A Lookahead Matheuristic for the Unweighed Variable-Sized Two-dimensional Bin Packing Problem

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

[a]*School of Information Technology, Deakin University, Geelong 3216, Australia*
[b]*Department of Engineering, Faculty of Natural, Mathematical & Engineering Sciences, King's College London, UK*

Abstract

The unweighed oriented variable-sized two-dimensional guillotine bin packing problem consists in packing without overlap small rectangular items into large non-identical rectangular bins, with the items obtained via guillotine cuts. It minimizes the waste of the used bins. It is herein approximately solved using a hybrid matheuristic that applies a sequence of low-level mixed-integer programs that reserve space for unpacked items and that are guided by feasibility constraints and by upper bounds on the objective function. The embedded constraints constitute a lookahead mechanism that prohibits the investigation of infeasible directions and constrains the search to improving ones. The matheuristic further employs high-level diversification and intensification mechanisms. The diversification incorporates a sequential value correction algorithm that tags a pseudo-price to each item to govern the fitness functions of mixed integer programs and subsequently their solution construction process. The intensification is a local search that investigates the neighbourhood of promising solutions. The extensive computational experiments provide evidence of the good performance of the proposed matheuristic. For the variable-sized bin packing benchmark instances, the matheuristic matches and improves 90.8% of the upper bounds. For the single bin-size bin packing benchmark instances, the matheuristic further proves the optimality of 82.6% upper bounds while it matches 14.4% and improves 2.6% existing bounds.

Keywords: Cutting; Variable-Sized Bin Packing; Matheuristic; Lookahead Search; Feasibility Constraints

1. Introduction

This paper addresses the unweighed oriented two-dimensional guillotine bin packing problem with variable bin sizes (VS2BP). This problem consists in

*Corresponding author.
Email addresses: sergey.polyakovskiy@deakin.edu.au (Sergey Polyakovskiy), rym.mhallah@kcl.ac.uk (Rym M'Hallah)

packing without overlap a set $I = \{1, \dots, n\}$ of n distinct rectangular items into large non-identical rectangular bins with the items obtained via guillotine edge to edge cuts. An item $i \in I$ of size $a_i = l_i w_i$ is characterised by its length $l_i \in \mathbb{N}_{>0}$ and its width $w_i \in \mathbb{N}_{>0}$, where $\mathbb{N}_{>0}$ is the set of strictly positive integers. There are m types of bins. Each bin type $t \in T$, $T = \{1, \dots, m\}$, is specified by its length $L_t \in \mathbb{N}_{>0}$, weight $W_t \in \mathbb{N}_{>0}$, and either its size $A_t = L_t W_t$ or its cost that is proportional to its size. There are at least n bins of type $t \in T$. Any item $i \in I$ can fit in at least one bin type, and any bin type $t \in T$ can fit at least one item. The problem searches for a feasible packing that minimizes the total packing cost of the items. This total cost is simply the total area of the bins selected to pack the items. Equivalently, the problem maximizes the packing density, known also as the utilization of the bin, defined as the ratio of the sum of the area of the items to the sum of the area of the used bins.

The unweighed guillotine VS2BP is academically challenging. It is an extension of the one-dimensional variant, which is in turn strongly NP-hard [8]. Thus, it is unlikely that any efficient algorithm solves medium or large-sized instances of VS2BP to optimality in a reasonable time. Yet, many problems -including scheduling, cutting, and maintenance- are equivalent to VS2BP [9, 28] or contain VS2BP as a subproblem. For example, glass cutting becomes a VS2BP in the presence of defects [14, 22]. The glass panel is divided with the objective of removing the defect. There are multiple ways of removing a defect; each of them resulting in different variable sized bins and involving the resolution of a VS2BP. In addition to its academic pertinence, VS2BP arises in numerous industrial cases such as biscuit, wood, plastic, metal and paper manufacturing, newspaper paging, telecommunication, shipping and transportation. For example, in the paper industry, rolls of various dimensions are cut into different paper sizes. In the apparel industry, patterns of different sizes of a garment are laid on rectangles, which in turn are cut from rolls of cloth of equal width but different lengths. In the delivery industry, boxes are assigned to a heterogeneous fleet of vehicles. In marble cutting, rectangles are cut from different sized slices of marble stones. In newspaper paging, advertisements are allocated rectangular zones of pages and individual advertisements are then fitted in these zones.

The literature on VS2BP is rather limited. For the non-guillotine variant of the problem, Pisinger and Sigurd [23] adapted a column generation approach, and opted for an exact branch-and-price algorithm. Liu et al. [15] developed two dynamic programming-based heuristics that aggregate the state space to limit the storage requirements. Alvarez-Valdes et al. [1] presented a GRASP algorithm for the (un)weighed two- and three-dimensional multiple bin-size bin packing problems. They then applied path relinking to combine the best solutions obtained in the iterative process. They also proposed two simple lower bounds that improved those of Pisinger and Sigurd [23]. Wei et al. [27] proposed a goal-driven approach.

Ortmann et al. [21] considered the unweighed oriented guillotine version of VS2BP. They developed a two-stage heuristic that packs items into strips and strips into large bins. It then tries to reposition items from larger bins into smaller ones with the objective of increasing the filling ratio. Their results were improved by Hong et al. [12] who also considered the oriented case of VS2BP. They dotted a mixed bin packing heuristic that fills single bins with a backtracking capability. They then hybridized it with an iterative simulated annealing and binary search. Their method is hereafter denoted as SA. To the best of the authors' knowledge, Ortmann et al. [21] and Hong et al. [12] are the only references dealing with the same problem investigated in this paper.

VS2BP has a huge solution space with many alternative solutions whose packing configurations are symmetric. Consequently, exact approaches generally fail to solve its medium and large sized instances unless they employ symmetry breaking constraints. In the presence of such constraints, exact approaches such as mixed-integer programming (MIP), constraint programming (CP) and decomposition techniques (e.g. column generation and Benders decomposition) may find optima for instances with up to 100 items. However, they may neither find the optima for instances with as few as 20 items nor guarantee a sufficiently good convergence to a global optimum.

VS2BP is an extension of the classical two-dimensional bin packing problem (2BP) [16]. Yet, the difficulty of 2BP restricted solution approaches to approximate ones that varied from simple best/first fit algorithms [6] to meta-heuristics [16] to hyper-heuristics [18] to multi-agent based algorithms [25] to a combination of CP and iterative packing heuristics [24]. Obviously, any heuristic that solves VS2BP solves 2BP, but not vice versa. In fact, VS2BP is harder than 2BP. Its larger number of variables and of constraints augment the problem's complexity. Subsequently, fewer VS2BP instances can be solved to optimality, and larger instances are much harder to solve than smaller ones. Thus, it is important to design an effective approximate approach for VS2BP. For this purpose, this paper proposes a matheuristic.

Matheuristics hybridize exact solution techniques with meta-heuristics. Their wider spectrum of application is due not only to the advance of mathematical solvers and computing technologies [20] but also to their success in identifying (near-) global optima in reduced run times [3]. In general, they limit the role of their mathematical programming component to iteratively solving a relaxed model or to improving an incumbent or to tackling one of the many aspects of the problem. This limitation is mainly imposed by the techniques employed by off-the-shelf solvers –such as branch and bound–. Such techniques explore the neighborhood of solutions obtained by the linear programming relaxation [14]. Thus, they tend to obtain integer solutions at leaf nodes; an aspect that makes them very time and storage consuming. Even in the presence of strong bounds / pruning rules, off-the-shelf solvers may not prove optimality or converge to a global optimum on large/difficult instances [14].

The proposed matheuristic (MH) avoids some of the limitations of MIP solvers in general and of the model of VS2BP in particular. It obtains partial solutions by iteratively pre-assigning some of the items, locks them, bounds the value of the objective function, augments the model with feasibility constraints, and complements solutions using a different mathematical program. It then explores the neighborhood of feasible integer solutions while restricting the search to improving directions. It iteratively imposes stricter upper bounds on the objective function value. That is, MH embeds its mathematical models within a heuristic that uses a proximity search [26] like concept. Yet, MH differs from the matheuristic of Kollsker and Malaguti [13] for the LEGO two-dimensional construction problem. Their matheuristic uses a constructive heuristic to pack large elements into regular patterns and the mathematical models to fit the small parts and ensure the stability of the solution. That is, as in many of the available matheuristics, they feed an initial solution to a modified simpler version of their mathematical model to address a component of their problem.

Because solving an instance of VS2BP is not trivial (i.e., every item has a large number of alternative positions within the bins, and many solutions have equal costs), MH enhances the chances of converging to a good-quality feasible solution to VS2BP within a reasonable time by implementing three strategies to its packing procedures. The **first** strategy reserves a free space for all unpacked items while it explores all available free regions and alternative cuts to extract the packed items. This feature distinguishes the proposed packing procedures from most existing heuristics, which pack items sequentially into bins [16]. Even though it enlarges the search space and the size of the MIP, reserving free space dots the packing procedure with a futuristic vision. It makes the packing procedure consider only potentially feasible packing alternatives; thus, makes the packing procedure less greedy. In addition, it decreases the number of iterations necessary to pack all items at the cost of a slightly higher computational effort per cycle. The **second** strategy augments the basic MIP model with a set of feasibility constraints that prohibit partial (feasible) solutions that would lead to infeasible solutions in future cycles. These constraints constitute a lookahead mechanism that directs the search towards a feasible packing and enables current decisions to account for their impact on future ones. It prunes infeasible parts of the search space. The **third** strategy imposes an upper bound on the total packing cost. It forbids the use of new bins when the sum of their costs and the cost of already used bins exceeds the bound. Along with the second strategy, it limits the number of candidate solutions. In summary, MH uses a fast constructive heuristic and limits the MIP search space by (i) using bounds on the objective function value; (ii) using regions to pack items thus limiting the choices of positions for every item and reducing the symmetry embedded in the problem, and (iii) by reinforcing improving search directions.

This paper is organized as follows. Section 2 presents the three guillotine MIP-based packing procedures of MH. Section 3 details the lower and upper levels of MH detailing how the three packing procedures are used. Section 4 describes the computational results. Finally, Section 5 concludes the paper and presents future directions of research.

2. Guillotine MIP-based packing procedures

Section 2.1 describes the feasibility constraints used as a lookahead mechanism within MIP whereas Sections 2.2-2.4 detail the principles of three MIP-based guillotine packing procedures that approximately solve VS2BP.

2.1. Feasibility Constraints

In multi-dimensional bin packing, dual feasible functions (DFFs) are known to yield tight lower bounds on the minimal number of bins required to orthogonally pack a given set of oriented items [2]. When applied to a single region (which may be a bin), DFFs can detect the infeasibility of packing the items into that region. When DFFs detect an infeasible assignment, MH rules out the packing of this subset into this region from further examination.

A DFF is a function $\varphi : [0,1] \to [0,1]$ such that $\sum_{q\in Q} q \leq 1 \Rightarrow \sum_{q\in Q} \varphi(q) \leq 1$ holds for any set $Q \subset \mathbb{R}_{\geq 0}$ of numbers. Let (φ_a, φ_b) be a pair of two DFFs. Then $(\varphi_a(l_i/L'), \varphi_b(w_i/W')) \in (0,1]^2$ represents the transformed dimensions of item $i \in I$ calculated from its original dimensions (l_i, w_i) with regard to a single two-dimensional region (L', W'). A feasible packing into (L', W') exists if and only if the sum of the areas of the modified items does not exceed 1; that is, when the following constraint holds

$$\sum_{i\in I} \varphi_a\left(\frac{l_i}{L'}\right)\varphi_b\left(\frac{w_i}{W'}\right) \leq 1. \tag{1}$$

Various combinations of DFFs may yield different modified items. Let F_k be a set of distinct combinations of two DFFs, φ_a and φ_b, and let $k \in K$ be a target region (or a bin) for packing of item $i \in I$, and K a set of regions. Then $\lambda_{ik}^f \in [0,1]$ is a transformed area of item i obtained via the combination $f \in F_k$ for region k. Applying the set F_k, one can derive a set of valid feasibility inequality constraints from Eq. (1). A feasibility constraint has the form

$$\sum_{i\in I} \lambda_{ik}^f x_{ik} \leq 1, \;\; f \in F_k,\, k \in K, \tag{2}$$

where the binary decision variable $x_{ik} = 1$ if item i is assigned to region k and 0 otherwise.

Some of the $|F_k|$ constraints of inequality (2) may be redundant. A constraint f, $f \in F_k$, is redundant if either $\sum_{i\in I} \lambda_{ik}^f \leq 1$ or there exists f', $f' \in F_k$, $f \neq f'$, such that $\lambda_{ik}^f \leq \lambda_{ik}^{f'}$ for all $i \in I$. MH ignores redundant constraints. In fact, to maintain a reasonable computation time, it only augments its mathematical models with the strongest f_{max} non-redundant constraints for each region $k \in K$; i.e., with those that maximize the values of $\sum_{i\in I} \lambda_{ik}^f$, $f \in F_k$, $|F_k| \leq f_{max}$. To build the set F_k, MH applies the step functions proposed by Fekete and Schepers [11] for 2BP, in particular $u^{(1)}$, $U^{(\epsilon)}$, and $\phi^{(\epsilon)}$.

MH does not enumerate all possible combinations of DFFs for each region that the algorithm may encounter during the search. Despite its constant run time, frequently computing λ_{ik}^f for a given region $k \in K$ and item $i \in I$ increases the overall computational time considerably. To reduce its runtime, MH partitions the two-dimensional space $(\overline{L}, \overline{W})$ into $\mu \times \mu$ zones, where each zone is associated with a pair of indices (μ_l, μ_w), where μ is an integer number of partitions of $\overline{L} = \max_{t\in T}\{L_t\}$ and $\overline{W} = \max_{t\in T}\{W_t\}$ such that $1 \leq \mu \leq \min\{\overline{L}, \overline{W}\}$.

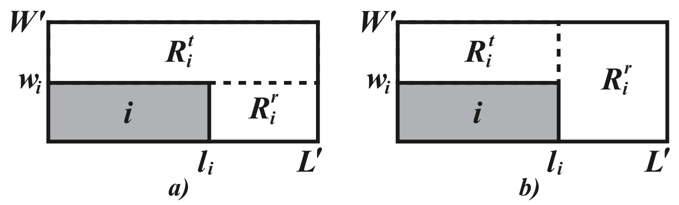

Figure 1: Illustrating the two cutting patterns resulting from a different sequence of cuts. The first cut is horizontal in pattern α (on the left) and vertical in pattern β (on the right).

Based on this partition, zone (μ_l, μ_w), $\mu_l = 0, \ldots, \mu - 1$ and $\mu_w = 0, \ldots, \mu - 1$, corresponds to a subset $K_{\mu_l,\mu_w} \in K$ of regions whose dimensions fall in the range of (μ_l, μ_w); i.e.,

$$K_{\mu_l,\mu_w} = \left\{ k \in K : \left\lfloor \frac{\mu_l \overline{L}}{\mu} \right\rfloor < L_k \leq \left\lfloor \frac{(\mu_l + 1)\overline{L}}{\mu} \right\rfloor, \left\lfloor \frac{\mu_w \overline{W}}{\mu} \right\rfloor < W_k \leq \left\lfloor \frac{(\mu_w + 1)\overline{W}}{\mu} \right\rfloor \right\}.$$

Regions $k \in K$ with $L_k > \lfloor \frac{(\mu-1)\overline{L}}{\mu} \rfloor$ belong to a zone with $\mu_l = \mu - 1$ whereas those with $W_K > \lfloor \frac{(\mu-1)\overline{W}}{\mu} \rfloor$ are in a zone with $\mu_w = \mu - 1$. Thus, MH associates the regions of set K with the zones according to size. It considers each zone (μ_l, μ_w) as a potential region, and uses the upper bounds on this zone's dimensions as the input for DFFs. In this way, MH computes up to f_{max} best combinations of functions for items that fit in this zone (μ_l, μ_w). During the search, MH builds F_k for region k using the combinations of DFFs of its zone (μ_l, μ_w) such that $k \in K_{\mu_l,\mu_w}$.

2.2. The MAXSUM *packing procedure*

The first packing procedure, labeled MAXSUM, is iterative. Its iterations stop when it identifies either a feasible packing of the n items or an infeasible direction of the search. At each iteration, it feeds an MIP with a set $\overline{I}$ of already packed items and their current cost of packing $\overline{U}$ (i.e. the total size of the bins used in the partial solution), a set I' of not-yet-packed items such that $\overline{I} \cup I' = I$ and $\overline{I} \cap I' = \emptyset$, an upper bound U^* on the objective function cost, and three sets R, S, and T of available regions. Whereas T is the set of regions whose sizes correspond to those of bin types, the sets R and S consist respectively of paired and simple active regions issued from the packing of the items of $\overline{I}$. They are dynamic sets obtained as explained below and updated in each call of MIP.

In guillotine packing, an item $i \in I$ can be obtained from a new bin of type $t \in T$ or from a two-dimensional region (L', W'), $L' \geq l_i$, $W' \geq w_i$, within a bin via a sequence of horizontal and vertical cuts.

– When i is the result of the first cut made **horizontally**, the region (L', W') is split -as shown in Figure 1.a- into two new regions: $R_i^{\alpha t}$ of size $(L', W' - w_i)$ above i and $R_i^{\alpha r}$ of size $(L' - l_i, w_i)$ to its right. In this case, i is the product of an α cutting pattern.

– When i is the result of the first cut made **vertically**, the region (L', W') is cut -as depicted in Figure 1.b- into two new regions: $R_i^{\beta t}$ of size $(l_i, W' - w_i)$ at the top of i and $R_i^{\beta r}$ of size $(L' - l_i, W')$ at the right of i. In this case, i is the product of a β cutting pattern.

That is, the first cut applied to item i exclusively determines the pair of generated active regions: Either $(R_i^{\alpha t}, R_i^{\alpha r})$ or $(R_i^{\beta t}, R_i^{\beta r})$. Regardless, the selected pair has one region above i and one to the right of i, with either or both being possibly of zero size. Let $R = R^{\bullet t} \cup R^{\bullet r}$ be a set of all regions derived from the packing of items of set $\overline{I}$, where $R^{\bullet t} = \cup_{i \in \overline{I}} \{R_i^{\bullet t}\}$ (resp. $R^{\bullet r} = \cup_{i \in \overline{I}} \{R_i^{\bullet r}\}$) is a set of regions at the top (resp. at the right) of items cut via a $\bullet, \bullet = \alpha, \beta$, pattern. When item $i \in I$ fits tightly in region (L', W') (i.e., $l_i = L'$ and $w_i = W'$), all four resulting regions $R_i^{\alpha t} = R_i^{\alpha r} = R_i^{\beta t} = R_i^{\beta r}$ are of zero size; thus, are eliminated from R. In addition, when $l_i = L'$ (resp. $w_i = W'$), $R_i^{\alpha t} = R_i^{\beta t}$ (resp. $R_i^{\alpha r} = R_i^{\beta r}$) because regions $R_i^{\alpha r} = R_i^{\beta r}$ (resp. $R_i^{\alpha t} = R_i^{\beta t}$) are of zero size. Thus, a single region results from the cutting of item i. This region, labeled 'simple', is excluded from R and is attached to S. Furthermore, when no item $j \in I'$ fits $R_i^{\alpha t}$ (resp. $R_i^{\beta r}$), region $R_i^{\beta t}$ (resp. $R_i^{\alpha r}$) is redundant, and the only useful region $R_i^{\beta r}$ (resp. $R_i^{\alpha t}$) leaves R and enters S. Finally, when $R_i^{\alpha t}$, $R_i^{\alpha r}$, $R_i^{\beta t}$, and $R_i^{\beta r}$ are of non-zero size but no item may fit in these regions, the four regions are removed from R.

Let $\hat{R} \subseteq R$ denote the subset of regions available for packing in the current call of MIP and let $\hat{I} \subseteq \overline{I}$ be the subset of packed items that generated $\hat{R}$. Similarly, let $\hat{S} \subseteq S$ be a subset of currently active simple regions. Let $K = \hat{R} \cup \hat{S} \cup T$ be the set of available regions and let $K_i \subseteq K$ denote the subset of regions that fit item $i \in I'$. Subsequently, let $I'_k \subseteq I'$ designate the subset of free items that fit into region $k \in K$. Along with the above sets, `MAXSUM` feeds two parameters to MIP:

- γ_i, the modifier adjusting the assignment cost of item $i \in I$ for any region $k \in K_i$, and
- $\Omega = \sum_{i \in I'} (\gamma_i \sum_{k \in K_i} (a_i / A_k))$ the penalty paid for each item that MIP does not pack. With this cost Ω, if the solution of the model leaves some items unassigned, no feasible solution with a cost less than U* can be found.

`MAXSUM` feeds the above input to an MIP that employs three categories of decision variables.

– The **first** category consists of three sets of binary variables:

  – $x_{ik} = 1$ if item $i \in I'_k$ is assigned to region $k \in K$, and 0 otherwise;
  – $y_{ik} = 1$ if item i reserves free space in region k, and 0 otherwise; and
  – $e_i = 1$ if item i remains unassigned, and 0 otherwise.

While x_{ik} determines whether i is packed during the current call of MIP, y_{ik} allocates space that could be used to pack i during a subsequent call of MIP. In addition, x_{ik} contributes the cost of packing i into k to the objective function of MIP, while y_{ik} does not. Indeed, y_{ik} only reinforces the constraints. It is a lookahead mechanism that reserves space for the not yet packed items of I'. Indicator variable e_i signals when i is neither assigned to a bin nor reserved a space for future assignment.

– The **second** category consists of the non-negative integer decision variables z_t, $t \in T$, which determine the number of new bins of type t required in addition to the regions available in $\hat{R}$ and $\hat{S}$.

– The **third** category consists of binary decision variables v_i, $i \in \hat{I}$, that select the cutting pattern (α or β) for every packed item i and activate the respective pair of regions: $v_i = 1$ if i is the product of an α pattern and its packing creates $(R_i^{\alpha t}, R_i^{\alpha r})$, and 0 if i is the product of a β pattern resulting in $(R_i^{\beta t}, R_i^{\beta r})$.

Using these three sets of variables, we model MIP as follows.

$$\max \sum_{i\in I'} \left(\gamma_i \sum_{k\in K_i} \frac{a_i}{A_k} x_{ik} - \Omega e_i \right) \tag{3}$$

$$\text{s.t.} \quad \sum_{i\in I'_k} x_{ik} \le 1 \qquad k\in K \tag{4}$$

$$e_i + \sum_{k\in K_i} (x_{ik}+y_{ik}) = 1 \qquad i\in I' \tag{5}$$

$$\sum_{i\in I'_k} a_i\,(x_{ik}+y_{ik}) \le A_k \qquad k\in \hat{R}\cup\hat{S} \tag{6}$$

$$\sum_{i\in I'_t} a_i\,(x_{it}+y_{it}) \le A_t z_t \qquad t\in T \tag{7}$$

$$\sum_{t\in T} A_t z_t \le U^* - \overline{U} - 1 \tag{8}$$

$$\sum_{k\in\left\{R_i^{\alpha t}, R_i^{\alpha r}\right\}} \sum_{j\in I'_k} (x_{jk}+y_{jk}) \le V_i^{\alpha} v_i \qquad i\in\hat{I} \tag{9}$$

$$\sum_{k\in\left\{R_i^{\beta t}, R_i^{\beta r}\right\}} \sum_{j\in I'_k} (x_{jk}+y_{jk}) \le V_i^{\beta} (1-v_i) \qquad i\in\hat{I} \tag{10}$$

$$\sum_{i\in I'_k} \lambda_{ik}^{f} (x_{ik}+y_{ik}) \le 1 \qquad k\in\hat{R}\cup\hat{S},\ f\in F_k \tag{11}$$

$$\sum_{i\in I'_t} \lambda_{it}^{f} (x_{it}+y_{it}) \le z_t \qquad t\in T,\ f\in F_t \tag{12}$$

$$e_i \in \{0,1\} \qquad i\in I' \tag{13}$$

$$z_t \in \mathbb{Z}_{\ge 0} \qquad t\in T \tag{14}$$

$$v_i \in \{0,1\} \qquad i\in\hat{I} \tag{15}$$

$$x_{ik},\, y_{ik} \in \{0,1\} \qquad i\in I',\ k\in K_i \tag{16}$$

Eq. (3) defines the objective function of MIP. It maximizes the net profit of the solution. This net profit is the difference of the profit earned by assigning items to regions and the penalty cost of non-assigned items. Eq. (4) prohibits the packing of more than one item into region k, $k \in K$, during the current call. However, there is no constraint on the number of items that can be allocated to k during future calls. Eq. (5) either packs item $i \in I'$ to one of the regions (i.e., when $x_{ik} = 1$, $k \in K_i$) or assigns it to a region that can potentially fit i in one of the later calls (i.e., when $y_{ik} = 1$, $k \in K_i$) or marks it as unassigned (i.e., when $e_i = 1$). That is, x_{ik} and y_{ik} can't be assigned a value 1 simultaneously; differently stated, item i is assigned to at most one region in K_i. Eq. (6) ensures that the area A_k of region $k \in \hat{R} \cup \hat{S}$ is large enough to accommodate all items assigned to it. Similarly, Eq. (7) ensures that the total area $A_t z_t$ of allocated bins of type t, $t \in T$, is sufficiently large for their assigned items.

Obviously, if no bin is used (i.e., $z_t = 0$), no items are assigned to the bin. Eq. (8) requires the total size of new bins to be less than the known upper bound U^* minus the size $\overline{U}$ of previously used bins. Eqs. (9) and (10) select the cutting pattern for item $i \in \hat{I}$; thus, choose between the respective pairs of regions $(R_i^{\alpha t}, R_i^{\alpha r})$ and $(R_i^{\beta t}, R_i^{\beta r})$. When $v_i = 0$, the first applied cut for i is vertical; therefore, Eq. (9) forbids the assignment of any items into $R_i^{\alpha t}$ and $R_i^{\alpha r}$. Here, $V_i^\alpha = \sum_{k \in \{R_i^{\alpha t}, R_i^{\alpha r}\}} |I'_k|$, $i \in \hat{I}$, is a "Big-M" value used to model the disjunction. On the other hand, when $v_i = 1$, the first applied cut for i is horizontal; hence, Eq. (10) implies that no item can be assigned into $R_i^{\beta t}$ and $R_i^{\beta r}$. Similarly, $V_i^\beta = \sum_{k \in \{R_i^{\beta t}, R_i^{\beta r}\}} |I'_k|$, $i \in \hat{I}$, is a "Big-M" value required to implement the disjunction.

Eqs. (11) and (12) are feasibility constraints. They strengthen the search for a feasible packing and cut off futile partial solutions earlier during the search. They reinforce the geometric feasibility of the MIP's solution. In fact, nowhere does MIP account for the geometric relationship. For instance, Eqs. (6) and (7) overlook the geometric relationship between pairs of items assigned into the same region. Jointly with Eq. (5), Eqs. (11) and (12) constitute a part of the lookahead strategy. They rely on the values of transformed areas λ_{ik}^f pre-computed for every region $k \in K_i$, an unpacked item $i \in I'_k$, and a combination of DFFs $f \in F_k$. Eq. (11) requires that the sum of the transformed areas of items being packed in the current MIP's call (i.e., $x_{ik} = 1$) and those to be packed in future calls (i.e., $y_{ik} = 1$) in selected region k be bounded by 1. Similarly, Eq. (12) estimates the required bins imposing that, for each type $t \in T$, the sum of the transformed areas of the items be less than the number z_t of allocated bins. Even though Eqs. (11) and (12) fathom many partial solutions that eventually lead to an infeasible packing, they do not guarantee that all not-yet-packed items get a feasible position during later calls. Finally, Eqs. (16)-(15) declare the types of the decision variables. When MIP finds a positive objective function value, it returns to `MAXSUM` a feasible (partial) solution. Before calling MIP again, `MAXSUM` updates the set I' by moving the packed items from I' to $\overline{I}$. Let I" denote the set of items packed during the current MIP's call. `MAXSUM` then removes the regions that have been used in the current call (i.e., those whose $x_{ik} = 1$ for some item $i \in I'$) from $\hat{S}$ and $\hat{R}$.

- It removes from $\hat{S}$ any simple region k that has been assigned an item, and checks whether this packing has produced a pair of coupled regions or a simple region. As a result, it adds new valid region(s) to the respective set $\hat{R}$ or $\hat{S}$.
- It discards from $\hat{S}$ any simple free region k that can't fit any of the items of the updated I'.
- It examines the four regions $\left\{R_i^{\alpha t}, R_i^{\alpha r}, R_i^{\beta t}, R_i^{\beta r}\right\}$ for each item $i \in \hat{I}$.
    - When any of the four regions is assigned an item $i \in \hat{I}$, it removes the four of them from $\hat{R}$, and determines the active pair: either $(R_i^{\alpha t}, R_i^{\alpha r})$ or $(R_i^{\beta t}, R_i^{\beta r})$. It further checks whether one or both regions of the active

pair are used. In the former case, it inserts the free region into $\hat{S}$ if the free region may pack an item $i \in I'$. For example, if $R_i^{\alpha t}$ contains an assigned item and $R_i^{\alpha r}$ is empty, then the latter is a candidate for inclusion into $\hat{S}$.

- When none of the four regions are used, it proceeds depending on whether there is an item $i \in I'$ that fits into any of them.
  * It excludes the four of them from $\hat{R}$ when none fits any $i \in I'$.
  * If only one of the four regions can hold an item $i \in I'$, it moves that region to $\hat{S}$ and deletes the three others from $\hat{R}$.
  * Otherwise, it keeps the regions in $\hat{R}$.

Next, `MAXSUM` checks the validity of the new pair of coupled regions $(R_i^{\alpha t}, R_i^{\alpha r})$ and $(R_i^{\beta t}, R_i^{\beta r})$ that emanated from packing item $i \in I$". It discards all four regions when they are either empty or too small to fit any item of I', but attaches them to $\hat{R}$ when they may fit at least one item of I'. In addition, when either $R_i^{\alpha t}$ or exclusively $R_i^{\beta r}$ may pack an item in a future call, it adds this simple region to $\hat{S}$.

For any value $z_t > 0$, `MAXSUM` treats bin $t \in T$ as a single simple region. It tests whether packing item $i \in I'$ gives a pair of coupled regions or a simple region, and adds the valid region(s) to the sets $\hat{R}$ and $\hat{S}$, respectively. It assimilates any $z_t > 1$ value to $z_t = 1$, $t \in T$, because Eq. (4) restricts the assignment to region t to at most one item. Indeed, variables z estimate the remaining space and halt the packing procedure as soon as they detect an infeasible solution (subject to the bound U^*). They further help MIP to locally select and adjust the right combination of bins that are required to pack the items in I'. Even though the current solution may have several z_t values greater than 1, not all the selected bin types are utilized immediately. Indeed, the packing procedure attempts to balance its greediness (using all allocated bins at once) and its providence (gradually adding new allocated bins allowing their combinations to change as its search progresses). Therefore, it sorts the selected bin types in the ascending order of their sizes and adds the largest one to the current (partial) solution. Each next largest bin is selected with a uniform probability χ. If the bin at hand is declined, then the remaining smaller bins are discarded straightaway.

`MAXSUM` iterates as long as the MIP solver returns a feasible solution (i.e., a positive objective value). It stops when $I' = \emptyset$ and returns a new incumbent solution. However, at one of the iterations, the solver may deem MIP infeasible; implying that some of the not-yet-packed items may not be packed without surpassing the current cost U^*. In such a case, any further search is futile. Therefore, the procedure terminates immediately signaling infeasibility.

`MAXSUM` implements a common greedy packing policy: The chance to find a suitable region for smaller items is higher than for larger ones, which therefore should be positioned first. According to the assignment costs specified in Section 3.2, the packing procedure treats larger items of VS2BP as complicated. It therefore packs them earlier during the search. At the same time, using the

y_{ik} assignment variables, it plans for space for other items too; thus, it aims to increase the overall packing density of items in the resulting solution. In practice, `MAXSUM` finds high-quality solutions in a very short time and facilitates the quick improvement of the upper bound U^*, which is fed back to the algorithm.

2.3. The `MINSUM` packing procedure

The second packing procedure, labeled `MINSUM`, generally outperforms `MAXSUM` in terms of solution quality but at the cost of a larger run time. Its MIP drops the terms involving the y_{ik} assignment variables from the inequalities (5)-(7) and (9)-(12) of `MAXSUM` while it maintains all the other terms of the constraints. It employs a real-valued variable $C \in \mathbb{R}_{\geq 0}$ that corresponds to the maximum cost across all regions, where the cost associated with a region $k \in K_i$ is the total cost of the items assigned to k. It uses parameter

$$M = \textstyle\sum_{i \in I'} \gamma_i \left(\sum_{k \in K_i} \max\{a_i/A_k, \lambda_{ik}^1, \ldots, \lambda_{ik}^{|F_k|}\} \right)$$

that represents the penalty cost of an unassigned item.

The model of `MINSUM` is as follows.

$$\min C + \sum_{i \in I'} M e_i \tag{17}$$

$$\text{s.t.} \sum_{i \in I'_k} \frac{\gamma_i a_i}{A_k} x_{ik} \leq C \qquad k \in K \tag{18}$$

$$\sum_{i \in I'_k} (\gamma_i \lambda_{ik}^f) x_{ik} \leq C \qquad k \in K,\ f \in F_k \tag{19}$$

$$\text{Eqs. } (5) - (15)$$

$$C \in \mathbb{R}_{\geq 0} \tag{20}$$

$$x_{ik} \in \{0, 1\} \qquad i \in I',\ k \in K_i \tag{21}$$

Eq. (17) defines MIP's objective function, which minimizes the sum of two terms. The first term is the maximum, across the available regions of K, of the sum of the costs of the items assigned to an available region. The second term is the penalty cost of unassigned items. Both Eqs. (18) and (19) ensure that the value C is larger than or equal to the sum of the item-to-region assignment costs of the items packed in region k, for each $k \in K$. In Eq. (18), each cost is proportional to the packing density a_i/A_k of the item i assigned to region k. On the other hand, in Eq. (19), each cost is proportional to the transformed area of item i when i is packed in region k. Both a_i/A_k and λ_{ik}^f are reals in $(0, 1]$. Eqs. (5)-(15) are those of `MAXSUM`. Finally, Eqs. (20)- (21) declare the decision variables' types.

As for `MAXSUM`, the current call of MIP can either obtain a feasible solution or indicate infeasibility. When there is a feasible (partial) solution, `MINSUM` filters the unused regions and computes new ones for the next MIP's call. Among all the items assigned to a region $k \in K$ (i.e., $x_{ik} = 1$), it selects one that maximizes the item-to-region assignment cost ($\gamma_i a_i / A_k$) while it treats the other items as though they were not yet packed, keeping them in set I' for the next call. On the other hand, in the presence of an item i, $i \in I'$, whose $e_i = 1$, `MINSUM` fails to pack all items without violating the current cost U^*; therefore, the search stops.

The packing procedure of `MINSUM` is diametrically opposed to that of `MAXSUM`. Rather being greedy and trying to fill as much free space of the available regions, `MINSUM` maximizes, over all regions of set K, the gap between the total size of items (related to their packing cost) and the size of the region that holds them. This rule strives to distribute the items evenly among the available regions and avoid their overfilling in later MIP calls. Despite the tightness of the bounds imposed by constraints (11) and (12), the would be needed space for packing items in later calls is larger than the reserved space. Thus, `MINSUM` creates a gap between the bound and the estimation to increase its chance of eventually obtaining a feasible packing solution.

2.4. The `MINMAX` packing procedure

The third packing procedure, labeled `MINMAX`, succeeds in many instances in finding hidden solutions that are reached by neither `MAXSUM` nor `MINSUM`. Of course, this occurs at the cost of increased computational effort. It adds the real-valued variables $c_k \in \mathbb{R}_{\geq 0}$, which represent the largest item-to-region assignment cost for region $k \in K$ among all items $i \in I'_k$. In addition to these variables, it uses parameter $\Theta = \sum_{i \in I'} \sum_{k \in K_i} \frac{A_k}{\gamma_i a_i}$ as the penalty of a non-assigned item. `MINMAX` drops the y_{ik} decision variables from constraints (5)-(7) and (9)-(12) of `MAXSUM` but maintains all the other terms. It is given as follows.

$$\min \sum_{k \in K} c_k + \sum_{i \in I'} \Theta e_i \tag{22}$$

$$\text{s.t.}\ \frac{A_k}{\gamma_i a_i} x_{ik} \leq c_k \qquad k \in K,\ i \in I'_k \tag{23}$$

$$\text{Eqs. (5)} - \text{(15)}$$

$$c_k \in \mathbb{R}_{\geq 0} \qquad k \in K \tag{24}$$

$$x_{ik} \in \{0, 1\} \qquad i \in I',\ k \in K_i \tag{25}$$

Eq. (22) defines the objective function of `MINMAX`. It minimizes the sum of the maximal item-to-region assignment costs over all regions of set K and the penalty cost of unassigned items. Eq. (23) calculates the maximal assignment cost for each region $k \in K$: It sets c_k larger than or equal to the cost $\frac{A_k}{\gamma_i a_i}$ of item $i \in I'_k$ when i is assigned to k. Eqs. (5)-(15) are those from `MAXSUM`. Finally, Eqs. (24)-(25) declare the types of the decision variables.

As when solving `MAXSUM`, the MIP solver may identify either a feasible solution or deem the problem infeasible. In the former case, `MINMAX` uses the (partial) solution to determine a set of prospective regions for the next MIP's call. Furthermore, it chooses among all the items assigned to a region $k \in K$ (i.e., $x_{ik} = 1$) the one that gives the smallest item-to-region assignment cost

$\frac{A_k}{\gamma_i a_i}$ while it discards the assignment of the other items to k and inserts them back into the set I' for the next call. In the latter case (i.e., if there is an item $i \in I'$ such that $e_i = 1$), `MINMAX` terminates the search as its MIP fails to pack at least one item without exceeding the current cost U^*.

While `MAXSUM` focuses on maximising the sum of item-to-region assignment costs of larger items, `MINMAX` strives to find better positions for items with low packing coefficients. Because it bases its reservation of free space for such items on a "worst-case assignment", it enhances its chances of packing them during subsequent MIP calls, and eventually obtaining a feasible solution. In this sense, `MINMAX` is not as greedy as `MAXSUM`.

3. The Lookahead Matheuristic

MH is an iterative algorithm that searches for a minimal cost feasible guillotine packing of the items of I. Its search applies diversification and intensification strategies that evoke a series of MIP-based packing procedures. Section 3.1 details the solution construction process of MH. Section 3.2 describes a diversification mechanism, achieved through a sequential value correction heuristic (`SVC`).

3.1. The Solution Construction Process

MH, whose pseudo code is given in Algorithm 1, constructs a solution as follows. Its input is an instance of VS2BP, defined by I and T, and a maximal runtime limit τ. Its output is (P^*, U^*) where P^* is the incumbent solution expressed as a set of packed bins and U^* is its corresponding cost. Its initialization phase sets (P^*, U^*) to $(\emptyset, \infty)$. Its iterative phase strives to obtain a (near-)global optimal guillotine packing. It is repeated as long as the runtime of MH does not exceed τ. At each iteration, it calls `SEARCH`(I, U^*), a subroutine that attempts to build a feasible packing of the set I of items such that the packing's cost is less than the upper bound U^*. `SEARCH` returns solution (P, U), where P is either a new feasible (incumbent) solution of cost $U < U^*$ or an infeasible solution recognized by its infinite cost U. In the former case, MH updates (P^*, U^*) while in the latter case, it simply discards P.

Subroutine `SEARCH` uses the input parameters $(\widetilde{I}, \widetilde{U})$ fed from MH and three iteration limits: η, η', and η", which, respectively, constrain the total number of non-improving calls of packing procedures, and the number of times `MINSUM` and `MINMAX` are called. `SEARCH`$(\widetilde{I}, \widetilde{U})$ is looking for a feasible guillotine packing of the set $\widetilde{I}$ such that the packing's cost $\hat{U}$ is less than $\widetilde{U}$. This packing can be the result of one of the three packing procedures: `MAXSUM`, `MINSUM`, or `MINMAX`. `SEARCH` is recursive. At each recursive step, it goes through an initialization phase, an iterative phase, and an update phase.

The initialization phase of a recursive step of `SEARCH` sets its candidate solution $(\hat{P}, \hat{U})$ to $(\emptyset, \widetilde{U})$. Initially, `SEARCH` opts for `MAXSUM` as its packing procedure: `PACK`:=`MAXSUM`. It then initialises the set B of densest packed bins to the empty set; i.e., no item has yet been assigned to any bin. Next, it initializes the set $\widetilde{S}$

of simple regions to the empty set and their maximal cost $\widetilde{C}{:=}\widetilde{U}$. With $\widetilde{S} = \emptyset$, `PACK` has no constraint on its selection mechanism of bins from T. Finally, `SEARCH` initializes the counter $q = 0$, which keeps track of the maximal number of bins in the packing solutions of `PACK`.

The iterative step of `SEARCH` repeats as long as the counter η of non-improving calls to the packing procedure `PACK` does not exceed η. It calls procedure `PACK` with three parameters: $\widetilde{I}$, $\hat{U}$, and $\widetilde{S}$ to obtain the solution (P, U), where P is feasible if $U < \hat{U}$, and infeasible otherwise.

– When P is feasible, `SEARCH` updates $(\hat{P}, \hat{U})$ and decrements the counter of iterations: $\eta = \eta - 1$. In addition, because $\hat{U} \leq U^*$, P is a new global incumbent solution.

– When P is infeasible, both the set $\widetilde{S}$ and the bound $\widetilde{C}$ are updated. Because `PACK` failed to identify a feasible packing whose cost is less than $\hat{U}$, `SEARCH` restarts `PACK` from a set of pre-allocated bins to guide the search towards feasibility. In this way, `SEARCH` acts in a "conservative"manner; making slower but steadier progress towards tighter bounds. To determine this specific set of bins, it uses subroutine `PREDEFINE`, which determines a set $\widetilde{S}$ whose cost C of bins is as close as possible to $\widetilde{C}$.

  Let $\underline{C}_P$ be the cost of bins used in the infeasible solution P, $\widetilde{C}$ the maximal cost of the current $\widetilde{S}$ of simple regions. In addition, let the integer decision variable ζ_t, $t \in T$, denote the number of bins of type t that should be included in set $\widetilde{S}$ to ensure a cost lower than $\widetilde{C}$. Using these parameters and decision variable, `PREDEFINE` follows.

$$\max C = \sum_{t \in T} A_t \zeta_t \tag{26}$$

$$\text{s.t.}\ \underline{C}_P \leq C \leq \widetilde{C} - 1 \tag{27}$$

$$\zeta_t \in \mathbb{N}_{\geq 0} \qquad t \in T \tag{28}$$

  Eq. (26) maximizes C, the total cost of selected bins. Eq. (27) bounds the value of C to the discrete interval $[\underline{C}_P, \widetilde{C} - 1]$. Finally, Eq. (28) defines the type of ζ variables. When `PREDEFINE` has a feasible solution, `SEARCH` sets $\widetilde{S}$ to ζ_t bins of every type t, $t \in T$, and sets $\widetilde{C} = C$. Otherwise, `SEARCH` resets $\widetilde{S}$ to the empty set and $\widetilde{C}$ to $\hat{U}$.

Independently of the feasibility of P, `SEARCH` records the maximum number q of bins filled across all `PACK`'s solutions: $q = \max\{q, |P|\}$. Next, it revises its set B augmenting B with any densely packed bins of P. (The cardinality of B is bounded according to a policy that only retains high-quality packing patterns. That is, $|B|$ is rather small.) Having updating B, `SEARCH` recalculates the pseudo-costs of the items of $\widetilde{I}$. Altering the pseudo-costs changes the structure of solutions obtained by future calls of `PACK`. That is, they are used for diversification purposes. They are set using a sequential value correction (`SVC`) heuristic as discussed in Section 3.2.

When η' successive calls to `PACK` fail to improve the local incumbent solution and `PACK`=`MAXSUM`, `SEARCH` changes `PACK` to `MINSUM` and resets $(\widetilde{S}, \widetilde{C}) = (\emptyset, \hat{U})$. Similarly, when the incumbent solution is still not improved after η" calls and `PACK`=`MINSUM`, `SEARCH` switches `PACK` to `MINMAX` and resets $(\widetilde{S}, \widetilde{C}) = (\emptyset, \hat{U})$. (This change of packing procedure is equivalent to a diversification strategy.)

After η non-improving calls to `PACK`, if $q > 1$, `SEARCH` applies an intensification strategy that explores the neighbourhoods of the obtained solutions. It exploits the promising parts of the search space in attempt to build a feasible improving incumbent solution. Specifically, `SEARCH` iteratively fixes a highly dense bin $b \in B$, and repacks all other items. Let $\texttt{I}(b)$ denote the set of items packed in b and let $\texttt{COST}(b)$ be its cost. Then `SEARCH` calls itself recursively with parameters $(\widetilde{I} \setminus \texttt{I}(b), \hat{U} - \texttt{COST}(b))$ to obtain a solution P of cost U. When P is feasible and $U < \hat{U} - \texttt{COST}(b)$, P improves the local incumbent $\hat{P}$. Therefore, $(\hat{P}, \hat{U})$ is updated. In this way, the recursive call not only decreases the solution's cost but also enhances the algorithm's performance. It investigates the most promising parts of the search space by keeping the most successful packing patterns fixed and rebuilding the solution's parts whose packing can be significantly improved. This strategy enhances the primal bound rapidly. Because it does not spend any effort/time on improving already dense bins, it solves fewer and smaller packing sub-problems; thus, requires less runtime while obtaining faster improvements. When $q \leq 1$, `SEARCH` applies its base case, where recursive calls are prohibited.

Finally, `SEARCH` returns its output to MH. If $\hat{P} \neq \emptyset$, it returns the local incumbent $(\hat{P}, \hat{U})$; otherwise, it returns the output $(\emptyset, \infty)$ signaling the impossibility of improving the current incumbent.

3.2. Sequential Value Correction

As a diversification strategy, subroutine `SEARCH` applies `SVC` to the pseudo-costs of the items. Initially, the pseudo-cost of item $i \in I$ is $\gamma_i = 1$. `SVC` alters the pseudo-costs of items to propagate information regarding the success of previous assignments to future iterations and to facilitate the packing of new dense bins. It then randomizes and updates each pseudo-cost at every iteration to reflect the unused area in the bin where the item is currently packed. These pseudo-costs are part of the objective function that `PACK` optimizes. Altering them makes `PACK` obtain different items to bins assignments. Their unequal values allow `PACK` to randomly pack items; thus, diversify the search and overcome the highly symmetric nature of bin packing solutions; an intrinsic feature of VS2BP [17]. This diversification has been found effective in strip packing [5], in one- and two-dimensional bin packing [4, 10], and in bi-objective bin packing with due-dates where the objective is to minimize both the number of bins and the maximum lateness of items [19].

When defining the pseudo-cost γ_i of item $i \in I$, two factors are of relevance:

- the portion $\rho_i = \frac{a_i}{A_t}$ of the area of bin t that item i is using, and
- the relative size of item i with respect to the largest item in I, expressed as $\frac{a_i}{a_{max}}$ where $a_{max} = \max_{j \in I} \{a_j\}$.

Algorithm 1 Solution process as a whole in MH for VS2BP

```
    Input: A problem instance $(I, T)$, $t$, $\eta$, $\eta'$, $\eta''$

    routine MAIN($I$, $T$)
 1: initialise the global incumbent solution $(P^*, U^*)$:=$(\emptyset, \infty)$
 2: while ($\tau$ is not exceeded) do
 3:     obtain solution $(P, U)$:=SEARCH($I$, $U^*$)
 4:     if ($U < U^*$) then update the incumbent solution $(P^*, U^*)$:=$(P, U)$
 5: return the incumbent solution $(P^*, U^*)$

    subroutine SEARCH($\widetilde{I}$, $\widetilde{U}$)
 1: initialise the local incumbent solution $(\hat{P}, \hat{U})$:=$(\emptyset, \widetilde{U})$
 2: select the packing procedure PACK:=MAXSUM
 3: initialise the set of most dense packed bins $B$:=$\emptyset$
 4: initialise the set of predefined bins and their maximal cost $(\widetilde{S}, \widetilde{C})$:=$(\emptyset, \widetilde{U})$
 5: initialise the counter $q$:=0
 6: for ($counter = 1$; $counter \leq \eta$"; $counter = counter + 1$) do
 7:     obtain solution $(P, U)$:=PACK($\widetilde{I}$, $\hat{U}$, $\widetilde{S}$)
 8:     if ($U < \hat{U}$) then
 9:         update the local incumbent $(\hat{P}, \hat{U})$:=$(P, U)$
10:         set $counter = counter - 1$
11:     else
12:         update $(\widetilde{S}, \widetilde{C})$:=PREDEFINE($T$, COST($P$), $\widetilde{C}$)
13:     update $q$:=max$\{q, |P|\}$
14:     update the set $B$ with regard to the packed bins of $P$
15:     update the pseudo-costs $\gamma_i$ for every item $i \in \widetilde{I}$
16:     if (PACK=MAXSUM and $\hat{P} = \emptyset$ and $\eta < counter \leq \eta'$ then)
17:         select PACK:=MINSUM and set $(\widetilde{S}, \widetilde{C})$:=$(\emptyset, \hat{U})$
18:     if (PACK=MINSUM and $\hat{P} = \emptyset$ and $\eta' < counter \leq \eta$" then)
19:         select PACK:=MINMAX and set $(\widetilde{S}, \widetilde{C})$:=$(\emptyset, \hat{U})$
20: if ($q > 1$) then
21:     for (each bin $b \in B$) do
22:         obtain solution $(P, U)$:=SEARCH($\widetilde{I} \setminus$ I($b$), $\hat{U} -$ COST($b$))
23:         if ($U +$ COST($b$) $< \hat{U}$) then
24:             update the local incumbent $(\hat{P}, \hat{U})$:=$(P \cup \{b\}, U +$ COST($b$))
25: if ($\hat{P} \neq \emptyset$) then
26:     return the local incumbent solution $(\hat{P}, \hat{U})$
27: else
28:     return the infeasible solution $(\hat{P}, \hat{U})$:=$(\emptyset, \infty)$
```

Based on these two factors, the pseudo cost γ_i of an item $i \in I$ that has been recently positioned in a bin is reset to

$$\gamma_i = \delta_i \gamma_i + \left(1 + \frac{a_i}{a_{max}}\right)^{\varepsilon'} \frac{(1-\delta_i)}{\rho_i}. \tag{29}$$

$\varepsilon' > 1$ is a constant that makes bigger items have higher costs. δ_i is generated randomly from the uniform$[0, \bar{\rho}_i)$, where $\bar{\rho}_i$ is the average packing density of bins where item i has been placed during the last κ iterations. That is, δ_i allows items with a historically larger packing density $\bar{\rho}_i$ to inherit a larger fraction of their previous pseudo-costs. The first term of Expr. (29) reinforces the information inherited from the previous patterns. On the other hand, the second term extracts information from the current pattern. The value of γ_i increases as ρ_i decreases; thus prioritizing the joint packing of an item i with other items when i has a low packing density. In addition, packing a large item is more challenging than packing a small one. Therefore, the class of large items is given a higher priority than the class of small items. This is further reinforced via parameter ε'.

On the other hand, the pseudo cost γ_i of an item $i \in I$ that has not yet been packed is reset at a higher value. This is to increase its chances of being packed in the next call of `PACK`. Specifically, its

$$\gamma_i = \frac{1}{\bar{\rho}_i} \left(1 + \frac{a_i}{a_{max}}\right)^{\varepsilon"}, \tag{30}$$

where $\varepsilon" > \varepsilon'$.

4. Experiments and Results

The proposed MH is implemented in C# and run on a personal computer with 16GB of RAM and an i7 2.8 GHz Intel Core processor. It is tested on the benchmark set of [23]. The set has ten-classes. For classes 1-6, the item sizes l_i and w_i emanate from the same interval and the bins are squares as described by Table 1. Classes 7-10, which use bins of size $L = W = 100$, reflect a more realistic situation because their items emanate from a mixture of uniform distributions. Indeed, the items are classified into four types, and are mixed as indicated in Table 2. Uniformly distributed dimensions of items do not make problems easy to solve as it is the case for the one-dimensional case. The cost of a bin (L, W) is its area LW. The instances of the variable bin sizes data set use the same items as those for the single bin type. However, each instance has five bin types where the bin dimensions are randomly generated from the Uniform$[L/2, L]$ and Uniform$[W/2, W]$, respectively, where L and W are the instance's original bin size obtained according to the class type. Regardless of the number of bin types of the data set, each class has five problem sizes: $n = 20, 40, 60, 80,$ and 100, and ten instances per problem type.

For all instances, the performance measure is the overall utilization of the used bins. The utilization, denoted hereafter u, is the ratio of the sum of the

Table 1: Description of the benchmark instances of classes 1 to 6

Class	$l_i,\ w_i$	$L = W$
1	$[1, 10]$	10
2	$[1, 10]$	30
3	$[1, 35]$	40
4	$[1, 35]$	100
5	$[1, 100]$	100
6	$[1, 100]$	$[1, 300]$

Table 2: Description of the benchmark instances of classes 7 to 10

			Class			
Item type	l_i	w_i	7	8	9	10
1	$[\frac{2}{3}L, L]$	$[1, \frac{1}{2}W]$	70%	10%	10%	10%
2	$[1, \frac{1}{2}L]$	$[\frac{2}{3}W, W]$	10%	70%	10%	10%
3	$[\frac{1}{2}L, L]$	$[\frac{1}{2}W, W]$	10%	10%	70%	10%
4	$[1, \frac{1}{2}L]$	$[1, \frac{1}{2}W]$	10%	10%	10%	70%

areas of the n items to the sum of the areas of the used bins. To obtain this measure, we run MH for $\tau = 1$ minute and for $\tau = 10$ minutes. Let u_{MH_1} and $u_{\mathrm{MH}_{10}}$ denote the corresponding obtained utilization. Both u_{MH_1} and $u_{\mathrm{MH}_{10}}$ are compared to the existing upper bound u_{SA} of [12] obtained using their SA based heuristic within a 1-minute runtime. Running MH requires the specification of its control parameters, which are detailed in Section 4.1. Section 4.2 presents and analyzes the results of MH for the variable bin types benchmark set. Section 4.3 indicates the applicability of MH to the identical bin type benchmark set too while proving the optimality of a large number of open problems. Finally, Section 4.4 highlights the role of MH's features.

4.1. Experimental Setup

The following settings, inferred from preliminary computational investigations, provide the best trade-off between MH's solution quality and runtime. MH evokes IBM CPLEX 12.10 to solve the MIP components of its three packing procedures. The MIP solver uses a single core. It is configured such that it generates more feasible solutions while optimizing the problem, at the cost of a slower proof of optimality. Its relative optimality gap is set to 0.01 in lieu of CPLEX's default value of 10^{-6}. This is consistent with the philosophy of MH. Its `PACK` procedures favour high-quality feasible MIP solutions to optimal ones when the former are obtained faster than the latter. In addition, their MIPs rely on fitness functions that do not necessarily lead to VS2BP's optimum. Thus, the 0.01 relative optimality gap does not affect the overall quality of MH's solution. Moreover, MIP's node selection is set to prefer the node with the best progress toward integer feasibility. This setting is recommended when it is difficult to find feasible solutions or when a proof of optimality is not crucial. Furthermore, we limit the execution time of the MIPs in `PACK` subroutines to 3 seconds. This runtime limit does not affect the MIPs: Even the largest instances of the tested instances obtained a solution in less than 0.1 second. Finally, when MIP of `PACK` selects several bin types as candidates to enter the partial solution, it adds the bin of the largest size and decides on each subsequent bin with a uniform probability $\chi = 0.5$ (cf. Section 2.2). All smaller bins are immediately discarded if the bin at hand is declined.

We generate the feasibility constraints of Eq. (2) according to the algorithm of Fekete and Schepers [11] whose control parameters (p, q) are selected from $\{0.05, 0.1, 0.15, 0.2, 0.25, 0.3, 0.35, 0.4, 0.45, 0.5\}^2$. From the set of constraints generated for each entry K_{μ_l,μ_w} of the data structure of Section 2.1 (with partition $\mu = 10$), at most $f_{max} = 10$ strongest constraints are injected into MIP while weaker constraints are discarded. A larger set of constraints does not necessarily tighten the lower bound on the free space available for packing, but does increase the time needed to solve MIP.

The three iteration limits, η, η', and η", of the `SEARCH` subroutine of MH are set, respectively, to 11, 5, and 9. That is, `MAXSUM` is executed at least 5 times. When `MAXSUM` is unsuccessful, `MINSUM` makes no less than 4 attempts to find a feasible packing. When both packing procedures fail, `MINMAX` tries at least twice to solve the problem. The total number of non-improving calls of packing procedures cannot exceed 11. When tested on the benchmark set, `MINMAX` took up to 1 second of run time. Therefore, it was rarely called. Because its goal is to instantly improve both the incumbent solution and the primal bound U^* when both `MAXSUM` and `MINSUM` fail, there is no need to increase its number of calls. The new bound U^* that `MINMAX` obtains helps `MAXSUM` and `MINSUM` to focus on subregions of the search space as guided by the findings of `MINMAX`.

`SEARCH` selects the size of the set B of densest packed bins adaptively. It uses a threshold parameter $\min\{0.97, 1.1 \sum_{i\in I} a_i/U^*\}$, where $\sum_{i\in I} a_i/U^*$ is the packing density of the current incumbent solution. When the packing ratio of the candidate bin is larger than this threshold parameter, `SEARCH` allows B to hold two best patterns; otherwise B holds only the best pattern. In this way, the threshold parameter forces `SEARCH` to launch more recursive calls when the bins in B have better than average density (i.e., when the bins in B exceed the packing density of P^*). Any pattern whose packing density exceeds 0.97 is considered high-quality. Therefore, `SEARCH` explores the promising parts of the solution space by having more recursive calls for items that are yet unassigned and are causing low-density bins. This rule allows `SEARCH` to quickly improve the primal bound and enhances MH's performance: It focuses on smaller subproblems that lead to faster improvements.

The upper bound $\bar{\rho}_i$ of the uniform distribution used to generate the weights δ_i for updating the pseudo costs is the average packing density of bins where item i has been placed during the last κ iterations. Herein, $\kappa = 15$. Finally, $\varepsilon' = 2$ in Expr. (29) whereas ε" $= 3$ in Expr. (30). Both values assign higher costs to larger items; thus, prioritise the packing of larger items at the onset of the packing procedures. ε" $> \varepsilon'$ so that items that remained unpacked during the previous run of `PACK` increase their chances of getting packed during the current call.

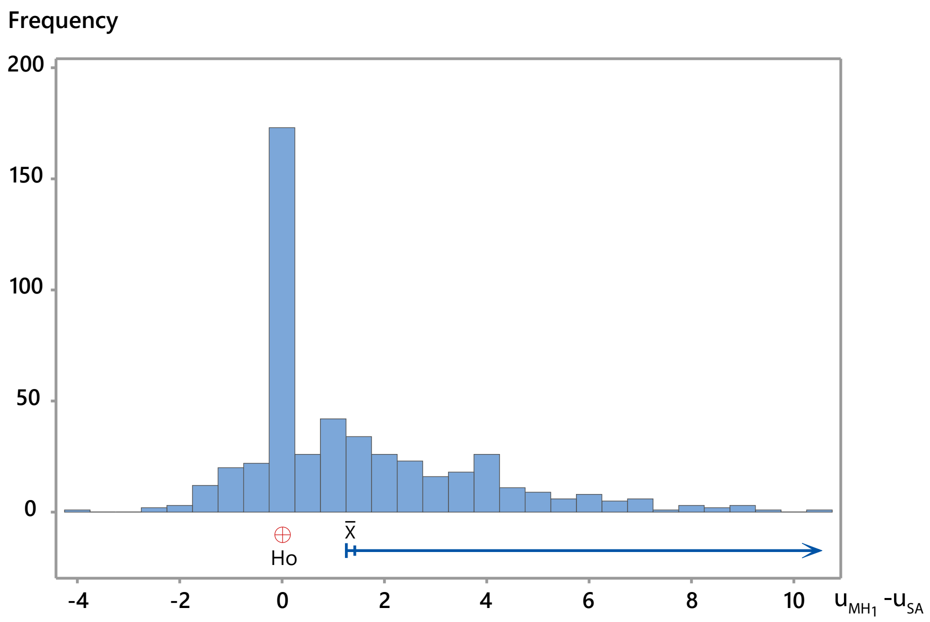

Figure 2: Histogram of u_{MH_1}-u_{SA} and one-sided 95% confidence interval of their mean differences

4.2. Multiple-Sized Bins

Table 3 summarizes the MH results. Columns 1 and 2 specify the class and number of items. Columns 3-5 report the average utilizations u_{SA}, u_{MH_1} and $u_{\mathrm{MH}_{10}}$ over each set of ten instances. Columns 6-8 give the number of times u_{SA} is better, the same, and worse than u_{MH_1} whereas Columns 9-11 provide the same information but for u_{SA} versus $u_{\mathrm{MH}_{10}}$. Finally, Column 12 indicates the number of time the longer runtime of MH improved its solution.

u_{SA} and u_{MH_1} are obtained within equal run times; yet, there is statistical evidence that the mean of u_{MH_1} is larger than the mean of u_{SA} at any level of confidence (p-value=0.0000), with a 1.42 difference. This is clearly depicted in Figure 2, which displays the histogram of the paired differences u_{MH_1}-u_{SA}, the null hypothesis H_0 that stipulates that the mean difference of u_{MH_1}-u_{SA} is 0 versus being strictly positive, and the 95% one sided confidence interval for the mean u_{MH_1}-u_{SA} with a 1.26 estimate of its lower bound.

Increasing the runtime of MH from 1 to 10 minutes amplifies the aforementioned advantages. This is reflected by the five-point summaries of u_{MH_1}-u_{SA} and $u_{\mathrm{MH}_{10}}$-u_{SA}. (A five point summary consists of the minimum, first, second and third quartiles, and the maximum observed difference over the 500 instances.) For u_{MH_1}-u_{SA}, the five point summary is $(-3.92, 0.00, 0.46, 2.57, 10.33)$. It is shifted to the right for $u_{\mathrm{MH}_{10}}$-u_{SA}: $(-2.58, 0.00, 0.76, 3.14, 10.81)$. This shift is depicted in Figure 3, which represents the same information as Figure 2 but for $u_{\mathrm{MH}_{10}}$-u_{SA}. Thus, increasing the runtime of MH reduces the gap for those

Table 3: Results of MH for VS2BP

					Number of times						
		Utilization			u_{SA} vs. u_{MH_1}			u_{SA} vs. $u_{\text{MH}_{10}}$			$u_{\text{MH}_1} < u_{\text{MH}_{10}}$
Class	n	u_{SA}	u_{MH_1}	$u_{\text{MH}_{10}}$	$>$	$=$	$<$	$>$	$=$	$<$	
1	20	90.18	90.74	90.74		8	2		8	2	
	40	93.70	95.17	95.32		2	8		2	8	3
	60	91.95	95.14	95.70	1		9			10	8
	80	90.25	94.87	95.19			10			10	6
	100	91.47	96.01	96.36			10			10	7
	Overall	91.51	94.39	94.66	1	10	39	0	10	40	24
2	20	86.62	86.62	86.62		10			10		
	40	97.42	96.81	97.20	3	7		2	8		1
	60	98.90	98.50	98.63	3	7		2	8		1
	80	98.70	98.17	98.36	7	3		6	4		3
	100	99.62	99.17	99.41	8	2		5	5		3
	Overall	96.25	95.85	96.04	21	29	0	15	35	0	8
3	20	84.38	86.09	86.09		4	6		4	6	
	40	88.08	89.93	90.04	2	1	7	1	1	8	4
	60	87.08	89.74	90.27	2		8	1		9	7
	80	85.55	90.32	90.81			10			10	6
	100	88.13	91.90	92.38			10			10	6
	Overall	86.65	89.59	89.92	4	5	41	2	5	43	23
4	20	80.82	80.82	80.82		10			10		
	40	91.93	91.93	91.93		10			10		
	60	94.99	94.22	94.60	5	5		2	8		3
	80	95.12	94.50	94.86	6	2	2	4	3	3	2
	100	95.82	95.20	95.26	6	3	1	5	4	1	2
	Overall	91.74	91.34	91.50	17	30	3	11	35	4	7
5	20	82.05	83.06	83.18	1	4	5		5	5	1
	40	85.64	87.40	87.85	2	2	6	1	2	7	6
	60	84.70	89.13	89.57	1		9			10	6
	80	84.49	89.33	89.80			10			10	8
	100	86.25	90.59	91.45			10			10	8
	Overall	84.63	87.90	88.37	4	6	40	1	7	42	29
6	20	81.25	81.25	81.25		10			10		
	40	90.18	90.01	90.18	1	9			10		1
	60	92.18	92.10	92.33	1	9		1	8	1	2
	80	93.49	92.99	93.25	5	5		4	4	2	3
	100	94.09	94.06	94.14	4	3	3	3	4	3	2
	Overall	90.24	90.08	90.23	11	36	3	8	36	6	8
7	20	83.99	84.13	84.17	1	5	4	1	5	4	1
	40	87.74	89.19	89.52		1	9		1	9	3
	60	89.52	90.83	91.07		1	9		1	9	3
	80	86.23	88.76	89.27			10			10	5
	100	87.19	89.94	91.17			10			10	8
	Overall	86.93	88.57	89.04	1	7	42	1	7	42	20
8	20	82.27	83.64	83.64		6	4		6	4	
	40	86.04	87.67	87.79	1	1	8		1	9	2
	60	87.92	89.82	90.03	1		9	1		9	4
	80	86.66	90.28	90.41			10			10	3
	100	86.41	90.32	90.75			10			10	5
	Overall	85.86	88.35	88.52	2	7	41	1	7	42	14
9	20	74.42	75.49	75.49		6	4		6	4	
	40	73.42	75.16	75.17		1	9		1	9	1
	60	73.07	74.15	74.15		1	9		1	9	
	80	74.67	75.84	75.84			10			10	
	100	75.84	77.23	77.27			10			10	1
	Overall	74.29	75.58	75.58		8	42		8	42	2
10	20	83.46	84.81	85.33		6	4		5	5	1
	40	90.86	91.56	91.77	1	5	4		3	7	3
	60	92.20	92.26	92.39	3	3	4	4	2	4	2
	80	92.39	93.04	93.24	3		7	2		8	4
	100	92.64	93.18	93.60	1	1	8	1		9	6
	Overall	90.31	90.97	91.27	8	15	27	7	10	33	16
Overall		87.84	89.26	89.51	69	153	278	46	160	294	151

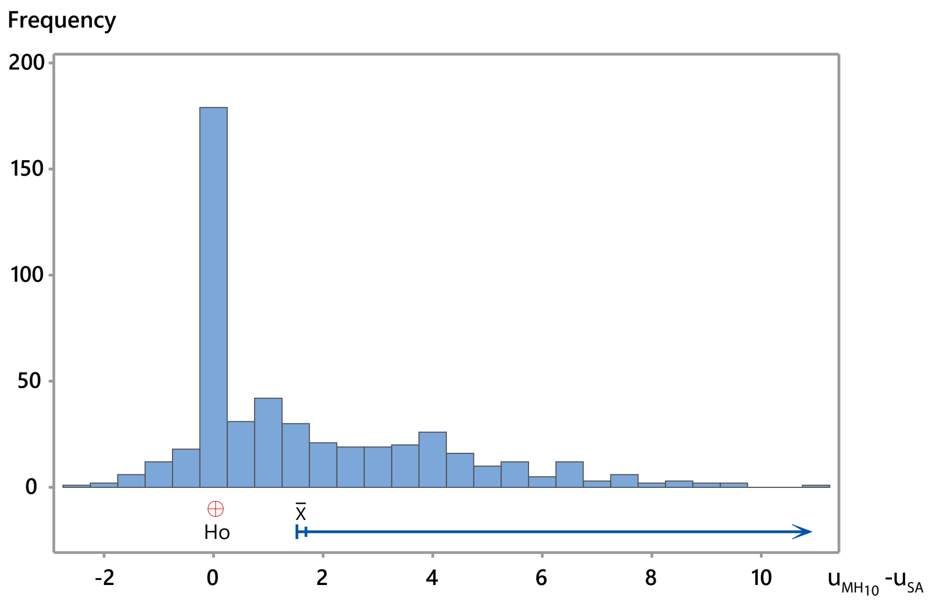

Figure 3: Histogram of $u_{\mathrm{MH}_{10}}$-u_{SA} and one-sided 95% confidence interval of their mean differences

instances where $u_{\mathrm{SA}}>u_{\mathrm{MH}_1}$ while it enlarges it for cases where $u_{\mathrm{SA}}<u_{\mathrm{MH}_1}$. Solutions of instances with $u_{\mathrm{SA}}=u_{\mathrm{MH}_1}=u_{\mathrm{MH}_{10}}$ are most likely optimal. Thus, further increasing the runtime of MH does not affect them. A paired statistical test confirms again that the mean of $u_{\mathrm{MH}_{10}}$ is larger than the mean of u_{SA} at any level of confidence (p-value=0.0000), with a larger average difference of 1.67 versus 1.42, and a larger lower bound estimate of 1.50 of the lower bound of a 95% one sided confidence interval estimate of the mean $u_{\mathrm{MH}_{10}}$-u_{SA} (versus 1.26 for u_{MH_1}-u_{SA}).

Figure 4 further explains the impact of runtime on MH. It presents the box plots of the ratios $\frac{u_{\mathrm{MH}_1}}{u_{\mathrm{SA}}}$, $\frac{u_{\mathrm{MH}_{10}}}{u_{\mathrm{SA}}}$ and $\frac{u_{\mathrm{MH}_{10}}}{u_{\mathrm{MH}_1}}$, as a function of the class of the instances. The comparison of the box plots of $\frac{u_{\mathrm{MH}_1}}{u_{\mathrm{SA}}}$ and $\frac{u_{\mathrm{MH}_{10}}}{u_{\mathrm{SA}}}$ depicts the impact of runtime on MH, and highlights the classes that are difficult for MH. These are, in general, classes 2 and 4, which use fewer bins than other classes. Therefore, the use of an extra bin or of a larger-sized bin makes a big difference on the result. For these classes, MH is limited by the behavior of the mathematical programming solver, which was observed to fail to identify feasible solutions that reduce the number of bins of its current solutions. In addition, DFF constraints appear loose for such instances. Their looseness weakens the lookahead mechanism and its power to guide the search process to feasible areas of the search space.

While $\frac{u_{\mathrm{MH}_{10}}}{u_{\mathrm{MH}_1}}$ assesses the solution enhancement brought to MH by the additional runtime for each class of instances, $\frac{u_{\mathrm{MH}_{10}}}{u_{\mathrm{SA}}}$ reflects the absolute solution

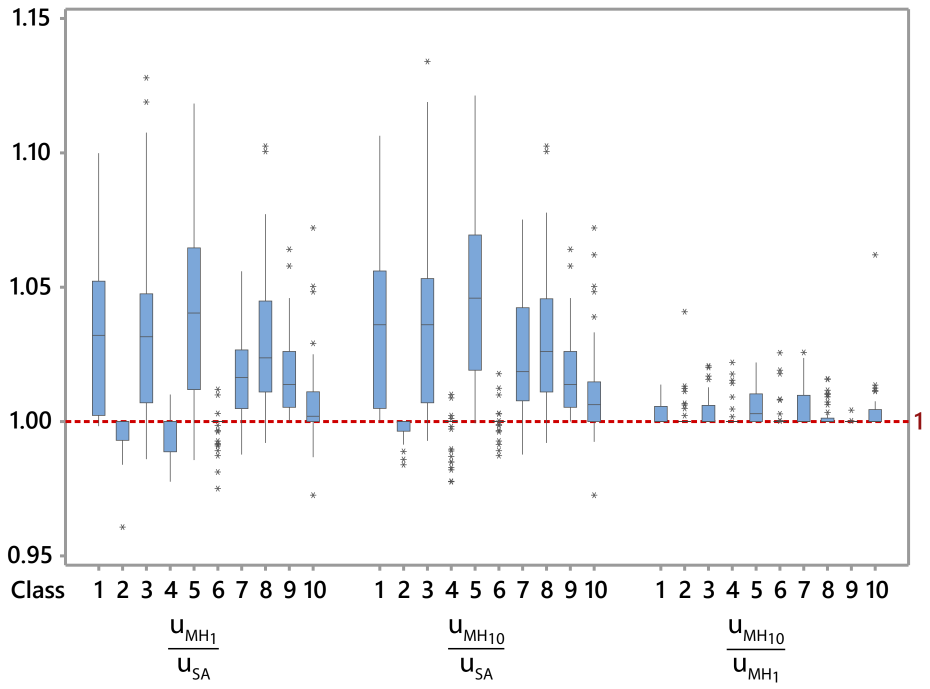

Figure 4: Box plots of $\frac{u_{\mathrm{MH}_1}}{u_{\mathrm{SA}}}$, $\frac{u_{\mathrm{MH}_{10}}}{u_{\mathrm{SA}}}$ and $\frac{u_{\mathrm{MH}_{10}}}{u_{\mathrm{MH}_1}}$

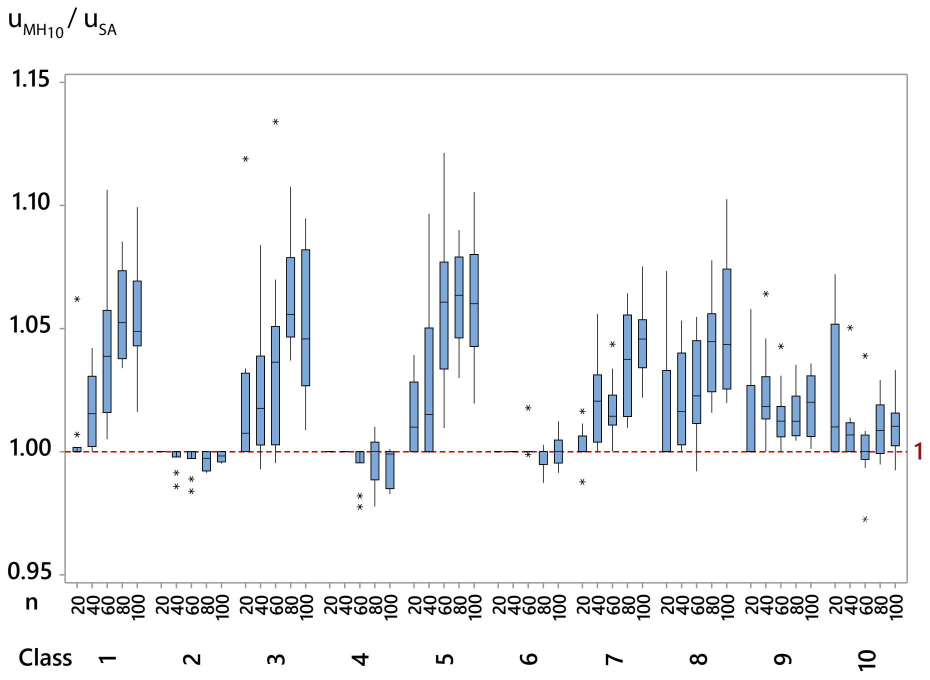

Figure 5: Box plots of $\frac{u_{\text{MH}_{10}}}{u_{\text{SA}}}$ as a function of class and problem size

quality enhancement with respect to the existing upper bound u_{SA}. The analysis of the cases where MH neither matches nor improves the existing upper bound u_{SA} indicates that those instances of classes 2 and 4 are independent of the problem size, as the box plots of Figure 5 further illustrate.

4.3. Single-Sized Bins

Table 4 gives the results of MH on the single bin size benchmark set, where b_{MT} denotes the best known bound on the number of identical bins needed (available from http://or.dei.unibo.it/general-files/best-known-solution-and-lower-bound-each-instance) while b_{CHBP}, b_{SA} and b_{MH} are the counterparts obtained by CHBP, the constructive heuristic followed by a biased sufficiency criterion and by post optimization of Charalambous and Fleszar [7], MH and SA, respectively. Columns 1 and 2 specify the class and number of items. Columns 3-6 report the sum of b_{MT}, b_{CHBP}, b_{SA} and b_{MH} over each set of ten instances. Columns 7 and 8 report the average utilization of u_{SA} and u_{MH} over each set of ten instances. Finally, Columns 9 and 10 indicate the number of instances where b_{MH} exceeds the lower bound b_{MT} of the number of bins and the number of times MH improves the solution of SA by using fewer bins; i.e., the number of times b_{SA} exceeds b_{MH} over each set of ten instances.

Table 4 shows that MH can be successfully applied to the case with identical bins. MH matches the lower bound on the number of bins for all but 85 instances, using one additional bin for all but the first instance of class 5 and $n = 100$ where it uses two additional bins. For all other 415 instances, MH solution is optimal: It uses the lowest number of bins b_{MT}. In addition, MH succeeds in matching all solutions of SA and enhances the best known solution of 13 instances.

4.4. MH's Features

This subsection highlights the merit of the new features of MH by testing MH

1. when only one of the three models `MAXSUM`, `MINSUM`, and `MINMAX` is used in lieu of their sequential iterative application;
2. when the DFF constraints are omitted; and
3. when `SVC` is omitted.

It tests these MH variants on representative instances of size $n = 20,\ 60$, and 100 of classes 5 and 8. Table 5 reports the average packing coefficients over each set of ten instances. Columns 1 and 2 indicate the class and size n of the instances. Columns 3-5 report the average utilizations u_{SA}, u_{MH_1} and $u_{\mathrm{MH}_{10}}$ over each set. Subsequent pairs of columns report the average utilization of MH when it uses only `MAXSUM`, only `MINSUM`, only `MINMAX`, and when it omits DFF, and `SVC`. The first and second columns of each pair correspond to the average utilization after 10 minutes and 40 minutes of run time, respectively.

Table 5 suggests that the new features of MH speed its convergence toward a (near-)global optimum. They amplify the diversification capabilities of MH during the early stages of the search and refine its intensification power during the later stages. Within 40 minutes of run time, the MH variants barely compete with u_{MH_1}.

Driven by their respective objective functions, `MAXSUM`, `MINSUM`, and `MINMAX` explore the search space differently; thus, attain different bounds with distinct convergence speeds. Within an equal run time, `MINMAX` performs fewer iterations than `MINSUM`, which in turn undertakes less iterations than `MAXSUM`. The search gets harder as the objective function is changed from `MAXSUM` to `MINSUM` to `MINMAX` (simply because their MIPs become computationally more expensive). Thus, the order of application of the models is important.

- `MAXSUM` obtains a good upper bound quickly; i.e., its search is greedy but fast. It is 'traditional' in the sense that it focuses on the packing coefficient. It speeds up the search when initiated from scratch or from a 'balanced' solution given by `MINMAX`.
- `MINSUM` is more pragmatic. It does not focus on the packing coefficient. It allows a more uniform assignment of future items by reserving space for not yet packed items **evenly** across all regions. Its tactic is to tolerate and recover from packing mistakes made during early iterations of the search.

Table 4: Results of MH for the identical bins case

		Number of bins				Utilization		Number of times	
Class	n	b_{MT}	b_{CHBP}	b_{SA}	b_{MH}	u_{SA}	u_{MH}	$b_{\text{MT}} < b_{\text{MH}}$	$b_{\text{SA}} > b_{\text{MH}}$
1	20	71	71	71	71	82.16	82.16		
	40	134	134	134	134	87.61	87.61		
	60	197	201	200	200	90.05	90.05	3	
	80	274	275	275	275	90.26	90.26	1	
	100	317	321	320	317	93.86	94.78		3
	Overall	993	1002	1000	997	88.79	88.97	4	3
2	20	10	10	10	10	64.30	64.30		
	40	19	20	19	19	69.42	69.42		
	60	25	26	25	25	81.76	81.76		
	80	31	33	31	31	89.30	89.30		
	100	39	39	39	39	85.74	85.74		
	Overall	124	128	124	124	78.10	78.10		
3	20	51	52	52	52	74.08	74.08	1	
	40	92	97	94	94	83.08	83.08	2	
	60	136	140	140	140	86.30	86.30	4	
	80	187	196	191	190	87.43	87.91	3	1
	100	221	230	227	225	88.92	89.79	4	2
	Overall	687	715	704	701	83.96	84.23	14	3
4	20	10	10	10	10	60.92	60.92		
	40	19	19	19	19	66.15	66.15		
	60	23	25	25	25	78.77	78.77	2	
	80	30	33	32	32	83.99	83.99	2	
	100	37	39	38	38	85.09	85.09	1	
	Overall	119	126	124	124	74.98	74.98	5	
5	20	65	65	65	65	74.31	74.31		
	40	119	121	119	119	82.40	82.40		
	60	179	183	181	181	84.21	84.21	2	
	80	241	247	247	247	84.96	84.96	6	
	100	279	288	287	285	88.40	89.05	5	2
	Overall	883	904	899	897	82.85	82.99	13	2
6	20	10	10	10	10	53.15	53.15		
	40	15	19	18	18	62.37	62.37	3	
	60	21	22	22	22	77.56	77.56	1	
	80	30	30	30	30	77.67	77.67		
	100	32	35	35	35	81.07	81.07	3	
	Overall	108	116	115	115	70.37	70.37	7	
7	20	55	55	55	55	77.95	77.95		
	40	109	112	111	111	82.61	82.61	2	
	60	156	160	159	159	84.43	84.43	3	
	80	224	233	232	232	83.44	83.44	8	
	100	269	275	273	273	85.55	85.55	4	
	Overall	813	835	830	830	82.79	82.79	17	
8	20	58	58	58	58	75.42	75.42		
	40	112	114	113	113	81.59	81.59	1	
	60	159	163	162	161	84.11	84.59	2	1
	80	223	226	225	224	84.86	85.27	1	1
	100	274	279	279	278	84.76	85.02	4	1
	Overall	826	840	837	834	82.15	82.38	8	3
10	20	42	44	43	43	75.54	75.54	1	
	40	74	74	74	74	85.56	85.56		
	60	98	103	102	102	88.34	88.34	4	
	80	123	130	130	129	90.25	90.93	6	1
	100	153	163	160	159	91.95	92.51	6	1
	Overall	490	514	509	507	86.33	86.57	17	2
Overall		7173	7312	7272	7259	79.36	79.46	85	13

Table 5: Average utilization of bins of different variants of MH

Run time↘		SA	MH		`MAXSUM`		`MINSUM`		`MINMAX`		No DFF		No `SVC`	
Class	n	1	1	10	10	40	10	40	10	40	10	40	10	40
5	20	82.05	83.06	83.18	79.24	79.36	83.06	83.06	82.76	82.76	82.81	83.06	81.88	81.88
5	60	84.70	89.13	89.57	86.82	87.79	89.25	89.56	87.24	88.66	88.99	89.53	88.64	88.94
5	100	86.25	90.59	91.45	86.65	88.17	89.36	91.24	60.66	82.27	90.54	91.34	89.06	90.56
8	20	82.27	83.64	83.64	81.74	82.11	83.47	83.47	83.58	83.58	83.64	83.64	83.58	83.58
8	60	87.92	89.82	90.03	86.98	87.96	88.48	89.45	87.71	88.45	89.23	89.95	89.35	89.80
8	100	86.41	90.32	90.75	88.79	89.80	87.73	89.59	48.75	77.10	89.82	90.38	88.97	89.98

- `MINMAX` works generally well when applied on already existing upper bounds. It can escape from local optima.
  - It beats `MAXSUM` on smaller instances but does not work well for large instances, for which it sometimes gets packing coefficients as low as 50%.
  - It takes longer than `MAXSUM`. It progresses slowly but steadily, and discovers 'hidden' solutions.
  - It balances the worst and the average item to region costs c_k, defined by the right hand side of Eq. (23). A tight bound limits the flexibility of the search while a loose bound gives the search a lot of freedom; therefore, allows diversification.
  - Starting `MINMAX` from scratch hinders its progress. It needs a long time to find a balanced compromise among the c_k variables that minimize its objective function. Therefore, it appears more effective when applied together with a good bound developed by `MAXSUM` and/or `MINSUM`.

In general, reaching the global optimum from a near-global optimum is very hard. The DFF constraints and SVC help MH achieve that transition, in particular for large instances whose convergence is slow. Omitting the DFF constraints does not particularly worsen the performance of MH. However, including them allows MH to reach the extra mile; getting better solutions fast. Similarly, without SVC, MH works well too. It loses 1-2% on average, but these additional few percents do matter. They are the hardest to get.

5. Conclusion

This paper approximately solves the unweighed fixed-orientation variable-sized two-dimensional guillotine bin packing problem. It applies a hybrid matheuristic that diversifies its search via three mixed-integer programs whose objective function coefficients are determined by a sequential value correction algorithm. The mathematical programs are dotted with lookahead mechanisms that reserve space for unpacked items while being guided by dual feasibility function constraints and by no-good cuts imposed by upper bounds on the objective function. These mechanisms make the mathematical programs focused on the most 'productive' part of the search space. This matheuristic is innovative in terms of modeling and solution approach: It opts for item to region assignment type (without explicitly considering the geometrical aspect of the problem), and for many items to many bins (not necessarily identical) assignments in contrast to existing sequential heuristics which pack items one by one filling one bin at a time. It outperforms state-of-the-art methods. It proves the optimality of 415 solutions of the 500 identical-bin benchmark instances, matches the best known results of another 72 instances, and improves the other 13. For the variable-size bin packing, it matches 160 and improve 294 best known results (out of 500).

This matheuristic represents a general framework that can be applied to other cutting and packing problems. It can be naturally adapted to the weighted case where the cost of a bin is not proportional to its area and to the net profit case where bins have costs and items generate profits. Similarly, it can be extended to the case where pieces can be rotated, and to packing with defects.

References

[1] Alvarez-Valdes, R., Parreño, F., Tamarit, J., 2013. A grasp/path relinking algorithm for two- and three-dimensional multiple bin-size bin packing problems. Computers & Operations Research 40, 3081–3090.

[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] Angelelli, E., Archetti, C., Peirano, L., 2020. A matheuristic for the air transportation freight forwarder service problem. Computers & Operations Research 123, 105002.

[4] Belov, G., Scheithauer, G., 2007. Setup and open-stacks minimization in one-dimensional stock cutting. INFORMS Journal on Computing 19, 27–35.

[5] Belov, G., Scheithauer, G., Mukhacheva, E.A., 2008. One-dimensional heuristics adapted for two-dimensional rectangular strip packing. Journal of the Operational Research Society 59, 823–832.

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

[7] Charalambous, C., Fleszar, K., 2011. A constructive bin-oriented heuristic for the two-dimensional bin packing problem with guillotine cuts. Computers & Operations Research 38, 1443–1451.

[8] Correia, I., Gouveia, L., Saldanha-da Gama, F., 2008. Solving the variable size bin packing problem with discretized formulations. Computers and Operations Research 35, 2103–2113.

[9] Crevits, I., Hanafi, S., Mahjoub, A., Taktak, R., Wilbaut, C., 2019. A special case of variable-sized bin packing problem with color constraints,

in: 2019 6th International Conference on Control, Decision and Information Technologies, CoDIT 2019, pp. 1150–1154. doi:10.1109/CoDIT.2019.8820707.

[10] Cui, Y.P., Cui, Y., Tang, T., 2015. Sequential heuristic for the two-dimensional bin-packing problem. European Journal of Operational Research 240, 43 – 53.

[11] 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.

[12] Hong, S., Zhang, D., Lau, H.C., Zeng, X., Si, Y.W., 2014. A hybrid heuristic algorithm for the 2d variable-sized bin packing problem. European Journal of Operational Research 238, 95–103.

[13] Kollsker, T., Malaguti, E., 2021. Models and algorithms for optimising two-dimensional lego constructions. European Journal of Operational Research 289, 270–284.

[14] Libralesso, L., Fontan, F., 2021. An anytime tree search algorithm for the 2018 ROADEF/EURO challenge glass cutting problem. European Journal of Operational Research 291, 883–893.

[15] Liu, Y., Chu, C., Wang, K., 2011. A dynamic programming-based heuristic for the variable sized two-dimensional bin packing problem. International Journal of Production Research 49, 3815–3831.

[16] Lodi, A., Martello, S., Monaci, M., Vigo, D., 2014. Two-Dimensional Bin Packing Problems. 2 ed.. John Wiley & Sons, Hoboken, NJ, USA. chapter 7. pp. 107–129.

[17] Lodi, A., Martello, S., Vigo, D., 2002. Recent advances on two-dimensional bin packing problems. Discrete Applied Mathematics 123, 379–396.

[18] 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.

[19] Marinelli, F., Pizzuti, A., 2018. A sequential value correction heuristic for a bi-objective two-dimensional bin-packing. Electronic Notes in Discrete Mathematics 64, 25 – 34. 8th International Network Optimization Conference - INOC 2017.

[20] M'Hallah, R., 2019. Women in Industrial and Systems Engineering, Women in Engineering and Science: Key Advances and Perspectives on Emerging Topics. Springer Nature. chapter Combining Exact Methods to Construct Effective Hybrid Approaches to Vehicle Routing. doi:10.1007/978-3-030-11866-2_19.

[21] Ortmann, F., Ntene, N., van Vuuren, J., 2010. New and improved level heuristics for the rectangular strip packing and variable-sized bin packing problems. European Journal of Operational Research 203, 306–315.

[22] Parreño, F., Alonso, M., Alvarez-Valdes, R., 2020. Solving a large cutting problem in the glass manufacturing industry. European Journal of Operational Research 287, 378–388. doi:10.1016/j.ejor.2020.05.016.

[23] Pisinger, D., Sigurd, M., 2005. The two-dimensional bin packing problem with variable bin sizes and costs. Discrete Optimization 2, 154 – 167.

[24] 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.

[25] 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.

[26] Rodrigues, F., Agra, A., Hvattum, L., Requejo, C., 2021. Weighted proximity search. Journal of Heuristics doi:10.1007/s10732-021-09466-0.

[27] Wei, L., Oon, W.C., Zhu, W., Lim, A., 2013. A goal-driven approach to the 2d bin packing and variable-sized bin packing problems. European Journal of Operational Research 224, 110–121.

[28] Witteman, M., Deng, Q., Santos, B.F., 2021. A bin packing approach to solve the aircraft maintenance task allocation problem. European Journal of Operational Research .