %-gap state the average optimality gap of final solutions and columns %-opt provides the percentage of optimally solved instances. Columns %-fail list the number of instances for which no feasible solution could be found within the time limit, whereas columns t list the average computation times. If no feasible solution can be found then the time limit is used to compute the average computation time. Column cuts list the average number of cuts applied to the MP. Generally, if the number of machines decreases, finding an optimal or even feasible solution is more difficult. This is explained by the fact that more machines offer more options to schedule items/batches. In all considered cases, the LBBD approach is able to prove more or the same number of instances to optimality than the other approaches in a shorter time. The largest average remaining optimality gap of 9.14% is obtained for instances with 80 items, three machines and ten family types. For this instance class the MIP approach can find just one feasible (and optimal) solution within the time limit and the CP approach provides for 22 instances a feasible solution with an average optimality gap of 80.77%. The MIP approach can solve more instances to optimality than the CP approach, whereas the CP approach finds overall more feasible solutions. On average, LBBD spends 5.9% of the time solving the MP and 94.1% solving the subproblems.

7 Conclusions

We considered the problem of grouping a set of rectangular items into batches and scheduling them on a set of machines s.t. 2DSSP packing constraints, starting time windows, and incompatible family types must be considered. The objective is to minimise the number of used batches and the over-

			LBBD					MIP				CP			
o	m	n	%-gap	%-opt	%-fail	cuts	t [s]	%-gap	%-opt	%-fail	t [s]	%-gap	%-opt	%-fail	t [s]
5	3	20	0.00	**100**	0	4.3	2	0.00	**100**	0	2	6.04	87	0	487
		40	1.06	**83**	0	339.9	681	0.93	73	7	1189	45.93	37	3	2391
		60	2.53	**57**	0	726.4	1679	1.47	47	40	2077	76.57	3	30	3572
		80	3.28	**47**	0	691.3	1933	2.11	30	50	2566	84.36	0	53	3600
		100	2.67	**37**	3	648.1	2351	1.34	17	70	3067	86.66	0	77	3600
		120	2.07	**40**	0	721.1	2209	0.92	23	63	2886	88.26	0	83	3600
		140	3.19	**20**	3	576.7	2886	0.00	3	97	3502	88.58	0	80	3600
	5	20	0.00	**100**	0	1.1	<1	0.00	**100**	0	<1	0.00	**100**	0	1
		40	0.00	**100**	0	2.0	<1	0.00	**100**	0	6	4.94	93	0	604
		60	0.32	**90**	0	208.3	381	0.32	**90**	0	434	70.68	7	0	3434
		80	0.02	**97**	0	31.1	189	0.12	90	7	619	77.58	0	3	3600
		100	0.87	**83**	0	157.0	762	0.78	60	27	1887	79.32	0	30	3600
		120	0.29	**87**	0	156.8	519	1.23	57	33	2492	81.72	0	30	3600
		140	0.20	**87**	3	68.4	624	0.00	20	80	3323	83.37	0	60	3600
	10	20	0.00	**100**	0	1.0	<1	0.00	**100**	0	1	0.00	**100**	0	1
		40	0.00	**100**	0	1.5	<1	0.00	**100**	0	15	17.45	77	0	1144
		60	0.00	**100**	0	1.9	<1	0.00	**100**	0	84	75.66	0	0	3600
		80	0.00	**100**	0	3.9	<1	0.00	**100**	0	403	78.18	0	0	3600
		100	0.00	**100**	0	6.0	<1	0.00	**100**	0	1204	78.22	0	0	3600
		120	0.00	**100**	0	9.5	2	0.04	53	43	2970	78.97	0	0	3600
		140	0.00	**100**	0	18.4	50	0.00	7	93	3568	80.01	0	0	3600
10	3	20	0.00	**100**	0	9.6	38	0.00	**100**	0	91	0.93	93	0	304
		40	5.12	**43**	0	296.3	2299	3.53	20	53	3040	37.66	17	0	3104
		60	5.91	**20**	0	338.4	3070	3.00	13	73	3194	54.85	13	17	3240
		80	9.14	**10**	0	264.4	3347	0.00	3	97	3501	80.77	0	27	3600
		100	6.88	**7**	0	245.8	3393	-	0	100	3600	84.60	0	23	3600
		120	7.15	**0**	3	263.8	3600	-	**0**	100	3600	87.51	**0**	60	3600
		140	6.09	**0**	3	256.9	3600	-	**0**	100	3600	89.70	**0**	83	3600
	5	20	0.00	**100**	0	1.0	<1	0.00	**100**	0	<1	0.00	**100**	0	<1
		40	0.00	**100**	0	7.0	5	0.27	93	3	378	6.68	83	0	618
		60	1.44	**63**	0	129.7	1337	0.77	50	30	1997	48.61	23	3	3017
		80	2.20	**43**	0	220.5	2208	1.28	17	77	3060	73.29	0	7	3600
		100	2.97	**47**	3	172.5	2089	0.00	3	97	3521	78.25	0	33	3600
		120	1.71	**43**	10	76.0	2176	0.00	13	87	3354	81.02	0	67	3600
		140	2.48	**30**	0	112.8	2726	3.72	0	97	3600	83.06	0	60	3600
	10	20	0.00	**100**	0	1.0	<1	0.00	**100**	0	<1	0.00	**100**	0	<1
		40	0.00	**100**	0	1.1	<1	0.00	**100**	0	6	0.00	**100**	0	54
		60	0.00	**100**	0	1.2	<1	0.00	**100**	0	30	21.79	60	0	2030
		80	0.00	**100**	0	1.7	<1	0.00	**100**	0	121	64.48	7	0	3445
		100	0.00	**100**	0	2.4	<1	0.00	**100**	0	424	71.99	0	0	3600
		120	0.00	**100**	0	4.6	47	0.54	93	3	1245	75.09	0	0	3600
		140	0.06	**97**	0	10.8	168	0.35	53	43	2585	75.66	0	0	3600

Table 1: Aggregated main results of LBBD, MIP, and CP.

all makespan. The problem arises in a more complex version in the industrial tool coating field. We devised an LBBD approach that decomposes the problem into a MP, a packing subproblem, and a scheduling subproblem. The approach is compared with a MIP and CP approach. Experimental results show that the LBBD approach can solve more instances to proven optimality in a shorter time than the other two approaches. Further, we modelled the 2DSSP constraints by a BDD-based NF formulation that is solved to check if 2DSSP constraints were violated. We investigated the option to add a linear relaxation of this model to the MP each time a 2DSSP constraint is violated. This strategy could significantly reduce the number of added feasibility cuts. However, on average, the total computation time could not be reduced due to the longer solving time of the MP.

Nevertheless, it seems promising to investigate this approach further by considering different strategies to incorporate BDDs into the MP, e.g. NF models based on relaxed BDDs that may result in smaller models. Further research may consider more complex constraints or aspects relevant to the industrial tool coating field, i.e., machines/batches with different dimensions.

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

Andersen, H. R.; Hadzic, T.; Hooker, J. N.; and Tiedemann, P. 2007. A constraint store based on multivalued decision diagrams. In *Principles and Practice of Constraint Programming, CP 2007*, volume 4741 of *LNCS*, 118–132. Springer.

Benders, J. 1962. Partitioning procedures for solving mixed-variables programming problems. *Numerische Mathematik*, 4(1): 238–252.

Bergman, D.; Cire, A. A.; van Hoeve, W.-J.; and Hooker, J. N. 2012. Variable Ordering for the Application of BDDs to the Maximum Independent Set Problem. In Beldiceanu, N.; Jussien, N.; and Pinson, É., eds., *Proceedings of the Ninth International Conference on Integration of Artificial Intelligence (AI) and Operations Research (OR) Techniques in Contraint Programming for Combinatorial Optimzation Problems, CPAIOR 2012*, volume 7298 of *LNCS*, 34–49. Springer.

Bergman, D.; Cire, A. A.; van Hoeve, W.-J.; and Hooker, J. N. 2016. *Decision Diagrams for Optimization*. Artificial Intelligence: Foundations, Theory, and Algorithms. Springer.

Bergman, D.; Cire, A. A.; von Hoeve, W.-J.; and Hooker, J. N. 2014. Optimization Bounds from Binary Decision Diagrams. *INFORMS Journal on Computing*, 26(2): 253–268.

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.

Castro, M. P.; Cire, A. A.; and Beck, J. C. 2022. Decision Diagrams for Discrete Optimization: A Survey of Recent Advances. *INFORMS Journal on Computing*, 34(4): 2271–2295.

Cire, A. A.; and van Hoeve, W.-J. 2013. Multivalued Decision Diagrams for Sequencing Problems. *Operations Research*, 61(6): 1411–1428.

Fowler, J. W.; and Mönch, L. 2021. A survey of scheduling with parallel batch (p-batch) processing. *European Journal of Operational Research*, 298(1): 1–24.

Hooker, J.; Yan, H.; Saraswat, V.; and Hentenryck, P. 1995. Verifying logic circuits by Benders decomposition. *Principles and Practice of Constraint Programming: The Newport Papers, MIT Press, Cambridge, MA*, 267–288.

Hooker, J. N. 2019. Logic-based benders decomposition for large-scale optimization. In *Large Scale Optimization in Supply Chains and Smart Manufacturing: Theory and Applications*, volume 149, 1–26. Springer.

Kinable, J.; Cire, A. A.; and van Hoeve, W. J. 2017. Hybrid Optimization Methods for Time-Dependent Sequencing Problems. *European Journal of Operational Research*, 259(3): 887–897.

Li, X.; and Zhang, K. 2018. Single batch processing machine scheduling with two-dimensional bin packing constraints. *International Journal of Production Economics*, 196: 113–121.

Minato, S. 1993. Zero-Suppressed BDDs for Set Manipulation in Combinatorial Problems. In *Proceedings of the 30th International Design Automation Conference, DAC'93*, 272–277. Association for Computing Machinery (ACM).

Mönch, L.; Fowler, J. W.; Dauzère-Pérès, S.; Mason, S. J.; and Rose, O. 2011. A survey of problems, solution techniques, and future challenges in scheduling semiconductor manufacturing operations. *Journal of scheduling*, 14(6): 583–599.

Queiroga, E.; Pinheiro, R. G. S.; Christ, Q.; Subramanian, A.; and Pessoa, A. A. 2021. Iterated local search for single machine total weighted tardiness batch scheduling. *Journal of Heuristics*, 27: 353–438.

Shahnaghi, K.; Shahmoradi-Moghadam, H.; Noroozi, A.; and Mokhtari, H. 2016a. A robust modelling and optimisation framework for a batch processing flow shop production system in the presence of uncertainties. *International Journal of Computer Integrated Manufacturing*, 29(1): 92–106.

Shahnaghi, K.; Shahmoradi-Moghaddam, H.; Akbari, K.; Sadjadi, S. J.; and Heydari, M. 2016b. A scenario-based robust optimization approach for batch processing scheduling. In *Proceedings of the Institution of Mechanical Engineers, Part B: Journal of Engineering Manufacture*, volume 230, 2286–2295.

Tang, T. Y.; and Beck, J. C. 2020. CP and hybrid models for two-stage batching and scheduling. In *Proceedings of the 17th International Conference on Integration of Constraint Programming, Artificial Intelligence, and Operations Research (CPAIOR 2020)*, volume 12296 of *LNCS*, 431–446. Springer.

Trindade, R. S.; de Araújo, O. C. B.; and Fampa, M. 2021. Arc-flow approach for single batch-processing machine scheduling. *Computers & Operations Research*, 134: 105394.

Wang, J.; and Luh, P. 1997. Scheduling Job Shops with Batch Machines Using the Lagrangian Relaxation Technique. *European Journal of Control*, 3(4): 268–279.

İbrahim Muter. 2020. Exact algorithms to minimize makespan on single and parallel batch processing machines. *European Journal of Operational Research*, 285(2): 470–483.