A Grid-Based Heuristic for Two-Dimensional Packing Problems

Lam T. Bui
the Department of Software Engineering,
Le Quy Don Technical University, Vietnam
Email: lam.bui07@gmail.com

Hussein A. Abbass
the School of SEIT,
University of New South Wales, Australia
Email: h.abbass@adfa.edu.au

Stephen Baker
the Land Operations Division,
DSTO, Australia
Email: steve.baker@dsto.defence.gov.au

Michael Barlow
the School of SEIT,
University of New South Wales, Australia
Springfield, USA
Email: m.barlow@adfa.edu.au

Axel Bender
the Land Operations Division,
DSTO, Australia
Email: axel.bender@dsto.defence.gov.au

Ruhul Saker
the School of SEIT,
University of New South Wales, Australia
Springfield, USA
Email: r.sarker@adfa.edu.au

***Abstract*— To solve two-dimensional (2D) rectangular packing problems, we introduce a new spatial method based on the discretization of the container into a grid of cells with predefined resolution. Before an item is added, grid cells are checked whether they can accommodate the item. If an appropriate empty cell cluster is found, the item is added and moved towards the bottom-left corner of the container. This placement and sliding method is supplemented by a heuristic that orders the items according to descending size. Order and rotation of items can be improved by hybridizing the heuristic with a genetic algorithm (GA) in which a population of order-rotation chromosomes is evolved.**

The method is tested on 47 benchmark problems and compared to other methods in the literature. This shows that it is fast and performs very well in finding close to optimal problem solutions. Particularly for large problem sizes, it outperforms some of the currently leading methods, such as heuristic recursive (HR). The hybridization with the GA meta-heuristic results in further performance improvements.

I. INTRODUCTION

Optimizing cutting/packing problems is NP-hard and has many applications such as wood, glass, paper or metal sheet cutting in manufacturing, vehicle and container loading in transportation, or memory allocation in computing [15], [5]. Although the objectives might be context-dependent, the abstract representation of these problems is centered on the concept of packing a set of items into the smallest possible space within an object. Alternatively, the problem can be seen as finding the smallest object that can accommodate a number of items.

This paper addresses the problem of **O**rthogonally **P**acking items into a single 2D **R**ectangular **C**ontainer, **OP1RC**, where '*1*' reflects that we are dealing with a single box. The description of OP1RC is as follows:

- A finite set of rectangular items.
- A single container with fixed width and open ended height.
- All items will be placed orthogonally. Rotation of 90^o is allowed.
- The items are non-guillotineable.
- The objective is to minimize the height of the container.

We propose a new spatial method for OP1RC. The unique feature of our method is that the container is discretized into a grid of cells with predefined and fixed size, i.e. predefined *resolution ratio*. These cells will be used for checking the available space whenever an item is being added. If there is an available space, the item will be placed, then might be moved towards the bottom-left corner, and the status of cells (that are occupied by the item) is updated accordingly. By using cells as well as allowing the items to slide, the spatial gaps between items are eliminated as much as possible. This means that the result of the packing does not depend on the chosen, predefined grid resolution. A heuristic is then used in which the sequence of placing items into the container is determined. The items are sorted by their height. When a tie occurs, the widths of items are used to break it.

The results demonstrate the advantage of the proposed heuristic in comparison to some of those commonly used in the literature. We further hybridize the heuristic with a genetic algorithm (GA) in which a found solution is used as a seed in the GA population. The hybrid method demonstrates an overall better performance, especially on larger problems.

The rest of the paper is organized as follows: in the next section we discuss the literature that is relevant to OP1RC. This is followed by a description of the spatial approach for OP1RC. An experimental study is then carried out in Section IV. In the last section, we present our conclusions.

II. BACKGROUND ON OP1RC AND RELATED WORK

The natural objective for OP1RC is the minimization of the container area needed to accommodate all items. This area comprises the area occupied by the items plus the wasted unoccupied areas between items and between items and container walls. Practically, other objectives can be used instead, including minimization of the height of the packing area, maximization of space utilization, minimization of total wasted area, or a combination of these objectives.

This paper also addresses the problems of bin packing, pallet/vehicle loading, strip packing, or stock cutting. While, in principle, these problems are different [20], they are equivalent in the special case of OP1RC considered here. Therefore, we use '*bin*', '*container*' and '*box*' interchangeably. OP1RC is called *2D packing* by Hopper and Turton [15], while Burke et al [5] call it *orthogonal stock cutting*. In [20], it is part of a bigger problem class called Open Dimension Problems.

There is a huge literature on the general packing problem, see [11], [10], [15], [20]. Methods to address OP1RC can be divided into three categories [5]: *exact*, *heuristic*, and *meta-heuristic*. For the exact methods, the final solution will be optimal. The problem is re-formulated in a way that allows the application of well-known exact algorithms such as linear programming [14], dynamic programming [4], tree search and brand and bound [8]. However, the application of exact methods is usually limited to smaller to medium size problems because of combinatorial explosion. Therefore, heuristics are sought as alternatives and possibly the only alternatives for large scale problems.

When using heuristics, the obtained results are not necessarily optimal, but they can be obtained within a reasonable time. The heuristic can decide on placing items in the container and the ordering of the items. The probably best known heuristic is the *bottom-left* method (BL) in which each is placed into the lowest possible position and then left-justified [3]. The items can be ordered according to their size (either width or height).

In the simple BL version proposed in [16], an item is first placed at the top-right. It then slides as far down to the bottom as possible, then the farthest to the left. [18] introduces an improved BL where the item can move down to the bottom whenever possible until no further leftward movement is allowed. BL is a fast algorithm with a worst-case time complexity of $O(n^2)$ [15]. However, the mentioned BL methods' obvious weakness is the creation of '*dead*' areas that cannot be filled with any of the remaining items.

Another BL version, called *bottom-left fill* (BLF), was proposed to overcome this bias. BLF creates and maintains a list of positions that can be used for placement. If an item needs to be packed, this list is searched in a bottom-left manner, i.e. the bottommost leftmost available position on the list is the first one that will be checked whether it can accommodate a given item. While BLF can fill dead areas, it is slower than its BL counterparts and its worst-case complexity is $O(n^3)$ [7].

Several efforts have been made to improve both quality and time of the BL methods [21], [5]. For instance, in [5] a fast method is proposed, in which a skyline is maintained over the entire width of the container. After every item placement, the values of the skyline are updated with the top height coordinate of the newly added item. The sequence of items is found as follows: the lowest area (in the skyline) is searched first and then an item that is a best-fit for the area is found; this is the reason why the method is called '*best-fit*' (BF). During the search for a best fit, items can be rotated. Although the real order of items is established during the placement process, the items are pre-ordered by descending height.

Solutions found by heuristics are sometimes far from optimal. Improvements can be achieved by hybridizing the heuristic with some meta-heuristics such as GA, tabu search or simulated annealing. In [16], [15], [2], [1], [13], several GA are proposed to enhance the BL or BLF placement strategies. Simulated annealing is also employed intensively [9], [17], [12], [5]. For instance, in [6] the authors make use of only a part of the BF list of items; packing the remaining items is left for simulated annealing to solve. In [15] the authors empirically study the performance of meta-heuristics, and find that meta-heuristics perform significantly better than their heuristic-only counterparts.

In summary, OP1RC is a combination of two sub-problems: bin layout and ordering. The challenge for heuristics and meta-heuristics is to avoid holes inside the packing area which can be brought about by the placement and/or the ordering of items.

III. New spatial approach to solving OP1RC

It is clear that for OP1RC the management of the available space is an important task. A poor placement strategy inevitably results in empty areas, that could still fit some of the remaining items. When having a good placement strategy, the order of items can play a '*strategic*' role because placing some items before others can reduce spatial waste areas. In this section, we introduce an OP1RC heuristic that makes use of a new placement method and combines it with an established ordering technique.

A. A spatial method for placement

In the proposed method, instead of using a list of positions as in BLF, or a skyline as in BF, we consider the container as being made up of a grid of cells. This grid is used when checking the availability of space during item placement. The resolution of the grid discretization is a user-defined parameter. Assume a container C has a width W. Given a resolution (i.e. cell width and height) δ, the number of columns N is determined as:

$$N = \text{ceil}\left(\frac{W}{\delta}\right). \quad (1)$$

The number of rows M is set to be sufficiently large. Initially, the grid is empty (the status s_{ij} of a cell c_{ij} is marked '*available*', i.e. $s_{ij} = 0, \forall i \in [1, M], j \in [1, N]$, see Figure 1a). When an item I^* is to be placed, we conduct a search for an available cell. The search progresses in a bottom-left direction (i.e. the lowest row is looked at first and within a row, the leftmost cell is the first one to be checked).

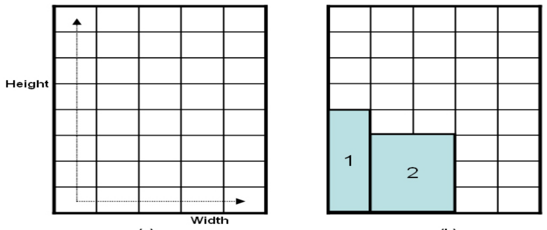

Fig. 1. Illustration of container discretization (a), and how the items are packed (b).

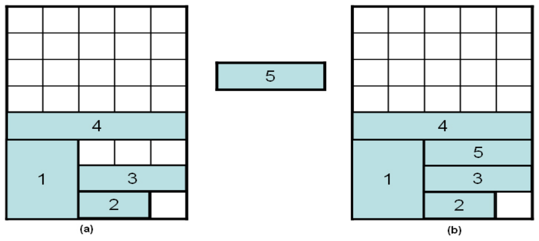

Fig. 2. Example of filling items into the container.

When an available cell $c_{\overline{ij}}$ is found, the algorithm checks whether the selected item can fit in the area comprising the found cell and available adjacent cells. With M set sufficiently large, we will always find a place to pack item I^*. So, the procedure for this can be formalized as follows:

- Find an available cell $c_{\overline{ij}}$ for item I^*
- Update the status of all related cells c_{ij} to $s_{ij} = 1 \; \forall i,j | c_{ij} = \text{Occupied}(I^*, i, j)$. If a cell c_{ij} is only partially covered by an item, its occupancy value still updates to $s_{ij} = 1$.
- Assign coordinates $(x,\ y)$ to item I^* where $(x,\ y)$ describes the coordinates of the bottom-left corner of the item after placement.

In the example of Figure 1b, the first item is placed at the first cell since it can fit and the second item is placed next to item 1. After the placement of an item, the formerly available cells that are now fully or partially occupied will be marked unavailable. If the item does not fit into the area, another available cell will be sought. An example is given in Figure 2 to demonstrate this case. Assume that we need to pack the fifth item. Previous packing of items left two empty and disconnected spaces in between the packed items: one consisting of a single cell in the bottommost row and the other comprises a cluster of three cells. In the proposed heuristic, these cells will be checked, starting with the one-cell cluster. The algorithm will find that it cannot fit the item and move on to the other cluster which can accommodate the item. Therefore, the item will be inserted into the unoccupied three-cell cluster.

Because we discretize the space, in some cases the resolution of the discretization introduces a bias that can generate empty spaces between items. This bias is eliminated through a *sliding method*: when an item is added, it does not stop at

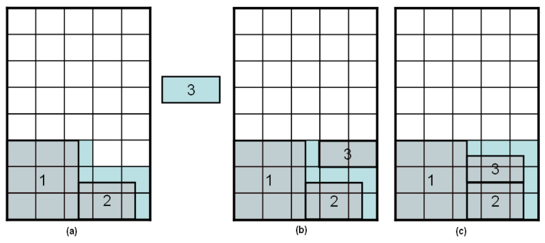

Fig. 3. Example of placing and sliding items.

the bottom-left boundaries of the unoccupied cells. Instead, it will be slid towards the bottom-left until it hits the boundaries of the neighboring items. For example, in Figure 3a, two items are already placed. Although they do not fully occupy all 13 cells, our algorithm still marks these cells unavailable. Item 3 looks for a position and the heuristic finds that the cells shown in Figure 3b are available. However, there is a gap between the group of items 1 and 2 and the newly placed item 3. Therefore, we allow item 3 to slide towards the bottom-left corner (Figure 3c). After sliding the item, the newly occupied cells are marked. The advantage of this process is that the discrete space is used to reduce the search complexity while the filling and sliding mechanism are used to overcome the bias generated by the predefined resolution of discretization. The introduction of a grid makes the heuristic faster than BLF since it does not need to check overlaps among all items. The sliding considers nearest neighbors only and thus is time efficient and independent of the total number of items.

Note that a reasonable choice for the grid resolution δ is important since it might affect the execution time of the algorithm. If δ is too small, the search for available space will be time consuming. If it is too large (i.e. the grid cells are big), the case in Figure 3 is likely to happen quite often and more time is spent on sliding items. Further, it is also likely that the rightmost column will contain some wasted gaps (see Figure 4). There are two ways to overcome this problem. The first one is to allow items to be placed outside the container on the right (in what we call a *virtual column*) and then slide them down to the wasted areas. The second is to tailor the grid to the problem at hand: the resolution (the cell size) δ can be set equal to the size of the smallest item. Assuming that each item *i* has width w_i and height h_i with $h_i \leq w_i$, we can define:

$$\delta = \min_{i \in I} h_i \tag{2}$$

For a δ defined this way, the problem illustrated in Figure 4 will never occur: if a gap between already placed items and the right container wall is big enough to accommodate a later item, then this gap has to be at least one cell wide. Thus the heuristic will find at least one empty cell in the gap cluster and thus can check whether a later item might fit. In this paper we therefore choose δ as in Eq. 2. Of course, further improvements are achievable by combining an appropriate

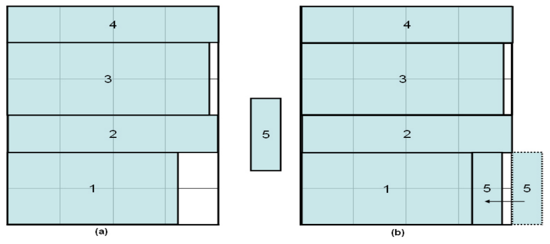

Fig. 4. Example of using a virtual column: Item 5 is allowed to drop down the virtual column on the right side of the container

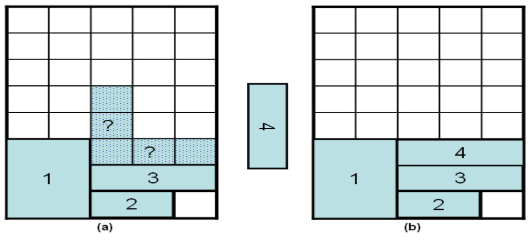

Fig. 5. Example of choosing the placement orientation of an item.

choice for δ (such as in Eq. 2) with the virtual-column technique.

B. Placement improvement policy

Item rotation is allowed when checking the available space. However, in many cases, both orientations of the item can be packed. Therefore, we can use one of two additional criteria to decide whether to rotate an item: we can optimize either the overall height *H* of the packed area or the space utilization $U(i)$ which, given packing item *i* is defined as follows

$$U(i) = \frac{\sum_{j=1}^{\overline{n}} a(j)}{\overline{A}}. \tag{3}$$

Here *a(j)* denotes the area of item $j \in \overline{I}$, $\overline{I}$ is the subset of all $\bar{n}$ already packed items including item i, and $\overline{A}$ is the area of the smallest boundary rectangle of the packing space.

The placement of an item within a container is carried out with both orientations. The orientation which provides the best value on the chosen criterion will be selected. In Figure 5, for example, item 4 can be placed in either orientation. However, the horizontal orientation will give a better placement with respect to both, overall height and space utilization. Note that when testing both criteria on benchmark problems, we could not discern any significant differences in performance.

C. Size-based heuristic for defining order of items

If we consider the layout of items in the container as a *tactical* decision, the heuristic for finding the order in which items are added becomes a *strategic* decision. Thus, we order items before we start their placement. As in [15], [5] we base our ordering heuristic on item size.

We sort the list $I = \{1, 2, \ldots, n\}$ of items i in descending order of their height; i.e. the taller of two items will be packed first. Let $\pi = \{\pi_1, \pi_2, ..., \pi_n\}$ be the found order of items. For two items *i* and *j*, if $h_i > h_j$ then $\pi_i > \pi_j$. Note that before sorting we swap width and height of any item $i \in I$ whose height is greater than its width, i.e. if $h_i > w_i$ then swap h_i and w_i. Further, if $h_i = h_j$ and $w_i > w_j$ then set $\pi_i > \pi_j$. If both item heights and widths are equal, we pick a random order. As a result of this ordering, the shortest (and thinnest) items will be processed last, which provides flexibility for filling gaps during the packing of items.

D. General structure of proposed method

We call our proposed method Grid-based Spatial Packing with Size-based ordering—**GSPS**. Its general structure is as follows:

- **Step 1.** Swap height and width of items in I such that $h_i \leq w_i$
- **Step 2.** Sort the items in I according to their height, thus defining an order π of items
- **Step 3.** Fix δ (e.g. as in Eq. 2) and discretize the container *C* with cells defined following Eq. 1
- **Step 4.**
  According to order π,
  - *For* each item $i \in I$ *do*
    - Place i into container C
  - *End*
- **Step 5.** Return a solution S

In the above structure, *Step 4* will call the following placement procedure:

- **Placement procedure**
  - Input: item i
  - Find an available space for item i
  - Check orientation for better performance
  - Update the status of the cells that are occupied by the item
- **End**

In general, our method still works in the BL manner that has been shown as one of the most theoretically as well as empirically grounded methods in the literature. However, here we introduce a new approach to manage the holes created within the packing area that is much less expensive than the BLF methods, see next section.

E. Computational complexity

GSPS makes use of a known sorting algorithm that has worst-case complexity of $O(n \log n)$ with n being the problem size (i.e. the number of items). We therefore merely discuss the computation time needed to execute the placement procedure in Step 4 of Section III-D. This time depends on n, the size of the container and the resolution ratio δ.

Firstly, for each item *i*, we need to search for an available cell. There are maximally $m = MN$ cells with M and N being, respectively, the number of rows and columns (Eq. 1). Once a cell is found, the area of the grid occupied by the

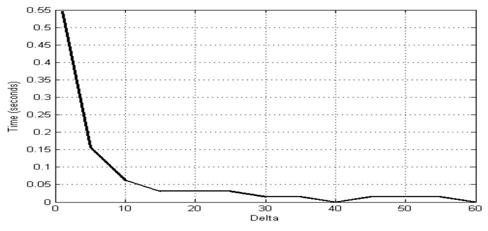

Fig. 6. Computation time of the proposed algorithm as a function of resolution ratio δ in the 50-items example of [2].

item (containing $\text{ceil}\frac{w_i}{\delta}\text{ceil}\frac{h_i}{\delta}$ cells) is filled. Since $\frac{w_i h_i}{\delta^2}$ is usually much smaller than $m = MN$, for each item i the computational complexity of searching and filling the grid of cells is $O(m)$; i.e. this part of the heuristic does not depend on the problem size n. In practice, thus, with increasing number n of items the time related to searching the discretized space of a container gets small compared with the time needed to process the sequence of items. This is particularly the case, if we choose a dynamic M value (to make it "sufficiently large" as stated earlier). At the start of the algorithm we set $M = 1$; then it will be determined based on the current height H of the container during the placement process: $M \equiv \text{ceil}\left(\frac{H}{\delta}\right) + 1$.

Secondly, when checking neighboring cells for the possibility of sliding an item, only a constant number of cells (at most eight in the case of Figure 3) is used. In the worst case, these cells can contain all items but one. Further, the number of cells that neighbor the bottom and left sides of the item to be slid are at most N in the horizontal direction and $M-1$ in the vertical direction. Thus at most $N(M-1)$ cells have to be searched for each item that needs to be slid to the bottom-left. Therefore, the sliding heuristic has a worst-case complexity of $O(nm)$, which means that the placement procedure in Step 4 of Section III-D has worst-case complexity of $O(nm^2)$ and increases only *linearly* with problem size n.

Although we propose to use the size of the smallest item to fix the resolution ratio δ (Eq. 2), it is worthwhile studying how δ affects computation time. Fig. 6 shows an example of such examination. As in [2], we selected a problem with 50 items. Its analysis is consistent with our discussion before Eq. 2, namely that with smaller δ, the heuristic takes longer to place items. The computation time reduces as δ increases. When δ is large enough (here: > 30), efficiency starts to fluctuate slightly indicating that the need to slide items becomes a dominating effect.

F. Hybridization with meta-heuristics

1) General structure: We hybridize the GSPS heuristic with a GA, noting that other meta-heuristics would also be possible. This generates an approach that we call **GPSP-M**. We allow both order and orientation of items to evolve. The hybridization is introduced as follows:

- **Step 1**: Use the spatial heuristic to find the initial solution and represent it as described below.
- **Step 2**: Randomly initialize a population of chromosomes for GA. Include the solution found by the spatial heuristic as a seed.
- **Step 3**: Evaluate and evolve the population to find a potentially better solution than the one found in Step 1.
- **Step 4**: Return the final, best solution for the OP1RC problem at hand.

2) Representation of solution: Each individual in the GA is represented by a diploid chromosome with two rows: the first one is for the permutations of the items' order, while the second one is for the orientations of all items. This means that an item's position in the order π and its orientation will be specified in a chromosome column. The first row in the chromosome is similar to that in a traveling salesman problem.

3) Solution evaluation: The primary objective of OP1RC is to minimize the height of the given container. However, two solutions with identical height will also need to be evaluated in terms of space utilization. For this reason, the objective function for the optimization will be derived from height and space utilization as follows:

$$F(s_i) = \lambda H(s_i) + \overline{U}(s_i) \tag{4}$$

where $H(s_i)$ is the overall height generated by solution s_i, λ is a scale coefficient, $\overline{U}(s_i) = 1 - U(s_i)$, and $U(s_i)$ is the space utilization after placement of the last item (see Eq. 3).

4) Computational complexity: Hybridizing GSPS with a meta-heuristic does not affect the computational complexity of the novel feature of GSPS—namely, its placement procedure. In GSPS-M, each item of an individual (representing sequence and orientations of items) is placed as in GSPS, except that its orientation is not subjected to any change. Therefore, the time to test an individual is slightly shorter than the time t needed to find a solution with GSPS. The total time for GSPS-M to find a packing strategy is thus not longer than tPG where P is the population size and G is the number of generations. For example, in benchmark problem P3 of C6 (to be discussed in the next section) the time to evaluate an individual took $t = 0.078$ seconds, $P = 100$ and $G = 1000$; so the total time for GSPS-M to find the final strategy is 0.078*100*1000 seconds, i.e. about 130 minutes.

IV. Experiments

A. Benchmark problems

In our experiments, we address eleven classes of benchmark problems which have been frequently used in the literature. As such, it is easy to compare the different solution methods. The problems represent a diverse set of OP1RC features, especially in terms of problem size. The first seven classes C1 to C7 consist of 21 problems used in [15], the class C8 has 13 problems used in [5], C9 comprises one problem from [2], and the last two classes, C10 and C11, have six problems each with real-valued representation of items [19].

TABLE I

AVERAGE RELATIVE DISTANCE TO OPTIMAL HEIGHT (IN %) FOR THE FIRST SEVEN PROBLEM CLASSES C1 TO C7. BOLDFACE TEXTS MARK THE BEST PERFORMANCE. BL, BL-DH, BL-DW, BLF, BLF-DH AND BLF-DW RESULTS ARE EXTRACTED FROM [15].

	C1	C2	C3	C4	C5	C6	C7
BL	25	39	33	33	31	34	41
BL-DH	17	68	27	21	18	19	31
BL-DW	18	31	24	18	22	21	29
BLF	14	20	17	15	11	12	10
BLF-DH	11	42	12	6	5	5	4
BLF-DW	11	12	12	5	5	5	5
GSPS	**8**	**7**	**7**	**3**	**3**	**2**	**1**

TABLE II

SAME AS TABLE I BUT COMPARING SOLUTIONS GENERATED BY BF (SOURCE: [5]) AND HR ([21]) WITH GSPS'S.

	C1	C2	C3	C4	C5	C6	C7
BF	12	7	10	4	3	3	2
HR	**8**	**4**	**7**	**2**	**2**	3	2
GSPS	**8**	7	**7**	3	3	**2**	**1**

B. Performance of GSPS

We first compare results obtained with GSPS on the first seven problem classes with those of the best-known heuristics BL and BLF (with and without pre-ordering of items). When pre-ordering is applied, the items are either sorted by descending height (DH) or by descending width (DW). In order to assess the performance of the heuristics, we use the distance between obtained and optimal container heights, measured in terms of rounded, integer-valued percentages of the optimal height [15]. This *relative distance* is averaged over all problems in each class.

The results in Table I show that the BL methods perform worst on all test problems. As pointed out earlier, this is likely because BL creates many 'dead' spaces during the placement process and has no mechanism for repair. As can also be seen from the table, in all problem classes our GSPS heuristic produced much better results than the BL and the BLF methods.

In Table II we compare some more recent heuristics including *best fit* (BF) [5] and *heuristic recursive* (HR) [21] with our GSPS heuristic. In general, BF performs worst, and GSPS and HR show similarly good performance. All approaches work better in larger (C6 and C7) than in smaller (C1 to C3) problems. This is consistent with the findings reported in [5] that heuristics tend to work more effectively in large-size problems.

As derived in Section III-E, the GSPS item placement procedure has a computational complexity of $O(nm^2)$, i.e. scales, at worst, quadratically with the width W of the container, and linearly with the number n of items. Thus, GSPS promises huge computational efficiency gains especially for large-n problems. Still, grid management incurs some time. To estimate this cost we experimentally measure GSPS execution time and compare it with times obtained for BF and HR. We use the BF results published in [5] as a yardstick for our analysis because BF is known to be the fastest heuristic in the literature. HR times [21] are shown since HR performs particularly well (actually, as well as GSPS, see Table II) but its placement strategy has a complexity of $O(n^3)$.

TABLE III

AVERAGE EXECUTION TIME (IN SEC) ON SEVEN CLASSES OF 21 TEST PROBLEMS

	C1	C2	C3	C4	C5	C6	C7
BF	0	0	0	0	0.003	0.003	0.003
HR	0	0	0	0.112	0.552	1.768	28.86
GSPS	0	0	0	0	0.015	0.031	0.171

In order to make fair comparisons, we converted the times reported in [5], [21] to our time scale. BF ran on a machine with 850MHz CPU, and HR on one with 2.4GHz CPU, while GSPS ran with a CPU of 3GHz. Thus, a fair comparison is achieved when dividing the reported times for BF and HR by 3.53 and 1.25, respectively, and is given in Table III. Although it is a rough measure, it is enough in this context. BF had very fast runs on all 21 test problems shown. GSPS came close in second place. Especially to solve the three problems of C7, GSPS took on average only 0.171 seconds, while HR took almost half a minute. HR with $O(n^3)$ is obviously the least efficient heuristic. Its computational time grows quickly as the problem size increases.

When comparing performance of GSPS in C6 with that in C5 we note that the ratio of execution times almost equals $\frac{n_{C6}W_{C6}^2}{n_{C5}W_{C5}^2}$. The same is true when comparing performance in C7 with that in C6 or C5. Thus, the examples are testing for GSPS as they appear to be close to the worst-case grid management scenario. We expect that grid management related execution times could be reduced through improved implementation of the GSPS heuristic.

C. Performance of meta-heuristic GSPS-M

In this section, we investigate the improvement of solution quality using a hybridization scheme. In all experiments, we set the population size to $P = 100$, the number of generations to $G = 1000$, and the crossover and mutation rates to 0.9 and 0.01, respectively. We generated 10 runs with different random seeds for each problem thereby following the experimental approach of [15], [6].

In Table IV we compare the performance of GSPS and BF with the meta-heuristic hybridization GSPS-M across all 47 benchmark problems. When the hybridization scheme is used, GSPS improves significantly. GSPS-M found better solutions than GSPS in 39 of the 47 problems; six of them are even optimal. From the table we can also see, once more, that GSPS overall performs better than BF. In particular, GSPS was superior to BF on 24, worse on 11 and equivalent on 12 problems.

GSPS found the optimal solution for two problems (P1 and P13 of C8), BF for none.

Figure 7 shows the layouts for problem P1 of C7 as obtained by the GSPS heuristic and the GSPS-M hybrid

TABLE IV

CONTAINER HEIGHT SOLUTIONS BY PROBLEM CLASS, ID AND SIZE (n). "OPT" DENOTES THE OPTIMAL SOLUTION, "BF", "GSPS" AND "GSPS-M" THOSE GENERATED BY THE RESPECTIVE HEURISTIC. BOLDFACE TEXT LABELS THE BEST RESULT AMONG BF, GSPS AND GSPS-M. UNDERLINED TEXT MARKS A CASE WHEN A HEURISTIC FOUND OPT.

Class	ProbID	n	Opt	BF	GSPS	GSPS-M
C1	P1	16	20	21	21	**20**
C1	P2	17	20	22	22	**21**
C1	P3	16	20	24	22	**20**
C2	P1	25	15	16	16	16
C2	P2	25	15	16	16	16
C2	P3	25	15	16	16	**15**
C3	P1	28	30	32	32	**31**
C3	P2	29	30	34	32	**31**
C3	P3	28	30	33	32	**31**
C4	P1	49	60	63	**62**	**62**
C4	P2	49	60	62	**61**	**61**
C4	P3	49	60	62	62	**61**
C5	P1	73	90	93	92	**91**
C5	P2	73	90	92	93	**91**
C5	P3	73	90	93	92	**91**
C6	P1	97	120	123	122	**121**
C6	P2	97	120	122	122	**121**
C6	P3	97	120	124	123	**121**
C7	P1	196	240	247	244	**242**
C7	P2	197	240	244	242	**241**
C7	P3	196	240	245	242	**241**
C8	P1	10	40	45	**40**	**40**
C8	P2	20	50	53	53	**50**
C8	P3	30	50	52	53	**51**
C8	P4	40	80	83	82	**81**
C8	P5	50	100	105	107	**103**
C8	P6	60	100	103	104	**101**
C8	P7	70	100	107	**101**	**101**
C8	P8	80	80	84	82	**81**
C8	P9	100	150	152	156	**151**
C8	P10	200	150	152	152	**151**
C8	P11	300	150	152	152	**151**
C8	P12	500	300	306	304	**303**
C8	P13	3125	960	964	**960**	**960**
C9	P1	50	375	**400**	**400**	**400**
C10	P1	25	100	107.4	107.3	**103.5**
C10	P2	50	100	108.5	109.2	**105.9**
C10	P3	100	100	**107.0**	113.0	107.6
C10	P4	200	100	**105.3**	110.6	105.8
C10	P5	500	100	103.5	103.9	**101.6**
C10	P6	1000	100	103.7	103.4	**101.4**
C11	P1	25	100	110.1	109.8	**102.4**
C11	P2	50	100	113.8	107.1	**102.5**
C11	P3	100	100	107.3	104.3	**102.3**
C11	P4	200	100	104.1	104.4	**102.8**
C11	P5	500	100	103.7	103.2	**101.8**
C11	P6	1000	100	102.8	103.4	**102.0**

scheme. Obviously, the GSPS-M solution has much less gaps than the GSPS solution indicating that permuting item order and changing their orientation prior to placement can result in marked improvements of solution quality.

We now compare GSPS-M to several other meta-heuristics. We choose BLF with genetic algorithm (BLF-M1), and with simulated annealing (BLF-M2). Best results for these schemes are extracted from [5], while results for BF with simulated annealing (BF-M) are quoted from [6], and HR with genetic algorithm (HR-M) from [22]. For HR-M only class-averaged results for the problem classes C1 to C7 are provided in the original paper. However, the general performance of HR-M on these problems is easy to infer.

As shown in Table V, the BLF methods (BLF-M1 and BLF-M2) perform worst. They are competitive only on classes C1, C2, C9, and problem P1 of C8. Meanwhile, it is interesting to see that BF-M and HR-M are competitive on C1 to C7, C9, C10 (BF-M only), while they have worse performance on C8 and C11 (BF-M only) when compared with GSPS-M. In the first seven classes, BF-M obtains the best results in C1, and C4, HR-M in C3 to C6, and GSPS-M in C2 and C5 to C7.

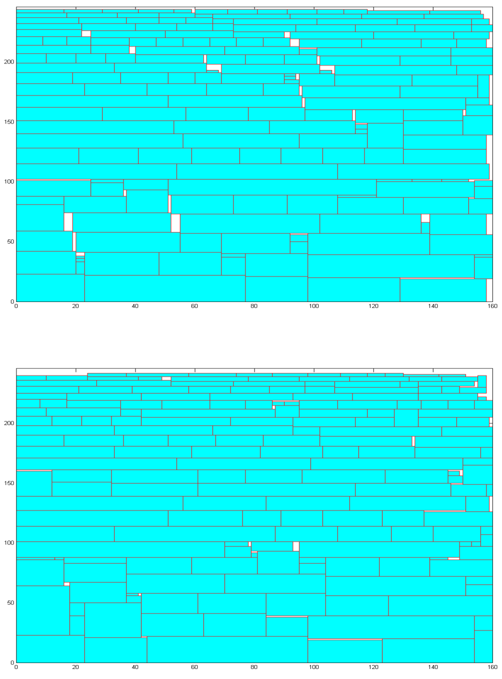

Fig. 7. Layout obtained in P1 of C7 by GSPS (top) and GSPS-M (bottom).

For C8, BF-M and HR-M are each best in only four of the 13 problems. Note that all HR-M solutions for class C8 are suboptimal, while BF-M found optimal solutions for the two smallest of the C8 problems. GSPS-M performs best on all 13 problems of C8, and finds the largest number of optimal solutions, including one for the largest of the C8 problems, P13. In C10, GSPS-M demonstrates good performance compared with BF-M. In particular, it is more effective in the large-scale problems, P5 and P6. In C11, it is better than BF-M across all problems.

V. CONCLUSIONS

In this paper we introduced the GSPS approach for OP1RC, a class of 2D packing problems. The novelty of our approach is the use of discretization of the packing space, i.e. it makes use of a placement heuristic in which the container's space is managed by a grid of cells. The status of these cells indicates whether a space is available or not. The item is first placed into an available cell that is found by a bottom-left oriented search. If necessary, the item is allowed to slide downwards and to the left. We combined

TABLE V

SAME AS TABLE IV BUT SHOWING SOLUTIONS FOUND BY HYBRIDIZING BLF [5], BF [6], HR [22] AND GSPS WITH META-HEURISTICS ('-' INDICATES THAT DATA ARE NOT AVAILABLE IN THE ORIGINAL PAPERS).

Class	ProbID	n	Opt	BLF-M1	BLF-M2	BF-M	HR-M	GSPS-M
C1	P1	16	20	**20**	**20**	**20**	-	**20**
C1	P2	17	20	21	21	**20**	-	21
C1	P3	16	20	**20**	**20**	**20**	-	**20**
	Avg			20.33	20.33	**20**	20.67	20.33
C2	P1	25	15	16	**16**	**16**	-	**16**
C2	P2	25	15	16	**16**	**16**	-	**16**
C2	P3	25	15	16	16	16	-	**15**
	Avg			16	16	16	**15.67**	**15.67**
C3	P1	28	30	32	32	**31**	-	**31**
C3	P2	29	30	32	32	**31**	-	**31**
C3	P3	28	30	32	32	**31**	-	**31**
	Avg			32	32	31	**30.67**	31
C4	P1	49	60	64	64	**61**	-	62
C4	P2	49	60	63	64	**61**	-	**61**
C4	P3	49	60	62	63	**61**	-	**61**
	Avg			63	63.67	**61**	**61**	61.33
C5	P1	73	90	95	94	**91**	-	**91**
C5	P2	73	90	95	95	**91**	-	**91**
C5	P3	73	90	95	95	92	-	**91**
	Avg			95	94.67	91.67	**91**	**91**
C6	P1	97	120	127	127	122	-	**121**
C6	P2	97	120	126	126	**121**	-	**121**
C6	P3	97	120	126	126	122	-	**121**
	Avg			126.33	126.33	121.67	**121**	**121**
C7	P1	196	240	255	255	244	-	**242**
C7	P2	197	240	251	253	244	-	**241**
C7	P3	196	240	254	255	245	-	**241**
	Avg			253.33	254.33	244.33	242	**241.33**
C8	P1	10	40	**40**	**40**	**40**	45	**40**
C8	P2	20	50	51	52	**50**	54	**50**
C8	P3	30	50	52	52	**51**	**51**	**51**
C8	P4	40	80	83	83	82	83	**81**
C8	P5	50	100	106	106	**103**	**103**	**103**
C8	P6	60	100	103	103	102	102	**101**
C8	P7	70	100	106	106	104	104	**101**
C8	P8	80	80	85	85	82	82	**81**
C8	P9	100	150	155	155	152	152	**151**
C8	P10	200	150	154	154	152	**151**	**151**
C8	P11	300	150	155	155	153	**151**	**151**
C8	P12	500	300	313	312	306	304	**303**
C8	P13	3125	960	-	-	964	-	**960**
C9	P1	50	375	**400**	**400**	**400**	-	**400**
C10	P1	25	100	108.2	109.3	104.0	-	**103.6**
C10	P2	50	100	112.0	112.6	**104.4**	-	105.9
C10	P3	100	100	113.0	113.3	**105.0**	-	107.6
C10	P4	200	100	113.2	113.4	**104.7**	-	105.8
C10	P5	500	100	111.9	111.9	103.5	-	**101.6**
C10	P6	1000	100	-	-	103.8	-	**101.4**
C11	P1	25	100	106.7	109.6	103.9	-	**102.4**
C11	P2	50	100	107.0	108.7	103.4	-	**102.5**
C11	P3	100	100	109.0	109.0	103.0	-	**102.3**
C11	P4	200	100	108.8	110.3	103.4	-	**102.8**
C11	P5	500	100	111.5	113.0	103.5	-	**101.8**
C11	P6	1000	100	-	-	102.9	-	**102.0**

the placement heuristic with an item sequencing heuristic inspired by those defined in [15] and [5]. We showed that the GSPS approach has lower computational complexity than several other approaches such as BLF or HR.

In a case study we demonstrated that the GSPS performs very competitively on 47 benchmark problems. We also found that the obtained solutions can be improved further by using hybridization with a meta-heuristic.

As for future work, our main motivation in this paper was driven by large-scale military vehicle loading problems. Our next step is to allow the addition of some domain specific mixing and layout constraints before deploying the heuristics in a loading software.

ACKNOWLEDGMENTS

We acknowledge the financial support from the Australian Research Council (DP0667123), DSTO and Vietnam's National Foundation for Science and Technology (Development Grant 102.01-2010.12). A thankyou goes to Prof. Edmund Burke and Dr. Glenn Whitwell for making C10 and C11 available to us.

REFERENCES

[1] A.R. Ahmad, O.A. Basir, M.H. Imam, and K. Hassanein. An efficient, effective, and robust decoding heuristic for metaheuristics-based layout optimization. *Int. J. of Production Research*, 44(8):1545–1567, 2006.

[2] AR Babu and NR Babu. Effective nesting of rectangular parts in multiple rectangular sheets using genetic and heuristic algorithms. *Int. J. of Production Research*, 37(7):1625–1643, 1999.

[3] B.S. Baker, EG Coffman Jr, and R.L. Rivest. Orthogonal Packings in Two Dimensions. *SIAM Journal on Computing*, 9:846, 1980.

[4] JE Beasley. Algorithms for unconstrained two-dimensional guillotine cutting. *J. of the Operational Research Society*, 36(4):297–306, 1985.

[5] E. Burke, G. Kendall, and G. Whitwell. A New Placement Heuristic for the Orthogonal Stock-Cutting Problem. *Operations Research*, 52(4):655–671, 2004.

[6] E. Burke, G. Kendall, and G. Whitwell. A simulated annealing enhancement of the best-fit heuristic for the orthogonal stock cutting problem. *JOURNAL ON COMPUTING*, 21(3):505–516, 2009.

[7] B. Chazelle. The Bottom-Left Bin-Packing Heuristic: An Efficient Implementation. *IEEE Tran. on Computers*, 32(8):697–707, 1983.

[8] VD Cung, M. Hifi, and B. Cun. Constrained two-dimensional cutting stock problems a best-first branch-and-bound algorithm. *Int. Tran. in Operational Research*, 7(3):185–210, 2000.

[9] CH Dagli and A. Hajakbari. Simulated annealing approach for solving stock cutting problem. *Systems, Man and Cybernetics, 1990. Conference Proceedings., IEEE Int. Conference on*, pages 221–223, 1990.

[10] K. A. Dowsland and Dowsland W. B. Packing problems. *European J. of Operational Research*, 56(1):2–14, 1992.

[11] H. Dyckhoff. A typology of cutting and packing problems. *European J. of Operational Research*, 44(2):145–159, 1990.

[12] L. Faina. An application of simulated annealing to the cutting stock problem. *European J. of Operational Research*, 114(3):542–556, 1999.

[13] Pablo Garrido and María-Cristina Riff. An evolutionary hyperheuristic to solve strip-packing problems. In *IDEAL*, volume 4881 of *Lecture Notes in Computer Science*, pages 406–415. Springer, 2007.

[14] PC Gilmore and RE Gomory. A linear programming approach to the cutting stock problemPart II. *Operations Research*, 11(6):863–888, 1963.

[15] E. Hopper and B.C.H. Turton. An empirical investigation of meta-heuristic and heuristic algorithms for a 2D packing problem. *European J. of Operational Research*, 128(1):34–57, 2001.

[16] S. Jakobs. On genetic algorithms for the packing of polygons. *European J. of Operational Research*, 88(1):165–181, 1996.

[17] KK Lai and J.W.M. Chan. Developing a simulated annealing algorithm for the cutting stock problem. *Computers & Industrial Engineering*, 32(1):115–127, 1997.

[18] D. Liu and H. Teng. An improved BL-algorithm for genetic algorithm of the orthogonal packing of rectangles. *European J. of Operational Research*, 112(2):413–420, 1999.

[19] C.L. Valenzuela and P.Y. Wang. Heuristics for large strip packing problems with guillotine patterns: An empirical study. In *Proceedings of the 4th Metaheuristics Internaternational Conference*, pages 417–421, 2001.

[20] G. Wäscher, H. Haußner, and H. Schumann. An improved typology of cutting and packing problems. *European J. of Operational Research*, 183(3):1109–1130, 2007.

[21] D. Zhang, Y. Kang, and A. Deng. A new heuristic recursive algorithm for the strip rectangular packing problem. *Computers and Operations Research*, 33(8):2209–2217, 2006.

[22] D.F. Zhang, S.D. Chen, and Y.J. Liu. An Improved Heuristic Recursive Strategy Based on Genetic Algorithm for the Strip Rectangular Packing Problem. *Acta Automatica Sinica*, 33(9):911–916, 2007.