Journal Pre-proof

Just-in-Time Two-Dimensional Bin Packing

Sergey Polyakovskiy, Rym M'Hallah

PII: S0305-0483(20)30665-4
DOI: https://doi.org/10.1016/j.omega.2020.102311
Reference: OME 102311

To appear in: *Omega*

Received date: 29 June 2019
Accepted date: 14 July 2020

Please cite this article as: Sergey Polyakovskiy, Rym M'Hallah, Just-in-Time Two-Dimensional Bin Packing, *Omega* (2020), doi: https://doi.org/10.1016/j.omega.2020.102311

Highlights

- We combine bin packing and just-in-time batch scheduling into a new problem
- We minimize total weighted earliness tardiness in a combined packing scheduling problem
- Our constraint program addresses the packing and scheduling aspects simultaneously
- Our branch-and-check approaches solve the problem under few heuristic assumptions
- Performance comparison of constraint versus mixed-integer programming master problems

Just-in-Time Two-Dimensional Bin Packing

Sergey Polyakovskiy[a], Rym M'Hallah[b,*]

[a]*School of Information Technology, Deakin University, Geelong 3216, Australia*
[b]*Department of Statistics and Operations Research, College of Science, Kuwait University, P.O. Box 5969, Safat 13060, Kuwait.*

Abstract

This paper considers the on-time guillotine cutting of small rectangular items from large rectangular bins. Items assigned to a bin define the bins' processing time. Consequently, an item inherits the completion time of its assigned bin. Any deviation of an item's completion time from its due date causes either earliness or tardiness penalties. This just-in-time two-dimensional bin packing problem (JITBP) combines two difficult discrete optimization problems: Bin packing and total weighted earliness tardiness single machine scheduling. It is herein modeled as an integrated constraint program, augmented with two sets of logically redundant constraints that speed the search. The first set uses the concept of dual feasible functions. It focuses on bin packing feasibility. The second is the result of a linear program that schedules filled bins on a single machine. As an alternative to this integrated model, this paper proposes two decomposition cut-and-check approaches that define the master problem (MP) as a relaxation of JITBP where the items are reduced to dimensionless entities. They then reestablish the geometric feasibility of the MPs' solutions by iteratively augmenting MP with Benders cuts generated from the subproblems. The two approaches are similar in concept except that one implements MP as a constraint program (CP) while the second implements it as a mixed-integer program (MIP). Because JITBP is computationally challenging, we test all approaches under a number of heuristic assumptions, which include a maximum runtime for the MIP and CP solvers. The results provide computational evidence that the integrated constraint programming approach performs relatively well, and outperforms the decomposition approach whose MP is a CP. However, both approaches are outperformed by the decomposition approach whose MP is a warm-started MIP.

Keywords: Weighted earliness tardiness; Bin packing; Scheduling; Constraint programming; Hybrid approach; Branch-and-Check; Approximate algorithm

*Corresponding author. Tel: +965-669-14150. Fax: +965-248-37332.
Email addresses: sergey.polyakovskiy@deakin.edu.au (Sergey Polyakovskiy), rymmha@yahoo.com (Rym M'Hallah)

1. Introduction

One of the stages of furniture manufacturing involves the cutting of two-dimensional (2D) small items from 2D large rectangular boards, referred hereafter as bins. The small items are then assembled during later stages. Cutting an item earlier than its due date induces unnecessary handling and storage that translate into earliness costs. Similarly, cutting an item after its due date causes the starvation of later stages and a tardy delivery of the final product. This yields a tardiness cost that includes compensations to end customers, wages of unwanted extra work hours, inefficient use of production resources, losses caused by starvation of later stages, extra mailing costs caused by rushed orders that must be handled as a priority, etc. That is, furniture manufacturers face the combined problem of (i) assigning 2D rectangular small items into identical 2D rectangular large bins and (ii) scheduling the on-time guillotine cutting of the bins. Obviously, this combined problem appears in other environments including steel manufacturing [4], ship lock scheduling [45], embarkation of cargos [10], defense, air cargo, and rescue operations.

This paper studies a new JIT 2D bin packing problem when the on-time guillotine cutting of the bins uses a single cutting machine. This problem, labeled JITBP, minimizes the total weighted earliness tardiness (TWET) of the small items while satisfying all standard cutting and scheduling constraints. It plans the cutting of the filled bins such that the cutting schedule minimizes the weighted deviation of the completion time of each item's cutting from the item's distinct due date. A bin's cutting time depends on the bin's content. As such, JITBP is very complex. It combines the difficulty of two $\mathcal{NP}$-hard problems: 2D bin packing problem (2DBPP) [25], and TWET single machine batch scheduling problem [36].

Combined problems consider different aspects of the supply and logistic chain [12]. They are more realistic than their single but inter-related components. They should be addressed simultaneously not only because they account for more constraints, consider the true nature of the variables, and build intricate interdependencies but also and, most importantly, because they enhance the chances of direct real-life implementation of their solutions without any modification and at no supplementary cost. JITBP is no exception. Solving it requires a thorough understanding of cutting, scheduling, and their interaction. Cutting involves non-overlap, containment, assignment, and disjunctive constraints. Scheduling involves non-overlap, non-preemption, and sequencing constraints. Optimizing material utilization and TWET can be either competing or cooperating goals depending on practical settings. The simultaneous satisfaction of cutting and scheduling constraints is challenging. In addition, altering the assignment of an item from one bin to another alters the two bins processing times; thus, affects the structure of the entire solution, and most importantly of its cost components. Finally, the search space contains a large number of infeasible and of symmetrical solutions.

This paper models JITBP as a novel integrated constraint program (CP). This model explores the strengths of two customized problem-specific constraints

that speed the reduction of variables' domains through inference mechanisms. The first constraint is linear programming (LP)-based. It determines a minimal cost schedule of a feasible assignment of items to bins. It prunes the search tree rapidly. The second constraint applies the concept of dual feasible functions in search for a feasible packing. It filters out many infeasible items-to-bin assignments as the CP-search progresses.

For large and difficult instances of JITBP, this paper investigates two branch-and-check (B&C) models that are inspired from logic-based Benders decomposition (LBBD). (LBBD techniques are notorious for successfully handling planning and scheduling problems [17].) The two proposed B&C approaches build a master problem (MP) that assimilates items to dimensionless entities, and entrust packing feasibility to the subproblems. That is, MP drops the geometric constraints that describe the relationships among packed items and between a packed item and its assigned bin. It assigns items to bins and searches for the bins' best completion times such that they minimize TWET. The subproblems, which focus on packing feasibility, eliminate infeasible cutting patterns by injecting Benders cuts to MP. Both approaches solve the subproblems via a CP-based packing algorithm. However, the first models MP as a mixed-integer program MIP while the second models MP as a CP. Thus, they explore two promising/interesting B&C decompositions of the integrated model. Because CP and MIP solvers benefit from a warm-start mode, this paper proposes a simple iterative two-phase heuristic for this purpose.

This paper assesses the performance of the proposed approaches. (Even though they can converge to global optima, all three proposed algorithms are run under heuristic assumptions, including a threshold runtime for MIP and CP solvers.) Specifically, this paper evaluates the optimality gap reduction of each approach, and infers the most appropriate way of formulating MP: using CP versus MIP. The computational investigation reveals that the decomposition approach whose MP is an MIP is preferred to the integrated CP based approach. In turn, this latter outperforms the decomposition approach whose MP is a CP.

This manuscript has six major contributions. First, it offers a novel integrated CP model that models an unlimited number of guillotine cuts by creating a constant number of regions. This model is a successful alternative to more traditional MIPs, which either limit the number of guillotine cuts per bin or resort to disjunctive constraints (and their intrinsic computational issues). This model is not intrinsic to the problem at hand. It is easily extendable to other guillotine packing applications. Second, to the best of the authors' knowledge, this manuscript is the first to consider dynamic cutting times for two-dimensional bins, where the cutting time of a bin is a function of its contents. This is a major advancement that distinguishes the paper from the state of the art, which assumes a constant bin's processing time regardless of a bin's content. Third, despite the prevalence of Benders decomposition, both its hybridization within CP and MIP and its application to JITBP are new. Fourth, presenting and comparing the three proposed approaches highlight the difficulties and advantages inherent to B&C for this specific JITBP. Not only does it assess the capability of each approach to solve JITBP, but it also investigates the modelling issues

brought up by decomposition. Fifth, JITBP is of sizeable importance to the industry, logistics, and defence force applications. While the packing component may differ from guillotine packing, minimizing TWET of batched items prevails. Finally, the proposed approaches are modular. They are extendable to any machine environment including parallel machines and (hybrid) flow shops. Changing the shop environment causes only small changes to the CP scheduling function evoked within the MIP and CP models.

Section 2 reviews pertinent literature. Section 3 defines the problem. Section 4 proposes an integrated model to JITBP. Section 5 details the two proposed decomposition based approaches. Section 6 presents the iterative two-phase warm-start heuristic. Section 7 discusses the results. Finally, Section 8 summarizes the paper and gives possible extensions.

2. Literature Review

This section highlights the difficulty of JITBP and of its components. Sections 2.1-2.3 present the recent work on, respectively, TWET single machine scheduling, bin packing (BP), and JITBP related problems.

2.1. Weighted Earliness Tardiness Scheduling

The simplest single machine TWET scheduling problem has unit weights and a common non-restrictive due date d; i.e., d is large enough not to constrain the scheduling process. Its optimal solution is V-shaped. Jobs completing before d follow the longest processing time rule, while jobs starting after d follow the shortest processing time rule. One job ends its processing on d, and another starts processing at d. There is no idle time between jobs. Finally, the starting time of the schedule is not necessarily zero. The weighted version of this problem changes its difficulty level making it $\mathcal{NP}$-hard [20]. Similarly, a restrictive due date alters some of the aforementioned conditions for the unit weight version. A job may be processed through the common due date and idle time may be inserted between successive jobs. When idle time is permitted, the problem reduces to identifying the optimal sequence and to inserting idle time between jobs of the sequence. The general case with distinct due dates has an optimal schedule that may contain idle time between adjacent jobs.

The single machine distinct due dates TWET scheduling problem is strongly $\mathcal{NP}$-hard [46], with few exact techniques [24, 43] and no polynomial-time algorithm with a guaranteed worst-case performance unless $\mathcal{P} = \mathcal{NP}$ [31]. For unit weights, exact approaches vary from dynamic programming [1] to branch-and-bound (B&B)[40]. The problem is generally approached heuristically [29, 30, 47]. Rocholl and Mönch [39] consider a variant that is close to JITBP: a total earliness tardiness single machine batch scheduling with a common non-restrictive due date. They use a random key genetic algorithm.

2.2. On Bin Packing

2DBPP allocates, without overlap, a set of small rectangular items into the minimum number of identical large rectangular bins. The problem is extensively studied. Among its effective exact approaches is a branch-and-price-and-cut algorithm [32]. The approach uses a constraint satisfaction formulation that is more flexible than its integer counterpart. Martello et al. [28] explore the aforementioned constraint satisfaction formulation to check the feasibility of a packing within a single bin. They use the feasibility check to speed up a column generation pricing approach for the three-dimensional case. Cote et al. [11] design a branch-and-cut and a Benders decomposition approach for 2DBPP with unloading constraints. These exact approaches highlight the difficulty of the problem, which is $\mathcal{NP}$-hard in the strong sense. They solve instances with up to 100 items but often fail to solve instances with as few as 20 items. Solving large or difficult instances exactly is time-consuming without any guarantee of a sufficiently good convergence to a global optimum. Approximate approaches to the problem vary from best/first fit algorithms [8] to meta-heuristics [25] to hyper-heuristics [42, 26] to multi-agent based algorithms [37] to a combination of CP and iterative packing heuristics [35].

2.3. On Scheduling and Bin Packing

Solving a combined problem is generally harder than tackling its components separately. Optimizing each component is suboptimal to the combined problem. Thus, an effective approach should optimize both components simultaneously. BP appears in conjunction with (un)loading constraints [44], vehicle routing [19], and JIT single / parallel machine scheduling [34].

Li [23] considers a variation of an industrial 2D cutting stock where meeting due dates is more important than minimizing waste. Hendry et al. [16] explore the same problem but address the two components individually within a two-stage solution procedure. Reinertsen and Vossen [38] investigate a problem that arises in steel manufacturing. They minimize the convex combination of makespan and total weighted tardiness of a one-dimensional cutting stock problem with due dates. They tackle the problem using an integer programming based heuristic. For the same problem, Arbib and Marinelli [4] propose exact approaches, including column generation ones. They apply time-indexed models, which contain a large number of variables. This number is reduced using an ad-hoc procedure. Braga et al. [9] consider the same problem. They review two exact formulations and embed a variant of the arc flow model into a meta-heuristic to approximately solve medium scale instances. Arbib and Marinelli [5] consider the one-dimensional BP where items are assigned to identical unit-length bins with the objective of minimizing the convex combination of makespan and maximum lateness. In this context (and in the previous two references), the makespan is the number of bins, and bins have a constant cutting time (not a necessarily realistic assumption as they point out) and integer due dates that are multiples of the cutting times. The problem is modeled as a time-indexed mixed-integer LP, and is tackled via column generation. It is

decomposed into an MP that schedules filled bins and into pricing problems that fill single bins iteratively. The model is further augmented with bounds on makespan and on maximum lateness.[1]

Bennell et al. [7] extend the above problem to 2D. They use genetic algorithms to tackle the non-oriented 2DBPP with due dates with two objective functions: minimizing the number of used bins and minimizing the maximum lateness of the items. Polyakovskiy and M'Hallah [35] address the same problem, but focus on the second objective. In addition to a tight lower bound on maximum lateness, they propose an MIP that solves small-sized instances exactly. To cope with larger-sized instances, they design a two-stage heuristic, which iteratively improves the incumbent via a series of assignment low-level MIPs guided by feasibility constraints. Their approach embeds a lookahead strategy that guards against infeasible search directions and constrains the search to improving directions only. Marinelli and Pizzuti [27] minimize both the number of bins and the maximum lateness of the 2DBPP with due dates using a sequential value correction heuristic. Arbib et al. [3] exemplify an industrial case study that illustrates a relationship between the two-dimensional packing and the scheduling problem where the objective is to minimize the weighted number of tardy jobs. Focusing on the scheduling aspects, they propose a branch-and-cut algorithm [22] for two scheduling environments: single machine with precedence relations, and identical parallel machines with unit operation times.

A prelude [33] to this paper addresses JITBP via an agent-based constructive heuristic (ABH) and via a CP-based approximate heuristic (CPH). ABH constructs a solution through repeated negotiations between agents representing the items and bins. The agents cooperate in a way that minimizes TWET. CPH adopts the impact-based search strategy that is implemented in the general-purpose solver IBM CP Optimizer. Their results stipulate that CPH is superior to ABH for small-sized instances while the opposite prevails for larger instances.

In this paper, we introduce a warm-start heuristic that consistently outperforms ABH. In addition, we adopt CPH's model as a core of a new integrated CP model and refine it considerably. We augment CPH's model with customized constraints that strengthen domain reduction and inference mechanisms while eliminating some unnecessary constraints. In this way, the new integrated model obtains better solutions than CPH in smaller run times. Subsequently, when initiated with the warm-start solution, the integrated model consistently outperforms both CPH and ABH. Therefore, neither CPH nor ABH are used for comparison purposes. Instead, the efficiency and efficacy of the integrated model are compared to those of the two new LBBD approaches. The comparison uses an existing benchmark set of instances whose due dates are amended to better reflect the challenges imposed by JITBP's components.

[1] For one-dimensional BP [5], it is reasonable in most applications to assume unit bin processing times. Variable processing times are considered in some 2DBPP cases (e.g. [3]) but not in others.

3. Problem Formulation

Consider a manufacturing process, which involves a cutting stage that produces a set $I = \{1, \ldots, n\}$ of n small rectangular items from a set $B = \{1, \ldots, m\}$ of large identical rectangular sheets of raw material, referred to as bins. The cutting stage uses a single machine, which obtains items by a series of edge to edge cuts that are parallel to the edges of the sheets. Item $i \in I$ is characterized by its integer length l_i, integer width w_i, and integer due date d_i that defines when i should ideally be produced.

In a JIT environment, an item's due date is agreed upon between the producer and consumer. (The producer and consumer may be two consecutive stages of the manufacturing chain.) Missing an item's due date may result in the loss of the customer or the need to compensate for the delay along the production or assembly line. On the other hand, cutting an item much earlier than its due date may cause unwanted inventory, handling, and potential damage. Let C_i denote the completion time of item $i \in I$ (i.e., the time its cutting is finished). When cut prior to d_i, item i is early. Similarly, when cut after d_i, item i is tardy. The length item i is tardy or early is important. Thus, any cutting schedule of a set of items should strive to avoid both earliness and tardiness costs. The earliness of item i is $E_i = \max\{0, d_i - C_i\}$ and causes an earliness penalty $\epsilon_i E_i$, where ϵ_i is a per time unit earliness cost of item i. This cost can be perceived as an inventory cost. Similarly, when produced later than d_i, item i has tardiness $T_i = \max\{0, C_i - d_i\}$. Consequently, item i incurs a tardiness cost $\tau_i T_i$, where τ_i is the per-unit tardiness cost of item i. This cost reflects the additional costs incurred by subsequent production stages and corresponding in-house logistics. Albeit determining τ_i and ϵ_i reliably may seem difficult, these values can be determined using cost accounting techniques, generally implemented in all shop floors. Regardless, the scheduling literature assumes that their values are known. In addition, their relative magnitudes are more important than their true values.

A bin $b \in B$ is characterized by its integer length L and integer width W such that $l_i \leq L$ and $w_i \leq W$ for any $i \in I$. Depending on the items' dimensions, it is possible to cut more than one item from a bin. A subset $I_b \subseteq I$ of items assigned to a bin $b \in B$ can not overlap and must be entirely contained in the bin. The subset forms a batch b whose items share its completion time C_b. The processing time of batch b is a function $\varphi(I_b)$ of I_b. Herein, we assume that $\varphi(I_b)$ is dynamic and linear in the number of items. Evidently, an empty bin has a zero cutting time. In this research, $\varphi(I_b)$ depends on the setup and handling time of the items not on their cutting time. The handling time is quite intricate because it involves the items' rotation and correct positioning under the cutter. On the other hand, the cutting can be very fast depending on industrial setting: it may simply consist of lowering a lever. That is, the items' cutting times per say can be negligible for all purposes because of their small magnitude. It follows that $\varphi(I_b)$ is not a linear function of the processing times but of the number of items in a bin.

JITBP searches for (i) a feasible guillotine packing of I into bins of B, and

(ii) a bin cutting schedule that minimizes TWET, defined by $\sum_{i \in I} (\epsilon_i E_i + \tau_i T_i)$. Clearly, $m \leq n$ as n is a valid upper bound on the number of bins. A feasible solution that uses less bins is more efficient from a packing perspective, but may not be as efficient from a scheduling perspective: It may cause a higher TWET.

The packing and scheduling components of this problem are strongly interdependent. They interact with each other either cooperatively or competitively depending on the problem parameters such that the due dates' distribution, setup and handling time. Relaxing the packing density (by assigning a single item per bin) does not necessarily reduce TWET. Similarly, a dense packing that assigns multiple items to a bin may either increase or decrease TWET. Thus, finding the optimal number of bins and their contents is not trivial.

JITBP is an $\mathcal{NP}$-hard combinatorial optimization problem that is computationally challenging. It has a huge solution space due to a large number of alternative solutions caused by symmetric packing configurations and schedules. It uses disjunctive constraints, which determine the completion time of the items. In MIP, each disjunctive constraint is represented via a "Big M" constraint. The "Big M" reformulation has a drawback: Its resulting relaxation is often too weak. It degrades the search and lengthens the partial enumeration [15]. It almost exhaustively enumerates the search space. This major drawback motivates the design of alternative effective solution techniques.

4. An Integrated Constraint Programming Approach

CP is a prominent modeling technique for BP and scheduling alike [18]. We therefore use CP to model JITBP, which combines packing and scheduling. In CP, a problem is modelled via a set $\mathcal{V}$ of variables and via a set of constraints $\mathcal{C}$ on $\mathcal{V}$. Each variable $\nu_i \in \mathcal{V}$ is defined by its domain $D(\nu_i)$. CP enumerates solutions traversing a search tree. At a node of the tree, it restricts the domain $D(\nu_i)$ of every variable $\nu_i \in \mathcal{V}$ subject to $\mathcal{C}$. Each constraint acts as a special-purpose filtering algorithm that excludes, from the domains of the variables, those values that lead to infeasible solutions. When $\nu_i \in \mathcal{V}$ is fixed, the search inspects those constraints that share ν_i. CP propagates the constraints; it checks whether fixing the value of ν_i eliminates values from the domains of other variables that are connected to ν_i via one or more constraints of $\mathcal{C}$. Thus, the results of one filtering procedure are repeatedly propagated to the others until a certain level of consistency is achieved. When it yields an empty $D(\nu_i)$, the filtering signals an infeasible solution. When the domain of ν_i is not empty but is not a singleton, CP branches on ν_i by partitioning $D(\nu_i)$. As the search descends into the tree, CP reduces the variables' domains. It obtains a feasible solution when the domain of every variable $\nu_i \in \mathcal{V}$ reduces to a singleton. When emphasis is on optimality, the search continues until either the optimum is found, or the exploration of the tree is unsuccessful.

Herein, we model JITBP using CP (cf. [21].) Section 4.1 defines the decision variables. Section 4.2 presents the model, which avoids the notorious weakness of MIP and of its "Big M"constraints. Section 4.3 explains the role of the logically

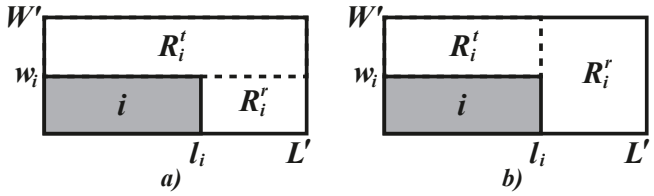

Figure 1: Illustrating the two cutting patterns resulting from a different sequence of cuts. The first cut is horizontal in pattern α (on the left) and vertical in pattern β (on the right).

redundant constraints included in the model. Finally, Section 4.4 details the adopted search strategy.

4.1. Decision and Auxiliary Variables

Our CP model for JITBP explores a new approach to modeling a not-a-priori fixed number of guillotine cuts. Every item $i \in I$ is obtained from a bin (L, W) or from a region (L', W'), $L' \leq L$, $W' \leq W$, of a bin via a sequence of horizontal and vertical cuts.

- When the first cut is horizontal, as depicted in Figure 1.a, the region (L', W') is split into two regions: R_i^t of size $(L', W' - w_i)$ at the top of i, and R_i^r of size $(L' - l_i, w_i)$ at the right of i. In this case, i is said to be the result of an α cutting pattern.
- When the first cut is vertical, as shown in Figure 1.b, the cutting of the bin produces two regions: R_i^t of size $(l_i, W' - w_i)$ at the top of i and R_i^r of size $(L' - l_i, W')$ at the right of i. In this case, i is said to be the result of a β cutting pattern.

Regardless of cut type, cutting item $i \in I$ creates two regions: R_i^t to the top and R_i^r to the right of i. When $L' = l_i$ (resp. $W' = w_i$), R_i^r (resp. R_i^t) is empty. Therefore, the extraction of the n items creates $2n$ regions: $R_1^t, \ldots, R_n^t$ and $R_1^r, \ldots, R_n^r$. Let the initial m unfilled bins correspond to regions $R_1 = \ldots = R_m = (L, W)$, and let $R = \{R_1, \ldots, R_m, R_1^t, \ldots, R_n^t, R_1^r, \ldots, R_n^r\}$ be the set of all regions. At no cutting stage, does R have more than $m + 2n$ regions.

The proposed CP model and search strategy assign the n items to the regions of R, determine each region's size, and schedule the first m regions. For this purpose, they use integer decision variables and auxiliary variables. The **first** set of decision variables assigns items to regions. It is a vector $\mathbf{e} = (e_1, \ldots, e_n)$, where $e_i = j$ if item i is positioned in region $j \in R$. Three scenarios are possible.

- $e_i \in \{1, \ldots, m\}$ when i is the first item packed into bin $j = e_i$.
- $e_i \in \{m + 1, \ldots, m + n\}$ when i is located in region $R_{e_i - m}^t$ on top of item $(e_i - m)$. Because i can't be packed on top of itself (i.e., i can't be in $R_i^t = R_{m+i}$), $e_i \neq m + i$.

- $e_i \in \{m+n+1, \ldots, m+2n\}$ when i is located in region $R^r_{e_i-m-n}$ to the right of item $(e_i - m - n)$. Because i can't be packed to the right of itself (i.e., i can't be in $R^r_i = R_{m+n+i}$), $e_i \neq m+n+i$.

The domain of e_i is therefore

$$D(e_i) = \{1, \ldots, m+2n\} \setminus \{m+i, m+n+i\}$$

with the two values, $m+i$ and $m+n+i$ excluded from $D(e_i)$.

The **second** set of decision variables $(\hat{\mathbf{l}}, \hat{\mathbf{w}})$ determines the sizes of the $m+2n$ regions: the length $\hat{\mathbf{l}} \in \{\mathbb{N}\}^{m+2n}$ and the width $\hat{\mathbf{w}} \in \{\mathbb{N}\}^{m+2n}$ where $\left(\hat{l}_j, \hat{w}_j\right)$ is the size of $R_j \in R$. Because $R_j = (L, W)$ for $j = 1, \ldots, m$, while $R_{m+j} = R^t_j$ and $R_{m+n+j} = R^r_j$ for $j = 1, \ldots, n$, the domains of $\hat{\mathbf{l}}$ and $\hat{\mathbf{w}}$ are:

$$D\left(\hat{l}_j\right) = \begin{cases} L & \text{if } j \in \{1, \ldots, m\} \\ \{l_{j-m}, \ldots, L\} & \text{if } j \in \{m+1, \ldots, m+n\} \\ \{0, \ldots, L - l_{j-m-n}\} & \text{if } j \in \{m+n+1, \ldots, m+2n\} \end{cases}$$

and

$$D(\hat{w}_j) = \begin{cases} W & \text{if } j \in \{1, \ldots, m\} \\ \{0, \ldots, W - w_{j-m}\} & \text{if } j \in \{m+1, \ldots, m+n\} \\ \{w_{j-m-n}, \ldots, W\} & \text{if } j \in \{m+n+1, \ldots, m+2n\} \end{cases}.$$

The **third** set of decision variables $\mathbf{x} \in \{\mathbb{N}\}^{m+2n}$ and $\mathbf{y} \in \{\mathbb{N}\}^{m+2n}$ determines the bottom-left coordinates (x_j, y_j), $j = 1, \ldots, m+2n$, of regions $R_j \in R$. These variables have domains

$$D(x_j) = \begin{cases} 0 & \text{if } j \in \{1, \ldots, m\} \\ \{0, \ldots, L - l_{j-m}\} & \text{if } j \in \{m+1, \ldots, m+n\} \\ \{l_{j-m-n}, \ldots, L\} & \text{if } j \in \{m+n+1, \ldots, m+2n\} \end{cases}$$

and

$$D(y_j) = \begin{cases} 0 & \text{if } j \in \{1, \ldots, m\} \\ \{w_{j-m}, \ldots, W\} & \text{if } j \in \{m+1, \ldots, m+n\} \\ \{0, \ldots, W - w_{j-m-n}\} & \text{if } j \in \{m+n+1, \ldots, m+2n\} \end{cases}$$

The **fourth** set of decision variables $\upsilon \in \{0,1\}^n$ selects the cutting pattern (α or β) for every item $i \in I$: $\upsilon_i = 1$ (or *true*) if i is the result of an α pattern, and 0 otherwise.

Finally, the **fifth** set of decision variables $C \in \{\mathbb{Z}_{\geq 0}\}^m$ defines the completion times of the m bins. Let C_{max} be an upper bound on the completion time C_b of bin $b \in B$. Then, the domain of C_b is

$$D(C_b) = \{0, \ldots, C_{max}\}.$$

$$\min \bar{u} = \sum_{i \in I} \max\left(\epsilon_i \cdot (d_i - \texttt{Element}[C,a_i]), \tau_i \cdot (\texttt{Element}[C,a_i] - d_i)\right) \quad (1)$$

$$\text{s.t. } \texttt{AllDifferent}\left[e_1, \ldots, e_n\right] \quad (2)$$

$$((e_i \leq m) \rightarrow a_i = e_i) \wedge (\neg(e_i \leq m) \rightarrow a_i = \texttt{Element}[a,(e_i - m) \bmod n]) \qquad i \in I \quad (3)$$

$$q_b = \texttt{Count}\left[a, b\right] \qquad b \in B \quad (4)$$

$$\texttt{Equals}\left[q_b, 0\right] \rightarrow \sum_{k=b+1}^{m} q_k = 0 \qquad b \in B,\ m_{lb} < b < m \quad (5)$$

$$\texttt{Max}\left[a\right] = z \quad (6)$$

$$\left(l_i \leq \texttt{Element}\left[\hat{l}, e_i\right]\right) \wedge (w_i \leq \texttt{Element}[\hat{w}, e_i]) \qquad i \in I \quad (7)$$

$$\left(v_i \rightarrow \hat{l}_{m+i} = \texttt{Element}\left[\hat{l}, e_i\right]\right) \wedge \left(\neg v_i \rightarrow \hat{l}_{m+i} = l_i\right) \qquad i \in I \quad (8)$$

$$\hat{w}_{m+i} = \texttt{Element}[\hat{w}, e_i] - w_i \qquad i \in I \quad (9)$$

$$\hat{l}_{m+n+i} = \texttt{Element}\left[\hat{l}, e_i\right] - l_i \qquad i \in I \quad (10)$$

$$(v_i \rightarrow \hat{w}_{m+n+i} = w_i) \wedge (\neg v_i \rightarrow \hat{w}_{m+n+i} = \texttt{Element}[\hat{w}, e_i]) \qquad i \in I \quad (11)$$

$$x_{m+i} = \texttt{Element}[x, e_i] \qquad i \in I \quad (12)$$

$$y_{m+i} = \texttt{Element}[y, e_i] + w_i \qquad i \in I \quad (13)$$

$$x_{m+n+i} = \texttt{Element}[x, e_i] + l_i \qquad i \in I \quad (14)$$

$$y_{m+n+i} = \texttt{Element}[y, e_i] \qquad i \in I \quad (15)$$

$$\varphi\left[a, 1\right] \leq C_1 \quad (16)$$

$$C_{b-1} + \varphi\left[a, b\right] \leq C_b \qquad b \in B \setminus \{1\} \quad (17)$$

$$\sum_{i \in I} \lambda_{k'i} \texttt{Equals}\left[a_i, b\right] \leq 1 \qquad b \in B,\ k' = 1, \ldots, k \quad (18)$$

$$\texttt{Schedule}\left[a, C\right] \quad (19)$$

Figure 2: The CP model for the JITBP.

Along with the aforementioned decision variables characterized by their restricted domains, the CP model uses three auxiliary variables. The **first** set $\mathbf{a} \in \{1, \ldots, m\}^n$ reveals the items' affiliations: $a_i = b$ when item i is assigned to bin $b \in B$. The **second** set $q \in \{\mathbb{Z}_{\geq 0}\}^m$ gives the number of items assigned to each bin. Let q_{max} be a known upper bound on the number of items that can fit into a single bin, and let $m_{lb} \leq m$ be the lower bound of [14] on the number of bins required to cut the n items. Forcing the first m_{lb} bins to contain at least one item, $D(q_b) = \{1, \ldots, q_{max}\}$ for every $b \leq m_{lb}$ and $D(q_b) = \{0, \ldots, q_{max}\}$ for each $b > m_{lb}$. This rule partially excludes symmetrical solutions emerging when a filled bin b empties all its items into an empty bin b' that precedes or succeeds bin b on the cutting machine. This exchange of items produces a different solution but the same objective function's value. The **third** set is the number z of used bins, where $D(z) = \{m_{lb}, \ldots, m\}$.

4.2. The Constraint Programming Model

The CP model, which restricts the variables' domains as described in Section 4.1, is given in Figure 2. Equation (1) defines the objective function

value $\bar{u}$ as the TWET of the items of I. Because a_i defines the bin to which item i is assigned, $\texttt{Element}[C,a_i]$ is the completion time C_i of i. In fact, the expression $\texttt{Element}[\upsilon,h]$ returns the hth variable of υ. When i is early, $E_i = d_i - \texttt{Element}[C,a_i] > 0$ while $\texttt{Element}[C,a_i] - d_i < 0$. On the other hand, when i is tardy, $d_i - \texttt{Element}[C,a_i] < 0$ while $T_i = \texttt{Element}[C,a_i] - d_i > 0$. It follows that the maximum of $\epsilon_i \cdot (d_i - \texttt{Element}[C,a_i])$ and $\tau_i \cdot (\texttt{Element}[C,a_i] - d_i)$ determines the weighted earliness tardiness of i.

Constraint (2) ensures that the n items are assigned to different regions. It uses CP-specific constraint $\texttt{AllDifferent}\,[\upsilon_1, \ldots, \upsilon_n]$, which ensures that variables $\upsilon_1, \ldots, \upsilon_n$ take distinct values. Constraint (3) identifies the bin a_i corresponding to region e_i where item i is assigned. If i is positioned in one of the first m regions of R, then it is the first item assigned to its bin; therefore, the constraint sets $a_i = e_i$ when $e_i \leq m$. Alternatively, i is positioned in a residual top or right region of a bin whose first positioned item is item $(e_i - m) \bmod n$; therefore, the constraint assigns to i the bin a_i that includes its region e_i, where $a_i = \texttt{Element}[a,(e_i - m) \bmod n]$. Constraint (4) calculates the number of items contained in bin $b \in B$. It applies CP expression $\texttt{Count}\,[\upsilon, h]$, which counts the number of variables of υ taking the value h.

Constraint (5) is logically redundant. It breaks the symmetry that is inherent in this model. If bin b is empty, then all its succeeding bins should be empty. Because the first m_{lb} bins must have at least one item, this constraint is only imposed for bins following bin m_{lb}. It uses the CP expression $\texttt{Equals}\,[\upsilon_i, h]$, which returns 1 if $\upsilon_i = h$ and 0 otherwise. Constraint (6) bounds bin a_i assigned to item i by the upper bound z. Because no item is allocated to an empty bin, and all empty bins are tagged to the end of the schedule, z is the number of non-empty bins. This constraint uses the CP expression $\texttt{Max}\,[\upsilon] = \max\{\upsilon_1, \ldots, \upsilon_n\}$, which returns the maximum value of the elements of υ.

Constraints (7)-(15) guarantee the packing feasibility. Constraint (7) guarantees that region e_i, which holds item i, is large enough to fit i; i.e., that length l_i and width w_i are less than or equal to the region's length and width, given respectively by $\texttt{Element}\left[\hat{l},e_i\right]$ and $\texttt{Element}[\hat{w},e_i]$. Constraints (8) and (9) compute the size of the top residual area R_i^t obtained when cutting item i. When the first cut generating i is horizontal (i.e., $v_i = 1$), constraint (8) implies that the length $\hat{l}_{m+i}$ of R_i^t equals the length of region e_i where i is positioned. On the other hand, when the first cut generating i is vertical (i.e., $\neg v_i$), the length of R_i^t equals l_i. Constraint (9) calculates the width $\hat{w}_{m+i}$ of R_i^t as the difference between the width of region e_i and the width of item i. That is, the width $\hat{w}_{m+i}$ of R_i^t does not depend on the cutting pattern. Similarly, constraints (10) and (11) specify the size of the right residual region R_i^r. Constraint (10) sets the length $\hat{l}_{m+n+i}$ of R_i^r to the difference between the length of region e_i to which item i is assigned and the length l_i of item i. That is, the length of R_i^r is independent of the cut that generates i. Constraint (11) sets the width of R_i^r. When i is extracted via an α pattern (i.e., $v_i = 1$), the width of R_i^r equals the width w_i of item i. On the other hand, when i is cut via a β pattern (i.e., $\neg v_i$), the width of R_i^r is the same as the width of region e_i. Constraints (12) and (13)

determine the bottom leftmost x and y-coordinates of region R_i^t. They set the x-coordinate of R_i^t to e_i's x-coordinate, and set the y-coordinate of R_i^t to the sum of the y-coordinate of region e_i and w_i. Likewise, constraints (14) and (15) compute the coordinates of region R_i^r. Equation (14) sets the x-coordinate of R_i^r to the sum of the x-coordinate of region e_i and l_i, while Equation (15) sets its y-coordinate to the y-coordinate of region e_i. Excluding constraints (12)-(15) does not make the solution of JITBP infeasible. It may even speed the search.

Constraints (16) and (17) address the scheduling component of the problem. Constraint (16) ensures that the schedule starts on or after time zero. Indeed, idle time may be inserted at the beginning of the schedule when it reduces TWET. Constraint (17) guarantees the no overlap of the processing windows of two consecutive bins. The completion time of a bin is the sum of its starting time and its cutting time. Because idle time is allowed, the starting time of a bin b is larger than or equal to C_{b-1}, the completion time of bin $b-1$; i.e., of the bin that precedes bin b. The cutting time of a bin b is defined by $\varphi[\mathbf{a}, b]$, which is a function of the items assigned to bin b and given by $\mathbf{a}$. Finally, constraints (18) and (19) are logically redundant.

4.3. Redundant Constraints

Constraint (18) applies the concept of dual feasible functions (DFF) to strengthen the search for a feasible packing while constraint (19) applies mathematical programming to obtain a minimal cost feasible schedule of packed bins.

The application of dual feasible functions (DFFs) yields strong lower bounds to the minimal number of bins that can pack orthogonally a given set of 2D oriented items. This is a 2DBPP [2]. In addition, the application of DFFs indicates whether a subset of items may be packed into a single bin. This is a 2D orthogonal packing problem (2OPP). When its solution value is larger than one, the subset of items under consideration can't be packed into a single bin, and the packing of this subset is fathomed from JITBP's search space.

A DFF is a function $\mu : [0,1] \to [0,1]$ such that $\sum_{s\in S} s \leq 1 \Rightarrow \sum_{s\in S} \mu(s) \leq 1$ holds for any set S of non-negative real numbers. Assume that (μ_1, μ_2) is a pair of two DFFs, then $(\mu_1(l_i/L), \mu_2(w_i/W)) \in (0,1]^2$ represents the transformed dimensions of item $i \in I$, derived from i's original dimensions (l_i, w_i). For a feasible packing into a single bin to exist, the sum of the areas of the modified items must be less than or equal to 1,

$$\sum_{i\in I} \mu_1(l_i/L)\,\mu_2(w_i/W) \leq 1. \tag{20}$$

Various combinations of DFFs may lead to different transformed items. For k distinct DFF combinations, a real-valued matrix $\Lambda = (\lambda_{k'i}) \in R_{\geq 0}^{k\times n}$ stores a collection of scaled areas of item i as an argument of two DFFs, μ_1 and μ_2, for every $i \in I$ and $k' = 1, \ldots, k$. The DFFs $u^{(1)}$, $U^{(\epsilon)}$, and $\phi^{(\epsilon)}$ are the step functions described by Theorem 14 of Fekete and Schepers [14] for 2DBPP. Here, they are used to deduce, from constraint (20), the valid feasibility inequality

constraint (18) for the CP model. This inequality is of the form

$$\sum_{i \in I} \lambda_{k'i} \texttt{Equals}\,[a_i, b] \leq 1,\ b \in B,\ k' = 1, \ldots, k. \tag{21}$$

Some of the k constraints of inequality (18) may be superfluous. A constraint k', $k' = 1, \ldots, k$, is superfluous if either $\sum_{i \in I} \lambda_{k'i} \leq 1$ or there exists k'', $k'' = 1, \ldots, k$, $k' \neq k''$, such that $\lambda_{k'i} \leq \lambda_{k''i}$ for all $i \in I$. Any superfluous constraint is omitted. Among the non-superfluous ones, only a subset of the constraints are entered in the model; i.e., those that maximize the values of $\sum_{i \in I} \lambda_{k'i}$. Section 7.2 explains how the constraints are selected and how the DFFs' control parameters are set.

Even though the CP model optimizes the bins' completion times, its search might be slow. CP's inference process takes the form of domain reduction, constraint propagation and generation of "no good" cuts. To overcome this glitch, we use LP to deduce the optimal completion times $C_1, \ldots, C_m$ of the m bins once $\mathbf{a}$ is fixed. Specifically, we strengthen CP's search via a customized constraint $\texttt{Schedule}\,[\mathbf{a}, \mathbf{C}]$. For a given input $\mathbf{a}$, $\texttt{Schedule}\,[\mathbf{a}, \mathbf{C}]$ determines the optimal $\mathbf{C}$ that minimizes TWET by solving an LP, via any general purpose LP solver. It has three sets of decision variables: $\hat{\mathbf{C}} \in \{\mathbb{R}_{\geq 0}\}^m$ representing the completion times of the m bins, $\mathbf{E} \in \mathbb{R}^n_{\geq 0}$ and $\mathbf{T} \in \mathbb{R}^n_{\geq 0}$ expressing the earliness and tardiness of the n items. It is defined as follows:

$$\min \hat{u} = \sum_{i \in i} (\epsilon_i E_i + \tau_i T_i) \tag{22}$$

$$\text{s.t. } T_i - E_i = \hat{C}_b - d_i \qquad i \in I,\ b = a_i \tag{23}$$

$$\varphi\,[\mathbf{a}, 1] \leq \hat{C}_1 \tag{24}$$

$$\hat{C}_{b-1} + \varphi\,[\mathbf{a}, b] \leq \hat{C}_b \qquad b \in B \setminus \{1\} \tag{25}$$

$$\hat{u} \leq \bar{u}^* \tag{26}$$

$$T_i \in \mathbb{R}_{\geq 0},\ E_i \in \mathbb{R}_{\geq 0} \qquad i \in I \tag{27}$$

$$\hat{C}_b \in \mathbb{R}_{\geq 0},\ \underline{C}_b \leq \hat{C}_b \leq \overline{C}_b \qquad b \in B \tag{28}$$

LP schedules the bins on the single cutting machine. Eq. (22), which defines the objective, minimizes TWET. Eq. (23) calculates the earliness E_i and tardiness T_i of item i, which is in bin b; i.e., $b = a_i$. It defines the lateness $T_i - E_i$ of i as the difference between the completion time $\hat{C}_b$ of its bin and its due date d_i. Eq. (24) guarantees that the schedule starts on or after time zero. It allows for the insertion of idle time at the beginning of the schedule. It sets the completion time of the first bin greater than or equal to the processing time of the first bin. Eq. (25) determines the completion time of each bin ensuring that the processing periods of two successive bins do not overlap in time. It allows for the insertion of idle time between any pair of successive bins if this decreases TWET. Eq. (26) bounds the objective value by the cost $\bar{u}^*$ of the incumbent when an incumbent is available. It fathoms any non-improving solution. Finally, Eqs. (27)-(28) declare the variables positive real values. Additionally, Eq. (28) restricts the

interval of $\hat{C}_b$ to $[\underline{C}_b, \overline{C}_b]$, $b \in B$, where $\underline{C}_b$ and $\overline{C}_b$ are lower and upper bounds retrieved from the active CP. When LP is infeasible, `Schedule` $[\mathbf{a}, \mathbf{C}]$ is violated.

4.4. Search Strategy

The search is sequential. Each phase instantiates a subset of variables. The subsets do not overlap. Choosing the "right" sequence of phases can drastically reduce the search time. The order of the variables within a phase is undefined. The search engine applies its built-in policies to explore the search space.

Our CP approach divides its variables into 3 subsets: $\{z\}$, $\{\mathbf{e}, \mathbf{a}, \mathbf{C}\}$, and all other variables. Using these three distinct subsets, the search engine undertakes three sequential phases. The **first** phase decides the maximum number of used bins. The **second** phase assigns the items to particular regions. It fixes $\mathbf{e}$, which, in turn, fixes $\mathbf{a}$. This assignment triggers the customized constraint `Schedule` $[\mathbf{a}, \mathbf{C}]$. Subsequently, `Schedule` $[\mathbf{a}, \mathbf{C}]$ either signals an infeasible partial solution or reduces the values of $\mathbf{C}$ to a vector of singletons.

- In the former case, `Schedule` $[\mathbf{a}, \mathbf{C}]$ rejects the current partial solution and halts the exploration of the branch, signaling a "no good" cut. Consequently, CP backtracks.
- In the latter case, the search engine reduces the domains of C to the single values of $\hat{C}_1, \ldots, \hat{C}_m$, and proceeds to phase three.

The **third** phase continues the search, which instantiates the remaining variables including the packing variables: It constructs a feasible solution.

Evidently, the inference process within a phase affects the domains of all the variables of the model, not only of the variables that phase. Therefore, the search engine might detect the infeasibility of a partial solution during phase two. In such a case, it backtracks.

The reduced CP, which excludes constraints (12)-(15), may run faster than the complete model and achieve a tighter upper bound. Its solution then serves as a partial solution to the complete model with the variables' domains restricted to the values of that solution. Then, the search must only fix the remaining part. We explore this trivial idea to improve the performance of the approach.

5. Decomposition-Based Approaches

Our two other approaches adopt the branch-and-check (B&C) form of the logic-based Benders decomposition (LBBD) [13, 41]. LBBD delays the enforcement of complicating constraints. It relaxes the complicating constraints to obtain an MP. It then iteratively augments MP with constraints deduced from its interrelated logic-based sub-problem(s). Specifically, LBBD obtains an optimal solution of MP. Keeping the variables of that solution fixed, it solves the sub-problem(s) and generates a Benders cut(s) for MP. When the solution for every sub-problem is feasible, a new incumbent is at hand. It then augments MP with all generated cuts and solves the resulting MP. It stops when MP does not improve the incumbent; i.e., the incumbent is a proven global optimum.

B&C and LBBD differ in terms of when to solve the sub-problems. While LBBD solves the subproblem(s) when MP's optimum is at hand, B&C solves the subproblem(s) whenever a feasible solution to MP is at hand. B&C explores the fact that identifying a global optimum or proving the optimality of the incumbent are generally the most time-consuming parts of a B&B search. Therefore, in lieu of seeking MP's optimality, it solves the sub-problem(s) whenever a feasible solution for MP is found during the B&B search. As a rule of thumb, problems having more difficult MPs than sub-problems are better suited to B&C, whereas those having more difficult sub-problems than MP are better adapted to LBBD [6]. In the following, we explain how we decompose JITBP and why our decomposition is better adapted to B&C than to LBBD.

In our decomposition, the packing constraints are considered the complicating constraints. They are dropped from MP and assigned to the subproblems. Consequently, MP considers items as dimensionless entities, assigns them to the bins, and simultaneously searches for the bins' completion times that minimize TWET. That is, MP drops the geometric constraints that describe the relationships between pairs of packed items and between a packed item and its assigned bin. Those constraints consider both the length and width of the items and bins, and ensure the non-overlap of pairs of items and the containment of an item in the bin in both dimensions. To overcome this glitch, our decomposition delegates the mission of detecting the feasibility of a packing to the sub-problems. The sub-problems may identify an infeasible packing of a subset of items in one or more used bins. When packing rules are violated, a sub-problem eliminates such infeasible pattern by generating a cut that is injected into MP.

Adopting the decomposition paradigm, we consider two models that take advantage of the respective strengths of MIP and CP.

- The **first** models MP as an MIP (without the packing constraints). Despite the strength of MIP in solving assignment problems, this model suffers from the "Big M" reformulation, which ensures disjunction in pairing items with their bins. It makes MP hard to solve exactly; thus, motivating our choice of B&C rather than LBBD.
- The **second** models MP as a relaxation of the full CP model of Section 4; i.e., dropping the packing constraints (7)-(15) and their associated variables. We consider this model so that we can compare the performance of CP and MIP as solution techniques for MP.

Hereafter, we detail both decomposition approaches. Sections 5.1 and 5.2 present the MIP and the CP based MPs while section 5.3 explain the 2OPP packing procedure that solves the sub-problems.

5.1. A Mixed-Integer Programming Based Master Problem

Herein, MP is modeled as an assignment MIP. Let $\mathbf{a}$ denote a matrix of binary decision variables such that $a_{ib} = 1$ if item $i \in I$ is assigned to bin $b \in B$ and 0 otherwise. Let E_i and T_i be two non-negative real-valued decision

$$\min \sum_{i \in I} (\epsilon_i E_i + \tau_i T_i) \tag{29}$$

$$\text{s.t.} \sum_{b \in B} a_{ib} = 1 \qquad i \in I \tag{30}$$

$$E_i - T_i + S_{ib} = d_i - C_b \qquad i \in I,\, b \in B \tag{31}$$

$$C_{max}\,(a_{ib} - 1) \leq S_{ib} \leq C_{max}\,(1 - a_{ib}) \qquad i \in I,\, b \in B \tag{32}$$

$$\varphi\,(a) \leq C_1 \tag{33}$$

$$C_{b-1} + \varphi\,(a) \leq C_b \qquad b \in B \setminus \{1\} \tag{34}$$

$$\sum_{i \in I} a_{ib} \geq 1 \qquad b = 1, \ldots, m_{lb} \tag{35}$$

$$\sum_{i \in I} a_{ib} \leq q_{max} z_b \qquad b = m_{lb} + 1, \ldots, m \tag{36}$$

$$z_{b+1} \leq \sum_{i \in I} a_{ib} \qquad b = m_{lb} + 1, \ldots, m \tag{37}$$

$$\sum_{i \in I} \lambda_{k'i} a_{ib} \leq 1 \qquad b \in B,\, k' = 1, \ldots, k \tag{38}$$

$$a_{ib} \in \{0, 1\} \qquad i \in I,\, b \in B \tag{39}$$

$$z_b \in \{0, 1\} \qquad b \in B \tag{40}$$

$$T_i,\ E_i \in \mathbb{R}_{\geq 0} \qquad i \in I \tag{41}$$

$$C_b \in \mathbb{R}_{\geq 0} \qquad b \in B \tag{42}$$

$$S_{ib} \in \mathbb{R} \qquad i \in I,\, b \in B \tag{43}$$

Figure 3: The mixed-integer master program of the B&C approach for the JITBP.

variables denoting the earliness and tardiness of item i. Let C_b be a non-negative real-valued decision variable expressing the completion time for bin b. In addition, let z_b be a binary variable equal to 1 when bin b is used, and equal to 0 when b is empty. Finally, let $S_{ib} \in \mathbb{R}$ be an auxiliary slack variable. Then the MIP model built on these sets of variables is depicted in Figure 3.

Eq. (29) defines the objective function, which minimizes the TWET of dimensionless items. Eq. (30) positions item i in exactly one bin. Eq. (31) calculates the earliness E_i and the tardiness T_i of item $i \in I$. Eq. (32) implements a "Big M" constraint that constrains a slack variable S_{ib} to 0 when i is assigned to bin $b \in B$. Otherwise, it allows S_{ib} to be any arbitrary negative or positive value within the bounded range $[-C_{max}, C_{max}]$. C_{max} is an upper bound on the maximal completion time of all bins. Eq. (33) ensures that the schedule starts after time zero. Eq. (34) schedules the bins: It calculates their completion times inserting idle time between consecutive bins when necessary. The completion time of a bin b is larger than or equal to the sum of the completion time of its predecessor bin $b - 1$ and the cutting time of bin b. Both Eqs. (33) and (34) use the linear function $\varphi\,(\mathbf{a})$, which defines the cutting time of bin b. This cutting time depends on the items assigned to bin b; i.e., on the items whose $a_{ib} = 1$. Eq. (35) ensures that each of the first m_{lb} bins has at least one item. In fact, there is no feasible packing that uses less than m_{lb} bins. Eq. (36) determines the used bins. This constraint uses an upper limit q_{max} on the

number of items packed into a bin. Eq. (37) forces bin $b+1$ to be empty when its predecessor bin b is empty, but allows the use of bin $b+1$ otherwise. It is a symmetry breaking constraint. Eq. (38) is a feasibility constraint of the form of inequality (20). Its inclusion in MP tightens the relaxation while its exclusion omits the layout aspect of the problem; thus, can not produce reasonably good bounds. Finally, Eqs. (39)-(43) declare the variables' types.

The search strategy sets a priority order for variables of $\mathbf{a}$ so that it branches on variables with larger areas first. It does not prioritise the variables of vector $\mathbf{C}$; thus, makes them equally important. It strives to assign the items to the bins. However, because it only branches on variables having fractional variables, the search may delay assigning large-sized items until late in the search tree.

A larger number of constraints derived from Eq. (38) does not necessarily strengthen MP, and may even slow down the search. Indeed, not all the feasibility constraints tighten the lower bound on the packing area. Here, we only augment the model with the existing constraints that have the largest total sums $\sum_{i \in I} \lambda_{k'i}$, $k' = 1, \ldots, k$. We treat the remaining constraints as lazy constraints that are only checked when an integer feasible solution is found. A lazy constraint is activated within the model when such a check reveals its violation. Section 7.1 further specifies the selection of the feasibility constraints along with other settings applied to the MIP solver.

The B&B search applied to MP is a source of multiple 2OPP sub-problems, which decide whether a subset of items is packable into a single bin subject to the existing geometrical constraints. Every time the search faces a new feasible integer solution that outperforms the incumbent, that solution is checked for packing feasibility of the used bins. The algorithm sequentially solves the 2OPP for each of the bins retrieving the information about the assigned items from $\mathbf{a}$. To solve the sub-problem, it calls the packing procedure described in Section 5.3. When the packing constraints are satisfied for every bin of the candidate solution, the search updates the incumbent and resumes exploring the solution space. If the routine fails to prove packing feasibility for a subset of items $I_b = \{i \in I : a_{ib} = 1\}$ assigned to bin $b \in B$, the MIP is augmented with one or more Benders cuts. One approach would augment MP with

$$\sum_{i \in I_b} a_{ib} \leq |I_b| - 1, \tag{44}$$

which forces bin b to drop one of its assigned items. Alternatively, we augment MP by a set of Benders cuts:

$$\sum_{i \in I_b \setminus \{j\}} a_{ib} + \frac{1}{|I^*|} \sum_{i \in I^*} a_{ib} \leq |I_b| - 1 \qquad \forall j \in I_b \tag{45}$$

where $I^* = \{j\} \cup \{i \in I \setminus I_b : l_j \leq l_i \wedge w_j \leq w_i\}$ is the union of item $j \in I_b$ with the set of items that are not included in bin b but that are identical to or larger than item j. There are as many Benders cuts as there are items in bin b: one cut per item $j \in I_b$. The cut restricts the mutual assignment to b of all the items of $I_b \setminus \{j\}$ and of any item of I^* (not only of item j). That is, every item

$$\min \bar{u} = \sum_{i \in I} \max\left(\epsilon_i \cdot (d_i - \texttt{Element}[C,a_i]), \tau_i \cdot (\texttt{Element}[C,a_i] - d_i)\right)$$

$$\text{s.t. } \texttt{AllDifferent}\,[a_1, \ldots, a_n] \tag{46}$$

$$\texttt{Packing}\,[a] \tag{47}$$

$$\texttt{Eqs. (4 - 6, 16 - 19)}$$

Figure 4: The master problem's CP model for the B&C decomposition approach.

$i \in I^*$ must be assigned to a bin $b' \in B \setminus \{b\}$ when all the items of $I_b \setminus \{j\}$ are assigned to b. Our experiments suggest that the cut of Eq. (45) performs better than the cut of Eq. (44), albeit not all items of I^* may replace j. In fact, the assignment of items into bins is also affected by the proximity of the due dates of items included in I_b and by the resulting penalties. When $I^* = \{j\}$ for any $j \in I_b$, then (45) reduces to (44), which is the cut herein applied.

5.2. A Constraint Programming Based Master Problem

Herein, MP is a relaxation of the CP model of Section 4. It assigns items to bins and schedules the bins on the cutting machine. The sub-problems search for a feasible packing. That is, MP, given by Figure 4, uses only a subset of the variables and constraints of the model of Figure 2. It uses decision variables $\mathbf{a}$, $\mathbf{C}$, q and z with $\mathbf{a}$ no longer being a vector of auxiliary variables. In addition, constraint (46) differs from constraint (2) in that it spans the variables of $\mathbf{a}$. Finally, the model is augmented with the customized constraint $\texttt{Packing}\,[\mathbf{a}]$, which relates to a series of 2OPP sub-problems. Despite these differences, MP adopts the same search strategy as the integrated model.

Constraint (47) is linked to the domain of $\bar{u}$. It is activated once the variable is reduced to a singleton. This occurs when all items are assigned to the scheduled bins. The filtering algorithm of $\texttt{Packing}\,[a]$ decodes the assignment to a series of 2OPP decision problems, and solves each via the packing procedure of Section 5.3. If PACK finds a feasible packing pattern for each of the used bins, then the current MP's solution is feasible and is a new incumbent. Otherwise, MP's solution is rejected, the set of items violating the packing constraints is cached, and the search backtracks. The search later uses cached results to avoid calling PACK when the subset of items is in the cache. To achieve better performance, we made caching subset-size specific; thus, two infeasible solutions are cached separately if they comprise a different number of items. Cache is checked in ascending order of the subset size starting from the subset of size two.

5.3. A Bin Packing Procedure

The 2OPP decision sub-problem is part of the search of both approaches. It checks whether a guillotine packing of a given set of items I_b, $b \in B$, exists. Here, we adapt the complete CP model of Section 4 for the case of a single bin. In contrast to the full version, the reduced model, referred to as PACK, operates on a set of $1 + 2n'$ regions, $n' = |I_b|$, so that the first region corresponds to the empty bin, while the remaining $2n'$ elements refer to $R^t_1, \ldots, R^t_{n'}$ and $R^r_1, \ldots, R^r_{n'}$; i.e.,

$$\texttt{AllDifferent}\,[e_1,\ldots,e_{n'}]$$

$$\left(l_i \le \texttt{Element}\left[\hat{l},e_i\right]\right) \wedge \left(w_i \le \texttt{Element}[\hat{w},e_i]\right) \qquad i \in I_b$$

$$\left(v_i \rightarrow \hat{l}_{i+1} = \texttt{Element}\left[\hat{l},e_i\right]\right) \wedge \left(\neg v_i \rightarrow \hat{l}_{i+1} = l_i\right) \qquad i \in I_b$$

$$\hat{w}_{i+1} = \texttt{Element}[\hat{w},e_i] - w_i \qquad i \in I_b$$

$$\hat{l}_{n'+i+1} = \texttt{Element}\left[\hat{l},e_i\right] - l_i \qquad i \in I_b$$

$$\left(v_i \rightarrow \hat{w}_{n'+i+1} = w_i\right) \wedge \left(\neg v_i \rightarrow \hat{w}_{n'+i+1} = \texttt{Element}[\hat{w},e_i]\right) \qquad i \in I_b$$

$$x_{i+1} = \texttt{Element}[x,e_i] \qquad i \in I_b$$

$$y_{i+1} = \texttt{Element}[y,e_i] + w_i \qquad i \in I_b$$

$$x_{n'+i+1} = \texttt{Element}[x,e_i] + l_i \qquad i \in I_b$$

$$y_{n'+i+1} = \texttt{Element}[y,e_i] \qquad i \in I_b$$

Figure 5: The CP model of packing procedure PACK.

to the regions produced by cutting the items $1,\ldots,n'$. Furthermore, PACK omits the objective function given by constraint (1) and drops all but $\mathbf{e}$, $\hat{\mathbf{l}}$, $\hat{\mathbf{w}}$, $\mathbf{x}$, $\mathbf{y}$, and $\mathbf{v}$ variables. This subsequently leads to the exclusion of constraints (3)-(6) and (16)-(19). This results in the CP model of PACK given by Figure 5. Herein, no particular CP search strategy is applied for PACK. Excluding decision variables $\mathbf{x}$ and $\mathbf{y}$ along with their related constraints from the model may speed-up the search of the decision problem.

6. Warm-Start Heuristic

Herein, we start the MIP and CP solvers from a feasible incumbent, which is the result of an iterative two-phase heuristic. The constructive phase builds a solution from scratch by iteratively adding one item until all items are packed. The improvement phase rebuilds the solution via a simple local search.

Each iteration of either of the two phases is a function of the item $j \in I$ that is to join a subset of already packed items $I' \subset I$ in a subset of bins $B' \subset B$. The packing of the items of I' into the bins of B' is equivalent to the current partial solution. With I' and B' fixed, the elements of the assignment vector $\mathbf{a}$ indicate the bin assigned to item i; i.e., $a_i = b$, $i \in I'$, $b \in B'$, when i is assigned to bin b. B' is built on k (partially) filled bins and $k+1$ dummy empty bins. The empty bins are used when item j can't fit into any of those k (partially-filled) bins. The $2k+1$ bins of B' are ordered such that every odd bin is empty while every even bin is filled. Thus, each iteration operates with $2k+1$ bins and aims to assign j to one of them. Clearly, j is either the first item of a new empty bin or an additional item in one of existing bins.

The choice for a bin can be undertaken iteratively by inserting the item into every bin and computing the resulting penalties. However, we herein opt for using an MIP that ignores the geometry of the items and bins. The MIP is faster than the iterative approach while it matches the iterative approach's results because it only considers a limited number of positions.

Let z_b be a binary decision variable that equals 1 when j is placed in bin b, and 0 otherwise. In addition, let C_b be a real-valued decision variable defining the completion time of b. Finally, let E and E_i (resp. T and T_i), $i \in I'$, be non-negative real-valued variables defining the earliness (resp. the tardiness) of item $j \in I \setminus I'$ and of the items in the partial solution. Using the above decision variables, the following MIP determines the bin b where item j is assigned.

$$\min \tilde{u} = \sum_{i \in I'} (\epsilon_i E_i + \tau_i T_i) + \epsilon_j E + \tau_j T \tag{48}$$

$$\text{s.t.} \sum_{b \in B'} z_b = 1 \tag{49}$$

$$E_i - T_i = d_i - C_b \qquad i \in I',\, b = a_i \tag{50}$$

$$d_j z_b - C_b \leq E \qquad b \in B' \tag{51}$$

$$C_b - d_j z_b - C_{max}\,(1 - z_b) \leq T \qquad b \in B' \tag{52}$$

$$\varphi\,(\varnothing, z_1) \leq C_1 \tag{53}$$

$$\varphi\,(I_b, z_b) + C_{b-1} \leq C_b \qquad b \in B' \setminus \{1\} \tag{54}$$

$$\tilde{u} \leq \bar{u}^* - 1 \tag{55}$$

$$z_b \in \{0, 1\} \qquad b \in B' \tag{56}$$

$$C_b \in \mathbb{R}_{\geq 0} \qquad b \in B' \tag{57}$$

$$T_i \in \mathbb{R}_{\geq 0},\; E_i \in \mathbb{R}_{\geq 0} \qquad i \in I' \tag{58}$$

$$T \in \mathbb{R}_{\geq 0},\; E \in \mathbb{R}_{\geq 0} \tag{59}$$

Eq. (48) defines the objective function as the TWET of already scheduled items and of the newly added item j. Eq. (49) sets j as an element of one of the bins. If Eq. (20) is violated for some k', $k' = 1, \ldots, k$, then $z_b = 0$, and z_b along with the associated constraints can be eliminated from the model. Eq. (50) recomputes the earliness and tardiness for items that have been placed into the bins during the earlier iterations; i.e., for items of the assignment vector $\mathbf{a}$. Should j join bin b, Eq. (51) and (52) calculate the resulting earliness and tardiness of j, where C_{max} denotes an upper bound on the maximal completion time of the bins. Eq. (53) defines the completion time of the first (empty) bin when its processing starts no earlier than time zero while Eq. (54) sets the completion times for all other bins. Both Eqs. (53) and (54) use the processing time $\varphi\,(I_b, z_b)$, which equals 0 if bin b is empty, and the processing time of the set of items $I_b \cup \{j\}$ when $z_b = 1$ or the set I_b when $z_b = 0$ and b contains items. Clearly, $I_b = \varnothing$ for every odd bin in B'. Furthermore, any dummy bin that stays empty in the model's solution is immediately excluded from further consideration. Eq. (55) bounds the objective value $\tilde{u}$ by the cost $\bar{u}^*$ of the incumbent when available; otherwise the model excludes this constraint. Finally, Eqs. (56)-(59) declare the variables' types.

The two phases of the warm-start heuristic use the above model. The first phase constructs a feasible solution from scratch. It iteratively finds the best position for the next item $j \in I$, where I is sorted in descending order of the items' areas. As no prior solution is available, this phase discards Eq. (55).

The second phase is a simple local search. It sequentially examines the packed items. It extracts one item from a bin and places the extracted item into a different bin or into a new bin if this improves the existing solution. Thus, this phase may increase/decrease the number of used bins. Solutions obtained within the second phase must outperform the incumbent as Eq. (55) implies. In the absence of an improving solution, the model is infeasible. When the second phase updates the incumbent, it does another round of checks for all the items. If it fails to relocate at least one of the items, it terminates; treating the incumbent solution as a local optimum. The incumbent provides the number of bins m as an input parameter to both decomposition-based approaches. Clearly, bounding the maximal number of bins by m might exclude a global optimum that uses more than m bins.

As the model assimilates the items to dimensionless entities, the algorithm tests whether the obtained solution satisfies the geometric packing requirements. To do so, it applies the packing procedure `PACK` of Section 5.3. If the packing constraints are satisfied for bin b when augmented by item j (i.e. when $z_b = 1$), the heuristic accepts the solution of the model. Otherwise (i.e., $z_b \leq 0$), a new constraint that prohibits the assignment of item j to bin b is added to the model, and the model is solved again. In this way, the first phase of the heuristic may always create a new bin. However, the second phase, whose objective is bounded by $\bar{u}^*$, may run out of alternative assignments for item j. The latter means that no feasible solution exists and j's current position is locally optimal.

7. Computational Results

The objective of the computational investigation is twofold: (i) to compare the performance of the three proposed approaches: 'I' the integrated approach, '1' the B&C using CP for MP, and '2' the B&C using MIP for MP; and (ii) to investigate the best suited MP for the decomposition based approach for JITBP: a CP or an MIP. To achieve these two objectives, we undertake an extensive computational experiment using 500 instances, and compare the relative performance of the three methods and the improvements they bring to the solution value of the warm-start heuristic H. We apply the appropriate statistical tests for all comparisons. We use a 5% significance level for all hypothesis tests and a 95% confidence level for all confidence interval estimates. A five point summary gives the minimum, first, second and third quartiles and the maximum. Based on the analysis of the computational results, we discuss all three approaches (how and why they work as they do), we point out when each approach should be used, we motivate our settings, and we indicate what makes an approach outperform the others.

Sections 7.1 and 7.2 present the benchmark instances and the algorithm settings while Sections 7.3 and 7.4 compare the results for small and larger sized instances. Section 7.5 is a concluding remark.

7.1. Benchmark Instances

We assess the performance of our approaches using the 2DBPP instances of [8] while making the items' due dates reflect the tradeoff between the scheduling and packing components of JITBP. The set has square bins of size $L = W$. It consists of 10 categories. The categories are grouped according to their item-to-bin size ratios. These ratios reflect the average number of items that can be packed into a bin: A low bin density set $\mathcal{L}$ with relatively large items and a high bin density set $\mathcal{S}$ with small items. Each category has five problem sizes: n = 20, 40, 60, 80, and 100. There are ten instances per category and problem size; thus, a total of 500 instances. Table 1 summarizes the characteristics of the instances. Columns 1 and 2 indicate the category and its set. Column 3 specifies the length of a side of a bin whereas Column 4 describes how the items' dimensions are generated. Categories 1-6 have homogeneous items that are randomly generated from a specific discrete uniform distribution whereas categories 7-10 contain heterogeneous items belonging to four types in various proportions. The four types correspond to items whose (l_i, w_i) are randomly generated from the respective discrete *Uniform* $\left(\left[\frac{2}{3}W, W\right], \left[1, \frac{1}{2}W\right]\right)$, $\left(\left[1, \frac{1}{2}W\right], \left[\frac{2}{3}W, W\right]\right)$, $\left(\left[\frac{1}{2}W, W\right], \left[\frac{1}{2}W, W\right]\right)$, and $\left(\left[1, \frac{1}{2}W\right], \left[1, \frac{1}{2}W\right]\right)$.

Table 1: Generating items' widths and heights

Category	Set	$L = W$	Item size (l_i, w_i)
1	$\mathcal{L}$	10	*Uniform*$[1, 10]$
2	$\mathcal{S}$	30	*Uniform*$[1, 10]$
3	$\mathcal{L}$	40	*Uniform*$[1, 35]$
4	$\mathcal{S}$	100	*Uniform*$[1, 35]$
5	$\mathcal{L}$	100	*Uniform*$[1, 100]$
6	$\mathcal{S}$	300	*Uniform*$[1, 100]$
7	$\mathcal{L}$	100	type 1 with probability 70%; type 2, 3, 4 with probability 10% each
8	$\mathcal{L}$	100	type 2 with probability 70%; type 1, 3, 4 with probability 10% each
9	$\mathcal{L}$	100	type 3 with probability 70%; type 1, 2, 4 with probability 10% each
10	$\mathcal{S}$	100	type 4 with probability 70%; type 1, 2, 3 with probability 10% each

The earliness ϵ_i and the tardiness τ_i per time unit penalties follow a *Uniform*$[1, 10]$. As in real life manufacturing, a bin's processing time accounts for its setup time t^l and an item's handling time t^h. Herein, the processing time of a non-empty bin b is $\varphi(I_b) = t^l + t^h \cdot |I_b|$ with $t^l = 180$ and $t^h = 40$. Furthermore, the items' due dates follow a discrete *Uniform*$[0.5\lambda, \lambda]$ with $\lambda = t^l \cdot m_{lb} + t^h \cdot n$. This distribution, determined after extensive computational experiments, accounts for the minimum number of bins m_{lb} and the problem size n. It makes the numbers of early and tardy items relatively balanced and "forces"each item to search for its ideal bin as in industry. Consequently, the proposed approaches must address the items' competition for available space within bins while minimizing TWET. Instances with sparse or dense due dates leave little room for optimisation. They often lead to trivial solutions. Sparse due dates suppress the packing component of JITBP gearing the problem toward a scheduling problem whereas dense due dates gear the problem towards a 2DBPP. This due dates' distribution, which is a function of m_{lb}, allows us to observe the effect of the scaling of the items' sizes to bins and of the number of items.

7.2. Algorithm Settings

The proposed approaches are implemented in Java, which evokes IBM ILOG Optimization Studio 12.8 to handle MIP and CP models. The models are run on a personal computer with 4GB of RAM and a 3.06 GHz Dual Core processor. However, for a fair comparison, we solve both the MIP and CP models using a single core and setting $C_{max} = \max_{i \in I}\{d_i\} + t^l \cdot m + t^h \cdot n$, where t^l and t^h are the parameters of the cutting time function φ. We subsequently set the upper bound on the number items that a bin can fit to $q_{max} = i^* : \sum_{j=1}^{i^*} l_{[j]} w_{[j]} \leq LW$, $\sum_{j=1}^{i^*+1} l_{[j]} w_{[j]} > LW$ and $l_{[1]} w_{[1]} \leq l_{[2]} w_{[2]} \ldots, \leq l_{[n]} w_{[n]}$.

We generate the feasibility constraints of Eq. (20) using a pair of control parameters $(p, q) \in \{0.1, 0.2, 0.3, 0.4, 0.5\}^2$. This generates up to $k = 55$ feasibility constraints. Only five feasibility constraints are injected in the CP models (both integrated and decomposed approaches). The MIP model of the MP of Approach 2 includes ten feasibility constraints while the other 45 constraints are injected as lazy constraints (cf. Section 4.3).

All proposed approaches can solve the problem exactly. However, JITBP is computationally challenging. Therefore, we focus on the quality of the primal bounds. First, we set a three-second time limit on `PACK`; making it act as a heuristic. If, after three seconds, `PACK` does not identify a feasible solution to 2OPP, we assume that the items can't be packed in a single bin. This setting, inferred from preliminary computational investigations, gives the best trade-off between the quality of the packing and runtime. Indeed, a larger time limit does not necessarily lead to better packing solutions while it unduly increases the runtime of the decomposition approaches. It may enhance the bound but because the number of calls of `PACK` is very large, it will increase the runtime of the dependent approaches and deteriorate the solution quality. Similarly, a shorter runtime often hinders `PACK` from reaching a feasible packing; thus causes poor quality solutions. The chosen set up ensures a balanced strategy between solution quality and runtime. The Benders cut uses the set I^* of items j such that $1.5l_j \leq l_i$ and $1.5w_j \leq w_i$. Because `PACK` is a heuristic, it generally fails to find a feasible packing within the time limit, even though packing larger items within a bin may be feasible by another run of `PACK`. Thus, the strict form of cut (45) may overlook many feasible solutions. This current choice of the set I^* therefore cuts off only those patterns that are most likely to be infeasible.

Second, we set the MIP solver CPLEX to emphasize the search of high quality hidden feasible solutions. This set up generates more feasible solutions during its search for the optimum, at the cost of a slower proof of optimality. The MIP solver CPLEX proves optimality when the relative optimality gap is at its default value 10^{-6}.

Third, we base MIP's node selection on the best estimate. This selection strategy favors the node with the best progress toward integer feasibility relative to objective function degradation. This setting is recommended when it is difficult to find feasible solutions or when a proof of optimality is not crucial.

Fourth, we initiate the MIP of approach 2 from H's feasible solution with at most m bins, where m is obtained by H. When cold started, CPLEX fails to

identify a good-quality feasible solution within any of the allocated run times. It fails to produce a dense packing. It focuses on various sparse assignments. Because of the large size of the feasible solution space and poor linear relaxation, CPLEX spends a large portion of its search exchanging pairs of items. Even though this may improve TWET, it doesn't improve the density of the bins. It is as if CPLEX were spending its search time solving the single machine TWET scheduling problem with little or no consideration to the BP aspect. Subsequently, in most cases, CPLEX returns low-density filled bins at the end of the allocated runtime. For the instances at hand, the larger the number of bins, the higher the TWET is.

To obtain a dense packing, the warm-start heuristic H decides iteratively where to pack the item under consideration: in a new or in an existing bin. CPLEX exploits H's solution to explore potential assignments of items into the m bins with the objective of minimizing TWET. It takes CPLEX hours of run time to reach the same solution quality as H. For fairness, we warm-start the three approaches, and we bound the maximal number of bins by m.

Finally, the performance of a CP model depends on its solver; particularly, on the filtering algorithms and on its search strategies. Here, we resort to the CP Optimizer with its search algorithm set to the *restart mode*. This mode adopts a general purpose search strategy inspired from integer programming techniques. It guarantees the optimality of the final solution of a problem. It is based on the concept of the impact of a variable. The impact measures the importance of a variable in reducing the search space. It is learned from the observation of the domains' reduction during the search. It helps the restart mode drastically improve the performance of the search. The inference level for all the constraints in CP-based models is set to medium. This strengthens the domain reduction of the search at the cost of a larger computational time.

7.3. Results for Small Sized Instances

Table 2 summarizes the computational results for small sized instances (i.e., $n = 20$ and 40). Columns 1 and 2 indicate the problem size and category. Column 3 displays the average, over all instances of a category, of the performance ratio of H. Columns 4-6, 7-9, and 10-12 report the average over all instances of a category of the performance ratio of the integrated approach, B&C with CP, and B&C with MIP. The three columns of each triplet give the average when the allocated runtime $t = 100$, 600, and 3600 seconds. The performance ratio of an approach $\bullet$, $\bullet = H, I, 1, 2$ is expressed as a percentage. It is defined by $100\% \frac{u_\bullet}{\underline{u}}$, where $\underline{u} = \min\{u_H,\ u_I,\ u_1,\ u_2\}$ is the value of the best local minimum over all proposed approaches, and $u_H,\ u_I,\ u_1,\ u_2$ are the values of the local optima obtained by the warm startup heuristic, the integrated approach, B&C with CP, and B&C with MIP, respectively. Column 13 reports the number of instances solved exactly by B&C with MIP (with m bins and a 3-second time limit on `PACK`); that is, the number of times $u_2 = u^*$, where u^* is the optimum. Column 14 gives the number of times B&C with MIP provides the tightest bound. The last two columns provide RT_H and RT, the average run times of H and B&C

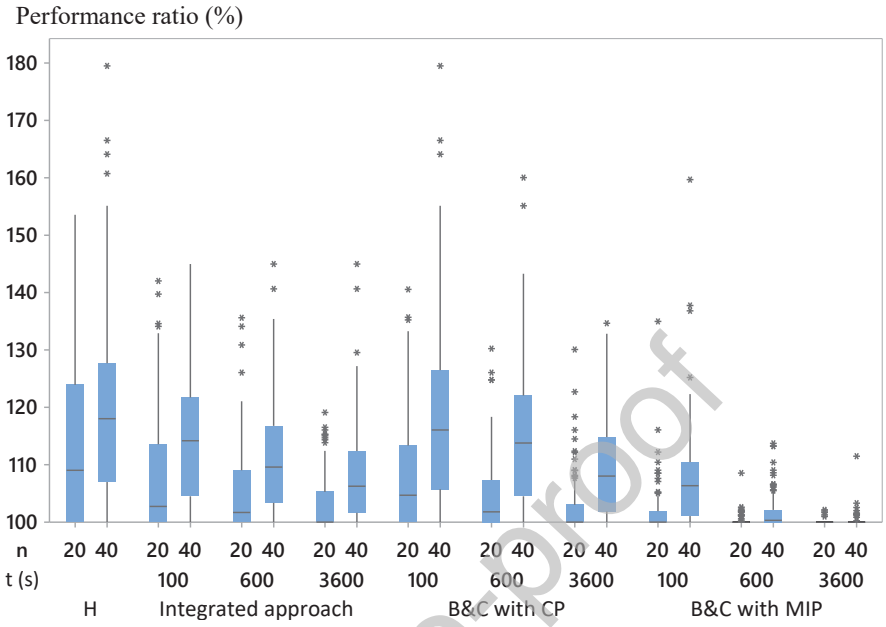

Figure 6: Box plots of the observed performance ratios

with CP when this latter proves the optimality of its incumbent. Next, Table 3 reports the number of times $u_H = \underline{u}$, and $u_\bullet = \underline{u}$, $\bullet = I, 1, 2$ when the allocated run times are $t = 100$, 600, and 3600 seconds. Finally, Figures 6 and 7 summarize the results. They present, respectively, the box plots of the performance ratios per problem size and 95% confidence intervals of their means.

The analysis of Tables 2 - 3 and of Figures 6 - 7 suggests that increasing the runtime enhances the performance of the three approaches. In addition, initializing the integrated approach with H's solution makes CPLEX improve that solution. In some instances, it allows CPLEX to converge to the global optimum u^* that is achievable with m bins and a 3-second threshold runtime for `PACK`. Paired t-tests confirm that increasing the runtime from 100 seconds to 600 seconds and from 600 seconds to 3600 seconds decreases both the mean TWET and the mean performance ratios for all three methods.

For set $\mathcal{S}$ and $n = 20$, H obtains the best solution in 28 out of 40 instances. The solution of each of these instances is suspected to be the global optimum (for m bins and a 3-second threshold runtime for `PACK`). It was not improved by any of the three proposed methods independently of the allocated runtime.

Even though both the integrated approach and B&C with CP rely on CP's search and inferencing, paired t-tests indicate that, for equal run times, the integrated method is on average better than B&C with CP. This is reasonable because the integrated approach has a better vision of the problem's specificity that is drawn from the extended set of problem-related constraints; e.g., the

Table 2: Average performance ratios of all proposed approaches, number of times B&C with MIP obtains the best bound and the optimum, and the average run times of the warm-up heuristic and of B&C with MIP for small sized instances

		Average Performance Ratio (%)												
		Heur.	Integrated appr. for time limit $t(s)$			B&C with CP for time limit $t(s)$			B&C with MIP for time limit $t(s)$			Number of times		Runtime (s)	
n	Cat.		100	600	3600	100	600	3600	100	600	3600	$u_2 = u^*$	$u_2 = \underline{u}$	RT_H	RT_2
20	1	125.3	112.0	108.0	105.6	115.5	107.4	104.6	103.7	100.0	100.0	1	10	1	2538
	2	100.5	100.1	100.1	100.1	100.1	100.1	100.1	100.0	100.0	100.0	10	10	1	3
	3	115.4	111.5	108.2	104.3	107.4	104.5	102.3	100.5	100.4	100.0	6	10	5	1229
	4	100.7	100.6	100.2	100.0	100.6	100.0	100.0	100.0	100.0	100.0	10	10	1	40
	5	124.4	116.2	110.2	108.4	118.1	110.9	107.7	102.9	100.0	100.0	4	10	2	1385
	6	100.1	100.1	100.1	100.1	100.1	100.1	100.1	100.0	100.0	100.0	10	10	0	2
	7	119.8	107.1	104.6	104.1	109.3	105.5	103.0	102.9	101.1	100.3	0	8	1	
	8	121.3	110.3	107.5	102.3	110.2	106.9	103.4	102.2	100.9	100.0	2	10	0	1566
	9	104.4	102.0	101.0	100.6	101.5	101.1	100.8	101.5	100.4	100.3	1	8	0	512
	10	122.7	117.3	115.6	108.2	118.5	111.5	105.4	104.2	100.4	100.2	8	9	6	829
Average		113.5	107.7	105.6	103.4	108.1	104.8	102.7	101.8	100.3	100.1	52	95	2	503
40	1	124.2	119.1	113.5	109.2	123.4	119.4	114.0	109.2	102.2	100.0	0	10	3	
	2	118.8	110.1	109.7	109.5	114.7	111.5	103.7	109.3	102.1	101.5	1	7	37	285
	3	113.6	111.9	111.0	108.7	113.2	112.6	112.0	108.8	103.7	100.0	0	10	23	
	4	118.1	111.0	109.1	108.0	115.5	107.1	104.4	110.9	102.3	100.4	1	7	31	2244
	5	114.1	112.3	110.2	108.3	114.1	113.8	112.4	107.7	101.1	100.1	0	9	12	
	6	114.5	105.9	105.8	103.6	114.1	109.9	101.5	103.9	100.0	100.0	2	10	21	549
	7	132.9	124.8	119.0	111.2	129.9	123.6	115.6	108.4	102.0	100.3	0	9	3	
	8	128.7	121.6	114.3	108.1	128.4	125.5	115.6	106.6	101.6	100.0	0	10	4	
	9	112.1	110.5	106.3	102.1	111.2	109.3	105.1	107.4	103.4	100.3	0	9	5	
	10	116.8	116.1	113.3	110.8	116.8	114.6	107.5	105.6	101.0	100.2	0	8	40	
Average		119.4	114.3	111.2	108.0	118.1	114.7	109.2	107.8	102.0	100.3	4	89	18	907

Table 3: Number of times the proposed approach obtains $\underline{u}$ for small sized instances

			Integrated appr. for time limit $t(s)$			B&C with CP for time limit $t(s)$			B&C with MIP for time limit $t(s)$		
		Heur.									
n	Set		100	600	3600	100	600	3600	100	600	3600
20	L	4	10	15	21	6	11	23	28	44	56
20	S	26	28	28	30	28	30	33	35	38	39
Overall		30	38	43	51	34	41	56	63	82	95
40	L	2	2	3	5	2	2	3	5	18	57
40	S	3	3	4	7	4	12	17	13	21	32
Overall		5	5	7	12	6	14	20	18	39	89

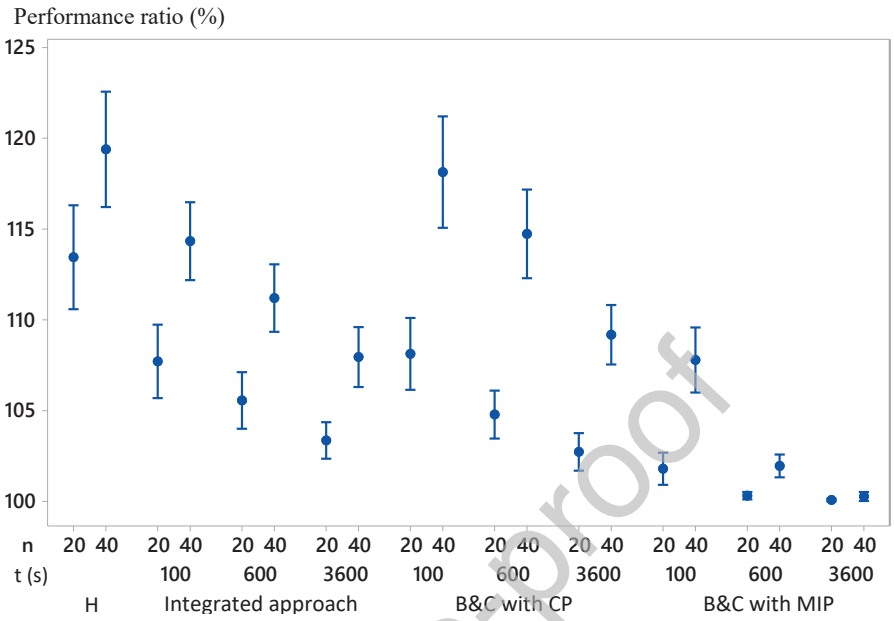

Figure 7: 95% confidence intervals for the mean performance ratios

packing constraints. Imposing more constraints is often beneficial for the CP search. It strengthens the inferencing and domain reduction. Thus, it makes the CP search more efficient when dealing with the integrated model than when solving the regular CP model of B&C with CP. When manipulating the integrated model, the search engine avoids the investigation of a large number of solutions; specifically those using a large number of bins. In this way, it avoids too much diversification while relying on intensification around solutions that have a dense packing. In addition, the integrated approach applies a "back engineering" approach by imposing an upper bound on the number of bins. On the other hand, when solving the model of B&C with CP, the constraint programming solver does not explore the additional information brought explicitly from the problem-specific constraints. Because its model is less specific, the search's progress is slower than when solving the integrated model.

Paired t-tests indicate that, for equal run times, the integrated approach is outperformed by B&C with MIP. However, this result must be interpreted with caution. If both approaches are cold started, the integrated approach obtains better quality solutions within a similar run time; i.e., the integrated approach's progress is quicker than the progress of B&C with MIP. When warm-started, B&C with MIP is faster and investigates more permutations for small instances. It undertakes an educated heuristic search around the initial solution. As the search space becomes large, the sampling becomes inefficient. Despite the valid statistical inference, the overall performance of the integrated approach is not much worse than the performance of the warm-started B&C

with MIP. Its average relative gap, which is of the order of 3%, is reasonably small. In the absence of such a warm-start strategy, the integrated approach is generally superior. B&C with MIP is myopic to the problem's specificity, thus does work on many heuristic solutions as part of the upper bounding process. The improvement of its results is not proportional to its search time. That is, B&C with MIP remains blind unless warm-started. Regardless, only 56 out of 200 instances are solved optimally by the warm-started B&C with MIP. A five point summary of their run times is $(0, 3, 47, 650, 3460)$ with an average of 532 and a standard deviation of 907; all in seconds. The large standard deviation reflects the different levels of difficulty of the instances. For the instances that were not solved to optimality, the average optimality gap, as reported by CPLEX, is 0.6766 with a standard deviation of 0.3015. The five point summary is $(0.0222, 0.4140, 0.8072, 0.9382, 1.0000)$.

Figure 8 illustrates the behavior of the proposed approaches as a function of the problem size and items' size. The mean performance ratio degrades as n increases from $n = 20$ to 40 because each item has more alternative positions. Therefore, the search must consider a larger search space. For set $\mathcal{S}$, the packing is dense. Therefore, regardless of the number of possible permutations, the items are easy to swap. Reshuffling the items will not have a large effect for two reasons: The total number of bins used is already reduced, and TWET is limited (because the due dates' distribution depends on m_{lb}). It is suspected that for $n = 20$, the warm-start heuristic does well because the search space is small. For $n = 40$, it most likely produces the same quality solutions as when $n = 20$ while CPLEX does well too.

As inferred by analysis of variance tests, the mean TWET for the instances of set $\mathcal{S}$ is smaller than its counterpart for set $\mathcal{L}$. In addition, the mean performance ratio of any approach is worse when the items are large than when the items are small. This is actually a result of the distribution of the due dates. The due dates are a function of m_{lb}, the minimal number of required bins. Thus, they implicitly address the items' sizes. A test instance with a prevalent number of small (versus large) items needs less bins; therefore its due dates are dense. The opposite prevails for an instance with many large items: the due dates become sparse while packing patterns become less dense. Furthermore, small items have generally more packing alternatives rather than larger items. This may force a large item to be packed in a bin whose completion time is far from the item's due date. These factors increase the chances of instances of set $\mathcal{S}$ to have their items cut around their due dates. It follows that the observed worse performance ratios for set $\mathcal{L}$ versus those of set $\mathcal{S}$ is reasonable.

Instances with larger items seem more challenging. However, packing checks are quite fast as the number of items per bin is small. Even though such instances call `PACK` more often, they run faster. Because the due dates are sparse, finding the ideal assignment of items to bins is hard. The local search tries to reinsert an item. Yet, the reinsertion has a very limited effect in terms of solution enhancement. Unless a major diversification is undertaken (a complete reconfiguration of the solution), it is almost impossible to escape from the local optimum by simply changing the position for an item. Because due dates

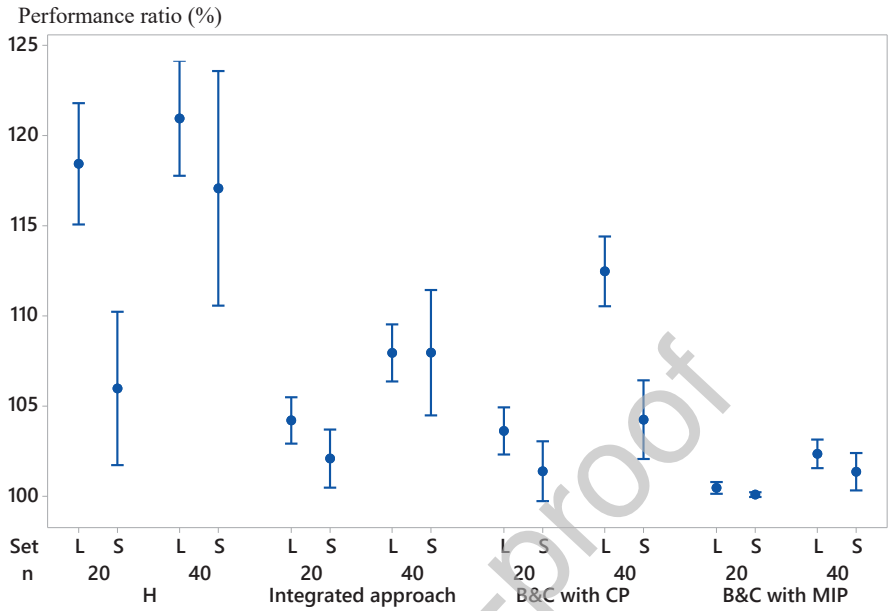

Figure 8: 95% confidence intervals for the mean performance ratio as a function of problem size and set

are sparse, the structure of the test instances limits the success of the local search operator. For small instances, swapping items or any another intensification operator does not decrease the objective function value. These $\mathcal{S}/\mathcal{L}$ related inferences/observations confirm that, for these test instances, the search focuses not only on the bin packing/cutting aspect of the problem but also on its scheduling aspect (which is driven by the due dates' distribution).

7.4. Results for Large Sized Instances

Table 4 summarizes the results for $n = 60$, 80 and 100. It reports the same information as Table 2 except that the allocated run times are t_1 =600, t_2 = 3600 and t_3 =14400 seconds. Again, when cold started, B&C using MIP fails to identify a good-quality feasible solution within the allocated runtime. In addition, it doesn't prove the optimality of the incumbent of any of the tested instances. Therefore, the statistic η is omitted; so is the runtime's statistic when CPLEX proves the optimality of its incumbent.

The analysis of Table 4 suggests that as n increases, H maintains the same level of solution quality while CPLEX starts getting stuck around its initial solution. This is most likely because of a larger search space.

Increasing the runtime enhances the performance of the three approaches. In addition, initializing B&C using MIP with H's solution makes CPLEX improve that solution. Paired t-tests confirm that increasing the runtime decreases both the mean weighted earliness tardiness and the mean performance ratio for the

Table 4: Average performance ratios (%) of all proposed approaches, number of times B&C with MIP obtains the best bound, and the average run times of the warm-up heuristic for large sized instances

		Performance ratio (%)										Number of times	Run time (s)
			Integrated appr. for time limit $t(s)$			B&C with CP for time limit $t(s)$			B&C with MIP for time limit $t(s)$				
n	Cat.	Heur.	600	3600	14400	600	3600	14400	600	3600	14400	$u_2 = \underline{u}$	RT_H
60	1	119.8	113.1	107.0	105.6	119.7	117.2	115.2	110.4	103.7	100.4	9	8
	2	126.7	118.9	115.2	113.9	120.0	114.7	108.8	103.6	101.1	100.1	9	125
	3	114.9	113.3	110.3	107.8	114.8	114.6	114.0	108.4	103.5	100.3	7	37
	4	125.2	121.1	120.0	118.7	122.4	114.9	108.9	106.2	102.1	100.9	8	117
	5	115.4	109.4	106.0	103.9	115.4	115.0	112.1	108.8	104.2	101.7	5	21
	6	120.2	116.5	116.0	114.7	116.6	114.1	107.8	101.7	100.6	100.0	10	90
	7	127.7	117.6	111.8	107.6	127.5	125.9	122.3	108.9	102.9	100.3	8	12
	8	133.2	122.5	113.5	108.5	132.3	129.8	123.2	107.9	104.4	100.0	10	9
	9	103.5	103.1	102.5	101.2	103.3	102.7	101.8	101.8	101.0	100.5	5	12
	10	116.8	116.4	114.9	113.1	116.4	116.0	113.8	106.7	103.2	100.0	10	89
Overall		120.3	115.2	111.7	109.5	118.8	116.5	112.8	106.4	102.7	100.4	81	52
80	1	114.2	112.2	106.7	103.4	114.2	113.9	113.2	110.2	105.4	100.7	8	11
	2	115.1	113.8	113.4	112.2	113.9	111.9	111.4	109.3	102.2	100.0	10	206
	3	116.2	115.6	114.1	110.9	116.2	116.1	116.0	110.8	104.8	100.0	10	76
	4	117.8	116.7	116.3	116.0	115.6	113.6	112.6	106.8	103.2	100.0	10	223
	5	111.6	110.3	107.6	105.1	111.6	111.6	111.2	108.2	105.7	101.6	6	39
	6	123.2	118.9	117.0	116.5	120.5	118.8	117.7	105.7	101.2	100.0	9	211
	7	129.3	123.1	116.5	110.6	129.2	128.6	126.7	120.9	111.5	100.0	10	19
	8	125.2	121.5	114.5	109.0	125.2	125.2	124.3	111.8	107.7	100.0	10	13
	9	105.6	105.5	104.9	104.1	105.3	105.1	104.1	103.8	102.1	100.1	8	22
	10	109.3	108.8	108.3	107.5	108.8	108.8	108.5	107.0	102.3	100.1	9	214
Overall		116.8	114.6	111.9	109.5	116.1	115.4	114.6	109.4	104.6	100.2	90	103
100	1	116.4	115.6	112.9	107.7	116.4	116.4	116.2	115.1	109.4	101.0	8	18
	2	111.7	110.8	110.0	109.9	110.2	109.8	109.8	105.9	103.5	100.0	9	270
	3	108.1	107.9	107.7	107.0	108.1	108.0	108.0	106.4	103.6	100.1	9	126
	4	109.3	107.1	107.0	106.9	108.1	108.1	107.8	108.5	103.3	100.1	9	315
	5	110.4	110.1	109.4	107.3	110.4	110.4	110.4	108.6	105.2	100.0	10	58
	6	115.2	113.8	113.5	113.4	113.6	112.7	112.1	108.8	102.0	100.0	10	242
	7	124.9	123.8	120.5	114.9	124.9	124.9	124.8	119.1	110.6	100.0	10	49
	8	121.6	119.8	116.4	109.4	121.5	121.3	120.0	112.3	106.3	100.0	10	18
	9	104.3	104.3	104.3	104.1	104.1	104.1	103.8	103.8	102.6	100.0	10	24
	10	113.1	112.9	112.8	112.8	113.0	113.0	113.0	111.3	106.3	100.0	10	297
Overall		113.5	112.6	111.4	109.3	113.0	112.9	112.6	110.0	105.3	100.1	95	142

three approaches. This is further elucidated in Figure 9, which displays the 95% confidence intervals of the mean performance ratio of all proposed approaches. It clearly puts in evidence the improvement brought up by the additional runtime.

Paired t-tests further indicate that, for equal run times, the integrated approach outperforms B&C using CP. This is expected because the integrated approach gains from the problem-specific constraints and from the broader model encompassing all aspects of JITBP. The additional information makes the search of integrated approach more efficient than the search of the B&C using CP.

Paired t-tests indicate that, for equal run times, the integrated approach is outperformed by B&C using MIP. However, in the absence of a warm-start strategy, the integrated approach is generally superior. B&C using MIP is my-

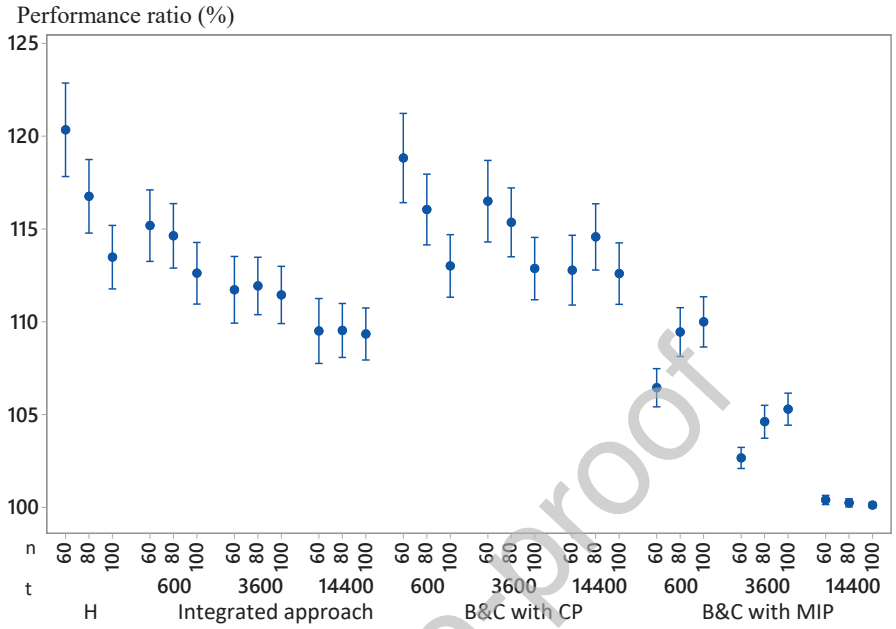

Figure 9: 95% confidence intervals for the mean performance ratios for large instances

opic to the problem's specificity; thus, works futilely on many heuristic solutions as part of the upper bounding process.

The behavior of the heuristics as a function of n is due to the deterioration of the quality of the best solution as n increases. This is, in turn, due to the sparser distribution of due dates when the instances get larger. Thus, the proposed methods cannot significantly improve H's solutions within the allocated runtime. This is further clarified in Figure 10, which illustrates the behavior of the heuristics as the number and size of the items vary. The integrated approach, for example, struggles with larger items because due dates are sparse and better solutions may need to reconfigure the whole incumbent. In the restart mode, the CP Optimizer follows a variable neighborhood search-like strategy. Therefore, escaping from a local optimum is challenging for such problems. For small items, it becomes easier for the solver to operate on the search space: Due dates are dense, and there is a large panoply of better alternative solutions that do not require the restructuring of large portions of the solution. This is clearly observed in Figure 10. Similarly, the performance of B&C using CP degrades. However, this behavior does not persist for H, which does not involve any randomness in its actions. H's performance is actually constant. The variation of H's performance ratio is due to the deterioration of the best bound. The improvements that B&C with CP brings to the solutions of H are less than their counterparts for B&C with MIP. B&C with CP stumbles at the first larger instances and its performance becomes worse than that of the inte-

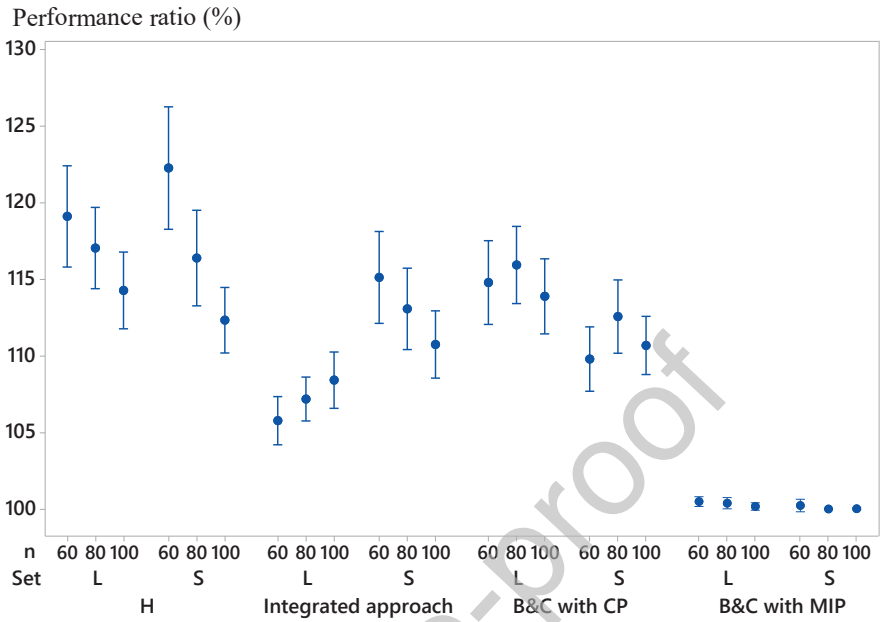

Figure 10: 95% confidence intervals for the mean performance ratio as a function of problem size

grated approach. It seems unable to cope with the size of the search space. In summary, B&C with MIP is the best approach for most large-sized instances.

The observed runtime of H is both category and size dependent, as elucidated by Figure 11. Its behavior is quite natural. Instances of set $\mathcal{S}$ need more time than instances of set $\mathcal{L}$. They have many small items. Therefore, the local search of H has more alternatives for reinserting an item; consequently, it undertakes, in general, more iterations. It calls `PACK` more often, and `PACK` may exhaust its time limit without identifying a feasible solution.

The mean runtime of H does not necessarily increase as a function of n because H does not call `PACK` when it fails to improve the scheduling solution (i.e., TWET of filled bins). In fact, H solves a MIP. When MIP is infeasible, H concludes that there is no better schedule and omits calling `PACK`. Thus, its runtime tends to increase because it gets more scheduling solutions improved during the search.

7.5. Remark

Scheduling the cutting of the bins on a single machine is not restrictive. Indeed, using multiple cutters for the purpose will only slightly alter constraints (16), (17), and (19). For a parallel machine environment with η machines, constraints (16) will apply for the first bin assigned to every machine in lieu to the first bin only. Constraint (17) will apply only for pairs of bins cut on the same cutter. Finally, constraint (19) will use CP functions that ensure

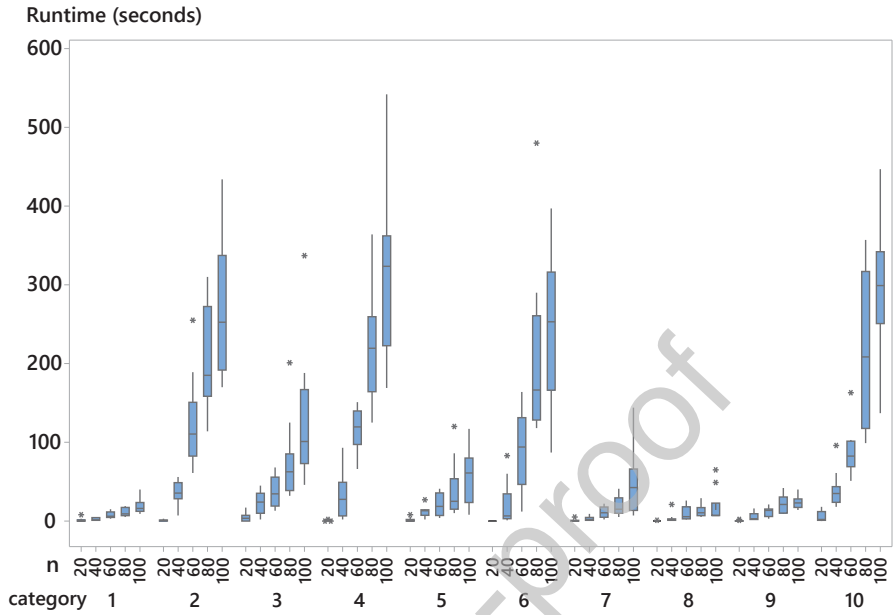

Figure 11: Box plot of the runtime of warm-start heuristic as a function of problem size and category

the no-overlap of the processing of bins assigned to a same cutter and the no duplicate assignment of a bin to more than one cutter. Similarly, for a flow shop environment with η consecutive machines, constraints (16) will apply for the completion time of first bin in every stage. Constraint (17) will apply for every stage. Finally, constraint (19) will use CP functions that ensure the no-overlap of the cutting of a bin on two consecutive stages in addition to the no-overlap of the cutting of two bins on a same stage.

8. Conclusion

This paper introduces a new discrete optimization problem that combines two $\mathcal{NP}$-hard problems: the two-dimensional bin packing and the total weighted earliness tardiness single machine scheduling. Each of these two problems is particularly difficult. The combined nature of the problem is further complicated by the interdependence of its two components. They interact either cooperatively or competitively depending on the problem's parameters. The problem considers the guillotine cutting of rectangular items from rectangular bins. Cutting an item earlier or later than its due date induces lateness penalties. The objective is to minimize the total weighted earliness and tardiness of the items. The problem has a wide panoply of interesting applications in the areas of fleet planning and logistics; in particular, in rescue planning and defense operations. It is herein formulated in a novel way that models guillotine cuts using a constant

number of regions. Meanwhile, the problem considers dynamic bins' processing times. In fact, the processing time of a bin is a dynamic function of the bin's content, as opposed to the similar state-of-the-art examples that assume an artificial constant cutting time.

This paper tackles the problem using a competitive integrated constraint programming approach and two branch-and-check decomposition based approaches. The first models the master problem as a mixed-integer program while the second models it as a constraint program. The three approaches react differently to issues that emerge from the interdependence of the two hard problems. The computational investigation reveals that the decomposition approach whose master problem is modeled as a mixed integer program is preferred to the integrated constraint programming based approach. In turn, this latter outperforms the decomposition approach whose master problem is modeled as a constraint program. The computational results further highlight the important role of warm-start heuristics in the search process of advanced solution techniques.

The ultimate goal of any future research is the development of tight / time-efficient dual bounds and of exact approaches. Albeit our approaches solve small-sized problems to optimality, their computational time (as for most approaches addressing problems with a weighted earliness-tardiness cost) grows rapidly for large and difficult instances. Loose dual bounds are intrinsic to such problems. A content-based bin's processing time further widens the duality gap; increasing the challenges imposed by the multiple aspects of the problem, and emphasizing the need for its investigation.

Other extensions of this research include the design of alternative warm-start heuristics and of adaptive enumerative techniques, such as evolutionary algorithms and meta-heuristics, to the problem. Another promising research direction is the development of hybrid approaches that benefit from the complementary strengths of Artificial Intelligence and Operations Research. These approaches should focus on effective diversification (rather than intensification) mechanisms and on ways to fathom the largest number of infeasible/dominated solutions from the search space. In addition, this problem can be extended to more complex manufacturing set ups, to cutting problems with irregular shapes and with constraints emanating from the apparel manufacturing, or to three-dimensional variable-sized variable-cost bin packing problems that arise in transportation.

References

[1] Abdul-Razaq, T.S., Potts, C.N., 1988. Dynamic programming state-space relaxation for single-machine scheduling. Journal of the Operational Research Society 39, 141–152.

[2] Alves, C., Clautiaux, F., de Carvalho, J.V., Rietz, J., 2016. Dual-Feasible Functions for Integer Programming and Combinatorial Optimization: Basics, Extensions and Applications. EURO Advanced Tutorials on Operational Research, Springer International Publishing.

[3] Arbib, C., Felici, G., Servilio, M., 2019. Common operation scheduling with general processing times: A branch-and-cut algorithm to minimize the weighted number of tardy jobs. Omega 84, 18–30.

[4] Arbib, C., Marinelli, F., 2014. On cutting stock with due dates. Omega 46, 11–20.

[5] Arbib, C., Marinelli, F., 2017. Maximum lateness minimization in one-dimensional bin packing. Omega 68, 76–84.

[6] Beck, J.C., 2010. Checking-up on branch-and-check, in: Cohen, D. (Ed.), Principles and Practice of Constraint Programming – CP 2010, Springer Berlin Heidelberg, Berlin, Heidelberg. pp. 84–98.

[7] Bennell, J.A., Soon Lee, L., Potts, C.N., 2013. A genetic algorithm for two-dimensional bin packing with due dates. International Journal of Production Economics 145, 547–560.

[8] Berkey, J., Wang, P., 1987. Two-dimensional finite bin-packing algorithms. Journal of the Operational Research Society 38, 423–429.

[9] Braga, N., Alves, C., Macedo, R., Carvalho, J.V.D., 2016. Combined cutting stock and scheduling: a matheuristic approach. International Journal of Innovative Computing and Applications 7, 135–146.

[10] Chircop, P., Surendonk, T., 2018. Constraint programming heuristics and software tools for amphibious embarkation planning. Journal of Defense Modeling and Simulation: Applications, Methodology, Technology , 1–22.

[11] Cote, J., Gendreau, M., Potvin, J., 2014. An exact algorithm for the two-dimensional orthogonal packing problem with unloading constraints. Operations Research 62, 1126–1141.

[12] Cote, J.F., Guastaroba, G., Speranza, M.G., 2017. The value of integrating loading and routing. European Journal of Operational Research 257, 89–105.

[13] Enayaty-Ahangar, F., Rainwater, C.E., Sharkey, T.C., 2019. A logic-based decomposition approach for multi-period network interdiction models. Omega 87, 71–85.

[14] Fekete, S.P., Schepers, J., 2004. A general framework for bounds for higher-dimensional orthogonal packing problems. Mathematical Methods of Operations Research 60, 311–329.

[15] Grossmann, I.E., 2002. Review of nonlinear mixed-integer and disjunctive programming techniques. Optimization and Engineering 3, 227–252.

[16] Hendry, L.C., Fok, K.K., Shek, K.W., 1996. A cutting stock and scheduling problem in the copper industry. Journal of the Operational Research Society 47, 38–47.

[17] Hooker, J., 2007. Planning and scheduling by logic-based benders decomposition. Operations Research 55, 588–602.

[18] Hooker, J.N., 2002. Logic, optimization, and constraint programming. INFORMS Journal on Computing 14, 295–321.

[19] Iori, M., Martello, S., 2013. An annotated bibliography of combined routing and loading problems. Yugoslav Journal of Operations Research 23, 311–326.

[20] Kellerer, H., Rustogi, K., Strusevich, V.A., 2020. A fast fptas for single machine scheduling problem of minimizing total weighted earliness and tardiness about a large common due date. Omega 90, 101992.

[21] Laborie, P., 2009. Ibm ilog cp optimizer for detailed scheduling illustrated on three problems, in: van Hoeve, W.J., Hooker, J.N. (Eds.), Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems, Springer Berlin Heidelberg, Berlin, Heidelberg. pp. 148–162.

[22] Li, C., Gong, L., Luo, Z., Lim, A., 2019. A branch-and-price-and-cut algorithm for a pickup and delivery problem in retailing. Omega 89, 71–91.

[23] Li, S., 1996. Multi-job cutting stock problem with due dates and release dates. The Journal of the Operational Research Society 47, 490–510.

[24] Liaw, C.F., 1999. A branch-and-bound algorithm for the single machine earliness and tardiness scheduling problem. Computers & Operations Research 26, 679–693.

[25] Lodi, A., Martello, S., Monaci, M., Vigo, D., 2014. Two-dimensional bin packing problems, in: Paschos, V.Th. (Ed.), Paradigms of Combinatorial Optimization: Problems and New Approaches. 2 ed.. Wiley-ISTE, Hoboken, NJ, USA, pp. 107–129.

[26] López-Camacho, E., Terashima-Marin, H., Ross, P., Ochoa, G., 2014. A unified hyper-heuristic framework for solving bin packing problems. Expert Systems with Applications 41, 6876–6889.

[27] Marinelli, F., Pizzuti, F., 2018. A sequential value correction heuristic for a bi-objective two-dimensional bin-packing. Electronic Notes in Discrete Mathematics 64, 25–34.

[28] Martello, S., Pisinger, D., Vigo, D., Boef, E.D., Korst, J., 2007. Algorithm 864: General and robot-packable variants of the three-dimensional bin packing problem. ACM Trans. Math. Softw. 33, 7es.

[29] M'Hallah, R., 2007. Minimizing total earliness and tardiness on a single machine using a hybrid heuristic. Computers & Operations Research 34, 3126–3142.

[30] M'Hallah, R., Alhajraf, A., 2016. Ant colony systems for the single-machine total weighted earliness tardiness scheduling problem. Journal of Scheduling 19, 191–205.

[31] Müller-Hannemann, M., Sonnikow, A., 2009. Non-approximability of just-in-time scheduling. Journal of Scheduling 12, 555–562.

[32] Pisinger, D., Sigurd, M., 2007. Using decomposition techniques and constraint programming for solving the two-dimensional bin-packing problem. INFORMS Journal on Computing 19, 36–51.

[33] Polyakovskiy, S., Makarowsky, A., M'Hallah, R., 2017. Just-in-time batch scheduling problem with two-dimensional bin packing constraints, in: Proceedings of the Genetic and Evolutionary Computation Conference 2017, ACM, New York, NY, USA. pp. 1005–1012.

[34] Polyakovskiy, S., M'Hallah, R., 2011. An intelligent framework to online bin packing in a just-in-time environment, in: Modern Approaches in Applied Intelligence. Springer. volume 6704 of *LNCS*, pp. 226–236.

[35] Polyakovskiy, S., M'Hallah, R., 2018. A hybrid feasibility constraints-guided search to the two-dimensional bin packing problem with due dates. European Journal of Operational Research 266, 321–328.

[36] Polyakovskiy, S., Thiruvady, D., M'Hallah, R., 2020. Just-in-time batch scheduling subject to batch size, in: Proceedings of the 2020 Genetic and Evolutionary Computation Conference, ACM, New York, NY, USA. pp. 228–235.

[37] Polyakovsky, S., M'Hallah, R., 2009. An agent-based approach to the two-dimensional guillotine bin packing problem. European Journal of Operational Research 192, 767–781.

[38] Reinertsen, H., Vossen, T.W., 2010. The one-dimensional cutting stock problem with due dates. European Journal of Operational Research 201, 701–711.

[39] Rocholl, J., Mönch, L., 2018. Hybrid algorithms for the earliness-tardiness single-machine multiple orders per job scheduling problem with a common due date. RAIRO-Operations Research 52, 1329–1350.

[40] Ronconi, D.A.P., Kawamura, M.A.S., 2010. The single machine earliness and tardiness scheduling problem: lower bounds and a branch-and-bound algorithm. Computational & Applied Mathematics 29, 107–124.

[41] Roshanaei, V., Booth, K.E., Aleman, D.M., Urbach, D.R., Beck, J.C., 2020. Branch-and-check methods for multi-level operating room planning and scheduling. International Journal of Production Economics 220, 107433.

[42] Sim, K., Hart, E., Paechter, B., 2015. A lifelong learning hyper-heuristic method for bin packing. Evolutionary Computation 23, 37–67.

[43] Tanaka, S., Fujikuma, S., 2012. A dynamic-programming-based exact algorithm for general single-machine scheduling with machine idle time. Journal of Scheduling 15, 347–361.

[44] Vega-Mejia, C., Montoya-Torres, J., Islam, S., 2019. A nonlinear optimization model for the balanced vehicle routing problem with loading constraints. International Transactions in Operational Research 26, 794–825.

[45] Verstichel, J., Kinable, J., De Causmaecker, P., Vanden Berghe, G., 2015. A Combinatorial Benders' decomposition for the lock scheduling problem. Computers & Operations Research 54, 117–128.

[46] Wan, L., Yuan, J., 2013. Single-machine scheduling to minimize the total earliness and tardiness is strongly NP-hard. Operations Research Letters 41, 363–365.

[47] Yuce, B., Fruggiero, F., Packianather, M., Pham, D., Mastrocinque, E., Lambiase, A., Fera, M., 2017. Hybrid genetic bees algorithm applied to single machine scheduling with earliness and tardiness penalties. Computers & Industrial Engineering 113, 842–858.

Credit Author Statement

Sergey Polyakovskiy: Conceptualization, Methodology, Validation, Formal analysis, Investigation, Resources, Writing - Review & Editing, Software, Project administration

Rym M'Hallah: Conceptualization, Methodology, Validation, Formal Analysis, Investigation, Data Curation, Writing - Review & Editing, Visualization